Technical University of Crete

School of Electrical and Computer Engineering

Interactive Dynamic Time Lapse Video for Immersive and Personalized Tourism Applications

Author: Theodoros Geronymakis

A thesis submitted in partial fulfillment of the requirements for the diploma of Electrical & Computer Engineer.

Committee:

Emeritus Professor Stavros Christodoulakis (Supervisor)

Associate Professor Katerina Mania

Associate Professor Antonios Deligiannakis

Acknowledgments

First of all, I would like to express my gratitude to my supervisor, Stavros Christodoulakis, for the patient guidance, encouragement and advice he has provided throughout my thesis. I would also like to thank Prof. Katerina Mania and Prof. Antonios Deligiannakis for serving on my thesis committee, as well as the members of the staff of the MUSIC laboratory for their help and support. Furthermore I would like to express my deepest gratitude to my parents, my uncle and especially my brother for their continuous support, endless encouragement, and confidence in me. Finally, I want to thank my friends for all the good times we had together during our studies.

“All's well that ends well.”

— William Shakespeare

Abstract

We present a system that can be used to generate web based interactive dynamic and personalized video based navigations through a city or geographic area. The dynamic video presentations are generated from a database of panoramic images for the area, and/or from StreetView panoramic images. User preferences and interactions can change what and how it is presented during the navigation and they give a sense of personalization but also freedom of navigation, flexibility and immersion to the final user.

Contents

List of Figures

List of Algorithms

Chapter 1

1 Introduction

1.1 Background

Culture and tourism applications allowing navigation through a city or area and viewing their important sights are significant classes of web applications that include mobile extensions, and have great social and economic significance. They typically allow their users to get to know from a distance a city or geographic area, its sights of interest, and its life, and also allow potential visitors with limited time available to preselect where to go when in the destination. The same applications can help the users using mobile devices while at destination.

Video based presentations have the capability to project a more integrated understanding and a feeling about a city, its architecture, the relationship of sights of interest with the rest of the city, and the life in the city. However, they also often restrict the user to a rigid presentation that reduces or removes the feeling of presence of the user in the city, and in addition the presentation may not match the user interests or time available for viewing.

In this Thesis we propose an interactive dynamic video environment that aims to remove these limitations of the existing applications and creates feeling of personalization, immersion and freedom to the end user of these applications.

We have designed and implemented a system that can be used to create video based and multimedia web and mobile applications for destinations that have the following characteristics:

The applications generated by the system are emphasizing video based navigation and narration in a city or geographic area. The presentations allow in parallel to visualize the navigation of the user on top of a Google map of the city so that the user has also a spatial understanding of where he is during the video presentation. They can also take into account the user interests in order to present personalized video navigations through the city or area. In addition, the applications created allow user interaction during the navigation to change the flow and the parameters of presentation, thus giving the sense of flexibility and immersion to the final user. To provide these functionalities to the final users the system employs interactive dynamic video creation mechanisms that control the presentation. Interactions can also allow the user to follow only parts of routes in the city, navigate only in order to view specific sights, or dive more deeply into the presentation of the sights of his interest. An important functionality of the system is that it allows to use both, the StreetView API and existing StreetView panoramic images, as well as to use a data base of user captured panoramic images and compatible image management functionality in order to create the video presentations. Presentations based on both types of panoramic images are often needed even in areas that are well covered by StreetView panoramic images for applications of culture and tourism.

1.2 Thesis Layout

- Chapter 2 presents the state of the art, the related publications and applications, and the technologies used in the development of our system.
- Chapter 3 describes the general functional requirements of our system and the research contributions of this work. It also presents the information modeling of the system and the reasons we decided to construct each information object and functionality.
- Chapter 4 describes the database we used for our system. It presents the Entity Relationship and UML Class diagram and gives a short description for the entities and functions of the Database.
- Chapter 5 analyzes the functionalities, described in the Requirement Analysis (Chapter 2), in depth. In this chapter the algorithms and methods used to develop each functionality are presented.
- Chapter 6 refers to the User Interface designed for our system and its usability.
- Finally, Chapter 7 presents the evaluation of the application, the conclusions and possible future work.

Chapter 2

2 Related Work & State of The Art

2.1 Introduction

In the era of rapidly growing IT technology, the idea of real world virtualization where every person is able to visit all corners of the globe without leaving the computer screen, becomes more and more important. Virtual Tours are designed to offer this kind of experience to users. Through the sequence of Panoramic Pictures the user feels like he is actually walking down the path of the Route. Using Markers and Narrations alongside the Route provides a more interactive and immersive experience to the users. There are many applications, websites and tools that try to provide the services and functionalities for designing and more often navigate a Virtual Tour. In this chapter we are going to analyze the most important of those applications, in our opinion, describing their approach in the process of creating virtual Tours and the differences with our approach.

2.2 Geographic Context Understanding with Mobile Applications in Culture and Tourism

Coupling GPS and compass information with cameras to capture multimedia information useful to culture and tourism is the subject of several research prototypes and systems in the last several years. Some of those systems also create semantic maps with information on important sights for culture and tourism within a city or in the countryside[1]. During the navigation with a mobile device these systems can point to the direction of sights of interest to the user [2]. During the navigation with a mobile device these systems can point to the direction of sights of interest to the user [3]. During the navigation with a mobile device these systems can point to the direction of sights of interest to the user. The access to additional information for individual objects of interest is based on user interaction with the mobile device. In general those systems target the

[1]Christodoulakis S., Foukarakis M., Ragia L., Uchiyama L., Imai H. : Semantic maps and Mobile Context Capturing for Picture Context Visualization and Management of Picture dDBMSs, Proceedings of ACM Mobile Ubiquitous Multimedia (MUM), Finland 2009.

[2] Christodoulakis S., Foukarakis M., Ragia L., Uchiyama L., Imai H. : Mobile Picture capturing for Semantic Picture Database Context Access, Browsing and Interaction, IEEE Multimedia 20010.

[3] S. Christodoulakis, P. Kontogiannis, P. Petridis, N. Moumoutzis, M. Anastasiadis, T. Margazas: MINOTAURUS: A Distributed Multimedia Tourism Information System

user who wonders within a city with a mobile device in his hands. They don’t present a simulated walk through routes of the city supported by dynamic personalized video to be used before an actual visit by the user, or a casual web user that wants to know about the city and its life. Interactive dynamic and personalized video gives the sense of emergence and provides much more feeling about a destination and its life than just interactive picture access.

2.3 Paneek: Virtual Tour Software

A system that aims to provide virtual Tours to a city is Paneek[4]. Paneek is a 360^{o} Virtual Software that offers both free and paid services for creating and projecting Virtual Tours. It is an interactive media platform that help users to create more engaging tours by adding rich media like links, photos, videos, product catalogs etc.

Users have to upload their Panoramic Pictures to the Paneek website. When a user uploads an image, the image needs to be processed in the server first. The process might take a few minutes and the user receives an email with the notification when the Panoramic Picture is ready for editing.

2.3.1 Navigation

After uploading the Panoramic Pictures the user can proceed to the Tour Creation Tool. There he can select which Panoramic Pictures will constitute the Tour, by choosing from the Panoramic Pictures he has uploaded. The navigation across the Panoramic Pictures is achieved in two modes:

Arrow mode

Each panorama in the tour has 8 available directions. These directions are in degrees, each arrow represent a direction. Directions are represented in 0^{o}, 45^{o}, 90^{o}, 135^{o}, 180^{o}, 225^{o}, 270^{o}, 315^{o} degrees in total sum 360^{o}, as we can see in the figure 2.1.

When creating the Tour, at the right side of each arrow there is a list of all the panoramas available in the tour. The user has to select the panorama that he wants to be loaded when that arrow is clicked. The user can also set the Point of View on which the Panorama will be loaded.

[4] Paneek Website: https://www.paneek.net/

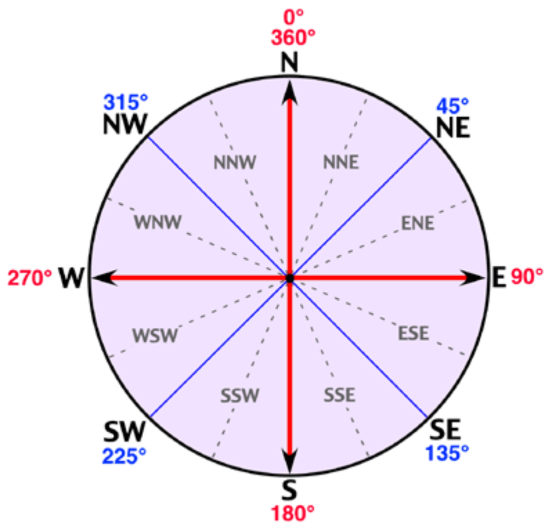
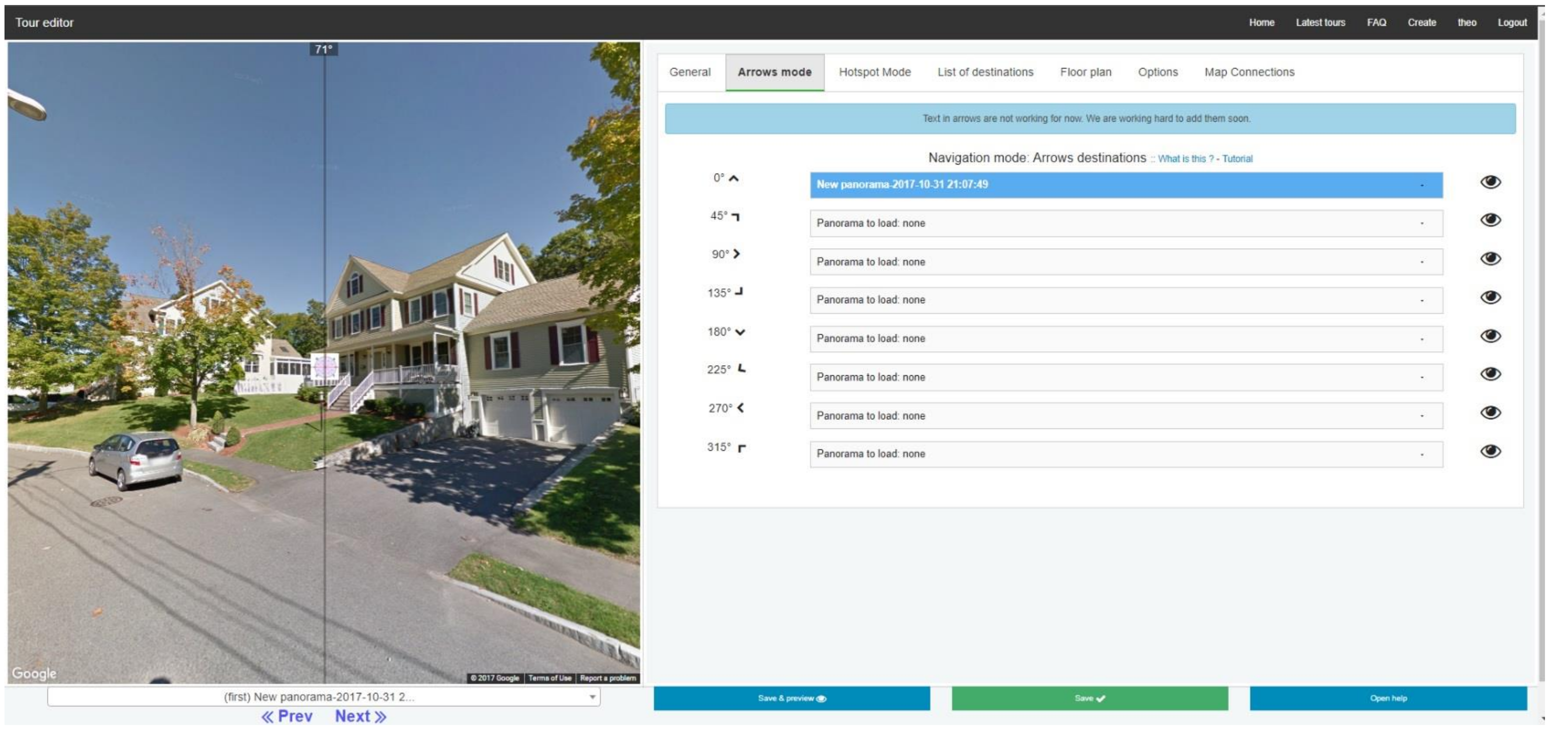


Figure 2.1 All the possible directions for a Panorama expressed in Degrees. The user links those directions with other Panoramic Pictures, constructing that way the Navigation between the Panoramic Pictures that consist the Tour.



Text in arrows are not working for now. We are working hard to add them soon.

Navigation mode: Arrows destinations :: What is this ? - Tutorial

0°	New panorama-2017-10-31 21:07:49
45°	Panorama to load: none
90°	Panorama to load: none
135°	Panorama to load: none
180°	Panorama to load: none
225°	Panorama to load: none
270°	Panorama to load: none
315°	Panorama to load: none

Figure 2.2 Paneek Tour Editor Interface. This tab handles the Arrow Navigation Mode, giving the Users the ability to set the destination Panoramic Picture from each angle.

Hotspot mode

In this mode the user can use hotspots to load locations. This method is particularly good for Virtual Tours that do not have a simple orientation. The hotspots can be placed anywhere in the Panoramic Picture and they lead to the next Panorama[5]. Customization of the Hotspots is available only for paid accounts.

2.3.2 Hotspots

The handling of visibility identifications on top of a Panoramic Picture, in the Paneek software, is done by the form of Hotspots. The user can place Hotspots on top of Panoramic Pictures within the Panorama Interface, to represent Points of Interest or other details. The basic forms of Hotspot are Text, Image, Video, and Sound. The user must provide the source files via URL. The user can point at any point of a Panoramic Picture using his mouse. The green target symbol shows the position of the mouse pointer at any time. Clicking on the map triggers an event that handles the storing of the X-Y coordinates of the clicked position and presents the Hotspot interface to the user. In that interface the user is asked to fill information such as the Name of the Hotspot, a Text Description and optional image, video files. The user can set a Custom Icon, replacing the Default Hotspot icon. He has to pass the image file as a URL source.

2.3.3 Differences with our application

Navigation of the Virtual Tour:

The main difference between the Paneek software tool and our application is the way each approach handles the Navigation of a user across a Virtual Tour. As we mentioned in subchapter 2.3.1 the Navigation across the Virtual Tour, within the Paneek application, is managed manually from the user, either by clicking on Arrows or on Hotspots that load the corresponding image. This form of Navigation is not guided, meaning that the user may not follow the sequence of the Panoramic Pictures the way the creator of the Virtual Tour intended. On the other hand our application offers Guided Tours, projecting a sequence of Panoramic Pictures as the user moves across the path of the Route in an automated way. The movement of the user is handled by algorithms that animate the movement of the user across the path of the Route on the Map Object and algorithms that calculate the appropriate heading and pitch angle of the Panoramic Picture loading at each point.

Objects of Interest:

Another important difference regards the identification of Objects of Interest that are associate with the Route. The identification of Objects of Interest within the Paneek application is done

[5] YouTube video sample of the Hotspot Navigation mode: Website: https://www.paneek.net/

only in the form of Hotspots (subchapter 2.3.2) on top of Panoramic Pictures. Those Objects of Interest can represent a specific detail on top of a Panoramic Picture. Unlike our application the Paneek does not offer the option of identifying Objects of Interest on top of a Map Object. The user can only set the position of each Panoramic Picture on top of the Map. Our application offers the option of placing a single Marker, a line or a Polygon on the Google Map to mark an Object of Interest. This gives the users the ability to add Footprints of buildings, Areas, Streets and roads as Objects of Interest. Besides identifying the Objects of Interest on the Map Object, the users also have the ability to mark specific details of each Object on the Panoramic Pictures associated with it. This is done by taking advantage of the Photo Sphere JavaScript Library, which allows us to place Custom Markers on Panoramic Pictures in the form of Image Markers (much like the Hotspots of the Paneek) but also with Polygon Markers. Additionality the user cannot set Types for the Objects of Interest he adds on his Virtual Tour. This option is available within our application, the user can set Objects with the type of Museums, Churches, etc. This way when he navigates across a Tour he can choose to get notified about Objects of Interest of a specific Type.

Camera Orientation and Tuning facing Objects of Interest

As we have mentioned, the navigation within the Paneek application is done manually by the users by clicking on arrows or hotspots that load the next Panoramic Picture. This approach does not allows the handling of the Orientation of the Camera to point towards the associated Objects of Interest, since the user can navigate to the Panoramic Picture in a different way than the one the creator of the Route intended. The camera does not change direction to present the user the Objects of Interest appointed on that Panoramic Picture. The user has to explore the Panoramic Picture on his own in order to view the Objects. On the contrary, our application contains the appropriate algorithms that calculate the corresponding Heading and Pitch angles and change the orientation of the camera when approaching an Object of Interest, depending on the way it's represented on the Map (Marker, Line, and Polygon). This way the attention of the user turns to the Object of Interest.

Narration – Story telling

Another difference between the two applications regards the way the story telling of the Virtual Tour is presented. In the Paneek application we can link an Audio file, Narration, to a Panoramic Picture. This Narration will play on the background when the Panoramic Picture is loaded. The users can also add Audio files on the Hotspots Objects, which are played upon clicking on them. Since, the orientation of the camera does not change in order to face the Objects of Interest, the Narration of the Panoramic Picture is not sufficient enough to notify the user about the appropriate Object.

In the contrary the story telling in our application is done through the Visibility Section Objects. Those objects define the sections on the Route from where the associated Object of Interest is visible. Upon entering a Visibility Section Object, the algorithms of our application handle the orientation of the camera towards the Object of Interest. In the same time the Narration regarding that Object of Interest starts playing notifying the user and providing him information, its exact position, historical background, etc. Each Object of Interest can also have an Audio File which can be accessed through the Object of Interest Pop up Interface and can contain more detailed information.

Mobile Navigation

Paneek application does not support the navigation across a Tour using a mobile device. It is built for desktop versions only. Our application on the other hand supports the navigation across a Route with the mobile by requesting access to the GPS position of the user, as he walks down the path of the Route. At each movement we check if the user approaches an Object of Interest that is within the types that he chose to get notified about. If the GPS position of the user falls within the points of the polyline of a Visibility Section, the user gets notified through the Narration regarding the Object of Interest. Besides listening to the Narration, the user can also see a Virtual View of the Object of Interest as he moves across the path.

2.4 Audio Tours with Mobile Device GPS Navigation

A large percentage of guide tour applications provide tours with storytelling in form of audio narrations. The users navigate along a specific route or along a city and get notified about Objects of Interest when approaching their location. We list some of those mobile applications below:

2.4.1 PocketGuide Audio Travel Guide:

PocketGuide[6] is one of the world's leading audio city guide application. PocketGuide reveals the best stories, insider hangouts and must-see sights in more than 150 major cities and tourist destinations.

The tours of the application are made by the professional local guide tours, who are aware about the most interesting sights of the corresponding city and can provide detailed information about them. These tours are tested after submission by the staff. The application guides the users through the city by voice, using the mobile device's GPS location. The users can listen and enjoy a city's attractions while the virtual tour guide explains it all and shares personal insights. The Objects of Interest along a Route are represented by Google Maps Markers on top of a Google Map. Each Marker has a specific radius which identifies when the user approaches the Object of Interest location and starts playing the associated narration.

[6] PocketGuide: http://pocketguideapp.com/

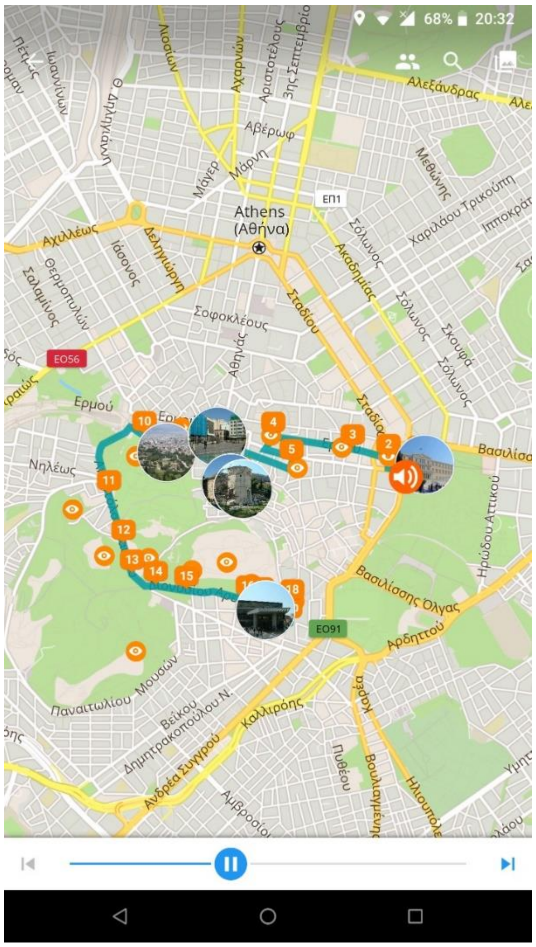
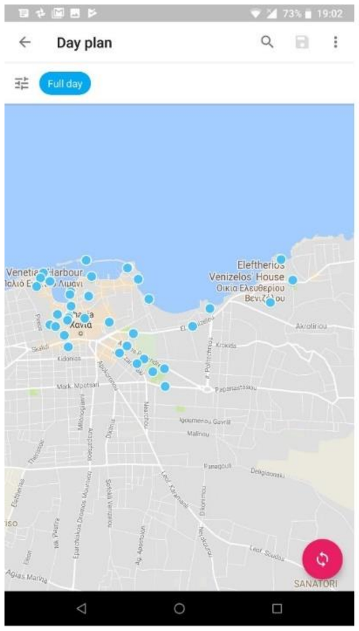


Figure 2.3 Interface of an Audio Tour. The Objects of Interest are placed on top of a Google Map in form of Google Markers.

2.4.2 Google Trips:

Google trips [7] is an android application developed by Google offering users the ability to plan their trip to a city and get information about its Objects of Interest. Within the application the users can view all of the city's well known tourist attractions displayed on top of a Google Map in form of Google Markers. They can interact with those markers by click events in order to view more information or to plan a route by picking the attractions he wants to visit. They can navigate those routes, walking along the city while holding their mobile device so they get notified about the selected attractions when approaching their locations.

[7] Google Trips: https://get.google.com/trips/



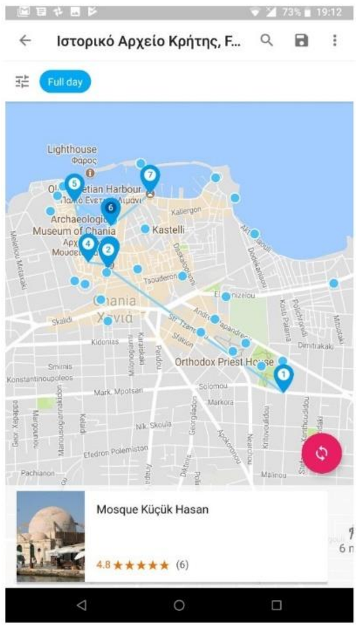


Figure 2.4 Nearby Objects of Interest from user's location. Automated Planning of a Route among Objects of Interest.

2.4.3 Audio augmented reality: a prototype automated tour guide

Augmented reality (or computer augmented environments as it is sometimes called) uses computers to enhance the richness of the real world. It differs from virtual reality in that it doesn't attempt to replace the real world. A system [8] that creates an automated tour guide and superimposes audio on the world based on where a user is located. It proposes this technique for use as an automated tour guide in museums and expect it will enhance the social aspects of museum visits, compared to taped tour guides. They don't present a simulated walk through routes of the city supported by dynamic personalized video to be used before an actual visit by the user, or a casual web user that wants to know about the city and its life. It offers narrations about sights of Interest within the user's given location. Upon approaching a registered location the user gets notified in form of audio notifications.

2.4.4 Detour - Immersive Audio Tours:

Detour [9] is an immersive city guide application [10] constructed around GPS-triggered audio and smart, witty narratives, available both for android and iOS users. Founded by Groupon co-

[8] B. Bederson, Audio Augmented Reality: A Prototype Automated Tour Guide, Proceeding CHI '95 Conference Companion on Human Factors in Computing Systems
[9] Detour Website: https://www.detour.com/

[10] Forbes Article regarding Detour: https://www.forbes.com/sites/angelinavillaclarke/2017/06/19/city-secrets-detour-the-coolest-walking-tour-in-the-world-arrives-in-london/#49108e762766

founder and former CEO Andrew Mason, continues to work on its bigger ambition as a business: a platform for anyone to create an immersive guide for a particular place. The application uses the mobile device GPS to keep the user's location in sync with the narrator—and the narrator delivers all of the navigation, so the user can just leave his phone in his pocket and have his eyes on the world instead of his screen. Detour is working with narrators [11] or 'guides' that users wouldn't normally have access to, in order to give the users a unique experience. For example, they worked with the film-maker Ken Burns' guide to Brooklyn Bridge and actor Joel Grey's take on Broadway.

2.4.5 Field Trip

Field Trip [12] is a guide developed to make the users aware of the interesting and unique things in the world around them. Field Trip runs in the background on the user's mobile device. When the user gets close to an interesting attraction (Object of Interest), a card with details about the location is displayed in form of a pop up notification. If the user has a headset or Bluetooth connected with his mobile device, it is able to read (play) those information in a form of a narration.

Field Trip lets the users to discover thousands of interesting places/experiences that fall under the following categories: Architecture, Historic Places & Events, Lifestyle, Offers & Deals, Food Drinks & Fun, Movie Locations, Outdoor Art and Obscure Places of Interest.

[11] Detour, PlayStore link: https://play.google.com/store/apps/details?id=com.detour.detour&hl=en

[12] Field Trip, PlayStore link: https://play.google.com/store/apps/details?id=com.nianticproject.scout&hl=en

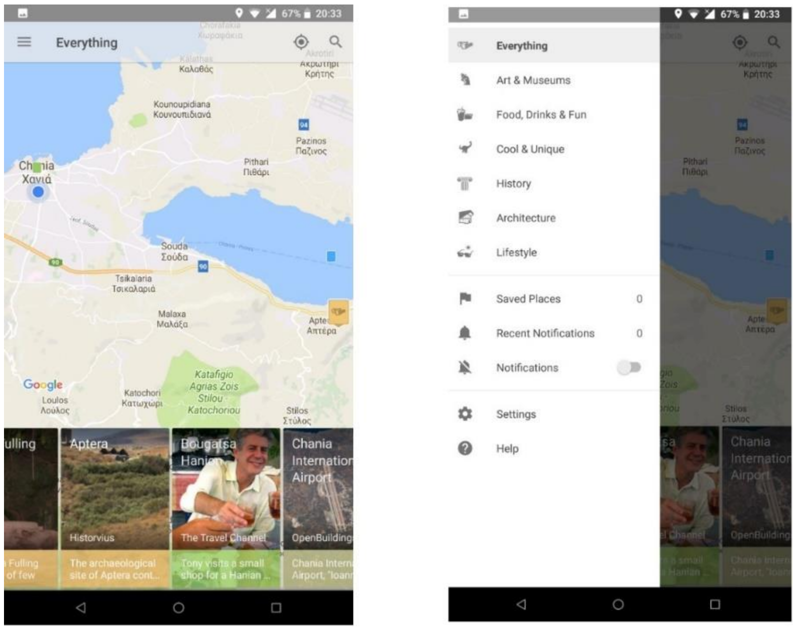


Figure 2.5 The user has the ability to view Objects of Interest that fall under specific categories. The user has the ability to view Objects of Interest that fall under specific categories.

2.4.6 Izi.Travel

We are going to give a more in-depth analysis to the one well known audio tour application. The rest of the applications listed above follow a similar approach regarding the navigation along a route, the triggering of the narrations and the identification of the Objects of Interest of a city or a geographic area.

Izi.Travel [13] is a professional tool to create mobile multimedia guides. There's no special hardware, programming or technical knowledge required. It offers users the ability of creating and publishing indoor and outdoor tours.

2.4.6.1 Tourist Attractions (Objects of Interest):

When a user selects to create an audio tour he is introduced to the Tour Creator's interface. On this interface he has access to a Google Map object on top of which he can define the Tourist Attractions (Objects of Interest) and design the path of the Route of his liking. Those options are available in form of buttons on the top left corner of the Google Map. Each option sets a click listener event on the map. When a user selects to add a Tourist Attraction, a click event is set on the Google Map Object which creates a Google Maps Marker Object on the LatLng position that the user clicked on the Map. Accordingly, when a user selects to set a tour line, the click event

[13] Izi.Travel Website: https://izi.travel/en

listener on the Map keeps track of the user clicked positions and connects them through a Google Maps polyline thus forming the path of a Route.

Each Tourist Attraction Object has a name, a GPS LatLng position, a text Description, an audio file, an optional video file, an optional set of images and an optional external link. It also has an associated circle or polygon object, set to define the trigger zone of the associate Narration (audio file).

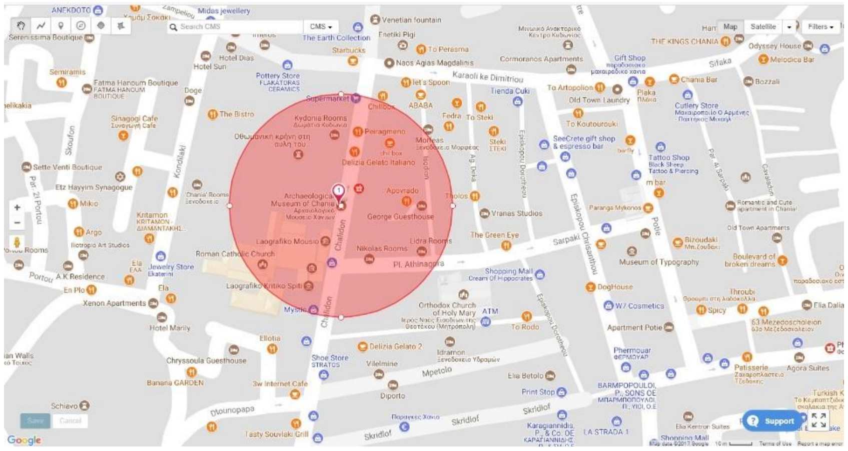
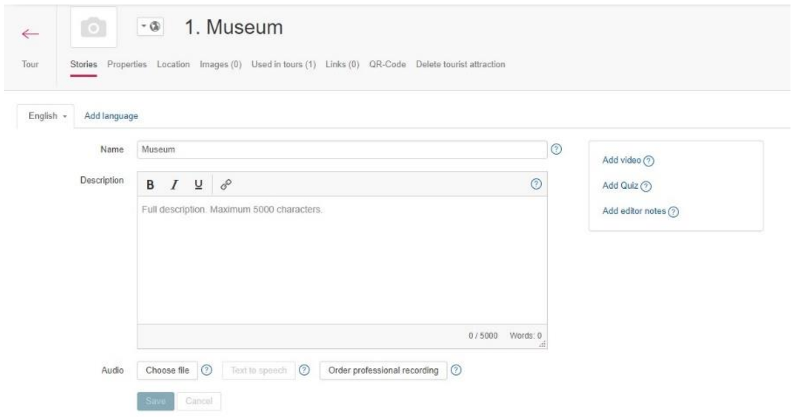


Figure 2.6 Tour Design Interface. The available options on the top Left edge. In this figure we see an example of a Tourist Attraction identification with its set Radius.



1. Museum

Name: Museum

Description: Full description. Maximum 5000 characters.

0 / 5000 Words: 0

Audio

Figure 2.7 Interface of Tourist Attraction. Here the User can define the Name, the Description and the Narration.

2.4.6.2 Navigation

The published guided tours are available both on the izi.TRAVEL website and the dedicated smartphone application. The website offers the user the ability to view the path of the tour and the list of its associated Tourist Attractions. The user does not have the option of navigating the Tour through the website, but he can click on the associated Tourist attractions, read the information and hear their narrations separately.

The navigation of the Tour is done through the smartphone application. The user walks along the designed path of the Tour carrying his mobile device. The application requires access to the GPS of his mobile device in order to keep track of his position along the Tour. When his current position falls into the radius of a Tourist Attraction, the user gets notified and the associated Narration starts playing providing information to the user.

The vast majority of Audio Tour applications uses a similar approach for the definition of Objects of Interest and the navigation of a Tour.

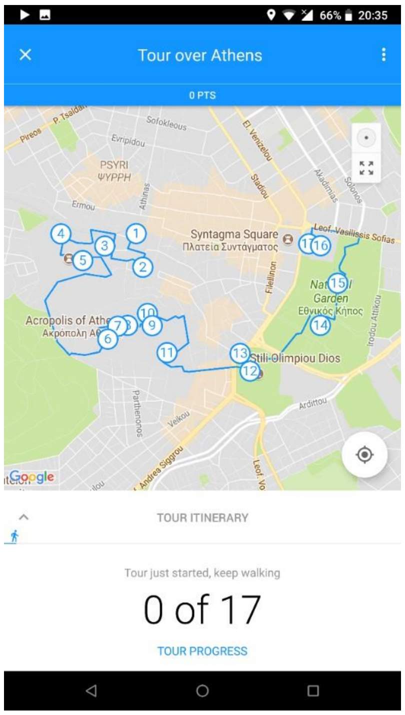


Figure 2.8 Izi.TRAVEL smartphone Interface. Preview of the designed Audio Tour and its Tourist Attractions.

2.4.6.3 Differences with our application:

Navigation of the Virtual Tour:

The main difference between our application and Audio Tour applications regards the form of the presentation of a Route and its associated Objects of Interest to the user. An audio tour or audio guide provides a recorded spoken commentary, normally through a handheld device, to a visitor attraction such as a museum. They are also available for self-guided tours of outdoor locations or as a part of an organized tour. It provides background, context, and information on the things being viewed. Through those recorder narrations the user is notified about the Objects of Interest he is approaching when navigating a published guided Tour or when walking along a city or a geographic area. Those narrations work well in the case of larger Objects of Interest (Museums, Churches, etc.) which the user can easily recognize and turns his attention towards them. There are cases though where the Objects of Interest may be small or specific details on larger object and they may be located high up on a building (like an architectural detail, a Venetian Thereos, etc.). It might be hard for a user to identify the location of those Objects by just hearing a narration about them.

In our system the user navigates through a route by actually walking or driving, for example by walking along a route of a city, carrying his mobile device (phone, etc.) with him. The mobile device notifies him when he approaches objects of interest along his path and if the user wishes he can be helped to view where in the 3D space the objects are located as he is approaching them. When he is walking down a route he can view a video of the route generated by the panoramic pictures retrieved by his current location. He also has the capability to navigate towards the objects, be helped to locate them, hear narrations associated with them, and interact with them in order to find more information.

Objects of Interest:

Another difference regards the identification of Objects of Interest that are associate with the Route. The identification of Objects of Interest within the listed Audio Tour applications is done only in the form of Google Maps Marker on top of a Google Map. Those Objects of Interest can represent a specific location and have an associated radius which is used to define if a user approached them in the navigation mode. Our application offers the option of placing a single Marker, a line or a Polygon on the Google Map to mark an Object of Interest. This gives the users the ability to add Footprints of buildings, Areas, Streets and roads as Objects of Interest. Besides identifying the Objects of Interest on the Map Object, the users also have the ability to mark specific details of each Object on the Panoramic Pictures associated with it. This is done by taking advantage of the Photo Sphere JavaScript Library, which allows us to place Custom Markers on Panoramic Pictures in the form of Image Markers (much like the Hotspots of the Paneek) but also with Polygon Markers. Additionality the user cannot set Types for the Objects of Interest he adds on his Virtual Tour. This option is available within our application, the user can set Objects with the type of Museums, Churches, etc. This way when he navigates across a Tour he can choose to get notified about Objects of Interest of a specific Type.

Narration – Story telling

Finally, another difference between our application and Audio Tours applications regards the way the story telling of the Tour is presented. In most of the Audio Tour applications we can link an Audio file, Narration, to an Object of Interest. This Narration will play on the background when the user's location falls within the set radius value of the Object of Interest.

In the contrary the story telling in our application is done through the Visibility Section Objects. Those objects define the sections on the Route from where the associated Object of Interest is visible. Upon entering a Visibility Section Object, the algorithms of our application handle the orientation of the camera towards the Object of Interest. In the same time the Narration regarding that Object of Interest starts playing notifying the user and providing him information, its exact position, historical background, etc. Each Object of Interest can also have an Audio File which can be accessed through the Object of Interest Pop up Interface and can contain more detailed information.

It must be noted that the izi.Travel and the PocketGuide applications offers the user the ability to add separate navigational story points along a route. Each point has an associated radius much like the Objects of Interest of the Route. When the position of the user is within that radius the associated Narration starts playing.

2.5 Augmented Reality Tourism Applications: Viewranger, Google Goggles, AR City

Another category of Tourist Guide applications are based on Augmented Reality technologies. Augmented reality (AR) [14] is a live direct or indirect view of a physical, real-world environment whose elements are "augmented" by computer-generated or extracted real-world sensory input such as sound, video, graphics, haptics or GPS data. Augmented reality is used in order to enhance the experienced environments or situations and to offer enriched experiences. Augmentation techniques are typically performed in real time and in semantic context with environmental elements, such as overlaying supplemental information like scores over a live video feed of a sporting event. With the help of advanced AR technology the information about the surrounding real world of the user becomes interactive and digitally manipulable. Information about the environment and its objects is overlaid on the real world.

2.5.1 Google Goggles

Google Goggles is an image recognition mobile app developed by Google. It is used for searches

[14] Augment Reality: https://en.wikipedia.org/wiki/Augmented_reality

based on pictures taken by handheld devices. For example, taking a picture of a famous landmark searches for information about it, or taking a picture of a product's barcode would search for information on the product. The response might be in form of a Wikipedia explanation, a price, directions, and so on. It works amazingly well with many worldwide tourist sites and product labels, but also works as a handy foreign language text translator. Users can snap an image of the words they want translated, then choose translate and their preferred language in the search results. Google Goggles can also work in the background of the user’s phone or tablet, and analyze photos as they're taken. The application will automatically notify the user when any search results match his capture.

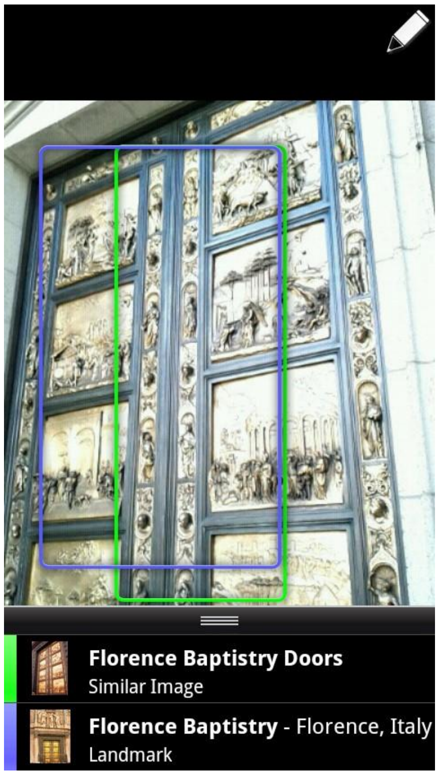


Figure 2.9 Landmark Recognition through the Google Goggles application. The user snaps a photo, the application searches for relevant information and notifies the user with a response.

2.5.2 Viewranger

ViewRanger is a digital guide to the outdoors, with hundreds of thousands of downloadable route guides, free worldwide maps, and powerful GPS navigation features. Viewranger application offers augment reality interaction with the Skyline extension. ViewRanger Skyline lets users explore the outdoors with augmented reality, using the phone or tablet's camera to label more than 9 million landscape features. Users have to simply hold their phones up to the landscape to identify everything from peaks, towns and villages to mountain passes, lakes, and glaciers. Skyline labels key points up to 20 miles away, transforming your traditional 2D map with augmented reality.

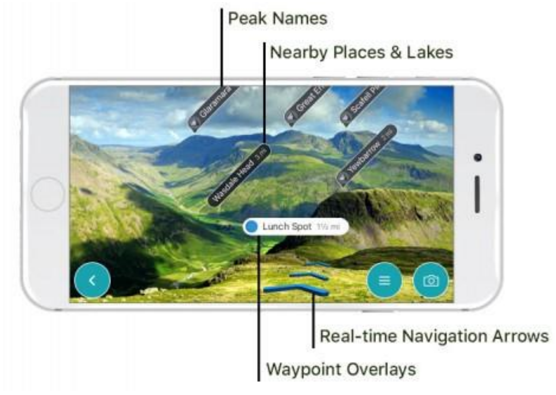


Figure 2.10 Real Time Navigation and Augmented Reality Overlays through the user's mobile device camera.

Skyline also offers real time navigation. It overlays waypoints and navigation arrows on the real landscape. Users can plan their own routes or download and view already published route guides.

2.5.3 Terrain Understanding from Pictures and Augmented Reality

In a similar spirit, one could use a picture captured by his mobile device, point to an interesting object in the picture and ask information about this object. This requires understanding of pictures of the earth taken from any location to wards any direction. A system developed with this objective [15] essentially creates the capability to annotate terrain pictures through image understanding. The GPS location of the user and the direction of the camera in combination with the 3D terrain elevation information is useful for the image understanding, but additional knowledge has to be employed for better accuracy of far located objects that can be very small in the picture. One way to do it is to identify with picture processing software easy to identify geographic features like the mountain skyline, the sea boundaries with the land, etc., also taking into account the time of the year day etc. In comparison to similar applications like Google Goggles that create augmented reality structures on top of buildings and roads, outside cities the capturing of information for all locations that the user may go is infeasible and the objects in general may be located far and be very small in the picture.

2.5.4 Collaborative Augmented Reality for Outdoor Navigation and Information Browsing

Augmented reality (AR) can provide an excellent user interface for visualization in a mobile computing application. The user’s view is augmented with location based information at the correct spatial location, thus providing an intuitive way of presenting such information. A system

[15] Christodoulakis S., Foukarakis M., Tsinaraki |C., Ragia L., Kanellidi, E. : Contextual Geospatial Picture Understanding, Management and Visualization, Proceedings 11th International Conference on Advances in Mobile Computing and Multimedia, MoMM 2013.

[16] that makes use of AR for collaborative navigation and information browsing tasks in an urban environment. A navigation function allows one or more users to roam through a city and guides them to selected destinations. Information browsing presents users with information about objects in their surroundings. Both functions feature support for collaboration. The developed system does not only concentrate on the user interface aspects but also provides a scalable infrastructure to support mobile applications. To this end we developed a 3-tier architecture to manage a common data model for a set of applications. It is inspired by current Internet application frameworks and consists of a central storage layer using a common data model, a transformation layer responsible for filtering and adapting the data to the requirements of a particular applications on request, and finally of the applications itself.

2.5.5 AR City: Augmented Reality Maps and Navigation (Beta)

AR City[17] is an iOS application. It is a navigation – tourist guide application build on top of Apple Maps also offers real time navigation. It overlays waypoints and navigation arrows on the real landscape. Users can plan their own routes or download and view already published route guides.

The application[18] helps users navigate and explore cities worldwide using augmented reality and computer vision at scale. In selected locations the AR experience is enhanced with urban visual positioning which localizes users with higher accuracy than GPS, thanks to computer vision. The instructions are displayed using the mobile device camera in form of 3D directions. While the user holds his mobile device in front of him with the camera pointing at the screen, the application displays navigation arrows on top of the street through the camera. As the user travels he will see the names of the streets and buildings appear in front of him.

The application has three key layers of information:

- **AR basic navigation**: a visualization of walking routes in augmented reality. AR basic navigation is available everywhere supported by Apple Maps. It visualizes the route to get users to their destination in augmented reality with 3D arrows. This feature uses GPS to estimate the absolute position of the user, and Visual Inertial Odometry (VIO) to track their local movement.
- **Enhanced map content**: AR overlays of information related to user’s location - for example streets and points of interest. Provides additional information about the places around the user’s location. Initially this includes street names, building names and local points of interest. All the information is integrated into our visual knowledge graph. It is

[16] G. Reitmayr, D. Schmalstieg: Collaborative Augmented Reality for Outdoor Navigation and Information Browsing. in: "Proceedings of the Second Symposium on Location Based Services and TeleCartography", TU Wien, 2004

[17] Documentation: https://blippar.com/en/resources/blog/2017/11/06/welcome-ar-city-future-maps-and-navigation/

[18] itunes: https://itunes.apple.com/us/app/arcity-ar-navigation/id1282527727?ls=1&mt=8

indented to introduce content from third-partners. To improve the visual experience, this second level of information uses “occlusions”, meaning that only points of interest within the user’s eyeline are displayed, and not for example those hidden by a building.

- **Urban visual positioning**: recognition, positioning, and directional information via computer vision. Currently only available in Central London, San Francisco and Mountain View.

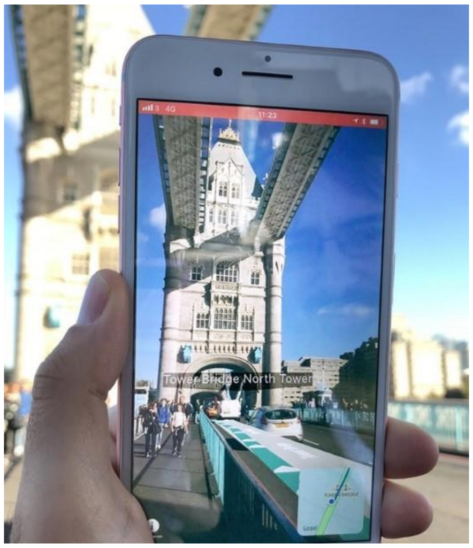


Figure 2.11 Example of a Navigation along a Route using the AR City application. The overlay of information on top of the Camera View.

2.5.6 Differences with our application

Navigation of the Virtual Tour:

The navigation on the AR applications listed in the sections 1.4.1, 1.4.2 and 1.4.3 is done on the user’s mobile device. The user has to physically walk (or drive) along a published route in order to experience its surroundings and objects of interest. The user walks along the path with his mobile device pointing upwards so that he can view and interact with the Augmented Reality overlays that are being displayed through his mobile camera. On the other hand, our system generates a video generated by panoramic pictures that can be viewed in an automated way both from his mobile device and his personal computer. It offers a similar way of interaction through the user of custom markers being displayed on top of the panoramic pictures. Those markers can act in a similar way with the Augmented Reality overlays, helping the user turn his attention to specific details on larger Objects of Interest.

Narration – Story telling

Another difference between those application and our system regards the narration across a Route. Some of the applications above offer (or will offer) the users the option to interact with the Augmented Reality Overlays that are displayed on the real world through their cameras.

Those overlays can be text, images, video or audio. The user has to point his camera at the point of the overlay to interact with it. Upon selecting it he can view the associate picture, video or read the description or hear an audio file. On the other hand our system offers storytelling on form of one or more narrations associated with an Object of Interest. Those narrations are attached to the Visibility Sections of the Object. Upon entering those sections the narration starts playing informing the users about the location of the Object or providing relevant information, so the user knows beforehand where to turn his attention.

2.6 Route Simulators with StreetView Integration

Another category of applications that integrate StreetView Imagery in their system are route simulators. Route Simulators can be in the form or Driving Simulators that project the route and provide instructions across the path or Time-lapse videos constructed by pictures across the Route.

2.6.1 Google Maps StreetView Player

This system[19] stitches Google's panoramic street view data into a short GIF movie. At the top of the page, the user can enter the start and end location of the trip. As the users clicks Play, the route is marked on the map to the right and the images will load and start playing on its left. If alternative routes are available, they can be selected from a table below the map. The user can control the pace of the time-lapse and pause the playback. If he enjoys the views, he can download the images as an animated GIF movie (see Figure 2.14.). The time-lapse is constructed with static StreetView images in contrast with our system that is based on StreetView Panoramic Pictures that offer more interaction with the surrounding of the Route. Additionally, this system retrieves only StreetView images thus not offering a simulated gif in parts of the Route where there is not available StreetView coverage. Finally, the system focuses only on the simulation of the path of the Route without taking into consideration Sights of Interest that may be located near (or far) and are visible from the Route[20].

[19] Google Maps StreetView Player: http://www.brianfolts.com/driver/

[20] YouTube video sample of StreetView Player: https://www.youtube.com/watch?v=TgTaXO1QW88

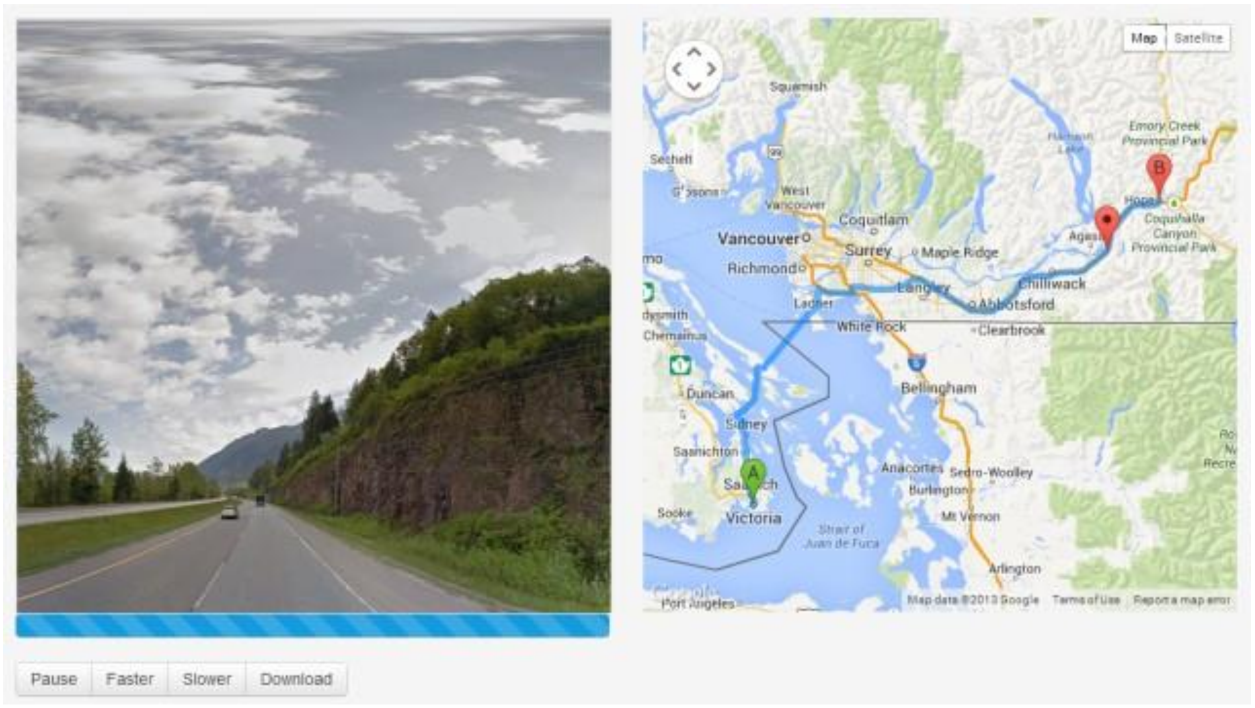


Figure 2.12 Interface of the Google Maps StreetView player. On the left screen the gif generated by the static images is projected while on the right there is a preview of the position on the map.

2.6.2 Gaiagi Driving Simulator

This online driving simulator[21] projects a generated video of the input Route while providing instructions to the users across the path. Users can define the start and end points of their desired tour in the top left. Directions will be shown directly below and they will be able to see their trip marked on the map in the top right. When the user starts the Driver mode[22], his progress will be shown in the satellite and map view on top, while he can enjoy the scenery in Street View mode in the bottom left. The bottom right shows either an up-close bird's view or panoramic pictures that were taken along the way. If he wishes he can stop the tour and resume it anytime. While the simulator is paused, he can click on Configurations button in the top left to adjust the settings. Much like the Google Maps StreetView player, this system focuses only on the simulation of the path of the Route, without giving attention to Sights of Interest that may be located near (or far) and are visible from the Route, thus the user does not get notified for possible attractions near him.

[21] Gaiagi Driving Simulator: http://www.gaiagi.com/driving-simulator/
[22] YouTube Sample video of Gaiagi: https://www.youtube.com/watch?v=UcHIsRzJhZ8

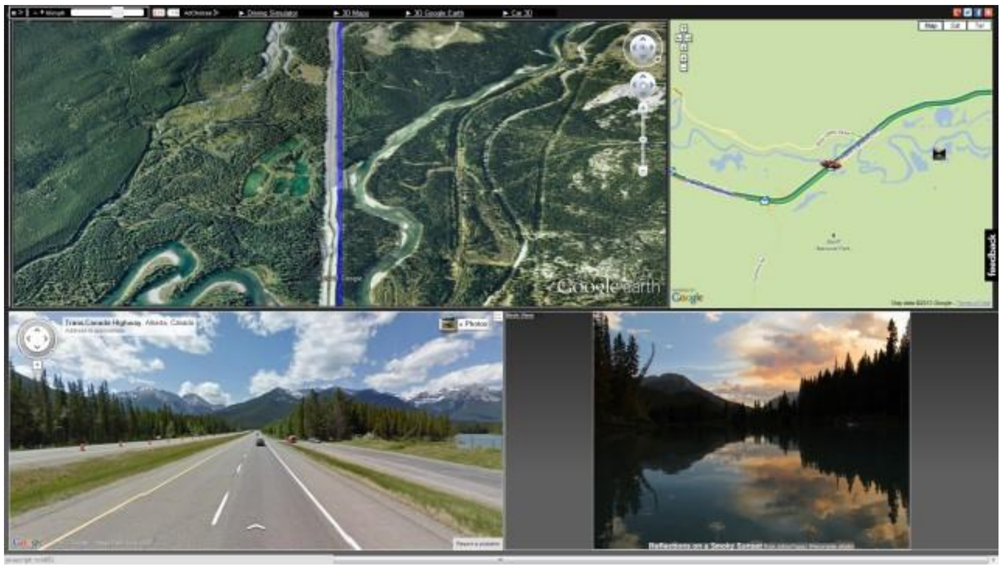


Figure 2.13. Interface of the Google Maps StreetView player. On the left screen the gif generated by the static images is projected while on the right there is a preview of the position on the map.

2.6.3 Hyperlapse

Hyper-lapse is a technique in photography that combines time-lapse with sweeping camera movements, often focused on a point-of-interest. Teehan-Lax Labs have combined the technique with images from Google Street View to create Hyperlapse. Hyperlapse is a demo product. Right now a user can create his own video by placing start (A) and end (B) markers on a Google map, wherever Google Street View images are available. Short stretches work best. The system offers an interface with a main screen that presents the Hyperlapse tour, by projecting a sequence of static StreetView images retrieved by the points of the Map. There is also a smaller screen located on the top right corner of the screen, providing him with information of the current position on the Route on top of a Google Map.

2.7 Technologies & Implementation *Architecture*

In this chapter a short description about the technologies we used in the implementation of our application is provided. Specifically, we are going to present the programming languages, the architecture and the tools we used within our application. In the subchapter 2.7.1 we are describing the markup language HTML5 and in 2.7.2 the programming language JavaScript that were used to design our User Interface and the client-side functionality of our application. We are also going to describe the Servlets in the subchapter 2.7.3, which we used to communicate with the Postgres Database. In 2.7.4 we are going to describe the Java Server Pages that helped us build dynamic pages in our application. In 2.7.5 we are going to give a short description about the Google Maps JavaScript API on which we build our application. In 2.7.6 we describe the PhotoSphereViewer library that we used for the handling of the Custom Panoramic Pictures.

Finally, in 2.7.7, we give a short description about the programming tools we used such as Eclipse and PostgreSQL (on which we build the database schema of our application).

2.7.1 HTML5

HTML5 is the latest version of Hypertext Markup Language, the code that describes web pages. It's actually three kinds of code: HTML, which provides the structure, Cascading Style Sheets (CSS), which take care of presentation and JavaScript, which makes things happen. HTML5 has been designed to deliver everything a user would want to do online without requiring additional software such as browser plugins. It is developed to do handle animations in applications, music and video files and also be used to develop complicated graphic applications that run in a browser. HTML5 isn't proprietary, meaning that a user is not required to pay royalties to use it. It is also cross-platform, which means it does not depend on the device the content is loaded. Whether the user is browsing through a tablet or a smartphone, a netbook, notebook or Ultrabook or a Smart TV, as long as the device's browser supports HTML5, it will work flawlessly.

HTML5[23] can be used to write web applications that are able work offline, applications that can retrieve information about the user's physical location and can handle high definition video and deliver high demand graphics. "It was published in October 2014 by the World Wide Web Consortium (W3C) to improve the language with support for the latest multimedia, while keeping it both easily readable by humans and consistently understood by computers and devices such as web browsers, parsers, etc. HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML."

2.7.2 JavaScript & Ajax

JavaScript is the most popular scripting language. Alongside HTML and CSS is one of the three cores of technologies of the World Wide Web content production. Today it is used on almost all web pages by adding functionality (component validation, browser distinction, server communication, response to events, creating cookies, reading and writing HTML items, etc.), and supported by all modern browsers (Internet Explorer, Firefox, Chrome, Opera, Safari). Its code can be embedded in an HTML file or called from one or more external files. Also its use is free (no licensed for use in commercial applications) as well as all the technologies used to develop our application. It is used to make webpages interactive and provide online programs, including video games. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded. Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime

[23] HTML5 Wikipedia Source: https://en.wikipedia.org/wiki/HTML5

environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets. We used JavaScript within our application to develop the functionality needed for the client-side of our system and also to do asynchronous calls to our Java Servlets through Ajax and JSON.

Ajax, stands for Asynchronous JavaScript and XML. It is the use of the XMLHttpRequest object to communicate with servers. It can send and receive information in various formats, including JSON, XML, HTML, and text files. AJAX's most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh the page. By using AJAX requests to render our interface we reduce the response time.

2.7.3 Servlets

Servlets are a technology used to capture and answer to requests from the web client, usually through the HTTP protocol. They are small pieces of code running on the web-server side. Each servlet has the following life cycle: at start it is constructed and initialized. After being called, the Servlet manages the request it received from the client and sends one answer as a response. Finally, after completing its work, the Servlet is destroyed. In our application this technology was used on the server side to responds to the requests submitted by the client. For example, if the user completes a form for creating a Route, a servlet is responsible for getting the input data, communicating with the database to store them and return the client a response about the status of his action. Servlets communicate with web servers through a container (Apache Tomcat for Apache Web Server).

The usage of Servlets has various benefits considering performance, usability, expressiveness, portability, low cost and security. **Performance**, it is not required to create a separate task for every received request. Each request is served by one thread, while the Java Virtual Machine loads only once and stays in memory. **Usability**, since they are written in Java, an ease to use and widespread language. **Expressiveness**, servlets can communicate with each other and with the server. They can also maintain information from request to request. **Portability**, through the appropriate container and Java Virtual Machine, Servlets can be installed in any web server or platform. **Low cost**, they can be used with free containers and free web servers. **Security**, a Servlet can be executed on a 'Sandbox' protecting the web server from any malicious Servlets.

The package responsible for the creation of HTTP Servlets is the javax.servlet.http. Every Servlet is a subclass of the HttpServlet Class. Each subclass inherits and builds at least one of those methods:

- DoGet: handles HTTP GET requests.
- DoPost: handles HTTP POST requests.
- DoPUT: handles HTTP PUT requests.
- DoDelete: handles HTTP DELETE requests.

- Init, destroy: capture and release resources.
- GetServletInfo: description of servlet.

When a user sends a request with a URL, the Servlets converts it to an HttpServletRequest Object and transfers it to the appropriate method which is defined by the URL. Upon finishing the task on the server side, the Java Runtime packages the result is in an HttpServletResponse and sends an HTTP Response to the client. Multiple requests and responses within a session are wrapped in an HttpSession Object.

2.7.4 Java Server Pages (JSP)

Java Server Pages (JSP) are a technology released by Sun Microsystems in 1999. It is a technology designed to help software engineers to create dynamically generated web pages. They are practically HTML pages which include Java language scripts that offer the dynamic content.

JSP are server-side application which means that they receive a request and returns a response. Those requests are made through a web client and the response is a produced HTML document which is sent back to the client. Since, the JSPs are server-side based it means they have access on server resources such as Servlets, EJBs and the PostgreSQL database.

There are many advantages on using JSPs. The use of Java language, which follows the write-once, run anywhere policy, allows JSPs to be able to run on any application Server that has a Java Virtual Machine. Another advantage is the use of tag libraries. The JSPs use tags, which are similar to those of HTML and XML, to introduce dynamic content. Tag libraries further define tags that can be used to replace code fragments.

Another important advantage of JSPs is the separation of roles. The specifications of JSPs allow to divide the load into two categories: in graphic content of the page and the dynamic content of the page. This means in practice that the group that does not know the Java programming language can create the graphic content of the page and a developer Java to create the dynamic content of the page. And finally that uses the powerful Java programming language to create the potential its content. This means that hundreds of classes are at our disposal (classes) and methods.

2.7.5 Google Maps JavaScript API

The main purpose of this thesis is to design routes within either a city, a region, a village or generally the nature (through a mountain, a gorge etc.) which will allow the users of the application to interact with the Points of Interest that are placed along their paths. These routes are constructed using the Google Maps JavaScript API. As we mentioned in the Requirements Analysis of our Model, the key factor that led us to using the **Google Maps JavaScript API** was that it included the Street View Coverage API. There are other APIs that offer imagery for streets and roads but the Street View Service developed by Google offers the best available coverage across most countries in the world, including both cities and countryside.

Besides offering the StreetView Coverage API, the Google Maps API allows us to display map objects on our application’s pages. On those map objects we are going to visualize the Entities of our application. Using the available shapes of the Google Maps Library we are able to construct the **Routes** representing them by Polyline Objects, the **Points of Interest** by Marker or Polygon Objects and finally the **Custom Panoramic Pictures** and **Markers** by representing them by Google Maps Marker Objects. A marker object identifies a location on a map. By default, a marker uses a standard image. Markers can display custom images, in which case they are usually referred to as "icons." Markers and icons are objects of type Marker. The Polyline Object defines a linear overlay of connected line sections on the map. A Polyline object consists of an array of LatLng locations, and creates a series of line sections that connect those locations in an ordered sequence. The Google Maps Geometry library also offers us some methods that can be used to construct the algorithms that handle the automated navigation across the **Routes** paths. We are going to analyze in depth how we used those Shape Objects and Geometry functions in the Application Analysis and Algorithms Chapter.

The **StreetView Coverage API** offers the methods that handle the requests to the StreetView service and return Panoramic Pictures in form of PanoramaData in response. Those Panoramic Pictures are used within our application to construct a sequence that is used to give the user a visual interactive presentation of the Route he has selected to view or to walk on (in case of a mobile Device). Street View images are supported through use of the StreetViewPanorama object, which provides an API interface to a Street View "viewer." Each map contains a default Street View panorama, which you can retrieve by calling the map's getStreetView () method. When you add a Street View control to the map by setting its streetViewControl option to true, you automatically connect the Pegman control to this default Street View panorama.

2.7.6 Photo Sphere Viewer

Photo Sphere Viewer is a JavaScript library which renders 360° Panoramic Pictures captured with Photo Sphere, the new camera mode of Android 4.2 Jelly Bean and above or by 360 cameras. It also supports cube panoramas.
Photo Sphere Viewer is pure JS and based on Three.js, allowing very good performances on WebGL enabled systems (most recent browsers) and reasonably good performances on other systems supporting HTML Canvas. WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. Three.js is a cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser. Three.js uses WebGL.

We used the **Photo Sphere Viewer** JavaScript Library within our applications in order to resolve two important issues we faced upon using the Google Maps JavaScript API. As we said on subchapter the **number** StreetView Service offers a large percentage of coverage in many countries. There are some cases though that roads, streets or areas in nature were not easily accessible by the StreetView car which meant it was not possible to capture Panoramic Pictures across their points. Our application gives the user the option to upload their own captured Panoramic Pictures either using their smartphone, their 360 camera or another Photosphere capable device. The Photo Sphere Viewer handles the projection of those pictures within our application, adding them to the sequence of the StreetView Panoramic Pictures. We handle these transitions, from the existing StreetView Panoramic Pictures and the uploaded Panoramic Pictures, by changing the visibility attributes of the two divs where the StreetViewPanorama and PhotoSphere Viewer objects are loaded.

One functionality that the **StreetView Service** did not offer was the placement of Marker Objects on a Panoramic Picture at a certain height (Point on the Y-axis). Even though, Marker Objects are within the supported Overlays within a StreetViewPanorama, alongside Info Windows and Custom Overlay Views, they markers appear at the base of the selected point of the Panoramic Picture. By displaying Panoramic Pictures within the PhotoSphere Viewer Object we can take advantage of its Marker library. The PhotoSphere Viewer Markers offer the option of placing the Icons, Polygons, Circles and Text on a Panoramic Picture on a set of X – Y coordinates in pixels. Therefore, we can retrieve Panoramic Pictures from the existing StreetView Pictures and use the Photo Sphere Viewer to project them, gaining that way the ability to place Markers on top of them. We are going to give an in depth description of the algorithms used to retrieve the Street View Panoramic Pictures source (jpg /png) in order to be able to load them on the panorama attribute of the Photo Sphere Viewer Object. We are also going to analyze the algorithms that calculate the X-Y pixels coordinates on which we place the icon or polygon Markers within the Panoramic Pictures, both in the case of the Existing Panoramic Pictures and the case of the Panoramic Pictures captured and uploaded to the application by users.
Those algorithms are based on spherical geometric equations which are presented on the Application Functionality and Algorithms chapter, alongside all the additional functionality of the Photo Sphere Viewer we used within our application.

2.7.7 Materialize CSS and JavaScript Library

In order to give a modern and material design to the interface of our application. Material Design, created by Google, is a design language that combines the classic principles of successful design along with innovation and technology. Material Design makes more liberal use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows. Google's goal is to develop a system of design that allows for a unified user experience across all their products on any platform. The Materialize[24] library offers us the CSS classes and JavaScript methods to present our HTML elements in a material format. We used this library to

[24] Materialize CSS: http://materializecss.com

create an interface familiar to the users, since Material design is the interface that Google uses in its Android Operating System (OS) that a large percentage of people use through their mobile devices. We take advantage of the cards class in the Materialize CSS file to design the main divs within our pages and the Points of Interest. We also use its classes in our forms, lists and popup windows. We had to modify those classes in order to adjust them to our color scheme and change their height and width attributes to fit to our HTML elements in a better way.

2.7.8 Programming Tools

Eclipse IDE

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and is written mostly in Java. Its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including C, C++, JavaScript, Perl, PHP, Prolog, Python, R, Ruby.

The initial codebase originated from IBM VisualAge. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Since the introduction of the OSGi implementation (Equinox) in version 3 of Eclipse, plug-ins can be plugged-stopped dynamically and are termed (OSGI) bundles.

Eclipse[25] software development kit (SDK) is free and open-source software, released under the terms of the Eclipse Public License, although it is incompatible with the GNU General Public License. It was one of the first IDEs to run under GNU Classpath and it runs without problems under IcedTea.

We used the Eclipse IDE to construct the classes that handle the matching between the Entity Classes and the tables in our database, the Object classes that handle the matching from the received MultipartForm Data to the entities of our application and finally to construct the Java Servlets that accept the client-side requests, communicate with the database and return the appropriate response depending the type of request. Using Java methods we are able to handle the upload of image, audio and video files transforming the Base64 encoded data to binary data which we can handle and store in the server in form of image (.png/ .jpg), audio (.mp3 / .mp4) and video (.mp4/ .mpeg) files. The communication with the PostgreSQL database is achieved by the use of the JDBC driver. A JDBC driver is a software component enabling a Java application to

[25] Eclipse Wikipedia Source: https://en.wikipedia.org/wiki/Eclipse_(software)

interact with a database. JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers. To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. Therefore by passing the appropriate driver we achieve the connection with the PostgreSQL.

PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users. In PostgreSQL, a schema holds all objects (with the exception of roles and tablespaces). Schemas effectively act like namespaces, allowing objects of the same name to co-exist in the same database. By default, newly created databases have a schema called "public", but any additional schemas can be added, and the public schema isn't mandatory. We used Postgres to design the schema of our database.

2.7.9 Google Camera

For the capture of the Panoramic Pictures we used in our system for the construction of UserCaptured Panoramic Pictures, that either add or replace parts of the Route that may or may not have StreetView coverage, we used the photosphere mode of the Google Camera Android application. A Huawei Nexus 6P mobile device was used to capture those Panoramic Pictures. The Nexus 6P has a rear facing Sony Exmor IMX377 sensor with f/2.0 aperture that can take 12.3-megapixel (4032 by 3024 pixels) photos. Its large pixel size of 1.55 micron assists photography in low light conditions. The photospheres were captured in normal resolution to minimize the loading time in the PhotoSphereViewer object.

Photo Sphere is a 360-degree panorama feature Google added in Android 4.2 (and originally with the Nexus 4) that lets users take immersive pictures with their mobile phone, then share them online. Users can pan and zoom, much like they can with the Street View feature on Google Maps. Through, the Google Camera application we can record the GPS coordinates of the location the user captured the Photosphere. The user has to enable the location on his mobile device before start shooting the Photosphere. The basic steps of capturing a Photosphere are listed below:

Take a photo sphere

Open your camera.

1. Tap Menu ☰ › **Photo Sphere**.
2. Hold the target circle over a white dot. The dot will turn blue, then disappear.

3. Move the camera to the next white dot, holding the lens steady.
4. Repeat until there are no more white dots, or tap done ✓ .

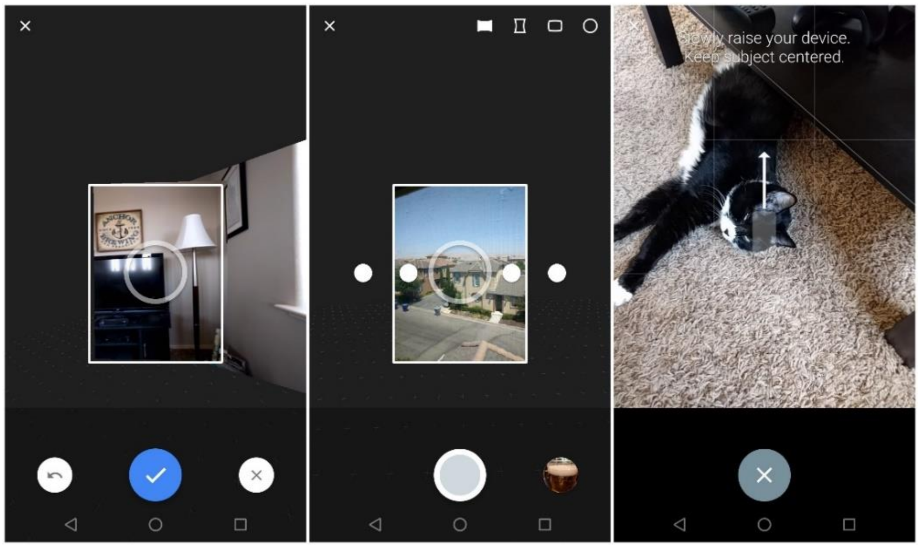


Figure 2.14 Process of Capturing a Photosphere using the Google Camera application.

Chapter 3

3 Functional Overview & Contributions

3.1 Introduction

In this chapter we are going to present the general functional requirements of our system and the research contributions of this work. Then we will present the information modeling of the system and the reasons we decided to construct each information object and functionality. The detailed analysis of the application's functionality will be described in the next Section. We are going to give an introduction of the Objects and functions on which our system is based.

3.2 Functionality Overview and Contributions

One of the best ways to get to know a city is to walk through the streets of the city. The navigation through the streets of the city gives a feeling of the life of the city, its architecture, the interrelationships of the features of the city, etc. Visitors with specific interests may want to view in detail objects of interest (like cultural objects of a specific era) and their interrelationships with other objects and the architecture and life of the city. However, the time that a potential visitor has in a city is typically limited, and he can only plan to follow certain routes and see objects of interest in those routes.

The objective of our system is to be able to create personalized and immersive culture and tourism applications that simulate (and also support through mobile devices) the navigation of the user through routes of interest of a geographic area or a city. The navigation is supported by the dynamic generation of personalized time lapse video and storytelling through routes of a city, as well as interaction with its objects of interest during the navigation, while at the same time offering the user real-time interactive control that enables him to change whenever he wishes the focus of his attention thought the route, thus taking a closer and more detailed look of his surroundings and objects that attract his attention.

To satisfy the needs of our system we proceeded in the development of three operational modes. One that handles the viewing of pre-specified virtual routes in an area and dynamically generates personalized time lapse video based on the user interests and supports the user navigation using this video. A second that dynamically generates time lapse video based on camera control parameters interactively determined by the user during his navigation. These two modes can be intermixed so that the user can switch very easily from one mode to the other and back. Lastly, a third mode that supports the user himself navigating in an area or city carrying a mobile device like a phone or a tablet and the application guides him through the objects of the city that match his personal interests.

In the first mode the user experiences a navigation through a planned route of his choice on the browser on his personal computer, following the pre-specified path. The personalized time lapse video is constructed based on his interests and focuses the attention to objects of his interest if they exist as the navigation progresses through the route and any associated narrations are played. The user can interact with these objects of interest in order to find and view more detailed information, in the form of pictures, audio or video about them, if he wishes. The objects of interest can be small or large and can be located at any height of the nearby buildings or ground formations, or they can be virtual.

In the second mode of operation the user immerses and interactively controls the video generation during the navigation through a route of the city in order to see and explore sights and objects along the route that attract more his attention. He is provided with capabilities that allow him to interactively control the direction, the zooming of viewing (zoom in and zoom out, the timing of presentation, to stop and look around him (pan movement of the head), and then continue, and also to explore in more detail information on specific objects. The user is also provided with information about the existence of objects of interest near the current location of the user that may or may not be currently visible by the user. The user may decide to diverge from his current route in order to view and interact with these objects.

The first and second functionalities combined offer the user an immersive environment supported with real time interactive, dynamic time lapse video generation that allows him to selectively interact with and explore objects according to his interests across a path, but also navigate and visually explore in a more active way other objects along a route of a city or area of interest.

In the third mode of operation the user navigates through a route by actually walking or driving, for example by walking along a route of a city, carrying a mobile device (such as a phone) with him. The mobile device notifies him when he approaches objects of interest along his path and if the user wishes he can be helped to view where in the 3D space the objects are located as he is approaching them. Much like the other two modes he has the capability to navigate towards the objects, be helped to locate them, hear narrations associated with them, and interact with them in order to find more information.

The three modes of operation are usable by any device that has access to the web. In the case of the mobile access, access to GPS functionality is required.

A number of challenges had to be solved for the implementation of such a system. The system has to be capable to construct cultural or tourism applications for any city or a geographic area.

We use Google maps to present the area of interest to a specific application. On top of the map specific routes of cultural or other interest are defined. The user can follow a complete route or a part of a route starting from an arbitrary point along the route. Objects of interest are defined along with their footprints, exact location (in 3D view), type of object and additional information enabling user exploration. In addition, sections along the route, from where the objects of interest can be viewed (viewing sections), and camera parameters that enable good visual observation of those objects can be specified.

The time lapse video presentation of a route is not predetermined but is dynamically assembled based on the personalized interests of a user so that he can focus his attention, during the navigation through the route, on object types or objects of interest to him. Thus at each point in his navigation he has immediate sense of the objects of his interest near his current location and he can view them, hear narration about them, and directly interact with them. However, the system also allows complete interactive control to the user. The user can stop the automatic navigation and look around him, or he can control the direction of viewing on a section of the route in which case the system will dynamically assemble a video for this part of the route according to the user's interests. The user, if he wishes, can also see and interact with objects of interest that are not preselected by the system according to his interests. He can then continue his route as planned originally. This functionality delivers a personalized, interactive, dynamic time lapse video based, immersive environment for navigation through an area, supported also by storytelling.

In addition, the user can have more direct control, if he wishes, on his navigation on a selected route. He can exit and reenter at any other point of the route so that he selectively explores only parts of the route that he wants, or he can revisit parts of the route that he liked or possibly revisit for viewing in detail objects that he may have missed during the personalized navigation. The system allows also direct access from the footprints of objects of interest to parts of the route that provide good viewing and storytelling for a specific object of interest thus providing an alternative way to view only certain selected objects of interest, in addition to the sequential navigation through the whole city route.

We have developed algorithms that automate as much as possible the dynamic video creation and viewing of objects of interest during the user navigation across a route. The interface allows both the route designer (application designer) as well as the final (end) user to view at the same time the route followed on top of the Google map, the current location of the user, and the information object footprints on top of the map. In addition the interface allows the user to view the video of the navigation played so that he has a good spatial understanding of his current views and the information of objects of interest. The algorithms that derive the dynamic video of a route take into account not only the information objects and the visibility sections along the route associated with those objects, but also the changes in the direction of the route (turns, curves, corners, heights, etc.).

Google Maps and StreetView only support markers to identify locations along a path. Markers are associated with a specific GPS location, not with a larger extent of the footprint of a large building or other object. The application however should be able to exploit larger footprints of objects in order to provide more information to the end users, and also in order to define the best viewing parameters as the user approaches those objects from different locations. It should also be able to associate different parts of narration with different visibility sections for those objects.

It is important to point out that objects of interest are not always easy to identify even when their approximate GPS location is known and visible on top of the Google map. The reason is that these objects of interest may be small or specific details on larger object and they may be located high up on a building (like an architectural detail, a Venetian Thereos, etc.). In addition, in some cases these objects are located far from the user's location on the route, visible in a distance, and the viewing from the route may show several other objects, thus making the identification of the object of interest difficult. To be able to specify the location and views of various shape objects that may not be located on the ground or near the route we had to develop additional functionality to extend the simple marker offered by Google with footprints of larger objects and viewing the location of those objects in the third dimension.

Our system dynamically generates time lapse video during the navigation through a route from panoramic (360 degree) pictures that have been taken along the route. The system records with every Panoramic Picture the angle with which the Panoramic Picture has to be rotated in order to be directed (point) to the True North. The dynamic video is produced by restricting the angle of viewing and determining an appropriate direction of viewing for each location of the user during the navigation along the route. The calculation of the parameters for viewing takes into account the user interests and the location of the objects on the Google Map. A rich set of viewing functionalities is provided by the system. These functionalities are available to the application designers for producing the most appropriate viewing of objects of interest during the navigation. The speed of navigation is variable, allowing the designer to allow for more time in the visualization and narration of the more interesting areas or objects of interest as well as faster viewing times for the navigation of the whole route.

A very important requirement in the development of the system was that the applications designed for different cities and areas of interest should be able to exploit as much as possible the panoramic pictures (360 degree pictures) of the StreetView service of Google wherever they exist. Whenever it is possible the system uses the StreetView panoramic pictures to dynamically identify parts of those pictures that can be used to synthesize the video that will be projected to the end user.

However, StreetView Panoramic Pictures do not exist everywhere. Even in cities that are well covered by StreetView content, routes that are of specific interest often do not have a good percentage of StreetView coverage across their paths due to various difficulties. For example, navigation along a large city Fort may require the user to follow paths that are not along the main roads of the city and as a consequence these paths have little chance to have associated StreetView Panoramic Pictures. Another problem may be that the StreetView Panoramic Pictures are taken with focus on the route's direction rather than paying special attention on objects of interest along the route. Therefore, it is possible that certain objects of interest may have been blocked by traffic, people walking, were under renovation at the time that the StreetView panoramic pictures were taken, etc. thus making their presentation for users that have specific interest for them unsatisfactory. There is a need therefore to be able to capture parts of a route through panoramic pictures in addition to the existing StreetView panoramic pictures.

Our system offers functionality that integrates both StreetView content and panoramic pictures along a route captured by users (route designers for the city tourism offices, etc.). The functionality offered for both cases is symmetric. It must be noted that the integration of panoramic pictures captured outside the StreetView is important also in order to define objects of interest that are at a certain height since StreetView and Google Maps do not offer such functionality.

In the following Section we are going to give a description of the main information objects that we have used in the development of our system and outline how we handled those challenges in order to develop the system's functionalities. We will provide an in depth analysis of the services offered and the algorithms used in the Application Functionality and Algorithms chapter.

3.3 Information and Services Model

In this section we are going to outline the basic information objects used in the development and use of our system.

- A **Google Map** is a representation of an area or a part of a city using the technologies and information of Google Maps and its associated JavaScript API.

- The **Google Maps JavaScript API** is used to interact with the maps of Google and gain access to the StreetView Service API.

- The **StreetView Service API** handles the requests for the StreetView panoramic pictures we use to generate the personalized video of a route.

- A **Google Map Route** (or **Route**) is an object that represents a navigation through a geographic area or city. It is described by a polyline object which indicates a path that the user follows. It has also a name, an Identifier, a type, a description and length (duration).
- **Panoramic Pictures a**re associated with points on a Route. They cover 360 degrees of view. Each Panoramic Picture has the GPS **Coordinates** of the camera which took the picture at the time it was taken. A Route can be subdivided int**o Route Sections**. Route Sections are consecutive parts of the polyline of the Route that have the property that each Route Section uses for the creation of time lapse video panoramic pictures of a specific type (StreetView panoramic pictures or User Captured Panoramic pictures). Depending on the type of pictures that it uses it can be **a StreetView Panoramic Route Section** or a **User Captured Panoramic Route Section.** A panoramic picture has **Orientation** which is defined by APN, the Angle of the direction of the camera when it takes the Panoramic Picture with the direction of the True North. This is easily done when the camera also has a compass API.
- If the camera does not support a compass functionality then the value of this parameter can be found interactively by finding the angle of the camera with respect to the road ahead or with respect to a visible object far from the camera. The direction of the True North can be computed from the direction of the road visible in the panoramic picture and pointed by the user by turning the panoramic picture to be directed towards the direction of the road. The Google map can be used to find the angle of the road with respect to the True North. If AR degrees is the direction of the road in the panoramic picture (indicated by the user), and ANR degrees is the direction of the road with respect to the True North (computed from the Google map), then the angle of the true North in the panoramic picture is APN = ANR + AR. The computation using a visible object in a distance is similar (finding the angle between the direction of the camera and the line from the camera to the object using the panoramic picture, and also finding the angle between the line from the camera and the object with the True North using the Google Map). PAN is stored with every Panoramic Picture, and used in real time to calculate the parameters needed for the dynamic video creation.

- The Panoramic Pictures in this model are compatible with the **StreetView Panoramic Pictures,** although the StreetView API hides some of those parameters and performs the calculations in the background**.** Any Panoramic Pictures that have not been taken from StreetView will be called **UserCaptured Panoramic Pictures**.
- A **Projection View** is what the user sees when he is looking at a specific direction. It is a part of a Panoramic Picture and is defined by the parameters described next.

- **True north** (geodetic north) is the direction along the earth's surface towards the geographic North Pole. Google True North on Google Maps is not shown, but for a normal Mercator projection, grid north and true north will coincide and it will follow any vertical line (or meridian) to the top of the map.

- The **Field of View** parameter is an angle that defines the extend of the observable world that is projected in the computer display at a certain point in time (90 degrees by default).

- The **Heading Angle** is an angle that defines the rotation of the focus of the camera on the x axis, for producing the Projection View (measured with respect to the true North).

- The **Pitch Angle** is the angle that defines the up or down direction of the camera in order to produce the Projection View.

  These parameters (angles) help to define a square window, on top of the 360^0 Panoramic View, that defines the Projection View. Given the GPS location of a user, and the location of an object on the Google Map, the direction (with respect to the true North) and the tilt of the camera in order to view the object can be calculated, and the coordinates of the window that the user has to view on top of a Panoramic Picture can be computed taking into account the angle of the direction of the panoramic picture with respect to the True North (which is stored in every panoramic picture).

- **Objects of Interest** are objects that have some importance to the users that follow the Route. They have an Identifier, a name, one or more types, a description, a picture, related links and a visual method to identify them. This method (**Object Location Visualization**) may be only on top of the Google map (through a Marker or footprint) or it may also be in term of an object Enclosing rectangle. A **Google Maps Marker** object can be used to point to the central location of an object on top of a Google Map. Alternatively for larger objects (for example a fort) the **Object Footprint** on top of the Google Map can be used. The footprint can be a form of a polyline or a polygon.
  In addition to the representation of objects on top of a Google Map, the system has the capability to represent these objects on top of individual Panoramic Pictures through a Google Maps Marker or an **ObjectEnclosingRectanlge**. The ObjectEnclosingRectanlge can only be done on top of Panoramic Pictures that are UserCaptured Panoramic Pictures. If there is a need to indicate the actual location of Objects in the third dimension on top of StreetView Panoramic Pictures, since Google does not offer this functionality, the system provides the capability to use the ObjectEnclosingRectangle in a sequence of User Captured Panoramic Pictures which can replace the StreetView Panoramic Pictures associated with those objects.

To facilitate user understanding and interaction a special marker, **ObjectEnclosingRectangle Marker** that is placed on top of the Google Map identifies locations from where ObjectEnclosingRectangles are used on top of User Captured Panoramic Pictures in order to indicate the exact location of objects of interest.

- In addition to the Objects of Interest, there are **Areas of Interest** and **Roads of Interest**. Their Visual Identification is similar to the Objects of Interest.

- **Object Presentation Sections** are sections of a Route on top of a Google Map that define specific visualization algorithms and parameters for creating the Projection Views of the time lapse video in this section of the Route and an attached associated voice narration. The default algorithm for creating the time lapse video using the Projection Views of existing Panoramic Pictures produces a visualization that resembles what a user that walks the Route looking ahead sees. In an Object Presentation Section the time lapse video produced uses one of a set of algorithms that has been chosen by the application designer for visualization of an object or area, if this object or area is of interest to the user.

- The application designer has the capability if he wants to specify a **Navigation SlowDown** in an Object Presentation Section. Navigation SlowDown allows slower speed of navigation to deliver better presentation and narration quality. The SlowDown enables the simulation of actual tourists and their tour guides when they approach areas of exceptional beauty or objects of high interest and much detail, where the tourists need more time to absorb the visual and narrative information. Navigation Slowdowns support the requirements to give more time to the final user of the application to observe objects of interest or wider views. For example in an area of interest where many objects of interests have to be observed and narrated, a Navigation Slowdown is required to give that time to the end user. The Navigation SlowDown can be such that arbitrary time is spent near a location in the navigation using smaller steps between two consecutive positions in the navigation so that enough time for narration and viewing is provided at a certain Viewing Section.

- The system provides different algorithms that may be used by the application designer for viewing an object or area of interest. These algorithms and parameters are associated with different Object Presentation Sections that have been specified on top of the Google Map. ObjectEnclosingRectangle Markers may be placed on top of the Viewing Sections to indicate the existence of more detailed information on where exactly the object is located.

  A particularly important use of the Object Presentation Section is when the user passes near an Object of Interest. In that case different algorithms in different Viewing Sections can be used to get overview or details of the Object of Interest as the user is walking by and also listen to the related narration. For example the system can calculate the angles to keep the object of interest in focus as the user passes by, or keep viewing all the object (for larger objects) or keep looking at a specific angle towards a larger object. Another example of algorithms simulates a user that walks with his head turned in an angle to the left because in that direction the views are more interesting (for example passing near a long fortification wall of an old city). Manual decisions on the camera direction, pitch and focus as the user walks are also facilitated and stored by the system.

- A **Panning Head Action** may be used to show a wider area around the user (up to 360 degrees around him for a given Head Tilt or Projection View Pitch). The Panning Head Action may also be used within a Navigation Slowdown to give an overview of an opening wider view with several interesting objects and may have a short narration and presentation timing associated with it. A **Tilting Head Action** may be used to show up or down (for a given Head Heading or Projection View Heading in the x axis) for example for viewing a very tall buildings that is located very near to the user. Both actions may be useful for viewing large nature objects or open areas with wide view, but they may be also very useful for the presentation of very large objects of interests like a large church Cathedral with many details and a long narration on its history composed of several Objects of Interest each with its own narration.

- **A Personalized Tour Visualization** is a sequence of images projected as a time lapse video. The images are Projection Views taken from a sequence of points along a Google Map Route. The Projection Views of a specific Guided Tour Visualization are selected based on the user interests and aim to focus the camera into objects of interest to the user as the user navigates through the route. Thus a Guided Tour Visualization is personalized to the interests of the final user.
The applications on a city or area allow the end users to specify a subset of Objects of Interest as important to his interests depending on their types, or even individually selecting some objects. The system will present all the Route synthesizing time lapse video with default parameters if the Object Presentation Sections do not contain Objects of Interest to the user. The end user can specify faster or slower speeds for navigation through them. For Object Presentation Sections that involve Objects of Interest to the user the specified algorithms for each one of them are used to create the time lapse video and determine the navigation speed to fit the narration time and presentation requirements.
Although the personalized presentation proposed by the system is followed by default, the user can interactively alter the algorithms and parameters for viewing. For example as he proceeds through a Route he may want to stop the navigation to explore around him in the same Panoramic Picture. Alternatively, he may want to proceed to on the Route but he may

not want to follow the default presentation (Projection Views that are directed ahead on the road or Projection Views constructed to visualize a specific Object of Interest on a Route or a Visibility Section). He can then define with simple interactions in real time his preferred Viewing Method which involves method and camera parameters for constructing, with simple interaction, the Projection Views in the current part of the Route that he is navigating. For example, he may define that he wants to walk with his head turn left in order to see an interesting wider view in that direction.

- A **Personalized Guided Tour** involves a Personalized Guided Tour Visualization and narration along the Route. Narration can be associated with different Visibility Sections and Objects of the map. It allows the final user to follow a predefined route through the streets of the city and view its parts, monuments and objects of interest, hear narrations and view related information.

In the remaining of this section we discuss in more detail how the main information objects discussed above are used to produce the functionality that the system offers.

3.4 Motivation and Information Details

As mentioned before, a Guided Tour is associated with a Google Map Route which is specified on top of a Google Map through a polyline which is defined relative to the coordinates of the map using the Google map API. We could have based our application using a different Map API and not the one developed by Google, for example the Here Maps API (by Nokia), which also provides the functionality of creating routes, add markers and specifying areas on the map that we use within the application. We decided to build our application on the Google Maps API, based on the fact that the main focus of the application is the visualization of each guided tour as well the visualization of the user’s navigation alongside a map when using his mobile device. Google Maps API has the great advantage of including the Street View Image API, which is the one handling all the requests for the panoramic images we are going to use. In our knowledge there are not any other APIs with similar technology as the Street view imagery, but even if there were they would not have the extended coverage Google has achieved over the years.

A Google Maps Route is a polyline object which indicates a path set in nature or in a city that the user follows in order to see parts of the city or the nature in its proximity. The construction of these paths is done interactively on top of a Google Map Object by assigning an array of GPS coordinates of each point that we want the route to pass by. The system simulates a walk of the user following this path using a video produced through a sequence of pictures along the path. Those pictures can be for example the responses we get by the Street View request at each point of the route or by picture sequence captured by other devices (smartphones, 360 cameras, etc.) providing the same functionality. The points that constitute a Route on a Google Map, do not need to have corresponding Street View image because the request is returning a picture that is found in a defined radius, which defines the distance in meters around each point's coordinates where the Street View request looks for panorama data. At each point in the route the simulation uses a picture in the direction that the user looks, and the video produced by the projection of the sequence of pictures in specified time intervals gives the visualization of the Guided Tour.

Normally the pictures of the visualization of a tour are taken from the Street view of Google for each point of the Route (whenever they exist) or a compatible technology (see discussion later). Each picture along the Route has an associated position on the map, from where the picture was taken (given by its GPS coordinates). The pictures captured are Panoramic Pictures, e.g. they cover 360 degrees view of the scenery around the position from which the picture was taken. A Panoramic Picture is a two-dimensional picture, it has an X axis which spans 360 degrees of view around the position from which it was taken, and a Y axis that spans 180 degrees the angle vertically from the position that the picture was taken (from +90 to -90 degrees with respect to the horizontal). The Panoramic Picture is defined by the GPS coordinates from where the picture was taken, the Viewing Focus variable that defines the rotation along the X-axis and the vertical direction of the camera on the Y-axis. These pictures are compatible with the StreetView mode of Pictures and thus the system can utilize StreetView Map pictures when they exist (and when we want to use them).

A Projection View is what the user sees from the position that he is looking at, with a specific direction. The projection view is a part of the panoramic picture and it is defined by a Viewing Focus which consists by the Field Of View, a Heading Angle for the X direction and a Pitch Angle for the Y direction. The Field Of View parameter is an angle (90 degrees by default) that defines the extent of the observable world that is seen on the display at any given moment.). The Heading Angle is an angle that defines the rotation around the camera focus for an image from the true North. This angle defines the part of the full Panoramic Picture the camera is pointing at. The Pitch Angle is the one that defines the up or down direction of the camera. These variables determine a square (field of view is 90 degrees for both axis, so they conclude a square) of viewing on top of the Panoramic Picture. The Projection View is the part of the Panoramic Picture enclosed by this square. We can zoom in and out in the view by changing the zoom attribute which affects the Field of View parameter. By zooming in, the Field Of View is decreased so we have a more narrow view of the panoramic picture. Accordingly by zooming out results in increasing the Field Of View, that way we end up with a wider view of the Panoramic Picture. We can see an example of how those angles are defined on top of a picture on figure 3.1.

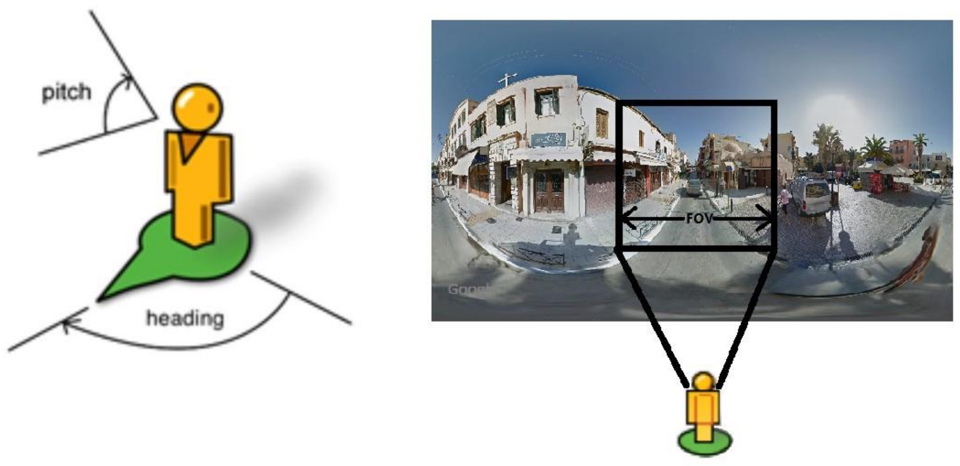


Figure 3.1 Heading, Pitch and Field of View angles defining the Projection View on top of a Panoramic Picture.

The StreetView service that returns the StreetViewPanorama response, containing the Street View Map Panoramic Pictures, offers four discreet levels of zoom. The StreetViewPanorama of View angle for each zoom level, the calculated angles are shown on Table 3.1.

Table 3.1. Street View Zoom Level depending on Field of View

Street View zoom level	***Field of View (degrees)***
0	***180***
1 (default)	***90***
2	***45***
3	***22.5***
4	***11.25***

Based on the values of the Table 3.1 we can define the function that calculates the final Field of View at any given zoom attribute.

Equation 3.1: *fieldOfView (zoomLevel)* = $\frac{180}{2^{zoomLevel}}$

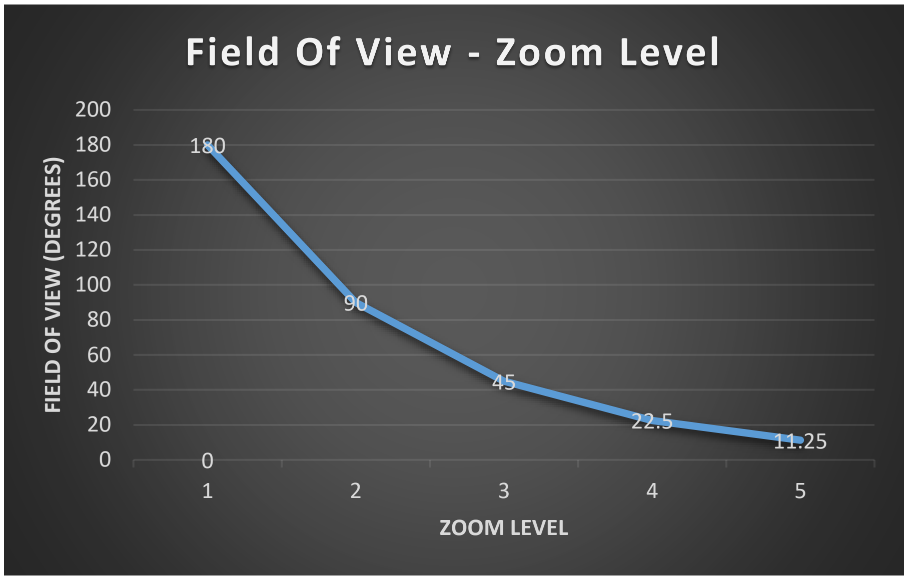


Figure 3.2 Field of View - Zoom Level Diagram based on equation 3.1.

As we mentioned above, the Field of View, the Heading and the Pitch angles consist the View of Focus parameter which defines the Projection View. In order to better understand how the final Projection View is calculated and presented to the user, we can have a look to the figure 3.3.

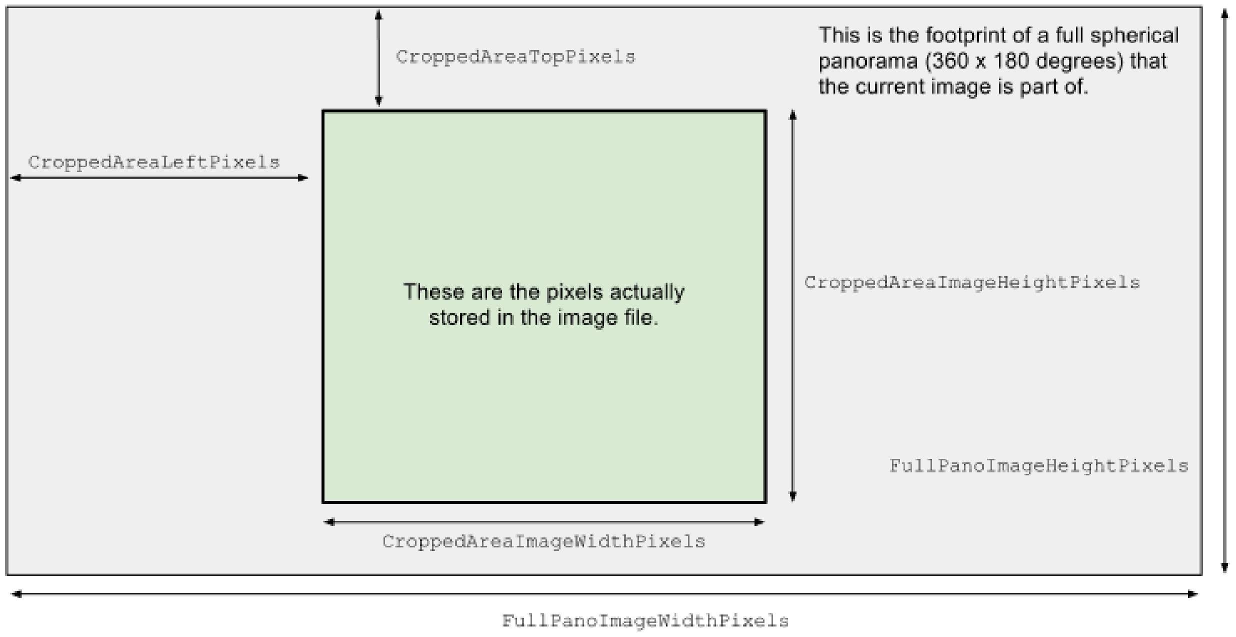

CroppedAreaTopPixels
This is the footprint of a full spherical panorama (360 x 180 degrees) that the current image is part of.
CroppedAreaLeftPixels
These are the pixels actually stored in the image file.
CroppedAreaImageHeightPixels
FullPanoImageHeightPixels
CroppedAreaImageWidthPixels
FullPanoImageWidthPixels

Figure 3.3 Parameters of the Projection View Diagram.

CroppedAreaLeftPixels: the coordinates where the left edge of the image was cropped from the full sized photo sphere. This attribute is defined by the heading angle, rotation of the camera on the x-axis, which calculates the number of pixels that the cropping of the current image view starts on the full panoramic image.

- **CroppedAreaTopPixels**: the coordinates where the top edge of the image was cropped from the full sized photo sphere. This attribute is defined by the Pitch Angle, up-down of the camera, which calculates the pixel position the current image view is cropped of the full panoramic image.
- **CroppedAreaImageWidthPixels**: the cropped image width in pixels of the image .This attribute is defined by the FOV angle which is by default 90 degrees.
- **CroppedAreaImageHeightPixels**: the cropped image height in pixels of the image .This attribute is defined by the FOV angle which is by default 90 degrees.
- **FullPanoImageWidthPixels**: the original width of the panoramic picture in pixels.
- **FullPanoImageHeightPixels**: the original height of the panoramic picture in pixels.

In order to design the diagrams for the Angle of Viewing and the Pitch angle we are going to use the following types, which we explain in depth in the technical analysis.

The Heading Angle parameter depends both on the value of the **Center Heading** angle, which defines the rotation from the real Panoramic Center as it is given by Google or the user that uploaded the corresponding 360 Panoramic Picture and the value of the user define bearing angle. Therefore we have the equation 3.2 for the calculation of the CroppedAreaLeftPixels:

Equation 3.2

- *centerHeading <=180*

$$X - Axis\ Center = \frac{heading}{360}\ panoramaWidth - \frac{180 - centerHeading}{360}\ panoramaWidth$$

- *centerHeading > 180*

$$X - Axis\ Center = \frac{heading}{360}\ panoramaWidth - \frac{360 - (centerHeading - 180)}{360} panoramaWidth$$

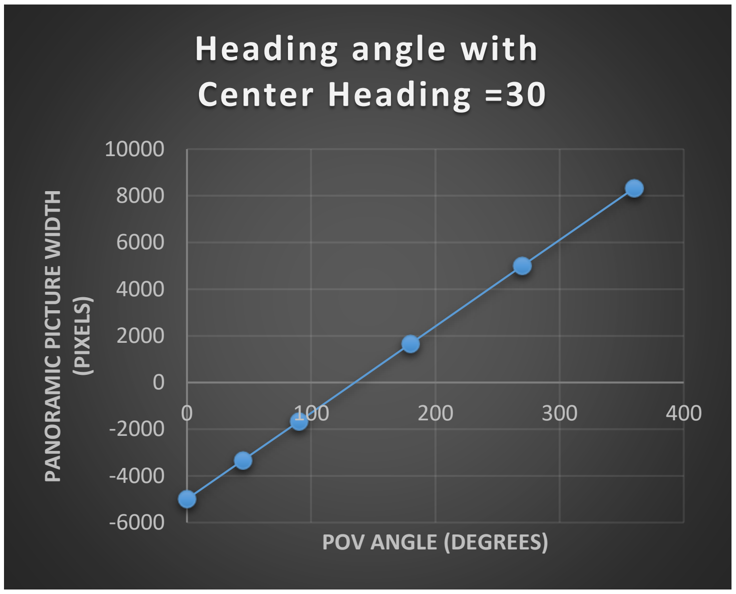


Figure 3.4 Pixels coordinates to crop on Original Width of the Panoramic Image, to return the appropriate Projection View. Case Center Heading <=180.

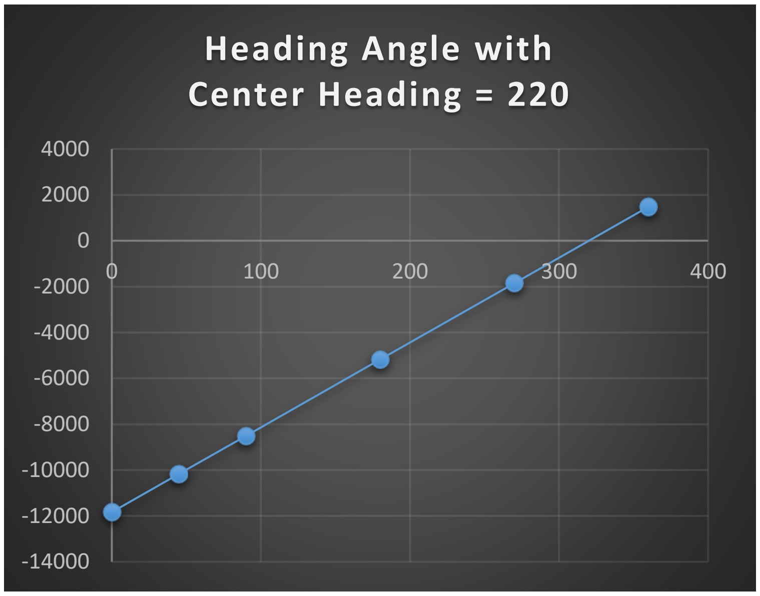


Figure 3.5 Pixels coordinates to crop on Original Width of the Panoramic Image, to return the appropriate Projection View. Case Center Heading >180.

Accordingly for the Pitch Angle parameter we use the following type:

$$\textbf{\textit{Equation 3.3}}: Y - Axis\ center = \frac{90 - pitch}{180}\ panoramaHeight$$

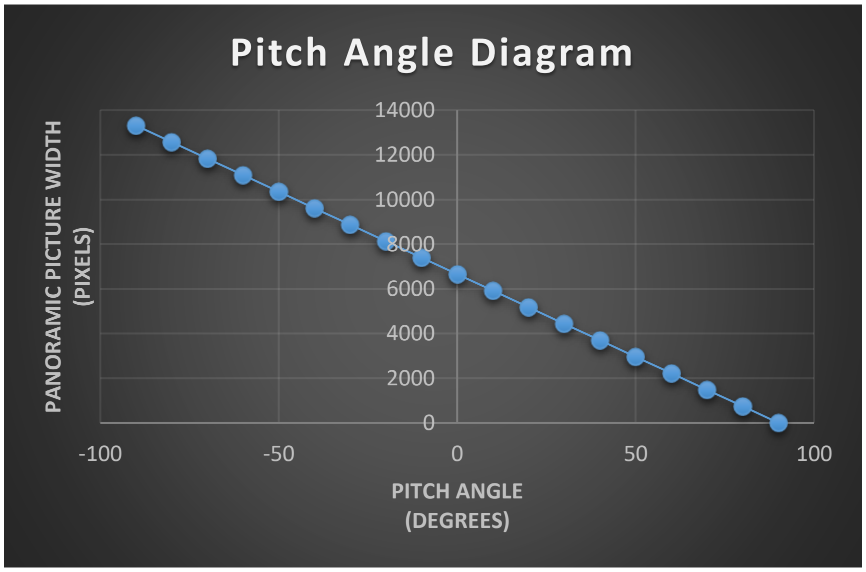


Figure 3.6 Pixels coordinates to crop on Original Height of the Panoramic Image, to return the appropriate Projection View depending the Pitch angle value.

The Projection View center focus, the point the of the Panoramic Picture that the camera focus at, is being calculated by the angle between two consecutive points in the path, if the angle is wider than what we defined as a straight line then we turn the camera in order to point to the right point of the panoramic picture, we explain this in depth in the Application Analysis and Algorithms chapter.

A sequence of Projection Views (definition above) from consecutive points along the route projected on the computer screen at projection time intervals (variable determined by the user that can be changed during presentation) forms a Tour Visualization along the chosen path. The tour can capture different parts of the surroundings at different parts of the route. This is done by changing the viewing focus (focus point and angle of viewing in the panoramic picture) for the pictures at different parts of the route, turning the camera to the desired object of interest.

At any point during the Tour Visualization the user can stop the visualization in order to browse the surroundings of the vicinity of the position of the route that the tour visualization currently is. This allows him essentially to browse in the X, Y dimensions of the Panoramic Picture of the position with different viewing focus. Zoom in an object in distance, and zoom out. By zooming in the object of interest, the viewing angles horizontally and vertically will also change along with the center of the picture window. When the user wishes he can continue the Tour Visualization.

A Tour Visualization simulates whatever the user wants to do while traveling at different parts along a route. He can look directly ahead, he can look on the right hand side or on the left, up or down. In addition he can focus to an object far away and he can keep looking at the object as he travels along. Finally he can stop at any point along the route in order to browse the surroundings.

In Tours that have been specified beforehand, those choices have been fixed before the visualization starts. A Guided Tour is a tour that presents information about specific parts of the surroundings that may be of interest to the user, in form of text description, voice narrations, etc. These parts of the surroundings may be different sections of a city, different roads passing through or intercepting during the following of the route, or different objects such as buildings, status, or smaller objects such as Venetian Thereos (family signs), architectural designs that reveal a specific era, (e.g., Venetian or Turkish architectural aspects), parts of the architecture that have different origins (e.g., remains of Ancient Greek, Byzantine, Venetian, etc., in the same wall), parts of ancient remains that have different functions that is interesting to explain, a rock composition or metallic signs on the side of a mountain etc. Architectural parts in the design of a building like windows, doors, decorations, writings, or thereos.

Objects of Interest in a tour have Object Visibility Identifications. They can be markers in the form of a simple icon presentation or of a parallelogram enclosing the object. In addition with each Object of Interest we associate a set of Route Visibility Sections for the object. These are parts of the route from which the object will be identifiable through different object visibility identifications.

Objects of Interest have an Object Unique Identifier, a set of Usual Names with which they can be searched, a Text Description, a Picture, an External Web Description (for example a link in Wikipedia or elsewhere), an optional Audio File for the voice description of the object and an optional Video File. The Route Visibility Sections that are associated with the object also have a Section Unique Identifier, a short Text Description, optional Image and Video Files presenting the Object from a particular point of view and a Voice Narration. The voice narrations correspond to what a tour guide would say about the object of interest during a tour. There may be two types of voice narrations, short and long that may correspond to two different types of tours (short and long). In addition certain narrations may be turned off based on their importance and the duration of the tour. Different narrations may correspond to different route visibility sections. For example an object of interest visible from far may have a very short narration saying that it can be seen in far, while it can have an extensive narration when is seen from near. Note that an object of interest may be visible from far and there may be a route visibility section for it, but the tour route may not pass near it. Narration can still describe the object even though the route does not pass besides it. Objects belong to Classes, and Classes are organized to Class hierarchies. Those classes are defined by Category Types and a set of Keywords. An Object of Interest may belong to more than one Classes. For example it may be a door (architecturally) and it may also be of Venetian origin (historically). Different Class hierarchies may be used, depending on the location, nationality, etc. User Interests may be associated with certain classes of objects only. Search capabilities may also exploit the classification of the objects of interest.

In addition to Objects of Interest there are also Areas of Interest and Roads of Interest which may be of importance to the Guided Tour. Their visual identification and descriptions are consistent with these of the Objects of Interest. Their visual identification and description are in the form of polygons. Typically their description through narrations will happen at the entrance to the area or road (or the intersection with the road). In some cases areas of interest may be visible from far in which case a visual identification and narration can be given for them. Additional information for areas or roads or larger object formations (such as mountains) can be given by identifying them through boundaries on a google map (interactively accessed by the user).

The presentation model described is based on Panoramic Pictures compatible to Street maps of Google. We will call them StreetView Panoramics. Thus the existing Panoramic Pictures of the Street maps can be reused. The Guided Tours will have to implement the additional functionality needed like the Routes, the Projection Views at the different parts of the routes, the objects of interest and their visual identifications, etc. on top of these panoramic pictures.

However, Street maps do not contain all the information needed. For example they may not have certain paths that are not “main roads” but they may be important for visiting a larger archeological site such as a fort. Another problem with StreetView is that the Panoramic Pictures are not always of good quality for the purpose of the Guided Tour. For example if the tour wants to show an Object of Interest like a statue but at the time that the Panoramic Pictures were taken there were a lot of cars in front of the statue then the video presentation is not really of high quality for the purposes of the Guided Tour. A third reason is that for some important larger monuments such as a church the tour may have to pass around the church in order to show all its architectural details. Street maps do not usually make such detailed coverage of monuments.

In all those cases there is a need to construct an additional set of Panoramic Pictures that will complement or replace a set of Panoramic Pictures of the street maps but provide the same functionality so that all the additional guided tour functionality is built on top of them as well and have a consistent interface to the users. We call these panoramic pictures UserCaptured Panoramic Pictures. The UserCaptured Panoramic Pictures are associated with a part of the route of a Guided Tour and they may or may not replace possibly existing Street map Panoramics.

There is another important reason for the UserCaptured Panoramic Pictures. Google Maps API allows to place markers on the ground but neither Google Maps API nor Street Map Image API allow the placement of markers or more general object visibility identifications such as rectangles on particular objects above the ground. In those APIs Markers will appear with their tails anchored to the location's horizontal plane within the Street View panorama. These visibility identifications however are important for guided tours. For example, for smaller objects such as architectural details on buildings, or objects in distance it is important to define more precisely the particular objects of focus, not just their position in x, y coordinates. This functionality can be implemented and supported on top of UserCaptured Panoramic Pictures.

A special case of the use of the object of interest functionality is to support advertisements for shops of various kinds along a tour. The **Advertisements** can be turned on and off. Advertisements can be placed within the application’s pages in form of divs or popups. The can also be placed as images on top of UserCaptured Panoramic Pictures.

In addition to the Guided Tour functionalities described above which are constructed beforehand, the user can use the system in a more flexible way of Interactive Viewing. He can do that using interactive control buttons in the interface that allow him to navigate in variable speed through a route and interactively determine the viewing angle and focus at the different parts of the route. At any point in time he can change the viewing angle and focus or stop the navigation in order to look at something more carefully or find more information about an object. He can then continue the navigation.

In the interactive mode the user is notified from the system for the existence of Objects of Interest to him that are nearby. **Notifications** are in form of narrations that guide him to turn his attention to the way the Object of Interest is located. These objects may or may not be along the path that he has selected but they are visible from it. In the case that he decides to see information about these objects a path to the object location is shown on the map and also the object visual identification as he approaches the object is visible.

This information is supported both in the workstation and the mobile. In the case of the mobile user, the user may prefer not to look at the mobile at all times. The user may select a route to follow. The mobile will notify the user for important changes in direction that he has to follow and he will always be able to see its current position with respect to the chosen route on top of the Google maps. In addition, the mobile notifies the user when he approaches a an Object of Interest and if the user wants he can see in the mobile a view of what is in front of him and where the object of interest is located. If he wants he can access more information about the object.

When the user walks through a Route he has the capability to identify objects of interest, take his personal pictures or video and link his own pictures with the existing information in the system about the specific object of interest so that he can review this information later (or copy some information). The system provides the capability to add a comment on the existing Object of Interest and accompany it with a picture and a text description.

Note that the same functionality for mobile capturing of objects of interest can be used by professionals. In this case off line creation of better visual identification and quality presentations and narrations will complete the description of the object and its association with route segments from which is visible.

The above description roughly outlines also the management capabilities needed for such as system (insertions, deletions, updates). They will be elaborated further on the Application Analysis and Algorithms chapter.

Chapter 4

4 Database Design

4.1 Introduction

In this chapter we are going to present all the information related to the database we used for our system. We are going to present the Entity Relationship and UML Class diagram of the database, give a short description for the entities and relationship tables and for the functions that handle the communication between the database and the server.

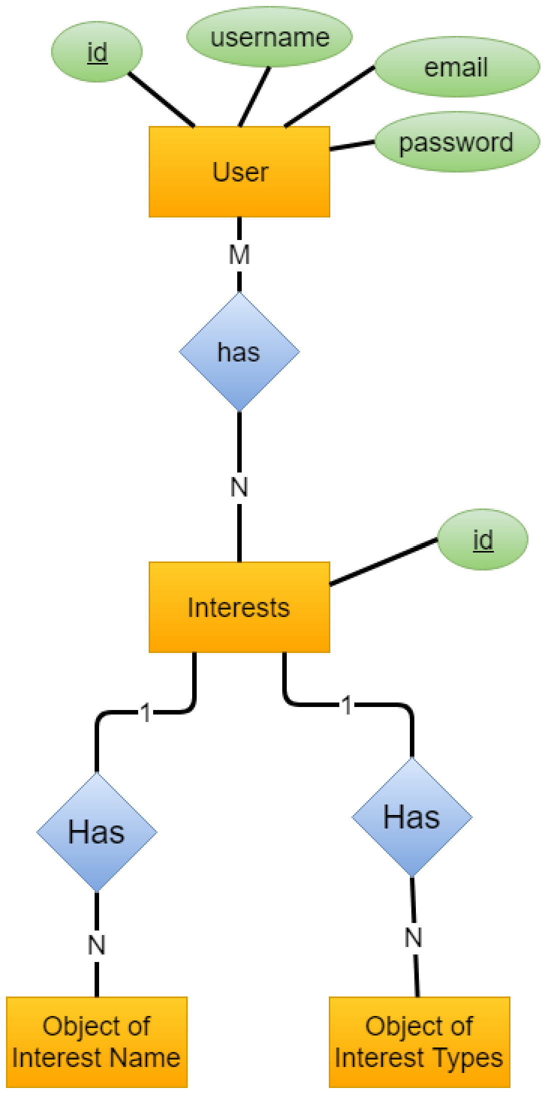


Figure 4.1 ER Diagram for User Object.

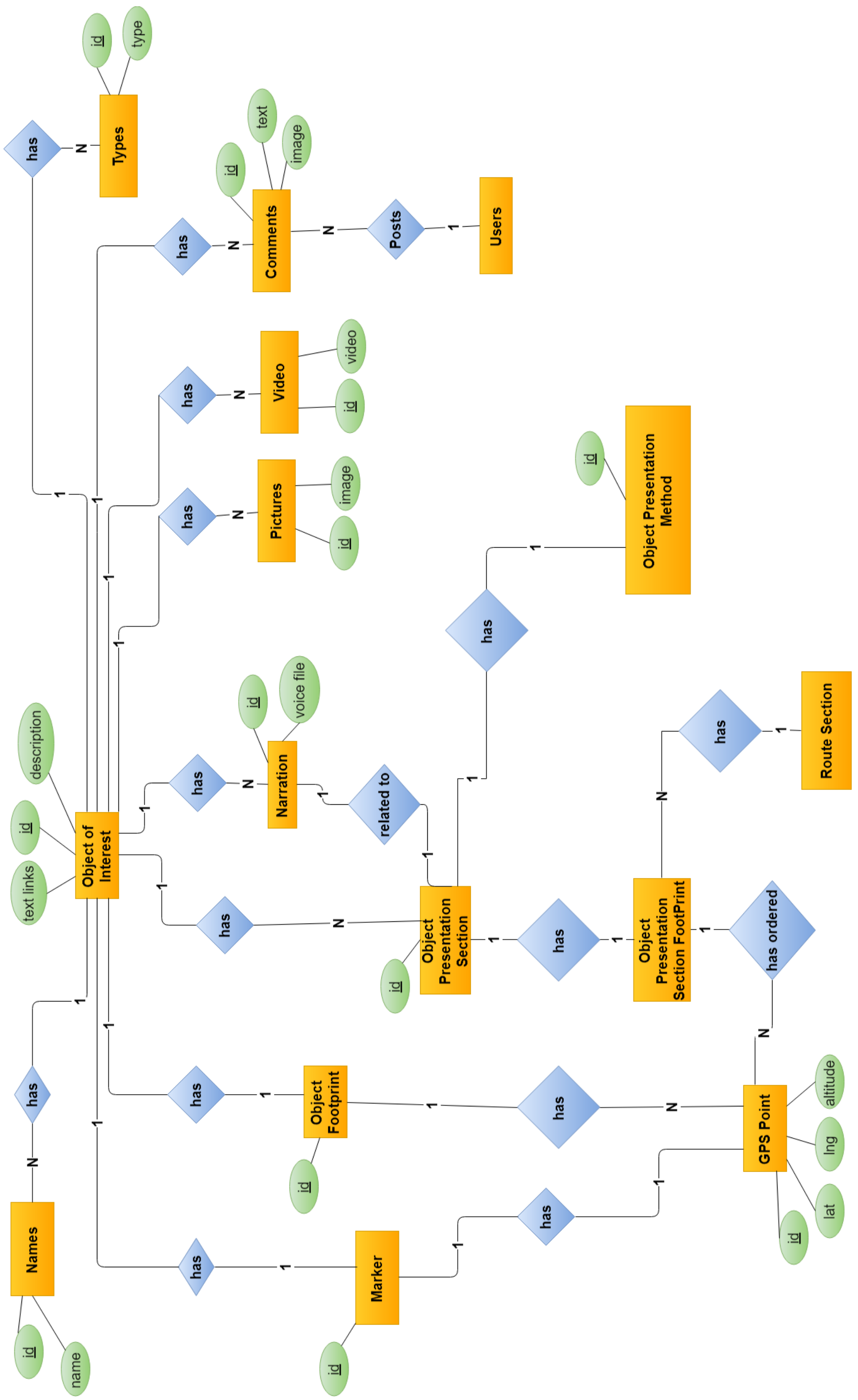


Figure 4.2 Entity - Relationship Diagram for Object of Interest Object.

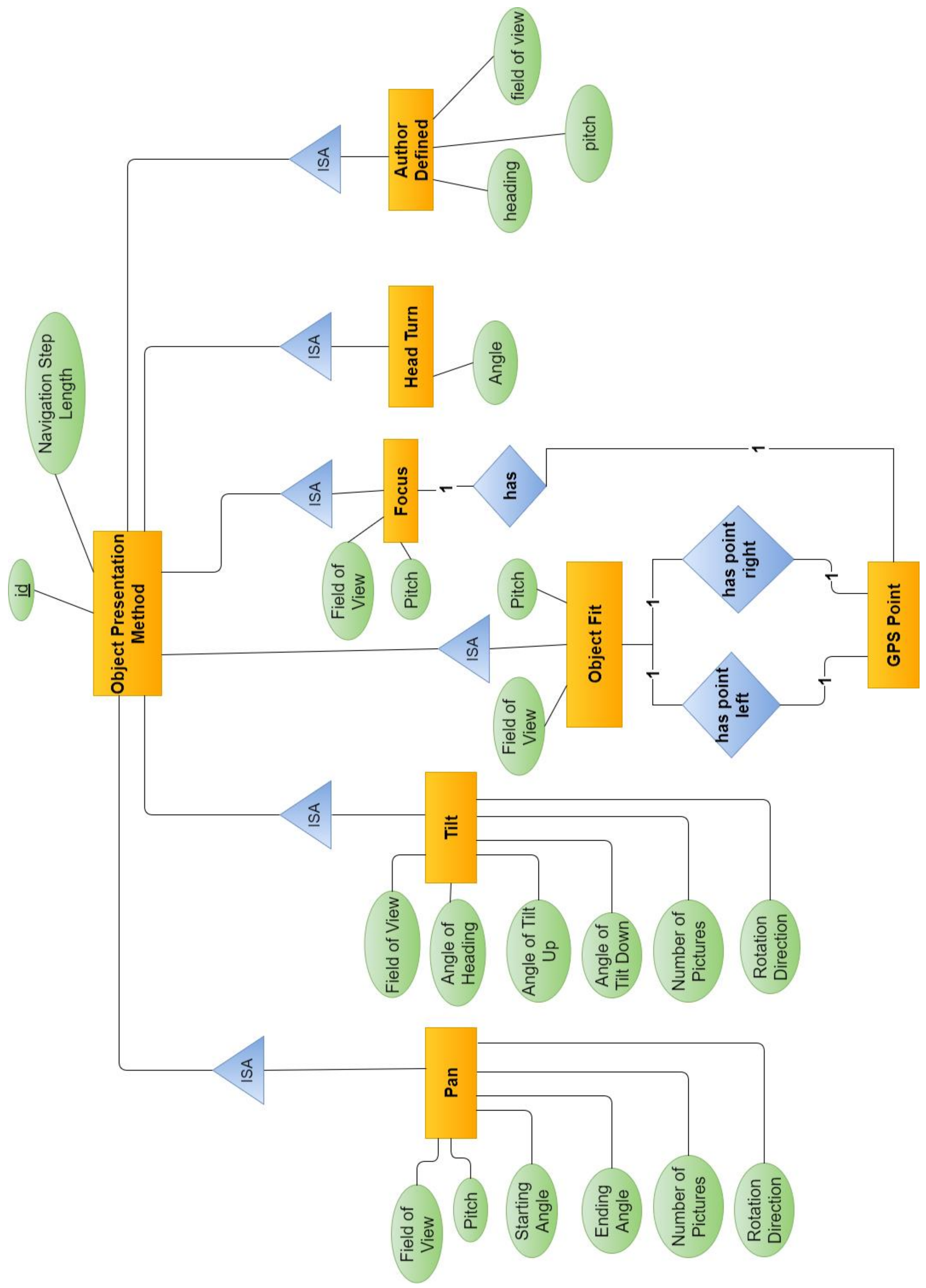


Figure 4.3 Entity - Relationship Diagram for Object Presentation Method Object.

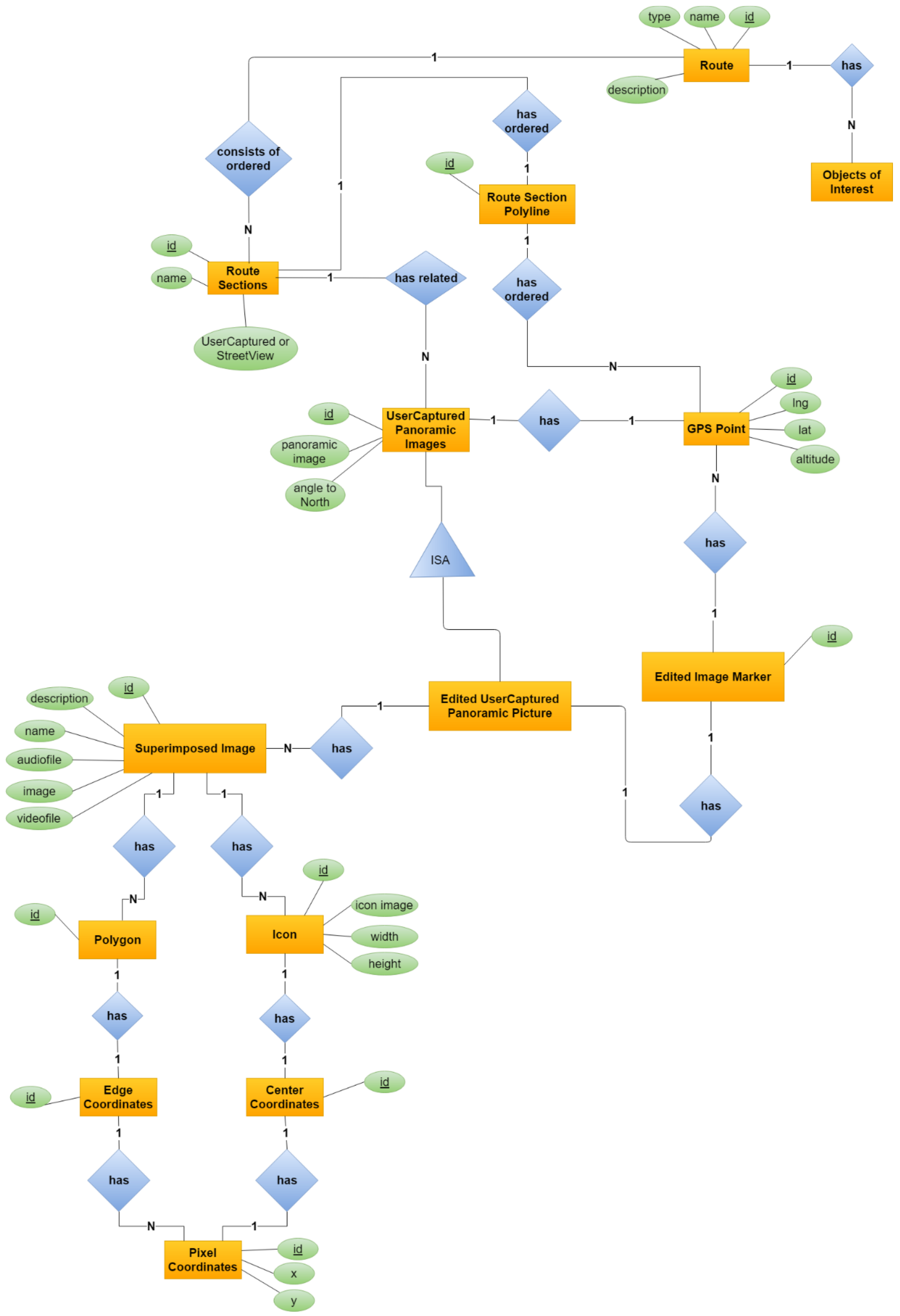


Figure 4.4 ER diagram for Route Object.

4.2 Entities and Relationships of the Database

We are going to give a short description about the some of the more complicated entities, as they are presented on the ER diagram on the figures 4.1, 4.2, 4.3 and 4.4.

1. Table ***Routes***: contains all the information for the retrieved Route, such the uploaded image file, a text description, the duration and the type of the Route.

2. Table ***Route Section***: contains all the information for a Route Section. A name, an identifier and a type, depending on the type of panoramic pictures, StreetView or UserCaptured Panoramic Pictures.

3. Table ***ObjectOfInterest***: contains the information associated with an ObjectOfInterest. The ObjectOfInterest's Name, description, type, any media files the creator has added, such as an image file, an audio file or a video file, the category of the Point, the type (Single or Polygon), an address and a telephone number.

4. Table **Pictures&Video**: contains the information about the multimedia associated with an Object of Interest.

5. Table ***Marker****:* entity for the case of a single point Object of Interest. It has one GPS Point.

6. Table **ObjectFootprint**: entity for the case of a polyline or polygon Object of Interest that represents a large object, a road or an area. It has two or more GPS Points.

7. Table **UserCapturedRouteSection**: contains the information about the Polyline that represent a UserCaptured Panoramic Route Section, on top of its path the sequence of the UserCaptured Panoramic Pictures are loaded. It has an integer identifier, a camera type, which defines the handling of the camera across the path and a navigation step value which defines the speed across the section.

8. Table ***UserCapturedPanoramicPicture***: in this entity we store all the information relatable to a User Captured Panoramic Picture. The name of the picture, the path to the uploaded panoramic picture file, the angle of the Panoramic Picture regarding the True North and the Pitch angle (always zero, since pictures always point to the horizon).

9. Table ***ObjectPresentationSections***: contains the information about the ObjectPresentationSection Object(s) associated with an Object of Interest.

10. Table **Narration**: contains the audio file of an Object Presentation Section with information about the associated Object of Interest.

11. Table **ObjectPresentationMethod**: contains the information about the presentation method that defines the algorithms and parameters applied to the projection views along an Object Presentation Section. It has subclasses like the Pan and Tilt actions, a manual definition of the camera parameters or an algorithm for the footprint of the associated Object of Interest.

12. Table **EditUserCapturedPanoramicPicture**: in this entity we store all the information relatable to an ObjectEnclosingMarker. The name of the ObjectEnclosingMarker, the uploaded panoramic image and the point of view variables that define the heading angle when the image is loaded, an audio file for the narration upon entering within the radius of the ObjectEnclosingMarker.

13. Table **ObjectEnclosingRectalngle**: contains the information associated with the CustomMarker Objects. Here we store the name of the Marker, any media file the creator may have added and a text description.

14. Table ***Comments***: we store the message body of the comment, the date the comment was submitted, the id of the user that posted it and an optional image file. Each Object of Interest can have zero or many comments submitted about it.

15. Table ***Users***: we store the information about the registered users of the application, such as the first and last name, a username, an email and a password.

16. Table **Interests**: the table contains the information of the user's interests, that can depend either on the type of the Objects or the Objects themselves.
17. Table **Names**: the table that contains the information related with the associated keywords of an Object of Interest. An Object of Interest can be associated with many keywords (names).
18. Table **Types**: the table that contains the information about the type of the Object of Interest.

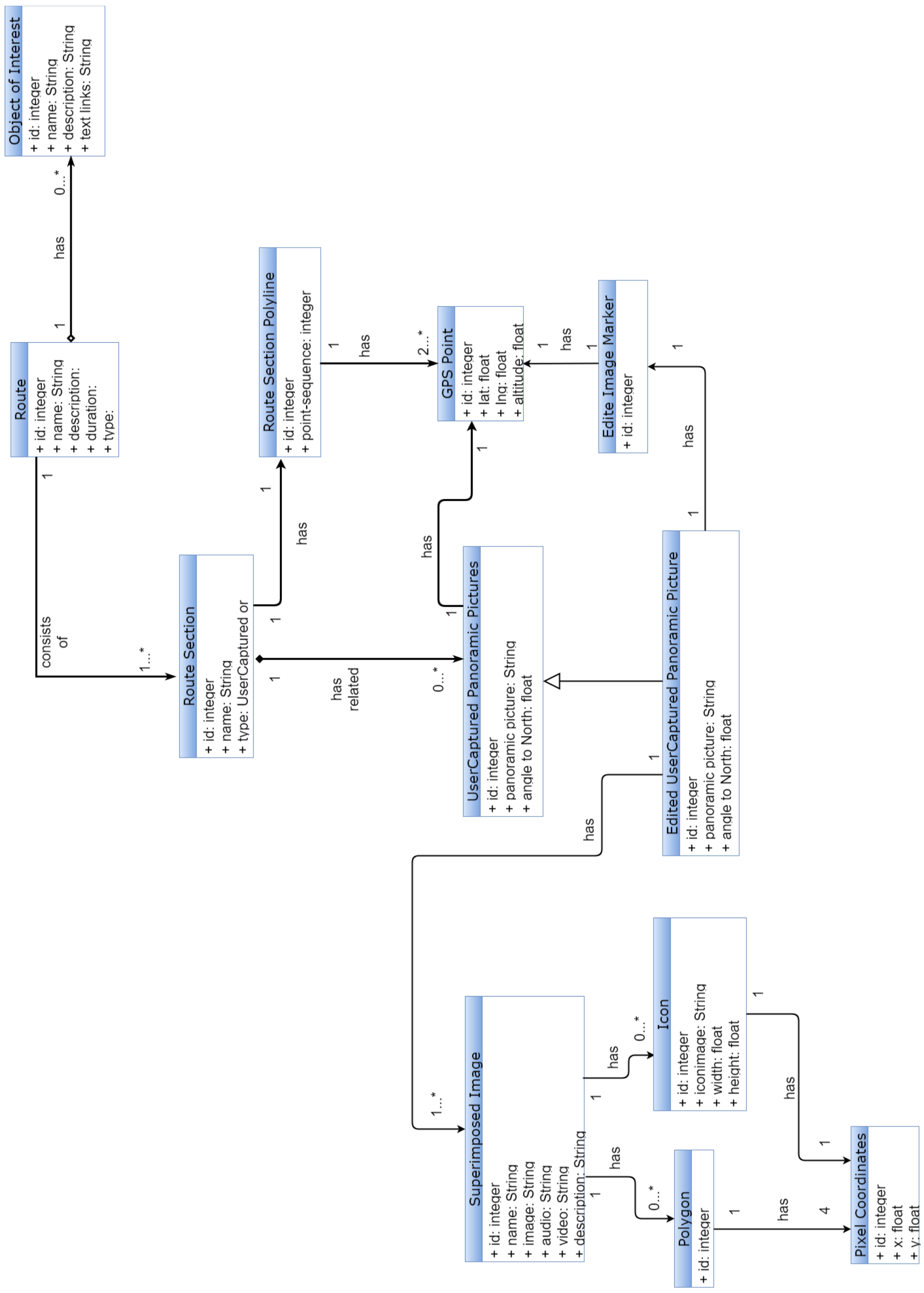


Figure 4.5 UML Class Diagram for Route Class.

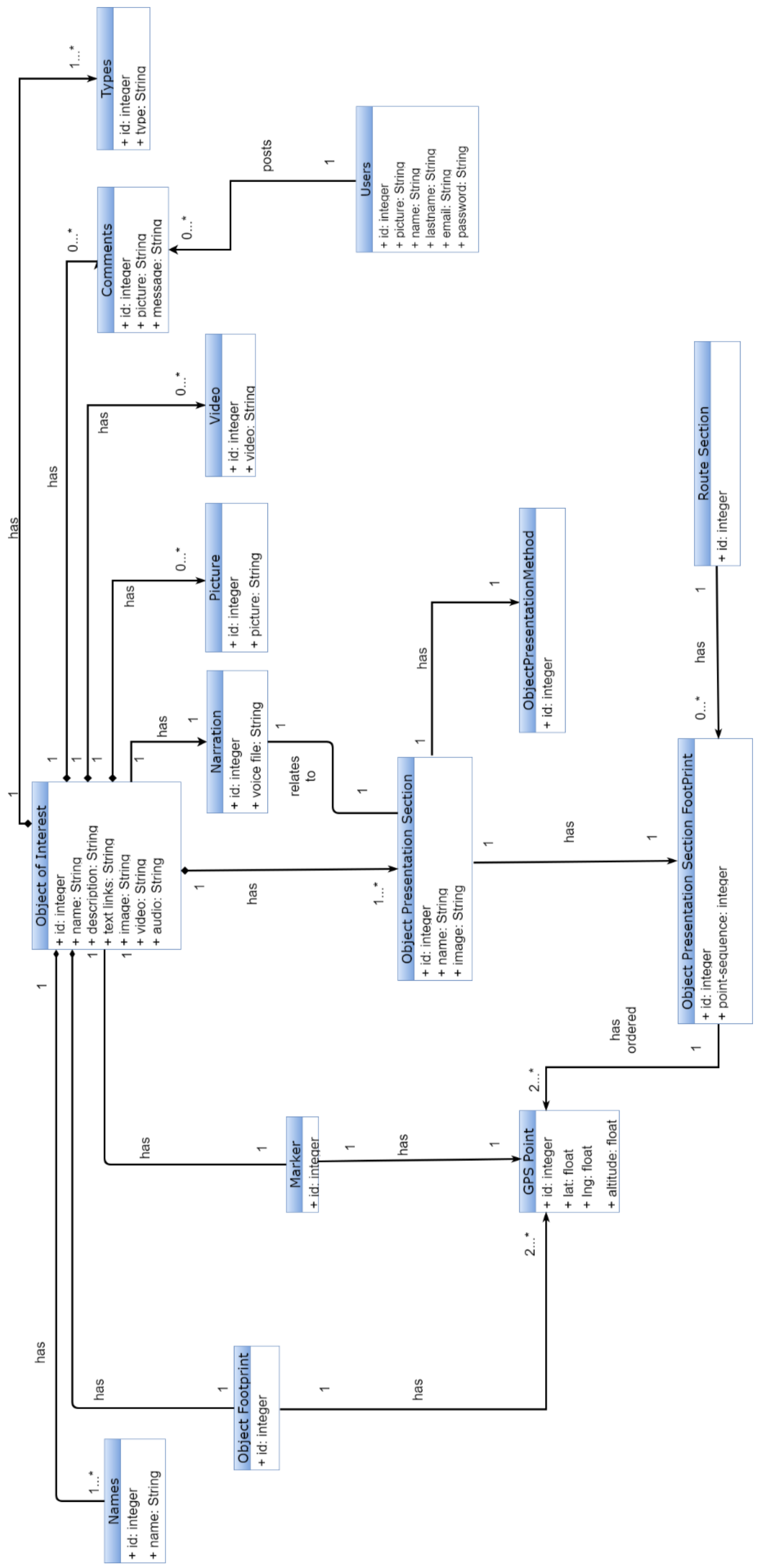


Figure 4.6 UML Class Diagram for the Object of Interest Class.

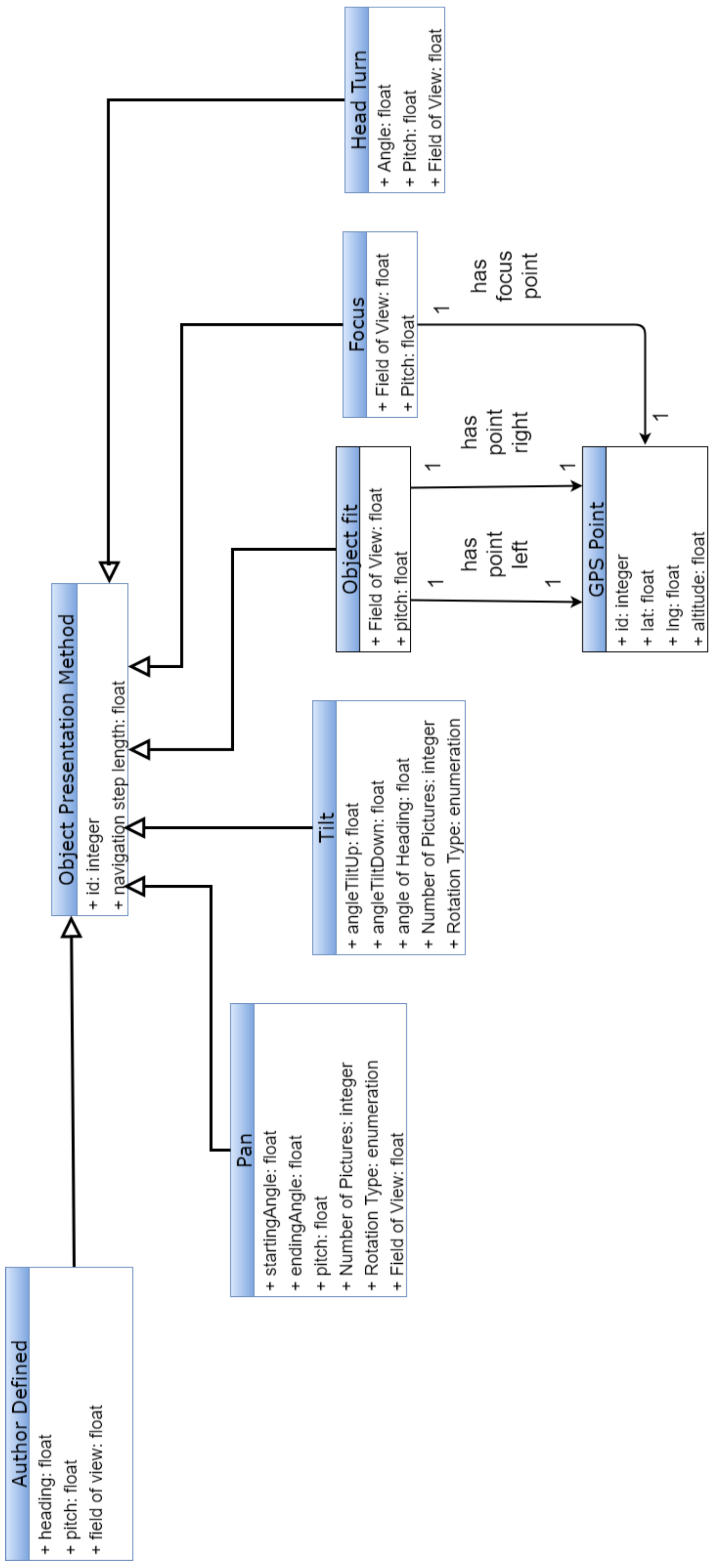


Figure 4.7 UML Class Diagram for the Object Presentation Method, of an Object Presentation Section.

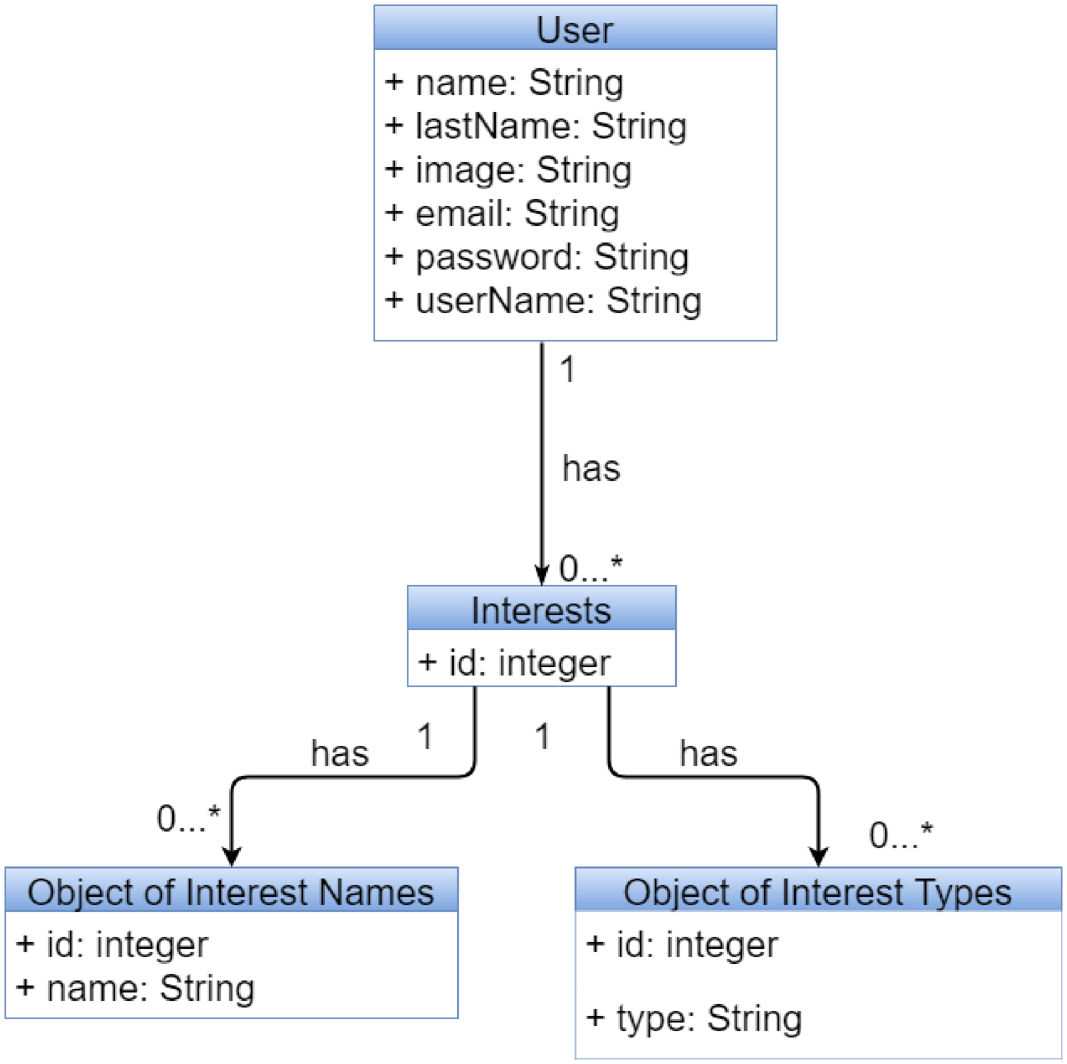


Figure 4.8 UML Class Diagram for the User Class.

4.3 Functions of the Database

Here we are going to analyze some of the basic functions constructed within Java Classes to allow us the interaction with the database Tables. Through those functions we store and retrieve the objects associated with each Route.

First we created Java Classes that handle the MultipartForm data we receive upon every form submission. We constructed classes for all the Objects in our database. In those classes we define the form parameters related to the forms of our interface, and construct set and get methods for each one of them. We include an example of one of those classes below:

```
public class ObjectOfInterestObject {

        @FormParam("ObjectOfInterestName")

        private String ObjectOfInterestName;
        @FormParam("ObjectOfInterestImageFile")

        private byte[]ObjectOfInterestImageFile;

.   .   .

        public String getObjectOfInterestName() {
return ObjectOfInterestName;}

        public void setObjectOfInterestName(String ObjectOfInterestName) {
this. ObjectOfInterestName = ObjectOfInterestName;}}
```

1. Function *public static String getAllRoutes( )*: returns all the created routes within our databases, in order to display them to our Planned Routes jsp page. The format response is a Json array containing all the routes which we constructed by using two functions of PostgreSQL the *array_agg* function that aggregates all the columns of each tuples in a string and the *array_to_json* that transforms the array to a Json array.

2. Function *public static int getLatestRouteID()*: returns the last inserted route's id.

3. Function *public static String searchRoutes(String tagname)*: returns all the created routes associated with the tag or tags the user has entered in the search functionality. The search tag or tags are contained within a string and separated with each other with a comma.

4. Function *public static String getNearRoutes()*: returns routes information alongside the Lat and Lng coordinates of the start Point of each route. We use those retrieved coordinates to find the closest routes to our position when using the application on a mobile device.

5. Function *public static String getRouteWithID(int id)*: returns the Route Object that corresponds to the given id.

6. Function *public static String getRouteMarkers(int id) [getRoutePointsofInterest (int id), getRouteSegments(int id), getRoutePanoramas(int id)]*: returns all the Marker(Segment/ Object of Interest/ Panorama/ Custom Marker) objects associated with the loaded Route. We parse the returned Json response and set the objects on the Map.

7. Function *public static String updateSegmentInfo(String segname, String segdescription, String segimage, String segaudio, String segvideo, int segid)*: returns the updated information of the Segment, which the columns we updated by passing its id in the function.

8. Function *public static String updateSegmentPath (String arrayPath)*: receives a string as input that contains an array of LatLng coordinates that represent the polyline's path, separated with a comma. This functions runs a for-loop for the length of that array storing each point to the SegmentsPointsPair table.

9. Function *public static String updatePolygonPath(String arrayPath)*: similar to the *updateSegmentPath* function, the string input parameter defines the closed polyline path that constructs the Polygon Object. This functions runs a for-loop for the length of that array storing each point to the PointOfInterestPointsPair table.

10. Function *public static String updatePanoramaPosition(String LatLngString, int id)*: returns the new position of the Panorama Object on the new LatLng coordinates. This functions makes a Update query on the PostgreSQL database on the Panoramas tables tuple with the given id.

11. Function *public static String getAllComments(int id)*: returns all the comments that users may have submitted for the Object of Interest with id matching the input.

12. Function *public static String getEditRoutePointsOfInterestsSegments(int routeid)*: returns the information of the Segment Object we are going to update so we can display them to Edit Segment Pop Up interface. Aside of loading the name, description, image file etc. of the Segments Table we also have to get an array of all the Custom Marker ids placed on the specific segment.

13. Function *public static String getRouteSegmentsCustomPanoramaMarkers(int segmented)*: returns all the *CustomPanorama*Marker Objects associated with the Segment Object with the given id.

14. Function *public void deletePointOfInterest(int id)*: handles the deletion of the object of interest with the given id. Before deleting the tuple of the PointsOfInterest table, we have to delete first the tuples of the tables that are associated with it, such as Segments, Custom Markers, and PointsofInterestPointsPair.

4.4 Relational Schema

We have designed the Relational Schema[26] of our system based on the UML Class diagrams.

- Route (**<u>RouteID</u>**, Name, Description, Type)

[26] Primary Keys are underlined and bold. Foreign Keys are mentioned by (FK) and bold.

- RouteSection (**<u>RouteSectionID</u>**, name, StreetView/UserCaptured, **SectionPolylineID** (FK), **SectionFootPrintID** (FK), **RouteID** (FK))
    - **SectionPolylineID**: Foreign Key refers to **RouteSectionPolylineID** of RouteSectionPolyline.
    - **SectionFootPrintID**: Foreign Key refers to **ObjectPresentationSectionFootPrintID** of ObjectPresentationSectionFootPrint.
    - **RouteID**: Foreign Key refers to RouteID of Routes.
- RouteSectionPolyline (**<u>RouteSectionPolylineID , PointOrder</u>**, **GPS_X** , **GPS_Y, GPS_Z**)
- UserCapturedImage (**<u>UserCapturedImageID</u>** , **RouteSectID**(FK), panoramicImage, angleToNorth, **GPS_X**, **GPS_Y** , **GPS_Z)**
    - **RouteSectID**: Foreign Key refers to **RouteSectionID** of RouteSection.
- EditedUserCapturedImage (**<u>UserCapturedImageID</u>**, image/ polygon, **ImposedImageID** (FK), **PolygonID** (FK))
    - **ImposedImageID:** Foreign Key refers to **SuperImposedImageID** of SuperImposedImage.
    - **PolygonID:** Foreign Key refers to **SuperImposedPolygonID** of SuperImposedPolygon.
- SuperImposedImage (**<u>SuperImposedImageID</u>**, name, description, audiofile, videofile, imagefile, iconWidth, iconHeight, iconimage, centerX, centerY)
- SuperImposedPolygon (**<u>SuperImposedPolygonID</u>**, name, description, audiofile, videofile, imagefile, x1, y1, x2, y2, x3, y3, x4, y4)
- EditedImageMarker (**<u>EditedImageMarkerID</u>**, **UserCapturedID** (**FK**), GPS_X, GPS_Y, GPS_Z)
    - **UserCapturedID:** Foreign Key refers to **UserCapturedImageID** of UserCapturedImage.
- ObjectOfInterest (**<u>ObjectOfInterestID</u>**, textlinks, description, **FootPrintID** (FK), MarkerID, MarkerGPS_X, MarkerGPS_Y, MarkerGPS_Z)
    - **PresentationSectionID:** Foreign Key refers to **ObjectPresentationSectionID** of ObjectPresentationSection

- **FootPrintID:** Foreign Key refers to **ObjectFootPrintID** of ObjectFootPrint

- ObjectNames (**<u>ObjectNamesID</u>**, **ObjectOfInterestID** (FK)**)**
  - **ObjectID**: Foreign Key refers to **ObjectOfInterestID** of Object
- ObjectTypes (**<u>ObjectTypesID</u>** , **ObjectOfInterestID** (FK))
  - **ObjectID**: Foreign Key refers to **ObjectOfInterestID** of Object
- ObjectFootprint (**<u>ObjectFootprintID , pointSequence</u>**, GPS_X, GPS_Y, GPS_Z)
- Narration (**<u>NarrationID</u>**, **ObjectOfInterestID** (FK), voicefile)
- ObjectPresentationSection (**<u>ObjectPresentationSectionID</u>**, **ObjectOfInterestID** (FK), navigationStepLength, **NarrationID** (FK), **ObjectPresentationMethodID** (FK), **RouteSectionID** (FK), **SectionFootPrintID** (FK))
  - **SectionFootPrintID**: Foreign Key refers to **ObjectPresentationSectionFootPrintID** of ObjectPresentationSectionFootPrint.
  - **NarrationID**: Foreign Key refers to **NarrationID** of Narration.
- ObjectPresentationSectionFootPrint (***<u>ObjectPresentationSectionFootPrintID, pointSequence</u>***, GPS_X, GPS_Y, GPS_Z)
- Images (**<u>ImagesID</u>** , **ObjectID** (FK), imagefile)
  - **ObjectID**: Foreign Key refers to **ObjectOfInterestID** of Object.
- Videos (**<u>VideosID</u>** , **ObjectID** (FK), videofile)
  - **ObjectID**: Foreign Key refers to **ObjectOfInterestID** of Object.
- Pan (**<u>ObjectPresentationMethodID</u>** , pitch, fieldOfView, startingAngle, endingAngle, numberOfPictures, rotationDirection)
- Tilt (**<u>ObjectPresentationMethodID</u>** , headingAngle, fieldOfView, angleOfTiltUp, angleOfTiltDown, numberOfPictures, rotationDirection)
- ObjectFit (**<u>ObjectPresentationMethodID</u>** , fieldOfView, pitch, leftGPS_X, leftGPS_Y, rightGPS_X, rightGPS_Y)
- Focus (**<u>ObjectPresentationMethodID</u>** , fieldOfView, pitch, FocusGPS_X, FocusGPS_Y)
- HeadTurn (**<u>ObjectPresentationMethodID</u>** , angleOfTurn, Pitch)
- AuthorDefined (**<u>ObjectPresentationMethodID</u>** , heading, pitch, fieldOfView)
- User (**<u>UserID</u>**, username, email, password)

- UserInterests (**InterestID, UserID** (FK), **NameID** (FK), **TypeID** (FK))
    - **UserID**: Foreign Key refers to **UserID** of User.
    - **NameID**: Foreign Key refers to **ObjectNameID** of ObjectNames.
    - **TypeID**: Foreign Key refers to **ObjectTypeID** of ObjectTypes.
- UserComments (**UserID** (FK), **ObjectID** (FK), commentMessage, imagefile)
    - **UserID**: Foreign Key refers to **UserID** of User.
    - **ObjectID**: Foreign Key refers to **ObjectOfInterestID** of ObjectOfInterest.

Chapter 5

5 Application Functionality and Algorithms

5.1 Introduction

In this chapter we are going to analyze the functionalities, described in the Requirement Analysis, in depth. We are going to present the algorithms and methods used to construct each of these functionalities. We also going to give a short description of the Objects and methods we have included from the Google Maps JavaScript library. In the subchapters 5.2 to 5.10 we are going to analyze the algorithms and functions regarding the Authoring Environment of the System, and in the subchapters 5.11 to 5.17 we are going to analyze the algorithms regarding Navigation Playback Environment.

5.2 Construction of the Map Object

This service handles the creation of the Map Object, on top of which the Route Object, the Objects of Interest and the Custom Panorama Objects are set. It takes as input the type of the Map, the LatLng position that represents the Center and a Zoom Value and returns the Map Object.

The main purpose of this thesis is to design routes within either a city, a region, a village or generally the countryside (through a mountain, a gorge etc.) which will allow the users of the application to interact with the Points of Interest that are placed along their paths. These routes are constructed using the Google Maps API Library.

Before proceeding with the design of the Route Object we need to set a Map Object on top of which all the other application’s Objects are set by adding the appropriate event listeners on it. When constructing a map object we must select a **MapType** object to hold information about these maps. A **MapType** is an interface that defines the display and usage of map tiles and the translation of coordinate systems from screen coordinates to world coordinates. There are four types of maps available within the Google Maps JavaScript API:

- *Roadmap*, displays the default road map view. This is the default map type.
- *Satellite*, displays Google Earth satellite images
- *Hybrid*, displays a mixture of normal and satellite views
- *Terrain*, displays a physical map based on terrain information.

Besides setting the MapTypeId, we must also define the center of the map and the zoom value. We calculate the LatLng object that we set as the center on the Route construction section. The LatLng Object corresponds to the coordinates system we use to locate places on earth, the terrestrial system. The coordinates in this system are Latitude and Longitude. Latitude is a geographic coordinate that specifies the north–south position of a point on the Earth's surface, it ranges from 0° at the Equator to 90° at the poles. Longitude, is a geographic coordinate that specifies the east-west position of a point on the Earth's surface. We load the google.maps.Map Object to an Html Div element we have created on the page.

```
// Construction of the Map Object
var myLatlng = new google.maps.LatLng(-34.397, 150.644);
var mapOptions = {
  zoom: 8,
  center: myLatlng,
  mapTypeId: 'roadmap'
};
var map = new google.maps.Map(document.getElementById('map'),
    mapOptions);
```

5.3 Construction of the Route Object

This service is responsible for the creation of the Route Object. It takes as input an array of LatLng Objects the user sets on the Map Object by click events, a Name, a Type (Historical, Sightseeing, and Outdoors), an Image file and a Text Description and returns a Polyline Object set on the Google Maps Object, which represents the Route.

The Google Maps JavaScript API offers the Directions Service that calculates directions between two given points. This service communicates with the Google Maps Directions API which receives directions requests and returns an efficient path. Travel time is the primary factor which is optimized, but other factors such as distance, number of turns and many more may be taken into account. Due to these factors though it is possible that the service will not return the desired path (restricted usage roads). One way to resolve this issue would be to add a sufficient number of waypoints, on the roads we want to include, between the starting and ending points which will force the returned path to pass through these desired roads (streets). There are some usage limits of the Direction Service that would prevent that work around to be efficient in cases where the routes passes through multiple roads with restricted access or restricted usage roads both on the Standard and the Premium Plan. These limits are:

-Standard Plan

- 2,500 free requests per day, calculated as the sum of client-side and server-side queries.
- 50 requests per second*, calculated as the sum of client-side and server-side queries.

- Both for Standard and Premium Plan:

- Up to 23 waypoints per request, plus the origin and destination.

We want to offer an interface that will allow users to design routes that will include any available streets or roads, even if they are restricted or do not have Street View coverage. Therefore we decided to make use of the **Google Maps Polyline** Object. By using the Polyline Object we can

represent each route with a polyline on the map. The Polyline Class on Google Maps API defines a linear overlay of connected line sections on the map. A Polyline Object consists of an array of LatLng locations, and creates a series of line segments that connect those locations in an ordered sequence.
The polyline is defined in JavaScript using the PolyineOptions, which are specifying the array of LatLng coordinates and the styles of the Polyline's visual behavior.

Algorithm 5.1: Route Construction algorithm. Basic steps.

```
Step 1: Set Click Event Listener on Map Object

Step 2: Retrieve the path of the Route Polyline.

Step 3: Add marker on the clicked LatLng position.

Step 4: Add marker on Markers Array. This Array represents the path of the Route during the
construction.

Step 5: Add LatLng point to the Polyline.

Step 6: Submit final path.
```

```
// Design polyline for route

var polyline;
var markersArray;
// Function that handles the onClick event of the Map Object
function addToPath(event){
    path = poly.getPath(); // Retrieving the path of the Polyline Object
    path.push(event.latLng); // Adding the LatLng Object to the Path
    var marker = new google.Maps.Marker;
    marker.setPosition(event.latLng);
    markersArray.push(marker);
}
```

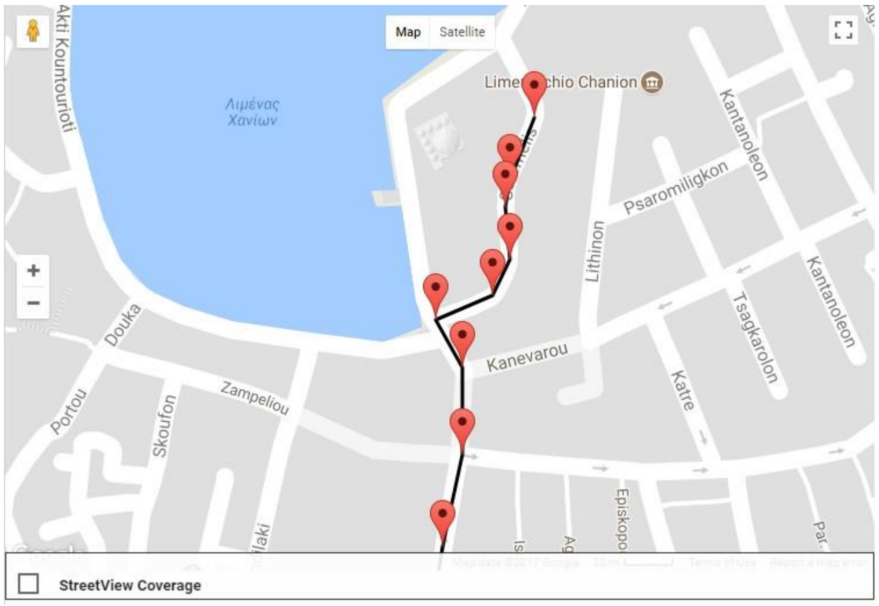


Figure 5.1 Construction of a Route Polyline. The clicked positions on the Map consist the path of the Route and are connected with a Polyline.

Aside of creating the path of the Route, the author must specify the Type of the Route, a Text Description for the Route, upload an Image File for the later presentation of the Route and finally define the duration of the Route. The Route Object will be next associated with the Objects of Interest that are placed along the route and will be used for the storytelling. To make this association the author can specify the footprints on top of the map of the important for the route objects of interest, so that the application derives the visualization (videos) taking into account the location of those objects, turning the camera appropriately, and defining the appropriate camera parameters for the visualization of the object. The footprint of an object of interest is in general a polygon, and in the special case it can be a line or a point.

5.4 Construction of Objects of Interest

This service creates the Objects of Interest that are associated with the Route Object. It takes as input an array of LatLng coordinates that represent the shape of the Object of Interest on the Map Object and returns the corresponding Single Object of Interest, if the array contains one LatLng coordinate or a Polygon Object of Interest if the array contains more than one LatLng coordinates.

The premade routes are designed in order to present Objects of Interest across an area to the users. In order to identify those Objects of interest on the map we will use markers to showcase Single Objects of Interest (Statues, Side of a Building, Fountain etc.) and Polygons to showcase larger Objects of Interests (Roads, Areas etc.) or to present various points of view of a building as the user walks by it. Both the Single and the Polygon Objects have click events that allow us to add listeners to bring up info windows displaying custom information.

Both types of Objects of Interest consist of a Name, a Unique Identifier, a text description, a video and an audio file, an image, a set of keywords (tags) and one or more Object Presentation Sections.

The differences concerns the visual identifications of the Object of Interest on top of the Map and the handling of the camera's direction towards it.

5.4.1 Single Object of Interest

The Single Objects of Interest are implemented with the use of the Google Maps Marker Object. A Single Object of Interest represents the LatLng position of the Object we want to associate with the Route. When constructing each marker the following fields are particularly important:

- Position (required), specifies a LatLng of the initial location of the marker.
- Map (optional), specifies the Map on which the marker is placed. If the map property is not specified the marker is created but is not displayed on the map. This can be done later by calling the method setMap ().

Similar to the Route design interface, we assign a click event listener to our Google Maps Object. We use this listener to get the Map LatLng coordinates that correspond to the user's click on the map. We assign a temporary Marker Object to that position and proceed to the Object Of Interest Interface.

Example of a Marker placement:

```
Function setMarker(event){
    var marker = new google.maps.Marker({
      position: event.LatLng,
      map: map,
      title: ‘Marker Name’,
      icon: 'images/markermoving.png'
    });
    //Marker Info
    google.maps.event.addListener(marker, 'click', function() {
      infowindow.setContent(contentString);
      infowindow.open(map,marker);
    });
   return marker;}
}
```

5.4.2 Polygon Object of Interest:

A Polygon Object of Interest can be the footprint of a building, the projection on the ground of a construction, or an area. The Array of LatLng coordinates that construct the Polygon, represent the shape of the building on the Map. A street or a road can also be represented by a Polygon Object of Interest by using a forming a straight line using the polygon Object.

A polygon represents an area enclosed by a closed path (or loop), which is defined by a series of coordinates. Polygon objects are similar to Polyline objects in that they consist of a series of coordinates in an ordered sequence. Polygons are drawn with a stroke and a fill. We can define custom colors, weights, and opacities for the edge of the polygon (the stroke) and custom colors and opacities for the enclosed area (the fill).

When constructing each polygon the following fields are particularly important:

• Path (required), an MVCArray. This array defines a separate sequence of ordered LatLng coordinates.

In order to give the user the ability to design the path that defines the Polygon, we added a click event on the Map Object which handles the creation of Marker Objects that correspond to the LatLng coordinates of every click. Those Markers are connected with each other with a Polyline. Upon clicking on the first created Marker the Polygon Object is closed and created. After the Polygon Object is created we present the Object of Interest Interface to the user so he can fill the required information.

- Map (optional), specifies the Map on which the marker is placed. If the map property is not specified the marker is created but is not displayed on the map. This can be done later by calling the method setMap ().
- Stroke and Fill Color, the colors of the Polygon Object, which can represent different Types.

Algorithm 5.2: *Algorithm for Object Footprint Construction. Basic steps.*

```
Step 1: Set Click Event Listener on Map Object

Step 2: Add marker on the clicked LatLng position.

Step 3: At each click, check to see if first Marker is clicked. If not, add the LatLng Object to a
polyline, representing the perimeter of the Polygon.

Step 4: If first Marker is clicked, pass the final path of the Polyline.
```

Step 5: If the path consists of three LatLng points, pass the polyline path to the Polyline constructor, else pass the path to the Polygon constructor to create the Footprint of the associated Object of Interest.

Pseudocode:

```
var polyline; // The Polyline Object we use to keep track of the user click events

var firstMarker // Variable we use to define the first Marker (start of the Polygon)

var markerIndex //  Variable we use to keep track of the Length of the Polygon

If (polygon is closed) then Return;

markerIndex = polygonPolyline.lenght;

isfirstMarker = markerIndex === 0; // Checking if the first Marker is clicked

polylinePath.push(clickEvent.LatLng) // Adding the clicked position on the Path

if (firstMarker) then // When the first placed Marker is clicked again, we close the polygon //
and create the Polygon Object that represents the Object of Interest

        polygon.setPath(polylinePath); // Assigning the created path to the Polygon Object

        polygon.setMap(map);
```

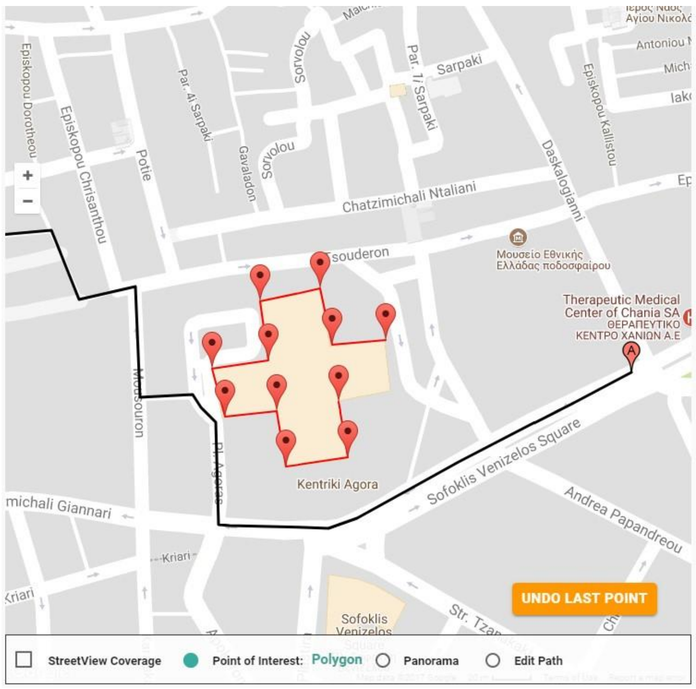


Figure 5.2 Process of Designing a Polygon Object of Interest. The click events on the Map are represented by Markers. Each LatLng object is added on an Array which will be passed to the Polygon constructor path attribute.

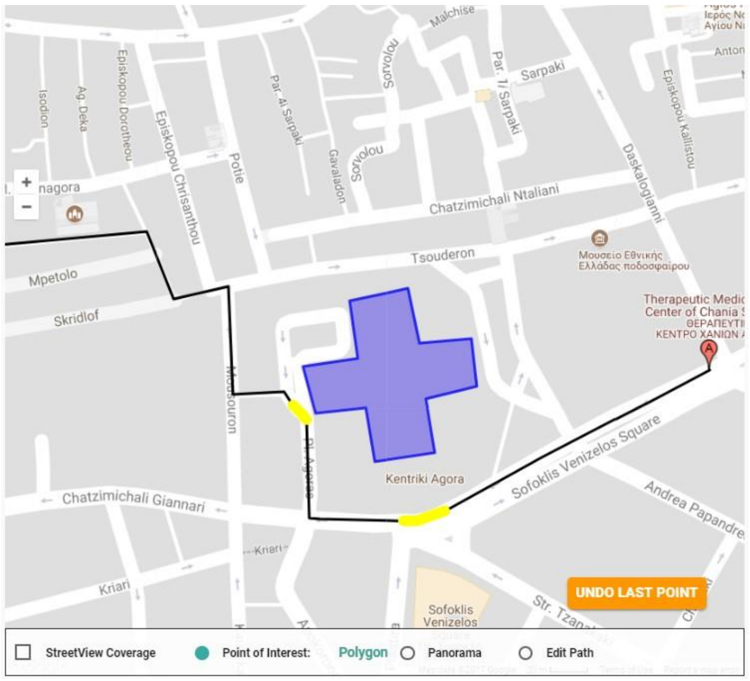


Figure 5.3 Upon clicking on the first placement Marker (first LatLng point), the Algorithm 5.2 passes the Array of the LatLng points to the Polygon construct, creating the corresponding Polygon shape or straight line.

5.5 Construction of Object Presentation Sections

This service is responsible for the creation of the Viewing Sections associated with the Objects of Interest. It takes as input an array of LatLng Objects, a Name, an Image file and an Audio file and a Video File. The Audio file is the narration being playing when the user approaches the relevant Object of Interest, notifying him about the Objects of Interest position and presenting information about it. The service returns a response in form of a Polyline that represents the Viewing Section on the Map.

As we mentioned earlier, every Object of Interest is associated with one or more Object Presentation Sections. These sections define the parts of the Route within the associated Object of Interest will be presented.

The user can define those sections in the Object of Interest interface. We allow this by adding a click event on the Route’s path (Polyline Object). Each click assign’s a Marker Object across the Route, which corresponds to a LatLng Object of the section. For each click we check if the LatLng Coordinates were within the designed Route’s path (thus making sure it is accessible). In order to check that we made use of the isLocationOnEdge () function of the google.maps.geometry.poly Library.

To determine whether a LatLng point of the Viewing Section falls on or near a Polyline, we pass the point, the polyline object and a tolerance value set in degrees to the google.maps.geometry.poly.isLocationOnEdge function. The function (point:LatLng, poly:Polygon|Polyline,tolerance?:number) returns true if the distance between the point and the closest point on the line falls within the specified tolerance.

Within the interface the author can have a preview of the point of view from that position. To calculate the direction of the camera within that position we calculate the angle between the Polyline Position and the Object of Interest position in case of a single type and the closest Polygon Point in case of a Polygon type. We will analyze the algorithms that calculate the closest point of a polygon and the angle that defines the camera later on. Finally the Marker Objects, that represent the clicked Positions on the map, are connected with a Polyline.

Algorithm 5.3: *Algorithm for Object Presentation Section Construction. Basic steps.*

Step 1: Set Click Event Listener on Route Polyline Object. Create a temporary Polyline Object which will keep the clicked LatLng points.

Step 2: At each click, check to see if LatLng is within the Route Polyline, using the isLocationOnEdge.

Step 3: If isLocationOnEdge returns true we add the LatLng point to a polyline path.

Step 4: Make a call to the calculatePolygonPointWithMinimumDistance, which returns the LatLng point of the Polygon perimeter closest to the new added section point.

Step 5: Present a preview of the Projection View of the clicked LatLng point of the section by making a StreetView request on the given point. We calculate the appropriate heading angle with the angleFromCoordinate function.

Step 6: After the author submits the constructed polyline, pass the path of the polyline to a new Polyline Construct which will create the final Object Presentation Section Polyline.

Pseudocode of Algorithm 5.3:

```
var polylineSegment; // The Polyline Object we use to keep track of the user click
events

var polylineRoute; // Polyline Object representing the Route, defined in Previous
steps

// Setting the Listener on the Polyline Object of the Route.

clickListener = google.maps.event.addListener(polylineRoute, 'click', function (event) {

        var pointForHeading = calculatePointWithMinimumDistance(event.latLng); // Calculating
the closest Polygon Point
```

```
        headingPov=angleFromCoordinate(event.latLng,polygonsTable[obj.pointsofinterestid].Get
PointAtDistance(pointForHeading));
        streetViewService.getPanorama({location);
        addLatLngToSegment(event);  //  Function  that  handles  the  creation  of  the  Segment
Polyline

});

function addLatLngToSegment(event){

   if (google.maps.geometry.poly.isLocationOnEdge(event.latLng, polylineRoute, 10e-6)) {

        var polylineSegmentPath = polylineSegment.getPath();

        polylineSegmentPath.push(event.latLng); // Add LatLng to Path

        var segmentMarker = new google.maps.Marker({position: event.latLng});

        segmentMarkersArray.push(segmentMarker); //Each click represented by a Marker Object

}
```

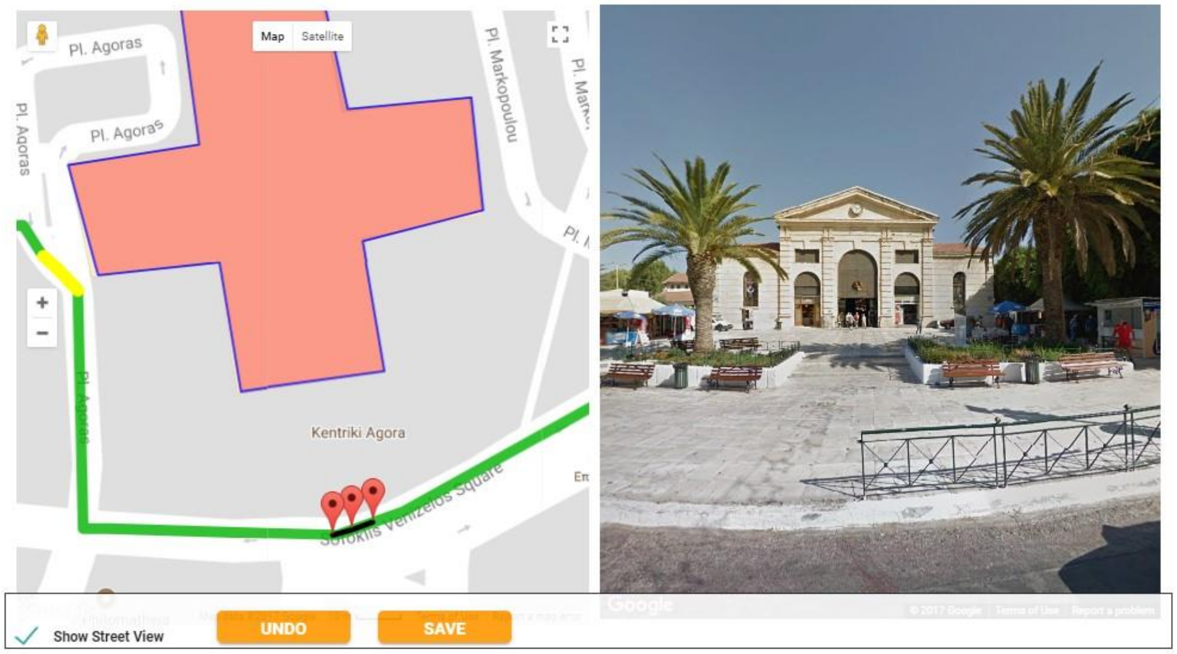


Figure 5.4 Process of designing an Object Presentation Section for an Object of Interest. The user adds the LatLng points across the Route's polyline.

5.6 Construction of Object Presentation Methods

After defining the path of an Object Presentation Section for the associated Object of Interest, the author has to define the algorithms and parameters that will be used in the generation of the Projection Views of the Time Lapse video in that section. These algorithms and parameters define the Object Presentation Method. The author is provided with the option to set interactively the values of each of the Projection View parameters, the heading angle, the pitch angle and the field of view. These values will be used along the movement on the Object Presentation Section. The author can set these values by interacting with a StreetView Panorama Object, which projects the Panoramic Picture that corresponds to the first LatLng point of the section, by rotating left or right to change the heading, up or down to change the pitch and by zooming in or out to alter the field of view. The system receives the updated values of the angles through the POV changed event of the StreetView Panorama Object. He can combine the manual definition of those angles with other Presentation Methods.

Additionally, the author of the system is provided with options to create a **Panning** or **Tilting Head actions**. The Panning action may be used to show a wider area around the user (up to 360 degrees around him for a given Head Tilt or Projection View Pitch). The Pan Head Action may also be used within a Navigation Slowdown to give an overview of an opening wider view with several interesting objects and may have a short narration and presentation timing associated with it.

The author of the Route has to set the start and end angle of the Panning Action and the number of Projection Views, number of Panoramic Pictures, he wants to be shown. He also has to define the rotation of the camera, either a clockwise or counter-clockwise turn. Much like the manual setting of the Projection Views parameters, the interface provides a similar interface to the author. He can define the start and end angles of the Panning through rotation events on a StreetView Panorama Objects.

The **Tilting** head action is handled in a similar way. This action may be used to show up or down (for a given heading or Projection View Heading in the x axis) for example for viewing a very tall buildings that is located very near to the user. The author of the Route, first set the heading angle, which points at the direction where the tilting action will be implemented. Likewise, the Panning action he sets the start and end angle of the Tilting Action and the number of Projection Views he wants to be shown. He also has to define the rotation of the camera, either upwards or downwards. Screenshots of this interface are provided on Chapter 6.

Finally, if the author does not provide a specific presentation method, the system follows the automatic algorithms, developed for the presentation of the associated Object of Interest, depending its shape. These algorithms will be analyzed in the following subchapters.

5.7 Construction of UserCaptured Panoramic Pictures Route Section

This service is responsible for creating a UserCaptured Panoramic Pictures Route Object. It receives a LatLng array that defines the path where the UserCaptured Panoramic Pictures are located, a number of UserCaptured Panoramic pictures uploaded by the author on the system. The service returns a response in form of a polyline on top of the Google Map and a sequence of Google Maps Markers that are associated with the UserCaptured Panoramic Pictures.

As we mentioned in the Requirements Analysis of our Model, the key factor that led us to using the Google Maps JavaScript API was that it included the Street View Coverage API. There are other APIs that offer imagery for streets and roads but the Street View Service developed by Google offers the best available coverage across most countries in the world, including both cities and countryside.

There are some cases though where parts of a street or an area do not have available imagery. This is an issue for our application since it is based on obtaining this panoramic imagery and presenting it to the users through a sequential projection, offering them that way an interaction with all the available Points of Interests that are placed across, around and even in a given distance of the path. We want to give the users the ability to set paths through areas that do not have available Street Coverage. To resolve this issue we give the users the ability to upload their own Panoramic Pictures and adding them across the path. This functionality could also be used to replace available imagery. This would be useful in cases such as where Objects of Interest were under renovation process or the view of the Google car's camera towards them was blocked by various obstacles (such as parked cars, etc.).

In order to develop this functionality a click event listener is set to the Polyline Object, representing the Route, which handles the creation of a polyline, constructed by the LatLng points that are clicked on top of the Route's path. After submitting the polyline the user is introduced to an interface where he can upload the UserCaptured Panoramic Pictures to our system. Upon uploading a UserCaptured Panoramic Picture, the location of the GPS coordinates of the location it was taken is displayed on top of a Google Map. Besides the GPS position the system needs to be aware of the angle of the direction of the camera with respect to the True North. In case of cameras with GPS and the capability of recording EXIF metadata regarding the angle of direction, the heading is easily retrieved. In case of the cameras used to capture the UserCaptured Panoramic Pictures are incapable of recording the angle of direction, the system needs to provide a way of calculation through the interface.

Through the UserCaptured Panoramic Pictures Route Section Interface, authors are required to define a second GPS point in order to align the heading of the panoramic picture with respect to the True North. After the second GPS point is placed on the map, the Panoramic Picture so that its center is focused towards the direction of that point. The GPS point used for the calculation of the Panoramic Picture's North Heading can be a point across the Route or an object visible from the Panoramic Picture. Points or Objects in far distance offering a better alignment with respect to the true North, since the GPS accuracy is better.

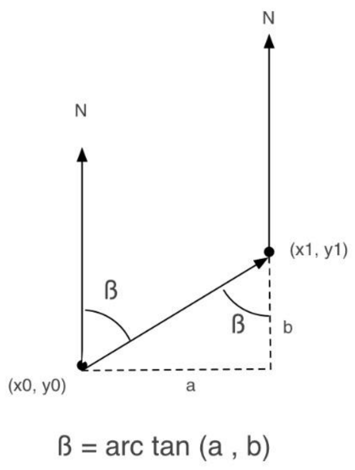
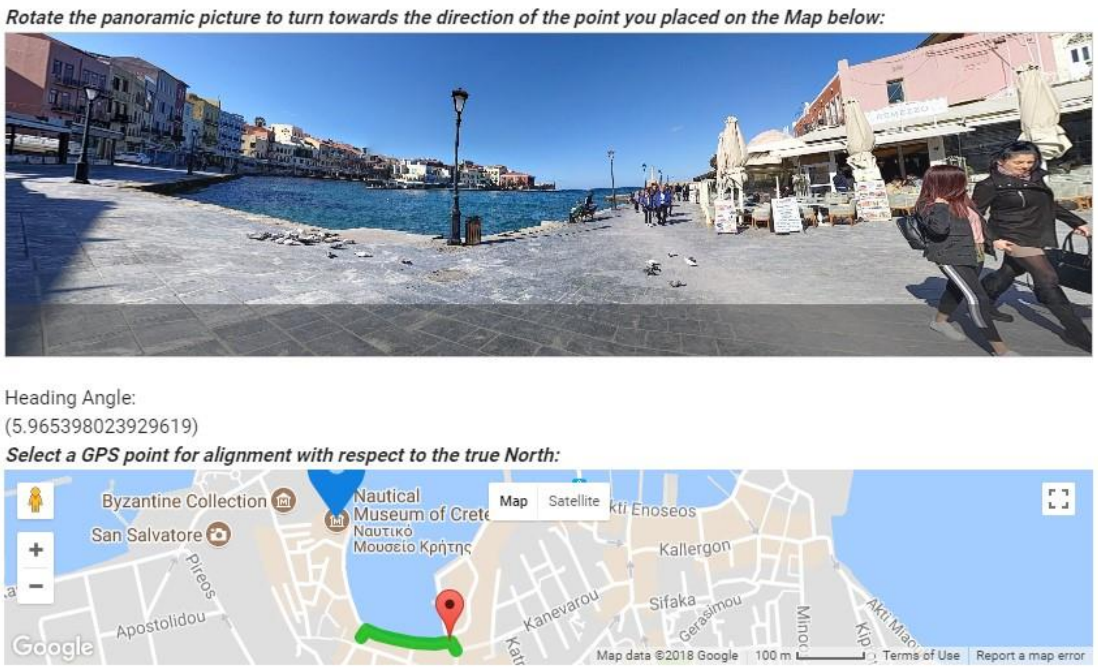


Figure 5.5 Heading Angle from point a to point b with respect to the true North. The angle is measured clockwise from the true North (0 degrees).

Let B the second GPS point we use for the alignment with the true North and A the GPS position of the uploaded UserCaptured Panoramic Picture. The angle from point A to point B with respect to the True North, angle BN (B-North), through the method angleFromCoordinate (algorithm 5.3). Angle BN must be measured clockwise. The direction of the Panoramic Picture describes the angle of the rotation of the Panoramic Picture in order to focus towards the point B. This angle will be called HB (Heading B). The heading angle of the Panoramic Picture (HN) with respect to the true North is the sum of the angles BN and HB.

Equation 5.1: HN = HB +BN

During the navigation through the UserCaptured Panoramic Pictures, the heading angle of the picture towards the road or to an Object of Interest is calculated by adding the angle from the GPS position of the UserCaptured Picture with the point the camera needs to turn to with the True North and the angle HN, which is calculated during the construction of the Route using the equation 5.1.


Rotate the panoramic picture to turn towards the direction of the point you placed on the Map below:

Heading Angle:
(5.965398023929619)

Select a GPS point for alignment with respect to the true North:

*Figure 5.6 UserCaptured Route Section Interface. In order to calculate the angle HN, the author has to define a second GPS point for the alignment and rotate the panoramic picture in order to point towards that point. The second GPS point can be a monument, visibl*e in distance, or a point on the road ahead.

The Projection View is defined by algorithms and parameters used in the construction on the time lapse video presentation of the Route. We associate each UserCaptured Panoramic picture with a Google Maps Marker. Clicking on the Marker gives the route designer the options to edit the parameters of the associated UserCaptured Panoramic Picture. The algorithm used for the construction of the polyline that is responsible for the loading of the UserCaptured Panoramic Pictures is the same one used for the construction of the Object Presentation Sections.

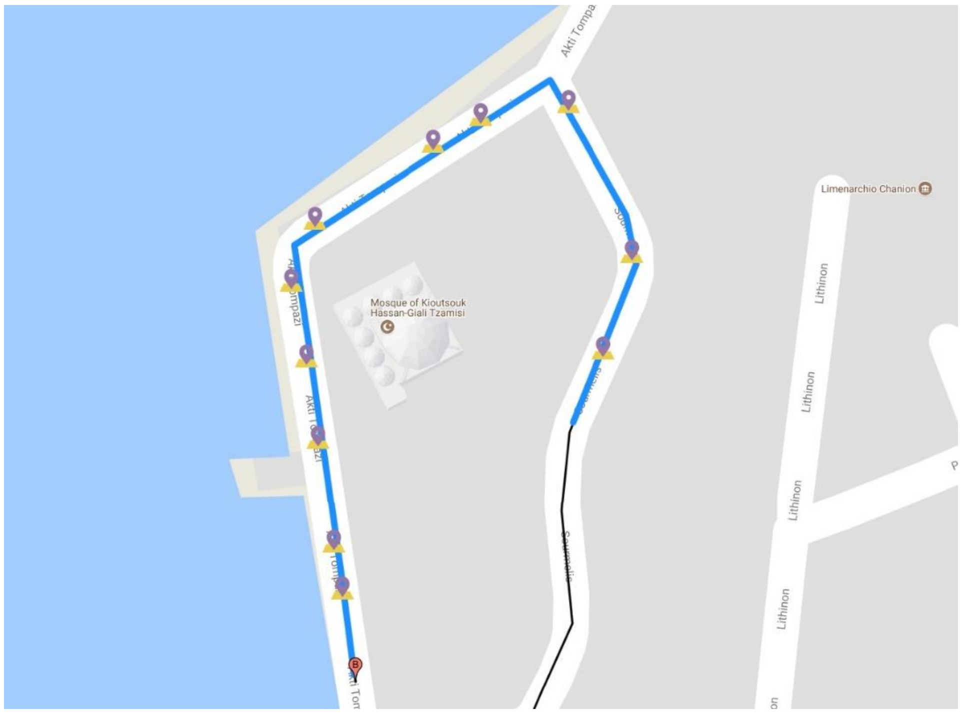


Figure 5.7 Example of Custom Panorama Objects set on top of a Route.

Algorithm 5.4: *Algorithm for UserCaptured Panoramic Picture Orientation alignment. Basic steps.*

Step 1: Upload the UserCaptured Panoramic Picture. The LatLng position of the Picture is displayed on top of the Map.

Step 2: The author submits a second GPS LatLng point, a point on the Route or a monument visible from the Panoramic Picture, which will be used as the point of alignment regarding the true North.

Step 3: After the second GPS point is submitted, the angle between the UserCaptured Panoramic Picture, the given point and the True North is calculated using the angleFromCoordinates method. This angle will be called BN.

Step 4: The author has to rotate the Panoramic Picture so that it points towards the direction of the second GPS point. The rotation angle will be called HB.

Step 5: The orientation of the UserCaptured Panoramic Picture is calculated by the sum of the angles HB and HN.

5.8 Construction of Superimposed Images on top of UserCaptured Panoramic Pictures

This service is responsible for creating the Custom Panorama Objects that have Custom Markers. It receives a Panoramic Picture, uploaded by the user, within the interface, a LatLng Position, two float values that set the point of view of the Panoramic Picture, a Radius value, a Name, an Audio File which is the narration that starts playing upon entering the Custom Panorama object and an Array with the Custom Markers Identifiers. The service returns a response in form of a Google Marker Object which is associated with a Photo Sphere Viewer Object.

Another functionality that our application supports is the assigning of Superimposed Images (pins) on top of the 360 panoramic imagery that is displayed when a user navigates across the path of a created Route. We give the Route's author the ability to add those pins either on top of the StreetView Panoramic Pictures on top of a UserCaptured Panoramic Picture, that he provides. We want the authors to be able to add markers at any position (pixel coordinates) of a Panoramic Picture, at any given location and height. As it has already been described in previous parts of this application the Google Maps API offers the possibility of placing markers both on the map and the Street View Panoramas. The google.maps.Marker constructor does not cover our needs in terms of the panorama mode, since it does not have an altitude property which could allow us to place a Marker Image (pin) at a certain height.

For that reason we have to use a custom library that allows the placement of a marker at any x, y coordinates of the Panoramic Picture. This library is the Photo Sphere viewer developed by Jeremy Heleine (posted on Github). Photo Sphere Viewer is a JavaScript library which renders 360° panoramas shots with Photo Sphere.

Photo Sphere Viewer is pure JS and based on Three.js. The **Photo Sphere Viewer** constructor handles the projection of the Panoramic Image on the given HTML div (container).

```
var photosphereDiv;

var PSV = new PhotoSphereViewer({
  panorama: 'uploadedPanoramicImage.jpg(/png)',
  container: photosphereDiv,
  loading_image: 'loadingImage.gif',
  default_fov : fov,
  default_lat: lat,
  markers: [ ],
});
```

Where:

- Panorama, is the path to the Panoramic Picture, which the user has captured using his smartphone or another 360 Photosphere capable device.
- Container, is the HTML element that will contain the panorama.
- Loading image, is the path to an image displayed in the center of the loading circle.
- Default_fov, initial field of view.
- Default_lat, Initial latitude, between -π/2 and π/2.

Types of ***PhotoSphereViewer Markers***:

```
var singleMarker = {
      id: markerID,
      x: xPos,
      y: yPos,
      image: 'pin.png',
      width: wSize,
      height: hSize,
      anchor: 'bottom center',
      tooltip: 'A image marker. <b>Click me!</b>',
      content: 'html Content'
    };

var polygonMarker = {
      id: 'polygon',
      polygon_px: [x0, y0, x1, y1, x2, y2, x3, y3],
      svgStyle: {
        fill: 'fill_Color',
        stroke: 'stroke_Color',
        'stroke-width': '2px'
      },
      tooltip: {
        content: 'A dynamic polygon marker',
        position: 'right bottom'
      }
    };
```

Where:

- X, texture coordinate (pixels) on the X-axis of the Panoramic Picture.
- Y, texture coordinate (pixels) on the Y-axis of the Panoramic Picture.
- Anchor, Defines where the marker is placed toward its position. Any CSS position is valid like bottom center or 20% 80% (Ignored for polygons).
- Tooltip, defines the html Content we show on Marker click.
    - Content, of the marker or object defining the tooltip.
    - Position, toward the marker. Accepted values are combinations of top, center, bottom and left, center, right with the exception of “center center”.
- Polygon_px**,** texture coordinates when placing a Polygon Marker.
- SvgStyle, attribute for setting the color options for the Polygon Style.

First we want to give the user the ability to select between using a default Street View Panoramic Picture (if available) on the position he decides to add the Custom Markers or using a Panoramic Picture he captured himself (using a 360 camera or a photosphere mode on his smartphone).

Therefore we present him an Interface with those available options (see User Interface chapter). We are going to analyze how we handle each option.

5.8.1 Retrieval of StreetView Panoramic Picture Binary Data

This service handles the retrieval of the binary data of an existing StreetView Panoramic Picture, which the system then uses as a UserCaptured Panoramic Picture in order to be able to set Superimposed Images on top of it. It relies on the getPanorama () method which sends a request to the StreetView Service. It takes as input the LatLng point coordinates and a given radius (around where the StreetView Service seeks for Panoramic Pictures). The StreetViewPanoramaData Object returned by the StreetView Service is handled with the methods of the HTML Canvas Object, to retrieve the binary data that represent the Panoramic Picture.

In this option the first step we have to make is to obtain the source Picture of the Street View Imagery on the selected position on the Viewing Segment's polyline. Upon clicking on the position on the map we make a **StreetViewLocationRequest** that searches for panorama data over a given area, given a passed LatLng. We make this request by calling the **getPanorama ()** function which returns a set of panorama data within a StreetViewPanoramaData object and a StreetViewStatus code. We can use the StreetViewPanoramaData Object to display the Panoramic Picture on a div. Google though does not allow the access to the source picture of that Object, which we need to use to handle the Custom Marker. Therefore we need to find a work around of getting the image (jpg /png).

We know from the Google Maps JavaScript API documentation that each Street View panorama is an image or set of images that provides a full 360 degree view from a single location. The StreetViewPanorama object uses images that conform to the **equirectangular** (Plate Carrée) **projection**[27]. Such a projection contains 360 degrees of horizontal view (a full wrap-around) and 180 degrees of vertical view (from straight up to straight down). These fields of view result in an image with an aspect ratio of 2:1. Panorama Pictures are generally obtained by taking multiple photos from one position and stitching them together using panorama software. We also know that the Street View Service supports different levels of image detail through the zoom functionality (5 levels of zoom). The StreetViewPanorama automatically calculates the appropriate field of view for the selected zoom level, and then selects imagery most appropriate for that resolution by selecting a tile set that roughly matches the dimensions of the horizontal field of view. The default zoom level within StreetView is 1 (one). The following fields of view map to Street View zoom levels:

[27] Equirectangular Projection: https://en.wikipedia.org/wiki/Equirectangular_projection

Table 5.2. Zoom Level to Field of View values

Street View zoom level	Field of View (degrees)
0	180
1 (default)	90
2	45
3	22.5
4	11.25

Each Panoramic Picture consists of an equirectangular projection with an aspect ratio 2:1. So if we use square tiles we ratio 2:1 it is easy to calculate the number of tiles consisting each Panoramic Picture. For zoom level 0 with have the base image for 2:1 tiles. **Tiles** (images) in Google Maps are numbered from the same origin as that for pixels. For Google's implementation of the Mercator projection, the origin tile is always at the northwest corner of the map, with x values increasing from west to east and y values increasing from north to south. Tiles are indexed using x, y coordinates from that origin. With each increasing level we have $4^{zoomLevel}$. For example for level 2 we have 4^2 = 16 tiles.

For our application we decided to use zoom level 4 that offers us the best image detail. Therefore we know we need to retrieve 4^4 = 256 tiles. There is not an available function in the google.maps JavaScript Library that returns those tiles. In order to be able to retrieve each tile we made a request using the following link (found in the Network tab, Chrome's Developer Tools):

URL 5.1:

https://geo1.ggpht.com/cbk?cb_client=apiv3&panoid="+streetPanoramaID+"&output=tile&x="+row+"&y="+col+"&zoom=4

Where:

- *StreetPanoramaID*: the panoID we retrieve by calling the getPanorama () function.
- *Row, col*: the index number of the rows in the x-axis and of y-axis.

The total number of rows are: $\mathbf{2^{zoomLevel}}$ **=** $\mathbf{2^4}$ **= 16 (16 x 16).**

Algorithm 5.5: Retrieval of the StreetView Panoramic Picture Tiles. Basic Steps.

Step 1: Make a StreetView request, passing the clicked LatLng point of the Object Presentation Section and a radius to the getPanorama () function.

Step 2: Run a for-loop, for the given zoom value, making calls to URL 5.1 using the retrieved StreetView panorama id obtained by the callback method of the getPanorama () function.

Step 3: Associate each retrieved tile to an Image Object. Change the Cross Origin attribute of each Image file, to bypass the security protocol in order to be able to use the source image on the Canvas.

Step 4: Draw each of these images on a Canvas element, using the drawImage method.

Step 5: Retrieve the source image of the Canvas element, using the toDataURL method.

Step 6: To be able to upload the image and use it as a UserCaptured Panoramic Picture, we need to retrieve the binary data. To achieve that we make a call to the dataURItoBlob() function which transforms the base64/URLEncoded data to raw binary data

---

This algorithm runs a for-loop depending on the Zoom value, making calls to URL 4.1 Each call retrieves an image source corresponding to a tile of the Panoramic Image. These images are set to the source attribute of the Image Objects of a two dimensional array. We also change the Cross Origin attribute of each Image file, to bypass the security protocol in order to be able to use the source image on the Canvas. Each Image Object of this array is drawn on a Canvas Object by making a call to the drawCanvasImage method.

Pseudocode of the Algorithm 5.4Q:

```
function retrieveTiles() {
        var grid = new Array();
        var totalRows = 17;
        var totalCols = 17;
        //Creating the Canvas Grid
        for (var k=0;k<totalRows;k++){
                 grid[k] = new Array();
        }
        // Tile Width and Height in pixels
        var pieceWidth = 256;
        var pieceHeight = 256;
        /*Preloading the images on canvas before we draw them.*/
     if (canvasStreet.getContext){
        var ctx = canvasStreet.getContext('2d');
        for (var row = 0; row < totalRows; row++) {

            for (var col = 0; col < totalCols; col++){

                grid[row][col] = new Image();

                var x = col * pieceWidth;

                var y = row * pieceHeight;

                //To avoid getting tainted images on the dataURL method we change
the CORS to unknown

                grid[row][col].setAttribute('crossOrigin', 'anonymous');

                grid[row][col].onload = drawCanvasImage(ctx,grid,row,col);

                //Requesting the appropriate tile

        grid[row][col].src="https://geo1.ggpht.com/cbk?cb_client=apiv3&panoid="+stre
etPanoramaID+"&output=tile&x="+row+"&y="+col+"&zoom=4&nbt&fover=2";;}}}

        if ((row == totalRows) & (col == totalCols)){

                canvasToImage(); }}
```

We draw each retrieved tile image on a HTML5 Canvas Element, by using the drawImage function:

```
var drawCanvasImage = function(ctx,grid,row,col) {

        return function() {

                //      x       The x coordinate where to place the image on the
canvas

                //      y       The y coordinate where to place the image on the
canvas

                //      width   The width of the image to use

                //      height  The height of the image to use

                //      ctx.drawImage( image, x, y, width, height);

                ctx.drawImage(grid[row][col], (row * tileWidth * multi), (col *
tileWidth * multi), tileWidth * multi, tileWidth * multi);

        }

}
```

Finally, in order to get the source image from the Canvas element we call the ***toDataURL ()*** method which returns a data URI containing a representation of the image in the format specified by the type parameter (defaults to PNG). We want to be able to upload this image to our server via our servlets so we need to convert the produced ***base64/URLEncoded*** data to raw binary data (Blob Object).

```
var streetViewPanoramaImage;
```

```
function canvasToImage() {
        var dataURL = canvasStreet.toDataURL('image/png'); // base64 encoded image
        streetViewPanoramaImage = dataURItoBlob(dataURL); // to convert to binary
        var canvasImageJPG = document.getElementById('streetPanoramaImg');
        canvasImageJPG.setAttribute( 'src', dataURL);
}
// Canvas Data URL to Binary Large Object
function dataURItoBlob(dataURI) {
    // convert base64/URLEncoded data component to raw binary data held in a string
    var byteString;
    if (dataURI.split(',')[0].indexOf('base64') >= 0)
        byteString = atob(dataURI.split(',')[1]);
    else
        byteString = unescape(dataURI.split(',')[1]);
   // separate out the mime component
    var mimeString = dataURI.split(',')[0].split(':')[1].split(';')[0];
    // write the bytes of the string to a typed array
    var ia = new Uint8Array(byteString.length);
    for (var i = 0; i < byteString.length; i++) {
        ia[i] = byteString.charCodeAt(i);
    }
    return new Blob([ia], {type: mimeString});}
```

5.8.2 Placement of Superimposed Images on top of the Existing StreetView Panoramic Pictures

This service calculates and returns the x-y Coordinates of the Custom Markers, in order to place them on top of the Panoramic Pictures retrieved by the StreetView Service (see subchapter 5.7.1). It takes as input the x-y coordinates returned by clicking on the Canvas Object that is projecting the Static StreetView Image. This Static StreetView Image is retrieved by making a call to the URL 4.2, passing the LatLng Location of the Custom Panorama Object that the Panoramic Picture is associated with, the pitch, the field of view and the heading angles. The calculation is handled by the equations we are going to analyze later on in this subchapter.

In the case of the StreetViewPanorama Object the user can interact with the retrieved panoramic image by rotating left or right, up or down till he finds the part of the picture on which he wants to add a marker. We allow that by adding an event listener on the panorama object which calls a function every time the *pov_changed* event is triggered. When the user is ready her selects to add a pin or a polygon at the panorama he is currently watching we sent a static image request to Google API, requesting a static image of the panorama, at the given heading and pitch, with height and width equal to 640px.

URL 5.2:

https://maps.googleapis.com/maps/api/streetview?size=640x640&location=lat,lng&fov=fovValue&heading=headingValue&pitch= pitch

After we receive the static image we display it to a canvas Html Element that is interactive with click events. When the users clicks inside the image we get the mouse coordinates (x, y) of the mouse and we add a marker on the image at the corresponding spot. Now, we have to add the marker at the matching pixels on the panorama. In order to do that we used geometric rules of circle and sphere.

Every StreetViewPanorama displayed by using the StreetView Service has a centerHeading angle, that is the center degree of the panorama and is specified by Google or by a user (if it is a custom panorama) when the Photosphere is created and uploaded.

This centerHeading defines the turn angle the Panoramic Picture has to turn to face the True North. In order to be able to match the static image pixels to the panoramic image, retrieved by the previous step, pixels first we had to find the distance between the center degree of the panorama and the real center of the panoramic image.

We know that the real center coordinates are:

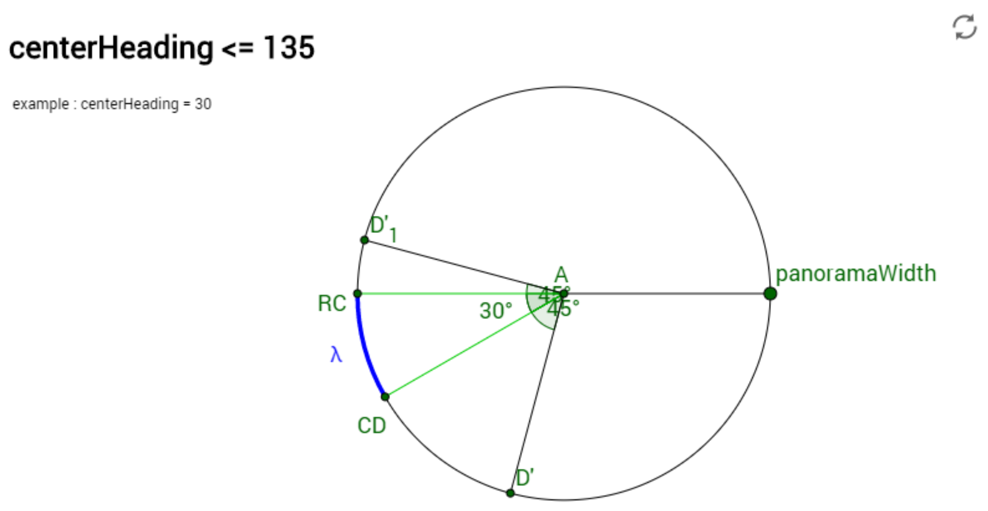
Equation 5.2:

- x-axis, x = $\frac{panoramaWidth}{2}$
- y-axis, y = $\frac{panoramaHeight}{2}$

Given the panorama centerHeading we had to calculate the pixels of the center degree of the panorama depending on the following cases:

- **centerHeading <= 135**

We know that the perimeter of the circle is 2πr, where r is the circle radius.



Where,

- RC is the real center of the image,
- λ is the arc of the $\widehat{RCACD}$,
- CD is the center degree of the panorama, the tile where heading and pitch are equal to zero,
- $\widehat{RCACD}$ is the centerHeading,
- panoramaWidth is the edge of the panoramic image,
- D', D_1' are the static image borders, expressed in pixel coordinates in the panoramic image.
- Pov, the heading point of view angle set by the pov_change listener.

We know from geometry that:

360	***2πr***
$\boldsymbol{\widehat{RCACD}}$	***λ***

In order to find λ we apply the simple method of the three:

$$\lambda = 2\pi r \frac{\widehat{RCACD}}{360}$$

Therefore the CD is equal to:

$$CD = RC - \lambda = \frac{180 - \widehat{RCACD}}{360}\, panoramaWidth = \frac{180 - centerHeading}{360}\, panoramaWidth$$

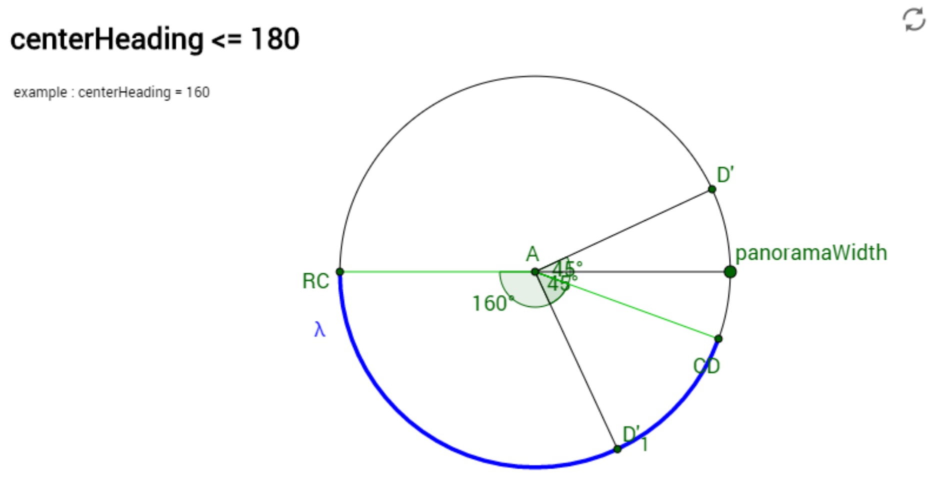
Given that we have a 90 degrees field of view when looking at a certain spot of the panoramic image it is easy to find first the borders of the static image within the pixels of the panoramic image. After we find the borders is easy to find the coordinates of the spot the user has clicked on the static image as we know both the dimensions of the static image and the panoramic image.

Borders of the static image in the case of centerHeading <= 135:

- $D'=\frac{pov}{360}$ ***panoramaWidth +*** $\frac{(180-centerHeading)+45}{360}$**panoramaWidth.**
- $D_1'=\frac{pov}{360}$ ***panoramaWidth +*** $\frac{(180-centerHeading)-45}{360}$**panoramaWidth.**

In the same way we calculate the center degree and the borders for the rest of the cases. The difference is that when the centerHeading above the 135 degrees the static image borders are within both edges of the panoramic picture.

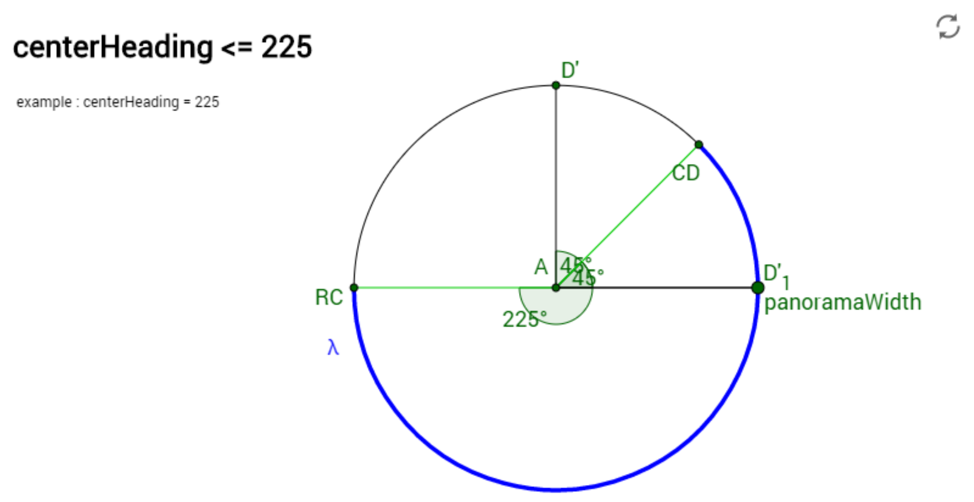
- ***centerHeading <= 180***



This differs from the previous case in that the static image borders appear in both parts of the panoramic picture edges.

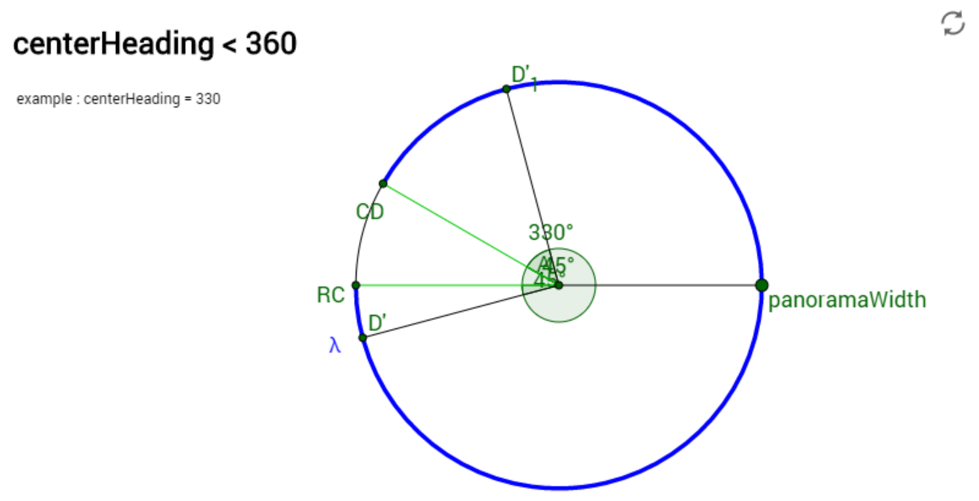
- **CD =** $\frac{\mathbf{180-centerHeading}}{\mathbf{360}}$ **panoramaWidth.**
- $\mathbf{D_1'}$ **=**$\frac{pov}{360}$ ***panoramaWidth +*** $\frac{\mathbf{(180-centerHeading) + (360 - 45)}}{\mathbf{360}}$ **panoramaWidth.**
- $\mathbf{D'}$ **=** $\frac{pov}{360}$ ***panoramaWidth +*** $\frac{\mathbf{(180-centerHeading) + 45}}{\mathbf{360}}$ **panoramaWidth.**

- ***centerHeading <= 225***



- $CD = \frac{360-(centerHeading-180)}{360}$ ***panoramaWidth.***
- $D' = \frac{pov}{360}$ ***panoramaWidth +*** $\frac{pov}{360}$ ***panoramaWidth +*** $\frac{360-(centerHeading-180)+45}{360}$ ***panoramaWidth.***
- $D'_1 = \frac{pov}{360}$ ***panoramaWidth +*** $\frac{pov}{360}$ ***panoramaWidth +*** $\frac{180-(centerHeading-45)}{360}$ ***panoramaWidth.***

- ***centerHeading < 360***



- $CD = \frac{360-(centerHeading-180)}{360}$ ***panoramaWidth.***
- $D'_1 = \frac{pov}{360}$ ***panoramaWidth +*** $\frac{360-(centerHeading-180)+45}{360}$ ***panoramaWidth.***

- $D' = \frac{pov}{360}$ ***panoramaWidth +*** $\frac{360-(centerHeading-180)-45}{360}$ ***panoramaWidth.***

These rules above calculate the center degree and the static image borders within the panoramic image coordinates. In order to calculate the coordinates of the spot the user clicked inside the static image canvas we apply once again the simple rule of the three to match the difference of the static image dimensions to the panoramic image.

X coordinate (click)	***640 (image width)***
X coordinate on panorama	***panoramaWidth***

$$x_{panorama} = \frac{x_{click}\, panoramaWidth}{640}$$

Therefore in order to add a marker to the panorama image we select the new x position to be at:

borderX2 + $x_{panorama}$

According the y coordinate of the marker on the panoramic image, we know it depends on the value of the pitch angle in the StreetView panorama. The pitch angle can take values on the range of -90 to 90 degrees. We also know the field of view is also 90 degrees, therefore the up and down borders of the static image inside the panoramic image are calculated based on the following type:

$$bordery1 = \frac{90-pitch}{180}\, panoramaHeight + \frac{panoramaHeight}{4}$$

$$bordery2 = \frac{90-pitch}{180}\, panoramaHeight - \frac{panoramaHeight}{4}$$

And the y coordinate of the marker in the panoramic image:

$$y_{panorama} = \frac{y_{click}\, panoramaHeight}{640}$$

Therefore in order to add a marker to the panorama image we select the new x position to be at:

borderY2 + $y_{panorama}$

After calculating the pixel coordinates of the Marker or Polygon on the Panoramic image we store the calculated coordinates in pixels on our PostgreSQL database so we can retrieve them later on and set them in the markers of the Photo Sphere Viewer Object that handles the Custom Panorama Object that holds the Custom Markers.

5.9 Placement of Superimposed Images on UserCaptured Panoramic Pictures

We want to give users the ability to add markers on UserCaptured Panoramic Pictures contained alongside the application's routes. In order to preserve uniformity alongside the application, we handle this added functionality the same way we do for the Google's Street View 360 Panoramas. When a user walks alongside either a premade route or an interactive one, we allow him at any point to stop and interact with the current panorama either it is a UserCaptured Panoramic Picture or a StreetView Panoramic Picture. We explained how we handle the case of the StreetMap Panoramics Pictures above, so now we have to analyze how to handle this functionality for the UserCaptured Panoramic Pictures.

So as we said in order to maintain uniformity we handle the functionality in the same way as in the case of Streetmap Panoramics, by returning the user a picture, which is a part of the full panoramic picture, depending on his viewing angle. In the StreetView case we get this picture by requesting a static image by passing the heading and pitch angles and the latitude and longitude coordinates as parameters. However in the case of the OutofStreetmap Panoramics we have to calculate and crop the corresponding image of the full panoramic image. To achieve that we use the canvas html element and its functions. We draw the image on the Canvas html element by passing the desired panoramic image from which we want to crop and take a part. To draw the image on the canvas we use the **drawImage** function by defining from which point on the X and Y axis we are going to crop, the original image's width and height (parameters that define the zoom value) and last the cropped image's size and width (dimensions of the image we present to the user).

context.drawImage (img, sx, sy, imageWidth/4, imageHeight/2, x, y, sWidth, sHeight);

Where,

img	, specifies the panoramic image we used
sx	, the x coordinate where to start clipping
sy	, the y coordinate where to start clipping

swidth	, the width of the clipped image (640 px)
sheight	, optional. The height of the clipped image (640 px)
x	, the x coordinate where to place the image on the canvas, in our case it is equal to zero.
y	, the y coordinate where to place the image on the canvas, in our case it is equal to zero.
width	, the width of the image to use (stretch or reduce the image). In our case it is equal to the ¼ of the original panoramic image's width, in order to take the right scale.
height	, the height of the image to use (stretch or reduce the image). In our case it is equal to the ½ of the original panoramic image's height, in order to take the right scale.

In order to find the x and y coordinates where to start cropping the panoramic image we used the following equation:

-For s_X **:**

Case longitude < π:

$$s_X = \left(\frac{panoramicWidth}{2} + \left(\frac{longitude}{2\pi} \cdot panoramicWidth\right)\right) - \frac{panoramicWidth}{8}$$

Case longitude > π:

$$s_X = \left(\left(\frac{longitude}{2\pi} \cdot panoramicWidth\right)\right) - \frac{panoramicWidth}{8} - \frac{panoramicWidth}{2}$$

-For s_Y **:**

$$s_Y = \left(\frac{panoramicHeight}{2} + \left(\frac{latitude}{\pi} \cdot panoramicHeight\right)\right) - \frac{panoramicHeight}{4}$$

Where,

- ***PanoramicWidth***, the width of the original panoramic image.
- ***Longitude***, the angle for the x axis (we divide by 2π because we have a range –π to π, 360 degrees).

- ***Latitude,*** the angle for the y axis (we divide by π because we have a range –π/2 to π/2, 180 degrees).

After presenting the picture to the user in a HTML5 canvas element, which is interactive in click events, he can add a marker at any point of the picture. This is done in a similar way with the case of the Streetmap panoramics. In this occasion though we do not have a centerHeading angle and the heading and pitch angles are replaced with latitude and longitude coordinates (angles) in the panoramic picture. So we have to adjust the equations for the static image side and up-down borders to the new variables. After we have calculated the image border's it is easy to find the x-y coordinates of the click event by using the simple rule of 3 as we have already discussed above.

Borders on the x axis:

$$\text{borderX1} = \left(\frac{panoramicWidth}{2} + \left(\frac{longitude}{2\pi} \cdot panoramicWidth\right)\right) - \frac{panoramicWidth}{8}$$

$$\text{borderX2} = \left(\frac{panoramicWidth}{2} + \left(\frac{longitude}{2\pi} \cdot panoramicWidth\right)\right) - \frac{panoramicWidth}{8}$$

Borders on the y axis:

$$\text{borderY1} = \left(\frac{panoramicHeight}{2} + \left(\frac{latitude}{\pi} \cdot panoramicHeight\right)\right) - \frac{panoramicHeight}{4}$$

$$\text{borderY2} = \left(\frac{panoramicHeight}{2} + \left(\frac{latitude}{\pi} \cdot panoramicHeight\right)\right) - \frac{panoramicHeight}{4}$$

In order to calculate the coordinates of the spot the user clicked inside the static image canvas we apply once again the simple rule of the three to match the difference of the static image dimensions to the panoramic image.

X coordinate (click)	***640 (image width)***
X coordinate on panorama	$panoramicWidth$

$$x_{panorama} = \frac{x_{click}\ panoramaWidth}{640}$$

Therefore in order to add a marker to the panorama image we select the new x position to be at:

$$\textbf{borderX1 + (1 + (}\frac{1}{croppedImageSize}\textbf{))} \cdot x_{panorama}$$

In a similar way we find the y position of the marker in the panorama image:

$$\textbf{borderY1 + (1 + (}\frac{\frac{1}{croppedImageSize}}{8}\textbf{))} \cdot y_{panorama}$$

After calculating the pixel coordinates of the Marker or Polygon on the Panoramic image we store the calculated coordinates in pixels on our PostgreSQL database so we can retrieve them later on and set them in the markers of the Photo Sphere Viewer Object that handles the Custom Panorama Object that holds the Custom Markers.

5.10 Database Connection

We store the created Route, Objects of Interest and Presentation Section Objects in our database by using REST services via Ajax requests. To save each Object we call a Post Request to our server that handles the connection with the PostgreSQL Database and makes the appropriate Insert Queries, via java servlets, through which we fill the Entities Tables.

5.11 Automatic Camera Orientation and Tuning

This service retrieves the Route Object and its associated Objects of Interest, along their Viewing Sections and Custom Markers, and the UserCaptured Panoramic Pictures in order to dynamically generate and present the personalized time lapse video of the Virtual Tour. It holds all the methods that receive the Route Object Identifier and by appropriate GET requests retrieves all the required Objects from the Database. This service contains the algorithms that are responsible for the automatic handling of the camera across the path or when approaching Objects of Interest creating that way the Story telling of the Route.

In this section we are going to analyze the algorithms and functions we used in order to construct the sequential projection of the StreetView Imagery across the path of a created Route and the handling of the camera's directions while navigating through the roads/ streets or when facing an Object of Interest. The main function of this application is to present the designed Routes to the application's users in a way that informs them about the location of the included points of interest creating that way a virtual tour.

After the user selects the Route he wants to view among all the planned routes that are introduced to him by the application's interface, the system retrieve the Route Object, the Objects of Interest with the Presentation Section Objects that are associate with them, the Custom Panorama Objects and the Custom Marker Objects if there are any available. The system recovers the tuples containing the information for those objects from the PostgreSQL Database for the application by making REST Get Calls via Ajax requests. Then it initializes those objects on the map representing them with the way we analyzed on the Route Creation Section. Specifically, the path (Route) and the sections are represented by google.maps.Polyline Objects, the single type Points of Interest, the Panorama Objects and the Custom Markers are represented by google.maps.Marker Objects and finally the polygon type Points of Interest are represented by google.maps.Polygon Objects. An example of a retrieved Route can be found in figure 4.5.

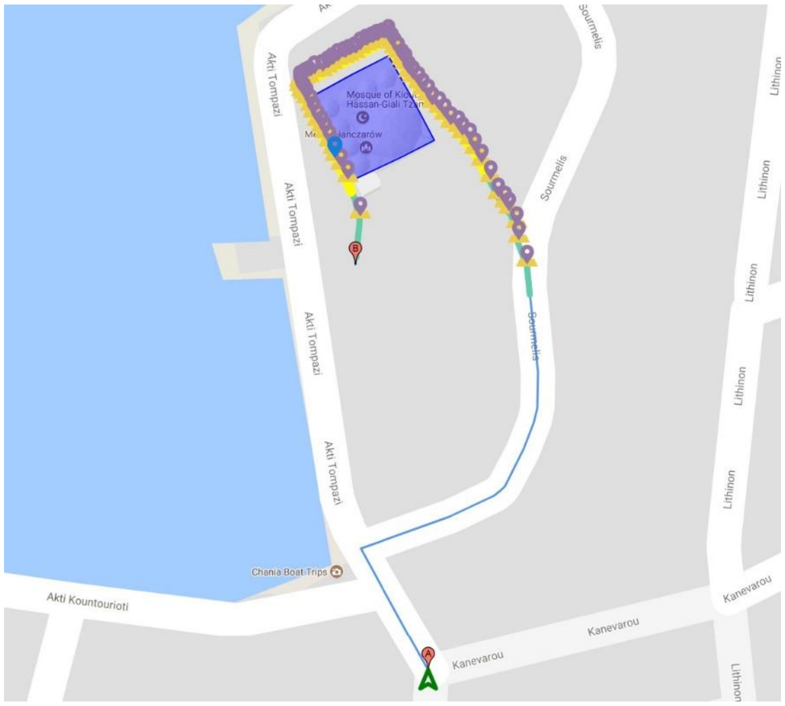


Figure 5.8 An example of a retrieved Route Object with the associated Objects of Interest, Viewing Sections, Panorama Objects.

In order to construct the serial presentation of panoramic images through which the system simulates the virtual navigation of the user along the route, the user must move along the path of the path to the map by calling StreetView service at each point. To achieve that the system creates a google.maps.Marker object that will represent the camera's fixed movement across the path (**Camera Marker**). First it creates an Icon Object that will handle the rotation of the Marker's image as the user moves across the path. One of the available google maps symbols, an Arrow, is used to simulate the camera's direction as we animate the marker on the polyline. The Camera Direction is a line that starts from the camera Marker and indicates the direction of the camera in order to produce the Projection View.

```
var icon = {
  path: google.maps.SymbolPath.FORWARD_CLOSED_ARROW, // From the Symbols Library
  scale: 6,
  strokeColor: 'green',
  offset: '25%' // Offset of the Icon according to the Marker's Position
};
//Function that sets the Marker we use for the Movement
function createMarker(latlng, label) {
        var marker = new google.maps.Marker({
           position: latlng,
           map: map,
           title: label,
           icon: icon // The Icon Object, holding the Arrow Image.
         });
         marker.myname = label;
        return marker; // Returning the Created Marker Object.}
```

We describe the functions and the algorithms that will handle the animation of the marker, the animation of the camera both for the StreetView and for the UserCaptured Panoramic Pictures. First we describe a function that initializes the variable of the Route's polyline and the Marker and calls the recursive function animate.

Algorithm 5.6: *Algorithm for Handing the Start, Pause and Termination of the Animation. Basic Steps.*

Step 1: Set the path of the Route's polyline.

Step 2: Start making recursive calls to animate function with a given timeout.

Step 3: If timeout is cleared, stop the animate Function.

Pseudocode of Algorithm 5.6:

```
var polyline; //The retrieved Route Object
var polyDistance; //The length of the Polyline
var startinStep; //The number of meters we move across the Polyline's path
function startAnimation() {
        if (timerHandle) clearTimeout(timerHandle); // Stop the recursion
        polyDistance =polyline[index].Distance();
        poly2 = new google.maps.Polyline({path:
[polyline[index].getPath().getAt(0)], strokeColor:"#FFFF00", strokeWeight:4});
        timerHandle= setTimeout(animate(startingStep),animationSpeed);   // Allow
time for the initial map display}}
```

Where,

- **Polyline**, the retrieved Route Object.
- **PolyDistance**, is the table holding the value of the route's distance.
- **Poly2**, is the table used to spawn a new polyline every 20 vertices. We initialize it at the first vertex of the Route's polyline.
- **StartingStep**, is the number of steps by which the camera (user) proceeds alongside the preplanned route expressed in meters.
- **AnimationSpeed**, is the variable that defines the speed we move alongside the route.

In the animate function we handle the movement of the marker across the path and the direction of the camera across the path. In order to create a uniform image sequence that handles as smooth as possible, manner the changes in direction, the virtual simulation we need to get appropriate orientation of the camera at each point of the path. We have to calculate the direction of the camera focus at any point of the path in order to present the right Projection View Object. We have to define the Field of view, the Heading and the Pitch angle.

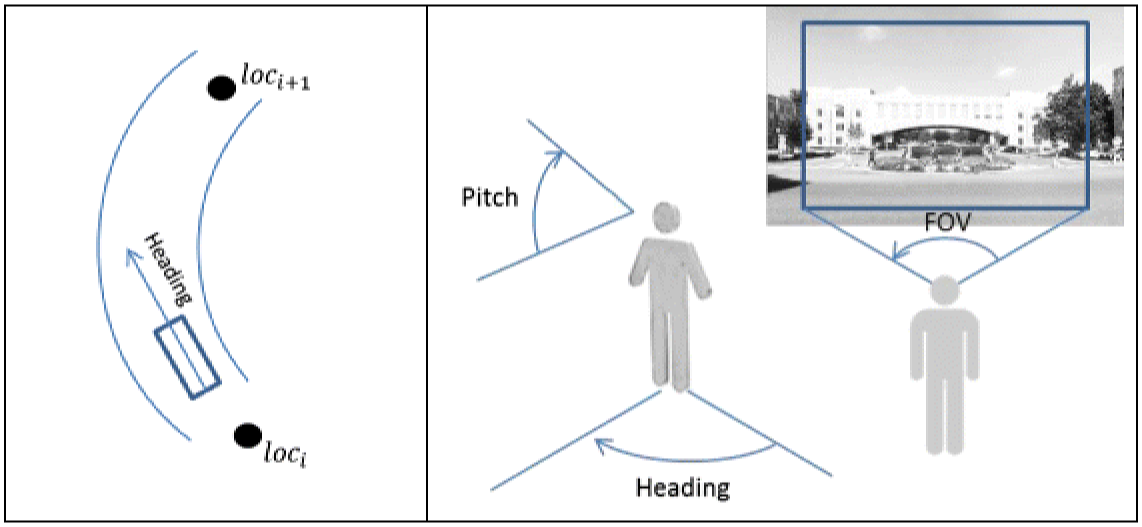


Figure 5.9 The three angles defining the Camera Direction.

The Field of View angle is set by default on 90 degrees. To calculate the Pitch angle, which defines the movement of the camera on the y-axis (up-down) we tried to use the Elevation Service of google maps. The elevation of a geographic location is its height above or below a fixed reference point, most commonly a reference geoid, and a mathematical model of the Earth's sea level as an equipotential gravitational surface. The Elevation Service request takes as input a set of one or more locations and returns the elevation value in an output Format either in JSON or XML.

Our thought was that if we calculated the elevation values of two successive points on the path we could find the difference in sea level between them. That way we could recognize if the camera was moving up or down a hill, up and down stairs in some street and calculate a pitch value that would represent that movement. The responses of the Elevation service though were not accurate enough when the successive points were within short distance and the elevation values were too similar to help us calculate the pitch angle.

In order to calculate the final pitch angle we used the Street View Service method getPhotographerPov (). This method returns the direction the Google car or trike was facing. We used this assuming the photographer is either facing upwards or downwards depending on the angle he wants to shoot, satisfying that way the case of a downhill or an uphill.

Finally, we have to calculate the Heading angle for the point of view. The heading angle shows the direction we are going to move next, which could be either forwards, either left or right. This way the visual presentation of the panoramic images guides the user across the path. To achieve this result we calculate the heading angle between two successive points in the path. Passing this angle and the current position of the camera to the Street View Service, the returned StreetViewPanorama Object is facing the next position. To calculate the heading (bearing) angle between the two points we used the following mathematic formula:

Equation 5.2: $$\theta = \operatorname{atan2}(\sin\Delta\lambda \cdot \cos\varphi_2 , \; \cos\varphi_1 \cdot \sin\varphi_2 - \sin\varphi_1 \cdot \cos\varphi_2 \cdot \cos\Delta\lambda)$$

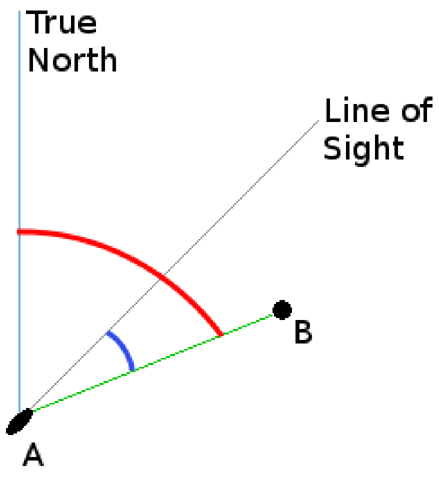


Figure 5.10 Heading angle between two points.

Where,

φ_1	, latitude of A
φ_2	, latitude of B
λ_1	, longitude of A
λ_1	, longitude of B
$\Delta\lambda$	, λ_2 - λ_1

When moving across the path (Algorithm 5.6), we also need to take into consideration the case of facing a turn.

Algorithm 5.7: *Algorithm for smoother Camera Orientation in case of a Turn. Basic Steps.*

Step 1: In order to detect if we are facing a turn we took three consecutive points of the path. The current position of the User, the next position and the position in x steps.

Step 2: These functions form a triangle. We calculate the angle B using the equation 5.3.

Step 3: If the angle B is greater than 170, the user is moving in a straight line. The angle of heading is the angle between the point of the current position, the point in the next and the True North.

If the angle B is less than 170, the user is facing a turn so the system has to start rotating the camera to perform a smoother transition. The heading angle is the angle between the current point, the point in x steps and the True North.

We calculate the B angle of the triangle with the equation below and we have two cases:

Equation 5.3: $$\cos \widehat{B} = \frac{(a^2 + c^2 - b^2)}{(2 \cdot a \cdot c)}$$

- **Case $\widehat{B}$ < 170 degrees :**

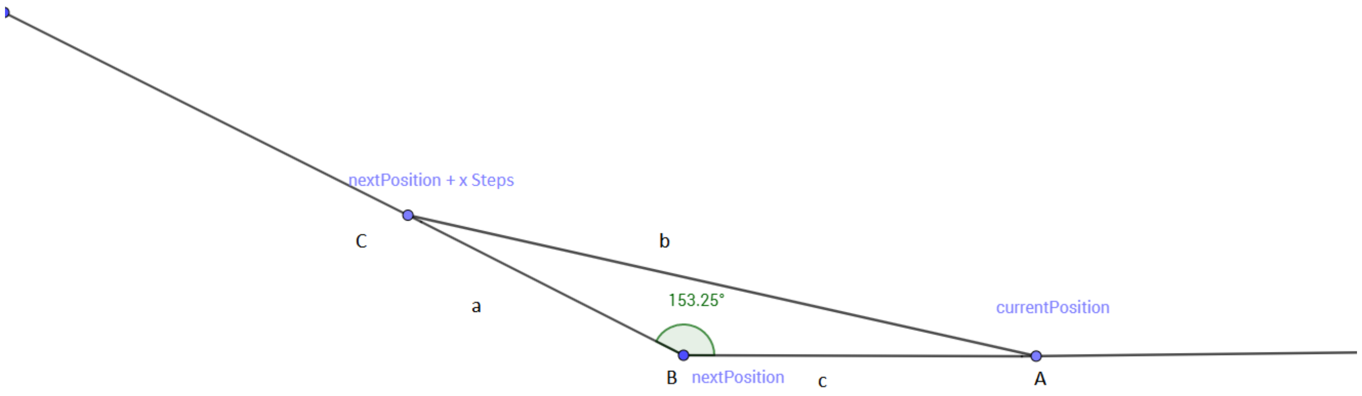


Figure 5.11 Case angle B <170 thus facing a turn.

If the angle B is less than 170 degrees (Figure 5.10) then we are facing a turn so we calculate the new heading angle by using the Geometric equation we mentioned on the previous step between the current position and the position+numberOfSteps ahead.

- **Case $\widehat{B}$ > 170 degrees :**

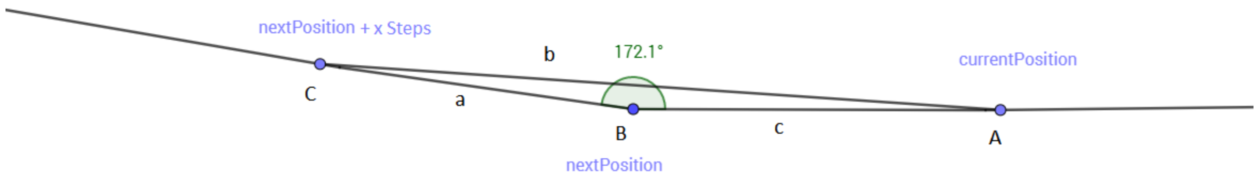


Figure 5.12. Case angle B > 170 thus facing a straight line.

If the angle B is more than 170 (Figure 5.11) then we are moving on a straight line so we calculate the new heading angle by using the Geometric equation we mentioned on the previous step between the current position and the position+10 steps ahead.
The function we used to calculate the **angle $\hat{\boldsymbol{B}}$**.

Pseudocode of Algorithm 5.7:

```
function calculateHeadingPov(a,b,c){

        //calculating the triangle sides

        var sideAB = getDistanceFromLatLonInKm (a, b);

        var sideBC = getDistanceFromLatLonInKm (b, c);

        var sideAC = getDistanceFromLatLonInKm (a, c);

        //calculating b angle

        var angleB = toDegrees(Math.acos(( Math.pow(sideBC,2) + Math.pow(sideAB,2) -
Math.pow(sideAC,2) ) / (2 * (sideBC * sideAB) )));

        return angleB;}
```

In order to calculate the sides of the triangle we used the Haversine Formula. If we have two different latitude – longitude values of two different point on earth, then with the help of Haversine Formula, you can easily compute the great-circle distance, which is the shortest distance over the earth's surface. We construct the JavaScript function getDistanceFromLatLonInKm based on this formula.

Haversine Formula[28]:

R = earth's radius (mean radius = 6,371km)

Δlat = lat_2 - lat_1

Δlong = $long_2$ - $long_1$

a = $\sin^2(\frac{\Delta lat}{2}) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin^2(\frac{\Delta long}{2})$

c = 2 · $atan(\sqrt{a}, \sqrt{(1-a)})$

Equation 5.4: d = R · c (final calculate Distance)

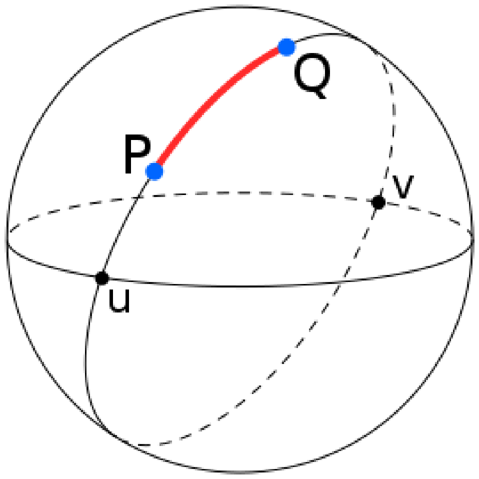


Figure 5.13 Great Circle Distance between points P & Q.

Now we need to construct the animation function. Inside this function we call the calculateHeadingPov function that defines the heaving angle of the street view in the case we are not approaching an Object of Interest, the function checkIfPointOfInterest that checks if we are approaching an Object of Interest or a UserCaptured Section and finally the updatePoly function that updates the polyline's vertices to simulate the movement across the path. Every time we call the animate function we check if the position we are about to move is surpassing the length of the Route's Polyline, if not we get that position on the polyline using the function GetPointAtDistance (index). After calling the two functions that calculate the appropriate heading angle we make a Street View Service request that returns a StreetViewPanorama Object which we add to our visual sequence.

[28] Haversine Formula: https://www.math.ksu.edu/~dbski/writings/haversine.pdf

Algorithm 5.8: Algorithm of Animation across a Path. Basic Steps.

Step 1: First check if the current point surpasses the total Distance of the Route path, if it does then exit the animate function.

Step 2: At each point, check if the road will face a straight line or a turn (see Algorithm 5.7).

Step 3: At each point check if the user moves along a UserCaptured Panoramic Pictures Route Section. If he does, load the PhotoSphereViewer Interface, which handles the Projection of the UserCaptured Pictures. If not, call the getPanorama method to retrieve the StreetView Panoramic Picture, passing the appropriate projection view parameters.

Step 4: Also check if the user is moving along an Object Presentation Section. If he does, check if the associated Object of Interest matches his interests.

Step 5: If the associated Object matches his interest, handle the Projection Views along the section depending on the defined Presentation Method.

Step 6: Rotate the Arrow Icon, representing the direction the user faces, to match the heading angle.

Step 7: Move to the next point and make a call to the animate function.

Pseudocode of Algorithm 5.8:

```
function animate(index, d) {
   //Stopping Condition, when reaching the ending point
   if (d > polyDistance) {
      return;
   }
    //Checking if there is a turn coming ahead
    var p = polyline.GetPointAtDistance(d); // Current Position
    position = p;
    var temp;
    var temp2;
    if (d + (2*step) < polyDistance) {
        temp = polyline[index].GetPointAtDistance(d+step);
        temp2 = polyline[index].GetPointAtDistance(d+ (2*step));
        var calcHP =  calculateHeadingPov(p,temp,temp2);
        if (calcHP > 170){
            headingPov = angleFromCoordinate (p, temp);
        }else{
            headingPov = angleFromCoordinate (p, temp2);
         }
    } else if(d + step < polyDistance) {
          temp = polyline[index].GetPointAtDistance(d+step);
          headingPov = angleFromCoordinate(p, temp);
```

```
     }else{
          temp = polyline[index].GetPointAtDistance(d+1);
           headingPov = angleFromCoordinate(p, temp); }
    sv.getPanorama({location: p, radius: 70}, processSVData); // Getting the
panorama object
    map.setCenter(p); // Updating the map center so that maps moves when we update
the polyline
    pitch = panorama. GetPhotographerPov().pitch;
    checkIfPointOfInterest(p); // Function that handles the Camera when approaching
an Object of Interest
    // Moving marker at the updated position
    icon.rotation = headingPov; // Rotating the icon according to the heading
    marker.setIcon(icon);
    marker.setPosition(p);
    startingStep = d + step; // Update the position on the next point
    timerHandle= setTimeout(animate(startingStep),animationSpeed); // Recursive
call to the animate function, to repeat process for the next position
}
```

We need to construct the function GetPointAtDistance which allows to move across the Polyline. This function will return null if the given distance (in meters) is bigger than the total length of the polyline. If the given distance is less than the polyline's length then we calculate the point at that distance by calculating an adjusting variable that we use to find the point between the last two vertices in that given distance. This algorithm will be used for the animation across the path of the Route's polyline.

Algorithm 5.9: Algorithm for Calculation of Point at a given distance on a Polyline path. Basic Steps.

Step 1: We pass the current distance as input.

Step 2: If the given distance is zero, the response is the first Vertex of the Polyline.

Step 3: If the length of the polyline is smaller than 2 meters or if the given distance is negative the algorithm returns null.

Step 4: The algorithm calculates the point by finding the two last vertices and by using an adjusting factor it calculates the point within these two vertices.

Step 5: The calculated point is returned as a response by the algorithm.

Pseudocode of Algorithm 5.9:

```
//A method which returns a GLatLng of a point a given distance along the path
//Returns null if the path is shorter than the specified distance
function  getPointAtDistance(meters) {
  // Some special cases
  if (meters == 0) return this.getPath().getAt(0);
  if (meters < 0) return null;
  if (this.getPath().getLength() < 2) return null;
  var dist=0;
  var olddist=0;
  for (var i=1; (i < this.getPath().getLength() && dist < meters); i++) {
   olddist = dist;
   dist+=getDistanceFromLatLonInKm(this.getPath().getAt(i),(this.getPath().getAt(i-
1)); // Total distance up to the given meters
  }
  if (dist < meters) { // path is shorter than the specified distance
    return null;}
  var p1= this.getPath().getAt(i-2); // Getting the last 2 vertices
  var p2= this.getPath().getAt(i-1);
  var m = (meters -olddist)/(dist-olddist); // Adjusting parameter
  return new google.maps.LatLng( p1.lat() + (p2.lat()-p1.lat())*m, p1.lng() +
(p2.lng()-p1.lng())*m);}
```

5.12 Detection of Objects of Interest and Handling of the Camera

On this section we are going to analyze how we handle the animation of the camera when we are approaching an Object of Interest. As we mentioned in the Route Construction section, the creator of the route has to associate each Object of Interest with one or more sections on the path. Using those sections we can detect if we are approaching an Object of Interest. Each Presentation Section Object is also associated with an audio file that alerts the user about the Objects of Interest which is located near him with a short narration. This narration can inform the user about the location of the Object of Interest or present a short description about it.

In the function checkIfPointOfInterest the system performs the necessary controls to determine whether if the user is within one of those sections, passing the current position as a single input parameter. When the user enters the first point of one section the narration associated with it starts playing in order to notify the user. As he moves across the section we must calculate and update the angle of the camera depending on the type of the Object of Interest that the section is connected with. In order to detect if he is near or on a section the system makes use of the isLocationOnEdge function of the google.maps.geometry.poly library that we explained earlier on the Object Presentation Section's part. It calls this function by passing as parameters the polyline object, which represents the path of the section, the current position of the user and a tolerance value defined at 10e-5 after multiple experiments. Even though this function is accurate enough, when two Presentation Sections are in close distance it returns true for both of them. To avoid this issue the system calculates the minimum distance of the current position and the section's polyline points and the distance of user's position from the polyline's first point as he moves across the section. If the closest point of the section's polyline is within a three meter distance of the current position and the distance of the first point is less than the total length of the section then we are moving within the section. Therefore in order to detect if the user is approaching one of the sections that are set across the Route, the system runs a for-loop for all the retrieved sections checking if the conditions above are satisfied.

Algorithm 5.10: Algorithm for Calculation of minimum distance between current Position and points across a Polyline. Basic Steps.

Step 1: We pass the current position of the user as input.

Step 2: It runs a for-loop for the Polyline's points calling the getDistanceFromLatLonInKm between the position and each point. Comparing and storing the minimum distance

Step 3: Returns the final minimum distance between the current position and the Polyline.

Pseudocode for Algorithm 5.10:

```
function calculateAreaPointWithMinDistance(currPosition, closeArea){
        var d=0; // Starting point
        var min = 10000; // Setting a big min value
        var minimumArray = new Array();
        var i =0;
        var pointOfMin = 0;
        while (d<closeArea.Distance()){
                var distance = getDistanceFromLatLonInKm (currPosition,
closeArea.GetPointAtDistance(d)); // Checking distance from each point
                minimumArray[i] = distance;
                i++;
                if (distance<min){
                        min = distance;
                }
                if ( (d + 1) > closeArea.Distance()){
                        break;
                }else{ d = d + 1; }
        }
        minDistance= min;
        return minDistance;
}
Function checkIfPointOfInterest(position){
    For (i=0 to segments.lenght)
        var minDistance =  calculateAreaPointWithMinDistance(positionLatLng,
segmentPolylines[i]); // Minimum distance from polyline
        if (google.maps.geometry.poly.isLocationOnEdge(positionLatLng,
segmentPolylines[i], 10e-5) && minDistance <= 3) {
           var distanceFromSegmentFirstPoint = getDistanceFromLatLonInKm
(positionLatLng, segmentPolylines[i].GetPointAtDistance(0));
if(minDistance <= 1 && distanceFromSegmentFirstPoint <=
segmentPolylines[i].Distance()){
                delay = 1500;
                .
                . // Here we are going to check for the Type of POI
                . // and custom Markers.
```

```
                    .
                    if (segmentAudioPlayed[i] == false ){ //Play the Narration
                            audio.play();
                             segmentAudioPlayed[i]= true; //So that it plays only once
                    }  }
```

After detecting that we are moving within an Object Presentation Section the system has to calculate the appropriate heading angle and change the camera's direction in order to face the Object of Interest. In order to calculate this angle, first the system has to identify the type of the Object of Interest. We have two cases:

5.12.1 Single Object of Interest:

As we already mentioned the Single Objects of Interest are represented with a google.maps.Marker Object on the map. We can use this form to identify certain parts of landmarks (for example the entrance of a museum) or landmarks with a smaller size (like a statue). By using the Marker Object as the focus of the camera the system can only point at a specific part of the Object of Interest. In order to calculate the angle of the camera the system uses the angleFromCoordinate function every time we move, passing as input parameters the LatLng position of the Object of Interest and the current LatLng position on the section. As we animate the user's marker using the getPointAtDistance function with a set number of steps, the angleFromCoodinate returns the appropriate angle which we use to retrieve the StreetViewPanorama Object. In Figure 5.14 we can see an example of this functionality. The yellow line symbolizes the polyline of the section, the green arrow the user's position on the map and the dotted blue polylines represent the direction the camera is facing at each point of the section's polyline. Next to the map we have an example of a StreetViewPanorama Object retrieved on the currentPosition of the figure 5.15 in order to show how the camera turns to face the Object of Interest.

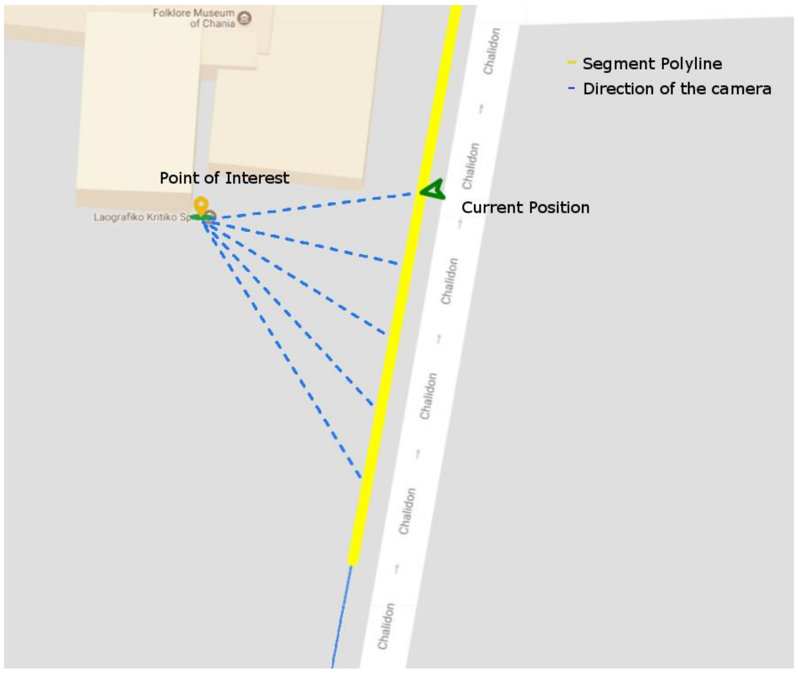


Figure 5.14 Example of the tuning of the Camera when approaching a single Object of Interest. The dotted lines represent the polyline created by the angle between the positon on the Visibility Section and the Object of Interest.

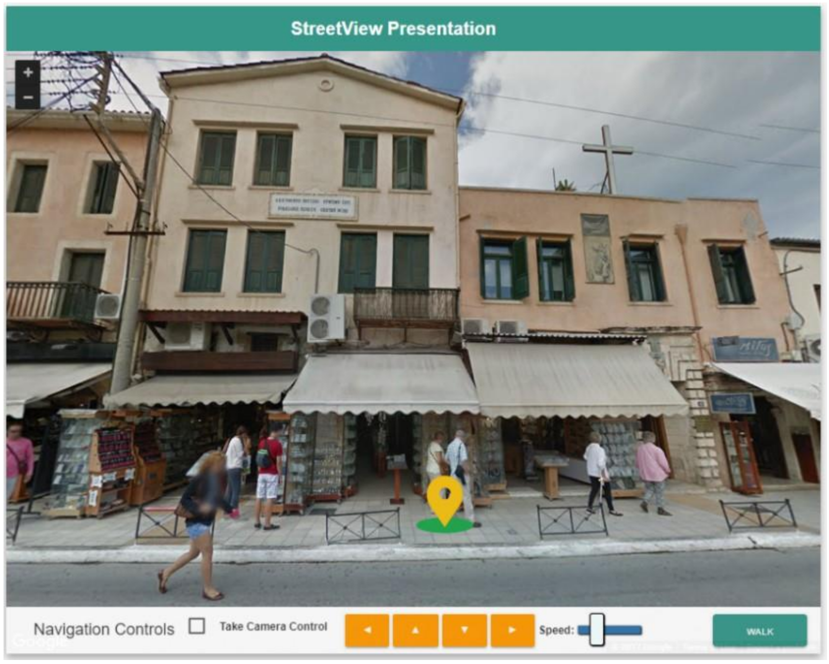


Figure 5.15 Example of the Panoramic Picture retrieved by passing the heading angle, calculated based on the Haversine Formula equations. (Folklore Museum of Chania).

5.12.2 Polygon Object of Interest

There are some cases regarding the way we want to present an Object of Interest to a user that a single point cannot satisfy. A user may want to add an area, a street or a road as an Object of interest so he can notify the user about it and draw his attention towards it. We use a google.maps.Polygon Object to represent such Points of Interest. By using a Polygon object we can also view different parts of a landmark. When handling the camera's rotation to face the polygon point there are two cases we need to take into consideration.

➢ **Overall View of the Object of Interest**

When the current position of a user, within the start or the end of a Viewing Section, is in sufficient distance from the associated Object of Interest we want to project an overall view of the Object. To accomplish that we guide the direction of the camera towards the point that the bisection separates the nearest side of the Polygon.

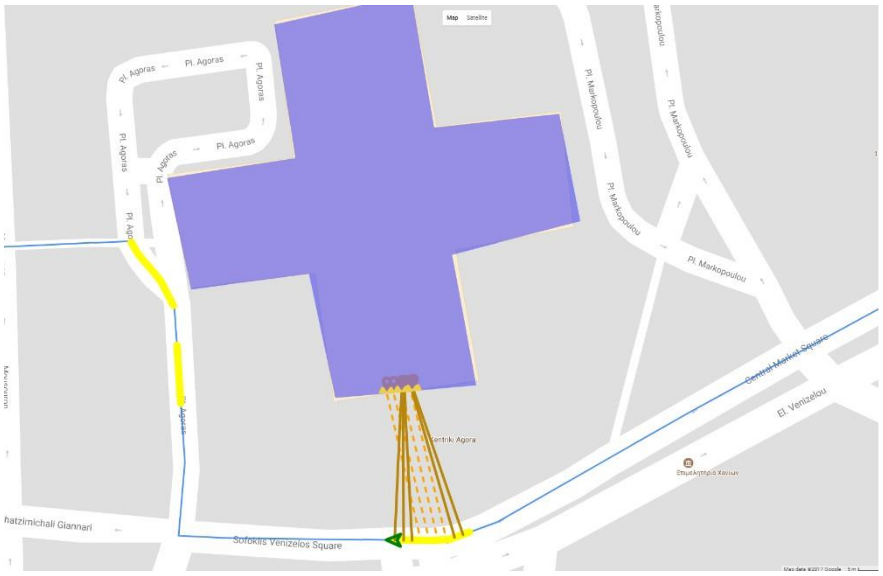


Figure 5.16 Automatic Orientation and Tuning of the Camera across a Viewing Section facing a Polygon Object of Interest. The start and end of the Viewing Section offer an Overall View of the Object (solid lines). The middle of the Section orients the Camera to offer a side view of the nearest side of the Object (dotted lines).

Algorithm 5.10: Calculation of the bisection point on the closest side of the Polygon

1. Calculating and Drawing the sides of the polygon Object of Interest.

First we retrieve the path of the Polygon Object. We run a loop for the length of this path in order to find the vertices of the polygon. We connect the vertices by pairs of two forming the sides of the polygon. For the final side we connect the last vertex with the first one. We also calculate a point located at the middle of each side. All the sides are drawn on top of the Google Map represent by a Polyline with black color.

Algorithm 5.11: Algorithm for calculating the Polygon Sides & Vertices. Basic Steps.

Step 1: Passes the polygon object to the function.

Step 2: It runs a for-loop for the polygon's length calling the getAt () method passing the index of each step in order to calculate the vertices.

Step 3: Constructing each side by passing a pair of two consecutive Vertices as a path of a Polyline Object. For the final side we pair the first and the last vertex of the Polygon.

Step 4: Storing the medium point of each side by making use of the Algorithm 5.9, finding the point at the half distance of the side.

Pseudocode of Algorithm 5.11:

```
var vertices = polygonObject.getPath(); // Retrieving the Path of the Polygon Object
var count = 0;
var sidesMedium = new Array(); // Storing the medium point of each side
for (var i=0; i<vertices.getLength()-1; i++){
        var verticePath = new Array();
        verticePath[0] = vertices.getAt(i);
        verticePath[1] = vertices.getAt(i+1);
        sides[i] = new google.maps.Polyline({ // Drawing each side
             path: verticePath,
             strokeColor : 'black',
             map: map});
        var markerMedium = new google.maps.Marker({
                position: sides[i].GetPointAtDistance ((sides[i].Distance()-1)/2)});
        sidesMedium[i] = markerMedium;
        count++;
}
var verticePath = new Array(); // Drawing the last side, with First and Last Vertices
verticePath[0] = vertices.getAt(count);
verticePath[1] = vertices.getAt(0);
sides[count] = new google.maps.Polyline({
     path: verticePath,
     strokeColor: 'black',
     map: map });
var markerMedium = new google.maps.Marker({
    position: sides[count].GetPointAtDistance((sides[count].Distance()-1)/2)});
    sidesMedium[count] = markerMedium;
```

2. Calculating the side of the Polygon Object of Interest closest to the user's position.

Algorithm 5.12: *Algorithm for calculating the closest side of the Polygon. Basic Steps.*

Step 1: Calculate the distance between the user's position and the medium point of each Polygon Side using the getDistanceFromLatLonInKm function.

Step 2: Compare the calculated distances through a for-loop to find the minimum.

Step 3: Store the index where the minimum distance was found. This index represents the closest side of the Polygon.

Step 4: Draw the closest side with an Aqua color to differentiate it from the other sides of the Polygon. (See Figure 5.16)

Algorithm 5.12: Calculation of the closest side of the Polygon

```
var d=0; // Variable to define the index of the closest side
var minimumArray = new Array();
var min = 1000000; // Variable for distance comparison
for (var j=0; j<count; j++){
     var distance = getDistanceFromLatLonInKm (sidesMedium[j].getPosition(),
position);
     minimumArray[j] = distance;
     if (distance<min){
        min = distance;
        d = j;
     }
}
var closestSide = new google.maps.Polyline({
    path: sides[d].getPath(),
    strokeColor: 'Aqua',
    map: map });
```

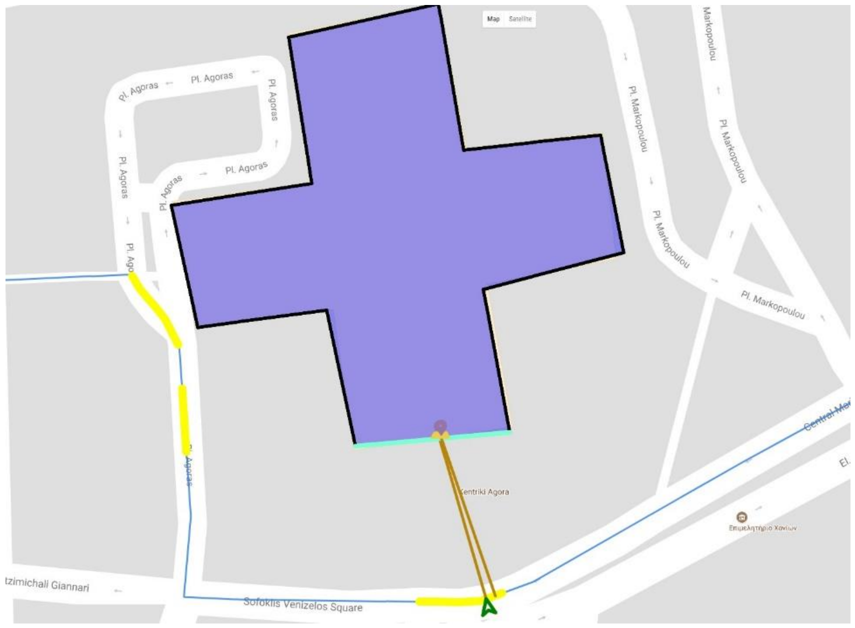


Figure 5.17 The sides of the Polygon Object. The closest side to the user's Position is represent with an Aqua Color. On this side we calculate the bisection which returns us an angle allowing an overall view of the Object.

3. Calculating the bisection from the user's position on the closest side.

In geometry, bisection is the division of something into two equal or congruent parts, usually by a line, which is then called a bisector. The most often considered types of bisectors are the section bisector (a line that passes through the midpoint of a given section) and the angle bisector (a line that passes through the apex of an angle that divides it into two equal angles). (See Figure 5.17).

To calculate the bisection of the angle that divides the closest side we used the known mathematical theorem.

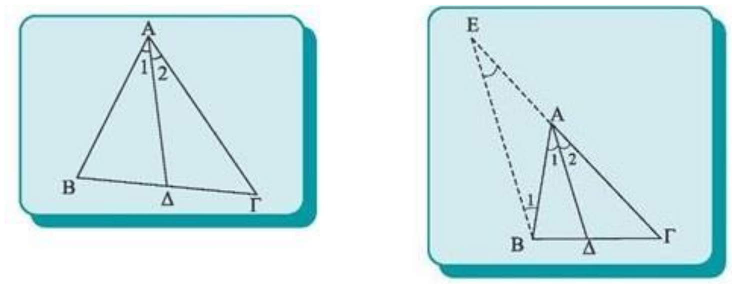


Figure 5.18 Bisection Δ of triangle ΑΒΓ. Drawing a parallel to side ΑΔ from point Β.

Theorem (internal bifurcated triangle)[29]: The bisection of the angle of a triangle divides the opposite side internally into a ratio equal to the ratio of the adjacent sides.

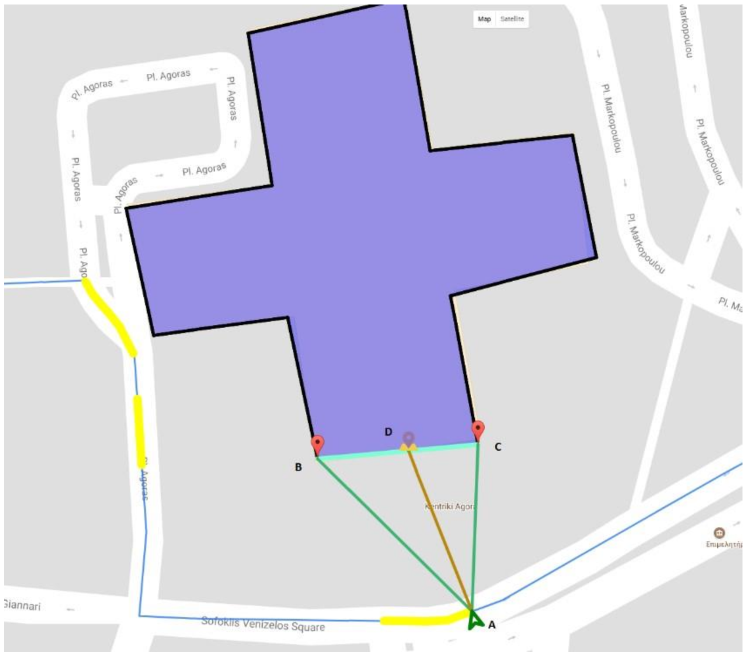


Figure 5.19 Triangle defined by the User's position and the Vertices of the closest side. We need to calculate the point D which divides the side to two parts.

If ΑΔ the bisection of triangle ΑΒΓ, then:

Equation 5.5: $\frac{\Delta B}{\Delta \Gamma} = \frac{AB}{A\Gamma}$

Proof of Theorem:

Supposing triangle ΑΒΓ and bisection ΑΔ (see figure 5.18). From edge B we draw a parallel line to side ΑΔ, which intercepts the extension of ΑΓ on E. From Thalis theorem on triangle ΓΕΒ we have:

Equation 5.6: ΔΒΔΓ = ΑΕΑΓ

In order to prove the theorem we just need to prove that AE = AB. Indeed:

- A1 = B1 (alternate angles of parallels ΑΔ and BE),
- A2 = E (corresponding angles of parallels ΑΔ and BE),
- A1 = A2 (ΑΔ bisection, divides in two equal angles),

Then we have B1 = E therefore:

Equation 5.7: AE = AB

[29] Theorem of Bisection: https://www.varsitytutors.com/hotmath/hotmath_help/topics/triangle-angle-bisector-theorem

From equations 5.6 and 5.7 we have that:

Equation 5.8: ΔΒΔΓ = ΑΒΑΓ

So by using the equation 5.5 we can calculate the point D that divides the closest Polygon side and point the direction of the camera towards it in order to project the overall view of the Object of Interest.

```
SideAC = getDistanceFromLatLonInKm (A, position);
SideAB = getDistanceFromLatLonInKm (B, position);
SideBC = getDistanceFromLatLonInKm (B, C);

SideCD = (SideBC * SideAC) / (SideAC + SideAB);

var pointD = new google.maps.Marker({
           position: sides[d].GetPointAtDistance(SideCD),
           map: map});
```

- **Side View of the Object of Interest**

When a user moves within the middle of the Viewing Section we point the camera to the closest point of the Polygon from our current position with a small angle. That way we can get a side view of the Object Of interest.

Algorithm 5.13: *Algorithm for presenting a Polygon Object of Interest with a specific angle. Basic Steps.*

Step 1: Calculate the closest side of the Polygon from the user's location in the same way as Algorithm 5.12.

Step 2: Calculate the closest point on that side from the user's location.

Step 3: Point towards that point by calculating the angle between the point, the user's location and the True North. Pass this angle to the heading parameter. (See Figure 5.16)

5.12.3 Manually Defined Projection View Parameters on Object Presentation Section

As we mentioned in section 5.5, when an author constructs an Object Presentation Section for an Object of Interest he can define the orientation of the camera across its path manually. He defines the angles that consist the final Projection View during the navigation, the heading, the pitch and the field of view angles.

When navigating across an Object Presentation section, of an associate Object of Interest that matches the user's interests, the system checks for the type of the Camera Orientation. If the camera orientation is set to automatic mode, then the orientation is handled by the algorithms and methods we described in chapters 5.12.1 and 5.12.2. If the camera orientation is set to manual mode, the system retrieves the defined values for the Heading, Pitch and Field of View angles and use them to construct the Projection View of each point in this Viewing Section. The interface that handles this functionality is shown in Chapter 6.

5.13 Panning & Tilting Head Action

An Object Presentation Section can also offer a Panning or Tilting Head Action. When moving along a Viewing Section with a Panning Head action, the system first needs to turn the camera towards the starting angle. Afterwards, the rotation of the camera starts in order to present the Panning action. The rotation angle depends on the difference of the start and end angle and the number of the Projection Views. The system also has to take into consideration the type of the rotation.

$$\textbf{rotationAngle} = \frac{[endAngle - startAngle]}{numberOfProjectionViews}$$

The system implements the rotation by using the **setPov** function in the case of StreetView Panoramic Pictures and the **rotate** function of the PhotoSphereViewer in the case of the UserCaptured Panoramic Pictures. These functions are called using an interval, which is the rotation speed, by adding or subtracting the rotationAngle from the current heading of the navigation depending on the rotation type (clockwise or counter clockwise). The rotation speed depends on the duration of the associated narration of the panning action and the number of the projection views. This way the system reassures that the camera rotates along the narration being played.

$$\textbf{rotationSpeed} = \frac{narration\ Duration}{numberOfProjectionViews}$$

The **Tilting** head action is handled in a similar way.

Algorithm 5.14: *Algorithm for Panning Head Action. Basic Steps.*

Step 1: Calculate the appropriate rotationAngle and rotationSpeed of the Panning Action using the equations above.

Step 2: Rotate the heading angle of the current Projection View towards the start angle of the Panning Head Action, using the setPov () or the rotate () method.

Step 3: After pointing to the start angle, the rotation of the Panning action starts by making repeated calls to the setPov () (or rotate ()) method, increasing or decreasing the heading angle by the rotationAngle value. The repeated calls are handled by an Interval with time equal to the rotationSpeed. <u>Special cases</u>, like the case the start angle is greater than the end angle and

the set rotation is clockwise, are taken into consideration in order to provide the appropriate type of rotation defined by the author. During the rotation, the associated narration is played.
Step 4: After the Panning Action finishes its rotation, the system rotates the camera towards the direction of the road.

Pseudocode for Algorithm 5.14: Algorithm for the Panning Head Action

```
Var startPov = panningActionStartAngle;
Var endPov = panningActionEndAngle;
Var angleStep = (endAngle – startAngle) / projectionViews;
var interval = window.setInterval(function() {
                         var pov = currentPanorama.getPov();
                         startPov = Math.round(startPov + angleStep);
                         pov.heading = startPov;
                         pov.pitch = pitch;
                         currentPanorama.setPov(pov);
                         icon.rotation = startPov;
                         marker[0].setIcon(icon);
                         marker[0].setPosition(p);
                         if (startPov >= endPov){
                                 clearInterval(interval);
                                 startPov = endPov;
                         }
                 }, rotationSpeed);
```

5.14 Detection of UserCaptured Route Section and ObjectEnclosing Markers

Finally, we have to define how we detect if we are within the path of a UserCaptured Route Section Polyline or near the location of an ObjectEnclosing Marker, which identifies the location of the Panoramic Picture that holds an ObjectRectangle that identifies the Object of Interest. To detect if we are within a UserCaptured Route Section Polyline we use a similar algorithm as the algorithm 4.9. If we are moving on the path of a UserCaptured Polyline we start projecting the sequence of the UserCaptured Panoramic Pictures. We load the UserCaptured Panoramic Picture that is associated with the marker closest to the user's position on the path. We find the closest marker by using the algorithm 5.15.

Algorithm 5.15: Algorithm for Retrieval of the Closest Panoramic Picture in a UserCaptured Section. Basic Steps.

Step 1: The algorithm takes as input the user's location and the identifier of the UserCaptured Panoramic Pictures Route Section that he is moving along.

Step 2: It runs a for-loop for the number of the UserCaptured Panoramic Pictures that consist the UserCaptured Route Section.

Step 3: For each Panoramic Picture, it calculates the geographic distance between its location and the location of the user. At each step it compares the calculated distance with the current minimum distance. If it is less than the minimum, then it stores it as the new minimum distance.

Step 4: The algorithm returns the index, identifier of the UserCaptured Panoramic Picture, where the minimum distance was found.

Pseudocode for Algorithm 5.15:

```
/* Calculating the minimum distance from the Segment points
 * in order to detect if the current position is within its range
 * so we can direct the camera at the direction of the object of Interest
associated with it.*/

function rad(x) {return x*Math.PI/180;}
function find_closest_marker(position, panoramasegmentid) {
    var lat = position.lat();
    var lng = position.lng();
    var R = 6371; // radius of earth in km
    var distances = [];
    var closest = -1;
    for( i=0;i<panoramaPolylineMarkers[panoramasegmentid].length; i++ ) {
                if (panoramaPolylineMarkers[panoramasegmentid][i] != null &&
panoramaPolylineMarkers[panoramasegmentid][i] != 'undefined'){
                        var mlat =
panoramaPolylineMarkers[panoramasegmentid][i].position.lat();
                        var mlng =
panoramaPolylineMarkers[panoramasegmentid][i].position.lng();
                        var dLat  = rad(mlat - lat);
                        var dLong = rad(mlng - lng);
                        var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                                Math.cos(rad(lat)) * Math.cos(rad(lat)) *
Math.sin(dLong/2) * Math.sin(dLong/2);
                        var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
                        var d = R * c;
                        distances[i] = d;
                        if ( closest == -1 || (d < distances[closest] && d < 25))
{
                                closest = i;
                        }
                }
    }
        return closest;
}
```

After finding the appropriate UserCaptured Panoramic Picture, the system loads the source file to the PhotoSphereViewer constructor. The system calculates the appropriate Projection View by adding the heading angle, which is calculated with the algorithms we analyzed in the previous subsections, to the Heading North (HN) angle of the Panoramic Picture.

We need to define a way to detect a Custom Marker as we navigate across a Route. Upon retrieving the Custom Marker Objects from our database and setting them on the Map containing our route, portraying them as google.maps.Marker Objects, we associate a google.maps.Circle object around them with its center being the LatLng position of the Marker and its radius the value calculated in the construction steps. We check if the current position is within one of those circle objects by making use of the function containsLocation. This function is part of the google.maps.geometry library, it takes as inputs the LatLng Object of our current position and the Circle Object associated with either a custom Panorama or Marker. The functions returns true if the point is within the Circle or on its perimeter. If the condition is satisfied then we set the visibility of the html div that the PhotoSphereObject is going to load to visible and the visibility of the div that loads are default StreetView Imagery to hidden. We set the associated image file as the panorama of the PhotosphereViewer and load the panoramic image. In the case of the Custom Marker we also set the custom markers that are set on top of the panoramic image. We pause the navigation for enough time, giving the opportunity to the users to interact with the 360 image, with the set Markers which are clickable and contain relatable to the Object of Interest information. We then change the visibility property back to hidden and continue the sequential navigation across the default StreetViewPanorama Objects.

The projections of the UserCaptured Panoramic Pictures and ObjectEnclosingRectangle is done with the use of the Photo Sphere Viewer Object. In subchapter 5.7 we described the algorithms that calculate the pixel coordinates of an ObjectEnclosingRectangle. Those coordinates are stored in our database in the Custom Markers table in form of float attributes, two in the case of a simple icon Marker and eight (two per edge) for a Polygon Marker. Upon navigating a route either in an automated way or in interactive way those pixel coordinates are retrieved with GET Rest requests and used in the Marker Objects of the Photo Sphere Viewer.

A Circle Object is similar to a Polygon Object in that you can define custom colors, weights, and opacities for the edge of the circle and custom colors and opacities for the area within the circle. Unlike a Polygon, you do not define paths for a Circle. Instead, a circle has two additional properties which define its shape:

- Center, specifies the google.maps.LatLng of the center of the circle.
- Radius, specifies the radius of the circle, in meters.

5.15 Info Window Visibility in StreetViewPanorama

In the subchapters 5.4.1 and 5.4.2 we described how we deploy the **Objects of Interest** of each **Route** in the **Map Object** depending on the type we chose to represent them. The type of each Object of Interest determines the algorithm that calculates and handles the rotation of the camera. The algorithms that handle the direction of the camera are described at the subchapters 5.10.1 and 5.10.2. Besides, identifying the Points of Interest on the Map we also have to deploy them on the **StreetView Panorama Object**.

The StreetView Service supports only three types of Overlays: **Markers**, **Info Windows** and **Custom Overlays**. Both types, Single and Polygon Object of Interest, are represented within our Application by using a combination of Marker Objects and Info Windows. In the case of a Single Object of Interest we place the Marker on the LatLng position of the Object of Interest. In the case of a Polygon Object of Interest, the Point is constructed by an array of LatLng points, so in order to set the Marker on the Panorama Object we have to find the center of the Polygon Object. We use the method of polygons.my_getBounds ().getCenter () which returns the LatLng coordinates of the Polygon's center. In order to set each marker, we first calculate the minimum Distance of the Point from the Route. This minimum Distance defines the size of the Icon we use to represent the Marker, the farthest from the Route the bigger the icon we use. Upon setting each Marker we associated him with an Info Window Object. The content of this Info Window contains an image file representing the Point, the name of the Point, a short Description and final a button that opens up the pop up window presenting all the Point's information.

Each created Route may contain many Points of Interest. By default, the set Markers on a StreetView Panorama Object are all visible from each point. By viewing all the Markers, the user can get confused which may lead him to an error. To resolve this issue we make each Marker visible only when approaching the corresponding Point. As we analyzed on subchapter 4.10 the detection of each Object of Interest is done by the Viewing Segments associated with it. Therefore, we make the Marker visible upon entering a Viewing Section of the Object of Interest.

Finally, as we have mentioned in the Requirements Analysis we give the users the ability to add Object of Interest at any distance from the Route, as long as they are visible within each Points. The Info Window Overlay by default opens the Window box only above the position of the set Marker. If the Object of Interest is located in a far distance from the Route, the Info Window Object will not be visible. To resolve this issue we set a Polyline Object, containing our current Position and the Position of the Point. Depending on the length of the Polyline, the distance between the two Points, we either open the Info Window on the default Position or in the middle Position of the Polyline, making it that way visible to the user. The distance between the Object of Interest Street View Marker and the Route is calculated by using the Algorithm **5.9**.

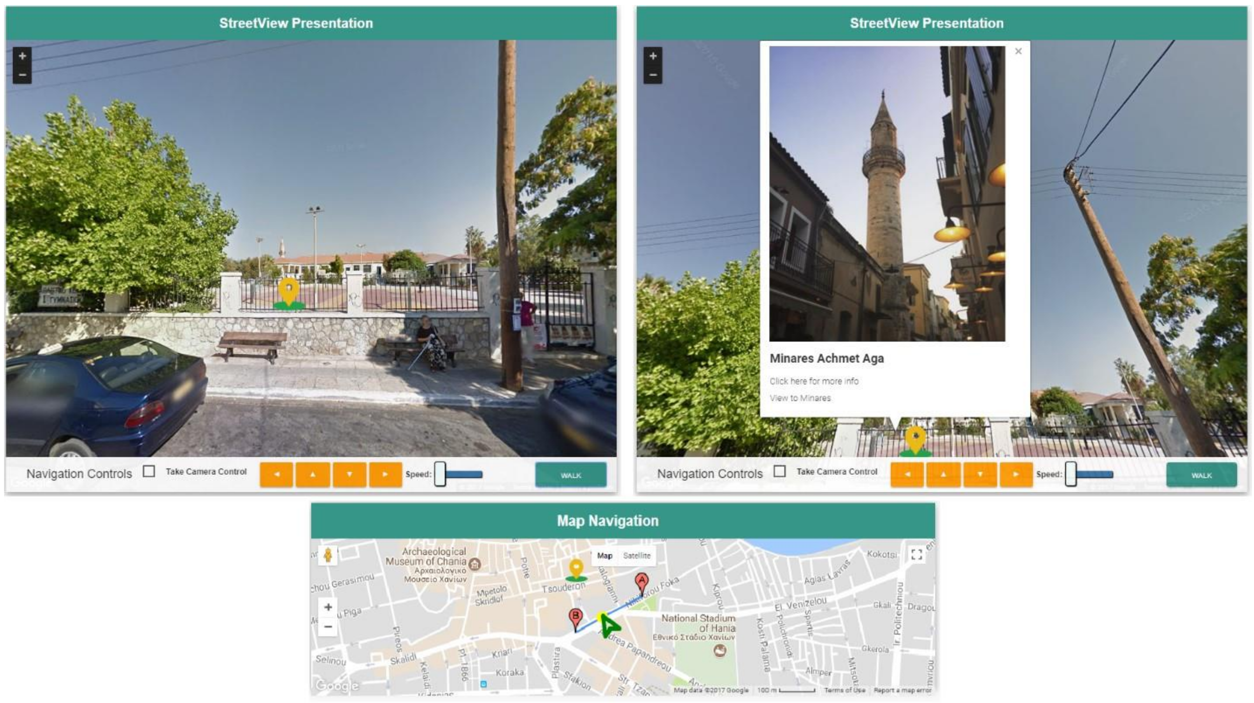


Figure 5.20 Example of Object of Interest in far Distance from the Route (Minares Achmet Aga).

5.16 Retrieval of StreetView Panoramic Pictures for Dynamic Time Lapse Video Generation

In order to generate the video presentation of the Route we need to retrieve the appropriate StreetView Panoramic Pictures that correspond to the LatLng position of the user's movement. This retrieval is handled by making a StreetView request through the getPanorama method. The getPanorama method receives a location, a radius, a set of optional preferences and a callback function. In this set of preferences we can define if we want the best or the nearest to the location available panoramic picture. StreetView documentation states that by passing a 50 meter radius and a preference of the Nearest Panorama, the StreetView request returns most of the times a response containing the panoramic picture that is closest to the location of the request. By testing this we noticed that quite often the response contains a Panoramic Picture that is not associated with the nearest available location that has StreetView coverage. In order to reassure that we always get the nearest available StreetView Panoramic Picture we edited the Callback function that processes the StreetViewPanorama Object response. First we make a call to the StreetView request with the default radius of 50 meters. In the callback function we take the location of the StreetViewPanorama Object response and we calculate the distance between this and the location of the request. We make a second StreetView request by passing this calculated distance as the new radius, thus reassuring that we will receive the nearest available Panoramic Picture. It should also be noted that we define a preference to receive only outdoor StreetView Panoramic Pictures.

Algorithm 5.16: Algorithm for Retrieval of the Closest Panoramic Picture in a StreetView Section. Basic Steps.

Step 1: Making a request to the StreetView Service passing the currentPosition and a fixed radius.

Step 2: If a StreetViewPanorama Object is found within the given radius, we retrieve the location of the found StreetViewPanorama Object and calculate the distance between this point and the location of the request. We use this distance as a radius to make a new StreetView request in order to obtain the StreetViewPanorama Object in the closest distance.

Step 3: If a StreetViewPanorama Object is not found within the initial radius, we make a new StreetView request increasing the fixed radius, till we obtain a StreetViewPanorama Object.

Algorithm 5.16: Algorithm for handling StreetView requests

```
sv.getPanorama ({location: currentPosition, radius: 50, preference:
google.maps.StreetViewPreference.NEAREST, source:
google.maps.StreetViewSource.OUTDOOR}, processSVData);

Function processSVData (data, status) {
        If (status === google.maps.StreetViewStatus.OK) {
                Var distance = getDistanceFromLatLonInKm (currentPosition,
data.location.latLng);
                sv.getPanorama ({location: currentPosition, radius: distance,
preference: google.maps.StreetViewPreference.NEAREST, source:
google.maps.StreetViewSource.OUTDOOR}, processSVData2);
        } else {
          radius = radius+200;
          sv.getPanorama ({location: currentPosition, radius: radius, preference:
google.maps.StreetViewPreference.NEAREST, source:
google.maps.StreetViewSource.OUTDOOR}, processSVData);
        }
}

Function processSVData2(data, status) {
        if (status === google.maps.StreetViewStatus.OK) {
                panorama.setVisible(true);
                if (panoIDPanorama1 == data.location.pano){
                        panorama.setVisible(true);
                        panorama.setPov({
                                heading: headingPov,
                                zoom: zoom,
                                pitch: pitch,
                        });
                }else{
                        panorama.setVisible(true);
                        panorama.setPano(data.location.pano);
                        panorama.setPov({
                                heading: headingPov,
                                zoom: zoom,
                                pitch: pitch,
                        });
                }
                panoIDPanorama1  = data.location.pano;
        }else {
                streetDistance = streetDistance+45;
                sv.getPanorama({location: currentPosition, radius: streetDistance,
preference: google.maps.StreetViewPreference.NEAREST, source:
google.maps.StreetViewSource.OUTDOOR}, processSVData2);
        }
}
```

5.17 Navigation using a Mobile Device

The system also supports the navigation of an end user along a Route using his mobile device. In this functionality the system records the movement of the user on the map, as he walks (or drives) along a route, by accessing the GPS mechanism of his mobile device. To achieve this, the system makes use of the geolocation API and the **watchPosition**() method. This method is used to register a handler function that will be called automatically each time the position of the device changes. You can also, optionally, specify an error handling callback function.

We animate the position of the user with the retrieved GPS position. The algorithms and methods used for the detection of the Object of Interest and UserCaptured Panoramic Pictures Route Section are the same as the functionalities for the web interface player.

Chapter 6

6 User Interface and Evaluation

6.1 Introduction

In this chapter we are going to present the Use cases of our application and give a short description about the task they represent and the type of actor that corresponds to it. We are also going to describe the principles we follow through our design process aiming to make our application robust and user friendly.

6.2 Use Cases

In the following table we are going to give a short description for the tasks related both to an Author and a user of the application.

#	Actor	Action	Description
UC1	User / Author	Login to application	User logs in to the application.
UC2	User / Author	Logout	Users exits the application.
UC3	User / Author	Register	User signs up to the application.
UC4	User	View a Planned Route	User picks to navigate an automated route.
UC5	User	View an Object Of Interest	User clicks on an Object of Interest to view its information.
UC6	User	View a Section	User clicks on the Section's polyline to view its information.
UC7	User	Adds a comment to an Object of Interest	User submits a comment related to the Object of Interest he is viewing.
UC8	User	Search Routes	User searches for routes by submitting keywords.
UC9	User	Mobile Navigation	User navigates a route with his mobile device.
UC10	Author	Create Route	Setting up a new path on which he might add Points of Interest.

UC11	Author	Create Object of Interest	Author adds an Object of Interest alongside his submitted route.
UC12	Author	Create Object Presentation Sections	Add sections associated with each Object of Interest.
UC13	Author	Create Custom Panorama	Add a custom Panorama on the route's path.
U14	Author	Create Custom Marker	Add a custom marker on a section's polyline.
U15	Author	Edit Route	Edits a route and all the objects associated with it.

Table 6.1 Use Cases of the System.

Detailed Description:

Before describing the use cases listed in the table Figure 6.1 we have to define the two types of users in the system. **Authors** are the users that create the Immersive and Guided Tours of the application. Through the system they define the path of the Route and the Objects of Interest that are located alongside it, either at a close or a far distance. They can be professional tour guides or locals with great knowledge of the surrounding of their city. **End Users** are the users that navigate those Tours either by viewing an automated generated video through their browser on their desktop pc or by walking down the path of a Route using their mobile device.

Use Case 1: Login to the Application (User /Author)

The users fills in the required login information, his email and his password and if they are correct he enters the application. If the input data are incorrect his access is denied and he gets notified with the appropriated message.

Use Case 2: Logout of the Application (User /Author)

The user can click on the logout button from every page within the application and exit the system. When clicking the button a message pops up requesting his confirmation. Upon his exit his session gets terminated and the cookies associated with his username and avatar are cleared.

Use Case 3: Register in the Application (User /Author)

When a user wishes to register in the application, he has to fill the required fields of the registration form. If the input data are correct he is redirected to the login page where he can enter the email and password he used to login. If the input data are incorrect the appropriate messages are appearing after the fields he filled incorrectly.

Use Case 4: View a planned Route (User)

The user navigate to the Planned Routes page form the home page of the application. There he is introduced to all the routes other users have created. The user can click on the image of each route and read some short description about it. When he wants to view an automated route he clicks on the View Route button located on each planned Route square.

Use Case 5: View an Object of Interest (User)

Within the main page that handles the projection and navigation across a selected route, a user can view an object of interest with two ways. Either by clicking on its corresponding marker on the map or Street view panorama or by clicking on the side panel that contains the list of all the Points of Interest included in the route.

Use Case 6: View a Section (User)

Within the main page that handles the projection and navigation across a selected route, a user can view a section associated with an Object of Interest by clicking on the polyline that represents it. Clicking on the polyline triggers an event listener that opens an info window which contains a short description and a link to open the popup div which displays the fill information.

Use Case 7: Add a comment (User)

When the user is viewing an Object of Interest included the route he selected to navigate, he can add a comment on the pop up interface that shows the Point's information. He has to be logged in order to be able to submit a comment.

Use Case 8: Search Routes (User)

Within the Planned Routes page there is a search bar where a user can type keywords in order to look for Routes that match them. The user can add any number of keywords as long as he separates them with a comma separator. The routes returned have points of interest across their path that match those keywords.

Use Case 9: Mobile Navigation (User)

Besides viewing automated routes, a user can also navigate a created route by using his mobile device. He must walk across the path of the route. Much like the automated routes, the user will get notified upon approaching points of interest.

Use Case 10: Create Route (Author)

A user can create a new route and add it to the application through the Route Design page. He can get to this page through the Home page by picking the middle option of the start page. This will lead him to the interface containing the necessary form and map object on which he will add the route's path.

Use Case 11: Create Object of Interest (Author)

A user can add an Object of Interest after submitting a new route. He is introduced to a new interface containing a map object on which the polyline of the route is set. On the bottom of the map object there is a div containing the option of adding a new Object of Interest. Clicking on that option sets a listener event on the map allowing the user to interact with it and add the point on his chosen position. Upon clicking on the map the pop up interface containing the form for the Objet of Interest is presented to the user.

Use Case 12: Create Object Presentation Section(s) (Author)

Within the pop up interface, that we mentioned above, regarding the Object of Interest, there is a map object with event listeners for creating polyline objects that will represent the sections of the route from where the associated Point is visible. After the user adds the points he wants to include within the section's polyline, he clicks on the Save button located on the bottom bar of the map. This will open the interface where he will submit the needed information about the section such as name, imagefile, etc.

Use Case 13: Create Custom Panorama Path (Author)

A user can add a custom Panorama after submitting a new route. He is introduced to a new interface containing a map object on which the polyline of the route is set. On the bottom of the map object there is a div containing the option of adding a new Panorama object. Clicking on that option sets a listener event on the map allowing the user to interact with it and define the path of the route on which he wants to add UserCaptured Panoramic Pictures. This path may already have available StreetView Panoramic Pictures or not. Upon defining the path, he is introduced with an interface which requires him to upload the UserCaptured Panoramic Pictures. Each of these Panoramic Pictures must have a direction of the road of the movement. Each UserCaptured Panoramic Picture is represented with a Google Maps Marker placed on the GPS position retrieved by the EXIF data of the picture.

Use Case 14: Create Custom Marker (Author)

A user can add a Custom Marker after submitting a new Presentation Section. Its section has a click event that opens an info window containing options such as adding a new Custom Marker. The new custom marker is added on the point of the click event on the polyline. Much like the Object of Interest and Panorama cases, a pop up interface is presented to the user with a form and the options of using a default street view image or an uploaded one, on top of which he is going to add the marker object in a form of either a simple pin or a polygon (4 edges).

Use Case 15: Edit created Route (Author)

If a user wants to edit an existing route he must navigate to the Planned Routes page from the Home page. Each route is represent in form of a card div. Each card div contains the options of either viewing or editing the route. Upon choosing to edit the route, the user is lead to the Edit Route page. There he is introduced to an interface similar to the Route Design page. He can add new Points of Interest, Panorama or edit the path of the route, extending or minimizing it. He can also edit the existing points of interest or panoramas, by clicking on the marker objects that represent them on the map.

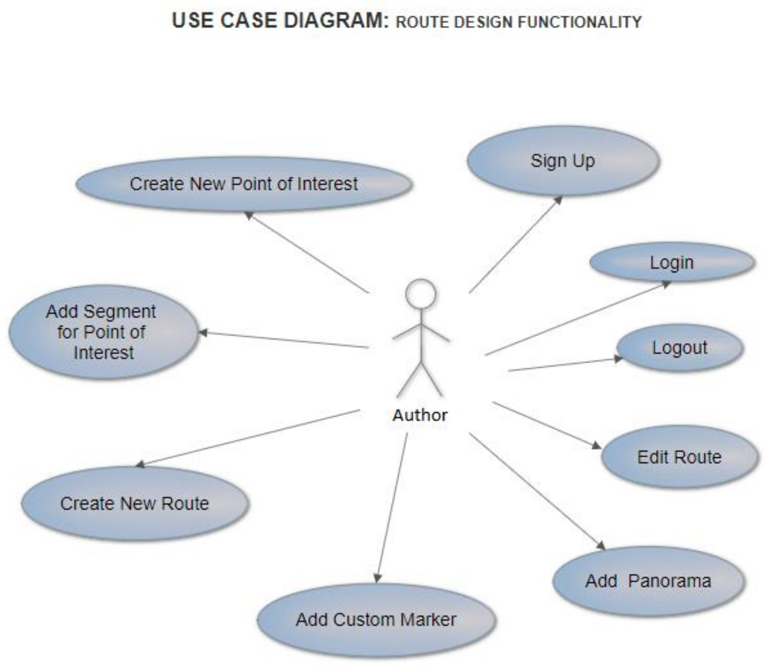

USE CASE DIAGRAM: ROUTE DESIGN FUNCTIONALITY
Create New Point of Interest
Sign Up
Login
Add Segment for Point of Interest
Logout
Author
Edit Route
Create New Route
Add Panorama
Add Custom Marker

Figure 6.1 Use Case Diagram for Author's Actions.

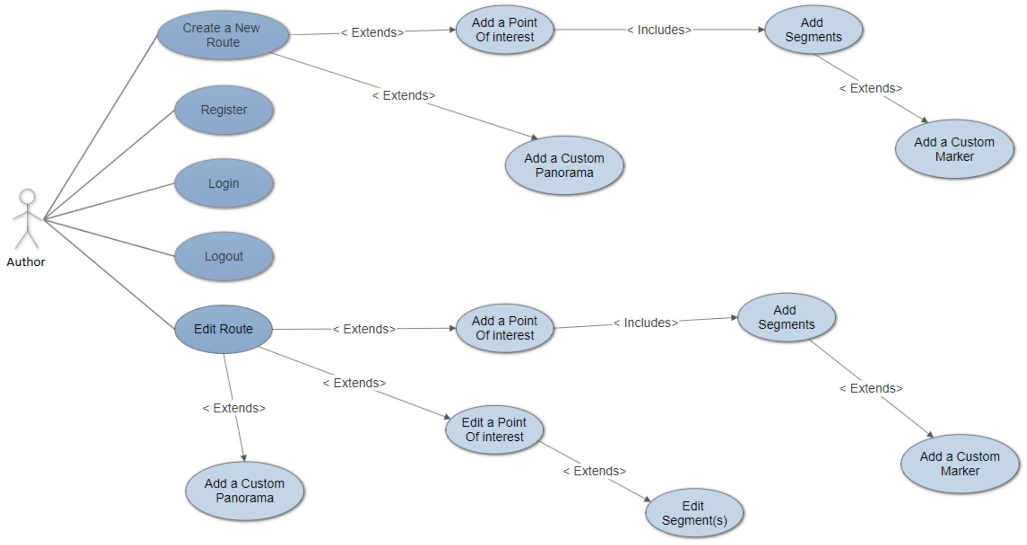


Figure 6.2 Detailed Use Case Diagram for Author's Actions.

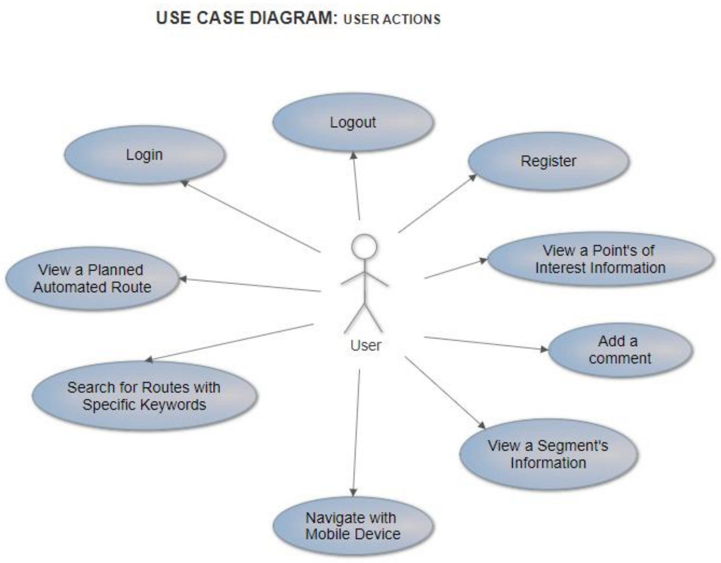
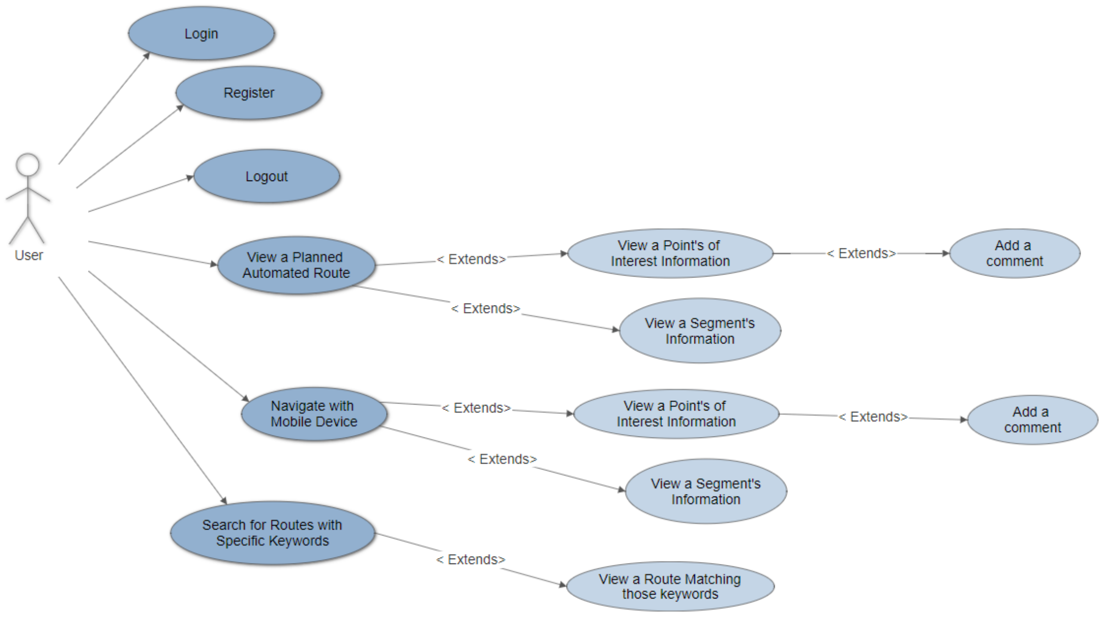

USE CASE DIAGRAM: USER ACTIONS
Login
Logout
Register
View a Planned Automated Route
View a Point's of Interest Information
User
Search for Routes with Specific Keywords
Add a comment
View a Segment's Information
Navigate with Mobile Device

Figure 6.3 Use Case Diagram for User's actions



Login
Register
Logout
User
View a Planned Automated Route
< Extends>
View a Point's of Interest Information
< Extends>
Add a comment
< Extends>
View a Segment's Information
Navigate with Mobile Device
< Extends>
View a Point's of Interest Information
< Extends>
Add a comment
< Extends>
View a Segment's Information
Search for Routes with Specific Keywords
< Extends>
View a Route Matching those keywords

Figure 6.4 Detail Use Case Diagram for User's Actions.

6.3 Principles of User Interface Design

In this subchapter we are going to give a short description for the basic principles we followed to design our application.

6.3.1 Ease of Navigation

One of the crucial parts of building a web application is to offer an easy navigation across its functionalities to the final user. We want the user to be able to reach to his final goal within a few clicks, without getting confused or lost. To achieve this we have to minimize the number of clicks he has to do in order to complete each of his actions on the application's pages. We also have to offer the user access to all the actions associated with each page. For example in our index page we offer him the options to view the planned routes on the automated mode for the desktop part, the option to create a new route and add it in our application and finally the option to use our application through his mobile phone, navigating the routes using the mobile's compass. Likewise, in the Route Design page we have included an easy access to all the functions related to adding a new Object Of Interest, to edit the constructed route Path and to add a new Custom Panorama. We also offer access back to the index page from every other page of the application by clicking on the logo icon of our application.

In the footer navigation bar we offer access to the main pages of our application (Planned Routes, Route Design) so the user does not have to get back to the home page to access them.

6.3.2 Consistency

The consistency in the application's interface is very important when developing an application. By keeping a consistent format in our pages we make the navigation among them easier for the user. The main reason is that it gives the feeling of stability gaining the user's trust. It also helps the users memorize the steps of every action making the use of the application easier every time. This happens because the user associates in his mind each action with certain colors, procedures and positions of buttons or forms in the interface. For example, by using other websites and applications users have become accustomed of a button on the top left corner of the page that lead back to the home page so this is going to be one of the first things he is going to look for in each of our pages.

6.3.3 Colors within our application

The basic colors we used across our application was white, green and orange. We decided to use those colors for the reasons we are going to analyze in detail. First of all the white color we chose as our main background is a neutral color which offers clarity and good contrast with the html elements we are going to place on top of it. The green and orange colors we used in the divs and forms of our pages have a far enough distance between them in the color wheel, offering good contrast. We also used those colors because the emotions they promote much the characteristics of our application. The orange color promotes friendliness, energy and creates a sensation of movement which matches well with our interactive tours application. Likewise, the green color expresses stability, growth and being the color that britches the gap between the cool and warm colors, it offers the same relaxing effects with the blue, but also retains some of the energetic qualities of the yellow color. For that reason we chose green as the color of our navigation bar and as the color of the title bar in all the divs and pop up interfaces in our pages. As we mentioned above we also used red color to point our error messages, since it is a color that expresses importance and draws the attention of the user.

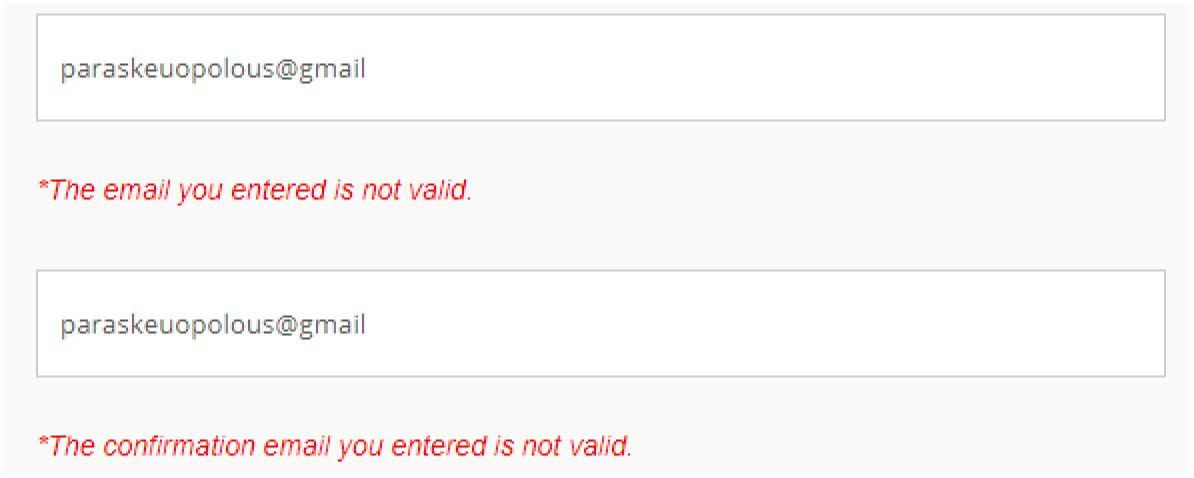
6.3.4 Typography

The fonts we used within our application we mainly used were ***Roboto*** and ***Sans Serif***. We decided to use those fonts since they provide good readability due their simplicity and clarity. Since they are easier to read, they are more relaxing to the users for large texts such as the descriptions of routes or points of interest.

6.3.5 Error Prevention and Handling

Our goal was to make an application that would minimize the possibility of errors when even a naïve user navigates across its pages. We tried to make the options and functions in each page, distinct and clear so that the user does not get confused. Although as in every application we cannot avoid the errors of a user.

In the case of an error we must show the user an appropriate message that will help him recover it. Within our application we handle the case of an error by displaying the appropriate message on the interface on which the user made the error. For example if a user makes an error when trying to input the email to complete his registration to the page, we alert him by adding an error message under the input box of the email. We use a red color in the error message text in order to get the user's attention and help him with a detailed message to recover from it and finish his action. We use the similar method to introduce the errors that the user might fall into. That way we also keep consistency among our pages, which as is also a crucial part of an application as we are going to analyze in the chapter 6.3.2. We have included an example of an error handling below:


paraskeuopolous@gmail

*The email you entered is not valid.

paraskeuopolous@gmail

*The confirmation email you entered is not valid.

Figure 6.5 Example of an error message when a user fills in the registration form.

6.3.6 Screen Sizes

Our application was mainly developed for desktop-side but we also designed some pages and functionalities that can be handled within a user's mobile device. One of the main issues when designing an application with mobile functionality are the variety of screen sizes that it has to support. In contrast with the desktop and laptop computers where the dimensions are for the most part constant and the proportions are roughly the same, the mobile platform has a huge variety both in resolutions and in screen ratios.

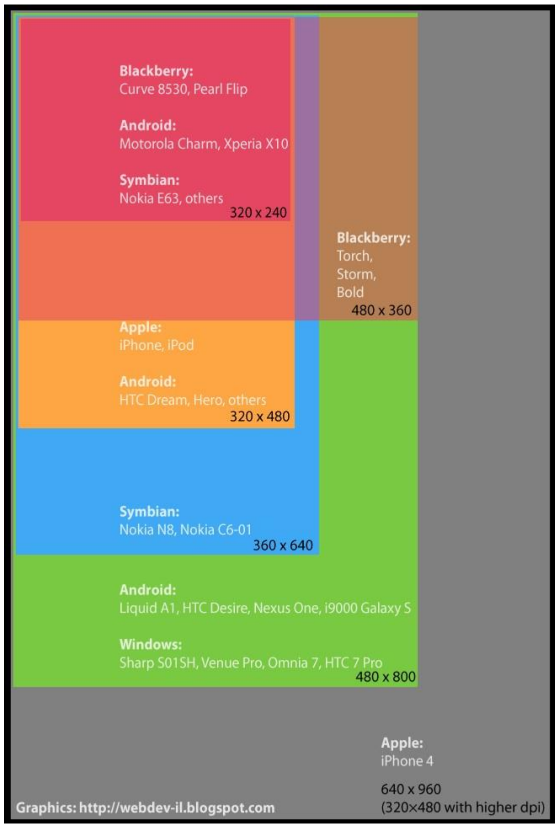


Figure 6.6 Difference in Screen Sizes[30].

We have to adjust our content depending on the screen size and resolution on which our pages are loaded. To achieve this we used css and JavaScript in order to resize the html elements of each page and adapt their width and height to each screen.

As we mentioned earlier our main interface is consisted by square div elements providing the information and the functionality of each page. In figure 6.8 we can see an example of our Index page in a desktop interface (min-width: 750px). As we can see our three main options are presented in a horizontal alignment next to each other, thus taking advantage of the width of the computer screen. Most mobile devices are used in portrait mode, which leads the manufacturers designing elongated screens. An elongated screen, meaning the height dimension is bigger than the width, the opposite of the desktop monitors and laptop screens. Therefore we have to change the horizontal alignment to a vertical one, based on the Vertical Design model. By changing the css attributes of our main divs we manage to align them under each other. We have included an examples of how we rearrange the divs in our interface to adjust to the mobile devices, see figures 6.7.

[30] Image obtained by: https://i.pinimg.com/originals/2e/b8/22/2eb822a4ad889d6e7a490fedb33f3fa9.png

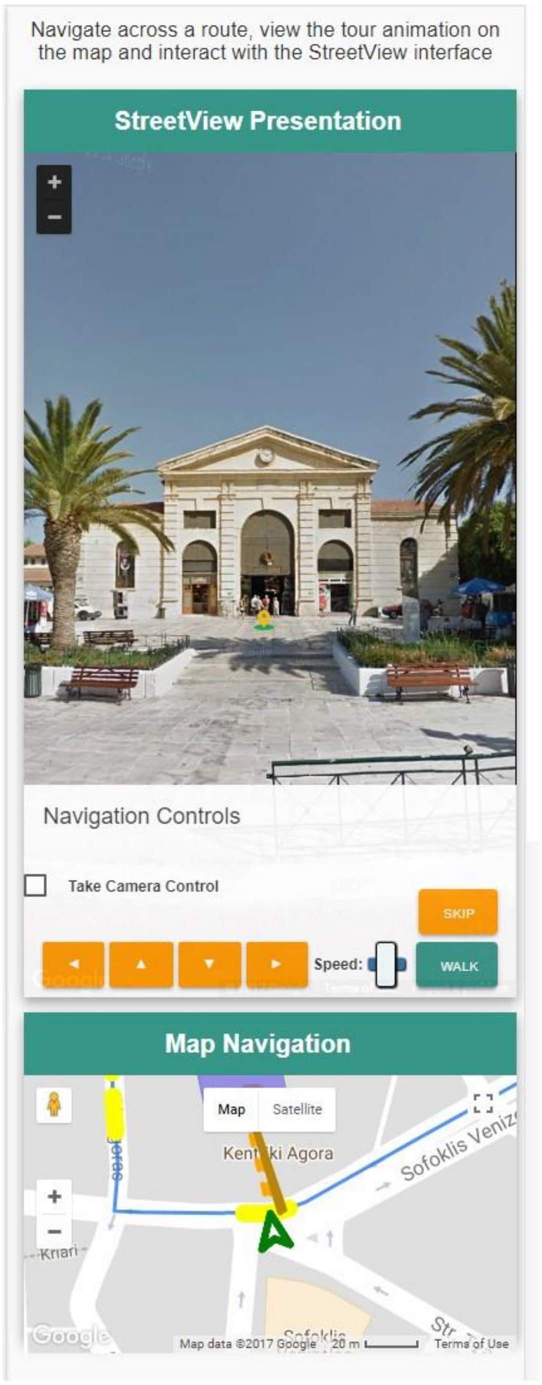


Figure 6.7. Layout of the Interface of the Navigation Page in Mobile Device (shown in Nexus 6p). The alignment of the StreetView and Map Divs is vertical in this case, taking advantage of the screen height.

6.4 User Interface

In this section we are going to list the basic elements of the interface of our application. We are going to provide screen captures to present the functionality provided by our system.

The basic elements of our interface:

1. ***Navigation bar***: Located at the top of every page. It contains two buttons. One if the form of an icon, the logo of our application that navigates the user back to the home page and one button that is responsible for the login or logout functionalities. When the user is logged in to the application it also displays the username and his avatar picture (profile), next to the logout button.

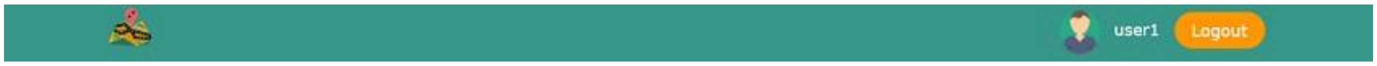


Figure 6.8 Navigation Bar. At the top of each page within the application.

2. ***Footer bar***: Located at the bottom of each page. It contains links for the main page of our application such as the Home Page, the Planned Routes and the Route Design. That way the user does not need to scroll all the way up if he need to navigate to another page.

3. ***Main Body***: the main body of our interface has a white background and it contains square

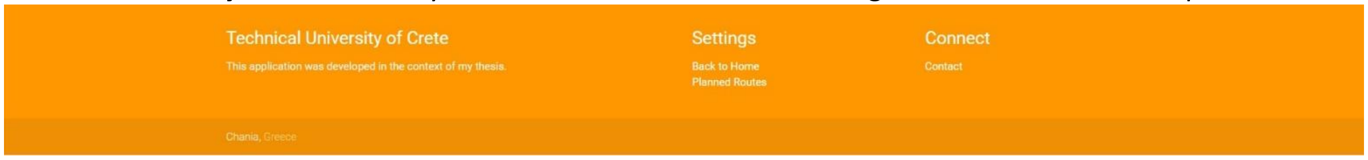
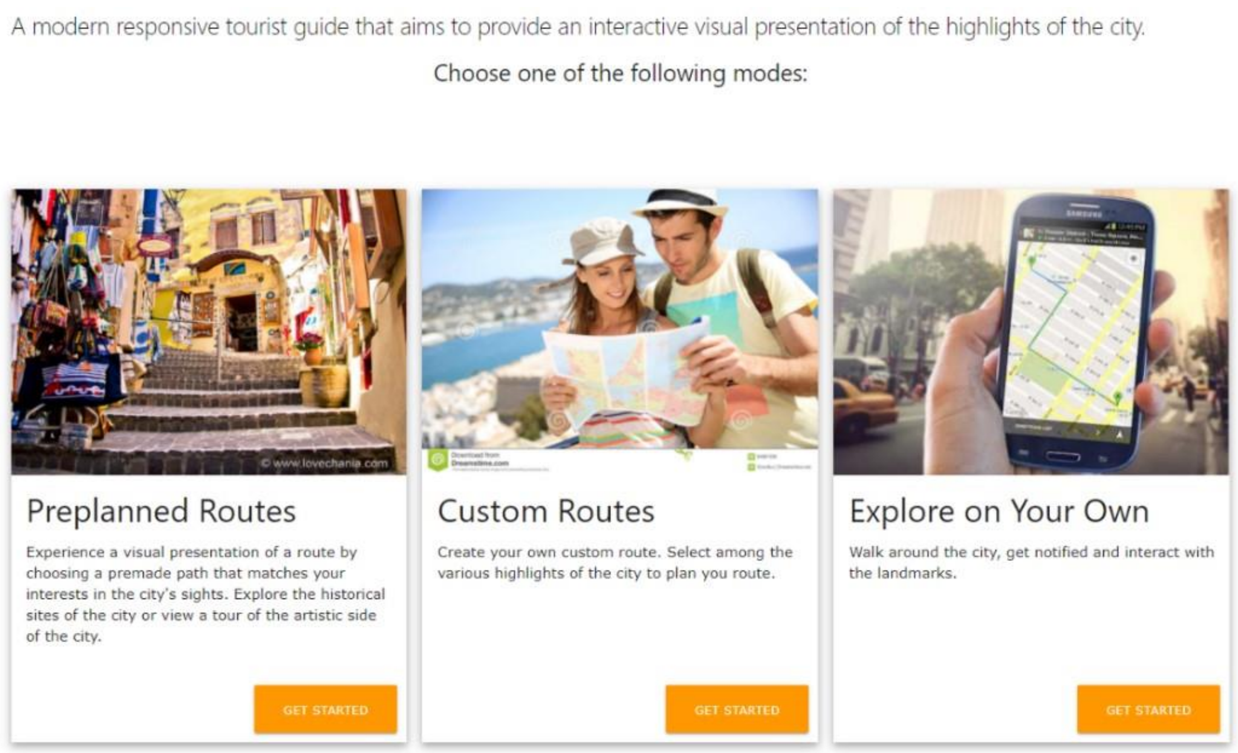

Technical University of Crete
This application was developed in the context of my thesis.
Settings
Back to Home
Planned Routes
Connect
Contact
Chania, Greece

Figure 6.9 Footer bar, located at the bottom of each page within the application.

divs that hold the information and the functionalities of our application. We decided to follow this design since we used three divs in our main page to display the information related to the route, the street view projection and the navigation across the map. In order to maintain the consistency within our interface we decided to present all the actions in a similar way. Therefore, the home page introduces the user to the three key functionalities of our application, the automated routes, the design of routes and the mobile navigation with three div squares (see figure 5.11).



A modern responsive tourist guide that aims to provide an interactive visual presentation of the highlights of the city.

Choose one of the following modes:

Preplanned Routes

Experience a visual presentation of a route by choosing a premade path that matches your interests in the city's sights. Explore the historical sites of the city or view a tour of the artistic side of the city.

GET STARTED

Custom Routes

Create your own custom route. Select among the various highlights of the city to plan you route.

GET STARTED

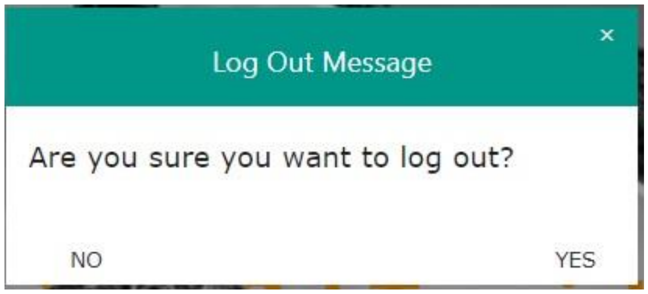
Explore on Your Own

Walk around the city, get notified and interact with the landmarks.

GET STARTED

Figure 6.10 Example of the Main Body of Our Application. The basic elements of each page are represent with square divs - cards.

4. ***Buttons***: we used three types of button within our application. The buttons that were responsible for navigating to the main functionalities of our application are square and orange and located at the bottom right corner of the div that contains them (see Figure 6.13). They have a big width and height so there easily noticeable by the users. For the forms of our interface we used a tag elements that represented the cancel and save buttons (cancel - confirm), we did that to make our interface similar to other applications the users may use, making it that way easier to understand and use (see Figure 6.12).



Log Out Message

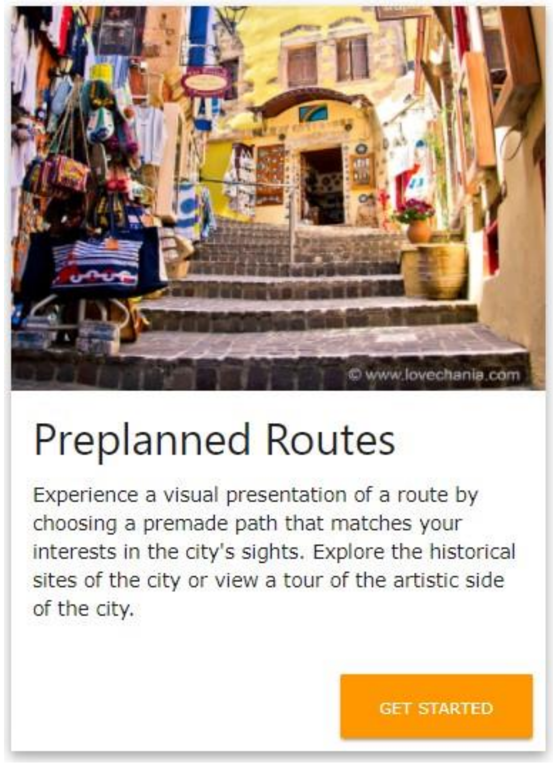
×

Are you sure you want to log out?

NO

YES

Figure 6.11 Example of tag Buttons in the Forms of the Interface.



Preplanned Routes

Experience a visual presentation of a route by choosing a premade path that matches your interests in the city's sights. Explore the historical sites of the city or view a tour of the artistic side of the city.

Figure 6.12 Example of buttons in the main elements of the Home Page interface.

Finally, some of our functionalities offer the users more than one option for his action. For example when creating a Route, the user can add Points of Interest or Panoramas. As we explain further in the Application Functionality and Algorithms chapter, we associate those options with event listeners on the Map Object. We represent these options by using Radio Buttons (see Figure 5.14). When a Radio Button is checked, that means its corresponding Event Listener is active and the rest are cleared.

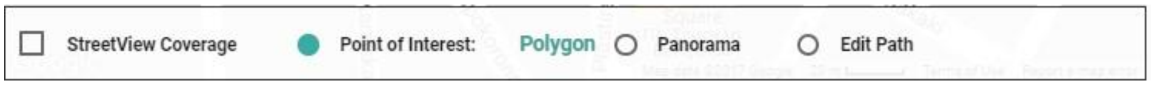
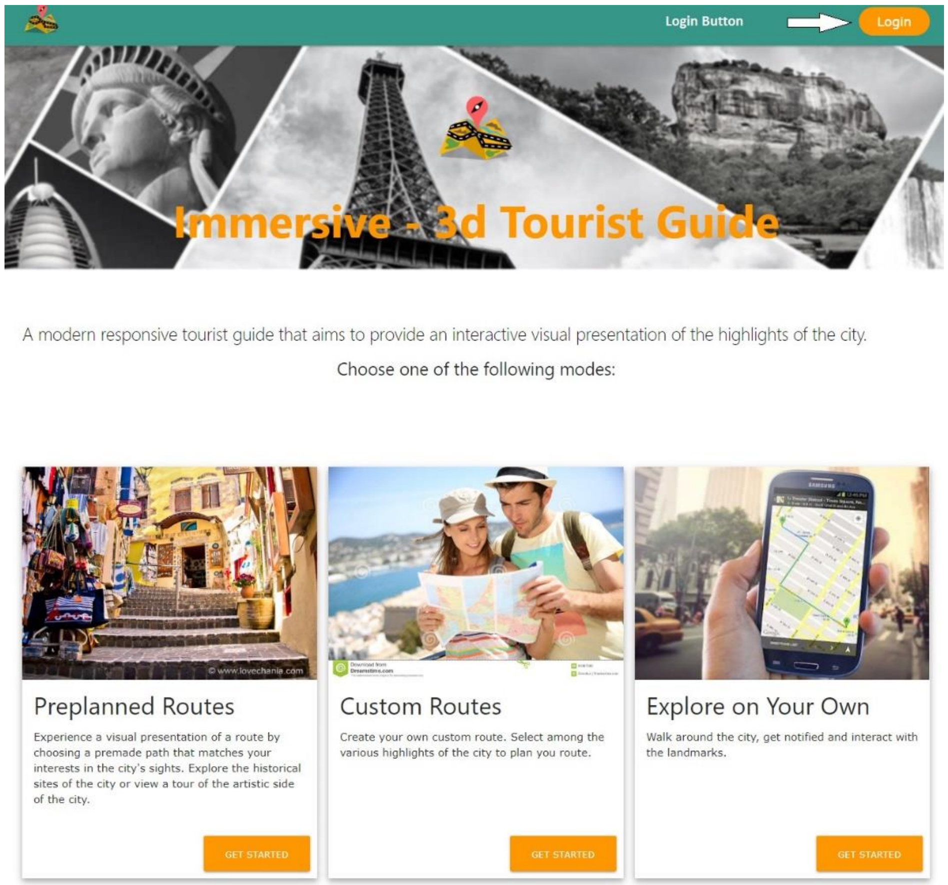


Figure 6.13 Example of radio buttons in the interface of the Create Route page. These radio buttons enable the event listeners on the map Object for the construction of the Route related Objects.

6.4.1 Login to the application



A modern responsive tourist guide that aims to provide an interactive visual presentation of the highlights of the city.

Choose one of the following modes:

Preplanned Routes

Experience a visual presentation of a route by choosing a premade path that matches your interests in the city's sights. Explore the historical sites of the city or view a tour of the artistic side of the city.

GET STARTED

Custom Routes

Create your own custom route. Select among the various highlights of the city to plan you route.

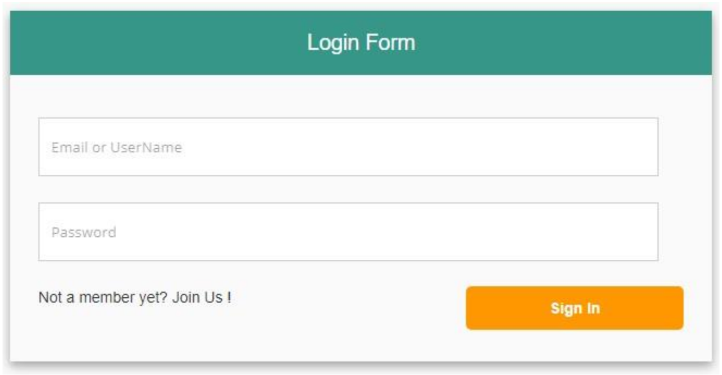
GET STARTED

Explore on Your Own

Walk around the city, get notified and interact with the landmarks.

GET STARTED

Figure 6.14 Location of the Login Button. It is located in the top right corner of every page.



Login Form

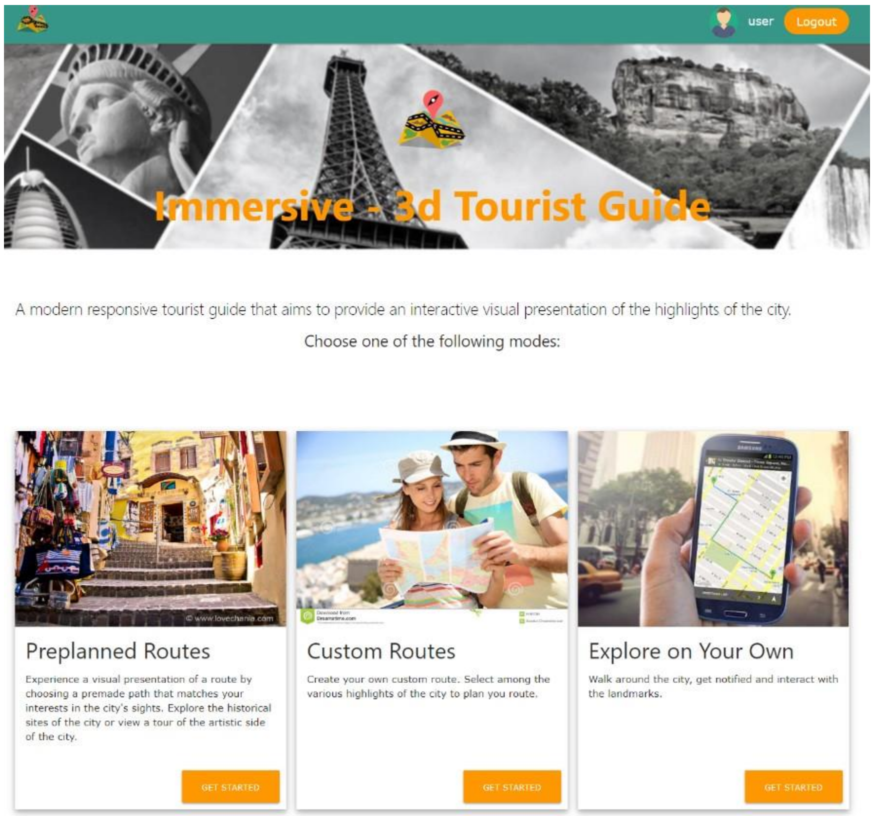
Email or UserName

Password

Not a member yet? Join Us !

Sign In

Figure 6.15 After clicking on the Login page the user is redirected to the Login page to fill in the form with his info. If he is not resisted he is redirected to the Register page with a similar form to sign up.



Immersive - 3d Tourist Guide

A modern responsive tourist guide that aims to provide an interactive visual presentation of the highlights of the city.

Choose one of the following modes:

Preplanned Routes

Experience a visual presentation of a route by choosing a premade path that matches your interests in the city's sights. Explore the historical sites of the city or view a tour of the artistic side of the city.

Custom Routes

Create your own custom route. Select among the various highlights of the city to plan you route.

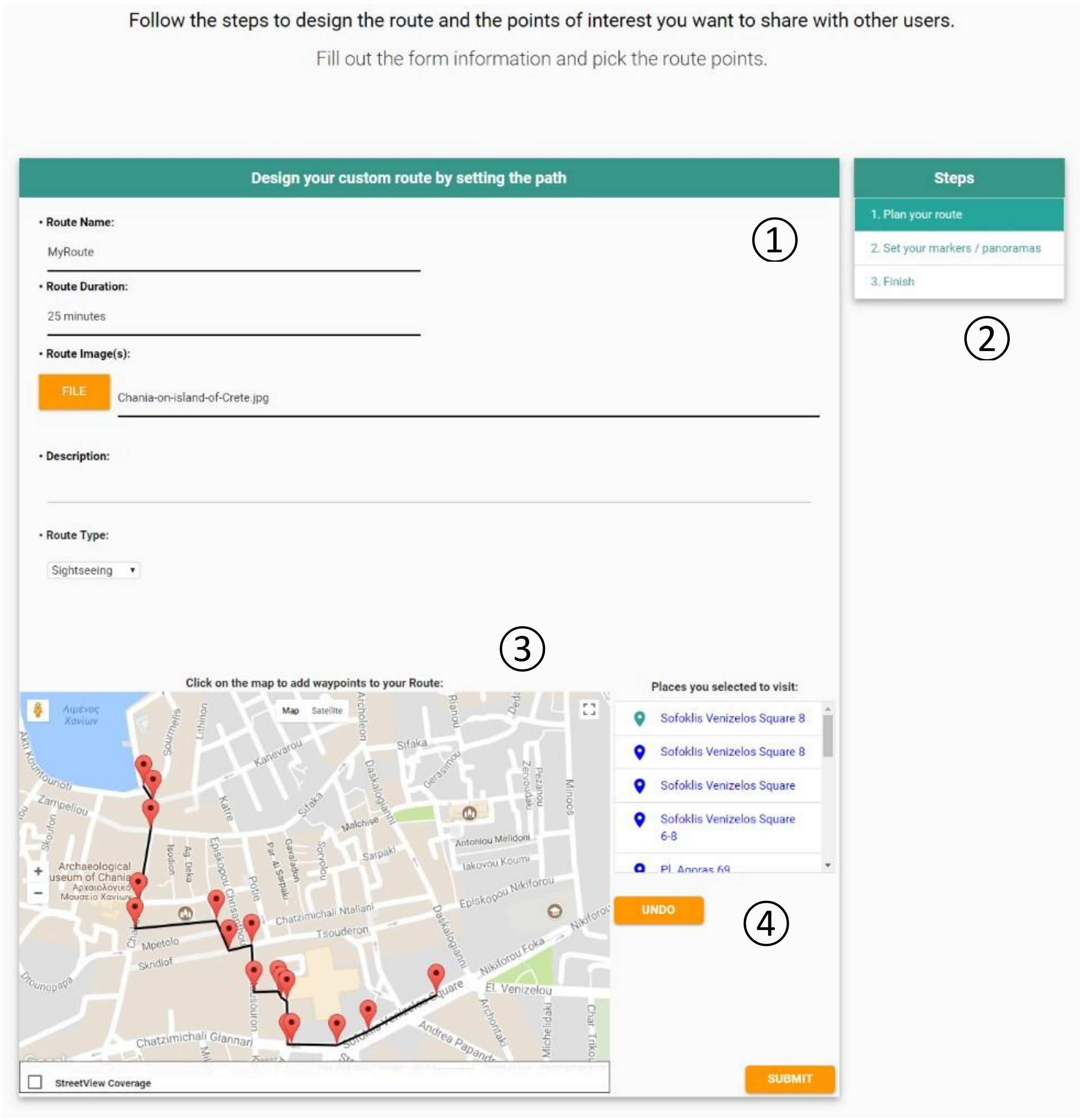
Explore on Your Own

Walk around the city, get notified and interact with the landmarks.

Figure 6.16 After a successful Login in the application, the user is redirected to the Home page. The user's nickname is displayed on the top right corner aside the Logout button.

6.4.2 Create a New Route

The user navigates to the Design Page through the home page by clicking on the Custom Routes option (see figure 6.16). There he is requested to fill in a form about the information of the Route he wants to create, such as name, duration, an image, a description, a category depending the various Objects of Interest that he is going to place along its path. The user defines the path of the Route on the provided Google Map by clicking on the GPS points he wants to include on the path. The added GPS LatLng positions are presented in a form of a list located on the right side of the Google Map. The user is also provided with an option to undo his last action, the last added GPS position, by clicking on the Undo button. When the user fills in the required fields of the form and defines the desired path he can proceed to the next step by clicking on the Submit button.


Follow the steps to design the route and the points of interest you want to share with other users.

Fill out the form information and pick the route points.

Design your custom route by setting the path

• Route Name: MyRoute

• Route Duration: 25 minutes

• Route Image(s): Chania-on-island-of-Crete.jpg

• Description:

• Route Type: Sightseeing

Click on the map to add waypoints to your Route:

Places you selected to visit:

Sofoklis Venizelos Square 8

Sofoklis Venizelos Square 8

Sofoklis Venizelos Square

Sofoklis Venizelos Square 6-8

Pl. Agoras 69

Figure 6.17 Interface of the Custom Routes page. Here the user can fill in the information about the Route and set its path.

We are going to analyze the basic elements of the Custom Routes page interface:

①Form Fields: The user can fill in the information of the created Route.

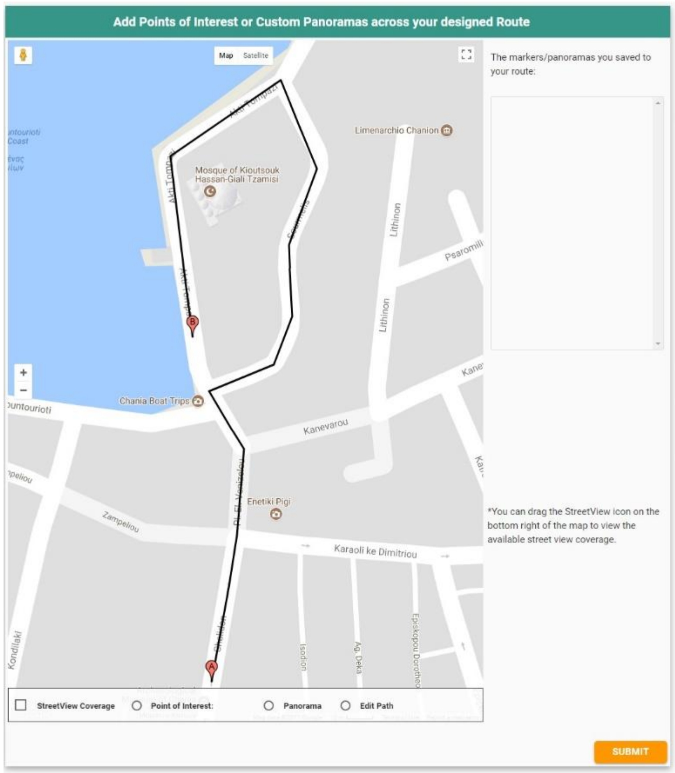
②Steps: Indication of the step in the Route creation page.

③Map: A Google Map object with event listeners responsible for creating the path of the Route, add Objects of Interest and Custom Panoramas.

④List: We use an html list object to keep track of either the waypoints of the Route or the created Objects of Interest and Custom Panoramas.

6.4.3 Create Custom Panorama

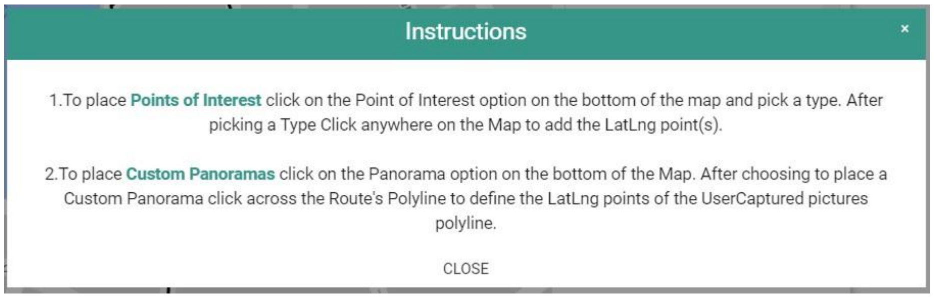
We are going to provide the sequence of steps a user has to do in order to create a Custom Panorama Object. First he has to pick the Panorama option from the available radio buttons on the bottom of the Google Map (see Figure 6.13). Upon selecting the option he has to create the path of the Custom Panorama polyline by clicking on the parts of the Route's path he wants to associate with UserCaptured Panoramic Pictures. He has the options of undoying his last action (last added GPS position) and submitting the designed path.


Add Points of Interest or Custom Panoramas across your designed Route

The markers/panoramas you saved to your route:

*You can drag the StreetView icon on the bottom right of the map to view the available street view coverage.

Figure 6.18 Interface on Custom Routes page after submitting the Route. He is presented with options for creating Objects of Interest and Custom Panoramas.


Instructions

1.To place **Points of Interest** click on the Point of Interest option on the bottom of the map and pick a type. After picking a Type Click anywhere on the Map to add the LatLng point(s).

2.To place **Custom Panoramas** click on the Panorama option on the bottom of the Map. After choosing to place a Custom Panorama click across the Route's Polyline to define the LatLng points of the UserCaptured pictures polyline.

CLOSE

Figure 6.19 Upon creating the Route the user has the options to add Points of Interest or Custom Panoramas along the path. He is given instructions on how to handle each option.

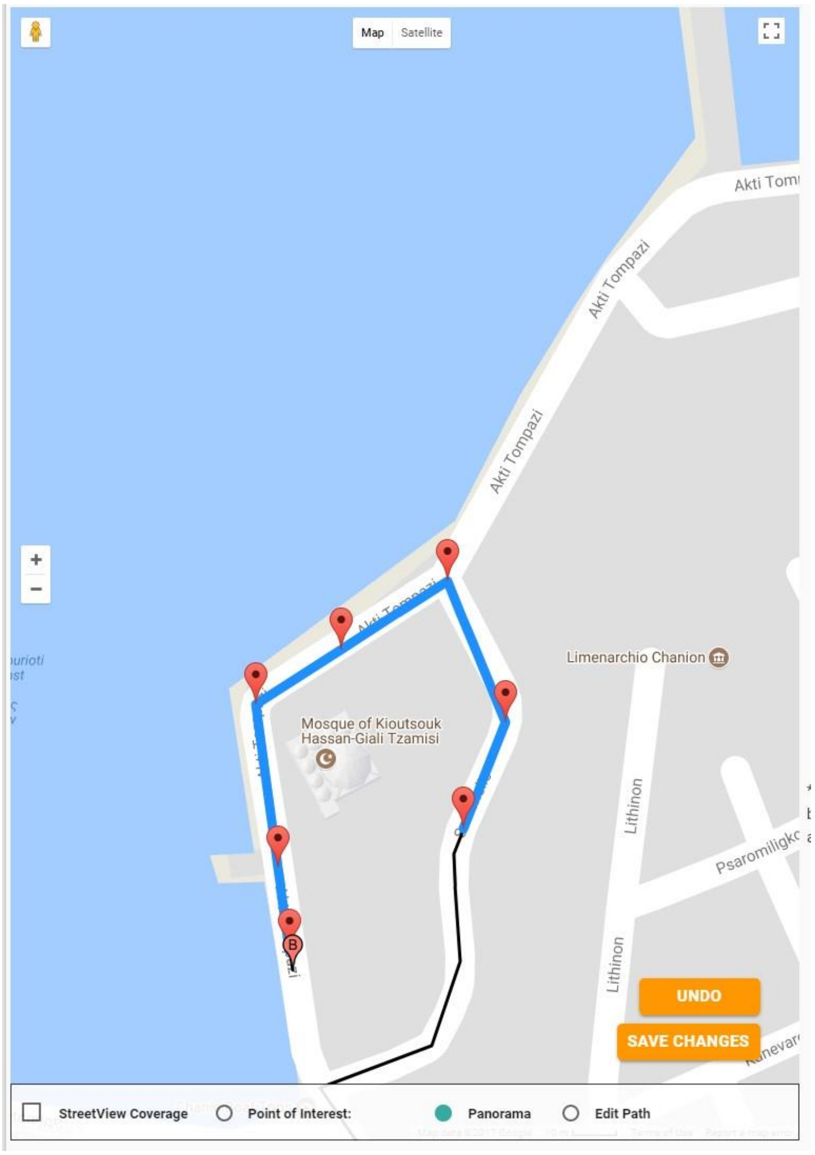
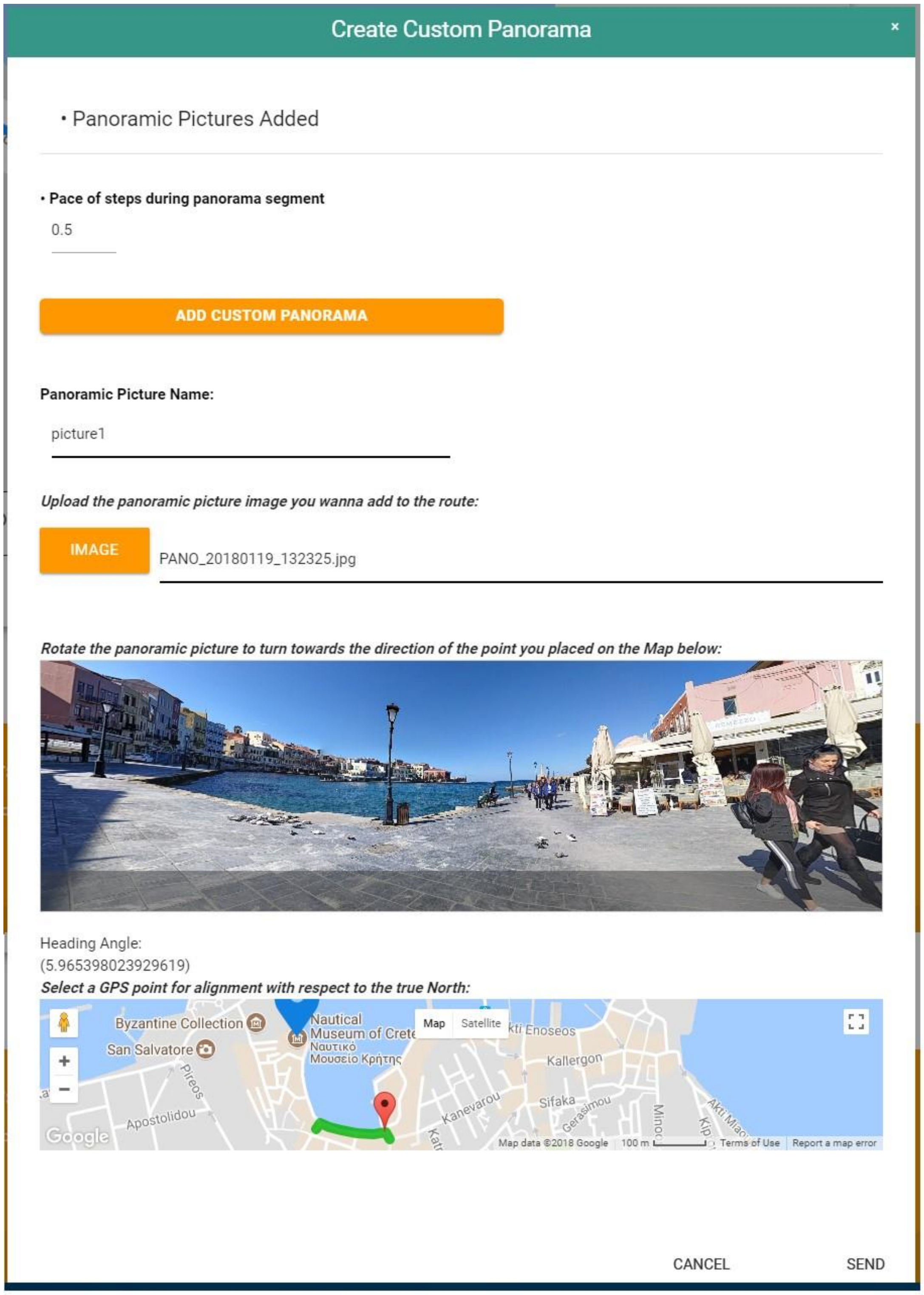


Figure 6.20 Upon selecting to place Custom Panoramas, the user enables the event listener responsible for creating a Polyline object representing the path of the Custom Panorama Objects containing the UserCaptured Panoramic Pictures.


Create Custom Panorama

• Panoramic Pictures Added

• Pace of steps during panorama segment

0.5

ADD CUSTOM PANORAMA

Panoramic Picture Name:

picture1

Upload the panoramic picture image you wanna add to the route:

IMAGE PANO_20180119_132325.jpg

Rotate the panoramic picture to turn towards the direction of the point you placed on the Map below:

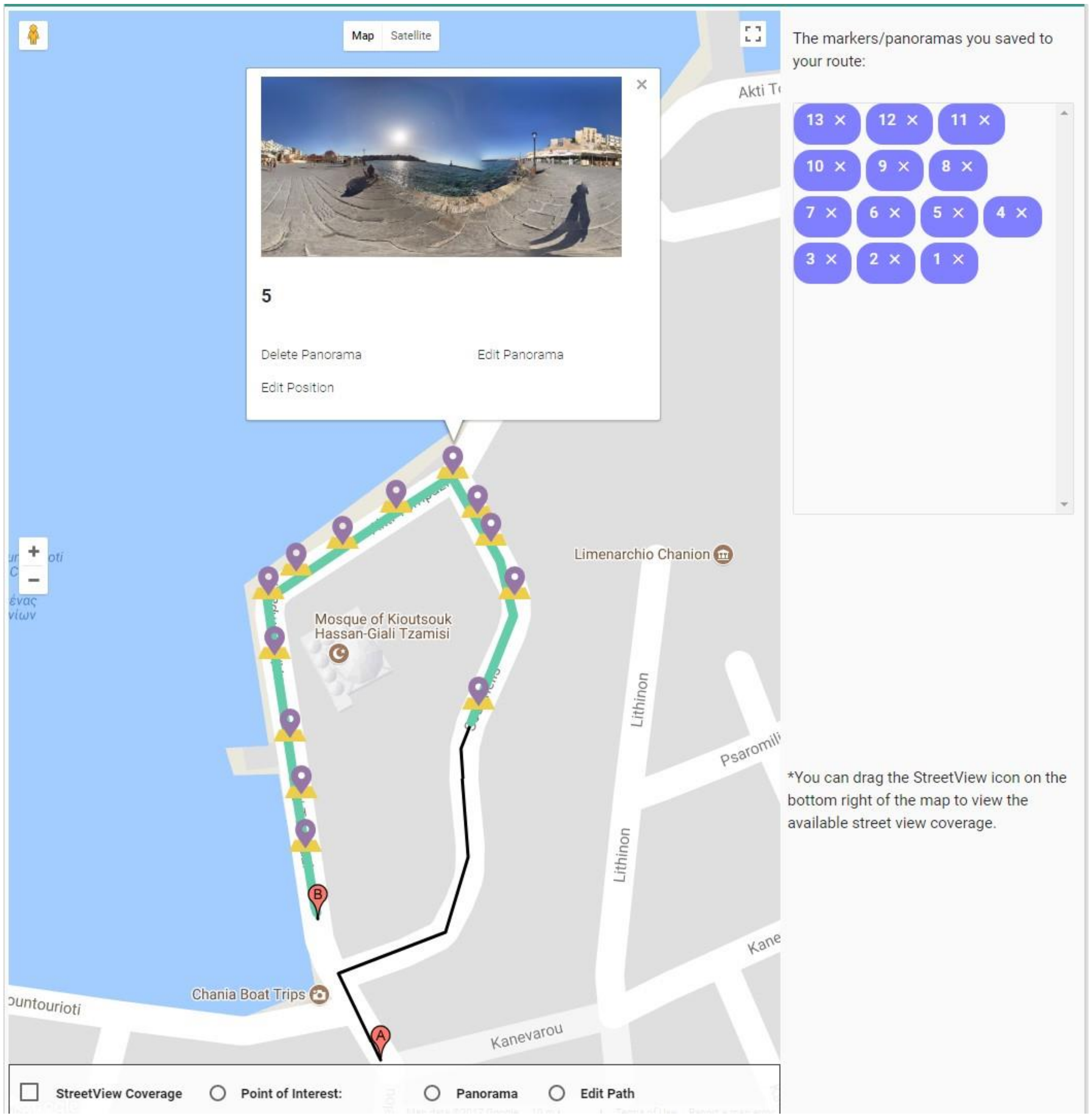
Heading Angle:
(5.965398023929619)

Select a GPS point for alignment with respect to the true North:

CANCEL SEND

Figure 6.21 Interface of the Custom Panorama popup. Here the user fills in the required information and uploads the UserCaptured Panoramic Pictures. He can define the navigation speed by setting the number of steps between two consecutive points. He also has to define a second GPS point in order to align the Panoramic Picture regarding the True North.

Upon submitting the path he is introduced with the appropriate interface where he can upload the UserCaptured Panoramic Pictures. He can define the navigation speed by setting the number of steps between two consecutive points. He also has to define a second GPS point in order to align the Panoramic Picture regarding the True North.



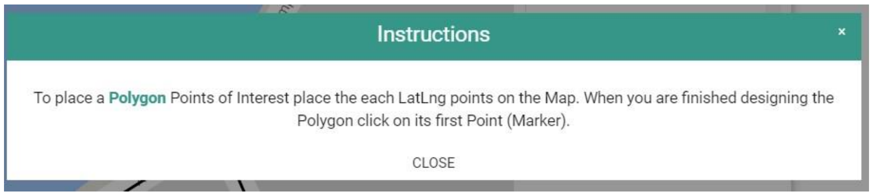
The markers/panoramas you saved to your route:

*You can drag the StreetView icon on the bottom right of the map to view the available street view coverage.

Figure 6.22 Example of a submitted Custom Panorama. Each UserCaptured Panoramic picture is represented with a Google Maps Marker. The Marker is interactive, giving the users the option of editing the point of view of the UserCaptured Picture, changing its location or deleting it.

6.4.4 Create an Object of Interest

In this section we are going to present the screen captures of the Interface associated with the functionality of creating an Object of Interest along a Route. Much like the Use Case of creating a Custom Panorama, the user is led to the interface shown in Figure 6.11 upon creating a new Route. There he has the options of creating an Object of Interest, a Custom Panorama or editing the path of the Route. When the user selects to create an Object of Interest a popup is displayed providing him the option of creating a Single Point or a Polygon Object of Interest. We have analyzed the differences about those two types in the chapter Application Functionality and Algorithms. We are going to give an example of the creation of a Polygon Object of Interest (same steps are followed for the Single Point). We provide the user instructions regarding the creations of the Polygon. After submitting the Polygon, the user has to fill in a form, displayed on a popup, about the Information of the Object of Interest. In this form he also has to define at least one Presentation Section associated with the Object of Interest.


Instructions

To place a **Polygon** Points of Interest place the each LatLng points on the Map. When you are finished designing the Polygon click on its first Point (Marker).

CLOSE

Figure 6.24 Instructions provided regarding the design of a Polygon on top of the Google Map.

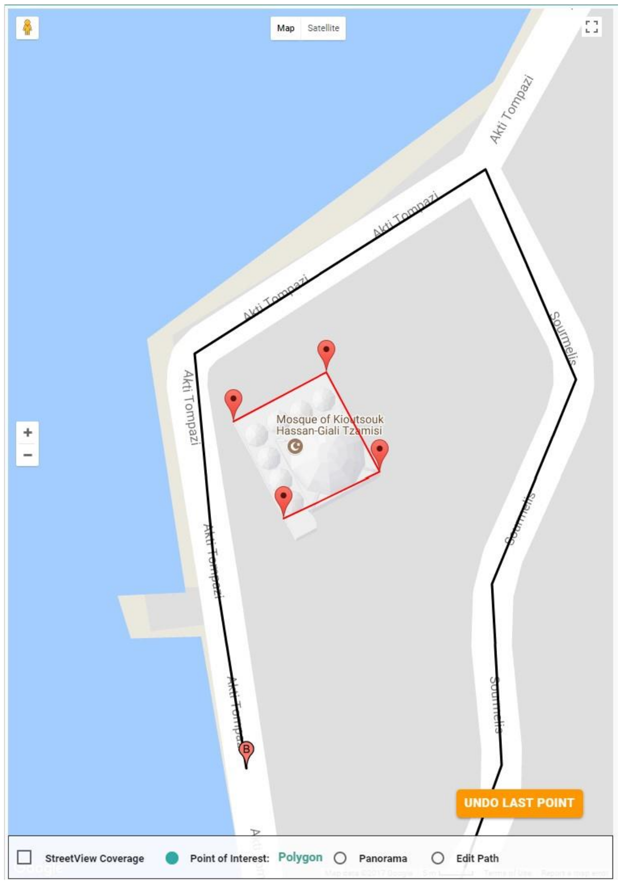


Figure 6.23 Interface of designing the Polygon on top of the Google Map. Clicking on the first placed Marker submits the final shape of the Polygon.

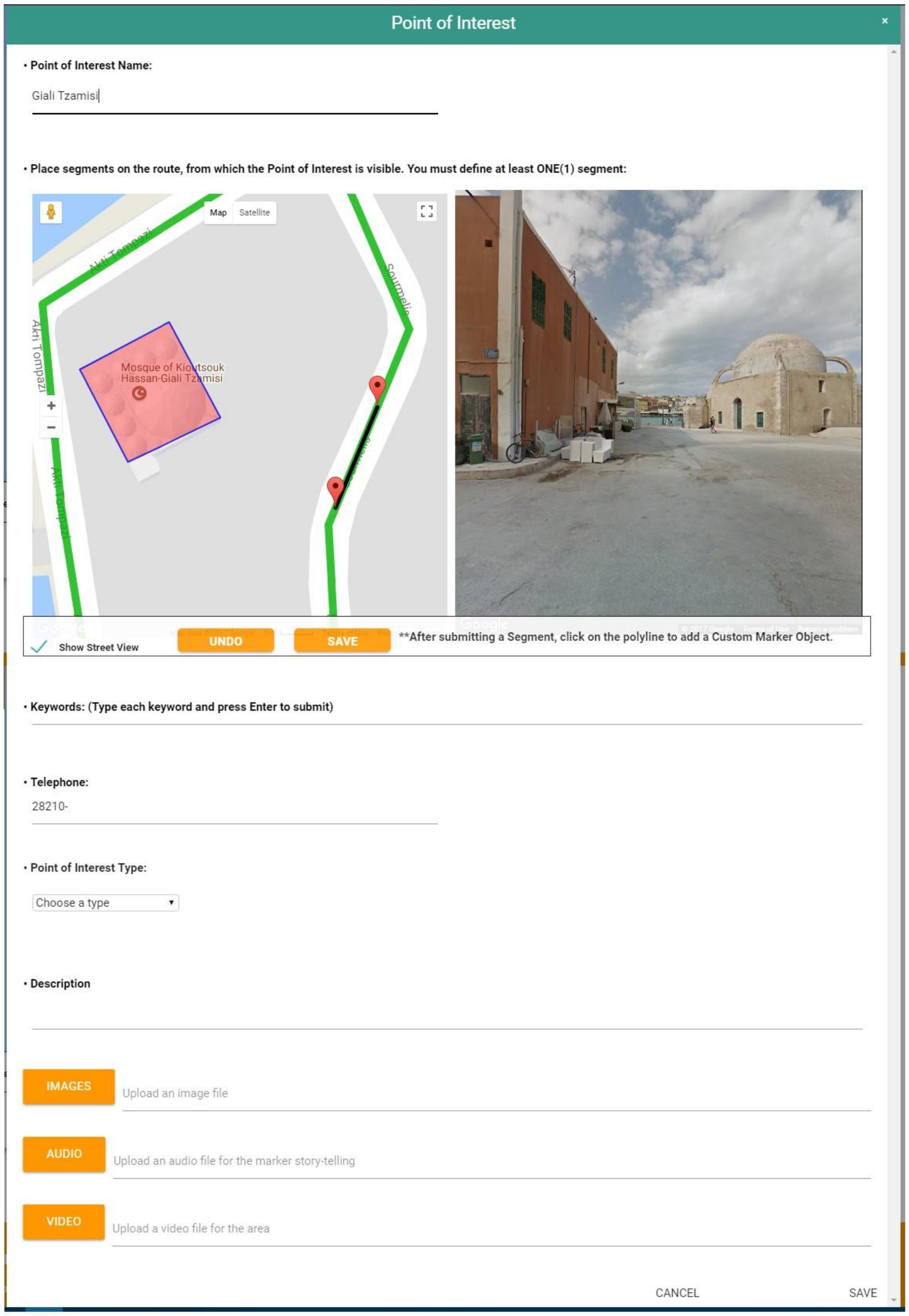
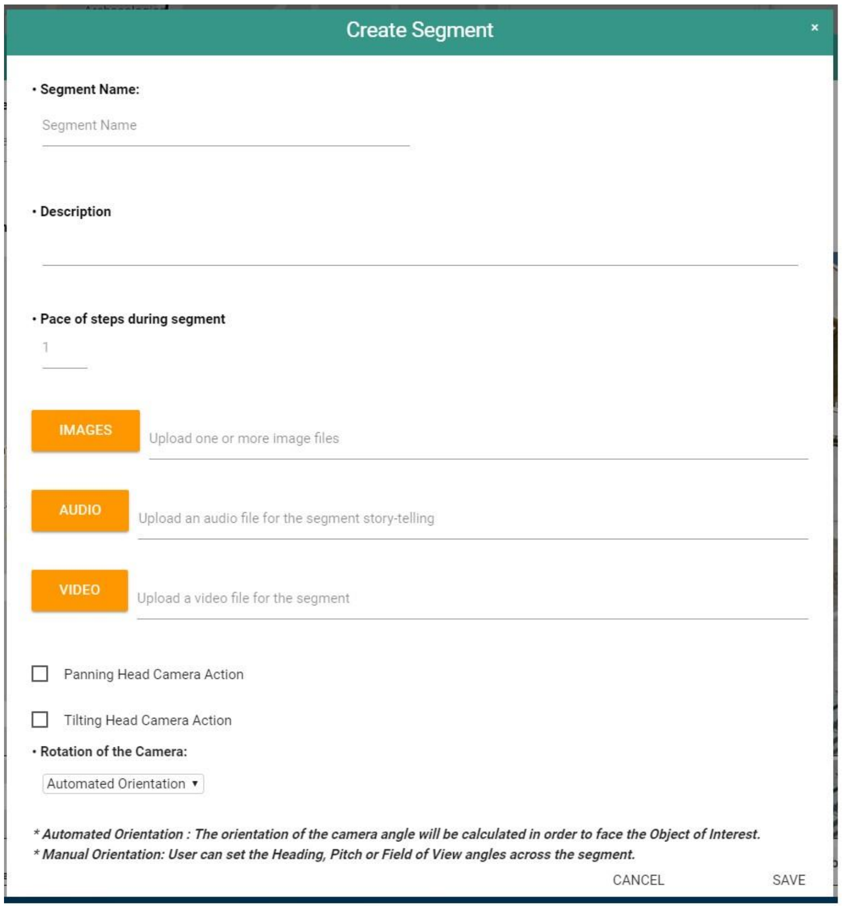

Point of Interest
×
• Point of Interest Name:
Giali Tzamisi
• Place segments on the route, from which the Point of Interest is visible. You must define at least ONE(1) segment:
Map
Satellite
Akti Tompazi
Sourmelis
Mosque of Kioutsouk Hassan-Giali Tzamisi
Show Street View
UNDO
SAVE
**After submitting a Segment, click on the polyline to add a Custom Marker Object.
• Keywords: (Type each keyword and press Enter to submit)
• Telephone:
28210-
• Point of Interest Type:
Choose a type
• Description
IMAGES
Upload an image file
AUDIO
Upload an audio file for the marker story-telling
VIDEO
Upload a video file for the area
CANCEL
SAVE

Figure 6.25 Interface of the Object of Interest Form associated with Use Case 11 (Create Object of Interest).

6.4.5 Create an Object Presentation Section

In this section we are going to present the screen captures of the Interface associated with the functionality of creating an Object Presentation Section associated with an Object of Interest. Upon filling in the form for the Object of Interest, the author is required to define at least one Object Presentation Section for it by setting its path on the Polyline of the Route displayed on the Google Map within the Object of Interest Interface. The author is provided with the options to undo the last added point of the Object Presentation Section and submitting the defined. Within the interface of the Object Presentation Section the author can define the information such as name, upload a picture, an audio file for the narration, a short description and an optional video file. He also can select the handling of the camera orientation towards the Object of Interest. He can select to let the camera's orientation be handled by the algorithms defined in our application, set the parameters of his liking that will define the Projection Views along the Object Presentation Section within the interface or he can set a Panning or Tilting Head Action which will be useful to present large nature objects or large Objects of Interest.


Create Segment

• Segment Name:

Segment Name

• Description

• Pace of steps during segment

1

IMAGES
Upload one or more image files
AUDIO
Upload an audio file for the segment story-telling
VIDEO
Upload a video file for the segment

☐ Panning Head Camera Action

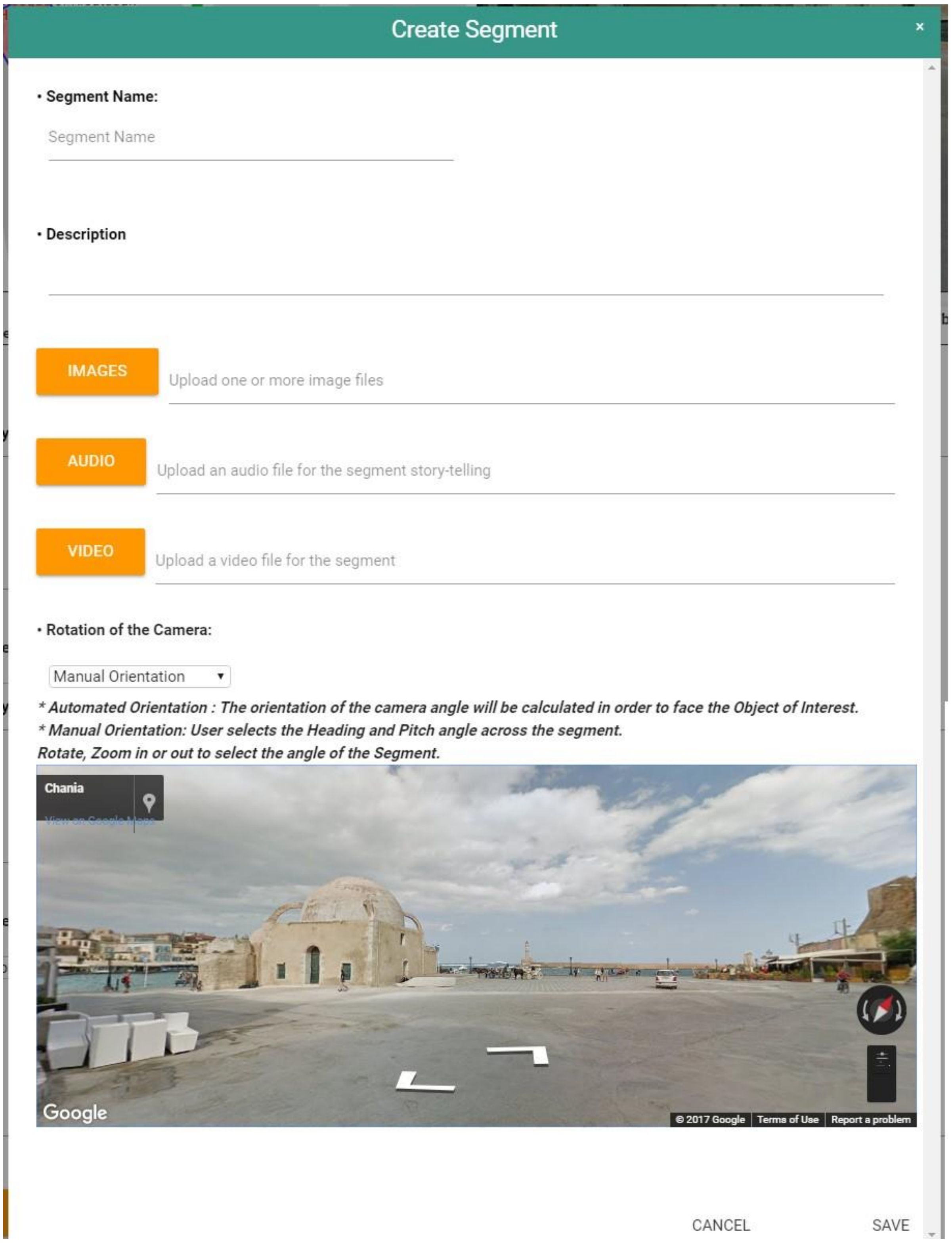
☐ Tilting Head Camera Action

• Rotation of the Camera:

Automated Orientation ▾

** Automated Orientation : The orientation of the camera angle will be calculated in order to face the Object of Interest.*
** Manual Orientation: User can set the Heading, Pitch or Field of View angles across the segment.*

Figure 6.26 Interface of an Object Presentation Section. The user can define the information of the Section and the handling of the camera. Example of Automated Orientation.



Create Segment

• **Segment Name:**

Segment Name

• **Description**

IMAGES Upload one or more image files

AUDIO Upload an audio file for the segment story-telling

VIDEO Upload a video file for the segment

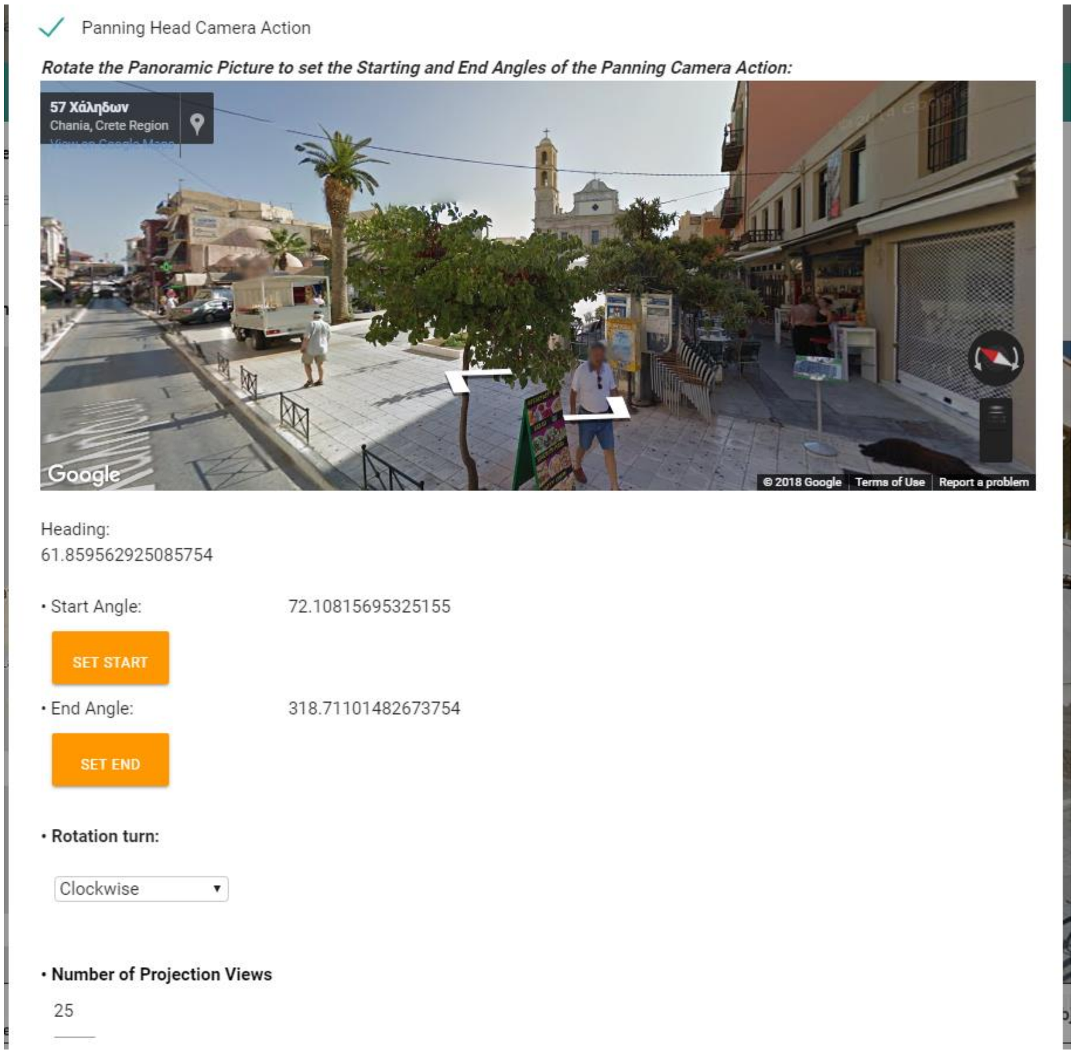
• **Rotation of the Camera:**

Manual Orientation

** Automated Orientation : The orientation of the camera angle will be calculated in order to face the Object of Interest.*
** Manual Orientation: User selects the Heading and Pitch angle across the segment.*
Rotate, Zoom in or out to select the angle of the Segment.

CANCEL SAVE

Figure 6.27 Example of Object Presentation Section with Manual Camera Orientation. The user is presented with the Panoramic Picture of the Section. Through StreetView events he can define the heading, pitch and field of view angles.


Panning Head Camera Action

Rotate the Panoramic Picture to set the Starting and End Angles of the Panning Camera Action:

Heading:
61.859562925085754

• Start Angle: 72.10815695325155

SET START

• End Angle: 318.71101482673754

SET END

• **Rotation turn:**

Clockwise

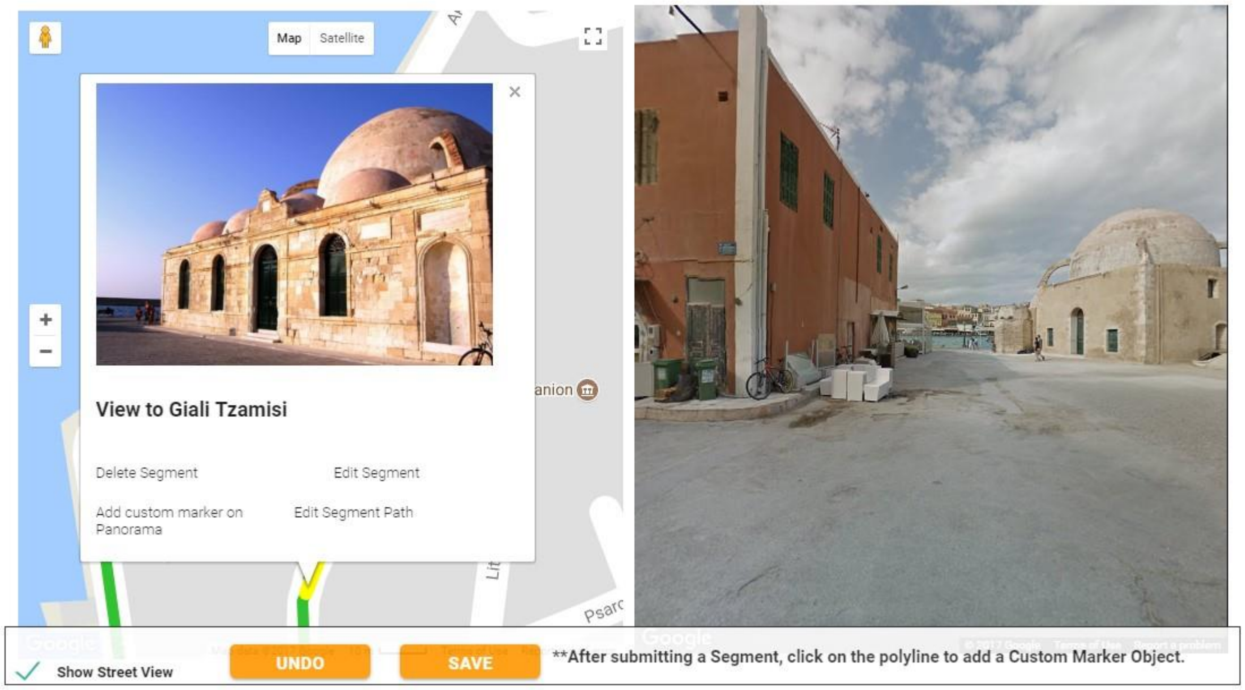
• **Number of Projection Views**

25

Figure 6.28 Example of Object Presentation Section with Panning Head Action. The author has to define the start and end angle of the Panning by rotating the panoramic picture and setting the angles. He also has to define the type of rotation, clockwise or counter clockwise, and the number of the Projection Views that will be generated for the Panning Action.

6.4.6 Create a Custom Marker

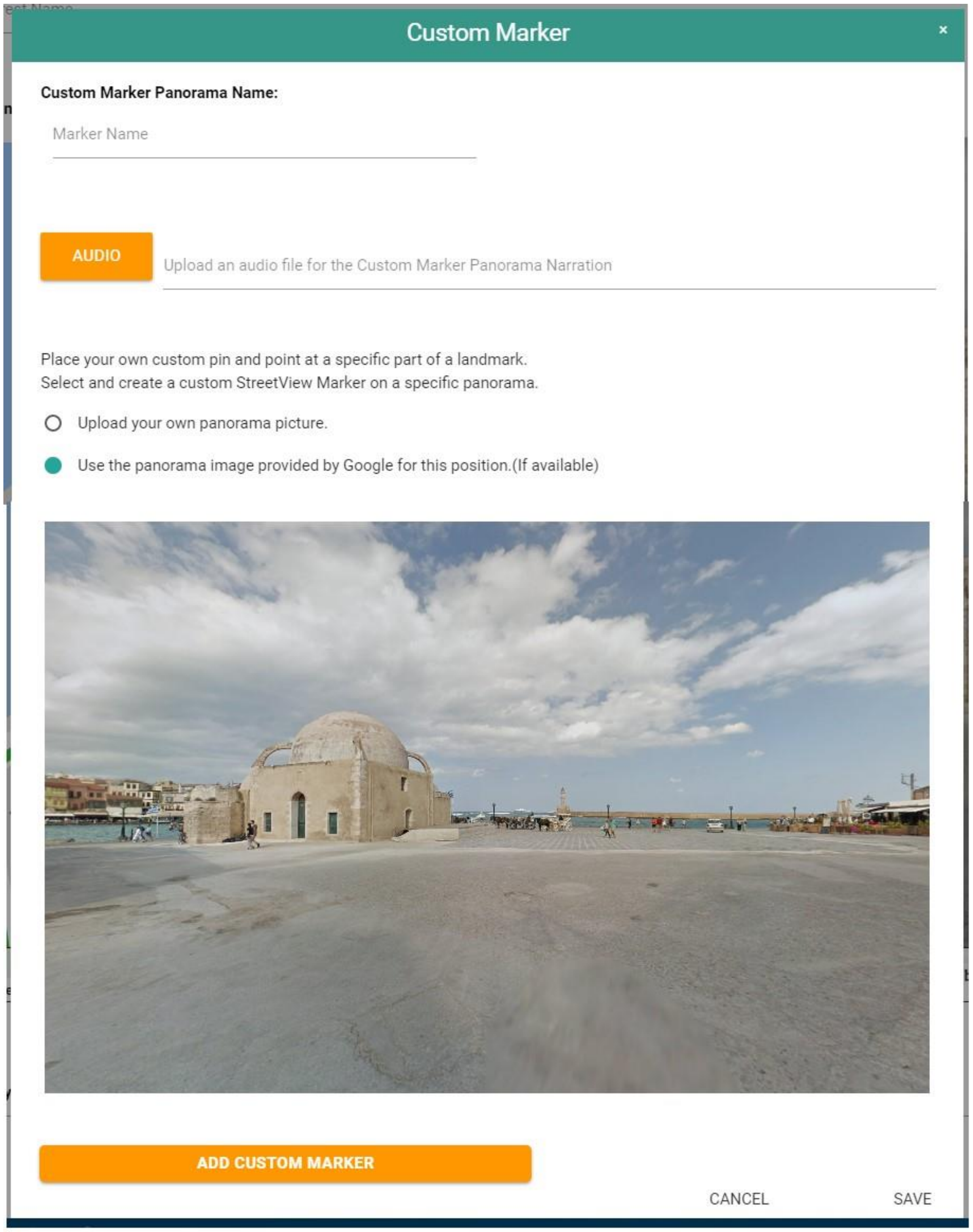
In this section we are going to present the screen captures that show the sequence of steps in the Interface associated with the functionality of creating a Custom Marker for an associated Object of Interest (Use Case. Upon submitting an Object Presentation Section for an Object of Interest, the user can interact with it, given the options of editing its information, its path, to delete it and to add a Custom Marker along the path (Figure 6.29).



**After submitting a Segment, click on the polyline to add a Custom Marker Object.

Figure 6.29 Interaction with an Object Presentation Section gives the user the option of placing a Custom Marker along its path.

After selecting to create a Custom Marker he is introduced with the corresponding interface (Figure 6.30). On this interface he can either upload the Panoramic Picture on which he will add the Custom Marker(s) or retrieve and use a StreetView Panoramic Picture, if one is available at the chosen LatLng position. After selecting the Panoramic Picture he can add one or more Custom Markers of his liking, either in form of a picture or an ObjectEnclosingRectangle. The user has to rotate the Panoramic Picture on the point of view of his liking. He is presented with the options of either placing an Icon Marker or an ObjectEnclosingRectangle. The placement of these markers are done within a Canvas interface which displays a cropped image depending the point of view the user defined (Figures 6.31 and 6.32).



Custom Marker Panorama Name:

Marker Name

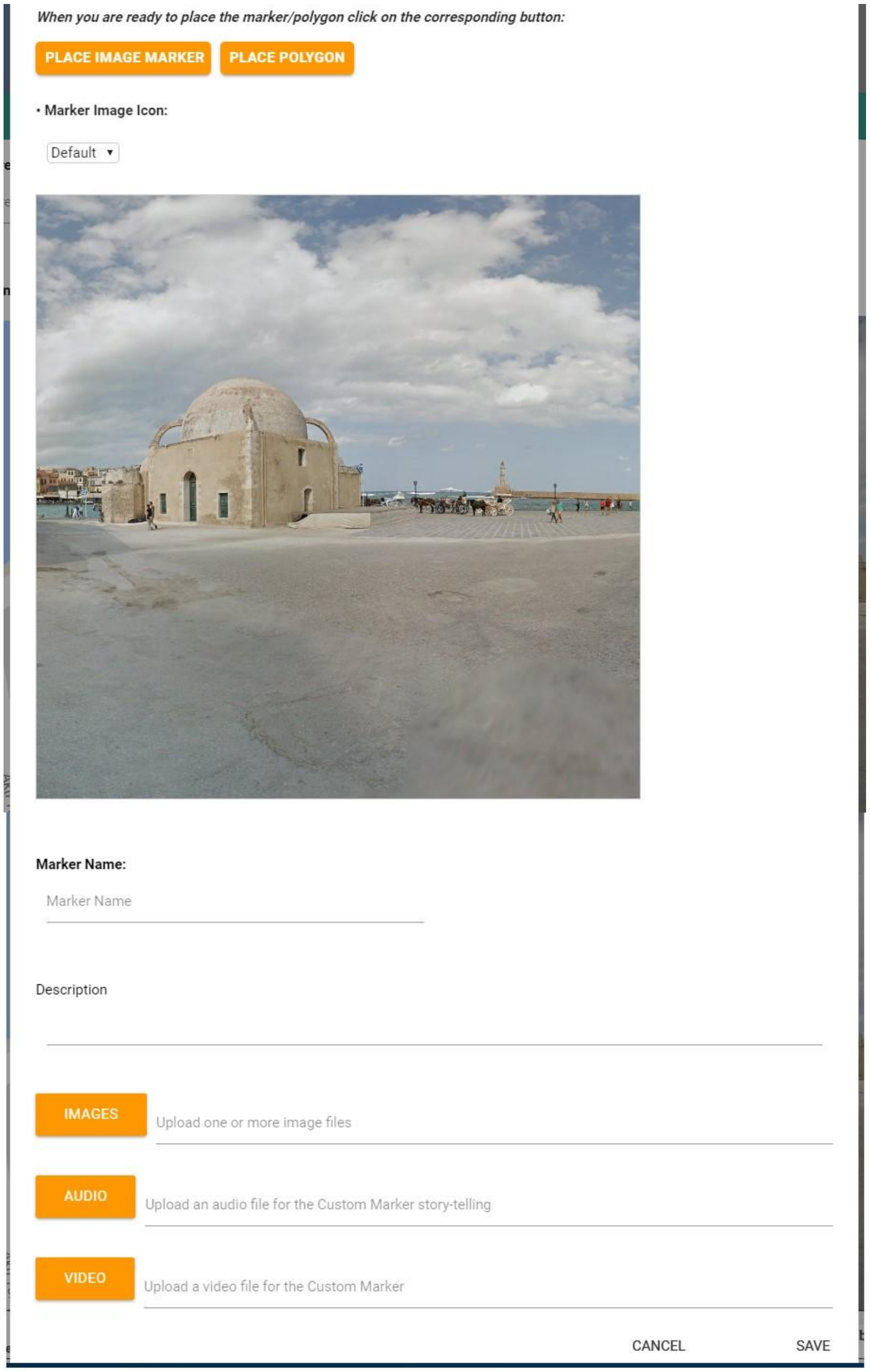
Upload an audio file for the Custom Marker Panorama Narration

Place your own custom pin and point at a specific part of a landmark.
Select and create a custom StreetView Marker on a specific panorama.

○ Upload your own panorama picture.

● Use the panorama image provided by Google for this position.(If available)

Figure 6.30 Interface of the Custom Marker popup. The user can upload an audio file for the narration of the marker and select to upload a UserCaptured Panoramic Picture or retrieve a StreetView Panoramic Picture if available. After selecting the Panoramic Picture he can place one or more Custom Markers.



When you are ready to place the marker/polygon click on the corresponding button:

PLACE IMAGE MARKER
PLACE POLYGON

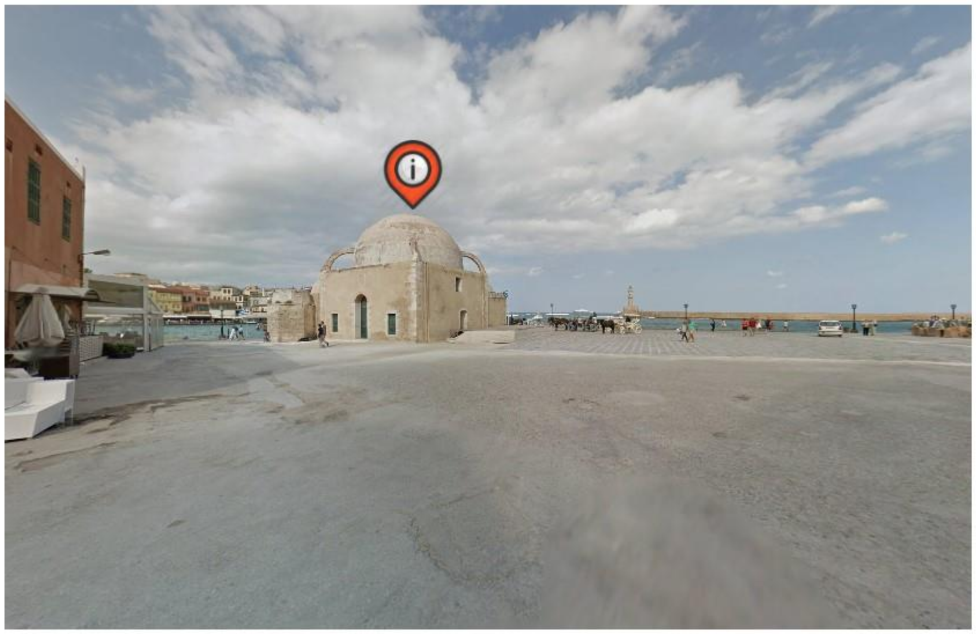
• Marker Image Icon:

Marker Name:

Description

IMAGES
Upload one or more image files
AUDIO
Upload an audio file for the Custom Marker story-telling
VIDEO
Upload a video file for the Custom Marker
CANCEL
SAVE

Figure 6.31 . Interface of placing an Image or a Polygon Marker on a Panoramic Picture. The interaction is done with a Canvas element that receives the clicks of the user and computes the appropriate pixels on the Panoramic Picture.



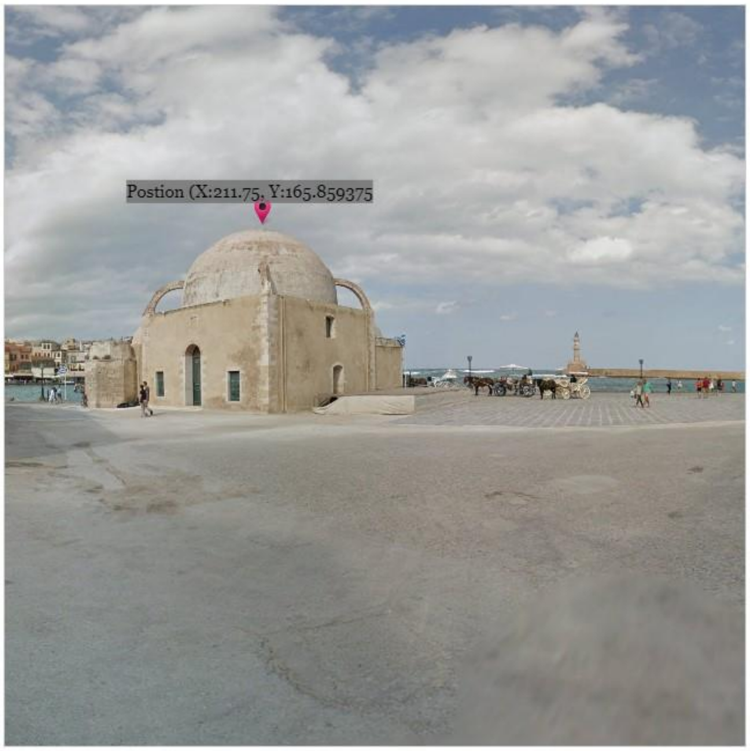


Figure 6.32 The user sets the points of view parameters, by rotating the Panoramic Picture. By picking to add an Image Marker (or Polygon) a cropped image based on the defined point of view is drawn on an interactive Canvas Element. The user clicks on the Canvas element and through appropriate algorithms and methods the corresponding position on the Panoramic Picture is calculated.

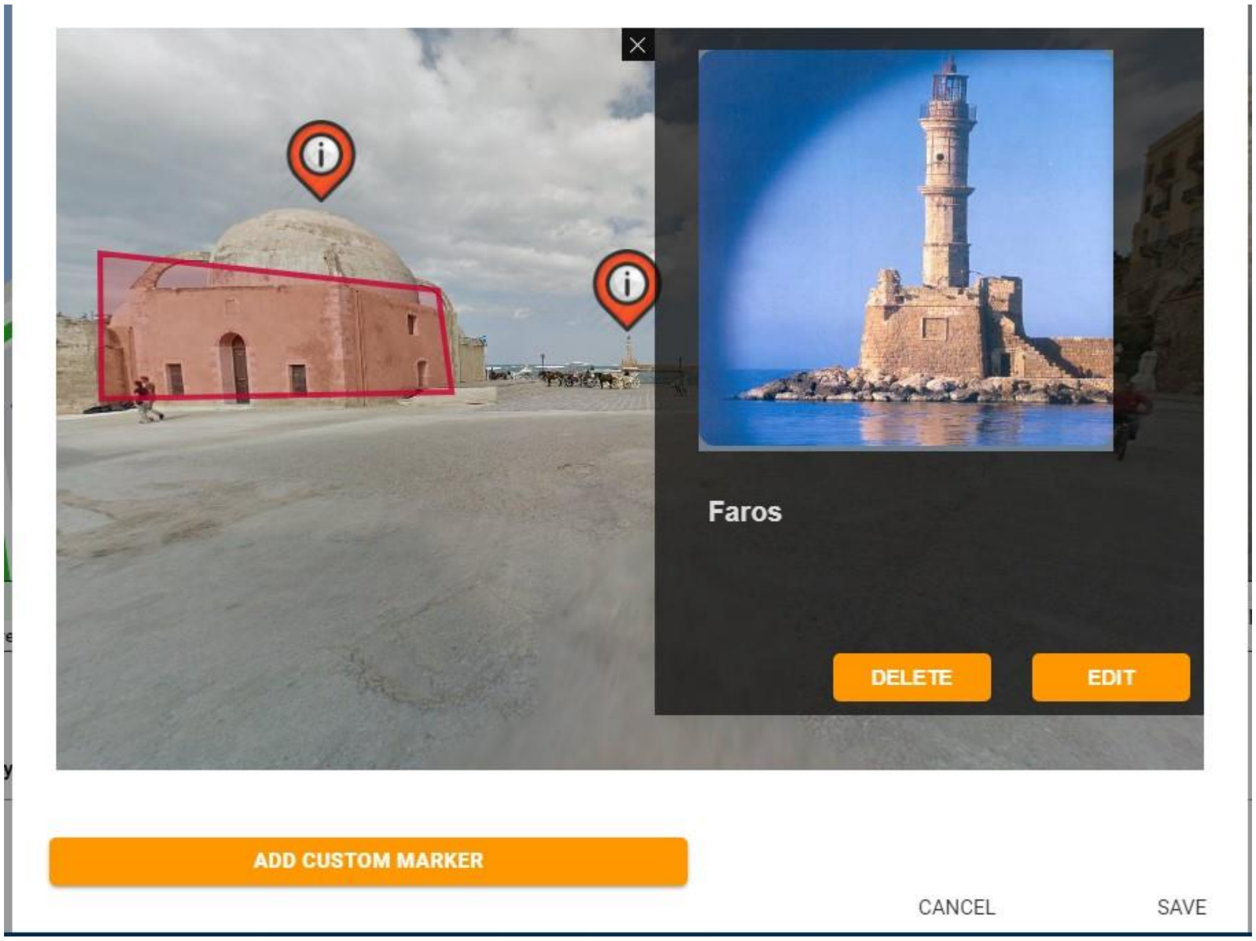
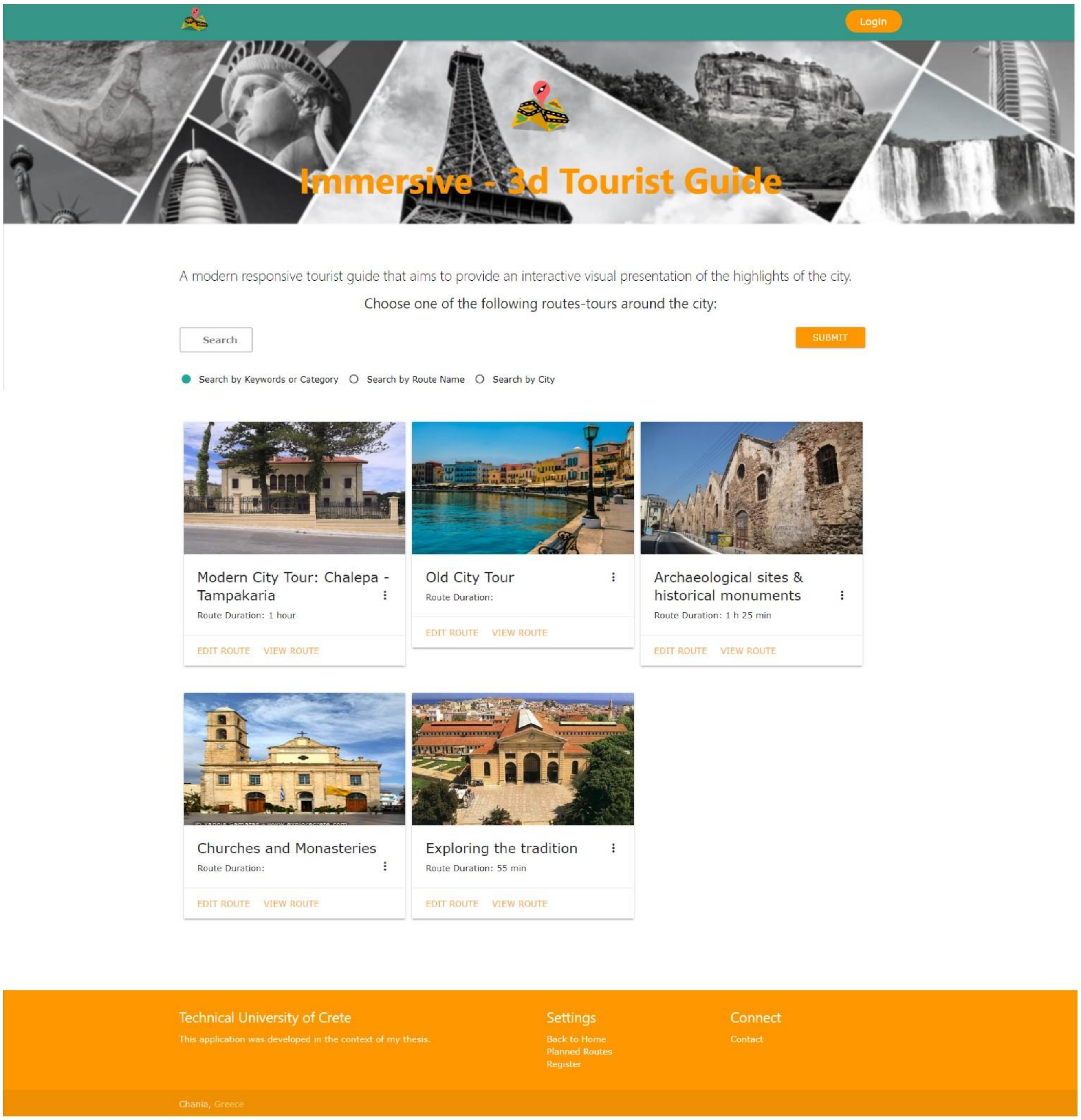


Figure 6.33 Example of created Custom Markers on top of a Panoramic Picture. Each Custom Marker is interactive within the Custom Routes Interface and give the user the options of edit and delete.

6.4.7 View a Planned Route

In this section we are going to present the steps a user follows in order to view and navigate a planned Route within our application. Within the Home page (see Figure 5.15) he is redirected to the Planned Routes page. There he is presented with an interface with all the created listed. There are search functionalities enabling him to look for Routes with specific Name, specific keywords or look for Routes in a specific city. Each planned Route is presented with a Card Interface. Each card presents the image of the Route, the name of the Route and its duration. There are also options for editing and viewing the Route. By clicking on the card's image more information are presented to the user, giving him a short description of the Route.



A modern responsive tourist guide that aims to provide an interactive visual presentation of the highlights of the city.

Choose one of the following routes-tours around the city:

Modern City Tour: Chalepa - Tampakaria
Route Duration: 1 hour

Old City Tour
Route Duration:

Archaeological sites & historical monuments
Route Duration: 1 h 25 min

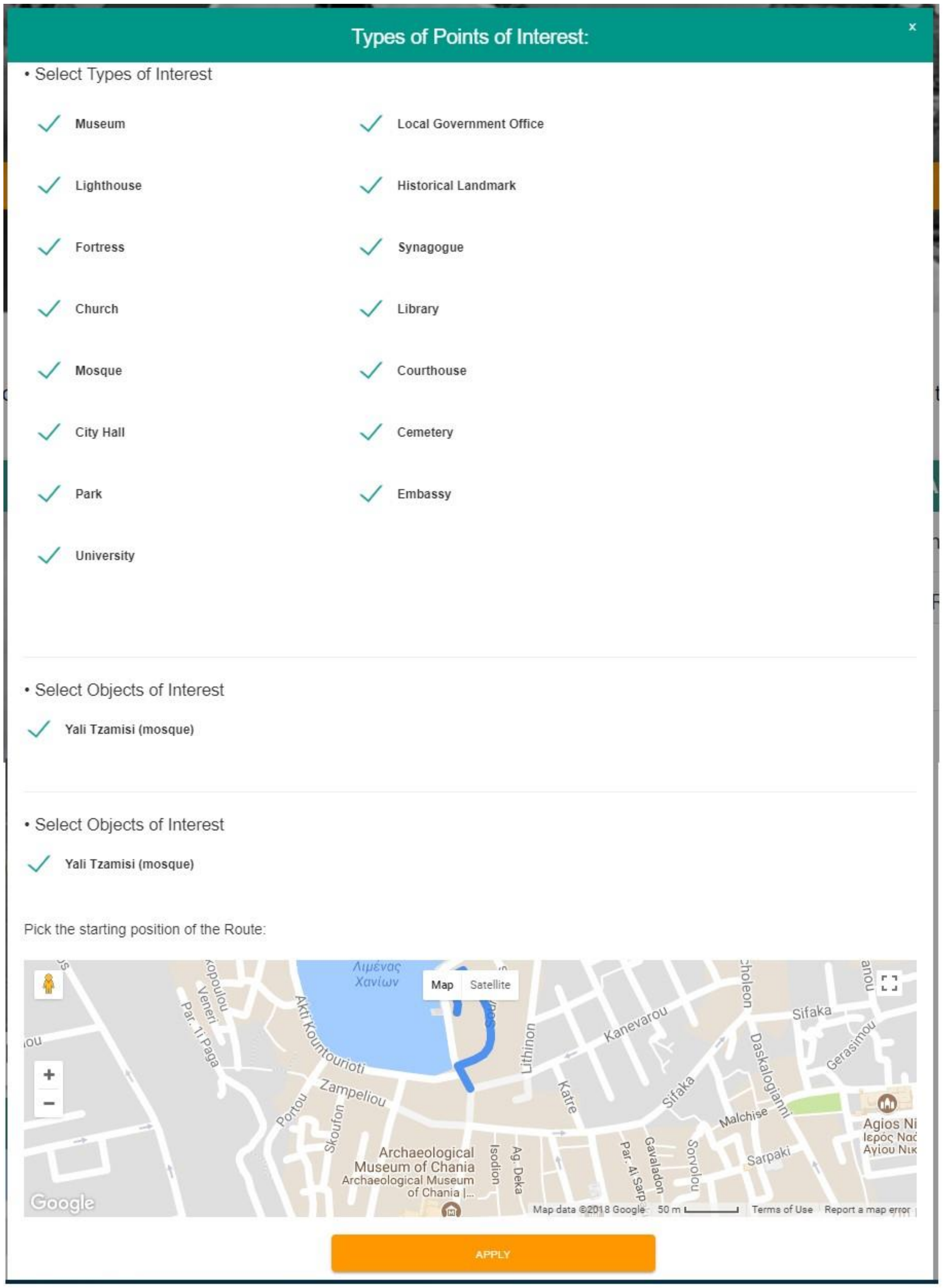
Churches and Monasteries
Route Duration:

Exploring the tradition
Route Duration: 55 min

Technical University of Crete
This application was developed in the context of my thesis.

Chania, Greece

Figure 6.34 Planned Routes page Interface. The created Routes of the application are listed here in form of cards. The user may choose to edit or view a Route.


Types of Points of Interest:

• Select Types of Interest

✓ Museum	✓ Local Government Office
✓ Lighthouse	✓ Historical Landmark
✓ Fortress	✓ Synagogue
✓ Church	✓ Library
✓ Mosque	✓ Courthouse
✓ City Hall	✓ Cemetery
✓ Park	✓ Embassy
✓ University	

• Select Objects of Interest

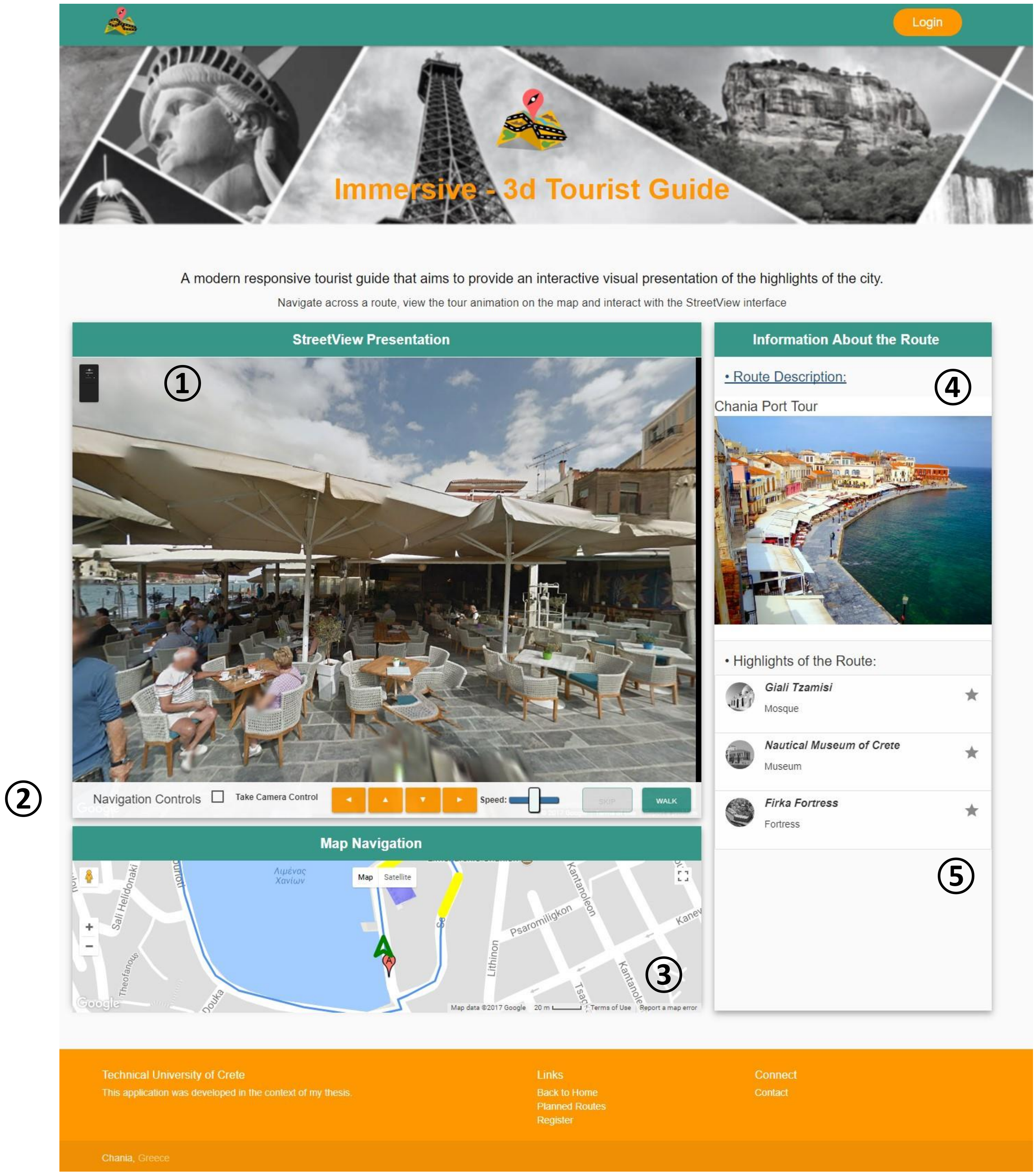
✓ Yali Tzamisi (mosque)

• Select Objects of Interest

✓ Yali Tzamisi (mosque)

Pick the starting position of the Route:

Figure 6.35 Interface on which a user defines his interests and the starting point of the Navigation. By clicking on each Object of Interest he can view the associated Object Presentation sections. This helps him point the starting point at the start of an Object Presentation section of the Object of Interest he wants to interact with.


A modern responsive tourist guide that aims to provide an interactive visual presentation of the highlights of the city.

Navigate across a route, view the tour animation on the map and interact with the StreetView interface

This application was developed in the context of my thesis.

Figure 6.36 Interface of the Navigation Page. Within this page the user views and interacts with the generated video of the planned Route he picked to view.

We are going to analyze the basic elements of the Navigation page interface:

① **Video presentation**: In this div element we present the Projection View of each Panoramic Picture (StreetView and UserCaptured) of the personalized generated time lapse video based on the Route.

② **Navigation Controls**: The user has the option to gain the control of the camera and point it at a specific direction. As long as this option is enabled the Route will follow those parameters to generate the Projection Views.

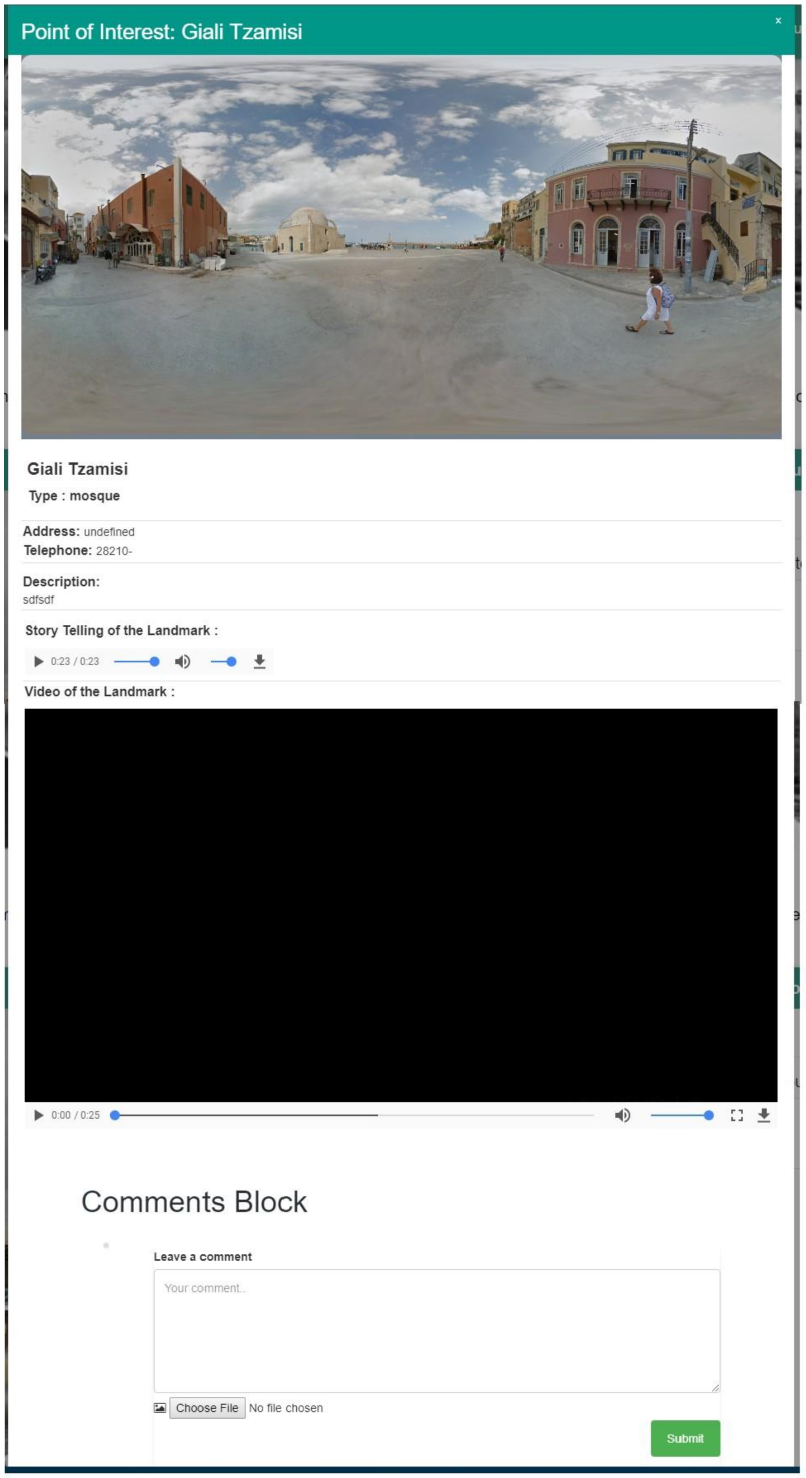
The user has also the option of stopping the Route at any time, in order to look around and interact with his surroundings. He can also change the speed of the generated video, increasing or decreasing it.

Finally, when a user walks within an Object Presentation Section he has the option to skip the presentation of the associated Object of Interest and the stop the narration being played.

③ **Map**: A Google Map object. On top of this Map the Route's path and all the Objects of Interest and their associated Object Presentation Sections and Custom Markers are set. The Objects of Interest are represented with either a Google Marker or a Polygon Object depending on their type. The Route and the Object Presentation Sections are represented with Polylines. All the objects of the Map are interactive on click events, presenting an info window with basic info such as the name of the Object of Interest and with a link giving the users the option to view more information.

④ **Route Description**: The information of the Route, such as the name, the description and the image are listed in this section.

⑤ **Highlights of the Route**: All the Objects of Interest of the Route are listed in this section. Each Object of Interest has a small preview of its image, its name and its category type. Clicking on one of those listed Objects of Interest opens the interface presenting all its information (see Figure 5.37).


Point of Interest: Giali Tzamisi

Giali Tzamisi

Type : mosque

Address: undefined
Telephone: 28210-

Description:
sdfsdf

Story Telling of the Landmark :

0:23 / 0:23

Video of the Landmark :

0:00 / 0:25

Comments Block

Leave a comment

Your comment..

Figure 6.37 Interface of the Object of Interest Presentation. The user can view the name of the Object, the type, information about its address and telephone. He can also hear a related audio file or view a video. Finally, if the user is logged in the system he can submit a comment regarding this Object of Interest.

6.5 Interface Evaluation

The system was evaluated during its development and after its final and stable form. The first evaluation took place in TUC-Music through a presentation of the functionality and the interface of the application to the laboratory faculty. The final evaluation of the system was done by providing human computer interface related questionnaires, by Chin[31] and Davis[32], both to experienced and inexperienced users in order to get an overall reaction for the final interface and functionality.

The presentation to the TUC-MUSIC was performed by presentencing the most demanding tasks of the system's functionality considering both the authoring and the navigation playback environments. The problems emerged of this process were:

- The speed of the navigation through StreetView Panoramic Pictures was causing the appearance of temporary black screen while moving from one Panoramic Picture to the next, since the request timeout between the StreetView request calls was too fast. We overcame this issue by projecting the StreetView Panoramic pictures using two div containers. The first container holds the StreetView currently used by the time lapse video, while the second container holds the Panoramic Picture of the second point as it is retrieved it in the background. This process gives satisfying time between the requests and improves the smooth generation of the time lapse video.
- The interface of the object visualization method needed to change in order to be easier for the authors of a Route to distinguish and use a single point a polyline or a polygon object in order to mark the Object of Interest footprint. We overcame this issue by providing instructions considering the construction of Objects of Interest.
- The error messages of the system need to be more distinct and clear in order to help the user and guide them successfully to recover from his error.
- Some of the fields in our interface are required to be filled in order to construct the associated (Object of Interest, Route, Section, etc.). There is need for a clear way to distinguish required and optional fields in the forms of the system's interface.

The final evaluation process was done through HCI questionnaires (6.1 and 6.2) provided to users. We used those questionnaires both to experienced users, like the laboratory of the TUC-Music, and inexperienced users.

[31] Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface, John P. Chin, Virginia A. Diehl and Kent L. Norman, Human-Computer Interaction Laboratory Department of Psychology University of Maryland College Park, MD 20742

[32] Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface, John P. Chin, Virginia A. Diehl and Kent L. Norman, Human-Computer Interaction Laboratory Department of Psychology University of Maryland College Park, MD 20742

Results & Conclusions of Questionnaires

Questionnaire Chin et al:

We asked **eight (8) users** to fill the questionnaires and we calculated the average score of each question of the **Chin et al b** questionnaire. Besides the laboratory staff, we distributed questionnaires to students of the Technical University. Two of which, were students of School of Electrical & Computer engineering (ECE), thus users with experience using a web application and understanding of the interface, but not as experienced as the MUSIC staff. Finally, the other two users were students of School of Environmental Engineering (ENVENG), thus less familiar with using a web application.

The scale of results ranged from 0 to 9, as we approach zero, the more negative the answer, the closer we get to the 9, the more positive.

Overall Reactions to the software:

1. Terrible/ Wonderful	**8.125**
2. Hard/ Easy	**7.125**
3. Frustrating/ Satisfying	**8**
4. Inadequate Power/ Adequate Power	**8.5**
5. Dull/ Stimulating	**8.25**
6. Rigid/ Flexible	**8.25**

Screen

1. Characters on the screen

Hard to read/ Easy to read	**8.5**

2. High-lighting on the screen simplifies the task

Not at all/ Very much	**8.5**

3. Organization of information on screen

Confusing/ Very clear	**8.5**

4. Sequence of screens

Confusing/ Very Clear	**8.5**

Terminology and System Information

1. Use of terms through the system

Inconsistent/ Consistent	**8.5**
2. Computer Terminology is related to the task you are doing	
Never/ Always	**8.5**
3. Position of messages on screen	
Inconsistent/ Consistent	**8.5**
4. Messages on screen which prompt user for input	
Confusing/ Very clear	**8.75**
5. Computer keeps you informed about what it is doing	
Never/ Always	**8.375**
6. Error Messages	
Unhelpful/ Helpful	**8.125**

Learning

1. Learning to operate the system	
Difficult/ Easy	**6.75**
2. Exploring new features by trial and error	
Difficult/ Easy	**7.35**
3. Remembering names and use of commands	
Difficult/ Easy	**8.75**
4. Tasks can be performed in a straight-forward manner	
Never/ Always	**7.625**
5. Help messages on screen	
Unhelpful/ Helpful	**7.75**

System Capabilities

1. System speed	
Too slow/ fast enough	**8.125**
2. System reliability	
Unreliable/ Reliable	**8.75**

3. System tends to be

Noisy/ Quiet	**8.75**

4. Correcting your mistakes

Difficult/ Easy	**8**

5. Experienced and inexperienced users' needs are taken into consideration

Never/ Always	**7**

Conclusions of Chin et al Questionnaire:

From the results above, we can assume that the users found the system satisfactory in most of its sectors. None of the users, despite been asked to answer objectively and without leniency, did not rate a question with a grade of less than six. This may be because users generally do not judge a system easily when asked to, because they do not want to reduce the work of the creators. Nevertheless, the questions that were given the lowest score on average regard the difficulty of the system, the appearance and handling of the error messages. These lower scores on those categories rely on the fact that the system targets more experienced users. Some of the functionalities considering the authoring environment of the system require deeper familiarity with the use of a website. Since the authoring environment targets users from a specific category, the related tasks demand knowledge about a city (or a geographical area) and its sights in order to be able to create a Route, using the provided methods and algorithms. The users also pointed out that some of the error messages in the system should be clearer.

Questionnaire Davis:

We also calculated the average score of each question in the Davis Questionnaire.

The scale of results ranged from 0 to 6, as we approach zero, the more positive the answer, the closer we get to 6, the more negative.

Perceived Ease of Use

1. Learning to operate the system would be easy for me.	**0.625**
2. I would find it easy to get the system to do what I want to do.	**1**
3. My interaction with the system would be clear and understandable.	**0.625**
4. I would find the system to be flexible to interact with.	**0.625**
5. It would be easy for me to become more skillful at using the system.	**0.375**
6. I would find the system easy to use.	**1.125**

Conclusions of Davis Questionnaire:

The majority of the users gave a positive score in the questions. Specifically, they thought that the system would help to achieve the goal, which is the construction and presentation of routes with interest to end users of a city or a geographical area. By analyzing the results, we can also see that the users think that the system is slightly difficult to use, but that also believe they will become familiar with the functionality and interface of the system rather fast.

Chapter 7

7 Summary & Future Research

7.1 Summary

Video based navigations and presentation of sights of cultural or tourism interest within a city or geographic area of interest are important means to give the final user of a web application a feeling on the interesting sights within the wider environment, the architecture of the city and the life of the city or area. However, video based applications tend to be rigid and they typically follow a strict predefined flow of video projection, thus not allowing the user to have control of what and how to view the sights around him while navigating in the city.

We have designed and implemented a system that can be used to create web based applications that allow much more flexible video based navigation and presentation of interesting sights in a city or geographic area that can be used for culture and tourism applications of a destination. The applications that the system generates are video based. They can take into account the interest of the user to generated personalized presentation of the navigation through a city or area controlling what sights are presented with more detail to the user.

In addition, the functionality offered allows interactive control of the user at any point in the navigation. He can stop, look around, continue viewing directly ahead or at any direction that he chooses, view video presentations of more interesting sights in addition to those determined by his profile, etc. Thus the applications offer personalization but at the same time complete interactive control of viewing and interaction during navigation through the city or area, thus creating a feeling of immersion for the final user.

An important consideration in the design was to be able to use in the interactive video creation both StreetView panoramic images as well as panoramic images captured by users and stored in a data base. This way the system exploits preexisting StreetView panoramic images if desirable, and at the same time it can replace StreetView images whenever it is desirable. High quality, and in shorter GPS intervals, user captured panoramic images may be needed for important sights of interest or for areas not covered by StreetView panoramic images.

7.2 Future Research

Several extensions of this area of research may be pursued. The applications allow the designer to specify routes of cultural and tourism interest in a city. Some of those routes can contain common parts. Thus an extension would be to allow sharing of parts of the designed routes for saving storage requirements. Another extension would be to allow the user to interactively select and combine parts of different routes in his navigation. A third extension would be to allow automatic selection by the system of parts of routes of the city to be used for navigation based on the user interests, thus offering also personalized routes, not only personalized navigation and presentation in preselected routes.

Another line of extension can be directed towards performance optimizations in the presentations. The presentations may be slow or faster, essentially using video form nearby locations on a route or from far apart locations. The presentation can be of variable quality for different parts of the navigation. For example sights that are of interest to the user can be projected with more detail (shorter spacing between two consecutive pictures). In addition the interests of the user could be specified and calculated with weights, thus the sights would have a variable degree of importance for the user, and they could be presented with different parameters based on their importance.

Clearly the application designer has better control on the distance and the location that panoramic images are placed when user captured panoramic images are used. Even in this case, the different parts of the route will have different importance for a final user, and the quality of the images and their GPS distance could be selected differently to minimize the use of the system resources while offering good presentation quality. Experimentation with different parameters for image quality and image distances as function of the user satisfaction will help in the determination of the best values of these parameters.

References - bibliography

[1] Christodoulakis S., Foukarakis M., Ragia L., Uchiyama L., Imai H. : Semantic maps and Mobile Context Capturing for Picture Context Visualization and Management of Picture dDBMSs, Proceedings of ACM Mobile Ubiquitous Multimedia (MUM), Finland 2009.

[2] Christodoulakis S., Foukarakis M., Ragia L., Uchiyama L., Imai H. : Mobile Picture capturing for Semantic Picture Database Context Access, Browsing and Interaction, IEEE Multimedia 20010.

[3] S. Christodoulakis, P. Kontogiannis, P. Petridis, N. Moumoutzis, M. Anastasiadis, T. Margazas: MINOTAURUS: A Distributed Multimedia Tourism Information System

[4] B. Bederson, Audio Augmented Reality: A Prototype Automated Tour Guide, Proceeding CHI '95 Conference Companion on Human Factors in Computing Systems

[5] Christodoulakis S., Foukarakis M., Tsinaraki |C., Ragia L., Kanellidi, E. : Contextual Geospatial Picture Understanding, Management and Visualization, Proceedings 11th International Conference on Advances in Mobile Computing and Multimedia, MoMM 2013.

[6] G. Reitmayr, D. Schmalstieg: Collaborative Augmented Reality for Outdoor Navigation and Information Browsing, "Proceedings of the Second Symposium on Location Based Services and TeleCartography", TU Wien, 2004

[7] A. Ajanki, M. Billinghurst, H. Gamper, T. Järvenpää Melih Kandemir, S. Kaski, M. Koskela, M. Kurimo, J. Laaksonen, K. Puolamäki, T. Ruokolainen, T. Tossavainen: An augmented reality interface to contextual information, Virtual Reality - Special Issue on Augmented Reality Volume 15 Issue 2-3, June 2011

[8] P. Gronat, M. Havlena, J. Sivic, T. Pajdla: Building StreetView Datasets for Place Recognition and City Reconstruction, CTU–CMP–2011–01

[9] David Wells, "*Guide to GPS Positioning*", Canadian GPS Associates, Fredericton, N.B, Canada, 1986.

[10] G. D.Abowd, C.G.Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton: Cyberguide: A mobile context-aware tour guide, Journal Wireless Networks - Special issue: mobile computing and networking: selected papers from MobiCom '96

[11] B. Hoffman-Wellenhof, H. Lichtenegger, J. Collins "*GPS Theory and Practice*", Springel-Verlag Wien New York, 2nd edition, 1993

[12] B. Chen, B. Neubert, E. Ofek, O. Deussen, and M. F. Cohen, "Integrated Videos and Maps for Driving Directions," Proc. Symp. User Interface Science and Technology 2009, pp. 223-232.

[13] Zornitza Yovcheva, Dimitrios Buhalis, Christos Gatzidis: Overview of Smartphone Augmented Reality Applications for Tourism, e-Review of Tourism Research (eRTR), Vol. 10, No. 2, 2012

[14] J. Kopf, B. Chen, R. Szeliski, and M. Cohen, "Street Slide: Browsing Street Level Imagery," ACM Trans. Graphics, vo. 29, no. 4, 2010, Article no. 96, (Proc. SIGGRAPH 2010).

[15] J. Heleine: Photo Sphere Viewer, a JavaScript library to display Photo Sphere panoramas, https://github.com/JeremyHeleine/Photo-Sphere-Viewer

[16] Google Maps JavaScript API:
https://developers.google.com/maps/documentation/javascript/

[17] StreetView Service API:
https://developers.google.com/maps/documentation/javascript/streetview

[18] RESTful Java with JAX-RS:
https://books.google.gr/books?id=wzkYAgAAQBAJ&pg=PA209&lpg=PA209&dq=jax+rs+servlet+context&source=bl&ots=dnszYvUxny&sig=LFSHO2bK3WbHV1sdcD7oYctViN4&hl=en&sa=X&ved=0ahUKEwiZp9mC7YnYAhXBvxQKHUawD0gQ6AEISDAE#v=onepage&q=jax%20rs%20servlet%20context&f=false

[19] PostgreSQL The most advanced open source database: https://www.postgresql.org/

[20] Calculate distance, bearing and more between Latitude/Longitude points: [https://www.movable-type.co.uk/scripts/latlong.html]

[21] Ajax with Servlets using JQuery and JSON: [http://www.programming-free.com/2012/09/ajax-with-servlets-using-jquery-and-json.html]

[22] Exchangeable image file format for digital still cameras: Exif Version 2.2: [http://www.exif.org/Exif2-2.PDF]

[23] Google Camera:
[https://docs.google.com/document/d/1VlPIDuSYuhHyeumdW3bCyO9eFvuHBvLrZyVneDaeVFQ/edit]

[24] Haversine Formula: Distance between points on Earth Surface,
https://www.math.ksu.edu/~dbski/writings/haversine.pdf