COLLECTION, UNIFICATION AND PRESENTATION OF ALTERNATIVE TOURISM DATA

By

Evangelos-Achileios N. Vatikiotis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DIPLOMA DEGREE OF

ELECTRICAL AND COMPUTER ENGINEERING

Chania February 2019

Περίληψη

Ο εναλλακτικός τουρισμός αποτελεί ένα συνεχώς αναπτυσσόμενο ρεύμα στο χώρο του τουρισμού και αναφέρεται στο σύνολο μορφών τουρισμού και δραστηριοτήτων, οι οποίες προσελκύουν ανθρώπους με συγκεκριμένα ενδιαφέροντα. Χαρακτηριστικά παραδείγματα εναλλακτικού τουρισμού αποτελούν ο αγροτουρισμός, ο οικοτουρισμός, ο πολιτιστικός τουρισμός κ.α. Σκοπός της διπλωματικής εργασίας είναι η σχεδίαση και η υλοποίηση μιας διαδικτυακής εφαρμογής, η οποία συλλέγει, διαχειρίζεται και αναπαριστά τοποθεσίες ενδιαφέροντος (Points Of Interest) αθλητικού τουρισμού. Συγκεκριμένα, το σύστημα υποστηρίζει την συλλογή δεδομένων για τοποθεσίες ενδιαφέροντος διαφόρων κατηγοριών μέσω τεχνικών διαδικτυακής εξόρυξης (web scraping), την ενοποίηση τους με την έννοια της κληρονομικότητας καθώς και την αναπαράσταση των τοποθεσιών ενδιαφέροντος μέσω της εφαρμογής. Τέλος, η διαδικτυακή εφαρμογή προσφέρει στους χρήστες την δυνατότητα να εξερευνήσουν νέα σημεία ενδιαφέροντος αθλητικού τουρισμού, καθώς και την δυνατότητα να συμβάλουν στην αξιολόγηση τους.

Abstract

Alternative tourism is a constantly growing trend in the tourism sector and refers to all forms of tourism that attract people with special interests. Typical examples of alternative tourism are agrotourism, ecotourism, cultural tourism and others. The purpose of the diploma thesis is to design and implement a web application that collects, manages and presents Points of Interest related to athletic and active tourism. In particular, the system supports three operations related to athletic or active tourism. Firstly, it supports the collection for points of interest of various categories through web scraping techniques. Secondly it implements their Unification using the concept of inheritance. Last but not least, it presents the Points of Interest and offers to users the opportunity to explore new points of interest in sports tourism, as well as the ability to contribute to their evaluation.

Table of Contents

List of Figures

List Of Tables

1 *Introduction*

Alternative tourism is a constantly growing trend in the tourism sector and refers to all forms of tourism that attract people with special interests. Typical examples of alternative tourism are agrotourism, ecotourism, cultural tourism and others. The purpose of the diploma thesis is to design and implement a web application that collects, manages and presents Points of Interest related to athletic and active tourism.

Web applications have changed almost every aspect of modern life. Among other things, along with GPS technology and interactive maps, they have changed the way people navigate and search for specific locations with certain properties. This has led to the rise of importance for what is called the Points of Interest (POIs). In particular, tons of applications have been developed based on POIs. A POI is a specific point location that someone may find useful or interesting.

The way people interact with the web has led to the fed and massive growth of POIs existence across web. According to Google [1] google-maps platform contains over 150 millions of registered POIs and serves over a billion of monthly active users. Beside google tons of other applications have been implemented based on POIs.

The primary intention of the current thesis is to build a big POI Database for activities. Beside the collection of the various POIs, which is referred to as scraping, the intentions of the current thesis include the implementation of services to support both B2B and B2C services based on the collected POIs.

In more detail, the current thesis discusses the design and implementation of a System that performs three tasks. The first task is to design and implement an application that performs collection and modeling of data related to specific POIs from the web referring to specific categories. In view of the fact that POIs may reside in different categories, unification is also an aspect that has to be taken into consideration. The second task is to use server-side logic that is able to support both B2C and B2B services and persist the POIs model among with other entities of the system. The third task conforms to designing and implementing a state of the art web application in order for users to interact with the collected POIs.

Scalability is a very important aspect for our system. In particular it is important for all the parts of our system parts to be able to scale in terms of supporting new POIs, POI categories and supporting any future functionality needed. Firstly, the Scraping Application must be able to scrape, model and persist new POIs of new categories effectively. On the other hand, the rest of the system must be able to extend easily in order to support new POIs and categories as well as conform with future requirements.

In order to fulfill the System needs the implemented architecture completely decouples the scraping logic from the web logic making it possible to integrate new POIs to the web logic with minimum configuration.

It is intended for the whole System to grow big and support lots of POI categories, thus the Agile methodology approach was chosen to design and implement the whole System. In the Agile methodology software development and requirements specification is a non-stopping procedure looking to satisfy the client, ourselves in the particular situation. Thus versions of the System are designed, implemented and evaluated in order to define the next's version requirements. The current document specifies the first version of the System which support only scuba-diving and kitesurfing POIs. Its main goal, is to build a robust infrastructure characterized by scalability.

The implemented application is deployed on a TUC server and can be accessed on the following address: http://147.27.41.133/adventurer/

2 *Important Terms*

This chapter specifies important terms that are used along the document to describe the implemented framework. All of the following terms are state of the art methods for handling and implementing specific operations. Section 2.1 describes the term of web scraping and its importance for today's enterprises. Section 2.2, 2.3 and 2.4 discuss how modern web applications are built. In particular, section 2.2 describe the term of web services, section 2.3 describes Representational State Transfer specification and finally section 2.4 describes the state-of-the-art for today’s web applications .

2.1 Web Scraping

International Data Corporation (IDC) forecasts that by 2025 the global datasphere will grow to 163 zettabytes (that is a trillion gigabytes). That’s ten times the 16.1ZB of data generated in 2016. All this data will unlock unique user experiences and a new world of business opportunities. [2]

While data grows in amount, variety, and importance, enterprises keep in track with information from various sources and focus on data that matters the most. Several terms introduced and became important due to data growth. Some of them are Web Crawling, Web Scraping, Meta Data, Meta Data Analysis, Big Data, Data Mining and others.

It is usual for Web Crawling and Web Scraping to be combined in order to collect Big Data. Web Crawling is used for Web indexing and sources defining, whereas Web Scraping is used for data extraction. The current thesis focuses on one of the aforementioned Methodologies and more specifically on Web Scraping. Figure 2.1 describes the two processes.

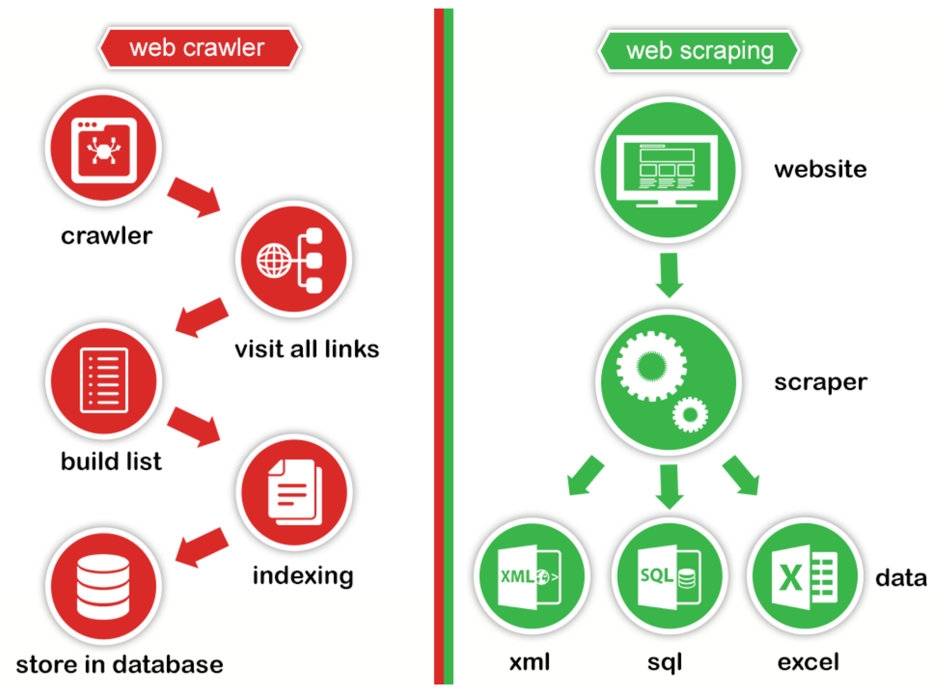


Figure 2.1: Web crawler vs web scraper[Web Scraping and Web Crawling – Author: Santosh Kalwar]

Data scraping, also known as web scraping, web extraction or harvesting, is a technique to extract data from the WWW and save it to a file system or database for later retrieval or analysis. Commonly, a web page may be reached utilizing the Hypertext Transfer Protocol (HTTP). The page retrieval is accomplished either manually by a user or automatically by a bot or web crawler. After page retrieval scraping methods may take place.

Due to the fact that an enormous amount of heterogeneous data is constantly generated on the WWW, web scraping is widely acknowledged as an efficient and powerful technique for collecting big data [2].

2.1.1 Methods of Web Scraping

The methods of Web Scraping evolved together with the WWW. Not all listed methods were available at the beginning. A programmer might use various methods in order to make Requests and retrieve data from a Web Server in various formats. The most common protocol for such a case would be the Hypertext Transfer Protocol (HTTP) or the HTTP Secure (HTTPS). After data retrieval the following methods could be applied.

Manual Scraping

Manual Scraping is the process where a user uses a web browser to extract data manually from web pages. Manual scraping is an option in specific situations where a small amount of data is required. Manual scraping is not an effective way to extract large amount of data, but in some cases may be useful.

HTML Parsing

Hypertext Markup Language (HTML) is the language for describing the structure of Web Pages [4]. Assuming that an HTML document is available from a server response, analysis on the structure of the document could show repeated HTML elements. Then with further analysis, selecting specific HTML elements and their value is possible.

DOM Parsing

The Document Object Model (DOM) introduced new ways of addressing HTML Elements. Programs and scripts implementing DOM are able to access and update the content, structure or style of HTML Elements dynamically. DOM parsing takes advantage of those ways in order to access the data. DOM parsing is a very effective way to perform Web Scraping because it enable many ways to access elements of a Web page.

Screen Scraping

This technique is used for saving and reusing entire or parts of web pages. Screen Scraping can be performed by just HTTP Programming or by combining it with DOM Parsing.

Server Side APIs

Server side APIs is a way companies or some entity lets you interact with their system. For instance, the Google Maps API is an interface for programmers to build applications on the Google Maps platform. Another example is openweathermap API which provides real time weather information in JSON or XML format. In most cases Server Side APIs provide data in a well structured way for developers to manipulate and manage them. Additionally, server side APIs include documentation explaining developers how to perform their requests in order to collect these data.

APIs are used a lot nowadays and whole applications are based on live feed from third party APIs offering B2B services.

2.1.2 Software and Tools

Since the evolution of the WWW and the use of dynamic content, Web Scraping processes are challenging and demanding. Furthermore, in view of the fact that Web Scraping is crucial for B2B and B2C processes, a variety of applications, programming frameworks and libraries have been published in order to keep in track with the scraping requirements of today's applications.

Web Scraping Cloud Applications, provide a user interface to the end user through a Web Browser while the application back end resides on the cloud server. Such a configuration limits the hardware requirements of end user to their minimum. As a result, large projects can be created and big data retrieved without additional hardware or extended internet bandwidth. Some available cloud related applications for Web Scraping are: Dexi.io and Octoparse.

In Desktop Applications developed for Web Scraping Web data are downloaded, parsed and saved locally. This process has considerable hardware and internet requirements. Examples: ParseHub, Fminer.

Even though Ready built Web Scraping applications and tools are powerful and capable of executing various tasks, in some cases a custom Web Scraping solution might be necessary. In such a case developers can use generic functionality provided by programming Libraries or Frameworks.

Scrapy

Scrapy is a web crawling framework for developer to write code to create spider, which define how a certain site (or a group of sites) will be scraped. The biggest feature is that it is built on Twisted, an asynchronous networking library. Using Twisted Scrapy is using a non-blocking (asynchronous) code for concurrency, which makes the spider performance very great [4].

Selenium

Selenium is a tool for writing automated tests for Web applications through scraping and further processing. It is a good option because it is powerful and beginner friendly. It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Groovy, Java, Perl, PHP, Python, Ruby and Scala.

Jsoup

Jsoup in an open source Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. Jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do [6]. Jsoup was one of the tools used to perform the scraping process.

Browser Extensions

Browser extensions like Outwit Hub or Web Scraper are also used but have limited potentials in comparison with aforementioned technologies.

2.2 *Web Services*

Web Services enable machine to machine communication by providing messages with machine readable content formats. Corresponding to Web applications, Web Services provide machine readable content to clients and have become the leading method to support client-side applications and communicate with other services. Particularly, web services provide an API to rich clients or other web services to interact with the system resources. Because of the machine readable content web services are often used for both B2C and B2B services.

Service Oriented Architecture (SOA) and Resource Oriented Architecture (ROA), are both architectural design patterns and the corresponding distributed programming paradigms. Both provide a conceptual methodology and development tools for creating distributed web services [7]. Distributed web services are regularly implemented nowadays to increase server side performance, fulfill the WWW needs and face a range of challenges like handling numerous requests and large amount of data in a reasonable time.

Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP and Web Services Description Language (WSDL) based Web services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers including Yahoo, Google, and Facebook who have deprecated or passed on SOAP and WSDL-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services [7].

2.3 Representational State Transfer (REST)

REST defines a loose client – server interaction whereas SOAP defines a more strict way for client – server communication.

REST is a Resource Oriented Architecture (ROA), therefore it is an architecture for designing web services based on resources. It defines how resource states may be addressed, processed and transferred over the HTTP protocol.

Web services implementing REST should fulfill some principles and guidelines. Firstly, resources exposure to the "outside world" is done in directory structure URIs. URIs are used in order to reference a system resource. In REST each system's resource state has its URI and each URI is used to retrieve, modify or delete resource data [8].

Furthermore, REST uses HTTP methods explicitly as described in HTTP/1.1 version of HTTP protocol. While each resource is linked with a URI, each CRUD operation is linked with an HTTP method. Therefore, GET requests refer to read, POST to create, PUT to update and DELETE to delete CRUD operations.

Combining URIs and HTTP methods as described, REST uses stateless interactions between the server and the client. In these interactions HTTP requests hold all the information needed by the server to execute the appropriate CRUD and computational operations to generate the right response. To comply with security issues the services consumers use expiring web tokens or API keys given by the server and session storage happens on client side.

Last but not least, HTTP request/response payloads are transferred in XML or JSON format.

2.3.1 RESTful Web Services

Web services must conform with all of the above conditions in order to be "RESTful". Because of the stateless and self defined requests REST makes the development a lot simpler in comparison to SOAP. Both client – server communication is easier to implement and load balancing is a lot simpler. Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers. Server proxies just forward requests to RESTful services while in SOAP simple forwarding is not enough as requests are not self defined.

2.4 Single Page Applications (SPAs)

A Single Page Application (SPA) is composed of individual components that can be replaced or updated independently without refreshing the whole page. In other words, the entire page does not need to be reloaded on each user action, which saves bandwidth as well as no loading of external files, like CSS files, every time the page is loaded. The purpose behind this is to make the subsequent page loads very fast, compared to traditional Request-Response cycle. SPAs are written in JavaScript programming language, and support HTML5 and AJAX calls. [10]

Web applications could be built with just plain JavaScript which is often referred as Vanilla JavaScript. The result would be a complex and difficult to maintain web application. SPAs frameworks are used in order to make the make the development and maintenance easier and more efficient. As a result, SPAs are getting a lot the likes of developers and enterprises for building their web applications.

2.5 State of the Art for Web Applications

The concept of dynamic web applications, where asynchronous client-server communication is implemented, introduced a series of advantages and terms to Web Development. Advantages like better UX, responsiveness and faster web applications and terms like AJAX, SPAs, DOM and Web Services.

When referring to asynchronous client–server communication, client invokes requests to the server based on user actions. In traditional web applications, user used to wait for the response and the reload of the whole page to perform more actions. In modern applications AJAX requests occur behind the scenes letting the user make other actions while waiting for the response. The server response is in machine readable format like XML or JSON and when it is retrieved it is consumed by the client-side application which make corresponding UI changes. JavaScript is used for creating the AJAX request, handle the response and make UI changes via DOM manipulation.

Web development was driven by fast networking, hardware evolution, AJAX and the ability to perform business logic on both the client side and the server side to building Rich Internet Applications (RIAs). RIAs promote web applications as desktop applications performing business logic operations on both the server and the client side. While client side applications are implemented with SPAs frameworks, server side applications are implemented with web services.

Web services run on the server and provide SPAs with an API to interact with various states of server resources. In greater detail, web services handle AJAX requests, access resources and generate the appropriate response for the request in machine readable format.

Figure 2.2 shows accordingly the traditional Client - Server communication cycle and the modern SPA Client – Server communication cycle. [11]

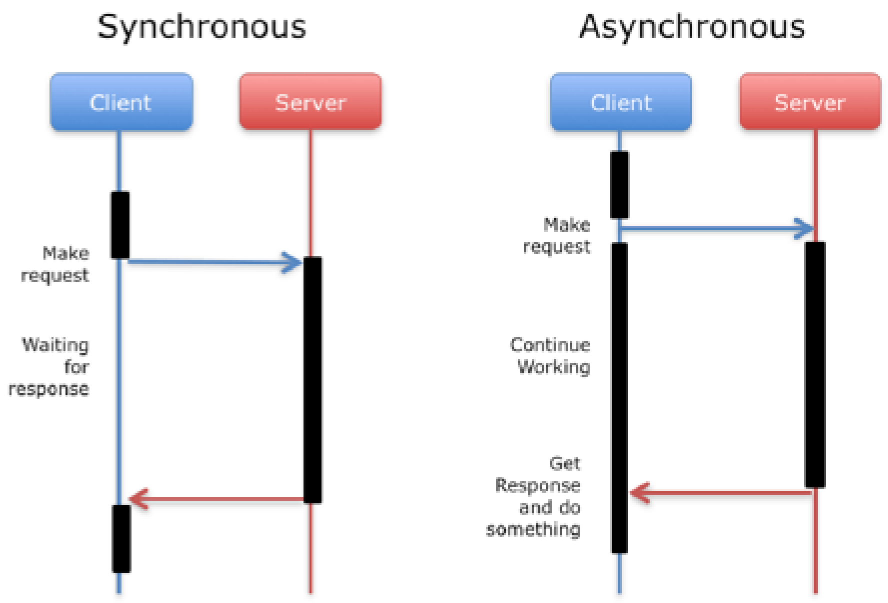


Figure 2.2: Synchronous vs Asynchronous client-server communication

3 *Implementation Technologies*

As follows the system’s core technologies are described. Section 3.1 describes the Spring Framework, section 3.2 describes Spring Boot, Spring’s extension and section 3.3 describes AngularJS SPA framework. It is worth to mention that for the source control management of our system multiple Git repositories were used hosted on github.

3.1 Spring Framework

Spring is an open-source java based framework. It is used for implementing lightweight, cross-platform and extensible J2EE architectures. It has become popular to Java community as an alternative to the Enterprise JavaBeans (EJB) model.

Spring was originally created by Rod Johnson and was first described in his book Expert One-on-One: J2EE Design and Development. Spring was created to address the complexity of enterprise application development, and makes it possible to use plain-vanilla JavaBeans to achieve things that were previously only possible with EJBs. But Spring’s usefulness isn’t limited to server-side development. Any Java application can benefit from Spring in terms of simplicity, testability, and loose coupling [12].

Spring provides comprehensive programming functionality for developing J2EE based applications. In particular, Spring’s Web component separates the application’s logic into different architectural tiers. It requires minimal configuration and enables developers use Spring modules at will, combining them in a way that suites the application being developed.

3.1.1 Inversion of Control and Dependency Injection

The most identifying concept of Spring is the Inversion of Control (Ioc). Under IoC concept, framework’s code invokes application code and coordinating the overall workflow, rather than application code invoking framework code. Spring accomplishes IoC with Dependency Injection (DI). DI is based on just Java construction methods and custom interfaces, rather than the use of framework specific interfaces.

Instead of application code using framework APIs to resolve dependencies such as configuration parameters and collaborating objects, application classes expose their dependencies through methods or constructors that the framework can call with the appropriate values at runtime, based on configuration. Dependency Injection is a form of push configuration; the container "pushes" dependencies into application objects at runtime. This is the opposite of traditional pull configuration, in which the application object "pulls" dependencies from its environment [13].

3.1.2 Spring Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test layers. Spring's modules are presented in figure 3.1

[14]

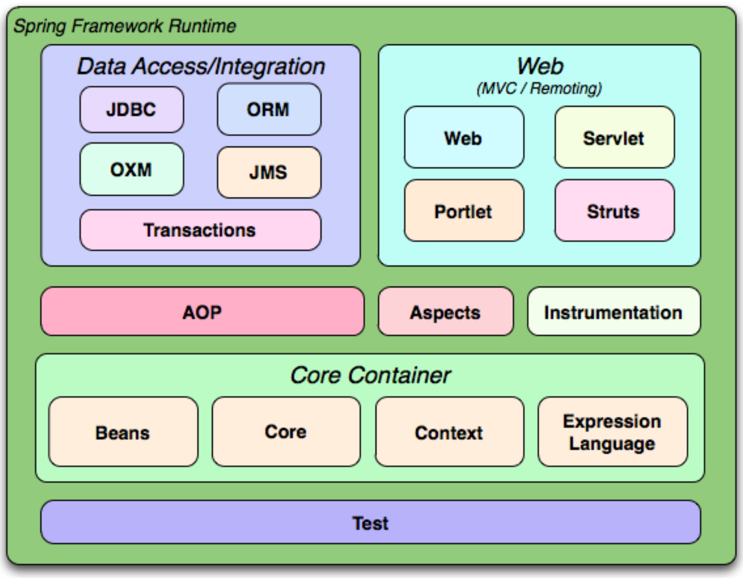


Figure 3.1: Spring Modules

Core Container

Core Container manages Bean creation and usage across the application. Core Container is Spring's coordinator, it is the portion of Spring that provides DI and defines the "scope" of a bean across other beans. Spring provides several bean scope configurations such as singletons, which scopes a single bean definition to a single object instance per Spring IoC container or prototypes, which scopes a single bean definition to any number of object instances.

Web/Remoting

Spring's Web and Remoting components implement Spring's MVC framework. It comes in either a servlet-based framework for conventional web applications or a portlet-based application for developing against Java Portlet API. The main way in which portlet-based differs to servlet-based is that the request to the portal can have two distinct phases the action phase and the render phase; for a single overall request, the action phase is executed only once, but the render phase may be executed multiple times. The current thesis is implemented based on the servlet-based framework.

In Spring the HTTP request is a busy fellow. From the time it reaches the server, it will make several stops, each time dropping off a bit of information and picking up some more. Figure 3.2 shows all the stops the request makes [12].

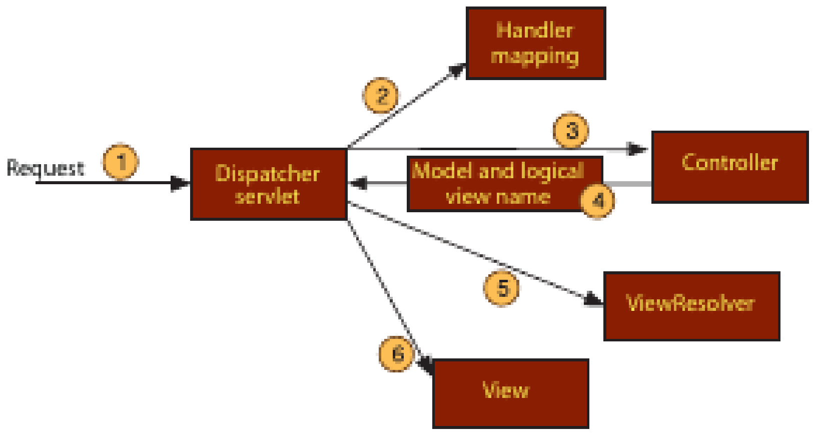


Figure 3.2: HTTP Request workflow in Spring framework

Spring MVC, like many other web frameworks, is designed around the front controller pattern where a central servlet, the DispatcherServlet, provides a shared algorithm for request processing while actual work is performed by configurable, delegate components[15].

Like any other Servlet, the DispatcherServlet needs to be declared and mapped to appropriate methods. In fact, the Dispatcher has certain bean instances in his scope defined in configuration. It uses HandlerMapping Bean to map the request to the right Bean. The DispatcherServlet maps the request to singleton based controller beans.

Web layer also provides several remoting options for building applications that interact with other applications. Spring's remoting capabilities include Remote Method Invocation (RMI), Hessian, Burlap, JAX-WS, and Spring's own HTTP invoker [16].

Data Access/Integration

The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores. Entity beans both enjoyed the rise and suffered the fall of EJB's popularity. In recent years, developers have traded in their heavyweight EJBs for simpler POJO-based development. This presented a challenge to the Java Community Process to shape the new EJB specification around POJOs. The result is JSR-220—also known as EJB 3. The Java Persistence API (JPA) emerged out of the rubble of EJB 2's entity beans as the next-generation Java persistence standard. JPA is a POJO-based persistence mechanism that draws ideas from both Hibernate and Java Data Objects (JDO), and mixes Java 5 annotations in for good measure. With the Spring 2.0 release came the premiere of Spring integration with JPA. The irony is that many blame (or credit) Spring with the demise of EJB. But now that Spring provides support for JPA, many developers are recommending JPA for persistence in Spring based applications. In fact, some say that Spring-JPA is the dream team for POJO development [15].

The Data Access/Integration layer or Spring Data consists among others of the JDBC and ORM modules.

JDBC module is a JDBC-abstraction layer that removes a lot of boilerplate code to get a connection, create a statement, process the result set, close a connection or handle SQL error messages.

The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, and Hibernate. Using the ORM module you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature [17].

Test

Automated testing has become a very important thing in application development. Automated application testing and application development often go side by side, referred to test-driven development, having test methods for each every feature and functionality the application provides providing feedback for the whole application.

Spring provides a component dedicated to testing Spring applications. It provides functionality for both unit and integration testing methods to be implemented.

3.2 *Spring Boot*

Spring Boot is an open-source framework introduced in 2003. It is used for developing Spring based applications with minimal configuration. Spring itself is characterized by "lightweight" component code, but it requires "heavyweight" XML and JavaBean configuration. Spring Boot reduces a lot Spring's configuration. Using annotations, starter dependencies and auto-configuration it provides a great way to configure and bootstrap a Spring application.

Spring starter dependencies describe the functionality the application needs. Spring Boot provides a big list of starter dependencies to configure specific functionality. As follows some of Spring's starter dependencies are described. Web starter dependency auto-configures both an embedded tomcat server and spring MVC. Data JPA starter dependency enables code written based on Hibernate and Object Relational Model implementing the Java Persistence specification. Moreover, Starter parent dependency, among other utilities, manages the versions of common dependencies. This dependency management ensure compatibility across the dependencies and let developers using spring omit version configuration for a variety of supported dependencies.

Annotations on the other hand, are used for JavaBeans configuration. In Spring beans used to be configured using boilerplate XML code.

3.3 *AngularJS*

AngularJS is an open-source framework packed with several utilities and services typically needed in single-page web applications [18]. It provides functionality for defining a tidy Model View Controller (MVC) architectural pattern. Applications written in AngularJS are cross-browser compliant as AngularJS automatically handles JavaScript code suitable for each browser [17].

In AngularJS the application's logic gets divided in different components and sub-components each one with it's own tasks. It provides developers with options of creating generic and reusable components in order to minimize the development code and make maintenance a lot easier.

AngularJS componentized structure specification decouples the view from the logic using a two way data binding between the model and the view. In more detail, AngularJS directives instantiated in the view, worry about displaying the referenced model values and AngularJS worry about updating the model when the values change. This enables developers work with the model, via angular controllers, without worrying about how the data is displayed or entered [18].

Figure 3.3 shows the workflow of an application developed with AngularJS.

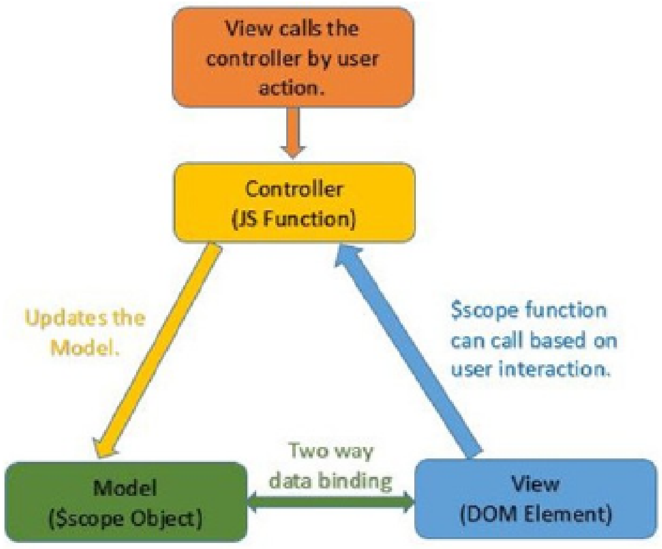


Figure 3.3: AngularJS Workflow

3.3.1 AngularJS Core Features

The core features of AngularJS are presented in figure 3.4 [19] and most of them are described as follows in greater detail.

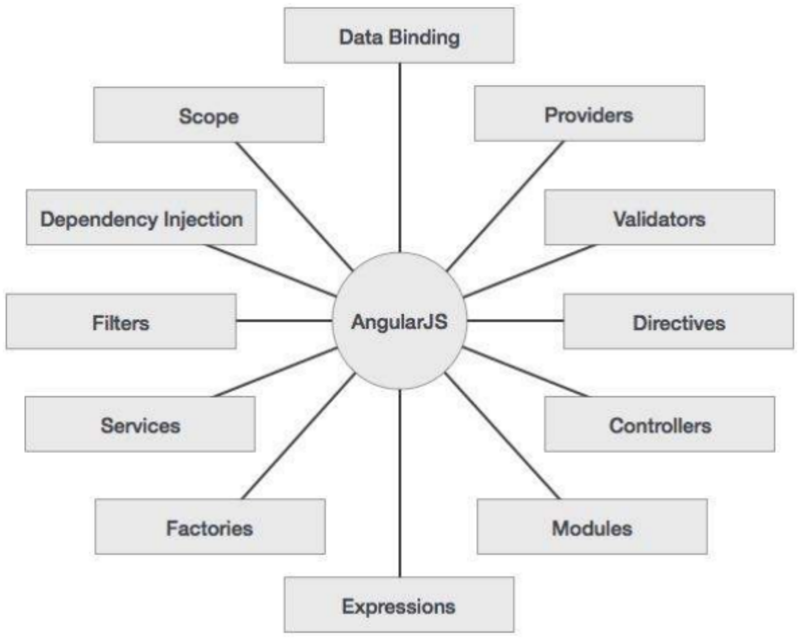


Figure 3.4: AngularJS features

Data binding

Data-binding in Angular apps is the automatic synchronization of data between the model and view components. The way that Angular implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change, and vice versa.

Templates

A template in AngularJS is HTML code that contains Angular-specific elements, the directives. Angular combines the template with the model to render the dynamic view a user sees on the browser.

Controllers and Directives

Controllers and directives is the way angular interlinks the template with the model in order to render the page.

Controllers are JavaScript functions used to provide the model to the template. In particular, when a Controller is attached to a DOM element via the ng-controller directive, angular uses a constructor function to augment the application’s model and provide the child $scope generated to the template. The child $scope is the local state of the specified controller and can be accessed by the template. As the application’s state, local state is a plain JavaScript object and may contain variables, arrays, key – value pairs or functions.

Directives, as specified, are angular-specific elements. They are used by the templates in order to access the model provided by controllers.

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS’s HTML compiler ($compile) to attach a specified behavior to that DOM element or even transform the DOM element and its children [17].

Model

The Model in AngularJS is a plain JavaScript object. The Model, as JavaScript objects, may contain variables, arrays, key – value pairs or functions. The application's model could be also referred as the state of the application defined in $rootScope variable. $rootScope is arranged in hierarchical structure which mimics the DOM structure of the application. On initialization, AngularJS instantiates the application’s model or the application’s global state, the $rootScope. $rootScope is the parent of all local states ($scopes), that is the way AngularJS “knows” which templates to change when the model state changes or vice versa, making both dynamic DOM manipulation and UX more efficient.

As components may exist inside other components a possible application’s state might look like figure 3.5 [20].

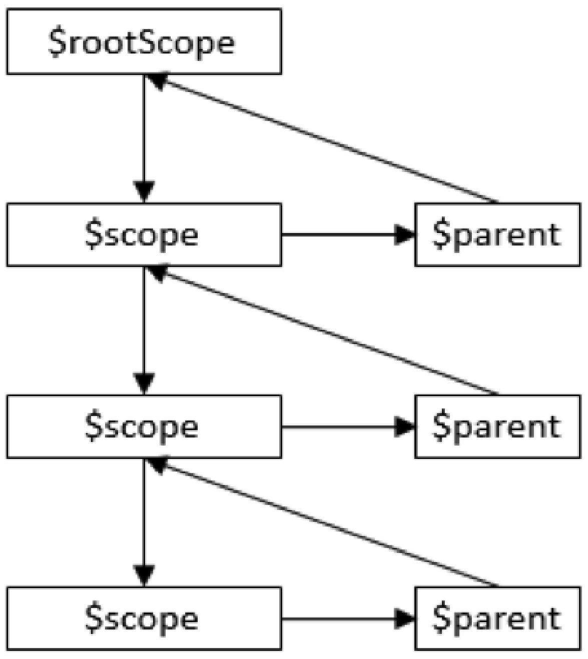


Figure 3.5: Possible angularJS global state

Dependency Injection

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies. The AngularJS injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested [21].

Services

Angular services are substitutable objects that are wired together using Dependency Injection (DI). Services are used to organize business logic and share it across the controllers. The Angular services are:

- Lazily instantiated - Angular only instantiates a service when an application component depends on it.
- Singletons - Each component dependent on a service gets a reference to the single instance generated by the service factory.

3.4 Data Persistence

For data persistence PostgreSQL was used. Postgres is an old time classic Relational Database Management System (RDMS). Over the years PostgreSQL has earned a strong reputation for reliability, robustness and performance.

PostgreSQL, also known as Postgres, was originally developed in 1986, by Michael Stonebraker and his team at the University of California at Berkley (UCB). Today, Postgres is an open-source Relational Database Management System (RDBMS) managed primarily by EnterpriseDB and monitored by Postgres development community. The code base takes regular contributions from the community and over the years has become a powerful extensible system free to purchase and modify[22].

Postgres was not only chosen because it is great as a Relational Database, but it has also added NoSQL support which may be an interesting feature for the scalability of our system.

4 *System Architecture*

The whole system is designed to collect information for POIs capable of hosting certain experiences or types of activities, that our system supports, and provide these information to rich clients in order to build beautiful Web Applications. This chapter describes the overall system architecture, identifies its basic components and specifies the functionality of the implemented components. The system architecture is presented in Figure 4.1.

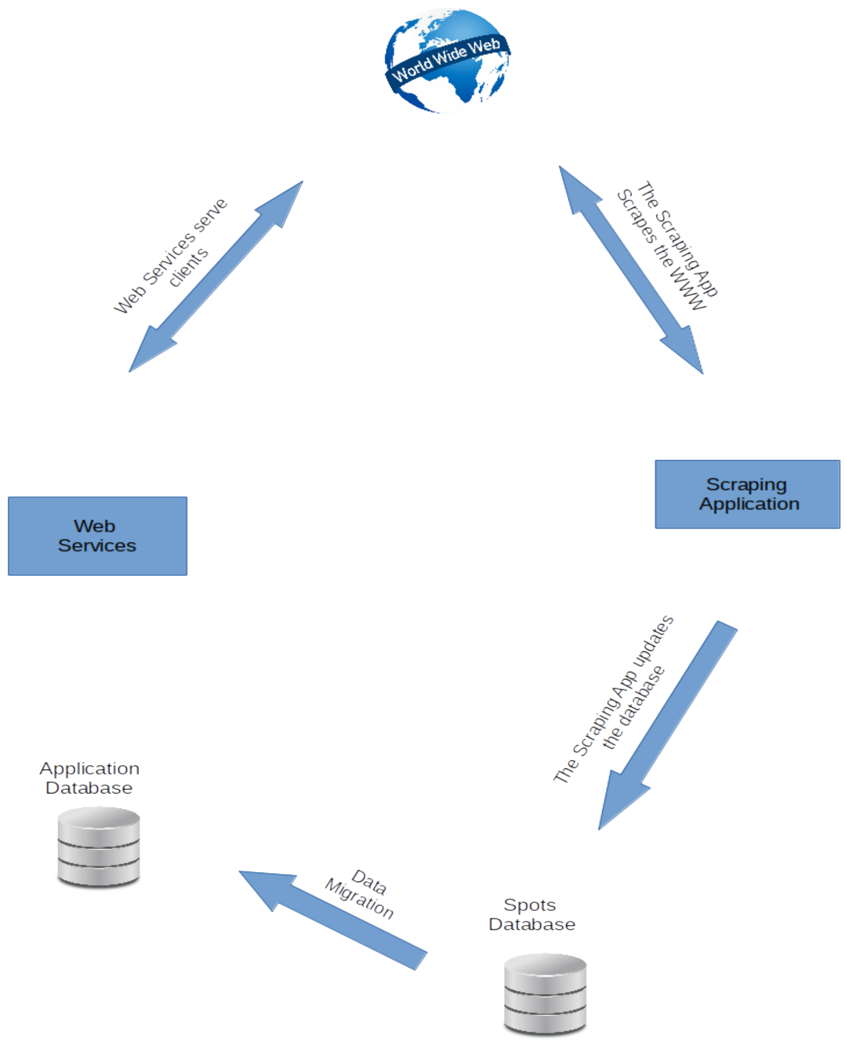

World Wide Web
Web Services serve clients
The Scraping App Scrapes the WWW
Web Services
Scraping Application
The Scraping App updates the database
Application Database
Data Migration
Spots Database

Figure 4.1: System Architecture

Scraping Application

The Scraping Application is responsible for fetching and modeling suitable information for our system. Modeling is an important aspect of the whole system described in detail in chapter 5. Scraping Application defines the POIs model which is persisted along the application. That way any new scraped POIs complying with the POIs model can be integrated in our application with minimum configuration.

POIs Database

The POIs Database persists all the information-data the Scraping Application gathers. In particular, it contains POIs in a pure state as gathered and handled by the Scraping Application and provide an infrastructure for the Application Database.

Application Database

The Application Database is an “extension” of the POIs Database. It contains all POIs that exist in POIs Database alongside with user-POIs relations like favourite POIs and the rating performed from a user on a specific POI.

RESTful Web Services

RESTful Web Services have been developed to support the client side application. The implemented web services run on the server and provide an API to clients like described in Section 3.2.

Client

On the client side resides and runs a SPA framework that consumes the API supported by the Web Services. As specified, the implemented client-side application is just an example of the API consumeness.

5 *Functional Specification*

This chapter describes the functional specification of the platform that have been developed. Section 5.1 specifies the technical requirements that had to be fulfilled while Section 5.2 discusses the functionality that had to be provided by our system in the form of use cases. It is worth mentioning that the development process started after the whole functional specification and design process were finished.

5.1 Technical Requirements

This section discusses the technical requirements that were identified and set for the development of the whole system. While each of the following sections specify the requirements of the system's basic components Section 8 describes the implementation of them.

A strong requirement that we set for our infrastructure is that the scraping logic should be completely decoupled from web logic. The solution came with the implemented Model specified in 5.1.1. Section 5.1.2 describes the requirements of the Scraping Application. Section 5.1.3 describes the requirements of the Client Side Application. Finally, section 5.1.4 describes the requirements our system's backend.

5.1.1 Model

As mentioned the model is an important aspect of the whole system, as it is the solution for the Scraping Application to co-exist with the rest of the system. The Model consists of two basic entities the POI and the User. In order to fulfill the needs of the system two requirements where specified:

- POIs should be easily extended, in terms of adding new POIs

- It should support data scalability. In greater detail, User-POI relations should be easily modeled and neither affected when adding new POIs nor when defining relations between users and new POIs.

5.1.2 Scraping Application

The requirements that we have set for the current version of the Scraping Application are to collect through scraping and persist the following:

- kitesurfing POIs
- scuba-diving dive sites
- scuba-diving schools

Another requirement specified as an outcome of a UI evaluation is the conversion of geographic coordinates to address. Before the evaluation, we used to provide search (see use case 10) via just POI name. After the evaluation and implementation of the specified conversion, the current version supports search via either POI name or POI address.

Last but not least, the scraping application should be able to implement new scraping services easily in order to conform with the evolution of the whole system.

5.1.3 Client Side Application

The intention for the Client Side Application was to build a rich application that is in line with the state of the art described in chapter 2.5. As a result, it should be a desktop-like, responsive and cross-browser compliant application and fulfill all of the following:

- The system must support two user roles (Actors): The Guest user and the signed-in user. The signed-in user will be referred as User and Guest user as Guest.
- The System must support registration and the password should be encrypted in the database.
- Basic authentication with password encryption is required for logging in.
- Actors must be able to interact with a dynamic map, see and search for POIs they are interested in.

- Users must be able to do more actions like rating or placing a POI to favourites.
- Users must have a profile.
- User must be able to see all POIs in interactive maps.
- User must be able to see all POIs in a list. Furthermore, the user should be able to apply meaningful filters and sorting on the POIs list.
- Each POI must have its details page where all of the POI's details should be presented.
- User must be able to rate a POI and the POI’s average rating should be provided properly.
- Users should be able to store their favourite POIs.

5.1.4 Server Side

The requirements that we have set for the backend of our system serving the client side application are summarized below:

- Provide state-of-the-art standards.
- able to support the persistence of users, POIs and user-POI relations.
- expose restful web services for allowing the access of the data from external applications.
- Perform efficient searching on data.
- Support scalability in terms of defining new services complying with the evolution of the whole system.

5.2 Use Cases

A use case describes the system’s behavior under various conditions as the system responds to a request from one of the actors, the Guest user and the logged in user.

Along the rest of the document the Guest User is referred to as Guest and the logged in User is referred to as User.

The actor initiates an interaction with the system to accomplish a goal. The system responds appropriately.

This section describes the system's behavior under its interaction with the actors. To this end, we exploit the method of use cases as they have been described by Alistair Cockburn [23]. For more understanding we present all of the use cases clustered in form of use case diagrams. Figures 5.1, 5.2, 5.3, 5.4 and 5.5 present the common use cases for both Guests and Users while figures 5.6 and 5.7 describe User only use cases. For each use case a dedicated table describes the internal functionality.

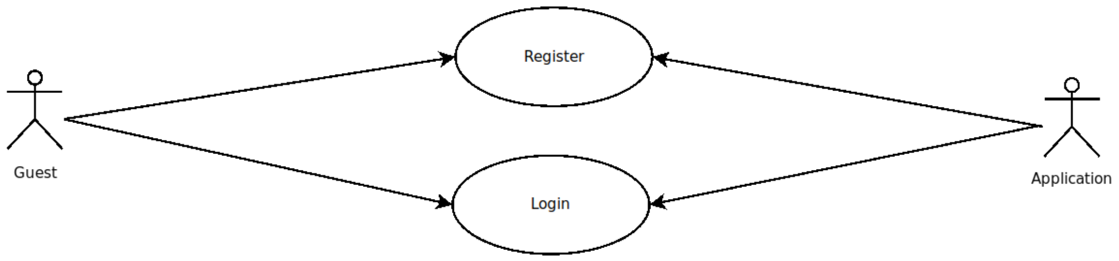


Figure 5.1: Guest only Use Cases

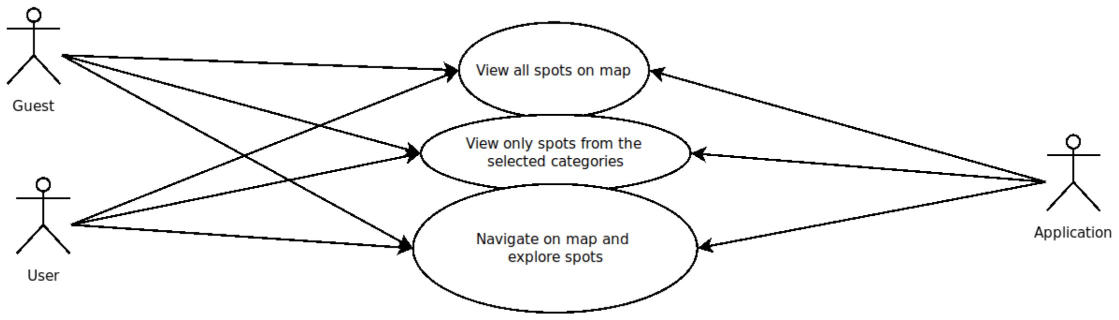

View all spots on map
View only spots from the selected categories
Navigate on map and explore spots
Guest
User
Application

Figure 5.2: Guest and User common Use Cases group 1

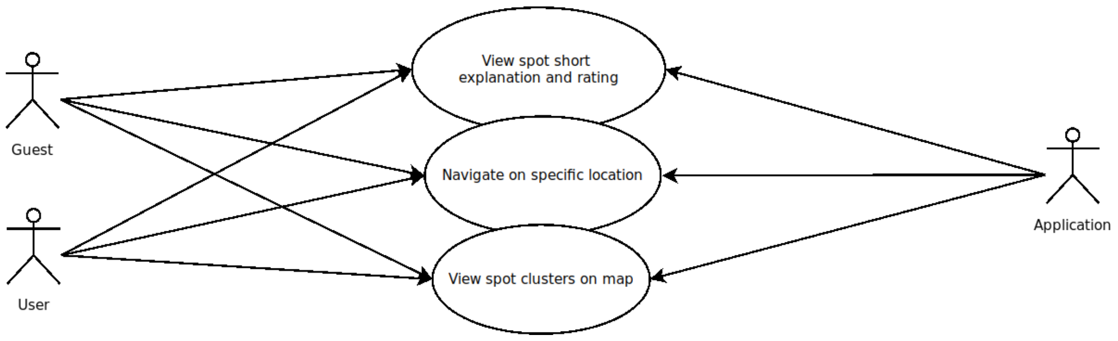

Guest
User
View spot short explanation and rating
Navigate on specific location
View spot clusters on map
Application

Figure 5.3: Guest and User common Use Cases group 2

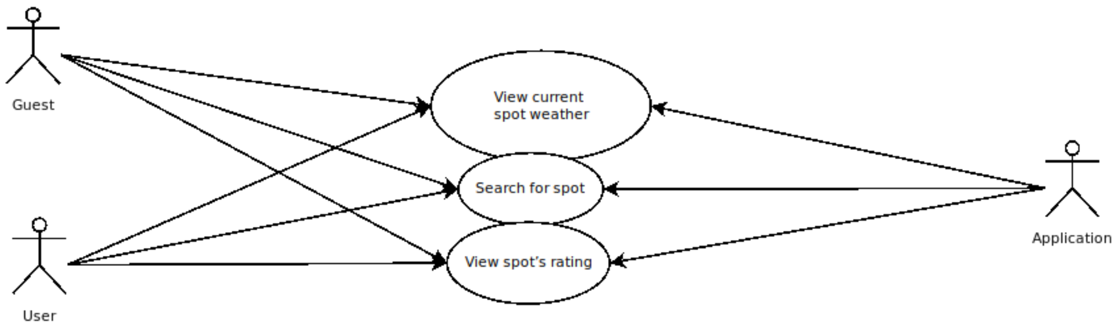


Figure 5.4: Guest and User common Use Cases group 3

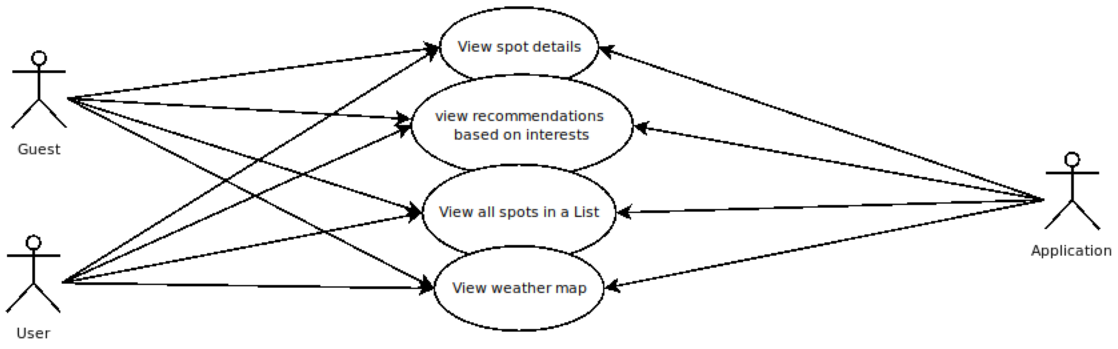


Figure 5.5: Guest and User common Use Cases group 4

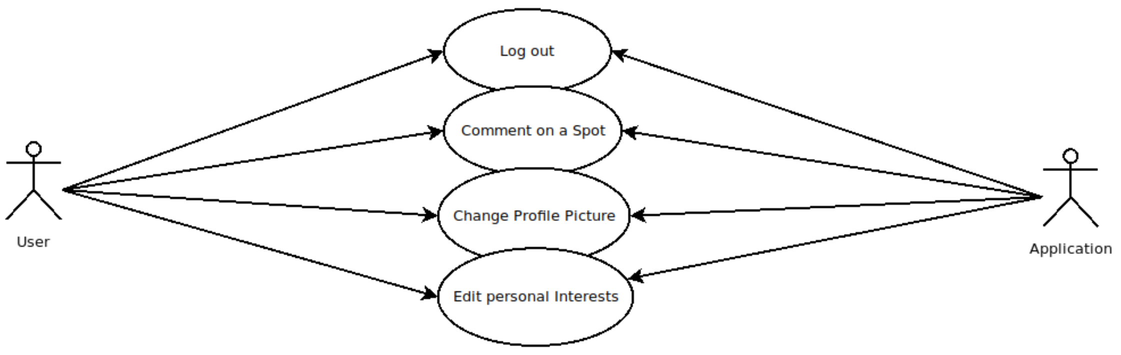


Figure 5.6: User only Use Cases group 1

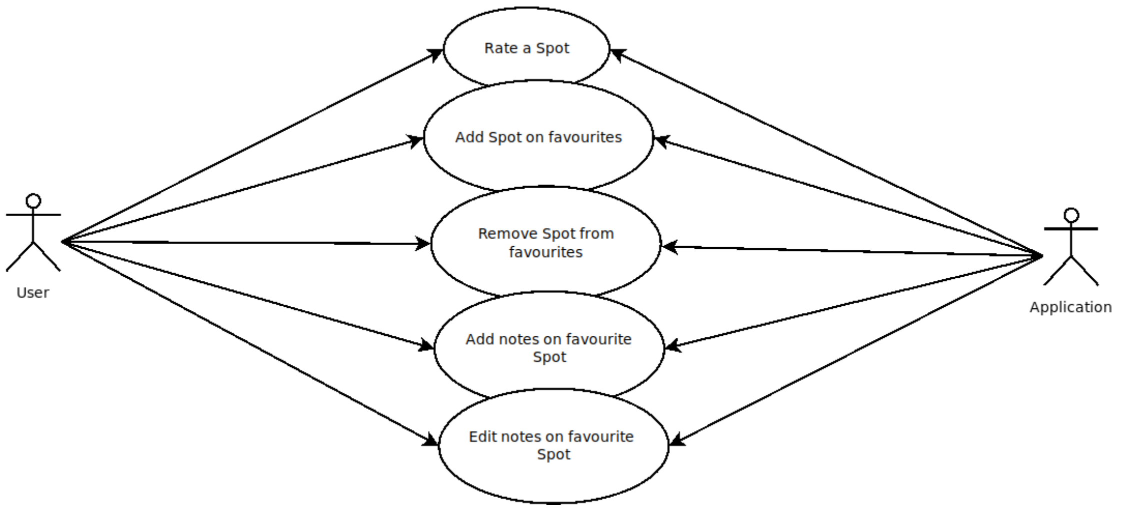

Rate a Spot
Add Spot on favourites
Remove Spot from favourites
Add notes on favourite Spot
Edit notes on favourite Spot
User
Application

Figure 5.7: User only Use Cases group 2

Table 1: Use Case 1: View all POIs on map

Use Case 1: View all POIs on map	
Context of Use	The User wants see all POIs on a dynamic map
Scope	System (Client Side Application)
Level	Summary
Preconditions	The System has load the application
Success End Conditions	The User sees all available POIs on a dynamic map

Failed End Conditions	The User doesn’t see all available POIs on a dynamic map	
Primary Actors **Secondary Actors**	User System (Client Side Application)	
Trigger	User selects to see the main map page from the navigation bar	
Description	**Step**	**Action**
	1	User clicks POIs on Map button from the navigation bar
Extensions	**Step**	**Branching Action**

Table 2: Use Case 2: View only POIs from the selected categories

Use Case 2: View only POIs from the selected categories		
Context of Use	User wants to see only POIs he is interested in	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has load the main map page	
Success End Conditions	User sees only POIs from the selected categories	
Failed End Conditions	1. User does not see POIs 2. User does not see only POIs related to the selected categories	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User selects a POI category	
Description	**Step**	**Action**
	1	The System shows all POIs related to selected categories
	2	User selects/deselects a POI category
Extensions	**Step**	**Branching Action**

Table 3: Use Case 3: Explore POIs on main map

Use Case 3: Explore POIs on main map		
Context of Use	User wants to explore Greece and interesting POIs through a dynamic map	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has loaded the main map page	
Success End Conditions	User explores Greece through a dynamic map	
Failed End Conditions	User is unable to explore Greece through the dynamic map	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User uses the dynamic map's interface to zoom in/out and move the map	
Description	**Step**	**Action**
	1	System loads the main map page
	2	User triggers the dynamic's map evnets
Extensions	**Step**	**Branching Action**

Table 4: Use Case 4: View POI short explanation and rating

Use Case 4: View POI short explanation and rating		
Context of Use	User wants to see an overview of a specific POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has loaded the main map page	
Success End Conditions	User sees a specific's POI short explanation and rating	
Failed End Conditions	User does not see a specific's POI short explanation and rating	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User clicks on a specific POI	
Description	**Step**	**Action**
	1	User clicks on a POI's marker
	2	The system shows the POI's short explanation and rating in a pop up window
Extensions	**Step**	**Branching Action**

Table 5: Use Case 5: Transfer on specific location

Use Case 5: Transfer on specific location		
Context of Use	The user wants the map to be transferred on a specific location	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has loaded the main map page	
Success End Conditions	The dynamic map gets transferred on a specific location	
Failed End Conditions	The dynamic map does not get transferred on a specific location	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User texts on a provided input and submits	
Description	**Step**	**Action**
	1	User texts on provided search input
	2	User submits the texted searched term
Extensions	**Step**	**Branching Action**

Table 6: Use Case 6: Cluster click

Use Case 6: Cluster click		
Context of Use	User wants to see what is underneath a marker cluster	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has loaded the main map page	
Success End Conditions	The map zooms in with center the cluster clicked	
Failed End Conditions	The map does not zoom in on cluster click	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User clicks on a POI cluster	
Description	**Step**	**Action**
	1	User clicks on a marker cluster
	2	The system makes the dynamic map to smoothly zoom in with center the clicked cluster
Extensions	**Step**	**Branching Action**

Table 7: Use Case 7: View all POIs in a list

Use Case 7: View all POIs in a list		
Context of Use	User wants to see all available POIs in a list	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The System has loaded the application	
Success End Conditions	User sees POI Finder page	
Failed End Conditions	User does not see POI finder page	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User selects to see the POI Finder page from the navigation bar	
Description	**Step**	**Action**
	1	User clicks POI Finder button from the navigation bar
Extensions	**Step**	**Branching Action**

Table 8: Use Case 8: View POI details

Use Case 8: View POI details		
Context of Use	User wants to see the details of a specific POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	User has navigated to the POI details page	
Success End Conditions	User sees the POI details	
Failed End Conditions	User doesn't see the POI details	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	Details link to POI details page	
Description	**Step**	**Action**
	1	User
Extensions	**Step**	**Branching Action**

Table 9: Use Case 9: View weather map

Use Case 9: View weather map		
Context of Use	User wants to see a overview of the weather in a dynamic map	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	User has navigated to weather map page	
Success End Conditions	User sees the dynamic weather map	
Failed End Conditions	User does not see the dynamic weather map	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User selects to see the weather map page	
Description	**Step**	**Action**
Extensions	**Step**	**Branching Action**

Table 10: Use Case 10: Search for POI

Use Case 10: Search for POI		
Context of Use	User wants to search for a specific POI	
Scope	System (Client Side Application)	
Level	Sub Function	
Preconditions	User has navigated to POI Finder Page	
Success End Conditions	User finds the POI	
Failed End Conditions	User does not find the POI	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User types in the search input field	
Description	**Step**	**Action**
Extensions	**Step**	**Branching Action**

Table 11: Use Case 11: View POI’s average rating and number of ratings

Use Case 11: View POI’s average rating and number of ratings		
Context of Use	User wants to see a POI’s average rating and the number of the performed ratings	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	User has navigated to proper view (POIs page, POI finder page, POI details page)	
Success End Conditions	User sees the average rating and the number of ratings performed for a specific POI	
Failed End Conditions	User doesn’t see the average rating and the ratings performed of a POI	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger		
Description	**Step**	**Action**
Extensions	**Step**	**Branching Action**

Table 12: Use Case 12: Register

Use Case 12: Register		
Context of Use	The user wants to register, so he can have full access to the system's functionality.	
Scope	System (Client Side Application)	
Level	Sub Function	
Preconditions	The User is not logged in the system	
Success End Conditions	The user registers successfully	
Failed End Conditions	The user could not register	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	User selects to register	
Description	**Step**	**Action**
	1	The System displays the register form
	2	The user fills the form
	3	The user submits the form
	4	The system validates the submitted form
	5	The system displays the first screen for logged in users
Extensions	**Step**	**Branching Action**
	1	4a1. Submitted data is incorrect
	2	4a2. The system informs the user that the filled form has invalid data

Table 13: Use Case 13: Login

Use Case 13: Login		
Context of Use	User wants to login	
Scope	System (Client Side Application)	
Level	Sub Function	
Preconditions	The user is not logged in	
Success End Conditions	The user logs in successfully	
Failed End Conditions	The user fails to log in	
Primary, **Secondary Actors**	User System (Client Side Application)	
Trigger	The user selects to login	
Description	**Step**	**Action**
	1	The System displays the log in form
	2	The user fills with his credentials the log in form
	3	The user submits the form
	4	The system validates the submitted form
	5	The system displays the first screen for logged in users
Extensions	**Step**	**Branching Action**
	1	4a1. Submitted data is incorrect
	2	4a2. The system informs the user that the filled form has invalid data

Table 14: Use Case 14: Rate POI

Use Case 14: Rate POI		
Context of Use	User wants to add his personal rating on a specific POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully rates a POI	
Failed End Conditions	The User fails to rate a POI	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	User selects an amount of stars	
Description	**Step**	**Action**
	1	The system has loaded the average rating and the number of ratings performed for a particular POI
	2	The User selects an amount of stars
Extensions	**Step**	**Branching Action**

Table 15: Use Case 15: Add POI to favourites

Use Case 15: Add POI to favourites		
Context of Use	User wants to add a specific POI to his personal favourites	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully adds a particular POI to his personal favourites	
Failed End Conditions	The User fails to add the POI to his personal favourites	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	User selects to add a specific POI to his personal favourites	
Description	**Step**	**Action**
	1	The system has loaded a page that gives the opportunity to the user to add a specific POI to his favourites
	2	The User selects to add the POI to his personal favourites
	3	The system loads a form with an input field for the user to add notes on that specific POI
	4	The User add his notes on the POI and submits the form
Extensions	**Step**	**Branching Action**
	1	3a1. The user aborts the procedure

Table 16: Use Case 16: Edit notes on favourite POI

Use Case 16: Edit notes on favourite POI		
Context of Use	User wants to edit his notes on a favourite POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	User is logged in and has at least one favourite POI	
Success End Conditions	User successfully edits the notes on a POI	
Failed End Conditions	User fails to edit the notes on a favourite POI	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects to edit the notes on a favourite POI	
Description	**Step**	**Action**
	1	The system has loaded the personal map page
	2	The User selects to edit his notes on a favourite POI
	3	The system loads a form with an input field filled with his notes which he can edit
Extensions	**Step**	**Branching Action**
	1	2a1. The User aborts the process

Table 17: Use Case 17: Remove POI from favourites

Use Case 17: Remove POI from favourites		
Context of Use	User wants to remove a POI from his personal favourites	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully removes the POI from his personal favourites	
Failed End Conditions	The User	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects to remove a POI from his favourites	
Description	**Step**	**Action**
	1	The System has loaded the personal map page
	2	The User selects a POI
	3	The System renders the POI overview with the notes
	4	The User chooses to remove this POI from his personal map
	5	The System asks the user for confirmation in order to remove the POI
	6	The User confirms and the system removes the POI from his personal map
Extensions	**Step**	**Branching Action**
	1	The user aborts the process

Table 18: Use Case 18: View all favourite POIs

Use Case 18: View all favourite POIs		
Context of Use	User wants to see all his personal favourite POIs on a dynamic map	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User sees a dynamic map with all his favourite POIs	
Failed End Conditions	The User doesn’t see any map or favourite POIs	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User navigates to his personal map through the navigation bar	
Description	**Step**	**Action**
Extensions	**Step**	**Branching Action**

Table 19: Use Case 19: View favourite POI’s short explanation and notes

Use Case 19: View favourite POI’s short explanation and notes		
Context of Use	User wants to see the short explanation and his notes of a specific POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User sees the short explanation and his notes of a specific POI	
Failed End Conditions	The User doesn’t see either the short explanation or his notes of the POI	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects a POI from his favourites	
Description	**Step**	**Action**
	1	The system loads the personal map page with all the User’s favourite POIs
	2	The User selects a POI
	3	The System renders the selected POI’s short explanation and notes
Extensions	**Step**	**Branching Action**

Table 20: Use Case 20: Change Password

Use Case 20: Change Password		
Context of Use	User wants to change his password	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully changes his password	
Failed End Conditions	The User fails to change his password	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects to change his password	
Description	**Step**	**Action**
	1	The System has loaded the user's profile page
	2	The User selects to change his password
	3	The System renders a form with his old password, his new password and a field to repeat the new password
	4	The User fills the form
	5	The User submits the form
	6	The System validates the form
	7	The System changes the user's password
Extensions	**Step**	**Branching Action**
	1	6a1. Submitted data is incorrect
	2	6a2. The system informs the user that the filled form has invalid data

Table 21: Use Case 21: Edit Interests

Use Case 21: Edit Interests		
Context of Use	User wants to edit his interests	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully edits his interests	
Failed End Conditions	The User fails to edit his interests	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects to edit his interests	
Description	**Step**	**Action**
	1	The system has loaded the profile page
	2	The User selects to edit his interests
	3	The System renders a form with all the supported interests
	4	The User chooses his interests
	5	The User submits the form
	6	The System validates the form
	7	The System changes the user’s password
Extensions	**Step**	**Branching Action**
	1	6a1. Submitted data is incorrect
	2	6a2. The system informs the user that the filled form has invalid data

Table 22: Use Case 22: Update profile picture

Use Case 22: Update profile picture		
Context of Use	User wants to update his profile picture	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully changes his profile picture	
Failed End Conditions	The User fails to change his profile picture	
Primary, **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User selects to update his profile picture	
Description	**Step**	**Action**
	1	The System has loaded the profile page
	2	The user selects to change his profile picture
	3	The system renders a file input field
	4	The User selects a picture
	5	The User submits
	6	The System changes the user’s profile picture
Extensions	**Step**	**Branching Action**
	1	The User aborts the process
	2	4a1. The selected picture is too big
	3	4a2. The System informs the user that the selected picture is too big

Table 23: Use Case 23: Add Comment on specific POI

Use Case 23: Add Comment on specific POI		
Context of Use	User wants to add a comment on a specific POI	
Scope	System (Client Side Application)	
Level	Summary	
Preconditions	The User is logged in	
Success End Conditions	The User successfully adds a comment on a specific POI	
Failed End Conditions	The User fails to add a comment on a specific POI	
Primary **Secondary Actors**	Logged in User System (Client Side Application)	
Trigger	The User submits the comment he typed on the comment input field	
Description	**Step**	**Action**
	1	The system has loaded the POI details page
	2	The User fills the corresponding input field
	3	The User submits the comment
	4	The system adds the user’s comment to the POI’s details page
Extensions	**Step**	**Branching Action**

6 Implementation

The functionality of the system and the architecture have been successfully implemented as designed. This Section discusses the implementation of the whole system. While chapter 6.1 describes both the POIs Model and the Application Model, via Entity Relationship diagrams and describes how their persistence was implemented via relational schemas. Chapter 6.3 describes the scraping application and last but not least section 6.4 describes the web infrastructure.

6.1 Model

This section specifies the basic entities of the whole system and describes how these were modeled and persisted.

Two entities have been identified in our system as basic entities, the POI and the logged in user. In order for the scraping logic to be decoupled from the web logic, two models are implemented. The POIs model containing all of the gathered POIs along with their details and the Application model that extends the POIs model and contains, along with the POIs the user specifications, the user-POIs relations, such as rating.

This Section describes how the data persistence was implemented for both our models.

POIs Database

As follows the implemented schema is presented via entity-relationship diagrams and specified by the provided relational schemes for both the POIs and the Application Databases. Figure 7.3 presents the ER diagram of the POIs Database.

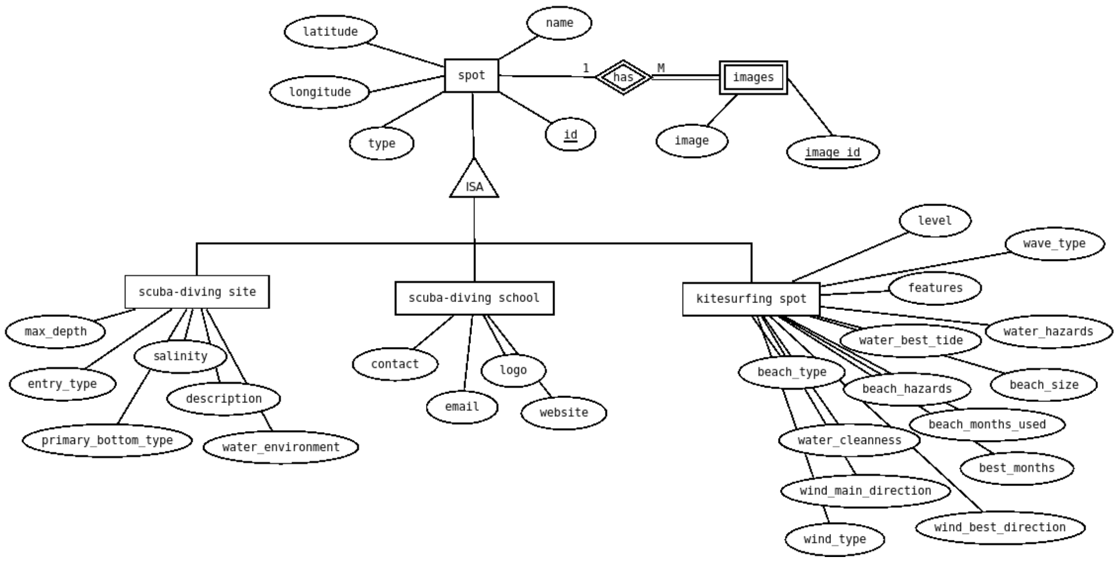


Figure 6.1: POIs Entity Relationship

The ER diagram presented in 7.3 is translated in the relational schema presented in table 7.1. In particular, each table of the database is given with its name, attributes, Primary Keys (PKs) and Foreign Keys (FKs). Primary keys are underlined, ex: <u>this_is_a_key</u> and foreign keys are shown with (FK) beside them, ex: this_is_a_foreign_key (FK). PKs are used to uniquely identify records of entities and FKs are used to define relationships between the entities.

Table 24: POIs Relational Schema

Table Name	**Attributes**
POI	id, name, latitude, longitude, type
POI_scubadiving	POI_id (FK), name, latitude, longitude, type, logged_dives, max_depth, entry_type, water_environment_type, salinity, primary_bottom_type and description
POI_scubadiving_school	POI_id (FK), name, latitude, longitude, type, address, contact details, email, website, image
POI_kitesurfing	POI_id (FK), name, latitude, longitude, type, water_type, level, features, wave_type, water_hazards, water_cleanness, water_best_tide, beach_type, beach_size, beach_hazards, beach_months_used, wind_type, wind_best_direction, wind_main_direction
Image	image_id, POI_id (FK), image

You may have noticed that the POI's abstract specification reside on all POI entities along with their specific attributes. Though it is not a common practice when handling inheritance, it is suitable in our situation. In greater detail, this is the way the Application's logic gets decoupled from the scraping logic. The Scraping application performs the scraping process and persist the gathered data in schema presented above. Attribute type in table POI refers to a specific POI category and thus the table that store the POI details. That way the Application Database schema can easily define relations on POIs via referring to one single table, the POI table. Last but not least, all of the POI data can be acquired in many ways in a single query.

Application Database

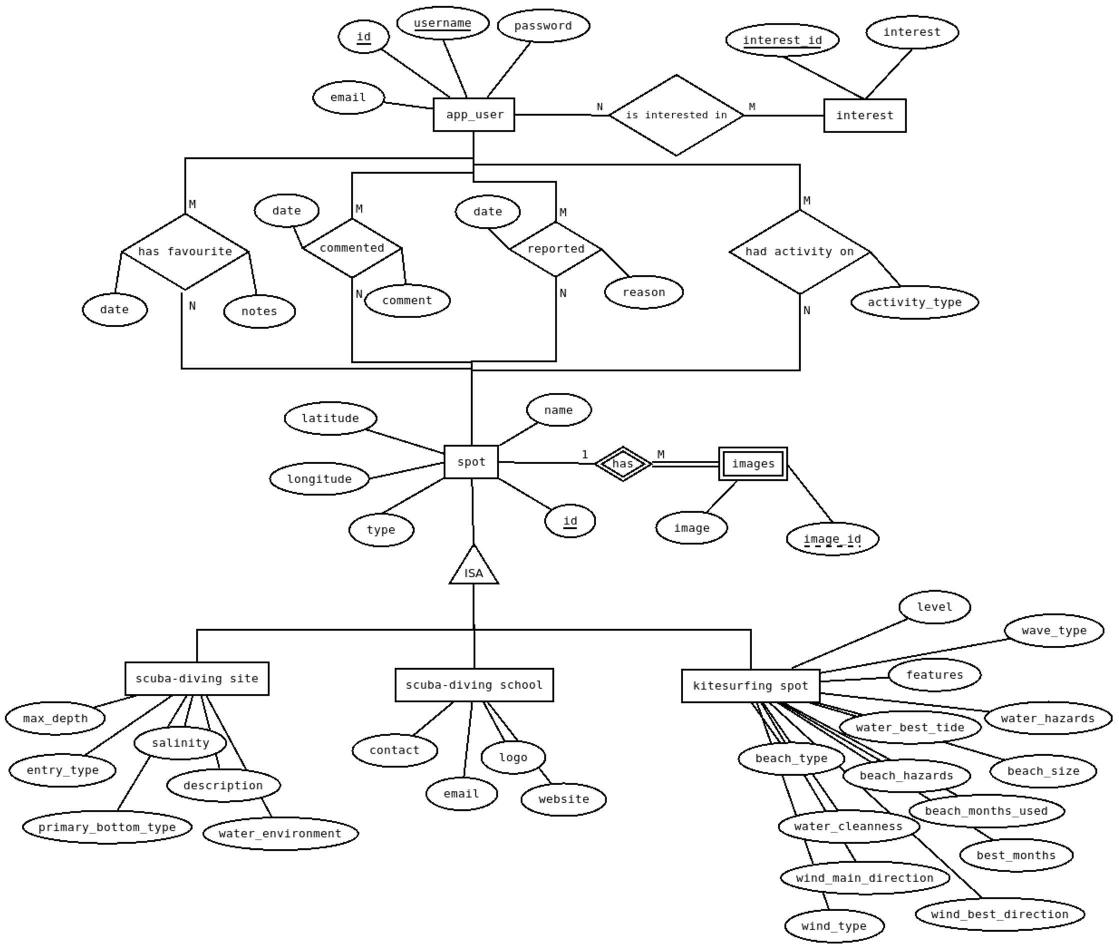


Figure 6.2: Application Entity Relationship

Like performed for the POIs Database Figure 7.4 presents the Application's Database ER diagram which gets translated to the Relational Schema presented in table 7.2. Both figures show how the Application database extends the POIs database.

Table 25: The Database Relational Schema

Table Name	**Attributes**
POI	id, name, latitude, longitude, type
POI_scubadiving	POI_id (FK), logged_dives, max_depth, entry_type, water_environment_type, salinity, primary_bottom_type, and description
POI_scubadiving_school	POI_id (FK), address, contact details, email, website, image
POI_kitesurfing	POI_id (FK), water_type, level, features, wave_type, water_hazards, water_cleanness, water_best_tide, beach_type, beach_size, beach_hazards, beach_months_used, wind_type, wind_best_direction, wind_main_direction
Image	image_id, POI_id (FK), image
app_user	username, password, email
interest	Id, interest
app_user_interest	user_id (FK), interest_id (FK), level
app_user_POI_comment	comment_id, app_user_id (FK), POI_id (FK), comment, date
app_user_POI_report	report_id, app_user_id (FK), POI_id (FK), reason, date
app_user_POI_rate	rating_id, user_id (FK), POI_id (FK), rating, date
app_user_POI_favourite	favourite_id, user_id (FK), POI_id (FK), notes, date

6.2 *Scraping Application*

The Scraping application implements a multi-layered architecture in Java with the Spring Boot framework (see 4.2) and it gets executed from the terminal via an executable JAR file. Section 7.1 describes the Application architecture and section 7.2 the process.

6.2.1 Architecture

Figure 7.1 presents the overall Scraping Application Architecture.

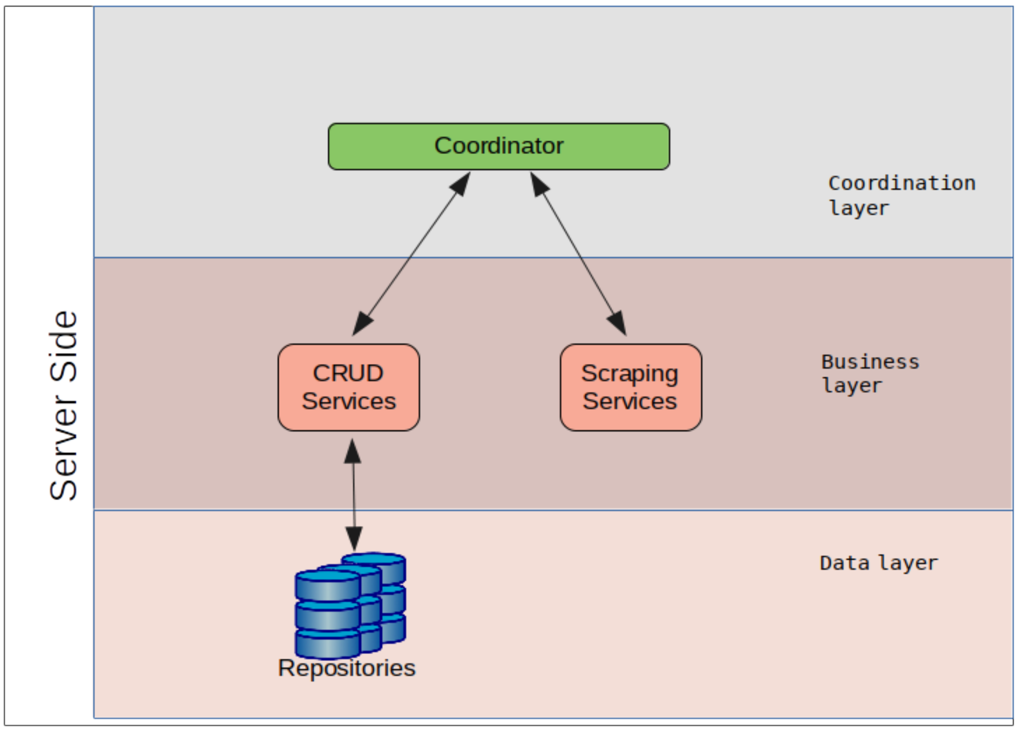


Figure 6.3: Scraping Application Architecture

Coordinator

The coordinator based on runtime attributes calls the appropriate services to perform the according operations. Depending on runtime arguments it supports 5 operations.

1) Perform kitesurfing POIs scraping
2) Perform scuba-diving dive sites scraping
3) Perform scuba-diving schools scraping
4) Perform scraping for all POI categories
5) For each POI perform coordinates to address conversion through scraping the google’s Geocoding API and update all POIs

Operation 5 was developed after the Web Application evaluation where it became clear the need to provide address for each POI for better searching and user experience.

Scraping Services

Scraping services execute all the scraping processes. They support methods for acquiring information for kitesurfing POIs, scuba-diving schools and scuba-diving dive sites. In particular, two web pages were targeted to gather the desired data, http://www.kiteforum.com/ to acquire information for kitesurfing POIs and https://www.scubaearth.com/ to gather information for scuba-diving dive sites and schools.

Another implemented Scraping procedure is the conversion of all addresses for the POIs that have been harvested through Google's Geocoding API.

CRUD Services

The specified processes require create and update CRUD services which where implemented for all the entities of the application defined in the POIs model.

POIs Model

Hibernate was used to implement the POIs Model (see 6.1). Hibernate is an Object Relational Mapping (ORM) framework capable of mapping any type of relations between entities and compatible with PostgreSQL.

Repositories

JPA repositories were used to access the POIs Database. Repository Components are used to index and persist the data of the application. JPA repositories provide a lot generic infrastructure and reduces a lot of boilerplate code.

6.2.2 Scraping Process

An executable JAR file is provided to run the Scraping Application. Based on runtime variables the coordinator "fires up" methods to either scrape kitesurfing POIs, scuba-diving dive sites or scuba-diving schools. All of the above were implemented with Jsoup (see section 3.1.3). In more detail, Jsoup was used to obtain the web pages, build their DOM like modern browsers do and perform DOM queries and CSS selectors to targeted information. The gathered information are modeled and persisted using Hibernate to model the Database Schema and JPA Repositories to access it. Last but not least there is a runtime option for converting geographical coordinates to addresses for all the gathered POIs. It was implemented using Spring's RestTemplate to make synchronous HTTP request to Google's servers and consume the JSON response of the Geocoding API.

6.3 Web Infrastructure

This chapter describes the implemented logic for the web related part of the system. Section 7.4.1 describes its architecture and advocates on some of the architectural decisions that were made for the most important application aspects. Section 7.4.2 describes the RESTful Web Services which reside on server side whereas section 7.4.3 describes the client-side Application.

6.3.1 Architecture

The system adopts the Rich Internet Application and the state of the art principles which promotes Web applications as desktop-like applications where business logic is performed on both the server and the client side. The client side logic operates within the web browser running on a user's local computer or "smart" device, while the server side logic operates on the web server hosting the application. To comply with the above, web services were used on server side which provides an API to the client side. The described architecture separates the server side from the client side logic simplifying the development process and increases the system's scalability, maintainability and robustness. Figure 7.6 presents the Architecture of the implemented Web infrastructure.

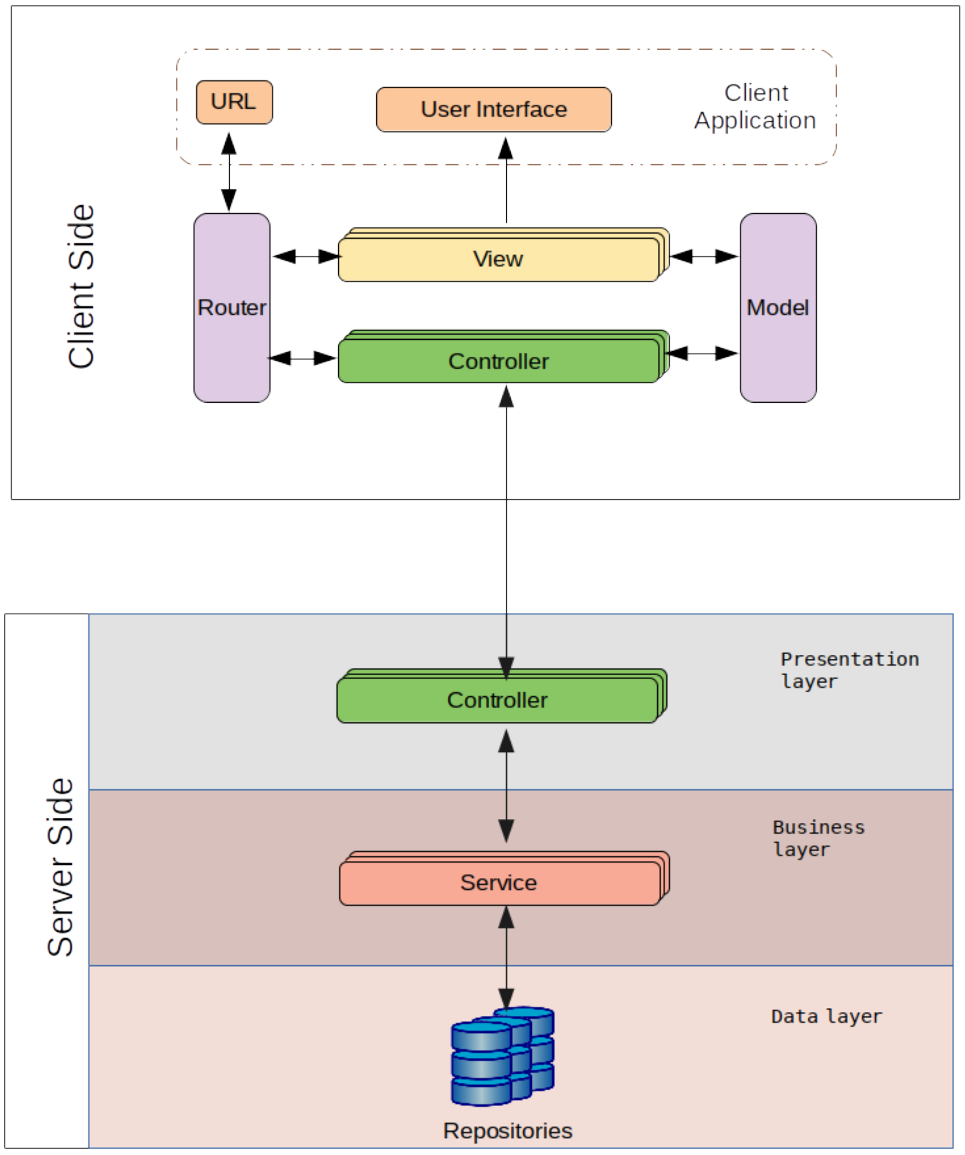


Figure 6.4: Overall System Architecture

6.3.2 RESTful Web Services

The Server Side of our System is based on Java programming language. Moreover, it is worth to mention that we have made extensive use of the Spring Boot Framework in order to tackle certain aspects of our backend.

The Server Side part of the developed framework provides RESTful Web Services to the client-side while follows a multi-layered architectural pattern. The multi-layered architectural pattern separates the client-server communication logic, the business logic and the data access logic in three layers the Service layer, the Business Logic layer and the Data layer. Thus, distinct components have been implemented operating in the three layers. As follows all the layers are being described.

The reason of using REST was based on the following facts: (a) less network bandwidth requirements compared to SOAP, (b) business logic reuse, (c) more standardized operations that are well understood and operate consistently based on HTTP GET, POST, PUT and DELETE methods, (d) development of human readable and testable components, and (e) there is no requirement to use complex data interchange formats like XML. To this end, we built a package of RESTful services exposing all the functionality of our system and we created CRUD (create, retrieve, update, delete) operations for every possible feature.

Service Layer

Service layer uses the Controller components to define API endpoints. In more detail, Controller components are the ones responsible for the configuration of the RESTful API and its endpoints provided to the client-side.

Business Logic Layer

The Business Logic Layer, also known as Domain Layer, contains the business logic of the application and separates it from the Data Layer and the Service Layer. Business logic refers to both business objects as well as operations on the objects. In more detail:

- The POI Management Module, is responsible for the POI entity management and definition
- The User Management Module, is responsible for the Entity User management and definition
- The UserPoiRelations Management Module, is responsible for the definition and management of the user’s interaction on the POIs

Data Layer

The Data Layer accommodates the external Database of the system via Repository Components. Repository Components are used to index and persist the data of the application.

6.3.3 Client Side Application

The Implemented Client Side Application is just an example usage of the API provided by the implemented server side web services. As the provided API is in machine readable format any other can be based on that API.

The Client Side of our System is based on JavaScript programming language. Moreover through the extensive usage of the AngularJS framework the MVC architecture described in chapter 7 was implemented. In particular AngularJS View, Directive, Controller, Service, Configuration components were developed to fulfill the needs of the application.

The main reason for using AngularJS is that it provides a great way for defining a clean MVC architecture with code and component reuse.

The Client Side of the application is responsible for the interaction with the user. All the actions performed by an individual using the system, are handled by the client side logic, which undertakes the presentation of the information as well as the communication with the server. In order to achieve a high level of decoupling between the components forming the client logic we adopted the Model View Controller (MVC) design pattern[24].

The usage of the MVC pattern introduces the separation of the responsibilities for the visual display and the event handling behavior into different entities, named respectively, View and Controller. Some of the advantages on this approach are: (a) maximization of the code that can be tested with automation (Web pages containing HTML elements are hard to test), (b) code sharing between pages that require the same behavior, and (c) separation of the business logic from the user interface to make the code easier to understand and maintain.

In what follows, we describe the components of the client-side architecture (see Figure 7.1) in more detail, focusing on their objective and internal functionality.

Model

The Model refers to the business objects of the system. Client side business objects refer server side business objects. Thus, when the system needs to present information about the business object, the client side requests the respective information. Similarly, Update, Create and Delete operations involve according requests to the server side.

View

The Views are responsible for the presentation of information in the user interface. Each view controls a number of widgets on the application's graphical user interface. It consists of several handlers that are responsible for listening user actions, as well as HTML templates that define the presentation of the widgets.

Controller

The Controller components are the modules that handles user actions and interaction with the Views in order to perform changes on the interface. Furthermore, they maintain the Model and change it appropriately. Every View has a dedicated Controller managing, handling and propagating any changes to be performed on the interface. In several cases Controller components manage other controllers in order to crate complex widgets.

Router

The Router is used for deep-linking URLs to Controllers and Views. It maps the URL of the client browsers to Views and Controllers providing a unique path to each distinct interface without the need to reload the whole page. When URL changes, the Router analyzes the new path and handles transition to the appropriate View.

6.4 Recommendation System

This chapter describes the implemented recommendation system. Up to this point it is in a very early stage and not integrated in the whole system. It is presented and described because it is an important component that has to be integrated to complete the whole system. In more detail, it is a user-based collaborative filtering recommendation system implemented with jaccard similarity [26].

Jaccard similarity is used for comparing the similarity of two sets and for sets A and B is defined as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

In particular, it was used to find similar users to the logged in based on his interests, favorites and reviews. The process of finding similar users to the logged in user is based 40% on favourites, 40% on interests and 20% on reviews. The problem is that it is desired to predict the user's rating on various POIs based on the ratings of his nearest neighbors, which are the most similar ones.

7 GUI

This section presents the methodology followed for designing the user interfaces of the web application, as well as the final product. Chapter 7.1 describes the designing phase and chapter 7.2 presents the outcome of the graphical user interfaces as it has been implemented. We used wireframes to design and specify the functionality of each page. All of the pages of the application and their internal links are presented in the sitemap are presented in figure 7.1.

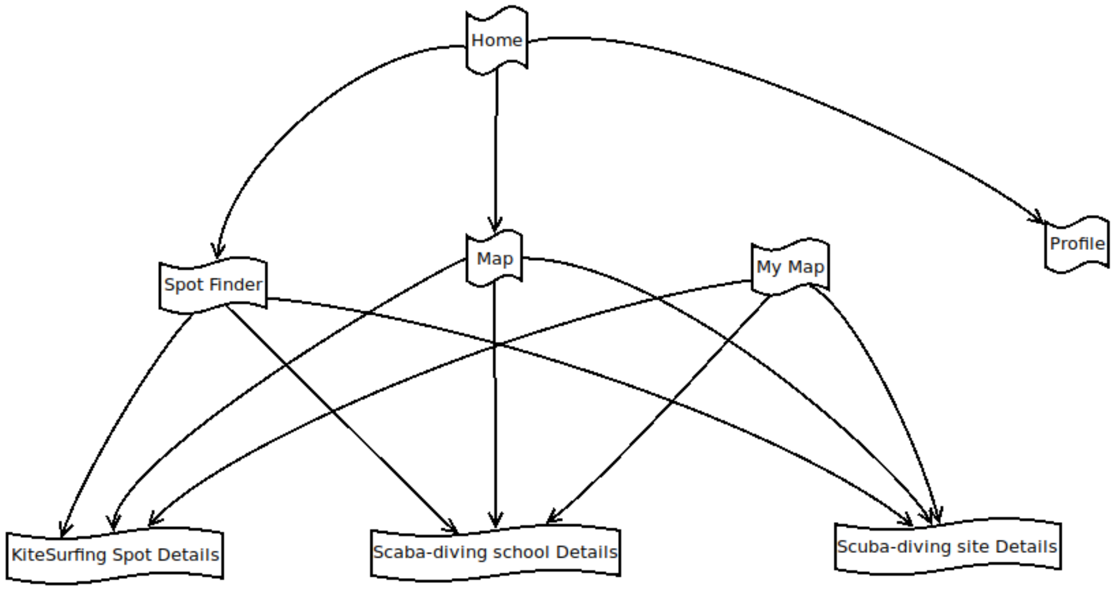


Figure 7.1: Web Application Sitemap

7.1 Wireframes

Using Wireframes is a common approach to design an application's UI. It is worth mentioning that the development process started after the whole functional specification and design process. Furthermore, inspiration was retrieved for designing the web application's wireframes from [24].

Both the System and the UI support two end-user roles. Figure 7.2 presents the templates designed for the Guest user and figure 7.3 presents the template designed for the logged in user. Both consists of a navigation bar with internal links other pages of the application and role-specific actions. Guest user is provided with links to the login page and register page, whereas a logged in user is provided with a dropdown button.

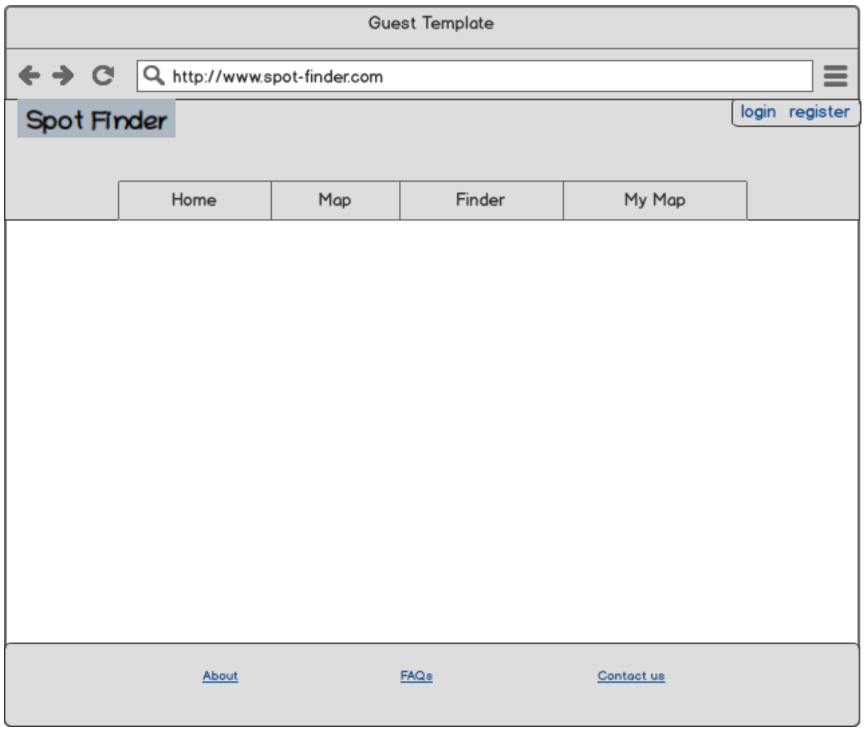


Figure 7.2: Guest Template

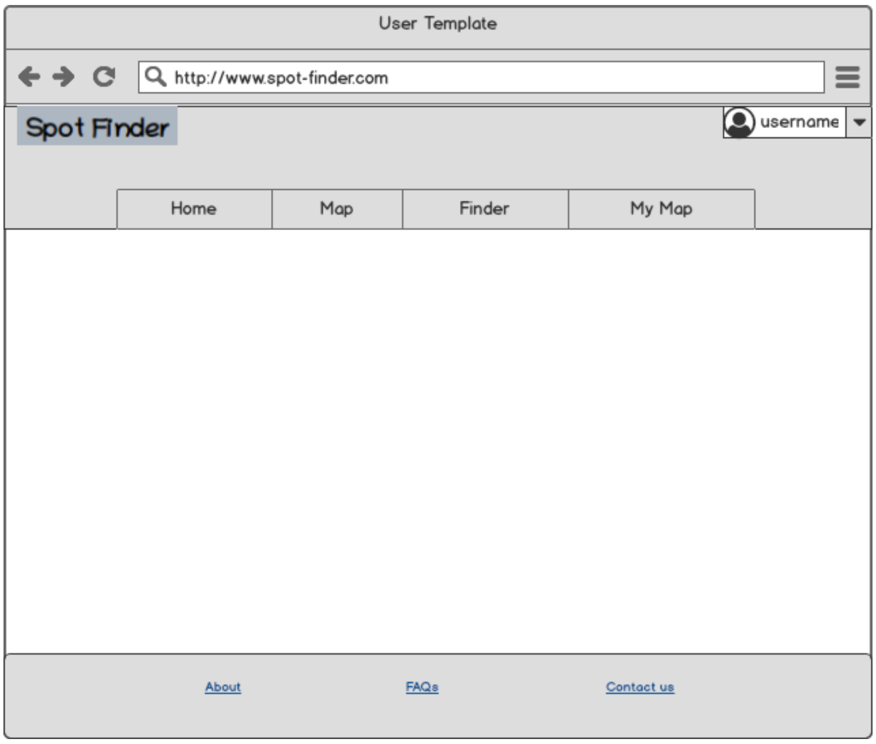


Figure 7.3: Logged in user Template

For each wireframe the consists a possible action, a dedicated wireframe to the referred action is presented. Figure 7.4 presents the logged in user's action clicking on <username> dropdown. The user is provided with an internal link to the profile page and the option to log out.

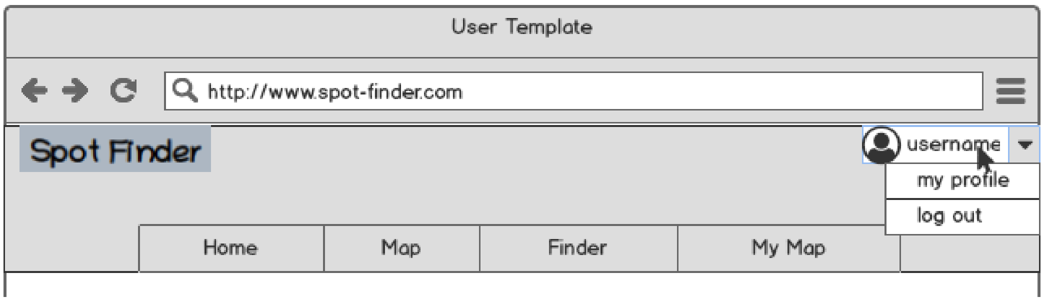


Figure 7.4: Logged in user action

Along the following lines are presented all the wireframes as they were initially designed. We have separated them is the following categories the guest only wireframes, common wireframes, which present common user-guest wireframes and logged in user only wireframes.

Guest only wireframes

Figure 7.5 presents both the Login and Register pages wireframes.

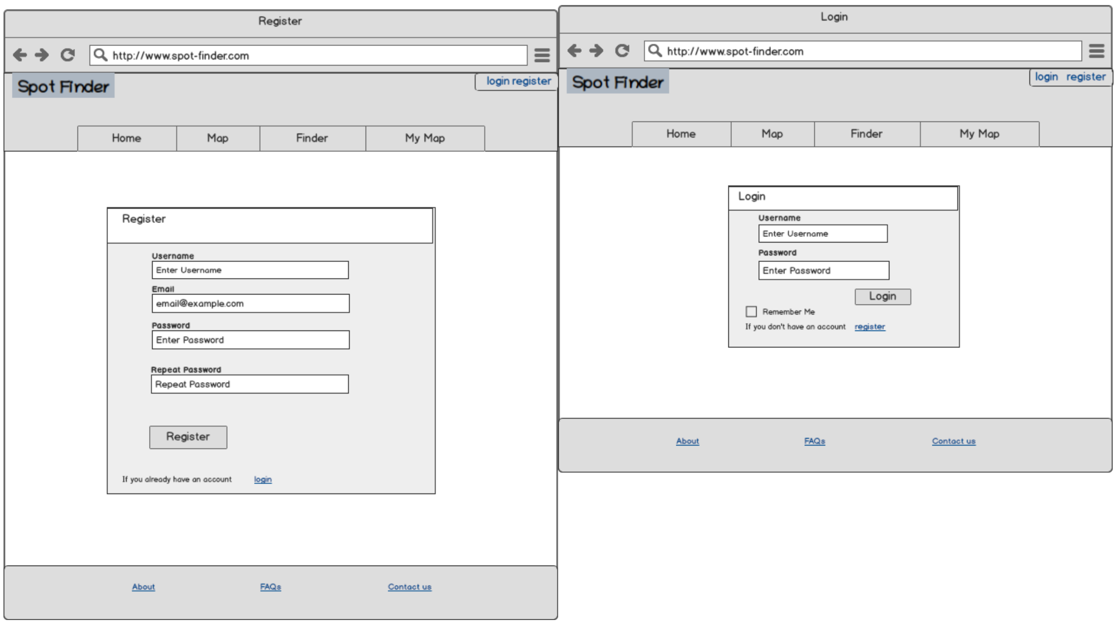


Figure 7.5: Login and Register pages wireframes

Guest and logged in user common wireframes

All of the following figures present the common wireframes for both of our user roles. For each on the right side is presented the Guest wireframe and the left side presents the user wireframe. Figure 7.6 presents the Map page wireframe. The Map in figure 7.6 is represents a dynamic interactive map like the well-known google-maps.

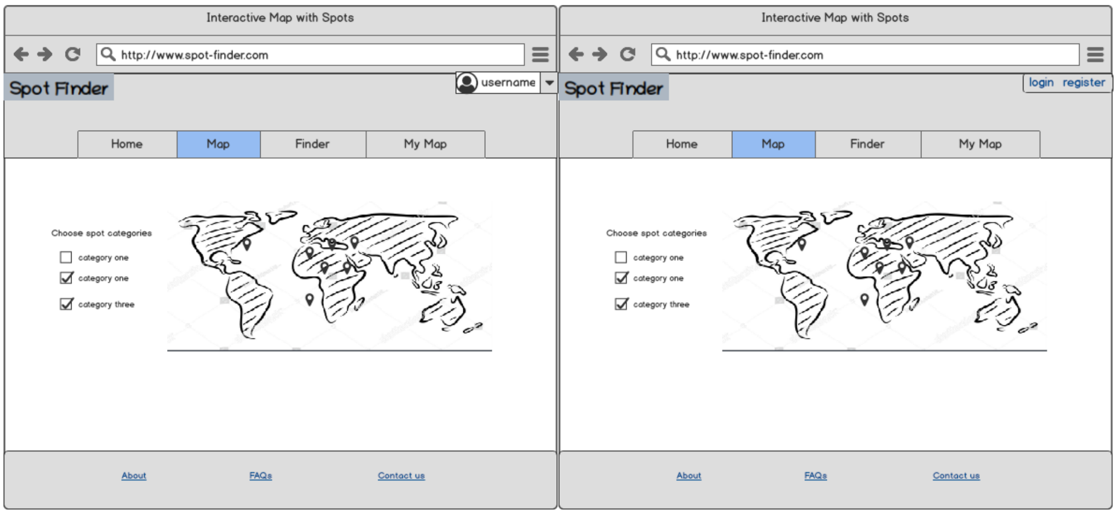


Figure 7.6: Map page wireframe

All of the POIs presented in the mentioned interactive map are clickable. Figure 7.7 presents the on a POI click action performed by the end-user.

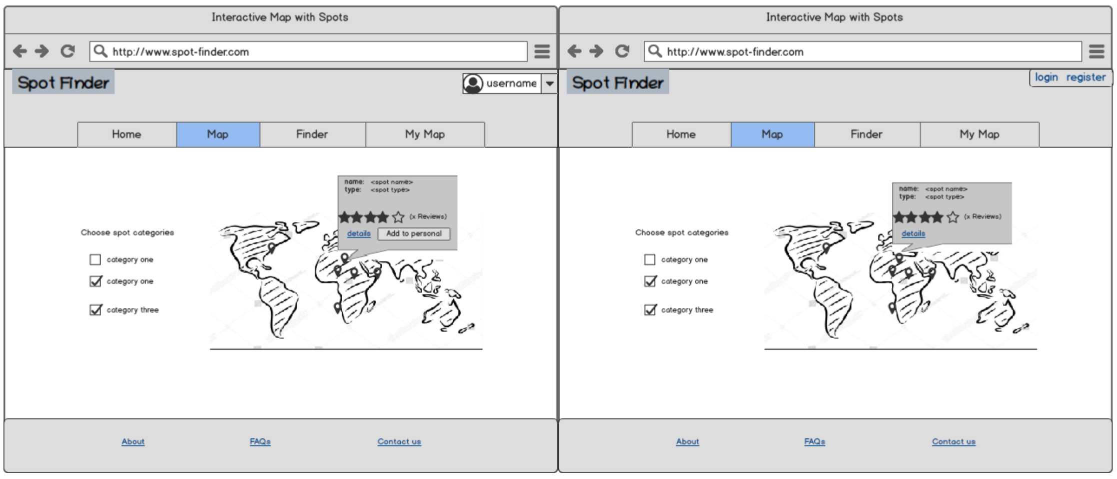
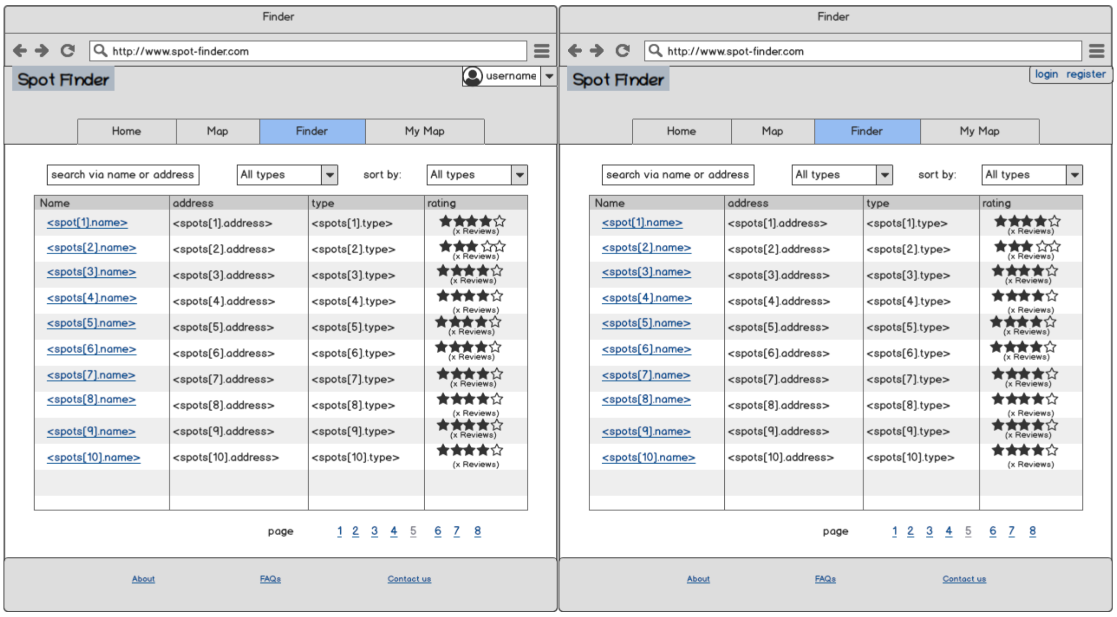


Figure 7.7: Map clicking on POI action wireframe

Figure 7.8 presents the finder page.



Name	address	type	rating
<spot[1].name>	<spots[1].address>	<spots[1].type>	★★★★☆ (x Reviews)
<spots[2].name>	<spots[2].address>	<spots[2].type>	★★★☆☆ (x Reviews)
<spots[3].name>	<spots[3].address>	<spots[3].type>	★★★★☆ (x Reviews)
<spots[4].name>	<spots[4].address>	<spots[4].type>	★★★★☆ (x Reviews)
<spots[5].name>	<spots[5].address>	<spots[5].type>	★★★★☆ (x Reviews)
<spots[6].name>	<spots[6].address>	<spots[6].type>	★★★★☆ (x Reviews)
<spots[7].name>	<spots[7].address>	<spots[7].type>	★★★★☆ (x Reviews)
<spots[8].name>	<spots[8].address>	<spots[8].type>	★★★★☆ (x Reviews)
<spots[9].name>	<spots[9].address>	<spots[9].type>	★★★★☆ (x Reviews)
<spots[10].name>	<spots[10].address>	<spots[10].type>	★★★★☆ (x Reviews)

Figure 7.8: Finder page wireframe

Within Finder page reside a search input and two dropdown buttons, all of which trigger user actions caught and handled by the application. The search input is supposed to perform live searching responding on user input. The first dropdown is supposed to perform filtering based on POI type selection and the second dropdown is supposed to perform sorting based on the dropdown selection. Figure 7.9 presents the options for the first dropdown button and figure 7.10 the options for the second.

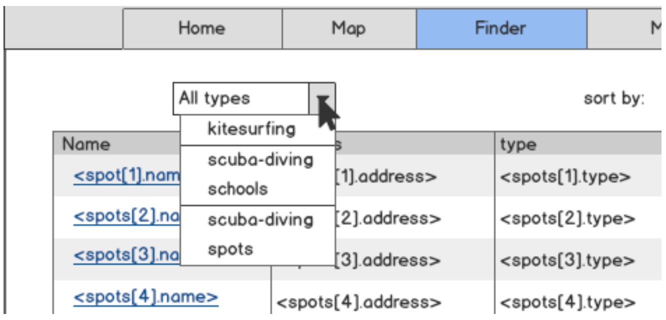


Figure 7.9: Filtering Dropdown button oprions

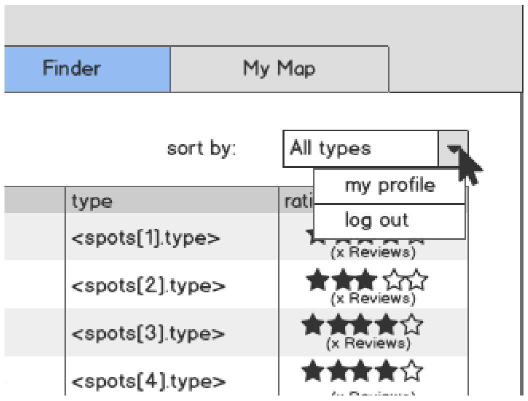


Figure 7.10: Sorting dropdown button options

Figures 7.11, 7.12 and 7.13 present the POI details wireframes page for each of the POI categories suspported. As performed in other common user and guest wireframes the right side represents the guest version of details page and the left the logged in user version of details page.

Figure 7.11 presents the kitesurfing POI details page wireframe.

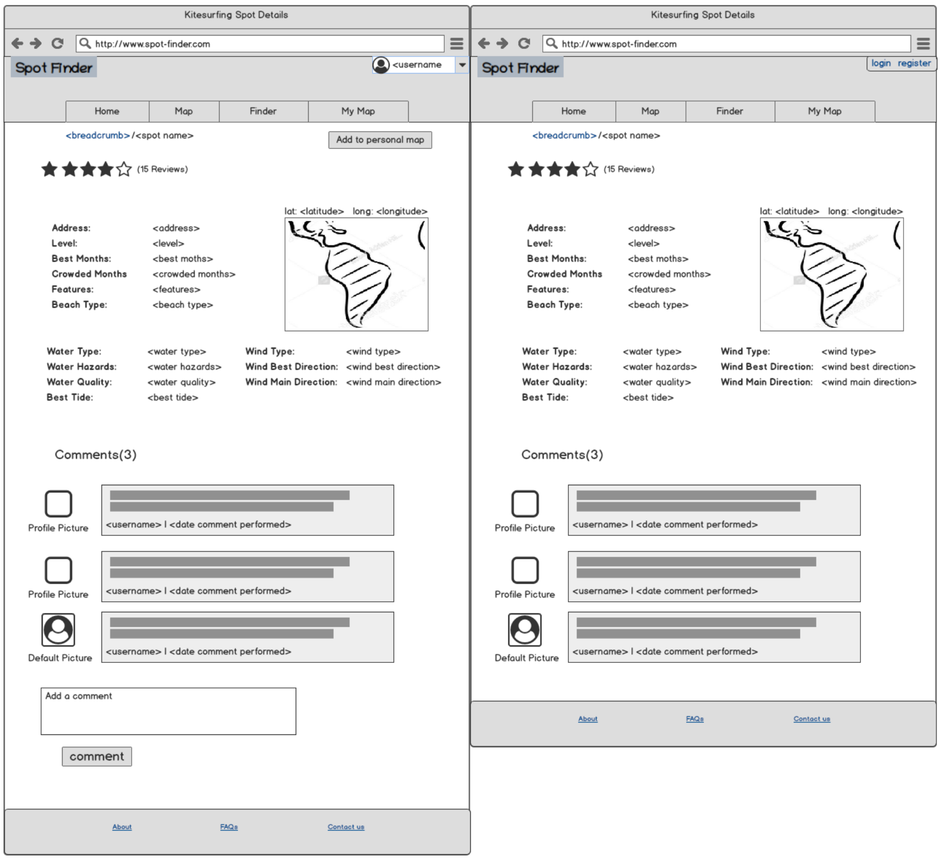


Figure 7.11: Kitesurfing POI details wireframe

Figure 7.12 presents the scuba-diving school details page wireframe.

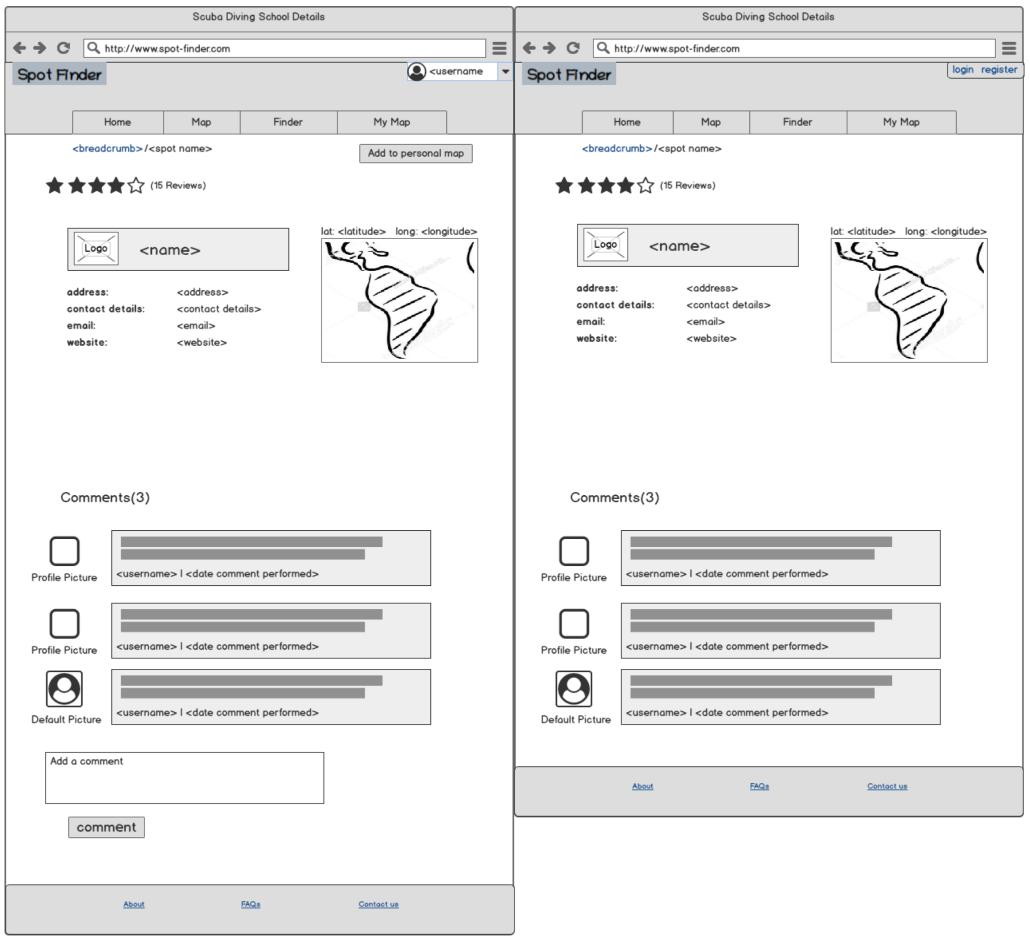


Figure 7.12: Scuba-diving school details page wireframe

Last but not least, figure 7.13 presents the scuba-diving site details page wireframe.

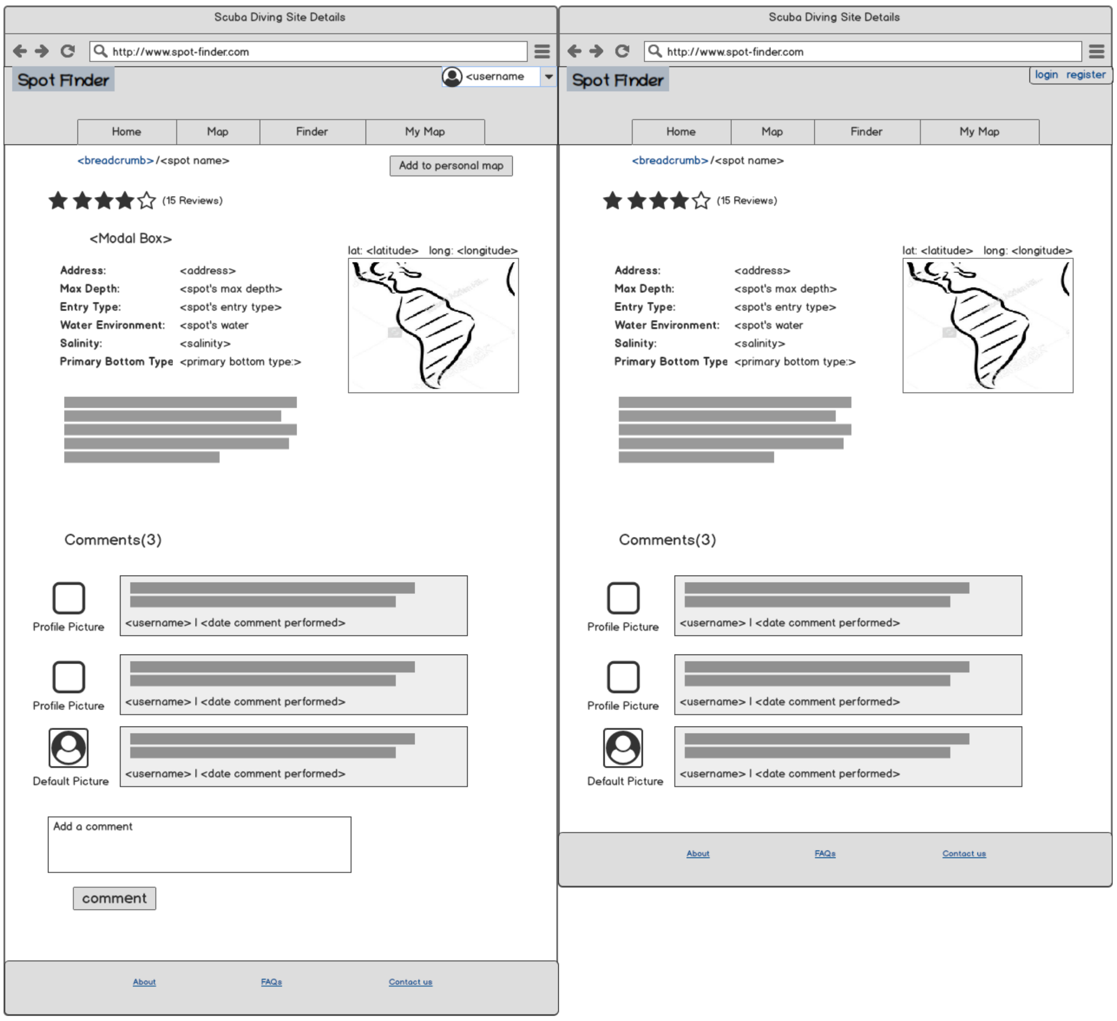

Scuba Diving Site Details
http://www.spot-finder.com
Spot Finder
<username
login register
Home
Map
Finder
My Map
<breadcrumb>/<spot name>
Add to personal map
(15 Reviews)
<Modal Box>
lat: <latitude> long: <longitude>
Address: <address>
Max Depth: <spot's max depth>
Entry Type: <spot's entry type>
Water Environment: <spot's water
Salinity: <salinity>
Primary Bottom Type <primary bottom type:>
Comments(3)
Profile Picture
Default Picture
<username> I <date comment performed>
Add a comment
comment
About
FAQs
Contact us

Figure 7.13: Scuba-diving site details page wireframe

For the logged in user version of all the POI details pages presented above the add to personal map button is supposed to trigger an action and handled by our application. Figure 7.14 presents how the on click action is handled. In particular on add to personal map button click the applications renders a modal box in order for the user to fill notes on the specific POI and add it to his personal map.

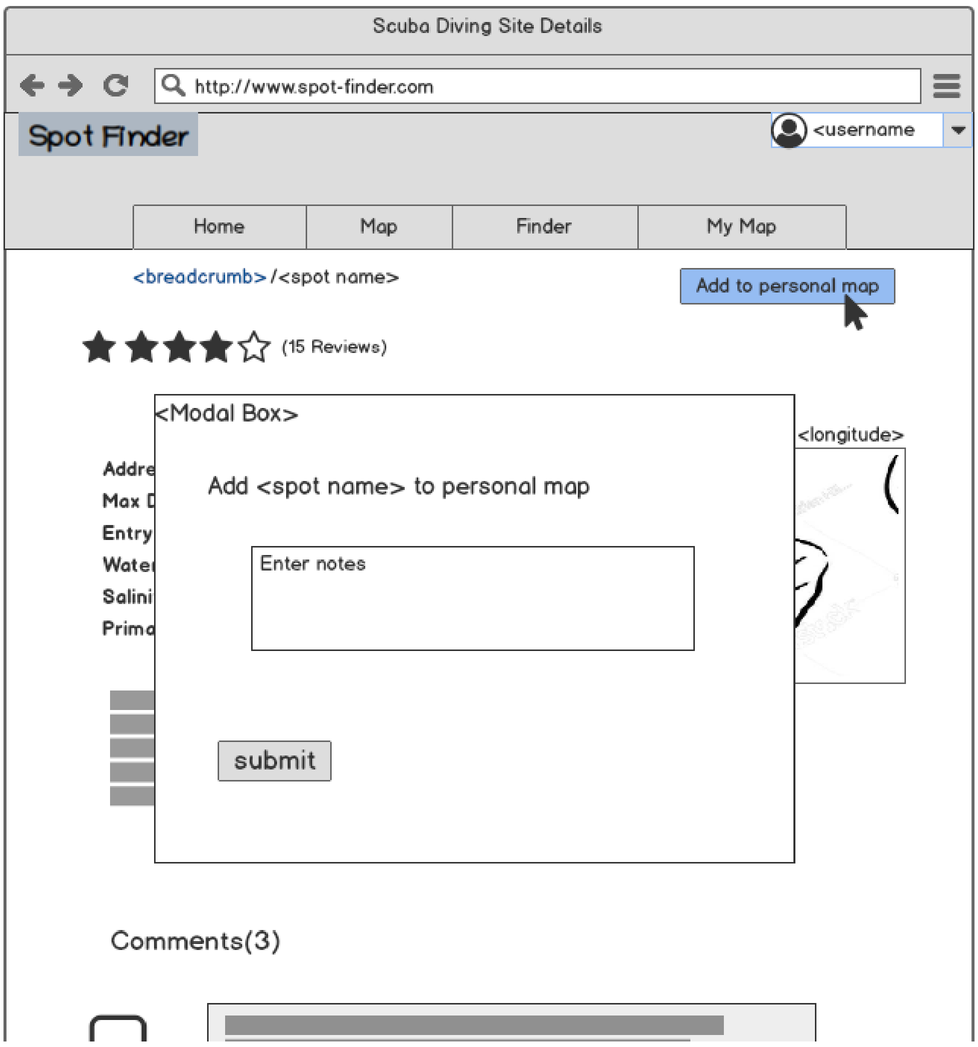


Figure 7.14: Add to personal map action

If a POI is already on personal map the button should be replaced with a heart.

Logged in user only wireframes

In the following lines all of the Logged user only wireframes are presented. By design if a guest user tries to render a user only page he should be redirected to the login page.

Figure 7.15 presents the personal map page wireframe in which all of the user favourite POIs are presented along with their notes. Figure 7.16 presents how the personal map page should handle the on POI click.

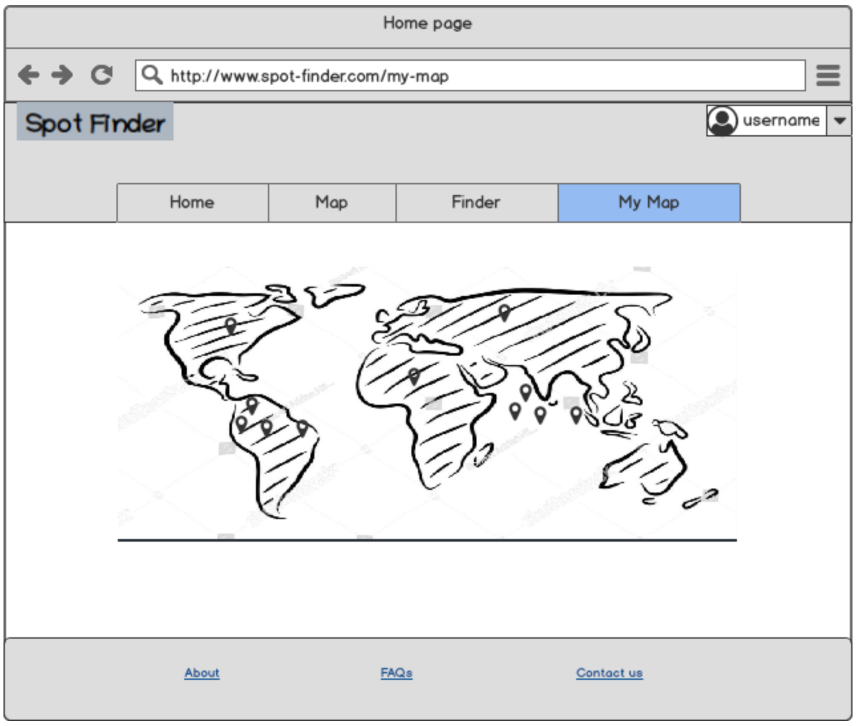


Figure 7.15: Personal map page wireframe

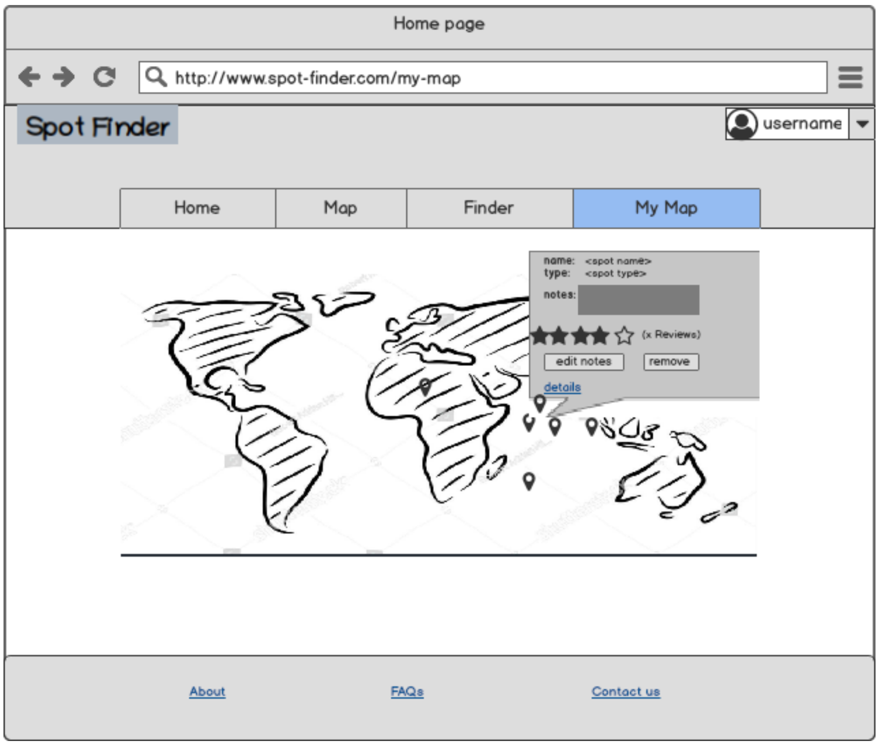


Figure 7.16: Clicking on POI action personal map page wireframe

Figure 7.17 presents both how edit notes and remove actions are handled, which are triggered by the edit notes and remove buttons. They both, as presenting, render a dedicated modal box.

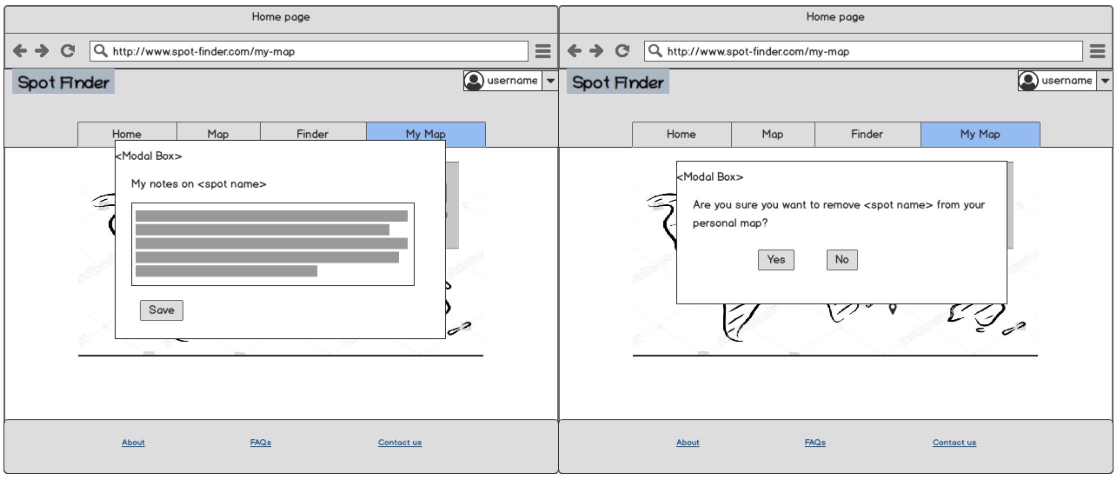


Figure 7.17: Edit notes and remove POI actions wireframes

Figure 7.18 presents the profile page wireframe.

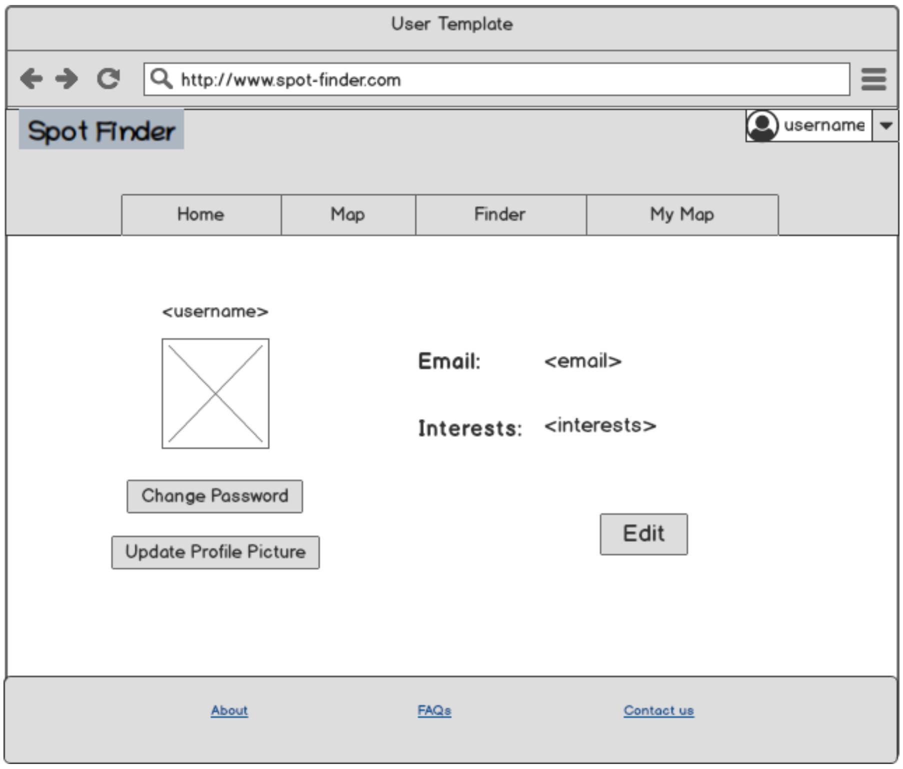


Figure 7.18: Profile page wireframe

In profile page reside three actions. The first is change password action, the second is update profile picture action and the third is Edit personal info action. All of the mentioned actions render a dedicated modal box.

Figure 7.19 presents how the application should handle the on edit button click.

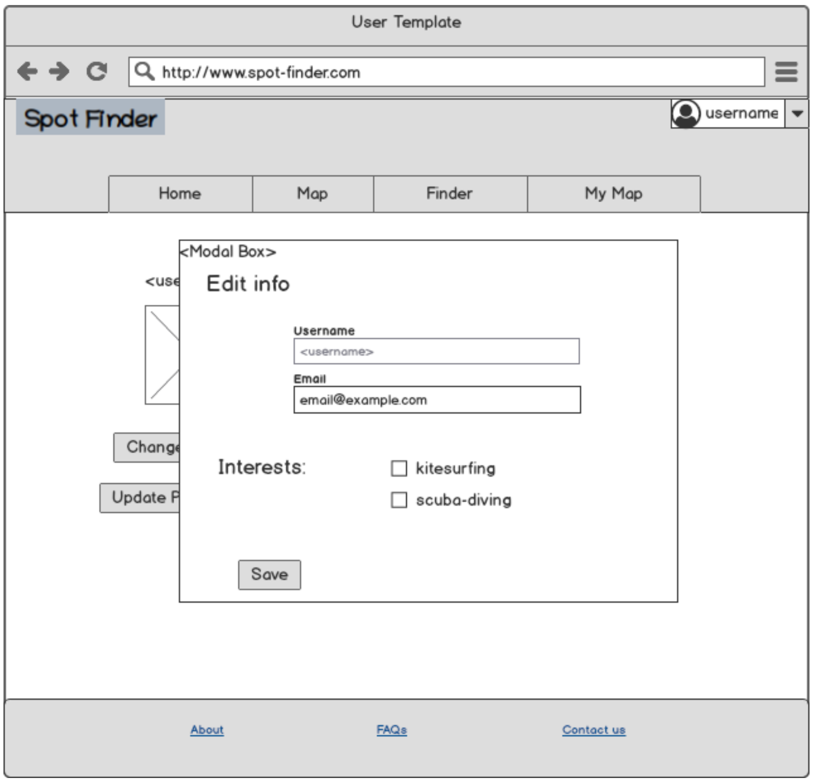


Figure 7.19: Edit personal info wireframe

7.2 *User Interface*

This section presents the outcome of the graphical user interfaces as it has been implemented. In greater detail, all the pages of the application are presented and described.

Figure 7.20 presents the Main Map page, in which all of the POIs are presented via the interactive google map. In this page the user is able to perform three major tasks. Firstly, he is able to apply filters on the POI categories he is interested in from the right side of the page. Secondly, he is able to navigate on the map and click on POIs to see an overview of information about the specific POI or navigate to details page to see all the information for that specific POI. If the user is logged in he is able to add it in his personal map. Last but not least, he is able to search a specific location and explore it via the input which resides above the map.

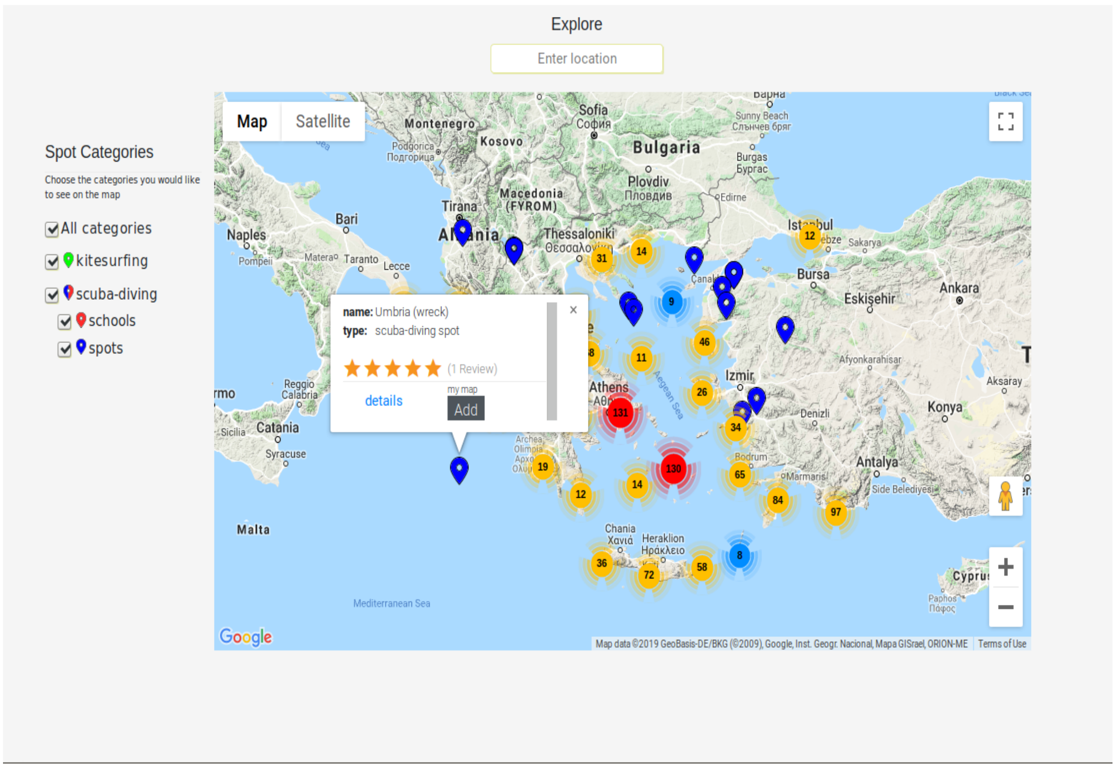
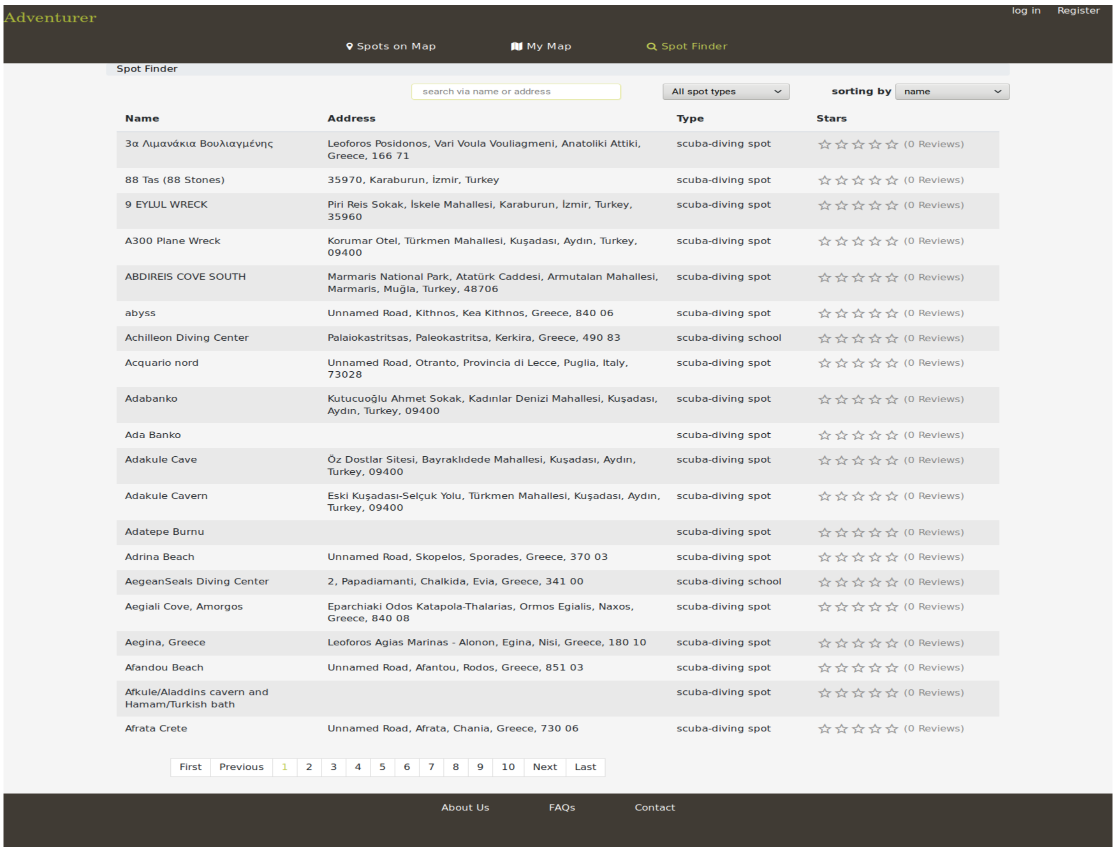


Figure 7.20: Main map page

The following figures present the details pages for all of our categories. Each one describes how a POI of a category is presented. In all of them, reviews comments and photos are presented the same way.

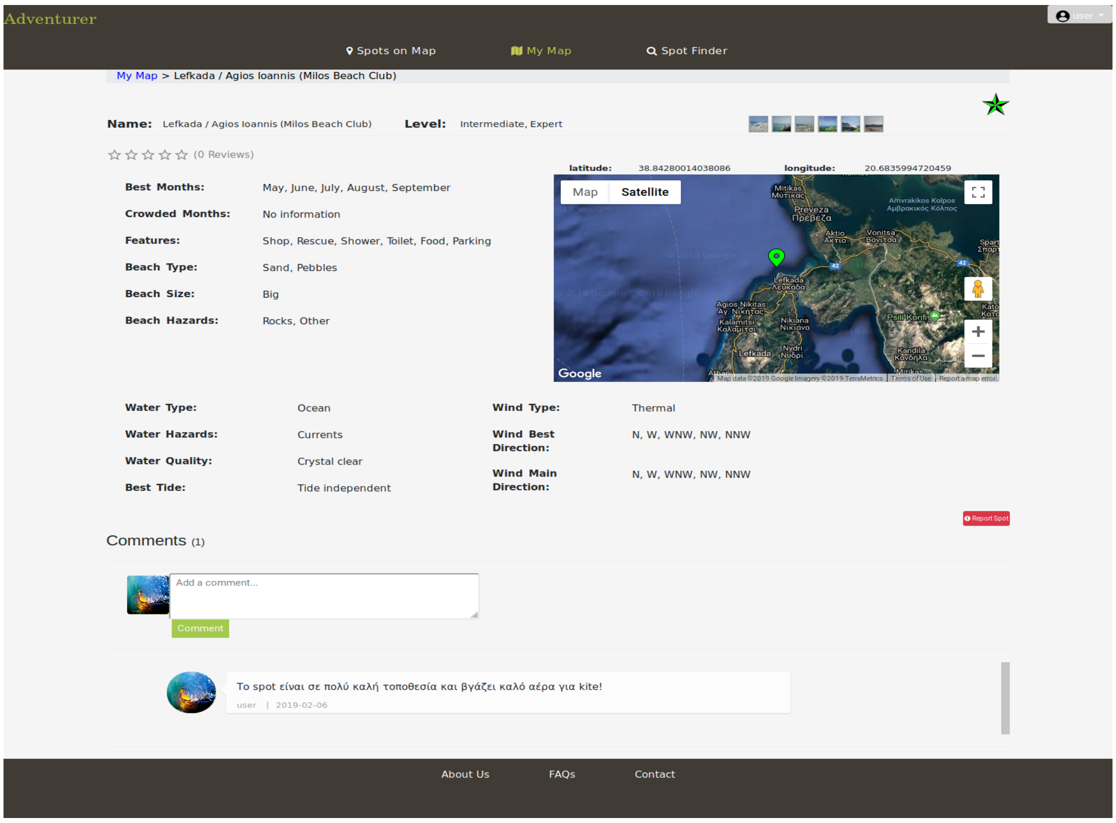
Figure 7.21 presents the spot finder page. In this page backend pagination is performed. Backend pagination means that the application renders only one page of data and if the user requests another page an ajax call is performed to fulfill the task. In this page user is able to filter the POIs by categories, search for POIs via name or address and sort the by name, category, rating score or by numbers of ratings. Furthermore, each row is a link to the details page of the POI.



Name	Address	Type	Stars
3α Λιμανάκια Βουλιαγμένης	Leoforos Posidonos, Vari Voula Vouliagmeni, Anatoliki Attiki, Greece, 166 71	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
88 Tas (88 Stones)	35970, Karaburun, İzmir, Turkey	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
9 EYLUL WRECK	Piri Reis Sokak, İskele Mahallesi, Karaburun, İzmir, Turkey, 35960	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
A300 Plane Wreck	Korumar Otel, Türkmen Mahallesi, Kuşadası, Aydın, Turkey, 09400	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
ABDIREIS COVE SOUTH	Marmaris National Park, Atatürk Caddesi, Armutalan Mahallesi, Marmaris, Muğla, Turkey, 48706	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
abyss	Unnamed Road, Kithnos, Kea Kithnos, Greece, 840 06	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Achilleon Diving Center	Palaiokastritsas, Paleokastritsa, Kerkira, Greece, 490 83	scuba-diving school	☆☆☆☆☆ (0 Reviews)
Acquario nord	Unnamed Road, Otranto, Provincia di Lecce, Puglia, Italy, 73028	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Adabanko	Kutucuoğlu Ahmet Sokak, Kadınlar Denizi Mahallesi, Kuşadası, Aydın, Turkey, 09400	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Ada Banko		scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Adakule Cave	Öz Dostlar Sitesi, Bayraklıdede Mahallesi, Kuşadası, Aydın, Turkey, 09400	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Adakule Cavern	Eski Kuşadası-Selçuk Yolu, Türkmen Mahallesi, Kuşadası, Aydın, Turkey, 09400	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Adatepe Burnu		scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Adrina Beach	Unnamed Road, Skopelos, Sporades, Greece, 370 03	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
AegeanSeals Diving Center	2, Papadiamanti, Chalkida, Evia, Greece, 341 00	scuba-diving school	☆☆☆☆☆ (0 Reviews)
Aegiali Cove, Amorgos	Eparchiaki Odos Katapola-Thalarias, Ormos Egialis, Naxos, Greece, 840 08	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Aegina, Greece	Leoforos Agias Marinas - Alonon, Egina, Nisi, Greece, 180 10	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Afandou Beach	Unnamed Road, Afantou, Rodos, Greece, 851 03	scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Afkule/Aladdins cavern and Hamam/Turkish bath		scuba-diving spot	☆☆☆☆☆ (0 Reviews)
Afrata Crete	Unnamed Road, Afrata, Chania, Greece, 730 06	scuba-diving spot	☆☆☆☆☆ (0 Reviews)

Figure 7.21: Spot finder page

Figure 7.22 presents the details page for a kitesurfing POI. It contains valuable information related to the kitesurfing activity, like information for the wind the water and the beach for that specific location



My Map > Lefkada / Agios Ioannis (Milos Beach Club)

Name: Lefkada / Agios Ioannis (Milos Beach Club) **Level:** Intermediate, Expert

☆☆☆☆☆ (0 Reviews)

latitude: 38.84280014038086 **longitude:** 20.6835994720459

Best Months:	May, June, July, August, September
Crowded Months:	No information
Features:	Shop, Rescue, Shower, Toilet, Food, Parking
Beach Type:	Sand, Pebbles
Beach Size:	Big
Beach Hazards:	Rocks, Other

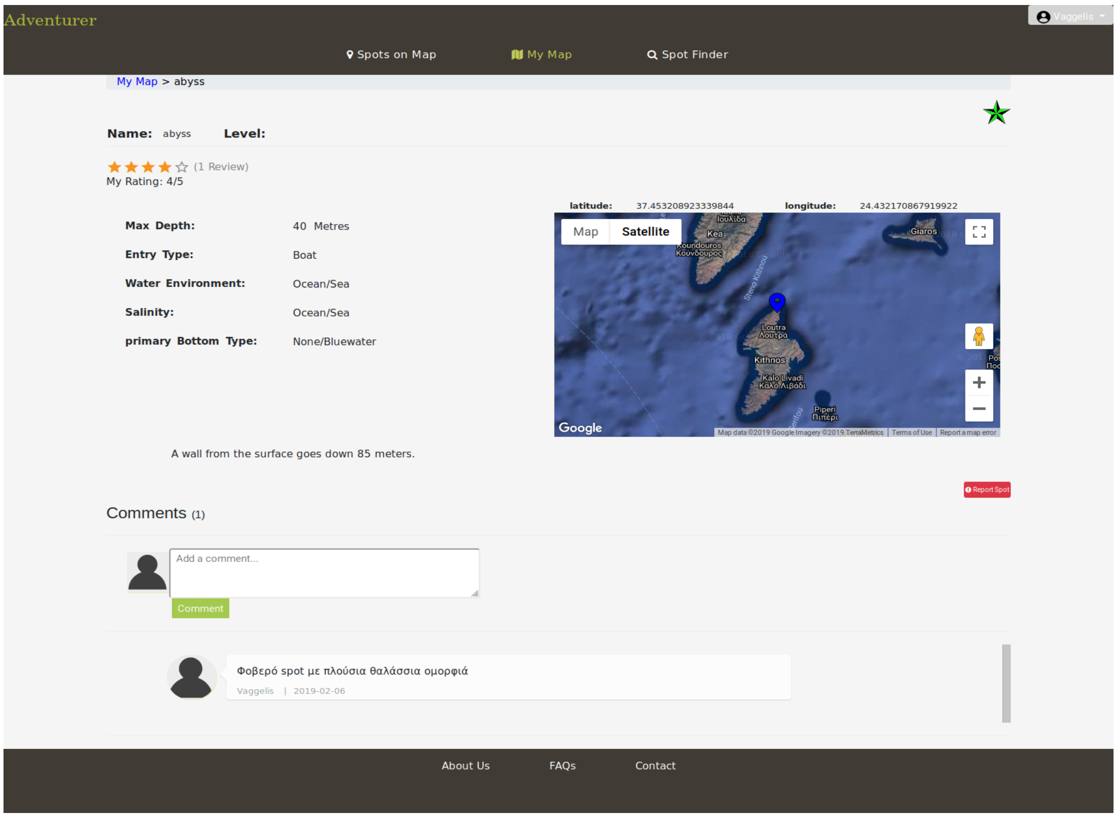
Water Type:	Ocean	**Wind Type:**	Thermal
Water Hazards:	Currents	**Wind Best Direction:**	N, W, WNW, NW, NNW
Water Quality:	Crystal clear	**Wind Main Direction:**	N, W, WNW, NW, NNW
Best Tide:	Tide independent		

Comments (1)

Το spot είναι σε πολύ καλή τοποθεσία και βγάζει καλό αέρα για kite!
user | 2019-02-06

Figure 7.22: Kitesurfing details page

Figure 7.23 presents the scuba-diving POI details page, which just like kitesurfing page contains valuable information for the scuba-diving activity on that specific spot. In more detail, it contains information about the water like among others max depth and the primary bottom type.



My Map > abyss

Name: abyss **Level:**

(1 Review)
My Rating: 4/5

latitude:	37.453208923339844
longitude:	24.432170867919922

Max Depth:	40 Metres
Entry Type:	Boat
Water Environment:	Ocean/Sea
Salinity:	Ocean/Sea
primary Bottom Type:	None/Bluewater

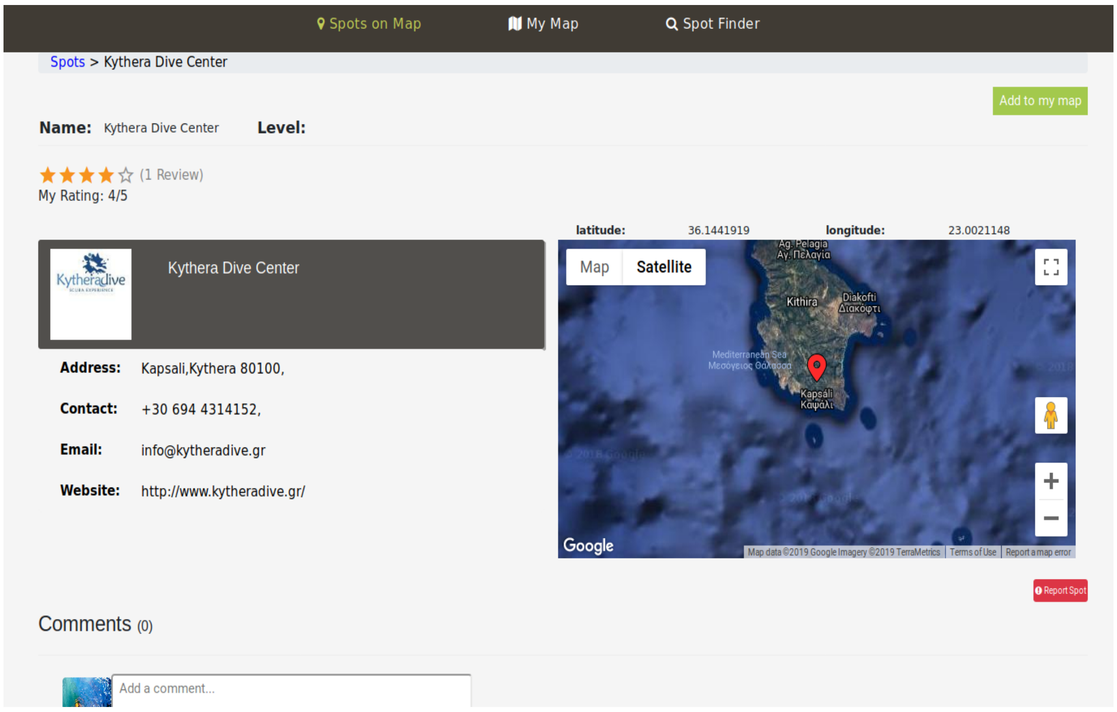
A wall from the surface goes down 85 meters.

Comments (1)

Φοβερό spot με πλούσια θαλάσσια ομορφιά
Vaggelis | 2019-02-06

Figure 7.23: Scuba-diving spot details page

Figure 7.24 presents the scuba-diving school details page. It contains information about the address, contact info and possibly a website.



Spots > Kythera Dive Center

Name: Kythera Dive Center **Level:**

(1 Review)
My Rating: 4/5

Kythera Dive Center

latitude:	36.1441919
longitude:	23.0021148

Address:	Kapsali,Kythera 80100,
Contact:	+30 694 4314152,
Email:	info@kytheradive.gr
Website:	http://www.kytheradive.gr/

Comments (0)

Figure 7.24: Scuba-diving school dtails page

Figure 7.25 presents the login page.

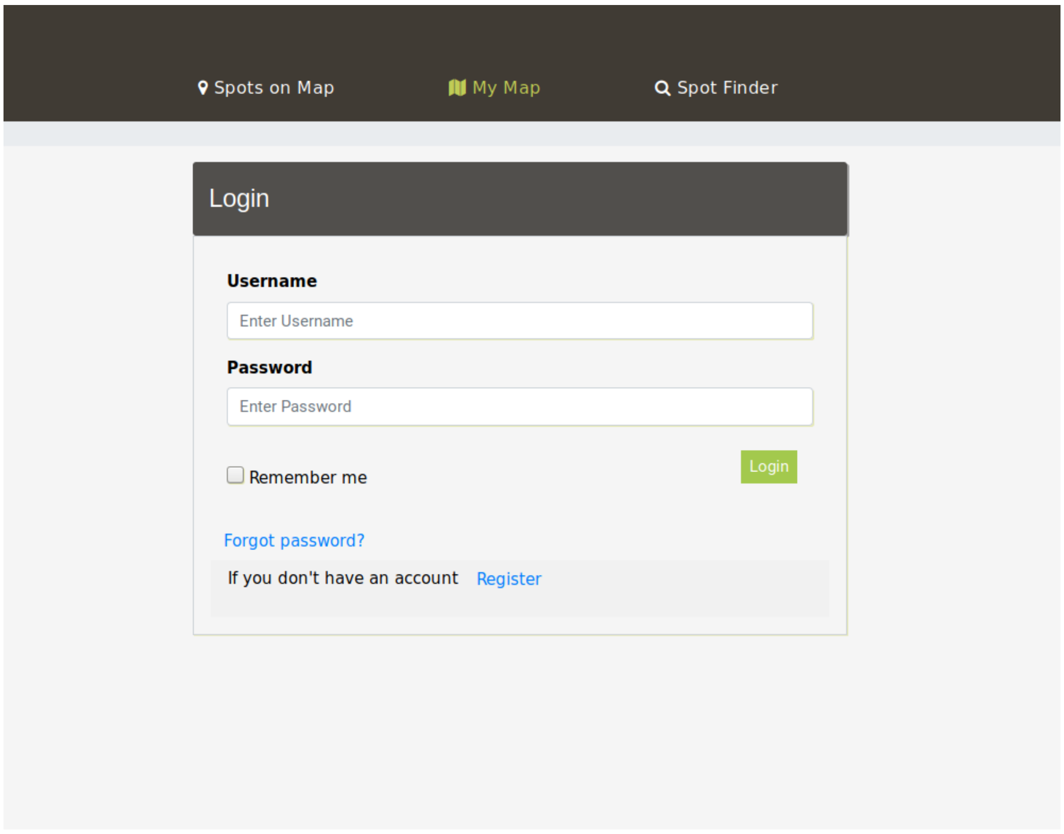


Figure 7.25: Login page

Figure 7.26 presents the profile page where user is able to edit his personal info and interests, change his password and update his profile picture.

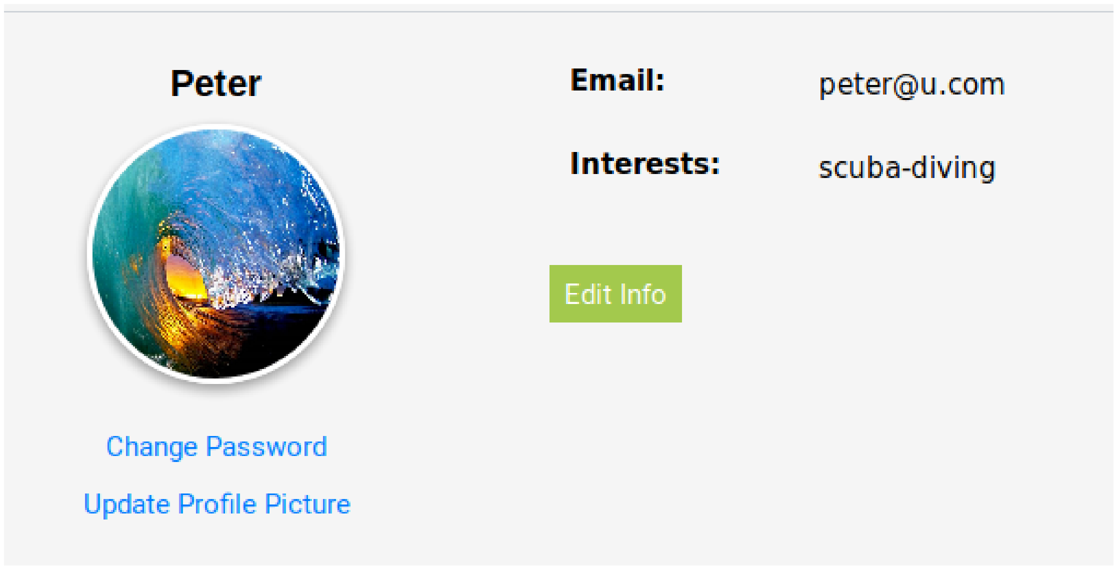
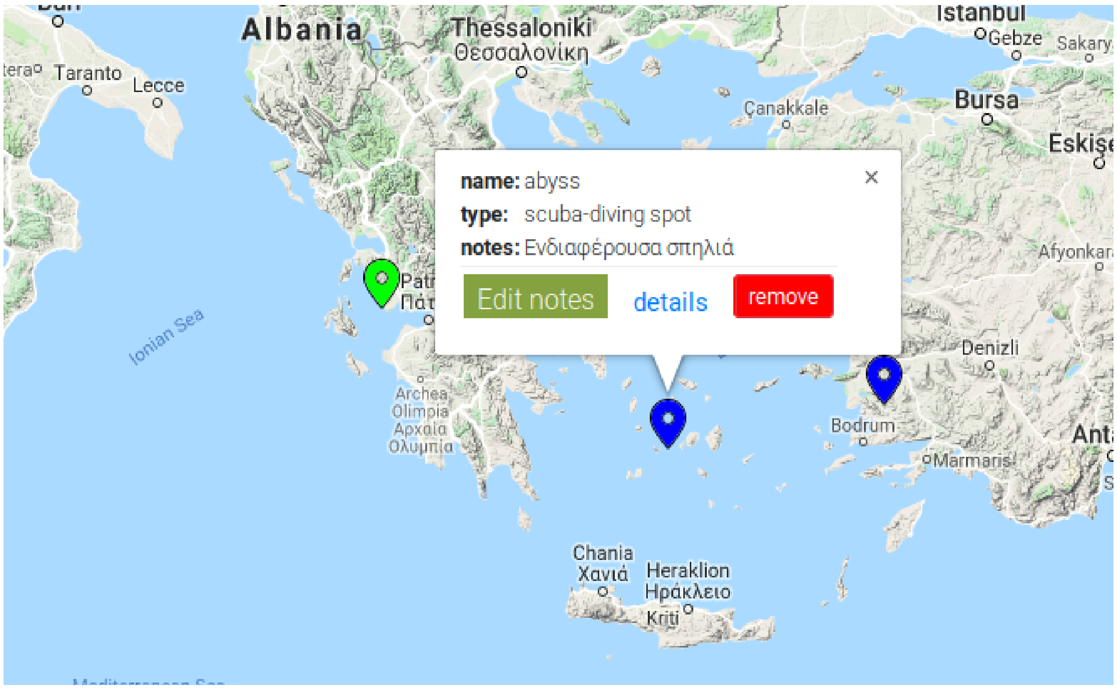


Figure 7.26: Profile page

Figure 7.27 presents the personal map page which can be accessed only if the user is logged in.



name: abyss
type: scuba-diving spot
notes: Ενδιαφέρουσα σπηλιά
Edit notes details remove

Figure 7.27: Personal map page

7.3 Evaluation

The GUI evaluation is an important process which specifies the effectiveness of the interface as well as the satisfaction of the User Experience (UX). Enterprises and UX experts use various techniques to evaluate a GUI. Some of the most popular methods are the the heuristics and the think aloud evaluation methodologies. Heuristic evaluations are performed by UX experts and determine weather or not the implemented interface fulfills global rules. Think aloud approach, on the other hand, specifies a series of tasks that some individuals must actualize on the platform. Both the individuals and the series of tasks should be selected carefully with ultimate goal of providing feedback for their UX. While the individuals try to complete the specified tasks they should, as specified by the name of the approach, be thinking aloud.

Think aloud was chosen to perform the evaluation of the implemented UI. Specifically, three (03) individuals were chosen a twenty seven (27) years old kitesurfer, a thirty (30) years old web designer UX expert and a fifty (50) years old individual. The 27 year old kitesurfer was chosen due to his familiarity with the website's content. The web designer was chosen because he can provide feedback related to UI groundrules and the 50 year old was chosen to observe how a non-familiar user to web technologies would behave.

The evaluation outcome was pretty satisfying. All the users found their workarounds, fulfilled all the specified tasks and provided the valuable feedback. The evaluation clarified that most of the pages should provide a clearer overview of he content and the functionality they provide. In greater detail, all the reported issues are presented in table 26.

Table 26: Evaluation Outcome

ID	**Reported Issue**	**Recommendation**	**Importance**
01	Use more distinct, consistent and clear buttons.	Make a button component and use it across the UI	high
02	On user registration the application should suggest the user to set his personal info and improve his UX	After successful user registration the application should guide the user to set his profile	high
03	The my profile menu should be more prominent.	This could be achieved by using a different colour or a bigger font.	high
04	Users landing on the application don't understand the functionality and the options it provides.	A home page or an About page should be developed informing users about the available functionalities of the application. This will enhance user understanding about the available options of the application and the tasks that they can perform	high
05	Some users needed guidance when interacting with the application and more specifically with the map. The fact that the pins on the map are	Add a text on the top of the page inviting users to click on the spot of their preference to get more information about it.	high

	clickable areas was not clear		
06	When using the free -text area, the zoom functionality could be improved. Two users could not identify the POI that they have searched for	Zoom more	high
07	My profile page is deficient. Users should see a clear overview of their interests and their level in each interest. Moreover, users could also see an overview of their actions across the application	• Unify the edit interests pop-up with my profile page. • Add info based on the user interaction with the system. In greater detail, a nice to have would be for the user to be able to see all his reviews and comments • Consider unifying the my map page to my profile, under my favourites	medium
08	Some of the POIs include images and a carousel to present them, both are not intuitive enough	Make the carousel accessible from both the map page and the spot finder page.	medium

8 Conclusions and Future Work

The Agile approach was chosen because a constantly scaling robust and maintainable system was desired to be implemented. Thus, the following requirements reside on the Agile backlog.

System Requirements

Up to this point we have almost decoupled the scraping and web components. The intention is to completely decouple those and integrate an independent recommendation. As a result, the system would have four separate sub-systems the scraping, the web services, the client side application and the recommendation system that could be scaled independently.

Server proxy integration

Rest empowers multiple networking configurations and simplifies request proxying. As a result, a server proxy integration would enable the various server side sub-systems run on different machines withou the client side knowing anything about it.

Security

Security is an important aspect of web applications and has to be in line with its state of the art. Furthermore, the system needs to have different user roles with different privileges and rights. As a result, the integration of a security system and user roles is important.

Test driven development

As the system scales the maintenance becomes really hard. Thus, automated tests guarantee a robust system. Test driven development specifies an approach in which each process has its automated test which as specified guarantees that it works properly. Thus, tests have to be written for both the implemented functionality and for each new process.

Weather Information Integration

Weather information are very important for outdoor activities. Furthermore, there are plenty of web services on the www that serve both real-time weather data and forecast for specific locations. Some examples are openweathermap and windy which both provide an API to consume such data. A live weather and a weather forecast presentation would be a very nice to have.

Last but not least, all of the issues specified by the evaluation process must be implemented and the UI should be reevaluated.

9 *Bibliography*

[1] G. M. Platform, “Places.” [Online]. Available: https://cloud.google.com/maps-platform/places/.

[2] D. Reinsel, J. Gantz, and J. Rydning, “Data Age 2025 : Don 't Focus on Big Data; Focus on the Data That’s Big Data Age 2025 :,” *IDC White Pap. Spons. by Seagate*, no. April, pp. 1–25, 2017.

[3] D. Brown, *Encyclopedia of Big Data Technologies*. 2018.

[4] W3C, “HTML Definition.” [Online]. Available: https://www.w3.org/.

[5] P. B. Informatics, “Term paper submitted in partial fulfillment of the requirements Web Scraping Data Extraction from websites,” 2018.

[6] J. Hedley, “jsoup: Java HTML Parser.” [Online]. Available: https://jsoup.org.

[7] K. Wagh and R. Thool, “A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host,” *J. Inf. Eng. Appl.*, vol. 2, no. 5, pp. 12–16, 2012.

[8] A. Rodriguez, “Restful web services: The basics,” *Online Artic. IBM Dev. Tech. Libr.*, no. November, pp. 1–11, 2008.

[9] S. Framework, “Understanding REST.” [Online]. Available: https://spring.io/understanding/REST.

[10] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, “Single Page Application using AngularJS,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 6, no. 3, pp. 2876–2879, 2015.

[11] K. Zoukas, “Searching Flight System,” 2018.

[12] T. E. rai. Walls, CWalls, C. (n.d.). Spring in Action, *Spring in Action, Third Edition*. .

[13] J. Wiley, A. Context, and R. Management, *Table_of_Contents*, vol. 137, no. 2. 2014.

[14] “Spring Modules.” [Online]. Available: http://techmyguru.com/spring.

[15] S. Framework, “Spring Documentation.” [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-servlet-context-hierarchy.

[16] T. Dalgleish *et al.*, “Learning Spring Boot,” *J. Exp. Psychol. Gen.*, vol. 136, no. 1, pp. 23–42, 2007.

[17] D.-I. E. Makris, “Design and Implementation of a Platform for the Development and Management of Learning Experiences in Location-Based Mobile Games,” 2015.

[18] P. Kozlowski and P. B. Darwin, *Mastering Web Application Development with AngularJS*. 2013.

[19] B. Cochior, “AngularJS at AIESEC Academy,” 2015. [Online]. Available: https://www.slideshare.net/bogdancochior/angularjs-at-aiesec-academy-15.

[20] Accelebrate, “Effective Strategies for Avoiding Watches in AngularJS,” 2014. [Online]. Available: https://www.accelebrate.com/blog/effective-strategies-avoiding-watches-angularjs/.

[21] AngularJS, “Services.” [Online]. Available: https://docs.angularjs.org/guide/services.

[22] P. Shaw, “Postgres,” 2013.

[23] A. Cockburn, “Writing Effective test cases,” pp. 1999–2000, 2000.

[24] T. Reenskaug, *The Model-View-Controller (MVC) Its Past and Present*. 2003.

[25] F. S. Systems and M. Farouk, *Infrastructure Software Modules for Enterprises*. .

[26] A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, S. Reiterer, and M. Stettinger, Basic Approaches in Recommendation Systems