TECHNICAL UNIVERSITY OF CRETE – DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

A Rich Media Mobile Web Application for Visitors and the Community of the Technical University of Crete

Christina Christodoulakis

A thesis presented to the Technical University of Crete in fulfillment of the thesis requirement for the degree of Bachelor in Electronic and Computer Engineering

8/3/2012

Acknowledgements

I would like to thank my thesis supervisor Prof. Antonios Deligiannakis, for his supervision, support and advice. I would also like to thank my thesis readers Prof. Athanasios Liavas and Prof. Michalis Lagoudakis.

Furthermore, I would like to thank some other people who contributed to the successful completion of this thesis. For their valuable advice and continuous support I would like to thank the members of the Distributed Information Systems and Applications Lab (MUSIC) of the Technical University of Crete, and especially Manolis Mylonakis, for his invaluable daily help, advice and support. For his help in preparing my video demonstrations, I would like to thank my friend Thodoris Mathioudakis. For their true friendship, all the close friends I made over the past six years. I wish them all a life of health and happiness.

Last but not least, I would like to thank my parents for their never ending love and support, and their patience with me especially over the past few months. I dedicate this thesis to them.

Christina Christodoulakis

Technical University of Crete

August 2012

Table of Contents

Acknowledgements 1

List of Figures 7

Abstract 7

Chapter 1 - Introduction and Objectives 12

1.1 Introduction 12

1.2 Outlook of Web Applications and Rich Mobile Web Applications 13

1.3 Mobile Spatial Visualization of University Objects and Events 14

1.4 Objectives of the Thesis 15

Chapter 2 - State of the Art in Web Applications: Towards Rich Internet Mobile Web Applications 16

2.1 State of the Art in Web Applications 16

2.2 Architectural Patterns in the Design of Web Applications and Web Services 18

2.3 Rich Media Mobile Web Applications and HTML5 23

2.4 Contributions 26

Chapter 3 - Requirements Analysis, User Interface Prototyping and Evaluation 28

3.1 Introduction 28

3.2 Requirements for Functionality and Interfaces 28

3.3 Personas 29

3.3.1 Rado, 57, Visiting Professor 29

3.3.2 John, 45, Professor 29

3.3.3 Jane, 18 Student 30

3.4 Use Cases 30

3.4.1 The User wants to know how to get to campus 32

3.4.2 The User wants to browse the campus map 32

3.4.3 The User wants to locate herself on campus 32

3.4.4 The User is looking for a building 32

3.4.5 The User wants to browse Building floor maps 33

3.4.6 The User wants to locate herself on Building floor map 33

3.4.7 The User is looking for a department 33

3.4.8 The User is looking for a lab 34

3.4.9 The User is looking for a person ... 34
3.4.10 User Logs in ... 34
3.4.11 The User subscribes to event types ... 35
3.4.12 The User browses events ... 35
3.4.13 The User creates an event ... 35
3.4.14 Saving data to browser cache ... 36
3.4.15 Removing data from browser cache ... 36
3.5 Storyboarding ... 36
3.6 Paper Prototyping ... 48
3.7 Evaluation and Refinements ... 52
3.8 Summary ... 52
Chapter 4 - Mobile Web System Architecture and Implementation Technologies ... 54
4.1 Introduction ... 54
4.2 Rich Mobile Architectural Styles: Online Only, Offline Only, Online Offline ... 54
4.2.1 Support for Online Data Access only ... 55
4.2.2 Rich Client Interaction Architecture and Ajax ... 58
4.2.3 The REST Service Layer in the Server ... 60
4.2.4 Support for Offline with Appcache Manifest and Web Storage ... 60
4.2.5 The Client Architecture ... 61
4.2.6 Rich Media Support ... 66
SVG Scalable Vector Graphics for dynamic graphics interaction ... 66
OpenLayers and OpenStreetMap Integration ... 67
Geolocation Support ... 68
4.3 Technologies Used ... 69
4.4 Summary ... 72
Chapter 5 - Data Base Design, Implementation and Data Base Interface ... 74
5.1 Introduction ... 74
5.2 Data Base Design ... 74
5.3 Relational Schema ... 79
5.4 Summary ... 81
Chapter 6 – Web Service Interface ... 82

6.1 Introduction 82
6.2 Service Interfaces and Message Formats 82
Get Building map features 82
Get campus parking lots 83
Get campus bus stops 84
Get campus athletic courts 85
Get all offices on floor 87
Get Labs listing on Floor 88
Get all Labs locations on floor 88
Get all amphitheatres and conference halls on floor 89
Get all elevators on floor 90
Get all washrooms on floor 92
Get Persons listing with office on floor 93
Get all Persons 93
Get all Persons listing (light) 94
Get all Departments 94
Get all Departments listing (light) 96
Get all Labs 96
Get all Buildings 98
Get all Buildings listing (light) 98
Get Person Details 99
Get Department Details 99
Get Lab Details 100
Get Building Details 100
Get Lab information Person belongs to 101
Get of Lab members listing 101
Get listing of Labs in a Department 102
Get listing of Persons in a Department 102
Get listing of Building options 103
Get listing of locations inside a Building 103
Chapter 7 - User Interfaces 105
7.1 Introduction 105

7.2 User Interfaces 105

The structure of the user interface. 106

7.3 Evaluation Indoors and Outdoors 130

7.4 Evaluation and Diverse Mobile Devices 130

7.5 Summary 130

Chapter 8 - Summary Conclusions 131

8.1 Summary 131

8.2 Contributions 133

References 134

List of Figures

Figure 1 User goals for users that aren't signed in 31
Figure 2 User goals for users that are signed in........ 31
Figure 3 Initial storyboard of Directory use. 38
Figure 4 Storyboards illustrating: a) functionality that provides the weekly restaurant menu, and feedback from customers as to which dish was the most satisfactory of the day (left) b) functionality that provides users with information on transportation means, bus schedules, real time bus location, personalized notifications when users are leaving the campus and offer rides. 39
Figure 5 Illustration of maps functionality and event browsing functionality........ 39
Figure 6 Storyboards illustrating personal location sharing and campus navigation. 40
Figure 7 Storyboards illustrating building floor map browsing, navigation and personal location sharing 41
Figure 8 Storyboard illustrations of personal location sharing and event browsing 42
Figure 9 top: Storyboard illustrations of visualization of location on campus and on floor, bottom: profle edit, location sharing visibility personalisation........ 43
Figure 10 Storyboards illustrating visualization of rides offered (above) and offering rides (below) 44
Figure 11 Storyboards illustrating an event wizard. 45
Figure 12 Storyboards illustrating floor map browsing, and displaying user location on floor map 46
Figure 13 Events, event visualization, event location visualization 46
Figure 14 Paper prototypes of screens for mapping functionality 49
Figure 15 Paper prototypes of screens for directory functionality 50
Figure 16 Paper prototypes of screens for event functionality 51
Figure 17 Architecture for server dependent functionality. 57
Figure 18 - Server side Architecture 59
Figure 19 Caching resources with the manifest file 61
Figure 20 Rich Media Mobile Web Client Architecture 64
Figure 21 Navigating pages that work offline with local storage and session storage 65
Figure 22 Navigating pages that work offline with local storage and session storage 66
Figure 23 visual comparison of detail provided by OpenStreetMap (left) versus Google Maps (right), as shown on GEOFABRIK tools 68
Figure 24 Entity University Person, with attributes 74
Figure 25 Entities, University Person and University Person Profile. Each person can have a profile, with which he accesses university web services........ 75
Figure 26 Description of University Person, Lab, Department, Building, Floor, and Indoor Location Entities, and their relations........ 76
Figure 27 A Building is located at specific GPS coordinates described in Outdoor Location. 76
Figure 28 Entities that describe Events........ 77
Figure 29 Complete Entity Relationship diagram describing the database design for the TUC Campus application 78
Figure 30 Application home menu........ 107

Figure 31 Access to Campus.......... 108
Figure 32 Route details that display when user selects an interactive line.......... 109
Figure 33 Campus map at default zoom. Map displays bus stops, parking lots and buildings. Geolocation is enabled by clicking Locate button, Map Layers are made active/inactive from Layer menu.......... 110
Figure 34 Map features are interactive, and display informative popup on selection. 110
Figure 35 Building Details Page. User arrives here, either from selecting the link on campus map, or from the building listing in the directory.......... 111
Figure 36 Directory listing, and Departments list.......... 112
Figure 37 list of People in Department.......... 113
Figure 38 Lab Listing in Department.......... 114
Figure 39 Lab details.......... 115
Figure 40 Person details page. Displays user photograph, and function in the University. Also access information (office, telephone and email). 116
Figure 41 Locate a place of interest on campus, displays a campus map and a floor map view. Maps are zoomed in on the feature, and feature is distinct from other surrounding.......... 117
Figure 42 Login dialog.......... 118
Figure 43 Home screen after login.......... 119
Figure 44 Events page, before and after login.......... 120
Figure 45 New Event form.......... 121
Figure 46 New Event form (cont'd).......... 122
Figure 47 New Event form (cont'd).......... 123
Figure 48 New Event form (cont'd).......... 124
Figure 49 Add indoor and outdoor locations to event.......... 125
Figure 50 new event dialog asking for submit confirmation.......... 126
Figure 51 Event Details.......... 127
Figure 52 new event notification is sent to subscribers with visibility privileges.......... 128
Figure 53 new event subscription email format.......... 128
Figure 54 Profile Page, Information and Subscriptions.......... 129

Abstract

We describe the design and implementation of a mobile information system that provides spatial information about indoor and outdoor locations in the TUC campus. This information includes buildings, parking spaces, athletic courts, lecture halls, offices, labs, coffee shops, etc. It can associate people and services with these locations. The information system can then be used for indoor and outdoor navigation in the campus.

The location of the mobile phone user outdoors is captured by the GPS capability of the mobile device and is displayed on top of the map so that the user can orient herself with respect to the buildings and other outdoor locations. Interactions with the map objects can display information about the name and functionality of buildings and other outdoor places. Each building has interactive floor maps associated with it. The floor maps include information about offices, lecture halls, labs, etc. The location of the user on the map is captured on request by selecting the room number nearest to the user from a selection list and the location is displayed on top of the interactive floor plan. The user can interact with the interactive floor plan to find out the function of the rooms near her, or the location of the rooms that she wants to visit. The maps and the floor plans can be scaled arbitrarily. The information system can provide information about the location of people (offices of professors, employees, graduate students, etc), displaying the campus map and highlighting the location of the building in which they are located, displaying the floor plan of the floor in which they are, highlighting the room in which they are located on top of the floor plan.

By interacting with locations on top of the university map and the floor plans the user can navigate to find more information about the location. For example, by selecting a lab from a floor plan the user can navigate to lab details and find information about the people that are associated with the lab, their room locations, their web site, etc. The information in the data base is very richly structured to allow complex navigations and information mining, including information that is associated with locations in the campus outdoors or the buildings.

The mobile information system can also operate off-line so that its functionality is delivered even in places which may not have (free) internet access and the user does not want to use mobile internet access for downloading large pieces of data such as large maps.

The mobile information system also manages information about university events such as a lecture, a meeting, an exam, a tennis game within the campus premises, etc. Events are associated with space and time, and therefore are associated with places on the interactive campus map and interactive floor diagrams. The information system organizes the events according to their time of occurrence, and the users can browse in this list to find events of interest. University users can be registered users and they can have profiles describing their interests in events. The system will only notify them for events that interest them.

The university community is structured into sub-communities based on their characteristics. Such sub-communities are for example professors, professors of the Electronic and Computer Engineering Department, graduate students, undergraduate students, university personnel, lab members of a certain lab, etc. These communities have different interests and they may have to be notified for different types of events. The originator of the event can restrict, if she wants, the visibility and the notification of events to certain sub-communities of the university. For example, an open meeting with a visiting professor who has interests related to web engineering and is going to take place between 2 and 4 in the departmental meeting room for ECE is an event that may interest only the professors and the graduate students of the ECE department. The system allows the user that creates the event to specify that the community that will be notified for the meeting is the community of professor and graduate students of ECE. Since she does not remember typically the room number of the meeting she specifies its function (departmental meeting room for ECE) and the system finds and completes in the announcement the number of the meeting room. The system automatically highlights the meeting room on the floor map. Our user can add indoor and outdoor locations to events, by progressively narrowing his selection (by type and building). System displays available locations and once added to the event, they contain a link to the campus and floor map, outlining the location on the map.

More informal community events can also be created. For example, a graduate student may want to notify the students of the lab that he belongs to that he will be departing from the university to the city of Chania with his car that is parked at a particular location in the campus, at a certain time, inviting interested fellow students for a ride.

The application developed is a Web Application accessible from mobile devices. It is a Rich Media Web Application since it is heavily focusing on interactive graphic content presentations and asynchronous communication protocols (AJAX). It also allows contextualized and personalized information access, where the context involves information relevant to time, space, and community.

In contrast to existing Rich Media mobile internet applications which are native applications developed for particular platforms, the application developed is a completely Web Application developed on top of the developing HTML 5 standards. The project showed the feasibility of developing large scale, complex, Rich Media, interactive Web Applications for mobile devices. This approach has a major advantage in comparison to developing native mobile applications which have high costs of development, maintenance and time to market, due to development in different platforms. Our application is accessible by all the major platforms of mobile devices.

To develop this application we designed a general architecture for Web Applications based on REST Web Services, as well as rich software architecture for the mobile device that may or may not be connected to the internet. The architectural design patterns that we followed are applicable to more general Rich Media Mobile Web Applications.

We present the detailed analysis of requirements, storyboarding, paper prototyping of user interfaces, evaluation of user interfaces using heuristic and think aloud methodologies, data base design and relational implementation, patterns used for the mobile web application development, the resulting Web Server and Web Client architectures which utilize local storage and caching in the mobile Web Client, asynchronous communication and synchronization with the Web Server, and capability for off-line operation. We also present the technologies and tools used for the development, as well as the difficulties found in the development process, and the solutions adopted.

Chapter 1 - Introduction and Objectives

1.1 Introduction

Web applications have been one of the primary concerns and targets for development of the IT industry for several years now. Web Applications have evolved from simple site presentations to very complex architectures that deliver enterprise functionality to the company and its partners. They are indispensable tools for every organization today to keep the employees of the organization up to date and in contact with the aims, targets, plans of the organization, for performing the everyday processes within the organization, and for promoting selling and cooperating with external bodies and final users.

Mobile phones are getting smart. They have acquired high CPU power, local storage, touch interfaces internet connectivity, and they sense the external context. They have become indispensable in the everyday life. People use them not only as a phone but also as a means of accessing the internet, connecting with their friends, finding what is going on, finding directions, finding what is available near them, and of course doing their work. Their use for accessing the internet and for running internet applications is increasing very rapidly.

Gartner predicts that by 2013, mobile phones will overtake PCs as the most common Web access device worldwide. [Gartner 2010] According to Gartner's PC installed base forecast, the total number of PCs in use will reach 1.78 billion units in 2013. By 2013, the combined installed base of smart phones and browser-equipped enhanced phones will exceed 1.82 billion units and will be greater than the installed base for PCs thereafter. **Gartner also predicts that by 2015, context will be as influential to mobile consumer services and relationships as search engines are to the Web.** [Gartner 2010] Whereas search provides the "key" to organizing information and services for the Web, context will provide the "key" to delivering hyper personalized experiences across smart phones and any session or experience an end user has with information technology. Search centered on creating content that drew attention and could be analyzed. Context will center on observing patterns, particularly location, presence and social interactions. Furthermore, whereas search was based on a "pull" of information from the Web, context-enriched services will, in many cases, pre populate or push information to users. According to Gartner the most powerful position in the context business model will be a context provider. Web, device, social platforms, telecom service providers, enterprise software vendors and communication infrastructure vendors will compete to become significant context providers during the next three years. Any Web vendor that does not become a context provider risks handing over effective customer ownership to a context provider, which would impact the vendor's mobile and classic Web businesses.

Forrester in a recent report (March 2012) presents statistics that show that the enterprises shift from the desktop applications to the web applications and the mobile applications, and that the rate of increase of the mobile web applications will generate extreme pressures to the web architectures used today. Another major problem for the mobile applications today is that they are native applications. They run on a specific platform utilizing the infrastructure of the operating system and various plugins in the platform, but they cannot run on multiple platforms. It is important therefore that we look both for methodologies that can be used for the development of attractive mobile web applications exploiting the capabilities of the new devices, and the context in which they are used, as well as to study new software tools and architectures that will deliver the web application functionality efficiently and effectively to the final users.

The university environment is a very good example of a large organization with people and infrastructure that is dispersed into space and events that are attended by the university communities as well as external visitors. Access to all this information will very frequently have to be done while the users are on the move, and the users pay only attention to events that interest them, or to their immediate needs for information, in a contextualized and personalized manner. This application environment is a very good example of a demanding mobile web application environment of the future and can be used to study requirements and architectures in this environment. In addition, the application itself is useful to the university communities and to the visitors of the university. In this Thesis we analyze the requirements of this application, study the functionality needed, design, implement and evaluate a rich mobile web application that supports this functionality, and demonstrate that the functionality is offered in a platform independent manner.

1.2 Outlook of Web Applications and Rich Mobile Web Applications

According to a recent extensive Forester study (March 2012), a drastic shift is visible and expected from the Desktop Applications to Web Applications. In 2009 62% of the applications were desktop applications. In 2014 the Desktop Applications will be 42% of the applications. In 2009 31% of the applications were Web Applications. In 2014 the Web Applications will be 48% of the applications. This trend will continue in the future. Enterprises will shift from Desktop to Web Applications in high rates.

According to Forester a tidal wave of dynamic web content is coming, and it will overwhelm the traditional Web Architectures. The dynamic web content includes dynamically constructed web pages, real time information piped into the browsers and to mobile applications, social networking services, application scripts executed inside a browser or on mobile application architecture.

Note that smart phones and tablets, both inherently demand dynamic content such as status updates on stocks, sales, news, contextual applications (for example location context), social networking applications. The dynamic Web Applications increasingly require more support from RIA (Rich Internet Application) infrastructures such as AJAX, Script tools (JavaScript, etc), XML, XSL, CSSs, graphics, animations, video, etc... RIA platforms went from 25% in 2008 to 43% in 2011, while the standard Web platforms are around 55% of the installations. The problem with RIA applications is that they currently offer the needed functionality in a platform dependent manner, and also using special purpose plugins, which makes their availability in different platforms difficult, expensive to build and maintain, and slow to deploy.

According to Forester the most important concerns of executives for the future is the offering of support for mobile devices and for dynamic content.

1.3 Mobile Spatial Visualization of University Objects and Events

The University environment offers a very good example of an environment that has a need for Rich Internet Application functionality offer to mobile devices. The Universities typically have buildings dispersed in a campus area, and the various buildings and spatial locations in the campus may offer different functionalities and may be the place where meetings and other events take place. The various community members have locations where they are related with, and various services are also offered in certain areas in the campus. Locations in the campus are indoor and outdoor. These relationships of people, events, and services with locations within and outside the buildings require a powerful graphics based visualization infrastructure which is typical of the highly demanding RIA applications.

Much of the university life is revolving around events and communities. The standard events include things like lectures, exams, talks, meetings, workshops, celebrations, external visits of schools etc. In these events various types of groups participate, i.e. the members of a lab, the members of a Department, external visitors, the members of a project, the graduate students, etc. There are also more dynamic events however, events that relate to real time university life, like a break in work for a coffee, for lunch, a tennis game in one of the courts, a walk around the campus for exercise, a meeting for a project in a course, a departure of a car for the city, etc. These dynamic events are also structured around communities of friends, collaborators, etc. In both cases the events are of interest to particular groups of people, so the information received should be personalized. The events are associated with space and time, which is contextual information which has to be supported by the application, and should be shown in mobile devices.

The development of such a web application for mobile devices involves several challenges due to the complexity of the application, the difficulties that the small screens of the mobile devices present in the design of user interfaces for mobile applications, the large number of new or even developing technologies, tools and standards involved in its materialization, the challenges imposed by the incompatibilities in the platforms of mobile devices, and the different kinds of tools supported by them, and the non standard approach to developing this application as a rich mobile web application, instead of the standard approach of developing and maintaining multiple native applications.

1.4 Objectives of the Thesis

The objectives of the Thesis are:

1. To analyze the requirements for offering university information related to university spatial information related to people, objects, events and services offered.
2. To analyze the requirements for supporting university individuals, university communities and external users with personalized spatiotemporal management of events
3. To design functionality and interfaces for offering these services on top of mobile devices
4. To explore standards, technologies, tools and architectures for offering this kind of rich multimedia functionality in diverse platforms utilizing a web application approach supported by emerging standards instead of developing multiple native applications.
5. To evaluate the application with real users, as well as the design and technological decisions taken, and the state of the technologies and tools used as well as their adoption, and propose future research and development efforts.

Chapter 2 - State of the Art in Web Applications: Towards Rich Internet Mobile Web Applications

2.1 State of the Art in Web Applications

The foundation of Web consists of a global hypertext system. The basic components are:

URL: A universal means for identifying and consuming context

HTTP: A protocol for client-server, stateless, communications

HTML: A simple markup language for presenting in diverse devices hypertext content.

Web Clients (Browsers) and **Web Servers** implement the client-server HTTP protocol. URLs are used within the HTTP protocol to identify resources or procedures to be executed by the Web Server in order to find the desired content to be returned to the Client. The HTTP requests carry the parameters for the procedures to be executed and return the results to the Client. The results are formatted in HTML for presentation. The Browser understands the structure and presentation tags of HTML, and uses them for document presentation in diverse devices. Processing of incoming HTTP requests in the Web Server is typically done by **Servlets** (typically written for example in Java Code) which are processed by the **Servlet Engine** of the Web Server. Derivation of the HTML output that will be sent to the Browser for presentation is typically done in the form of **JSPs (Java Server Pages)** by a **JSP Engine** in the Server. There are also alternative technologies that can be used within the Web Server for output preparation.

In 1997 **HTML4** was introduced and also **Cascading Style Sheets** (**CSS's).** This introduced a separation of the presentation information (style rules of CSS's) from the structure of the web content defined by the markups. CSS's reduce the need to invent continuously new tags for presentation reasons.

The introduction of **JavaScrip**t and **HTML Document Object Model (DOM)** were used to introduce behavior of the interface. DOM is a programmatic interface (API) to a parsed HTML document which is used to manipulate HTML from within JavaScript. The JavaScript code can be used for example to check for errors within the Browser when the user inserts values in a field of the page, instead of sending the values to be checked in the Web Server. If there is an error detected (for example if an error is detected when the user inserts alphabetic characters in a phone number field of the page), the error is detected by the JavaScript code in the Browser, and the HTML of the page is modified to present an error message and possibly an action to be taken, instead of sending the erroneous input to the Server to be detected and a new page with the error message and the corrective action to be sent back from the Web Server to the Web Client.

Originally the **Web Sites** were delivering mostly content. By 2004, the line between content and applications was blurred. For example shopping catalogues provided real time auction sites, allowing users to buy and sell. **Web Applications** were becoming of very high importance to the industry. They typically involve **complex application logic** which is supported by **complex navigation structures in the data base to produce multiple inferences and presentation views.** They may also involve **participation and collaboration of users** within the Web Application. Many Web Applications are **transactional** in nature, and the application logic as well as the interface should support this (for example no back buttons in the transactions, secure environments, etc.). A Web Application may utilize data from different, **remotely located, databases or queues**, and the **transaction logic** should guarantee data transformation to common formats and data integration, as well as correctness in this **distributed environment**, including **complex, business oriented, transaction protocols** (like **multilevel compensations**). In this changing environment, HTML, DOM, and JavaScript formed the **basic Programming Model of the Web Applications.**

Traditional Web Applications are essentially **Form Processing Applications**. The user interaction typically results into a request to the Web Server to change the whole page that the user is viewing. In the Dynamic HTML environment, this typically involves the retrieval of data for the various **Form Fields** from the data base, and the formation of a new HTML page (Form) which is sent to the user. There is no capability to replace only parts of the Form. In addition, there are no capabilities for **graphics presentation and interaction**, and no capabilities for **multimedia processing** like **streaming, animations, video presentations**, etc.

More recently **Rich Internet Application** infrastructures were developed to cover this need. Rich Internet Applications utilize some special pluggins which exist outside the Web Browser (such as Flash and others) in order to achieve this needed multimedia functionality. However, plugins outside the Browser are always problematic since they are not supported by the standards and a particular site may not have them. In addition since these plugins are not standards, their existence as functionality and uniform support in all platforms cannot be guaranteed. As an example, Adobe recently announced that Flash is not going to be further developed and supported in the future. Nevertheless, these plugins allowed the creation of some specialized plugin-dependent, but powerful, Web Applications.

Recently, in the Rich Internet Application environment, the **AJAX technologies** are promoted to avoid the replacement of the whole page (Form) in demanding modern Rich Internet Applications. AJAX is based on an **asynchronous communication protocol** of the client with the server. In Web Applications, asynchronous request for replacement of parts of the page can be sent from the Web Client to the Web Server, while the user keeps doing his work within the same page. When the results arrive back, they are incorporated in the specific part of the page that they belong. Impressive graphics presentations with large maps that were visualizing dynamically changing database information on top of a map were demonstrated by Google using this technology. Ajax uses **XML** and an **Asynchronous HTTP Protocol** to implement **asynchronous JavaScript based interactive applications**. The **XMLHttpRequest protocol** is not a standard yet, although the World Wide Web Consortium (W3C) has released the first draft specification for it, trying to create a Web Standard. When using Ajax technologies the Browsers can support partial refreshment of the page in the user interface. **Interactive animations** in parts of the page become also feasible. When XML is used for data transmission then XSLT is used for data transformation. However, it is not necessary that data is transmitted in the bulky XML format. It may be transmitted in the more compact JSON (JavaScript Object Notation). The Ajax technologies are criticized in that they may create a large number of requests to the Web Server. Another is that crawlers cannot use JavaScript to do indexing. Some other problems like limitations with the use of the back button and problems in bookmarking may be eliminated in HTML 5.

2.2 Architectural Patterns in the Design of Web Applications and Web Services

Web Applications are very complex software systems which have to be designed following a concrete **Software Architecture** ([Gordon 2011], [Shaw and Garlan 1996], [Taylor et al. 2010], [Brown and Wilson 2011]) in order to be able to deliver the functionality envisioned and to minimize the maintenance effort in the future. A definition of Software Architecture which is based on the book of Mary Shaw and David Garlan and others is: "Software Architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

A description of the **main themes in Architectural Decisions** (by Martin Fowler) is: "The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; that is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is." [http://www.pearsonhighered.com/educator/academic/product/]

An **Architectural Style** or **Architectural Pattern** is a set of principles that provides an abstract framework for a family of systems. An Architectural Pattern improves partitioning and promotes design reuse by providing solutions to frequently recurring problems. You can think of Architecture Patterns as sets of principles that shape an application. Garlan and Shaw define an Architectural Style as: "…a family of systems in terms of a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined. These can include topological constraints on architectural descriptions (e.g., no cycles). Other constraints—say, having to do with execution semantics—might also be part of the style definition."

The **Layered Architectural Pattern** [Microsoft 2009] focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. The main benefits of the Layered Architectural Pattern include:

Abstraction. Layers allow changes to be made at the abstract level. You can increase or decrease the level of abstraction you use in each layer of the hierarchical stack.

Isolation. Allows you to isolate technology upgrades to individual layers in order to reduce risk and minimize impact on the overall system.

Manageability. Separation of core concerns helps to identify dependencies, and organizes the code into more manageable sections.

Performance. Distributing the layers over multiple physical tiers can improve scalability, fault tolerance, and performance.

Reusability. Roles promote reusability. For example, in MVC (Model View Controller), the Controller can often be reused with other compatible Views in order to provide a role specific or a user-customized view on to the same data and functionality.

Testability. Increased testability arises from having well-defined layer interfaces, as well as the ability to switch between different implementations of the layer interfaces. Separated Presentation patterns allow you to build mock objects that mimic the behavior of concrete objects such as the Model, Controller, or View during testing.

An example of a layered architecture in the **Client Server Architecture** which has two layers **Client** and **Server**. In data base applications this architecture is used to separate the **interface concerns** from the **data management concerns.** It is also used for distribution of functionality in different machines.

For Web Applications the two layer separation may not be enough. **Multilayer architectures** have been proposed. A **Three Layer Architecture** separates the **Data Management Layer** which is responsible for the management of data on permanent storage from the **Business Layer** which is responsible for the business functionality of the application (implementing the business rules and policies and changing the state of the business data). The third layer is the **User Interface layer**. This separation essentially allows the Business Layer to communicate with more than one data base behind it and also to implement effective strategies for scalability (replicate the Business Layer servers and implement load balancing strategies).

Many Architectural Patterns for Web Applications have been described in the literature ([Abbott and Fisher 2011], [Alur et al. 2008], [Brown et al. 2003], [Gamma et al 1995], [Fowler 2003], [King 2008], [Microsoft 2009]). A very high level Architectural Pattern which covers the overall architecture of Web Applications is the **BCED (Boundary, Control, Entity, Data Base Interface) pattern**. It is a special case of the Layered Architectural Pattern. The BCED pattern promotes the structuring of the Web Applications into four layers:

The **Boundary Layer** is concerned with the user interface of the Web Application.

The **Control Layer** is responsible to interpret user input and transfer the control to the appropriate parts of the application that will implement the business logic of the application.

The **Entity Layer** contains the entities (objects) that will carry out the business logic and change the long term state of the system. The Entity Layer objects are retrieved from permanent storage, and changed values are store back to permanent storage after the execution.

The **Data Base Interface** Layer is responsible for communicating with permanent storage (data bases) for retrieving and storing back data in them. The data in the data bases are typically in a relational model format (relations, tuples, etc.), and they are converted in an object oriented format in the Entity Layer.

The data base itself is not described by the BCED pattern. The Data Bases may preexist, and the Web Application may have to assemble and convert data from several data bases. This work is the focus of the Data Base Interface Layer, which is kept separate from the Entity Layer. The Entity Layer represents and manages the content of the objects of the Web Application at run time. The Entity Layer and the Control Layer implement the Business Logic of the application.

There are several other patterns that have been described in the literature that apply to the design of parts of the overall architecture, not the whole architecture.

The **Model View Controller (MVC) Pattern** is one of several patterns that emphasize the separation of the interface from the management of the content. It describes three components in the architecture, the Model with responsibility the management of Data, The view with responsibility the presentation in the interface and the Controller who is responsible to notify the Model of any changes which are results of the interaction of the user with the user interface. The important point in here is that the Model sees neither the View, nor the Controller. This allows the Model to be independent from the View and the Controller. If this is not taken into account and the Model calls the View to notify the View about the changes in the Model, every time that a new client device has to be added the code of the Model would have to change to call the new device. Most user interface patterns are based on the **Event Processing Pattern**. User interactions with the user interface produce events of different types which are processed by the Controller. For example an event type may indicate that the mouse passed above a button in the user interface. Another event may indicate that the user pressed the button. User interface events are processed by the Controller in the MVC Pattern. Processing may result in that the controller asks modification of the user interface (for example to indicate that the button is pressed). It may also result in that the Controller calls the Model because the user has added or deleted some data.

The **Model View Presenter (MVP) Pattern** does not work in a circular fashion the way MVC does. It relies on the Presenter to coordinate everything, to take data from the Models and populate the Views. When an action is taken on the view, the Presenter intercepts the action and coordinates the work with other services, sending changes back to the Model. When the model is up to date, the Presenter takes the model changes and makes the corrections in the View.

The **Publish Subscribe Pattern** is also related with the user interface design and the MVC pattern. The pattern proposes a solution for the modifications that may be happening in the Model and how they are communicated to the View. Since the Model cannot call the View, the View could be periodically calling the Model to ask if there are any changes that have to be reflected in the interface. This would generate a tremendous traffic to the Model, even though there may be no changes in the Model. A solution is that we associate with the Model a subscription list where the clients that are interested to be notified about a change of a particular type in the Model are subscribing. Whenever there are changes in the data of the Model an event of change of a specific type is sent to the subscription list and all the subscribers that are interested in this event type are notified. The event type may be determined by the data that the user sees on his interface. The Publish Subscribe Pattern is applicable in several different design decisions, not only in the user interface. It is based on the Hollywood principle: "Don't call us. We will call you". Sometimes it is seen with the name **Observer Pattern**.

There are several Design Patterns that are associated with the Controller of the BCED Pattern. One of the most important patterns is the **Application Controller Pattern**. It advocates that there should be a centralized point for handling the screen navigation and the flow of the Web Application. The Model View Controller (MVC) Pattern can make some of those decisions. Typically however an MVC controller applies to one view (one page), and there are many MVC controllers. When the application is complex this can lead to duplicate code because several controllers need to know what to do in certain conditions. In addition when the control logic is highly distributed it is difficult to have a cohesive view on how the application behaves. The maintenance and the expansion of the application become very difficult. The Application Controller Pattern places all the flow logic in one Application Controller thus avoiding all the above problems. In some cases the Application Controller may become very heavy. In these cases the Application Controller can only make the central work, and then invoke more specialized **Use Case Controllers** to do the work of the specific use cases. In addition, the Application Controller often has some functions to perform which are rather independent on the request. Such functions relate to validation on if the client has a valid session, if the browser type of the client is supported, what kind of data encoding is supported, data conversions, traffic logging, etc. These functions are state independent and independent on each other or the order in which they are executed. Often they are separated from the Application Controller and create a series of **Intercepting Filters** that can be applied before the Application Controller leaving the Application Controller with the flow of control functionality.

With the advent of technologies such as REST Applications, Web Services, Widgets, and Mash up applications, it becomes important to be able to organize the functionality offered by the system so that it is seen at high level of overall functionality offered (and not the thousands of objects that constitute the actual implementation of the system). This API would then be used to support the derivation of new applications on top of the infrastructure already available by the system. The eventual export of this functionality to other systems and applications could be done via Service or REST interfaces for example.

General APIs that expose the business logic of (possibly partitions) of the application so that they can be reused are described by **the Application Façade Pattern**. The Application Façade Pattern is an optional component which presents a simplified interface to the business logic components, often by combining multiple business operations into a single operation.

There are also several patterns described for the Data Base Interface Layer of the BCED Pattern. **The Data Access Object (DAO) Pattern** delegates all the storing and retrieving to and from the persistent storage of the entities to custom Data Access Objects instead of embedding all the storage and retrieval functionality (which may involve conversions, etc.) to the entity object itself. Another related Pattern is the Data Transfer Object which may use the Data Access Object to obtain data to build a composite transfer object it needs to assemble and send to the client. The Data Transfer Object is a useful pattern because otherwise higher tires that want lists of data from lower tiers of the architecture may use multiple method calls across tiers. In many cases the calls may be remote object calls which are very expensive.

A **Web Service** is a method of communication between two machines over the Web. In the recent years great emphasis was placed to define standards and architectures for Web Services. They come with the name of **Web Services** or **SOAP based Services**, or **Service Oriented Architectures**. Their basis for data exchange is XML, and the emphasis is on message based communication between the machines based on the Simple Object Access Protocol (SOAP). SOAP messages carry XML data, and are often used to materialize a platform independent RPC protocol. By its nature is strongly typed approach to distributed system development. Many standards have been developed and are developing for supporting various functionalities of the distributed systems like exposing the interfaces for the calls (WSDL), security, authorization, orchestration of services, composition of services, etc. [Alonso et al 2004]. The SOAP approach requires heavy investment in learning and the decisions for the protocol stack to be employed every time are difficult, and thus is more appropriate for materializing critical business processes of large organizations.

Very recently the style of **REpresentational State Transfer (REST)** [Fielding 2002] style of service design and implementation has been given great emphasis due to its simplicity of use, smaller learning curve, and lighter implementations (the XML wrappers are very expensive to transfer). The REST style of service implementation is based on the following principles:

(1) Resource and Service Identification through URIs. A Restfull service exposes both data and services in a uniform way.

(2). Uniform Interface: Resources are manipulated using PUT, GET, POST, and DELETE. GET retrieves the current state; POST transfers the new state in the resource.

(3)Self Descriptive Messages: Resources are decoupled from their representation. Content can be accessed in a variety of formats (HTML, XML, PDF,...). Metadata about the resource is available and used to control caching, negotiate the representation format, perform authentication.

(4) Statefull Interactions: Request messages are self-contained. The resource is stateless. This involves explicit state transfer, where the state is embedded in the message.

The disadvantages of the REST style include that it provides less security in comparison to developed security standards developed in SOAP services, and that the type handling is better in SOAP. However the fact that in REST the Server is stateless allows services of services to be built to any depth [Pautasso et al. 2008].

REST implementations often use the JSON format for transferring data instead of the verbose XML format which is heavy in the internet.

The **JavaScript Object Notation (JSON)** format is a text based standard designed for human readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays called objects. It is language independent and parsers are available for many languages, not only JavaScript.

The **JSON Schema** is a specification for a JSON-based format for defining the structure of JSON data. The JSON Schema provides a contract for what JSON data is required for a given application and how it can be modified, much like the XML Schema provides for XML. JSON Schema is intended to provide validation, documentation, and interaction control of JSON data. JSON Schema is based on the concepts from the XML Schema.

2.3 Rich Media Mobile Web Applications and HTML5

Mobile applications are special because the nature of **mobile is anywhere, anytime** [Gualtieri 2011]. Users take their mobile devices as they would a watch, wallet, handbag. Mobile devices are personal. They are an integrated part of life. People feel naked without their mobile. The **mobile context** [Beyer and Holtzblatt 1998] is king when it comes to mobile application design. Context describes the environment in which the users might use the application and the features that would be most valuable in this environment. Mobile application design should take into account l**ocation** (where she is) and **locomotion** (needs that the user has during movement, for example walking, in a car, in a boat, etc). Another best practice for mobile application design is **immediacy**, the ability to have the appropriate functionality available, or even to anticipate the user needs based on his location and locomotion. Mobile application design should also respect **intimacy**, or the personal differences among users. Some users for example may want to be notified for the presence of a special offering in a shop nearby, others may dislike it intensely. Finally mobile devices have **different characteristics**, capabilities, screen sizes. The interface should adapt in the differences offering the best presentation possible.

To design pleasant and usable user experiences ([Buxton 2007], [Moggridge 2007], [Snyder 2003]) the designer should create and refine detailed **mobile personas** that will represent the segments of the users of its application. Some companies like Forrester Research even create life-size cardboard figures of their personas. Personas should be refined and tested by going to places where the actual people that the personas represent to observe the life and the emotions. The design should focus to deliver for the mobile context and for **emotional design** [Norman 2004]. The designer should **validate the design in a**

mobile context ([Nielsen 1993], [Nielsen and Mack 1994], [Beyer and Holtzblatt 1998]), not sitting behind a desk.

As the power of the mobile devices and the battery life increases **Mobile Web Applications** become very important. Mobile Web Applications are **inherently contextual in nature** and **more personal** than the traditional Web Applications. The users carry their smart phones at all times and they can use them for both pleasure and work. Very frequently the use of the applications will depend on the spatial context around the user. The user may want to go to a particular location where a meeting related to her job is taking place, and she needs the specific special instructions and visualizations to go there. In another case the user just browses in a city that she is visiting and she is interested to explore what is near her, and what is offered near her. In many mobile applications the context involves place and time. **Events** that are taking place in a specific place and time are typical examples. Thus mobile applications will very frequently have to support Rich Internet Applications with interactive graphics, pictures, video, sound, animations, etc. The smart phones are also a prime means for **social communication and engagement** [Kerpen 2011]. In this example context may be the friends of the smart phone user. They should be notified for his availability for company (if she wants), his location (if she wants), and she should be able to ask them for recommendations or advice if he needs to. On the other hand the owner of the smart phone should be able to **personalize the application** according to his interests, specify what kinds of events she is interested to know about, what places are of interest to her, and from what friends she wants to hear from. These are significant examples of the need for personalization of the mobile web applications and involve asynchronous communications.

However, the mobile applications built today are **native applications**, specific to a particular device type (for example iPhone). The applications exploit the functionality offered by the operating system of the mobile to offer the desired functionality to the user. For example if the user needs access to local storage the function calls will be specific to the operating system of the device. The same is of course true when the applications try to offer **Rich Media Mobile Web Application functionality** with plugins that relate to the specific platform, or when they try to access contextual information (GPS, direction, etc.) through the device. In addition device sizes vary drastically across manufacturers. This is a very major problem for software manufacturers for Rich Mobile Web Applications. Different applications have to be built for different device types, and this is very expensive to build, very expensive to maintain, and delays the introduction of products in large portions of the market (and in many cases only certain portions of the market will have the applications).

HTML5 ([Freeman and Robson 2011], [Smitt and Simpson 2011]) is a developing standard which aims to meet the requirements for a standard that will avoid all the above problems for developing Rich Mobile Web Applications. The HTML5 standard will probably take many years to stabilize, but some of the functionality that it proposes is more stable and as such the major equipment and operating system manufacturers are already supporting it. The following is a list of some of the more stable and important functionalities described by the standard.

The Canvas Element: The Canvas element is a resolution dependent bitmap canvas that can be used for rendering graphs, game graphics or other images on the fly. The user is given a rectangle where he can draw. The canvass model is an imperative style to draw. Essentially it stores commands for drawing. JavaScript can be used to draw anything. Scripts allow the user to draw from point to point. The Canvas API is a set of functions defined in HTML5 for drawing shapes, defining paths, etc. The canvas element is an immediate mode of graphics rendering in application program interface. It does not save the list of objects rendered before. To recreate them you have to reissue all the object drawing commands you gave before. If the user simply wants to move the object in another location he has to redraw the object. Canvas is raster based. It forgets how the objects were drawn. Conceptually canvas is lower level API on top of which you could support other APIs like SVG.

The SVG Element: The SVG element allows specification of vector based graphics elements in a declarative manner. An image is described as a collection of objects using XML. SVG scene graphs allow event handlers to be associated with declared objects. This is more difficult to do in canvas.

The Video Element: The video element allows embedding video within a web page without using external plugins (like flash and Quick time). This could not be done until now. The video formats may be different and there is no universally accepted standard yet. The h264 baseline video and the WebM (open source) are the main formats supported by different platforms.

Geolocation: HTML5 supports a Geolocation API. There is more than one way to estimate the device location. The IP address, the wireless network connection, which call tower the device is talking to, the GPS location computed by satellite info, captured by specialized hardware. The GPS hardware draws a lot of power on the device. Thus phones and other devices turn it off until it is needed. There is a startup delay to find the satellites. Google maps initially uses the call tower and gives an approximation (large circle), then uses triangulization from other call towers which gives better results. Then the GPS (whenever the satellites are accessible) gives an accurate point.

Many other new elements are described by HTML5 including **<MathML>, <section>, <header>, <new>,** etc.

Support for Off-line Web Applications: Using HTML5 one can build a web application that works off-line. This is similar functionality to the mail or Google docs. It is clear that this is a very important functionality for mobile application environments since the mobile device may not have access to the network, or it may be very expensive or battery consuming to access data through the network from the current location of the user. An off-line Web application starts out as an on-line web application. The first time of access of the Web Server, the Server informs the Browser which files it needs in order to work off line. Files can be HTML, JavaScript, images, video, etc. Once they are loaded the application can work with them even if the Browser is not connected in the internet. The Browser can detect if there is an internet connection. When the Browser becomes connected to the internet again any changes made to them are transferred to the Web Server.

Resource caching: All web browsers store resources like HTML, CSS, images and JavaScript when page first loads, but the developer and user have no control over how long the browser will continue to cache the resources. HTML5 allows persistent caching of resources with Appcache. Developers put together a manifest file, which essentially lists all URLs of resources that the browser should cache in persistent storage. The Web Browser reads the manifest file, downloads the resources, and caches them locally. Once cached, the files are updated in browser cache only when the manifest file is modified on the server, not when the files themselves change. Once the application has been accessed with an internet connection, and the resources are stored in persistent cache, then from then on, with or without internet connectivity, the application always accesses the resources stored locally. Applications can be made to cache data lazily, ex. Separate the manifest file into partitions that are placed in different pages. A partition will be downloaded the first time that page is visited.

Web Storage: Applications need to access and generate data even when they are offline. To store generated data they need local storage in the client. Native applications (applications managed by the specific operating system where the Browser resides) use the functions of the operating system to store and retrieve application data. This makes them operating system dependent applications, and the solutions do not generalize since they are not supported by Browser standards. HTML5 provides standard functionality for storage and retrieval of named key/value pairs locally within the Browser. The data is not transmitted to the Web Server, except if the application explicitly does it manually. Since this functionality is implemented within the Web Browser it can run in any platform. JavaScript is used to access the data from a local storage object. A local storage object can be treated as an array. Data can be any type supported by JavaScript including strings, Boolean, integer, floats. Storage events can be used to track the changes made to the objects in local storage. The storage event object stores the id of the old value, the new value, and the URL of the page that caused the change. By default 5 Mbytes local storage is allocated to the browser application. Changes that are made while the device is disconnected persist. A resume function allows the storage in the server when connected. The data in the local storage can only be read and modified by the web site that has created it.

Web SQL Database: HTML5 provides an API wrapper around an SQL data base that can be accessed from JavaScript. SQL is a SQLLite version, not really supported by the standards.

Indexed DB: HTML5 provides an object store API. The model includes databases, records, and fields. Methods are provided to access the content.

Web Workers: HTML5 provides a standard way for the Browsers to run JavaScript in the background. It allows multiple threads to be run at the same time. The threads can do checks, mathematical calculations, access local storage as response to user actions.

2.4 Contributions

The Rich Media Mobile Web Applications is a very demanding but also very promising class of web applications with large expected market in the near future. Today the State of the Art in mobile applications is that they are native applications specific to a platform. Thus many different applications have to be generated and maintained in order to support similar functionality in different platforms. HTML 5 is a developing standard that enables platform independent rich mobile applications that support advanced multimedia user interactions, in a contextual and personalized environment.

An example of demanding rich mobile web application is a map based mobile information system for a university that supports information finding about the location of objects and events. Very few such applications exist in top international universities, and they are all implemented in native platforms. In addition, these systems in their majority do not also offer the rich browsing and information finding capabilities related to people, places, and events in the university life within the buildings and in the outside spaces of the university, and they do not offer contextualized and personalized information.

Our Thesis shows the feasibility of building such systems in a platform independent fashion and proposes system architecture, architectural patterns, functionality offered, user interfaces, and appropriate tools and methodologies for building such systems.

Chapter 3 - Requirements Analysis, User Interface Prototyping and Evaluation

3.1 Introduction

Software development requires that design and development team work together following a series of steps. These steps ensure that, when finished, the product under development meets user needs, with an intuitive interface. They also ensure the time and resource constraints are kept in perspective.

Once user groups and user needs are defined, they are further mined to define user goals, and a set of user requirements can be defined. Once designers know what the product is about, what it offers and who it is offering it to, they can start designing the interfaces to be usable. After iterations of evaluations and design, interfaces are ready, application logic has been decided upon, and developers can build the application. Throughout the development process all steps involve constant evaluations. The more the evaluations in the early stages of design and development, the less and less expensive are the problems to fix down the way, and the better the flexibility to accommodate future work.

In this Chapter we describe the process of requirement analysis and prototyping that we used. The tools that we used were use Personas definition, Use Cases, Storyboarding, and Paper Prototyping. During the phases of prototyping we used Brainstorming and Think Aloud methodologies for exploring new ideas and evaluating the interfaces. The evaluations lead to rounds of new designs.

3.2 Requirements for Functionality and Interfaces

The application we are developing is to function as an aid to visitors of the University Campus, looking for buildings, services, departments, labs, persons, facilities or events of the University. Such information is mostly needed when users are mobile, and therefore we seek to target **mobile** devices. Mobile devices are devices that a user can view on the go, and range from phones to tablets to laptop computers. Our emphasis is in the new generation of smart phones that are personal devices carried anytime anywhere.

Up to now, mobile internet applications were built based on functionality provided by particular platforms. They were therefore platform dependent. The development of platform dependent (native) applications is a very expensive and very time consuming proposition. The W3C is however in the process of developing standards (HTML 5) that will make possible the building of Rich Media Mobile Web Applications. Although this process will last for several years, some aspects of the emerging standards have already been incorporated by new smart phone devices and tablets, thus enabling platform independent, Rich Media Mobile Web Applications.

Our application must therefore be cross platform, and shall be a web application, and all that our user’s devices need to have to access it, is an internet browser. Computers, tablets and smart phones are all equipped with browsers, of different companies and capabilities. The application must therefore also work on any browser.

All devices mentioned vary greatly in dimensions. Our application must first target the small screen sizes of mobile phones, and adapt to greater screen sizes of tablet devices and laptop and desktop computers.

3.3 Personas

Personas are fictional users that correspond to the different profiles of real world users. By describing user personas, the designer understands user needs and can deduce the goals the users will be looking to achieve when using the application. In the process of creating personas we investigated many types of potential users of the application and what they would like to do with it. These involve visitors of the university that they want to attend various events (student graduation, etc.), visiting professors, professors of the university that have meetings or attend talks within the campus, visitors for project or workshop meetings, visitors of the city browsing the campus, new students of the university that do not know the location of facilities, professors and labs, students, professors and personnel of the university that wants to know when and where university events are taking place, students that take exams in different locations on the campus, persons notifying a university community (lab, department, etc) for some event, and others. We have distilled the requirements of those classes of users and we have concentrated into a more limited set of personas that we believe that they cover all the functionalities needed by the above groups of people.

3.3.1 Rado, 57, Visiting Professor

Rado is a professor from another University in Europe. He collaborates frequently with members of the Technical University of Crete, and his collaboration involves visiting the University for project-meetings and workshops. Rado would find use in an application that could help him get to the campus, navigate around the campus, find people’s offices or the labs he must visit, and find locations where project events will be taking place.

3.3.2 John, 45, Professor

John is a professor at the Technical University of Crete. He has meetings with professors from other departments, which he is not terribly familiar with, and wants to be able to find their offices. He is also interested in lectures from visiting professors, but doesn’t always remember where the amphitheatres are on campus, when they are not in his building. He often is in charge of notifying the community of academic events, such as lectures, thesis defenses of his students, or exams. He coordinates international projects which often meet on various locations on the campus, and he wants to inform his international colleagues how to find the campus, as well as about the various events of the meetings and the locations in which will take place.

3.3.3 Jane, 18 Student

Jane is a first year student at the Technical University of Crete. She has complete ignorance of the grounds and the services of the University, lecture halls, professor offices, labs etc. She would need a map of the grounds with facilities outlined, information about buildings and facilities. She is eager to interact with other students in her Department, meet up with them for lunch at the University restaurant or cafeterias, and would like to hitch rides into town from the campus, as buses are scarce and expensive.

3.4 Use Cases

Use Cases describe the main User Actors that utilize the functionality of the system, and the interactions with the system until they achieve their objectives or fail.

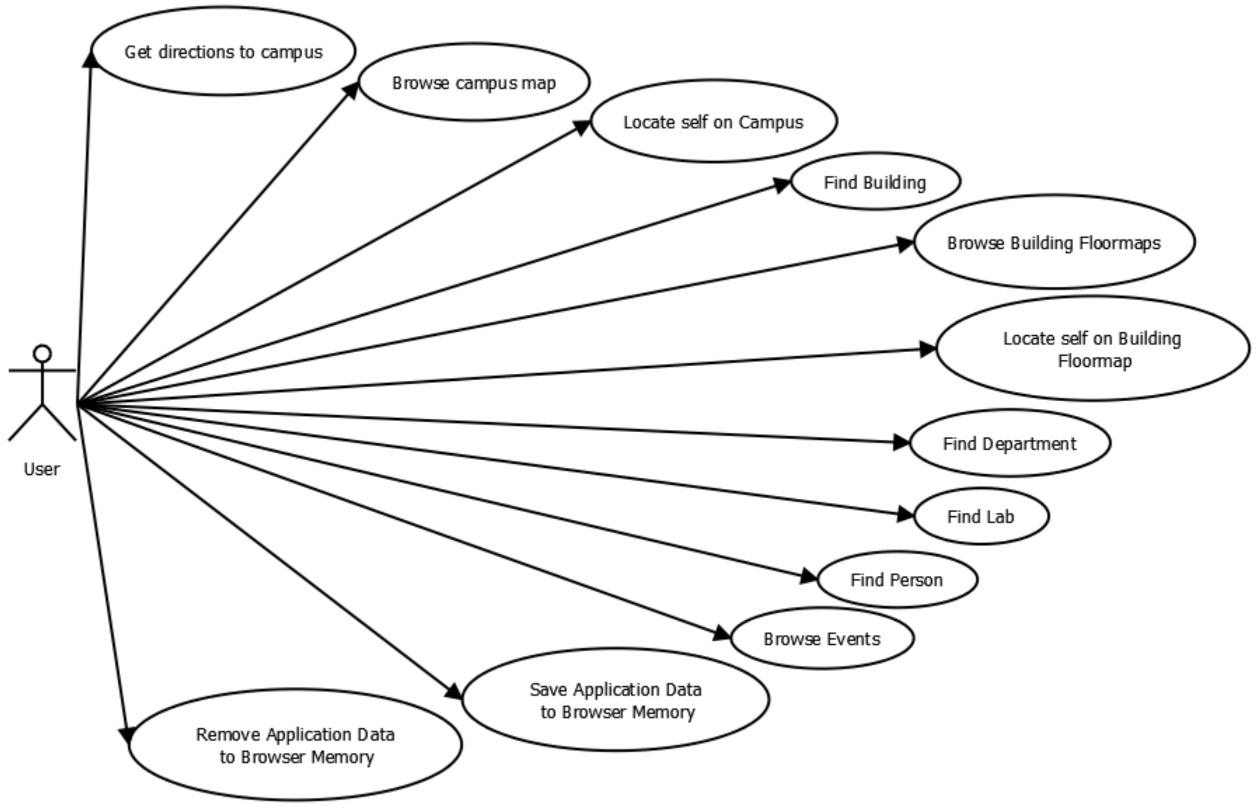

Get directions to campus
Browse campus map
Locate self on Campus
Find Building
Browse Building Floormaps
Locate self on Building Floormap
User
Find Department
Find Lab
Find Person
Browse Events
Save Application Data to Browser Memory
Remove Application Data to Browser Memory

Figure 1 User goals for users that aren't signed in

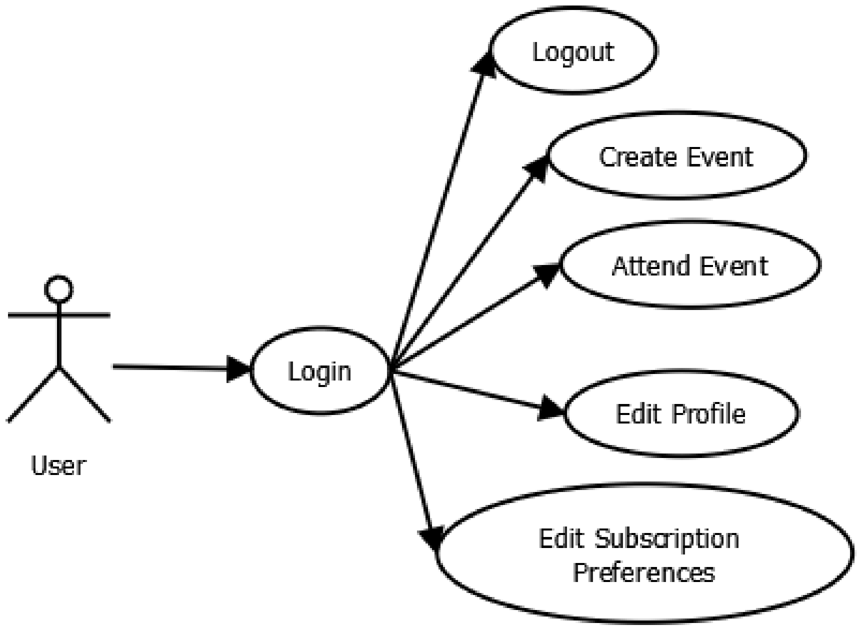


Figure 2 User goals for users that are signed in

3.4.1 The User wants to know how to get to campus

The User selects Access to Campus from the application main menu. The System displays a map of the area of Akrotiri with interactive lines outlining routes leading to Campus from places of interest (Airport, City center, Harbor, Highway). The User selects an interactive line and the system displays an information page with route description and transportation information from that point to the University Campus.

This use case interests mainly University visitors like Rado. Rado might view the route from his computer back home, and when he gets to the airport he gets his phone out and interacts with the interface to see how to get to the campus for his meeting.

3.4.2 The User wants to browse the campus map

The User selects Maps from application home menu. The System displays the map of Campus, with interactive buildings and icons displayed for parking lots, bus stops, buildings and other facilities. The User selects a building and system displays tooltips with the building code and name. If the User selects tooltip, the system displays Building details.

3.4.3 The User wants to locate herself on campus

The User selects Maps from application home menu. The system displays the map of Campus. The User selects locate from the page footer, and the system pinpoints the device GPS position on the map, by drawing a position indicator on the map and zooming in to that position.

This would be useful mostly for users visiting the Campus for the first time, or users that are fairly new to the campus. In our case, Rado might want to locate himself on Campus, and so would Jane, during her first few visits.

3.4.4 The User is looking for a building

The User selects Directory from application home menu. The User selects Buildings from the Directory menu, browses the list of buildings or uses the filter at the top to filter through the long list of buildings and finds the building he is looking for.

This is again useful for users that are relatively new to the University campus or in the case that a campus is large, to all members or the community.

3.4.5 The User wants to browse Building floor maps

The User selects Directory from application home menu. She then selects Buildings from Directory menu, and browses the names of buildings till she finds the one she wants. The user selects building and the system displays building details.

Variation: the user selects Maps from the application home menu. The system displays the map of Campus, with interactive buildings and icons displayed for parking lots, bus stops and other facilities. The user selects buildings and system displays tooltips with the building code and name. The user selects a tooltip and system displays building details.

The user selects a floor from the list of floors provided in building details page, and system displays the floor map with interactive labs, offices, and conference rooms and lecture halls, and facilities (stairs, elevators, exits, café) are shown with icons.

This use case interests all users, visitors and new and seasoned members of the University community. People only know the buildings they work, study in, or visit on a daily basis. When visiting a building for the first time it is useful to know where simple things like facilities (washrooms, café, etc) are, and also people's offices, labs and lecture halls.

3.4.6 The User wants to locate herself on Building floor map

The user is viewing a building floor map. The user selects the locate button from the page footer. The system displays a modal window with a list of all location codes on that floor. The user looks around her for a location code. The user filters through them and selects the location code she found near her. The system displays a graphic target around the centre of the map feature whose code the user just selected.

3.4.7 The User is looking for a department

The user selects Directory from application home menu. Then she selects Departments from the Directory menu, browses the short list of departments, and finds the department she is looking for. The system displays department details.

The user might know which building houses the Department. Therefore she may first look for the building and then select the department from the list of departments housed in the building. The system displays Department details.

This is useful mainly to visitors, but on occasion when members of the community have intra departmental relations it is useful, as it is also a quick filter for other directory lists like labs and people.

3.4.8 The User is looking for a lab

The user selects Directory from application home menu. She then selects Labs from Directory menu and browses through lab listing or uses the filter at the top of the list to filter through the list of labs. The User selects a particular lab and the System displays Lab details. The user selects show on map and System displays the Campus map with the building that the lab is located in outlined. She then selects locate on floor and the system displays the floor plan that the lab is located on with the lab distinctly outlined.

When the user is looking for a lab, he might already know what department the lab is located in. In this case, the User selects Department as described above, and then she selects Labs in this department. The user browses the list of labs, and she selects the lab he was looking for. He then views the locations of the lab.

3.4.9 The User is looking for a person

The user selects Directory from application home menu. Then she selects People from Directory menu. The system displays an alphabetical list of all people that work or study at the University. The user can browse the list till he finds the user he is looking for or she uses a filter facility that is located above the list of people to search for the person she is looking for. The user selects Person, and the system displays Person details. The user can click on the Person's contact details to call or email her, or to select her office number if supplied, to see the location of her office on Campus map and Floor map as with the case of Labs.

The user might know a person that he is looking for is a member of a department or a lab. By selecting either People or Departments from the Directory menu, the user can select to see a list of People from the details page of Lab, or Department, and browse the list or use filter in order to find the person she is interested in.

3.4.10 User Logs in

The user selects the Log in button from the home page header. The system displays a popup with a log in form. The user fills out the username and password and selects log in. The system verifies the username and password and if they are correct the user is logged in to the system. If the username and password are incorrect the system displays an error message and prompts the user to try again.

This is a use case that will be used only by persons with a University issued account.

3.4.11 The User subscribes to event types

The user logs in to System. She selects Profile from the application home menu. The system displays the profile page. The user selects Settings from the Profile page. The user fills out/ edits a subscription form, with the email he wants to receive subscriptions at, and the types of events he wants to be notified of. The user selects save preferences and system saves his preferences.

This use case is used only by persons with a University issued profile. All members of the academic community will be interested in this functionality. Jane would be interested in subscribing to events concerning rides, exams, lectures, thesis defenses and so forth, and John would want to be notified of lectures, conferences, workshops etc.

3.4.12 The User browses events

The user selects Events from application home menu. The system displays the list of all types of events. The user selects a type of events from a list and the system filters events by the type selected. The system displays the events of today ordered by time. To go to a previous or next day, the user selects the arrow buttons to the left and right of the date. To select a specific day he is interested in, the user selects date, and system displays a calendar that the user can select a day from. If the user selects a day, the system displays all events of the previously specified type on that day. To view an event the user clicks on the event title and system displays event details.

The event browsing concerns all types of users.

3.4.13 The User creates an event

The User logs in to the system. The User selects Events from the application home menu. The system displays the Events page. The User selects Create event button. The system displays a New Event form. The user fills out the event type, or leaves it undefined. She fills out the event title and description. The User selects event start date, start time, end date and end time, or defines the event as an all day event.

The User selects visibility to specify who he wants the event to be visible to. If she selects nothing the default visibility is public.

The User selects Add indoor location. The User selects Building, and then type of location, (office, lab, conference rooms and amphitheatres, or facilities). Depending on the type that the User selected, the system displays the list of locations of that type in that building and the location is added to the event. To add another indoor location, the user repeats the process.

The User selects Add outdoor location. She then selects type of location (Building, parking lot, bus stop, etc) and the system displays a list of all outdoor locations of that type. The User selects a location and the location is added to event. To add another outdoor location the User repeats the process.

The User selects the Save button from the page header. The System displays a popup window with an overview of the new event, and information about required fields that are missing. If the required fields are empty, the Save this event button is disabled and the User must return to the event form. If the event form was complete and the User does not wish to make changes, she clicks Save this event button and the event is saved. The system displays the newly created event.

If during the new event creation the User wants to stop creating the event, the User selects cancel button from header left, and the system displays the Events page.

Event creation is available only to users with University issued accounts. Secretaries would create academic events from their office computers, as would professors. Event viewing would be of high interest to all mobile users. Students would use event creation to share recreational events (lunch, game of basketball, etc) or rides (offering or requesting rides).

3.4.14 Saving data to browser cache

The User selects Cache Data Offline from the application home menu. The System displays the Cache Data page; the User selects the Download Data button. The System asks for user confirmation of action, the User confirms and she waits while data is retrieved from server and stored in browser cache.

This is a use case for all types of users. Everyone would like persistent data to be retrieved once over a wifi connection and remain available to the user whether the device has internet connectivity or not. In the case that the User has internet connectivity due to a network provider, it is costless to interact with the data straight from the browser cache, and even in the case that connectivity is provided over wireless connection (and the cost is not a problem) it is still a smoother user experience to get data straight from the browser cache.

3.4.15 Removing data from browser cache

The User selects Cache Data Offline from the application home menu. The User selects Clear Application Cache. The system displays a popup asking the user to confirm her action. The User confirms and all locally stored data is deleted from browser.

All users will at some point use this use case. They will all find use in caching data, but periodically data must be downloaded again as it grows stale. In addition, in some cases the users might have the need to keep the browser storage uncluttered.

3.5 Storyboarding

At the initial stages of the design process, we used storyboards to visualize and organize our ideas and communicate them with potential users in order to obtain feedback. A storyboard is a low fidelity prototype consisting of a series of screen sketches.

Benefits of Storyboarding can be summarized as following:

- Storyboards provide a bird’s eye view of the system.
- They demonstrate functionality of the system being designed
- They demonstrate navigation across screens
- They are easy to evaluate and change on the fly
- Brainstorming provides opportunities for the creation of new ideas, and the Storyboards facilitate their presentation within the overall functionality, regardless of time and resource constraints. It is the fastest way of illustrating all functionality potential, before selecting what will be kept.

In the images below, we illustrate how in this project ideas were initially conceived and illustrated, and how selections were made once all ideas had been demonstrated to match time and resource constraints, as well as how requirements were trimmed or added.

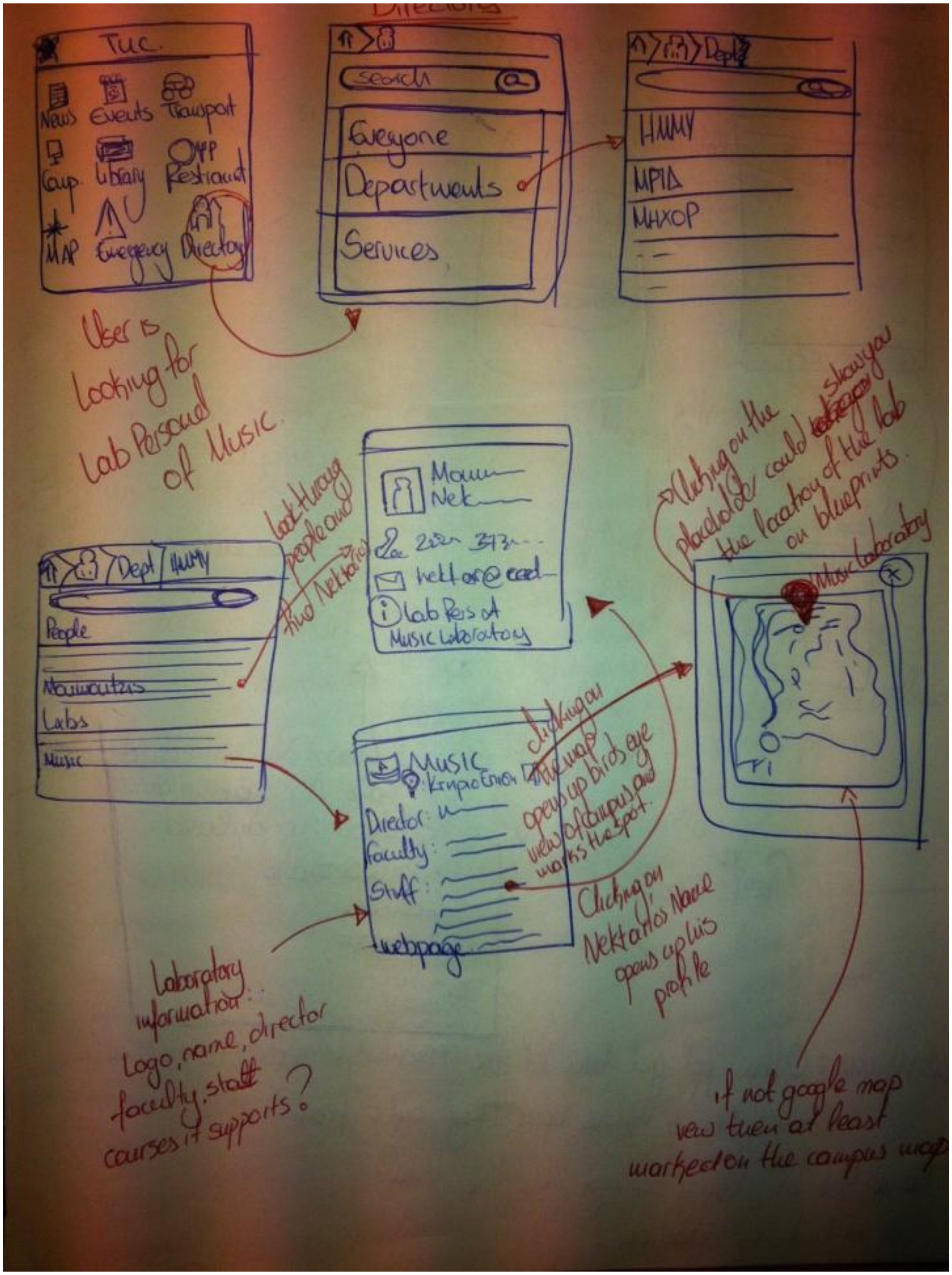

TUC
News
Events
Transport
Comp.
Library
Restaurant
MAP
Emergency
Directory
Search
Everyone
Departments
Services
Dept
HMMY
MPIΔ
MHXOP
User is looking for Lab Personel of Music
People
Labs
Music
look through people and find Nektarios
Nektarios
lab Pers at Music Laboratory
Music
Director:
Faculty:
Staff:
webpage
Clicking on the map opens up birds eye view of campus and marks the spot.
Clicking on Nektario's Name opens up his profile
Laboratory information: Logo, name, director faculty, staff courses it supports?
Clicking on the placeholder could show you the location of the lab on blueprints.
Music Laboratory
if not google map view then at least marked on the campus map

Figure 3 Initial storyboard of Directory use.

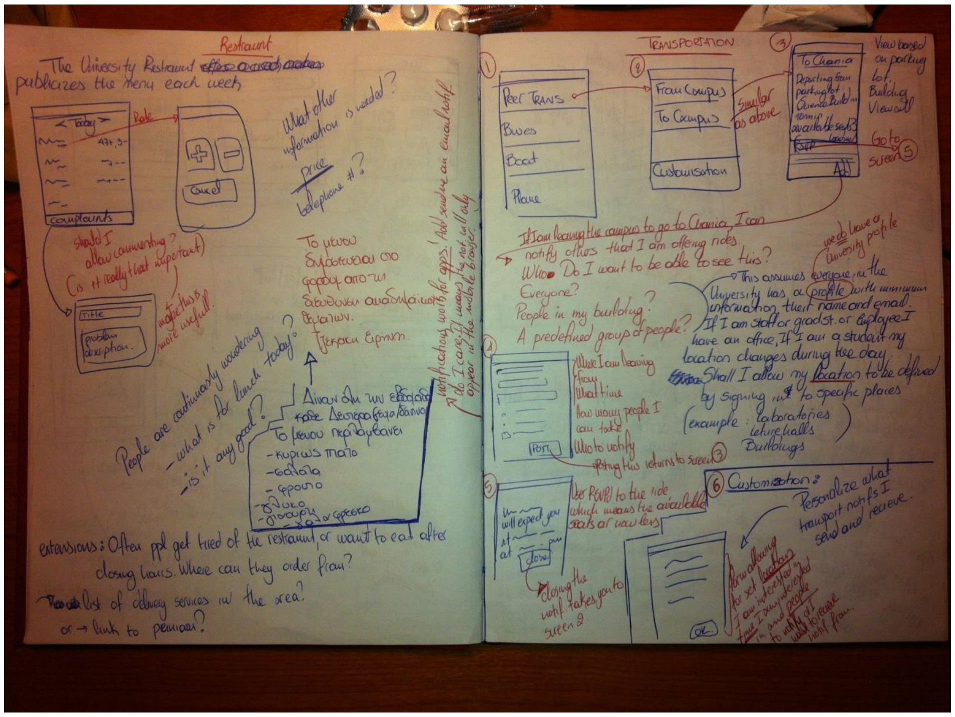

Restaurant
The University Restaurant publicizes the menu each week
TRANSPORTATION
Peer Trans
Buses
Boat
Plane
From Campus
To Campus
Customization
To Chania
Customization:
Personalize what transport notifs I send and receive.

Figure 4 Storyboards illustrating:
a) functionality that provides the weekly restaurant menu, and feedback from customers as to which dish was the most satisfactory of the day (left) b) functionality that provides users with information on transportation means, bus schedules, real time bus location, personalized notifications when users are leaving the campus and offer rides.

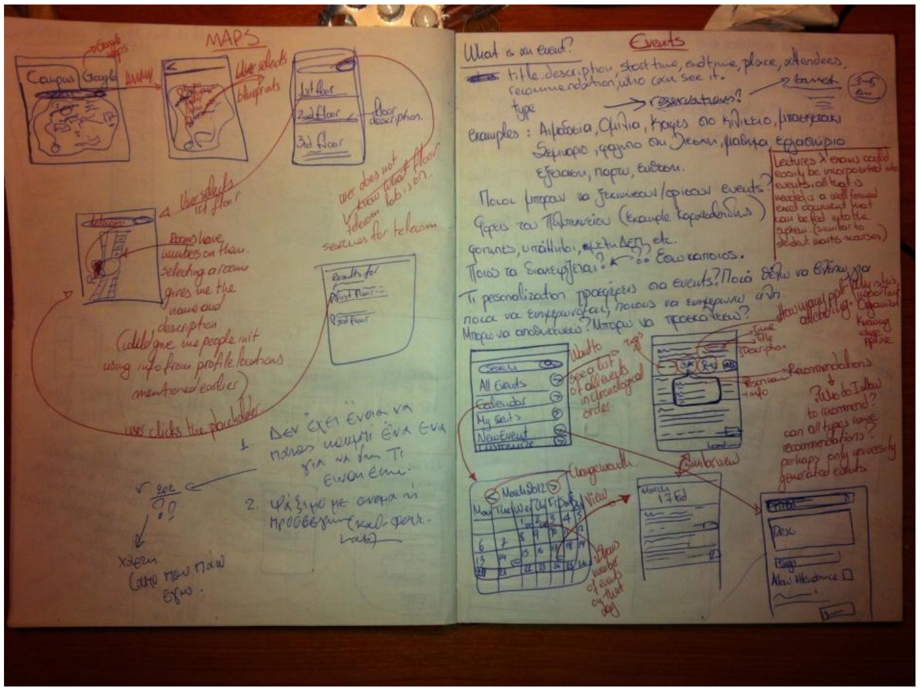


Figure 5 Illustration of maps functionality and event browsing functionality

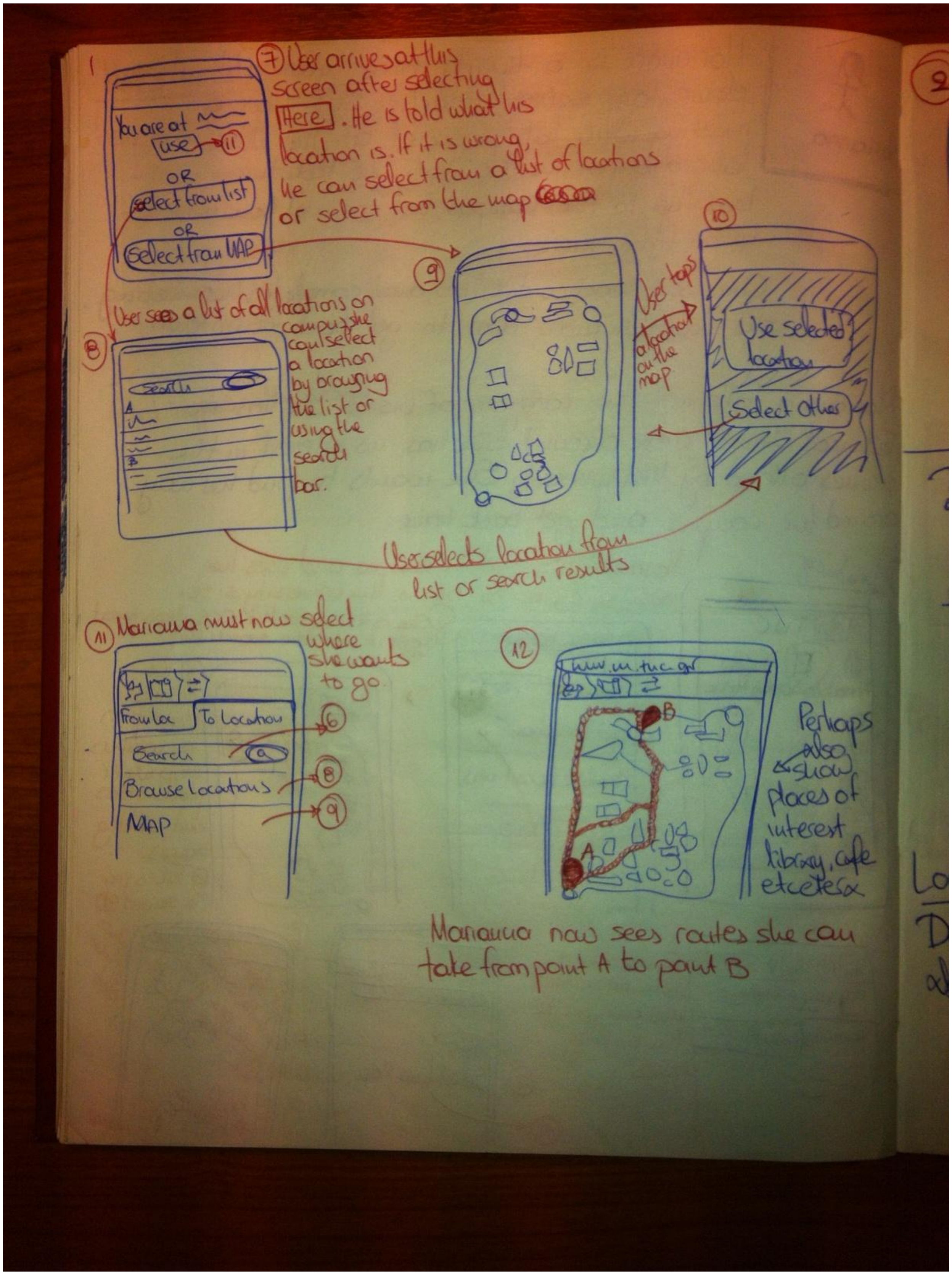

7 User arrives at this screen after selecting Here. He is told what his location is. If it is wrong, he can select from a list of locations or select from the map
You are at
Use
11
OR
select from list
OR
Select from MAP
User sees a list of all locations on campus she can select a location by browsing the list or using the search bar.
8
Search
A
B
9
User taps a location on the map
10
Use selected location
Select Other
User selects location from list or search results
11 Marianna must now select where she wants to go
From Loc
To Location
Search
Browse Locations
MAP
6
8
9
12
www.m.tuc.gr
B
A
Perhaps also show places of interest library, cafe etcetera
Marianna now sees routes she can take from point A to point B

Figure 6 Storyboards illustrating personal location sharing and campus navigation.

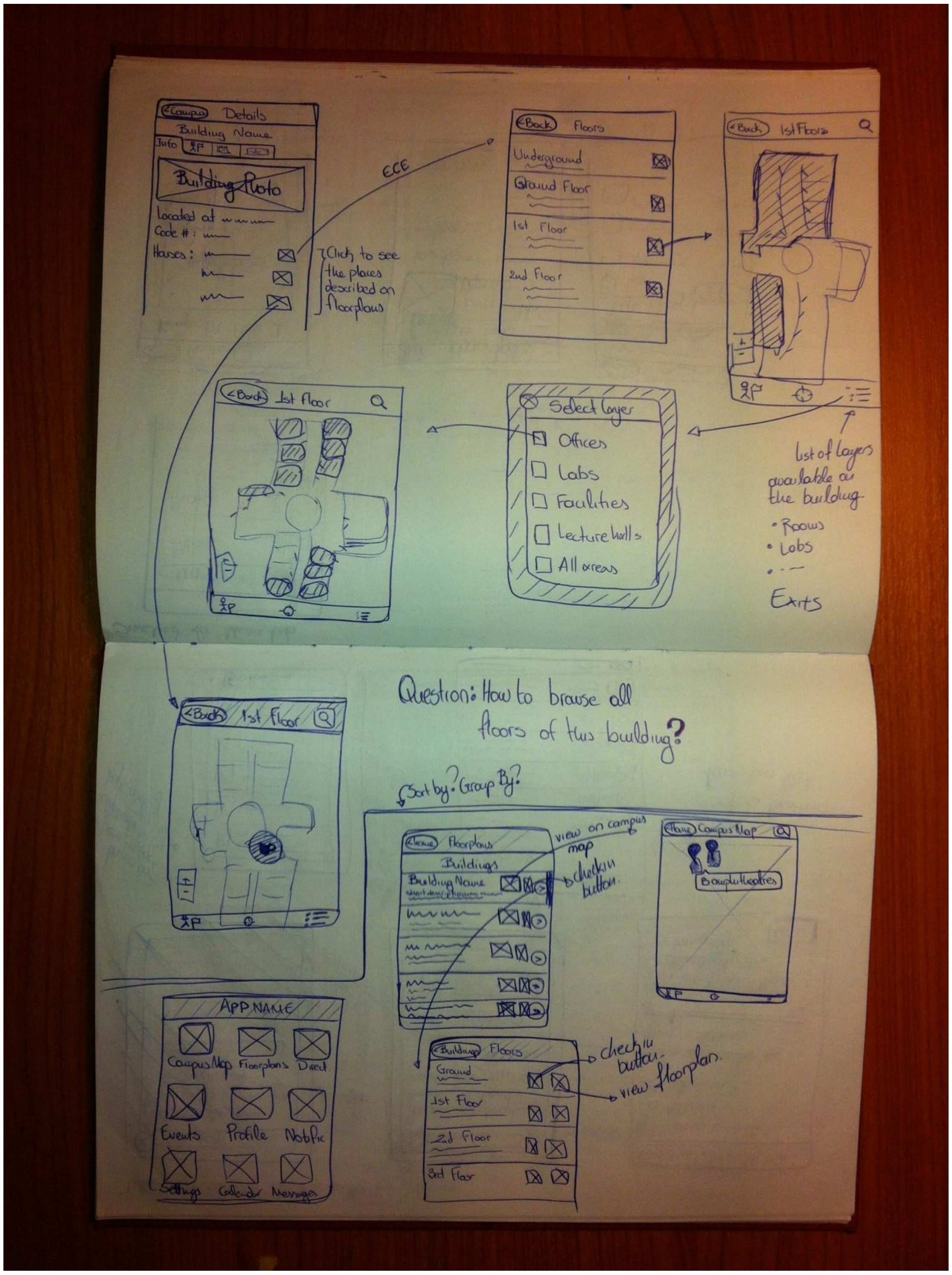


Figure 7 Storyboards illustrating building floor map browsing, navigation and personal location sharing

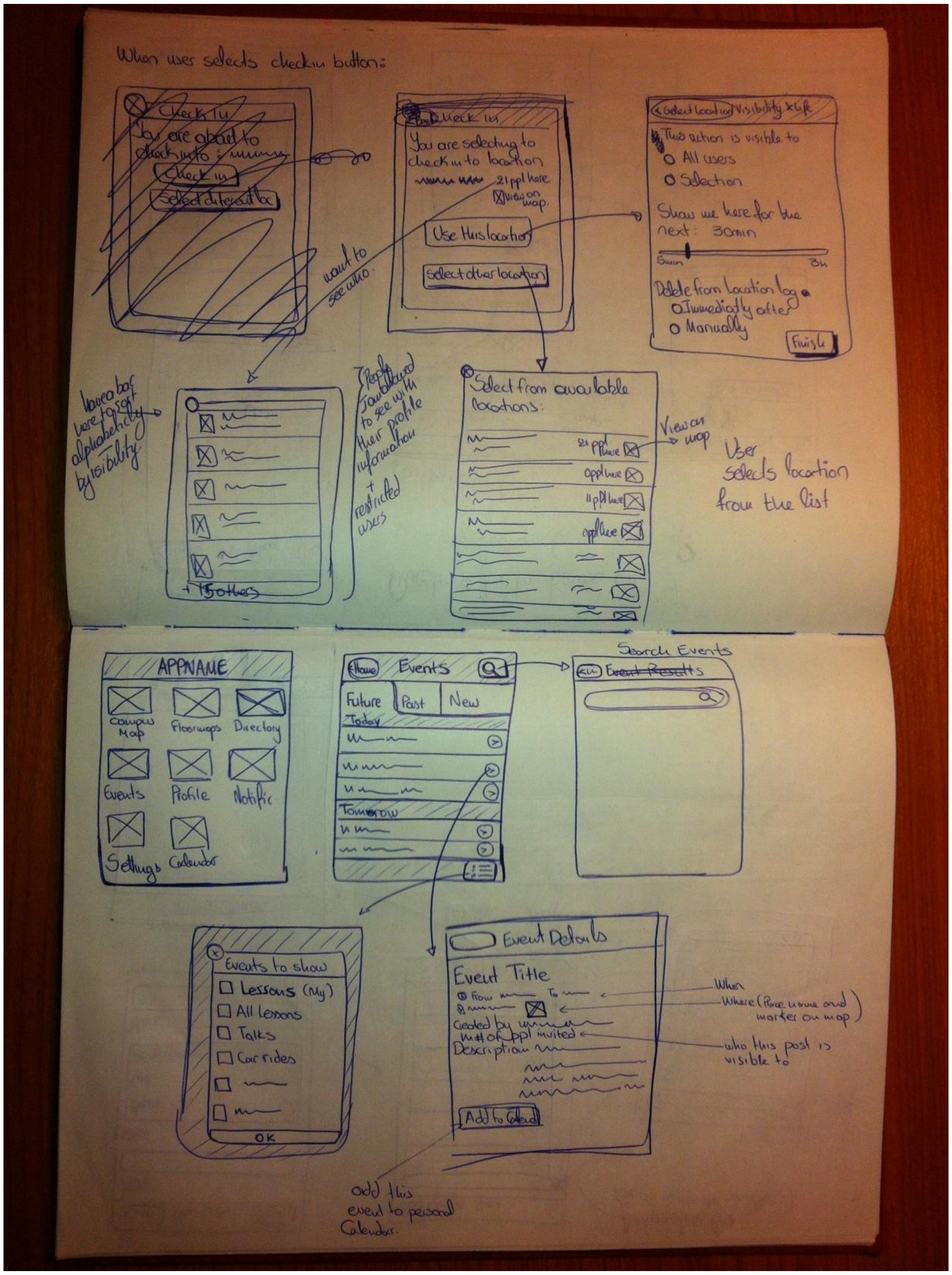


Figure 8 Storyboard illustrations of personal location sharing and event browsing

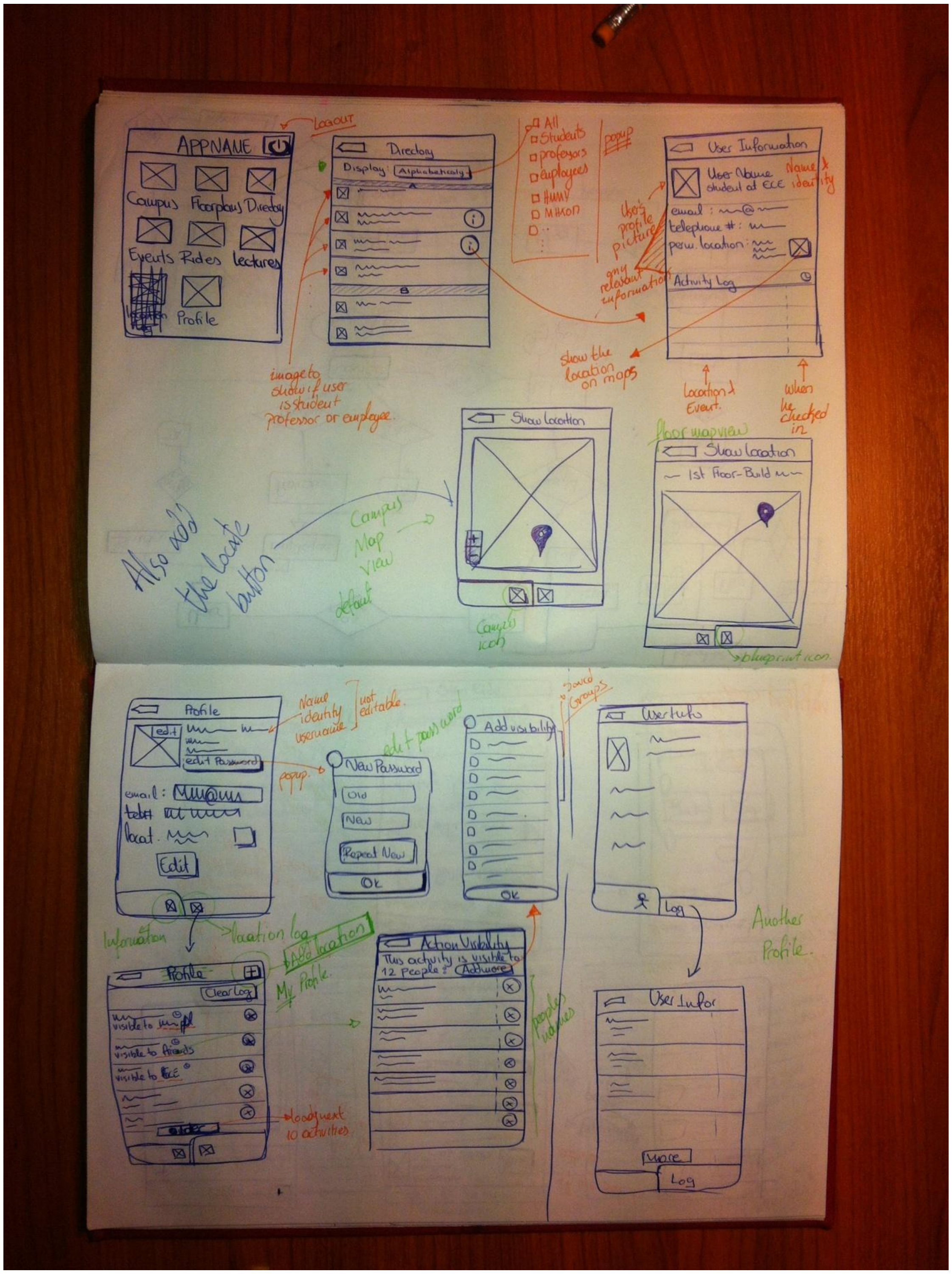


Figure 9 top: Storyboard illustrations of visualization of location on campus and on floor, bottom: profle edit, location sharing visibility personalisation

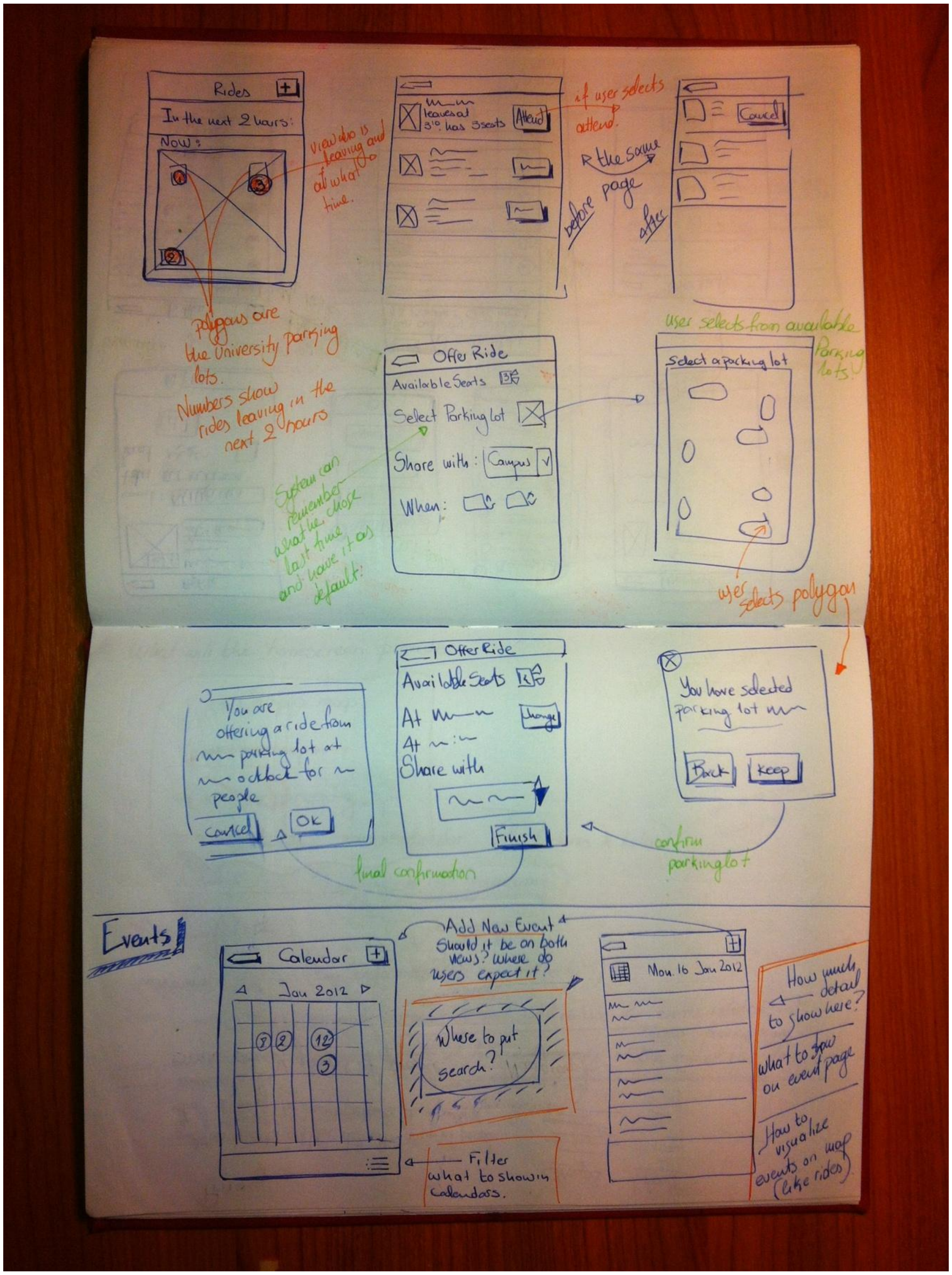

Rides
In the next 2 hours:
Now:
view who is leaving and at what time.
polygons are the University parking lots.
Numbers show rides leaving in the next 2 hours
leaves at 3:10 has 3 seats
Attend
if user selects attend.
the same page
before
after
Cancel
Offer Ride
Available Seats
Select Parking lot
Share with: Campus
When:
System can remember what he chose last time and have it as default!
user selects from available parking lots!
Select a parking lot
user selects polygon
You have selected parking lot
Back
Keep
confirm parking lot
Offer Ride
Available Seats
At
Change
At
Share with
Finish
final confirmation
You are offering a ride from parking lot at o'clock for people
Cancel
OK
Events
Calendar
Jan 2012
Add New Event
Should it be on both views? Where do users expect it?
Where to put search?
Filter what to show in calendars.
Mon. 16 Jan 2012
How much detail to show here?
what to show on event page
How to visualize events on map (like rides)

Figure 10 Storyboards illustrating visualization of rides offered (above) and offering rides (below)

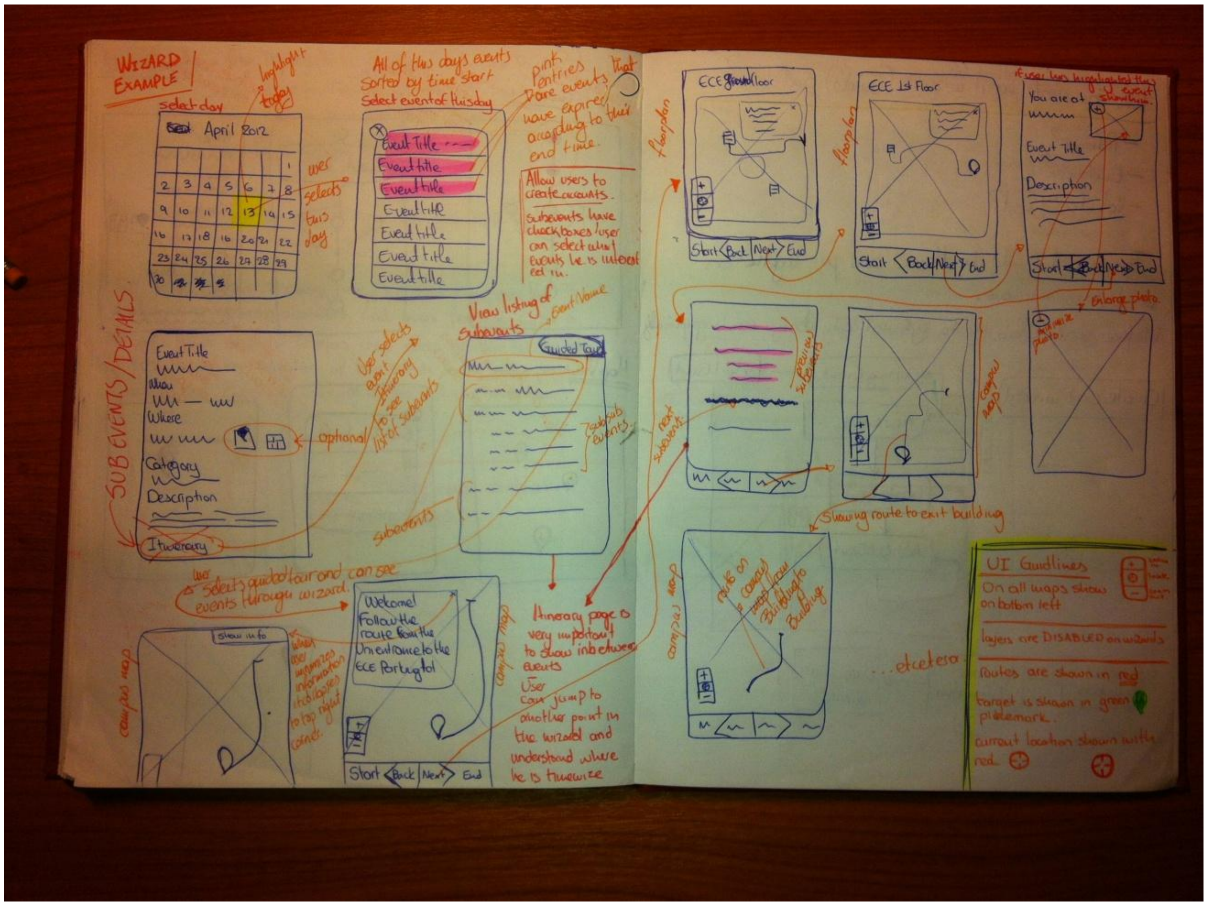

WIZARD EXAMPLE
select day
highlight today
April 2012
user selects this day
All of this days events sorted by time start
Select event of this day
pink entries are events that have expired according to their end time.
Allow users to create accounts.
subevents have checkboxes user can select what events he is interested in.
SUB EVENTS/DETAILS
Event Title
When
Where
Category
Description
Itinerary
optional
User selects event itinerary to see list of subevents
View listing of subevents
Event Name
Guided Tour
sub events
subevents
user selects guided tour and can see events through wizard.
Show info
Welcome! Follow the route from the University to the ECE Portugal
campus map
Start
Back
Next
End
Itinerary page is very important to show in between events
User can jump to another point in the wizard and understand where he is thereafter
ECE Ground floor
ECE 1st Floor
floorplan
You are at
Event Title
Description
enlarge photo
next subevent
Showing route to exit building
...etcetera
UI Guidlines
On all maps show on bottom left
layers are DISABLED on wizards
routes are shown in red
target is shown in green placemark
current location shown with red.

Figure 11 Storyboards illustrating an event wizard. Showing the user a step by step guide for complicated events, consisting of subevents at multiple locations

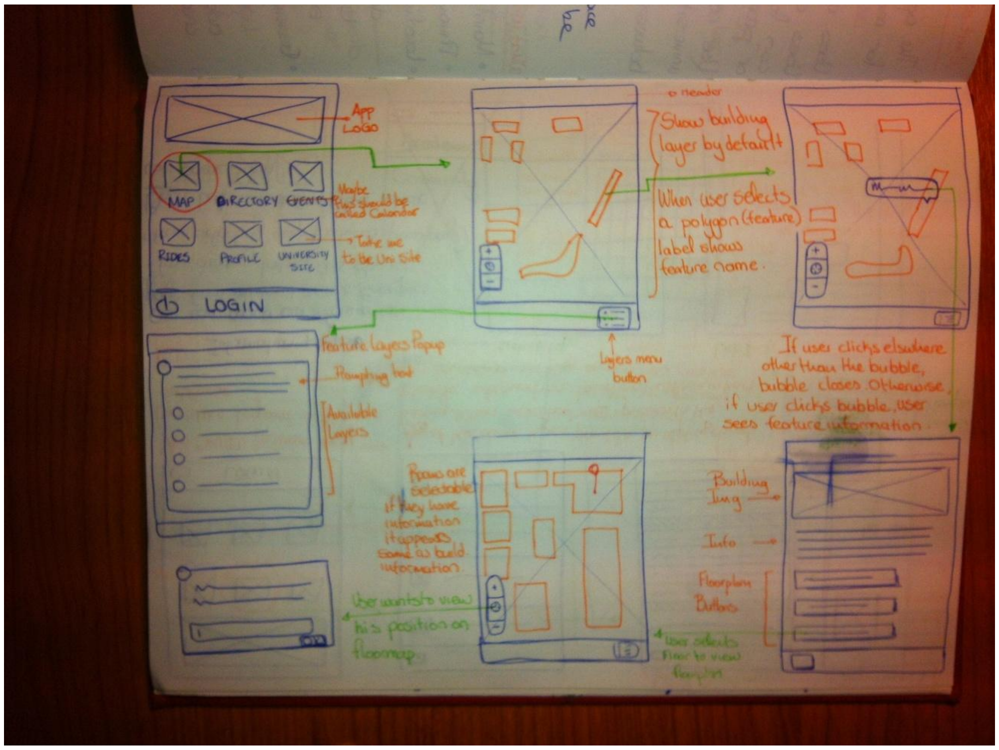

App Logo
MAP
DIRECTORY
EVENTS
RIDES
PROFILE
UNIVERSITY SITE
LOGIN
Take me to the Uni Site
Show building layer by default
When user selects a polygon (feature) label shows feature name.
Layers menu button
Feature Layers Popup
Available Layers
If user clicks elsewhere other than the bubble, bubble closes. Otherwise if user clicks bubble, user sees feature information.
Building Img
Info
Floorplan Buttons
User selects floor to view floorplan
User wants to view his position on floormap

Figure 12 Storyboards illustrating floor map browsing, and displaying user location on floor map

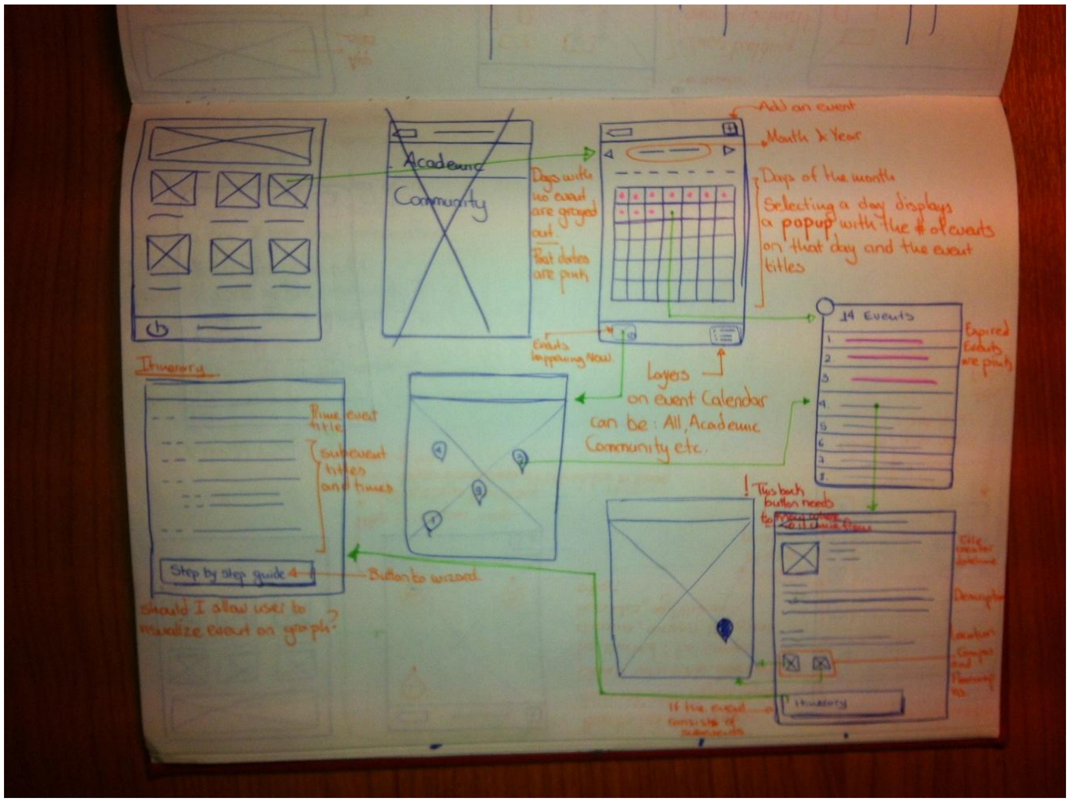

Academic
Community
Add an event
Month & Year
Days of the month
Selecting a day displays a popup with the # of events on that day and the event titles
Past dates are pink
Events happening Now
Layers on event Calendar can be: All, Academic, Community etc.
14 Events
Expired Events are pink
Itinerary
Time, event title
Sub event titles and times
Step by step guide
Button to wizard
Should I allow user to visualize event on graph?
Description
Itinerary

Figure 13 Events, event visualization, event location visualization

As illustrated in the figures above storyboards are simple screens depicting functionality, navigation and visualization. The Storyboards are a first attempt to user interface design and navigation design based on the use cases described. Once storyboards have been created, it is easy to communicate with end users and refine the user requirements, and understand better the system and technology requirements for the implementation. Even though much of the functionality is fixed from the initial development process stages, some ideas are still considered for future implementation and they are taken into consideration during architectural and database design.

When the Storyboards of this project were shown to a panel of development experts and University users, the functionality was trimmed to match time constraints, taking into consideration what was the most useful for the community. For example, event wizards and complicated events were considered and they were eliminated in favor of standard events. Personal Location sharing was also considered a lesser priority, as it targeted a smaller group of people in the community (mostly students). Restaurant menu and food marking were also considered low priority. These may be added in future versions.

The Campus Map, the Floor Maps and the Directory infrastructure were given the highest priority, and it was agreed upon that the functionality provided should also work when the application is offline, as the data is semi persistent and is not updated very frequently. These functionalities are useful to both the University community and visitors to the community. Also, many of the other functionalities described can be added on top of the directory and location information.

3.6 Paper Prototyping

Once Storyboards had been evaluated and prioritized, the second design phase involved Paper Prototyping. Paper prototyping again utilizes simple office supplies, like pen and paper. The designer uses paper cutouts the size of the screen she is targeting, and draws all elements of the user interface on the screens, (menus, dialogs, error messages) creating all screens for each use case that the system will support.

Paper prototyping is used to design and test user interfaces. They are an extremely important part of the design process, and saves valuable time down the road.

Benefits:

- Prototypes have the advantage that they are good in conveying the interface to a user.
- They allow usability testing early in the development process.
- They are extremely easy to modify and experiment with, and can be produced and refined in real time during system evaluation to try a new idea.
- They are essential to finding serious errors in the interface, and save time when the interface requires complete redesign.
- During usability testing the designer discovers user priorities in real time.
- Evaluation is relatively fast and the users are not afraid of criticizing work that looks like doodling.

In all, a total of approximately 60 paper screens were created, before the user interface design was completed. The result was approximately 35 paper prototypes of mobile phone screens with the user interface of the system. Below are examples of the prototypes created.

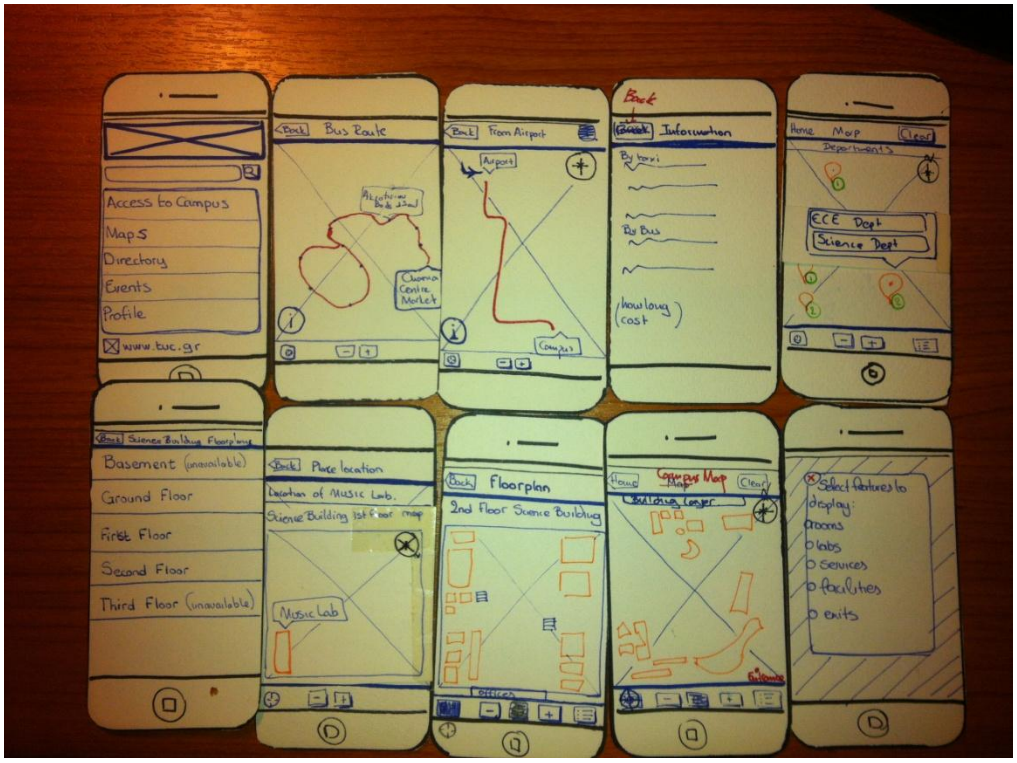


Figure 14 Paper prototypes of screens for mapping functionality

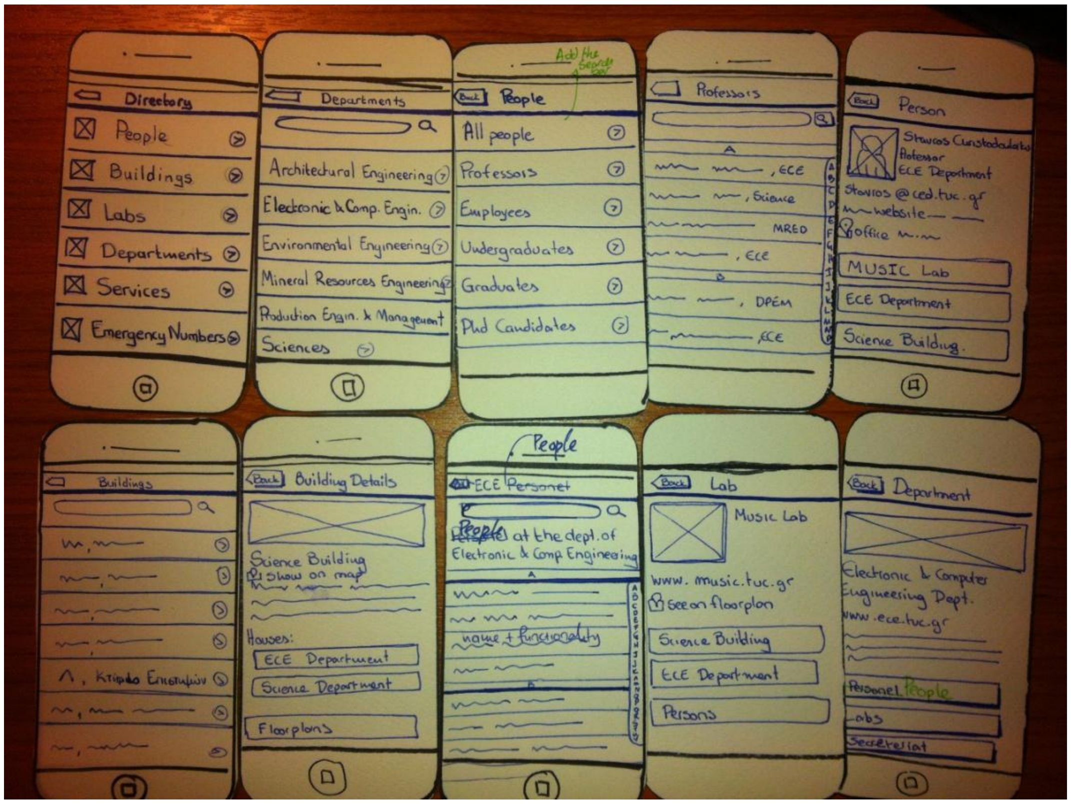


Figure 15 Paper prototypes of screens for directory functionality

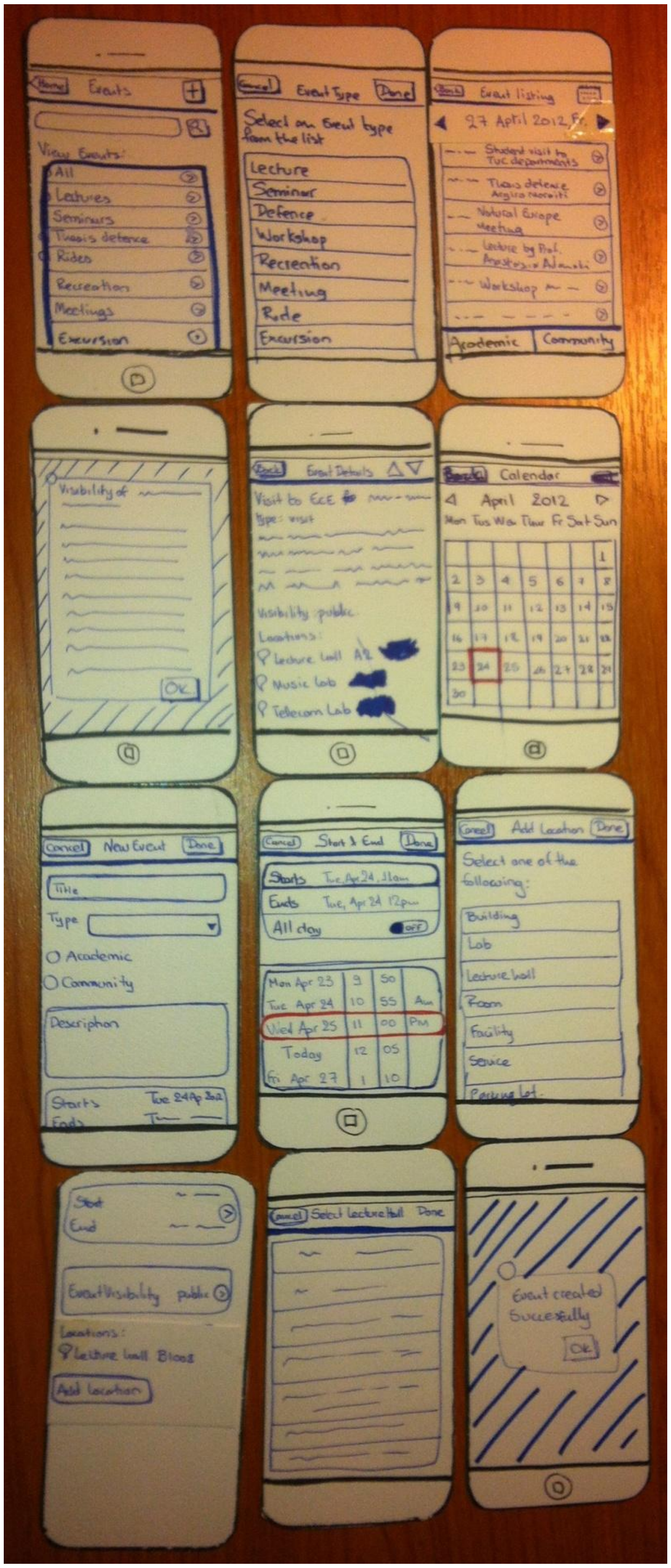


Figure 16 Paper prototypes of screens for event functionality

3.7 Evaluation and Refinements

Constant Evaluations are a critical part of Software Development. During and after each stage of design and development, our application underwent a series of evaluations, some with usability experts, and some with users selected from the community the application is being built for (as seen in the persona description above).

Evaluation of the paper prototypes was conducted with three Users.

The first evaluation was a Cognitive Walkthrough conducted with a user interface expert and it produced the most valuable feedback on the user interfaces. The second and third evaluations were conducted with students from the University, (one graduate student and one first year student), and provided mostly feedback on extra functionality that they would be interested in. The evaluation with the university students followed the think aloud methodology. The purpose of the system was described to the users. Then specific tasks were given to them to be carried out. They were not helped during the execution of the tasks, but they were encouraged to talk what was in their mind. During the process of think aloud I was keeping notes about their thoughts and the problems encountered.

All three users considered a home icon to be of prime importance on all screens. All users agreed that they wanted all available features displayed on maps when loaded, and the ability to remove features from maps at will. None of the Users thought a clear map function was of any use, they said it only cluttered the screen. Users requested that a label be added to maps when location was being shown so they could remember what they had asked to view location for. All users agreed it was important to show a compass on maps. All users wanted map information to increase on different zoom levels. User 3 requested that all profile information be editable on profile page in one form, instead of individual forms.

3.8 Summary

In this Chapter we described the requirement analysis and the prototyping phases of the project. Initially we investigated the potential users of the application system that we had in mind. We considered several possible user candidates, including professors, students, new students, visiting professors, and visitors from the city or from other places, visitors that are coming to attend events. We found that they had common requirements, and that the requirements could be described with a rather small number of Personas. We gave the characteristics of those Personas and we used them in the analysis of use cases and the user interface prototyping. Then we described the use cases of the system and we created Storyboards to outline how this functionality was going to be presented in the user interface, as well as to present the navigation across screens. The Storyboards are informal diagrams of user interfaces that can be used for clarifying how the application system will be used by the end users. The Storyboards were used for presenting the ideas of the system to wider audiences (HCI class) and getting feedback about possible changes or extensions required. Then the most important functionalities were selected (based on engineering criteria of usefulness and time to project completion) and the interfaces for those functionalities were paper prototyped. The paper prototypes were very easy to create and modify when changes were decided, a truly useful tool for requirements capturing and interface design of large applications. The paper prototypes were evaluated with both heuristic evaluation and think aloud methodologies, and the results were used to refine the interface and navigation design of the system.

Chapter 4 - Mobile Web System Architecture and Implementation Technologies

4.1 Introduction

In this Chapter we describe the architecture of the Web Server and the Web Client of our application. Some parts of the application, those related to the event management, due to the high expected update rate, are used in a traditional on-line architectural style, where all the page preparation takes place in the Web Server, and the Web Client is a thin client. The functionalities of the application that relate to the management of maps, diagrams, places and structure of the university personnel are structured as a rich application that my work in an on-line or off-line fashion. The Web Server offers a REST Service to the outside world that the clients can use it to offer rich functionality. In our implementation, the REST Service serves http requests that offer the desired functionality and deliver JSON objects to the client. The Web Client of our application is structured in a Model View Presenter architectural style, and the components support replacement of parts of the presentation page, dynamic graphics visualizations, and work in on-line and offline fashion. We describe the architecture that supports these functionalities in the Web Server and the Web Client, as well as the functionality of the components, and we highlight the control logic during execution. Finally we describe the technologies and tools used in the implementation.

4.2 Rich Mobile Architectural Styles: Online Only, Offline Only, Online Offline

There are three alternative architectural styles for mobile web application development.

The Online Only Style implements the mobile web applications as usual web applications with thin clients. All the data and the programs reside on the server side. The client is only responsible for presentation of the prepared web pages, and submission of the user interactions to the Server. This architecture is appropriate when the client is not a powerful device (and therefore it is desirable that all processing is going on in the server. It is also appropriate when the application is heavily update oriented, in which case the data values in the server change continuously and it makes no sense to cache them in the client. It also makes sense when plenty of space exists for caching in the client, and the applications are not competing against each other for space. Finally it makes sense when there is continuous internet connection, and especially free internet connection.

There are a few problems with the Online Only Architectural Style. First of all they cannot work when there is no internet connection, and they may be costly if there is no free internet connection. This may be important in several application environments. Second, mobile internet access may be of limited bandwidth. If the applications continuously goes back to the Server to carry data, especially Rich Media data (images, graphics, animations, etc.), this may be a very slow process. The application will be much slower than if the data were cached locally in the client, and the server will be much more loaded.

The Off Line Only Architectural Style downloads all the data and the programs in the client, and then it proceeds independently on the server. At the end of processing or periodically it synchronizes with the server and possibly carries its updates or the new data to the Server. This Architectural Style is desirable when there are no updates to the data, when the data is big objects, or when the updates do not affect other users (for example archival data gathering). This Architectural Style is problematic when the data values must be updated frequently and the newest values should be seen by the users. In addition, since the page preparation takes place in the client, the client should have enough processing power. Finally the available memory in the client may be a problem, since the applications of the client will compete for storage space.

The On-line Offline Architectural Style supports both; it caches some static data to the client in order for the application to be able to work to a large extent off-line and with good response times for interactions with Rich Media Objects; it also behaves like thin client with the highly volatile data that have to be current, requesting the prepared pages from the Server.

Our application has two types of data: maps and floor diagrams are large graphic objects that are static. In addition some parts of the data base including the directory data are static or very rarely changing. This part of the data could be cached to the client. On the other hand, events are dynamic in nature, we anticipate that there will be frequently new events generated, and the interest is usually in the newest events. This type of data is better managed by a Thin Client Architectural Style. Thus we have decided to opt for an On-Line Offline Architectural Style.

4.2.1 Support for Online Data Access only

The application we developed supports the generation of dynamic events. Events are typically created very often, and are of interest in real time. Event viewing and creation happens when the User has internet connectivity. In this case, html pages can be created on the Server before being sent to the Client to render in the browser. Event related pages are not available offline, therefore need not be cached. They are generated on the server, populating dynamic parts of the page with Java Object methods.

The **Thin Client Mobile Web Architecture** that we created follows the Web Development Patterns that we described in the State of the Art section. It follows the **BCED** (Boundary, Control, Entity, and Data Base Interface) Pattern.

The **Data Base Interface Layer** is solely responsible to map Entity Layer objects to **persistent relational data base objects** and vice versa. It uses the **DAO** (Data Access Object) pattern to do this. It also uses the **Transfer Object Pattern** to transfer collections of objects to higher levels of the Architecture. The Data Base that supports the application is a relational database, and it communicates with the application with the database interface layer. Thus the Entity layer is independent from the data base implementation.

The **Entity Layer** (or Domain Layer) contains the objects that have data with archival life beyond execution of the application. The implementation of the entity layer was done with Java Bean technology.

The **Control Layer** is based on the **Application Controller Pattern**, providing a single point of entrance in the Server. The **Application Controller** is responsible for the flow of control and the execution of the business logic of the application. As a single point of entrance it is also responsible for the authorization functionality of the application. The Control Layer essentially supports access to the business functionality of the application. This functionality is available through the Application Façade of the application. The technologies used to implement the Control Layer include Servlets and Java Beans.

The **Boundary Layer** is responsible for the interface of the application. User input is managed through the Event Pattern. Events are sent to the Application Controller for further processing. The Boundary Layer is also responsible for preparing the presentation of the pages in the user interface. The technologies used for the implementation of the Boundary layer are **HTML, JavaScript, Java Server Pages (JSP's), Java, XML/XSL and JSON**. JSPs are in fact documents like HTML, and they are constructed much like HTML documents, often using the same HTML editor. JSPs are translated into Servlets by the Web Container. The Servlets return the resulting HTML documents for presentation to the Browser. Thus JSPs exist more for programming reasons (it is difficult to create HTML documents programming in Java). In this approach, the whole process of creating the View that the interface will present resides in the Server, and it requires a thin Client that will only need to present HTML pages.

The **Model View Controller** pattern is supported by this Architecture and implementation. The Model is fulfilled by the Entity Layer, The Controller from the Controller Layer which in effect is implemented by the Servlets in the Server; the View corresponds to the Boundary Layer and is implemented by the JSPs in the Server. The Controller is responsible for accessing the information from the Model (Entity Layer) and to decide which Views (JSPs) will be created, and send the Model data needed for their creation. The important point is that the Model does not communicate with the Controller or the View, thus the application can accommodate new clients without changing its code.

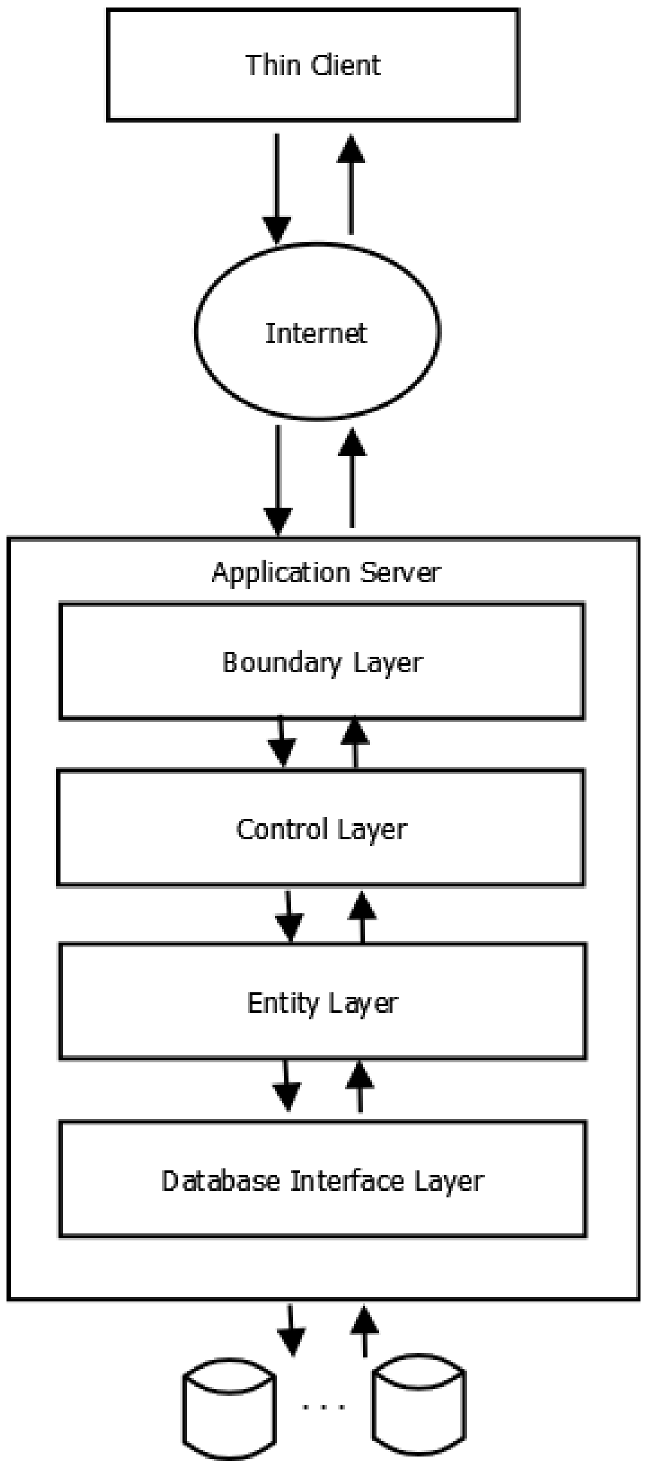


Figure 17 Architecture for server dependent functionality. Requests are mapped by the controller to Servlets, which use Java Objects from the Entity Layer to store and retrieve data from the database, via the Database Interface Layer. The Servlets are in charge of Specifying which page in the Boundary Layer is to be populated dynamically with the information retrieved

4.2.2 Rich Client Interaction Architecture and Ajax

In the traditional Web Architecture shown above, the client is a thin client that does minimal processing of pages (forms). This is not a very interactive style of communication. The **Rich Internet Applications** allow the clients to issue requests that do not replace the whole page but part of the page. These requests can be sent asynchronously to the Server, and when the response comes it is integrated with the rest information and presented in the user screen. The technologies that support the **Rich Client interactions** and the asynchronous replacement of parts of the pages are known as **Ajax Technologies**.

To support Rich Internet Interactions there is a need to manipulate the structure of the **Document Object Model** (DOM tree, see State of the Art) of the page to be presented. In our architecture this is done in the client. Thus the client acquires logic (written in JavaScript) to manipulate the DOM tree as well. The new data that will change a part of the screen presentation are going to replace the contents of an element in the DOM tree. The data may come from the server as JavaScript Object Notation (JSON) data and they will be used by the script logic in the DOM to format the replacement of a certain element of the current page. They may also come as ready to be presented elements, formatted in HTML (widgets). Widgets have the advantage that they are formatted data and they do not have the overhead of processing for preparation. In addition, if they are used in several places in the interface, their format is centrally controlled.

The architecture of the Server remains the same, but now the Client and the Server communicate in addition to the usual way that retrieves pages with an asynchronous HTTP request manner that transfers requests for data or presentations for parts of the page. As result of the Asynchronous http request, the Server may prepare and return JSON objects that carry the data that will be used to prepare an element.

The Rich Client Architectural Style puts emphasis in the user interface and the user interactions with the interface. The Architectural Pattern of **Composite View** which separates the form of presentation from the processes that are used to populate the various fields is important. A View is composed of sub-views. HTML skeletons are used to describe the structure of the pages that will be populated.

The technologies that were used to implement the Rich Internet Interactions in our application were JQuery, Ajax, JavaScript, JSON, HTML and DOM.

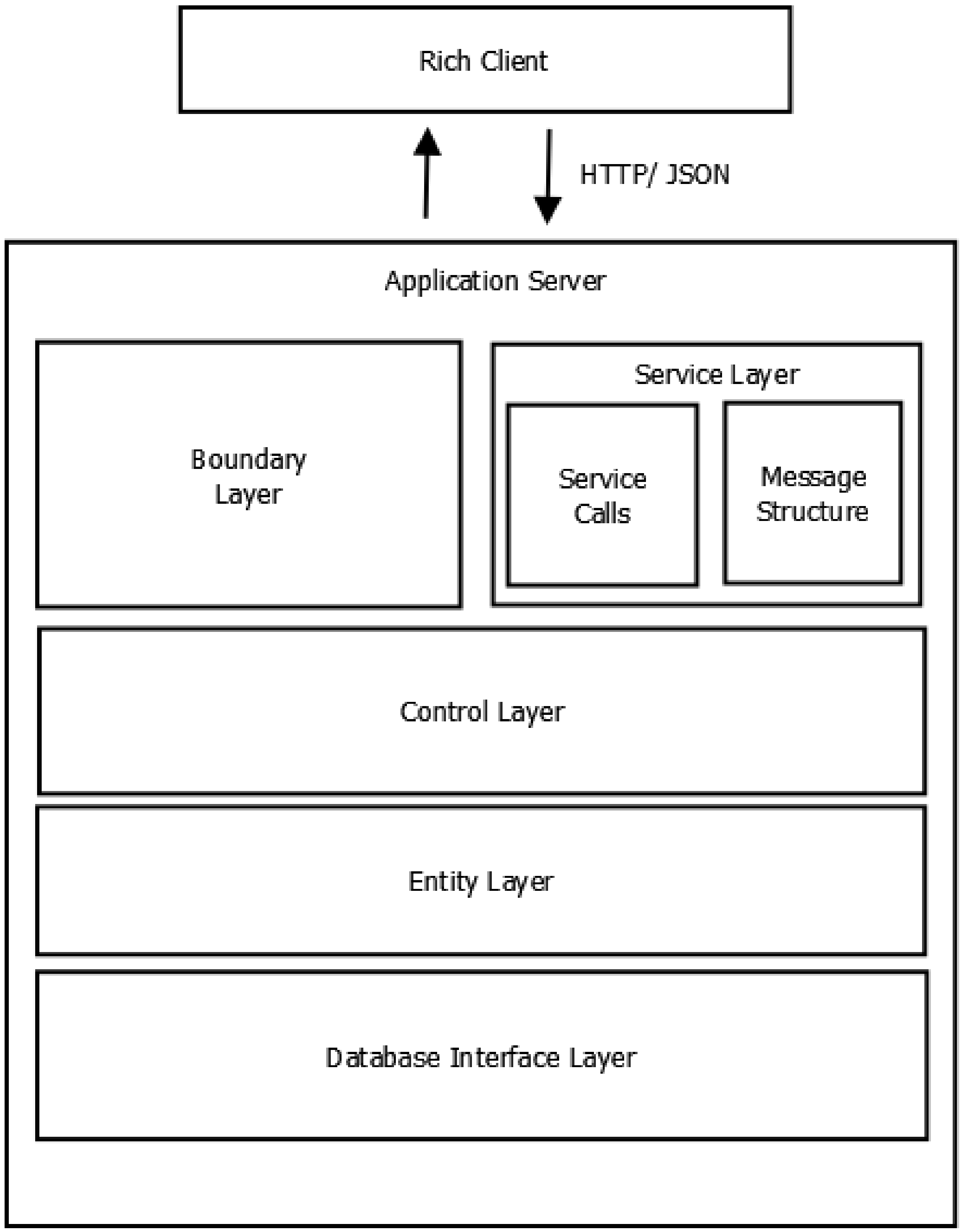


Figure 18 - Server side Architecture

4.2.3 The REST Service Layer in the Server

To support Rich Internet Applications in the Client, the Client sends HTTP request to the Server and gets back Model information structured in JSON format. This information is to be manipulated in the client and inserted in the DOM tree for presentation. This is in contrast with the traditional Web applications where the server was delivering to the thin client formatted HTML pages. The new functionality that the Server offers is **Service functionality**. The Server exposes an interface (HTTP calls) that can be used by applications that want to use the functionality that the Server offers. In addition the server is encoding the responses in structured JSON objects that have a specified format. These objects can be decomposed and used in other applications that use the basic functionality offered by the Server.

The Server in our system does not maintain Session State between HTTP calls. Thus the Service provided is a **RESTfull** type of Service.

In our system, the functionality of the maps and directories is exposed through the Service Layer of the architecture since future applications may need the map capabilities. The functionality of events is offered through a normal web interface.

4.2.4 Support for Offline with Appcache Manifest and Web Storage

An important requirement of our mobile application is to offer part of its functionality offline (the more static parts). As a consequence, the application must work both with server calls, and independently of server calls. To achieve this, on the server side we have implemented the **BCED** (Boundary Control Entity Database Interface) pattern. Data is summoned from the database and passes through Database Interface, Entity and Control layers to reach either the Boundary Layer, where it is transformed into html, and the html is sent to the client to render in the browser, or data is retrieved by the client from the Service Layer with AJAX calls in JSON format and then parsed to retrieve the data relevant, and inserted into the html document. To support the off-line operation we use in the architecture the facilities provided by the developing HTML5 standard. In particular we utilize the Manifest File and the Local Storage facilities of the HTML5.

The **Manifest File** specifies in the **Cache** section of the file a list of resources which are downloaded the first time that the Client will access the Server, and after that they remain resident in the Client. The list of resources is a list of URLs from the Server. The URLs point within the directory of the Server to resources such as HTML skeletons for presentation of interface pages in a certain form, JavaScript files, and CSS files. These files are downloaded to the Client so that they allow the creation of presentations of information in JSON format which may be coming to the Client in real time (from the Server or from the Local Storage).

Network is a section of the Manifest file that specifies URLs of resources that the Client has to have access to internet in order to access them. The **Fallback** section of the Manifest File specifies alternative URLs to be shown if at some point in time the Client does not have internet access.

We store in the Manifest File the resources related to maps and directories since we want this part of the application to work even without network access.

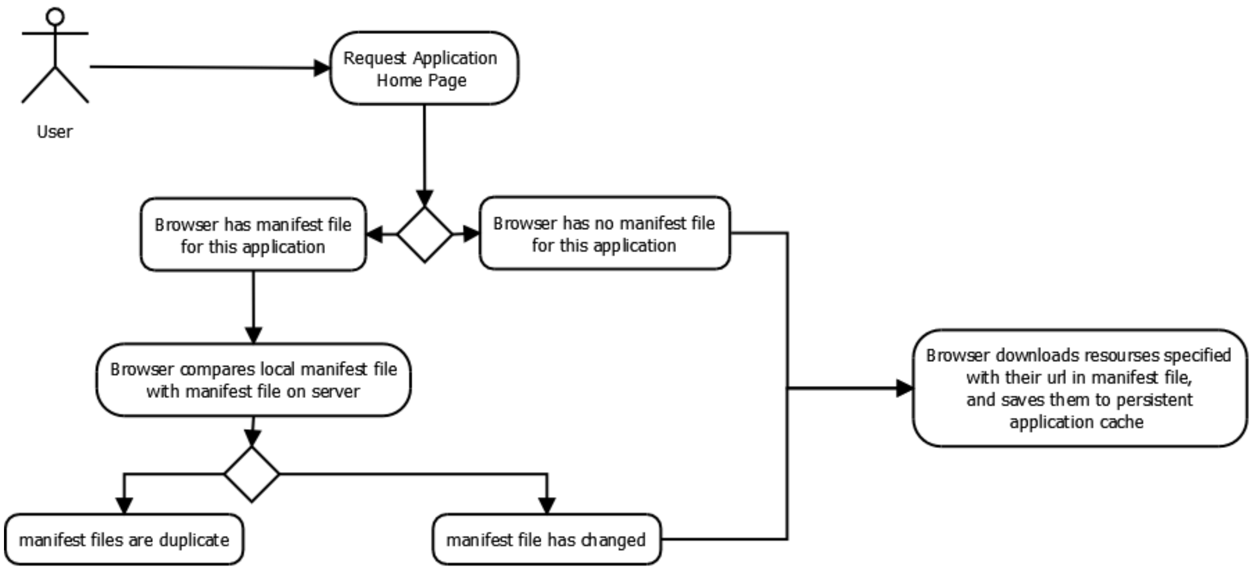

User
Request Application Home Page
Browser has manifest file for this application
Browser has no manifest file for this application
Browser compares local manifest file with manifest file on server
Browser downloads resourses specified with their url in manifest file, and saves them to persistent application cache
manifest files are duplicate
manifest file has changed

Figure 19 Caching resources with the manifest file

The data that will populate the **Manifest File HTML skeletons** should be resident in the client in order to be able to show them off line. In our system the user can choose when to download this data to the Client (typically when she has inexpensive internet access). This data for our application is the data that is needed for the functionality of maps and directories. The data are downloaded in JSON format and are stored using Local Storage (persistent storage of the Web Storage protocol) of the Client. The Local Storage of the Client has size limitations (5 Mbytes). Thus larger applications will have to invent more intelligent ways or personalized ways for managing the Persistent Local Storage.

The APIs for accessing the Manifest File and the Web Storage have been standardized by HTML5.

4.2.5 The Client Architecture

The mobile Client is based on a **Model-View-Presenter** pattern.

The **View** is simply responsible for rendering the HTML presentations, and creating events for the interactions of the user with the interface (touch and click interactions).

The **Presenter** integrates the functionality of **Application Controller** and a **Presentation Component** which is responsible for the creation of the views to be presented. The Application Controller takes actions for the events in the View. Depending on what the event was, it may modify the Model, request new data for presentation from the Server or the **Local Storage**. It may also save some data in the **Session Storage**, issue a request to the Server for a new skeleton page, and send the retrieved information to the Presentation Component to format the new page. The Presentation Component is responsible for the formation of Views. It uses JavaScript to manipulate the DOM tree and insert or replace information in it. Elements in the DOM tree are specified by the HTML tags, and they are identified by unique ids. Thus each part of the presentation on the screen is identified by a unique id. To fill this space the Application Controller will first search in the Persistent Local Storage, and if the data object it needs is not there, the Application Controller issues an Ajax request to the server and gets back a JSON object which is structured information that is part of the Model. The JSON object is associated with the id of the HTML element, and is sent to the Presentation Component to be used to complete the information in the DOM tree.

In some cases the information requested by the Client is ready pages (for the parts of the application that work on-line, like the events). In this case the pages are prepared by the Boundary Layer of the Server and are sent to the Client. Thus the Application Controller accepts information from the Server either in JSON form or in JSP form. The Application Controller sends to the Server requests in the form of HTTP requests or updates.

The Application Controller also has to communicate and extract information from the **device sensors**. In our project this is important since the application is heavily map oriented, and the use of the **GPS sensors** of the mobile devices is needed so that the location of the device can be positioned on the map. The Application Controller communicates with the GPS device using the HTML 5-standard **Geolocation API.**

The technologies used in the Application Controller and the Presentation Component are JQuery, JavaScript, DOM, HTML.

The Model Component of the Client is composed of structured JSON objects. These objects may be main memory resident or they may be stored in the Persistent part of Web Storage, Local Storage, so that it also works offline. The **Local Storage** stores the objects as **key value pairs**. The key part is used to store a key for the item and the value part is used to store the structured JSON string. The objects that are stored in the JSON strings roughly correspond to the objects stored in the Entity Layer of the Server. The API for communicating with Web Storage is standardized by HTML5.

The Client does not only communicate with the Server of our application. It also communicates with the Server of the OpenStreetMap server in order to get map information outside the TUC campus. This information is presented as the user browses the map away from the University (following for example the routes to access the University). The library **Open Layers** has been installed in our Server and is downloaded to the Client persistent cache with the Manifest file. When the Client operates offline an **SVG** map of the TUC campus is used and layers of information are superimposed on the map. When the user requests map information a request is issued by the Client to the OpenStreetMap server for map information. The OpenStreetMap server responds returning the tiles of the map requested, centering the map at a given GPS point and zooming in to a given zoom. These tiles are placed as images in the DOM of the current page and can be used for superimposing other layers above.

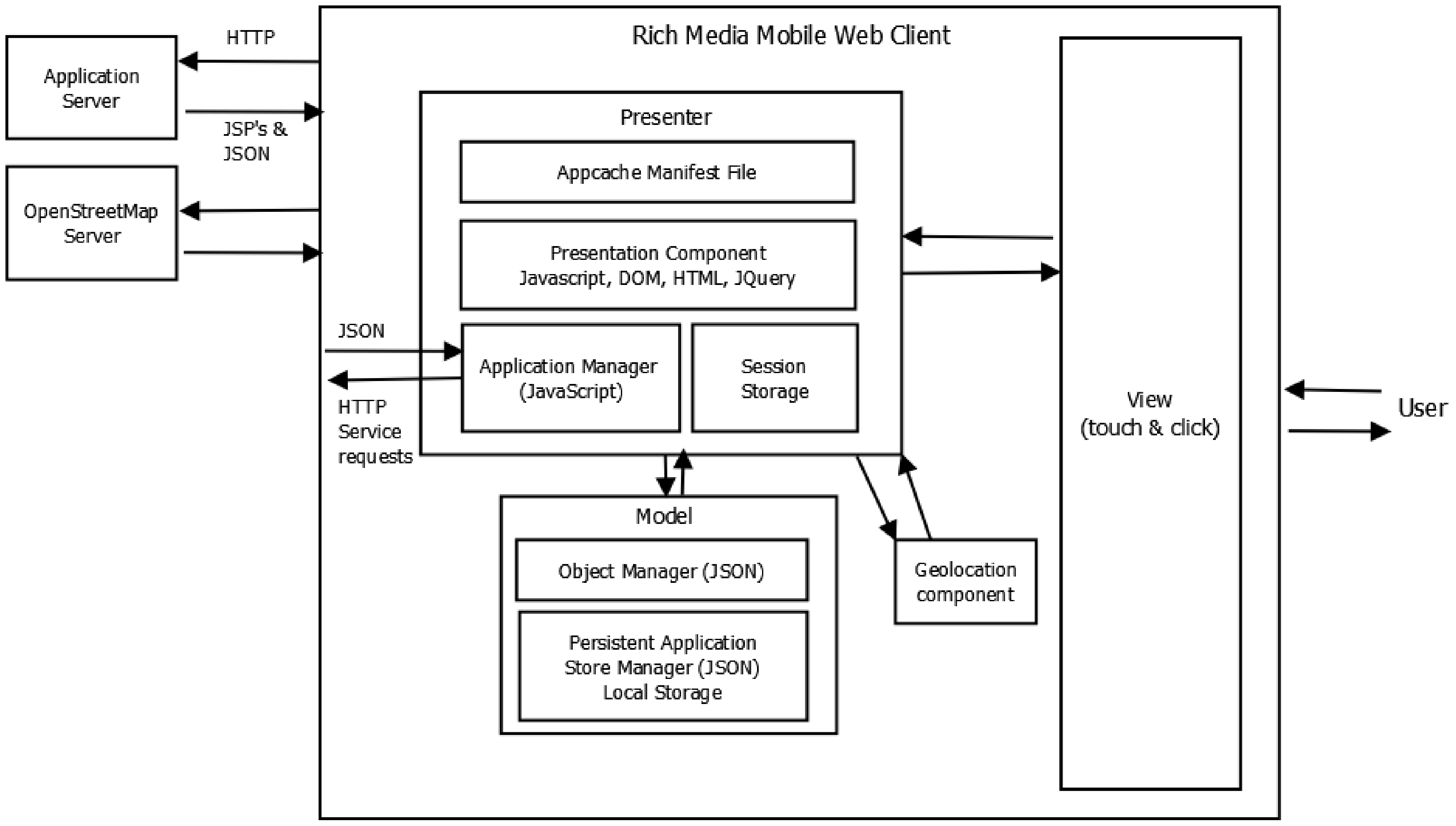


Figure 20 Rich Media Mobile Web Client Architecture

For example, when the user first accesses the application, she does so with an internet connection. Resources described in the manifest file are saved to persistent storage of the browser (Appcache). When User navigates to Directory, and selects Departments, the Web Browser searches the Manifest File and finds the HTML page skeleton for the Departments and it displays the page. The Browser also retrieves from the Manifest File the JavaScript files that are linked with this page and initiates their execution. The JavaScript in charge of filling the Departments list tries to retrieve the object describing all departments from Local Storage. The data has not been downloaded there yet, so the retrieval fails, and as a fallback the JavaScript makes an Ajax call to the server, requesting a JSON object of the same structure but describing only all department names and their associated ids. The Application Manager receives the JSON object and stores it in a local variable as part of the Model. Then it sends it to the Presentation Manager where it is parsed to extract its components and the DOM of the page is modified to include the department listing. The HTML with the department listing is sent to the View for presentation.

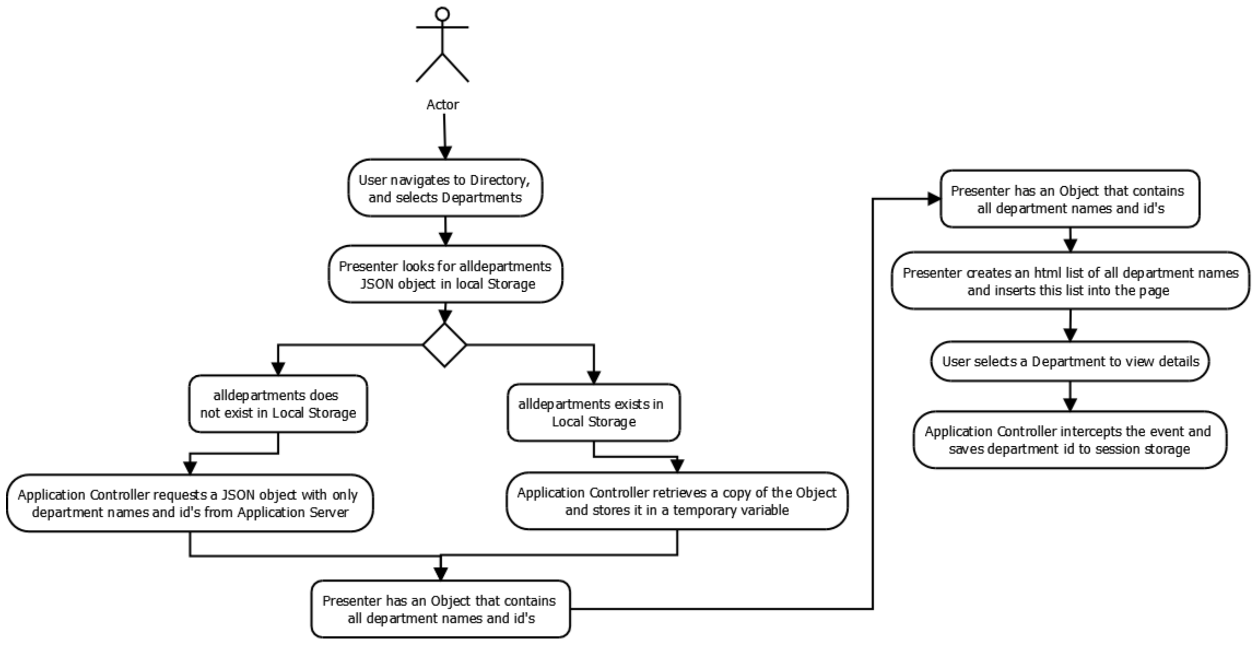

Actor
User navigates to Directory, and selects Departments
Presenter looks for alldepartments JSON object in local Storage
alldepartments does not exist in Local Storage
alldepartments exists in Local Storage
Application Controller requests a JSON object with only department names and id's from Application Server
Application Controller retrieves a copy of the Object and stores it in a temporary variable
Presenter has an Object that contains all department names and id's
Presenter has an Object that contains all department names and id's
Presenter creates an html list of all department names and inserts this list into the page
User selects a Department to view details
Application Controller intercepts the event and saves department id to session storage

Figure 21 Navigating pages that work offline with local storage and session storage

When the user selects a Department, the id of the department selected is saved to Session Storage. The browser loads the Department Details page, which has been saved in the browser’s persistent resource cache, and looks for the id of the Department whose information it must display in Session Storage. Again, the JavaScript in charge of displaying the dynamic information looks for the alldepartments JSON object in Local Storage. When it does not find it, it sends an Ajax call to the server, to request a JSON object containing only information about that Department, but with identical structure to the object it would have found containing alldepartments. The JSON object is parsed for information with JavaScript, and the information is injected into the HTML document.

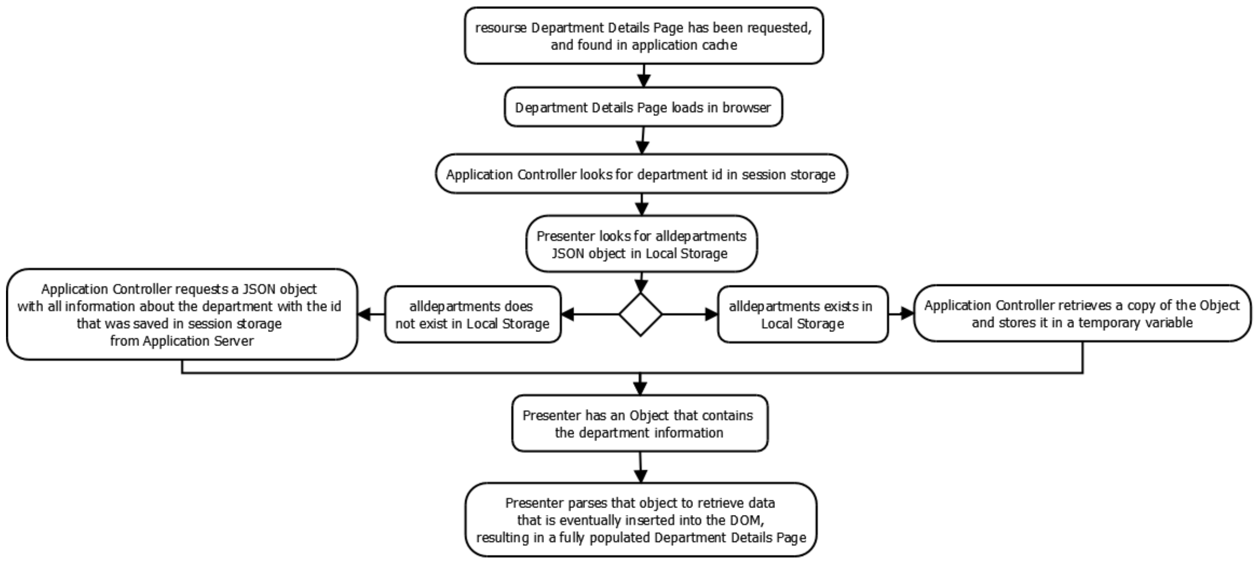

resourse Department Details Page has been requested, and found in application cache
Department Details Page loads in browser
Application Controller looks for department id in session storage
Presenter looks for alldepartments JSON object in Local Storage
Application Controller requests a JSON object with all information about the department with the id that was saved in session storage from Application Server
alldepartments does not exist in Local Storage
alldepartments exists in Local Storage
Application Controller retrieves a copy of the Object and stores it in a temporary variable
Presenter has an Object that contains the department information
Presenter parses that object to retrieve data that is eventually inserted into the DOM, resulting in a fully populated Department Details Page

Figure 22 Navigating pages that work offline with local storage and session storage

If the User opts to download application data locally, then when the Directory pages are loaded, the JavaScript that is in charge of the dynamic information finds the JSON Objects in Local Storage, and does not need to retrieve anything from the server. This essentially means that all application data associated with the Directory and Locations on the map need only be downloaded once, (preferably over a WIFI connection), and after that all browsing can be done irrelevant of internet connectivity.

4.2.6 Rich Media Support

The application that we developed belongs to the class known as **Rich Internet Applications (RIA Applications).** RIA applications are characterized both by the rich interactions like the Ajax type interactions that replace part of the screen at a time using the asynchronous http protocol, as well as by the use of Rich Media Types such as graphics, animations, video, etc. Our application heavily depends on rich media types, and in particular on Graphics data using SVG for diagrams of internal spaces and superimposing layer information on maps and diagrams, on Geographical type of data for presenting external spaces and on geolocation information and presentation. HTML 5 provides support for Rich Media types.

SVG Scalable Vector Graphics for dynamic graphics interaction

Scalable Vector Graphics is a widely deployed, royality-free graphics format developed and maintained by the W3C SVG Working Group. It is a markup language for describing two dimensional graphics applications and images, and a set of related graphics script interfaces. In 2001, the mobile industry chose SVG as the basis for its graphics platform. SVG, with its powerful scripting and event support, can be used as a platform on which to build graphically rich applications and user interfaces. With SVG, developers are able to use a collection of open standards.

Specifically Geographic Information Systems have very specific requirements, rich graphics features, support for vector and raster content and the ability to handle a large amount of data.

We take advantage of SVG in mapping, using it to render interactive vectors on our maps, for floor plan maps, allowing zoom with no information loss, and as a fallback for the Campus map when unable to connect to the mapping server.

OpenLayers and OpenStreetMap Integration

OpenLayers is a JavaScript library for making interactive web maps, cross browser compatible (even IE6), with no server side dependencies. OpenLayers implements a JavaScript API for building rich web based geographic applications, similar to Google Maps API. It is extensively used, free, has a full (yet painful to use) documentation with plenty of helpful examples, and a passionate support community. Also, very important, it supports modern mobile and touch devices. The API is the de facto web mapping library used by almost every open source geo related project.

The idea behind OpenLayers is that there are Base Layers and Overlay Layers. One base layer can be activated at a time, and can be a map provided by any map provider of choice (Google maps, Bing Maps, Yahoo! Maps, OpenStreetMap etc), or can display map tiles from a local file system, or even display an image, or a scalable graphic. Overlay Layers can be activated more than one at a time, and display interactive features such as markers, vectors, and other information.

All maps have a certain distortion, the reason being that they attempt to present three dimensional objects in two dimensions. Projection defines the decision on how the 3 dimensions are flattened out to two dimensions. There are various map projections available, each one makes a tradeoff between some characteristics, the main three being **area** (the size of the map), **scale** (ratio of map distance to actual distance) and **shape** (relative angles to all points are correct). OpenLayers uses the ESPG (European Petroleum Survey Group) classification system to refer to projection.

Why OpenStreetMap and OpenLayers?

The first map provider people think of is Google Maps. We chose to implement our service using maps provided by **OpenStreetMap**, for a variety of reasons:

- Google Maps has begun charging high volume users for its map API. The first companies to abandon Google maps for this reason have been Foursquare and Apple, with many more following suite.
- Google Maps has started offering some of its services only for Android users (Indoor maps and offline maps).
- OpenStreetMap provides crowd sourced maps, much like Wikipedia. It is open source, free, and much more detailed in some parts of the world
- Because OpenStreetMap data is generated by volunteers, features can be added by people who really know the area.

- OpenLayers provides flexibility and customization

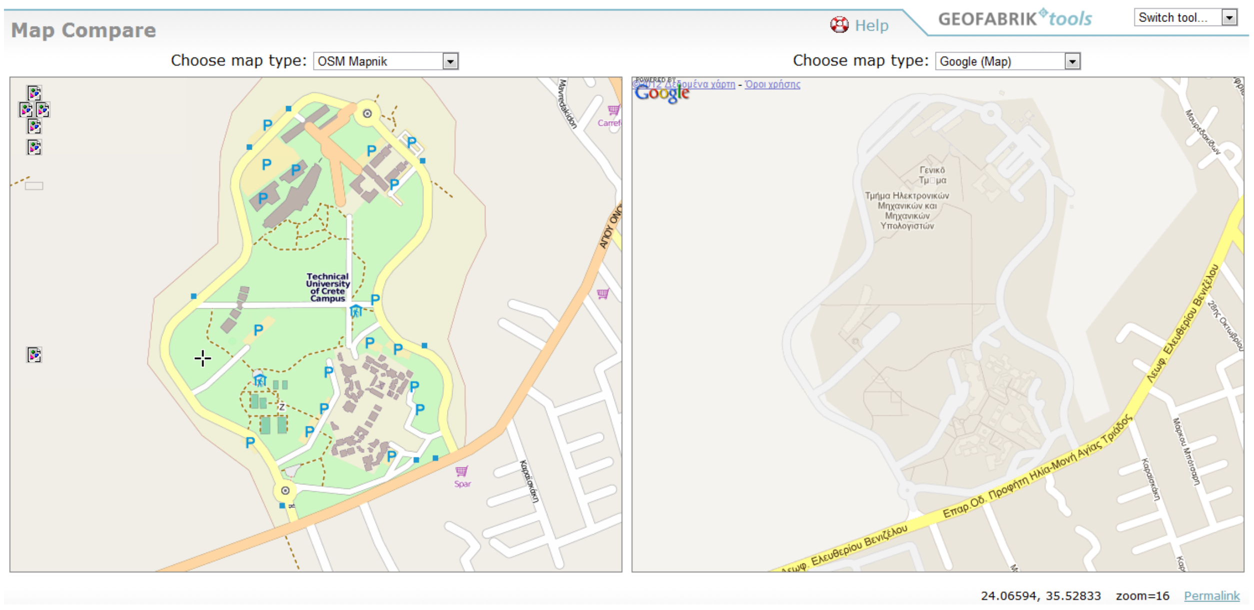

Map Compare
Help
GEOFABRIK tools
Switch tool...
Choose map type: OSM Mapnik
Choose map type: Google (Map)
Technical University of Crete Campus
24.06594, 35.52833 zoom=16 Permalink

Figure 23 visual comparison of detail provided by OpenStreetMap (left) versus Google Maps (right), as shown on GEOFABRIK tools

Geolocation Support

The **Geolocation API** defines a high-level interface to location information associated only with the device hosting the implementation, such as latitude and longitude. The API itself is agnostic of the underlying information sources. It does not rely on a single location technology, yet it decides between GPS, assisted GPS, or location inferred from network signals such as WIFI positioning, Cell information, IP Address, Carrier connection. The accuracy of the mentioned methods varies, but the specification states that the geolocation API should provide accuracy information, with a minimum of 95% confidence.

The application uses the Geolocation API to find the location of our user outside, and display it on the campus map.

4.3 Technologies Used

4.3.1 HTML5

APPCACHE API

HTML5 Application Cache interface allows developers to specify in a manifest file, which resources the browser must cache in persistent memory, and be made available to offline users. A manifest file is a simple text file that lists the resources.

Appcache offers the following:

- Offline page browsing and navigation
- Speed. Resources are retrieved once and kept locally, therefore load faster.
- Reduced server load.

A manifest file has three distinct sections, Cache, Network and Fallback.

Cache contains explicitly cached master entries, Network indicates resources that require the user to be online, and Fallback provides alternative resources to be served in place of other files.

Once resources have been cached offline, they remain cached until one of the following happens:

a) The user clears browser data storage
b) The manifest file is modified
c) The app cache is updated programmatically.

Web Storage

Web storage is a collection of software methods and protocols used for storing data in the Web Browser. Web storage supports persistent data storage. There are two main web storage types: local storage and session storage. They correspond to functionalities for persistent cookies and session cookies of the traditional Browsers. The Web Storage was originally part of the HTML5 specification, but it is now a separate specification.

Web storage provides far greater storage capacity (5 MB at least per domain compared to 4 kB (around 1000 times less space) available to cookies. Web Storage information is not directly writeable from the Server, and it is not transmitted to the Server in every http request.

Unlike cookies, which can be accessed by both the server and client side, web storage falls exclusively under the purview of client side scripting. Web storage data is not transmitted to the server in every HTTP request, and a web server can't directly write to Web storage, but can of course issue read and write requests.

Web storage offers two different storage areas—local storage and session storage—which differ in scope and lifetime. Data placed in local storage is per domain (it's available to all scripts from the domain that originally stored the data) and persists after the browser is closed.

Session storage is per-page-per-window and is limited to the lifetime of the window. Session storage is intended to allow separate instances of the same web application to run in different windows without interfering with each other, a use case that's not well supported by cookies.

Web storage currently provides a better programmatic interface than cookies because it exposes an associative array data structure where the keys and values are both strings http://dev.w3.org/html5/webstorage/

Javascript and DOM

JavaScript was formalized in the ECMAScript language standard and is used in the form of client side JavaScript which is implemented as part of a Web Browser in order to give enhanced user interfaces and dynamic web sites. This enables programmatic access to computational objects within a host environment. The term **dynamic HTML**, or **DHTML is used to indicate** a collection of technologies used together to create interactive and animated Web Sites by using a combination of a static HTML a Client side scripting language (such as JavaScript, a presentation definition language (such as CSS), and the Document Object Model.

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, after the page has been fully loaded and during the viewing process. For example DHTML allows the page author to use a form to capture user input, and then process and respond to that data without having to send data back to the server or to include drop-down menus.

A page in the presentation can be seen as a hierarchy of presentation objects that contain other objects. Thus the nodes of every document are organized in a tree structure called the Document Object Model (DOM) tree. The topmost node in the DOM tree is the Document object. Each node has zero or more children. When an HTML page is rendered in a browser, the browser parses the markup (e.g. HTML), downloaded from the web-server into an in-memory DOM. The DOM is used to construct additional internal structures used to display the page in the browser window. The name "Document Object Model" was chosen because it is an object model in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes of the tree do not represent a data structure; they represent objects, which have functions and identity. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects

The list of all properties, collections and methods of the W3C DOM that can be reliably used in all major DOM browsers are defined by the DOM specifications and include the definition of keys, creating nodes, walking the tree, attaching, copying or removing nodes, node information, element node attribute information, node style, element node size and position, style sheets, adding or removing event listeners, creating event objects, preparing event objects, firing events, getting information about event objects methods and properties.

jQuery

jQuery is a cross browser JavaScript Library designed to simplify the Client Side Scripting of HTML, and it is the most popular JavaScript in use today.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations handle events and develop Ajax Applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.

jQueryMobile

jQuery mobile is a “touch-optimized web framework for Smartphone’s and Tablets.” It is a “unified, HTML-5 based user interface system for all popular mobile device platforms, built on jQuery and jQuery UI. Applications developed with Query mobile framework work on all popular smart phone, tablet and desktop platforms. It is very easy to use, and has great free support on the web. We used it for development of the elements of the user interface in the mobile Clients and the handling of the events.

Servlets

A **Servlet** is a Java-based server-side web technology. A Servlet is a Java Class in Java EE that conforms to the Java Servlet API and that serves a Web Client request and receives a response from the Web Server. A software developer may use a servlet to add dynamic content to a Web Server in the java platform. The generated content is commonly HTML. Servlets are the Java counterpart to non-Java dynamic Web content technologies such as PHP and QASP. NET. Servlets can maintain state in session variables across many server transactions by using HTTP cookies or URL rewriting. To deploy and run a Servlet, a Web Container must be used. A Web container (also known as a Servlet container) is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights. We used Servlets in the Control Layer of the Server.

Java Servlet Pages

JavaServer Pages (**JSP**) is a technology that helps software developers to create dynamically generated web pages based on HTML. JSPs are translated into servlets at runtime; each JSP's servlet is cached and re-used until the original JSP is modified. JSP can be used independently or as the view component of a server-side Model View Controller design, normally with JavaBeans as the model and Java Servlets as the Controller.

JSP allows Java code and certain pre-defined actions to be interleaved with static web markup content, with the resulting page being compiled and executed on the server to deliver a document. The compiled pages, as well as any dependent Java libraries, use Java bytecode rather than a native software format. Like any other Java program, they must be executed within a Java Virtual Machine that integrates with the server's host operating system to provide an abstract platform-neutral environment. JSP pages use several delimiters for scripting functions. The most basic is **<% ... %>**, which encloses a JSP *scriptlet.* A scriptlet is a fragment of Java code that is run when the user requests the page.

We used JSPs in the Boundary Layer of the Server for constructing the output pages.

mySQL

MySQL is a open source data base management system. We used it for managing the data base of the application.

mySQL workbench

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, and much more. MySQL Workbench is available on Windows, Linux and Mac OS. We used it for designing the application database.

Netbeans IDE

The NetBeans IDE is an integrated development environment available for Windows, Mac, Linux, and Solaris. The NetBeans project consists of an open source IDE and an application platform that enable developers to rapidly create web, enterprise, desktop, and mobile applications using the Java platform, as well as PHP, JavaScript, Ajax, Groovy and Grails, and C/C++. We used it as a development environment for all our code.

Inkscape

Inkscape is an open source, SVG-based vector drawing program

It is useful for drawing:

- Illustrations for the Web
- Graphics for mobile phones
- Simple line drawings
- Cartoons
- Complex works of art
- Figures for articles and books
- Organization charts

The file format that Inkscape uses is compact and quickly transmittable over the Internet. Yet it is powerful and can describe complex drawings that are scalable to any size. Support for the format has been added to web browsers and is already included in many mobile phones.

Inkscape supports the drawing of regular shapes (rectangles, circles, etc.), arbitrary paths, and text. These *objects* can be given a wide variety of attributes such as color, gradient or patterned fills, alpha blending, and markers. Objects can be transformed, cloned, and grouped. Hyperlinks can be added for use in web browsers. The Inkscape program aims to be fully XML, SVG, and CSS compliant.

4.4 Summary

In this Chapter we described the Architecture that we designed of the Application Development as well as the tools that we have used for the implementation of the architecture.

Chapter 5 - Data Base Design, Implementation and Data Base Interface

5.1 Introduction

In this chapter we describe the Database we designed and implemented to support the application. The structure of the data base offers category directories as well as complex associations for browsing, navigation, and data mining in order to facilitate users to browse and search through information to find what is desirable. The design is shown with ER diagrams, while the implementation with the relational schema and the data base transactions supported.

5.2 Data Base Design

University Person is an entity that describes all persons that work or study at the University. Their functionality at the university may be one of [professor, undergraduate, graduate, PhD candidate, employee, research collaborator or secretary]. A person has a *first and last name, photo, email address, url,* and *telephone number.* That person might also have an *office* somewhere.

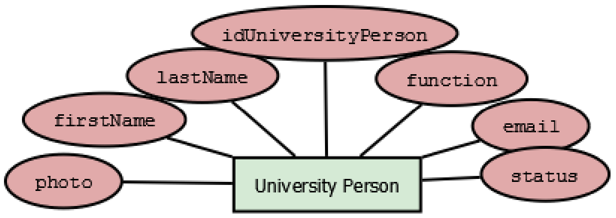


Figure 24 Entity University Person, with attributes

Each University Person may request that the University issues them a **University Person Profile**, with *username and password,* which she uses to log in to university services.

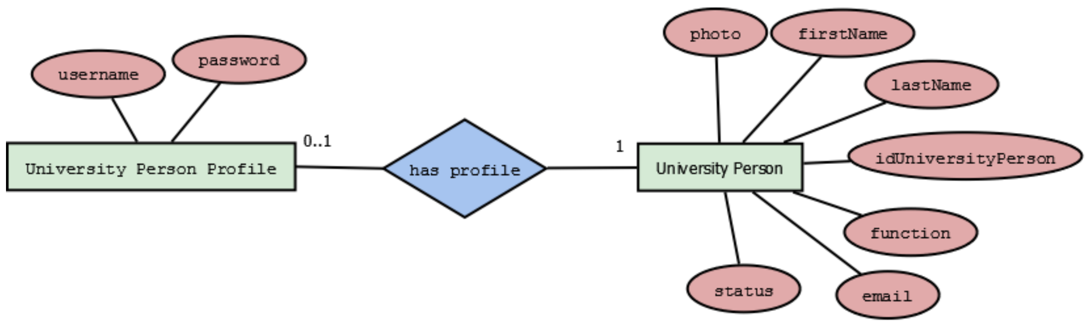


Figure 25 Entities, University Person and University Person Profile. Each person can have a profile, with which he accesses university web services

The University has **Departments**, which in turn have **Labs.** Each person is a member of a Department and may be a member of a Lab.

A **Department** is of *type* academic or service, has a *short name* (ex: ECE), a *long name* (ex: Electronic and Computer Engineering), some *information*, a *URL* of the department website, and a department *chair.*

A **Lab** has a *short* and *long name, logo, description* and *URL* of the lab homepage. Persons are members of labs, and function as employees, director, and faculty, undergraduate, graduate of PhD candidates.

The University Campus has **Buildings** that house **Departments,** and **Facilities.** Each Building has **Floors**, where labs, person's offices and other places are located, and thus tied to an **Indoor Location**. A person may specify a room as his office, and a lab may be located at more than one indoor locations.

A **Floor** has a *floor plan*, and is at a *level* in a building.

An **Indoor Location** describes a geometry on a Floor's floor plan, with *geometry type* and *geometry coordinates,* is given a *code* and its functionality is described with its *attribute* (office, lab, stairs, elevator, etc..).

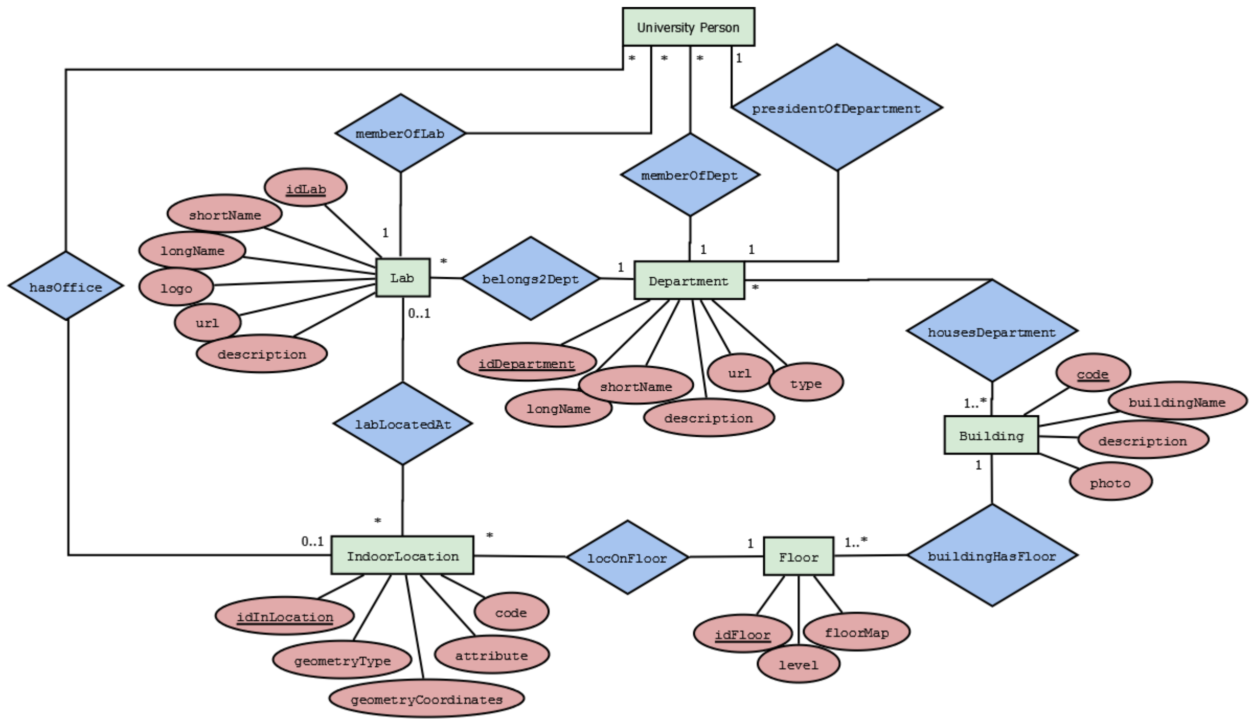


Figure 26 Description of University Person, Lab, Department, Building, Floor, and Indoor Location Entities, and their relations

Each Building is tied to an **Outdoor Location** which describes geometry on the campus map. Outdoor Locations are described by their *geometry type* and *geometry coordinates, attribute* (building, parking, bus stop, etc..) and *code.*

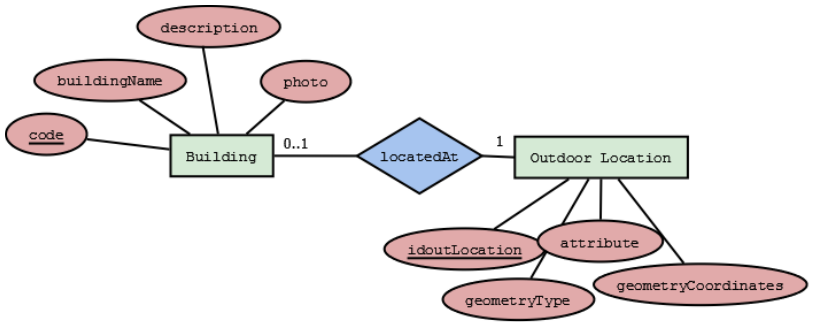


Figure 27 A Building is located at specific GPS coordinates described in Outdoor Location.

A user can create **Groups** of University Persons, giving the group a *name.* She can also create **Events**, where each event is of a certain *type* (Uncategorized, Lecture, Seminar, Thesis defense, Ride, Lab, Exam, Workshop, Conference, Recreation, Excursion), has a *title* and a *description,* is defined by time and thus has a *start date, start time, end date, end time,* may be an *all day event,* and might be open for limited persons with an *attendance limit.* Events can be publicly visible or have a *visibility* limit to University users or University subgroups : (Ece Users,Mred Users, Mped Users, EnvEng Users, ArchEng Users, Gen. Science Users, users that belong to a lab, users that have offices in a building or on a floor), or may be limited to custom visibility. When visibility is custom event creator selects specific users event is to be visible to. An event occurs at any number of locations on campus, indoors or outdoors. A User may subscribe to events of a certain type, or posted by certain users.

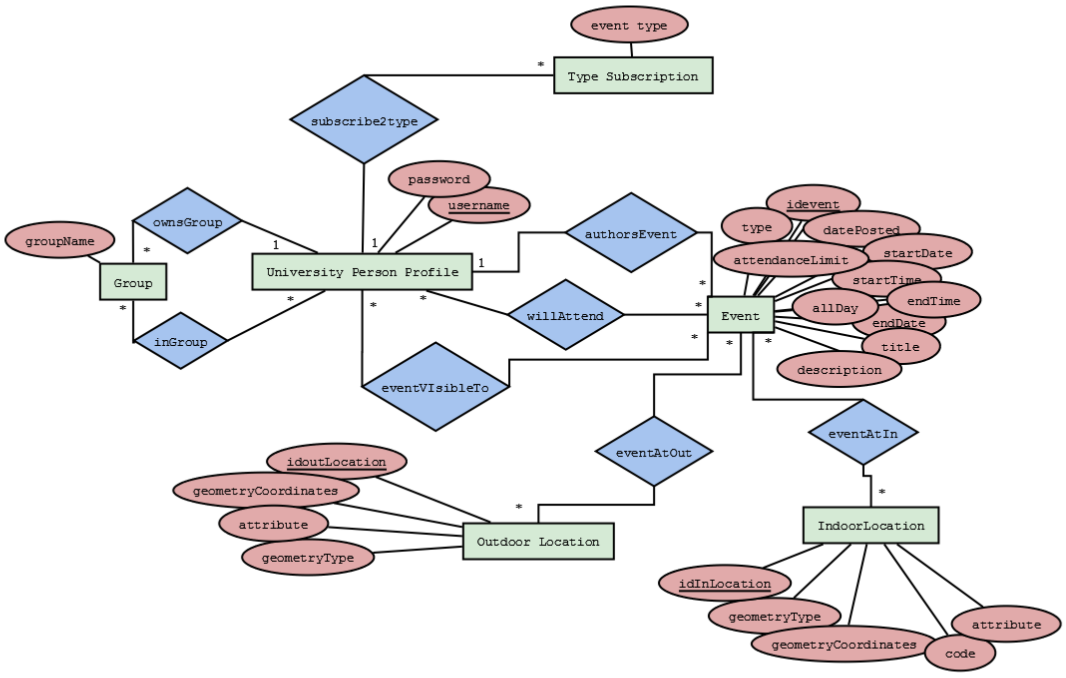


Figure 28 Entities that describe Events. An event is created by a Person, can have specific visibility, is tied to space and time, can have limited attendance capacity and Users can subscribe to events of a certain type, or posted by certain persons

The complete Database Entity Relationship Schema is shown below:

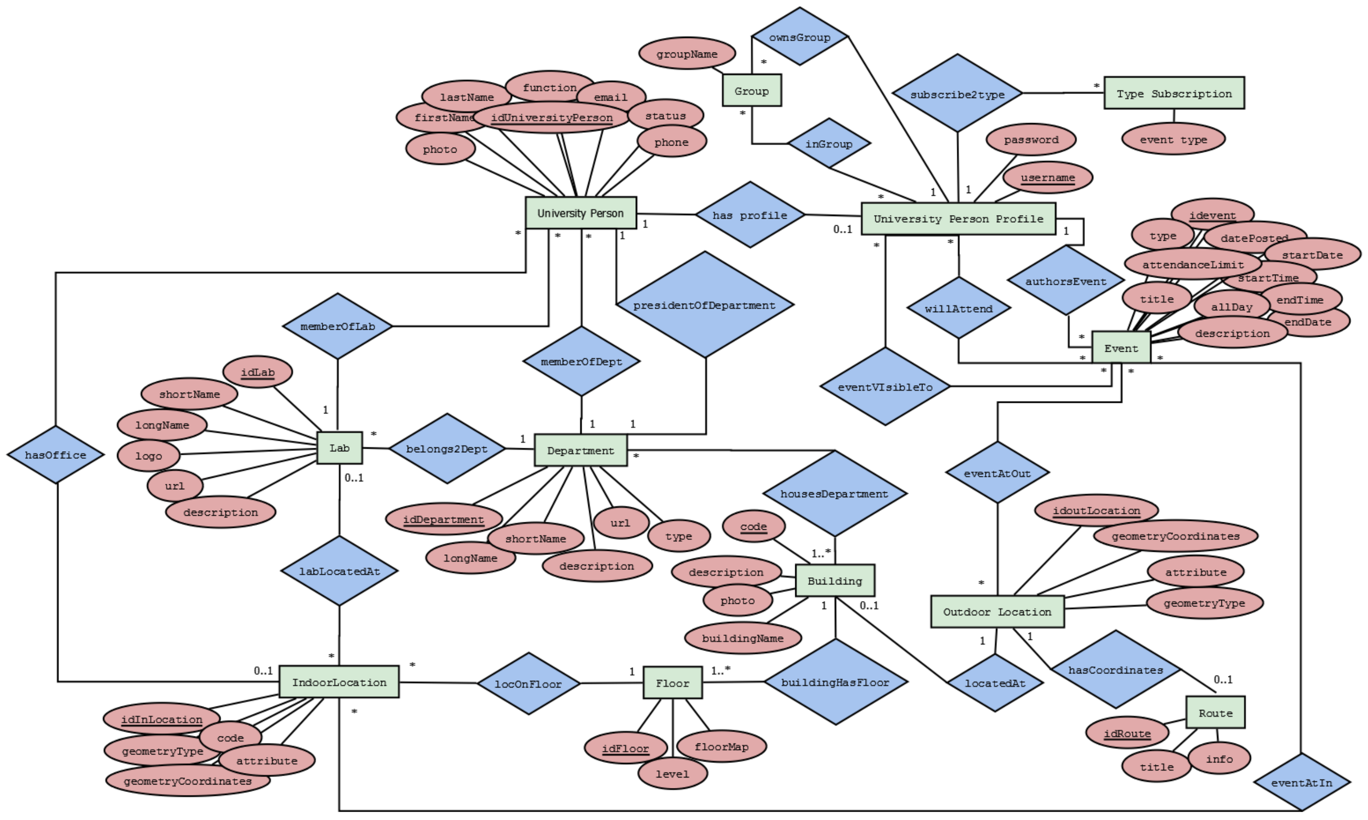


Figure 29 Complete Entity Relationship diagram describing the database design for the TUC Campus application

5.3 Relational Schema

The ER schema of the database is translated into a Relational Schema that reflects the tables used in the database. Each table name is given with all the table attributes. Attributes that function as primary keys are underlined, ex: <u>this is a key</u> , and all foreign keys that are used to keep information about a relationship are shown with (FK) beside them, ex: this_is_a_foreign_key (FK).

University Person Entity is translated into a table, with all its attributes plus two extra attributes that function as foreign keys to show the relation between a person and a department, and the relation between a person and a location (office).

University Person Profile is translated into a table, with a foreign key to show who which person the profile is associated with.

Locations on Campus are all described in table out_locations.

Building entity is translated into a table with a foreign key to the outdoor location

Building Floor entity is translated into a table with a foreign key to the building it belongs to.

An Indoor location is translated into a table with a foreign key to the building floor it refers to.

A Department is housed in at least one or more Buildings, and in turn a Building may house a number of Departments. This relationship is kept in a table dept_housed_in_building and holds the primary keys of building and departments paired together, where the pair is unique and thus the combination can function as a primary key.

A Lab is saved to table lab, and is connected to a department and an indoor location with foreign keys to those relations.

A user's membership in lab is described by table memberOfLab, and preserves the relation with foreign keys to lab and person.

An Event is saved to table event. The relation also keeps an attribute person that works as a foreign key to University Person so we know who the event owner is.

When an event's event visibility is "custom", the users that can view the event are found in table event_visibility, with foreign keys to the event and the persons that have access to the event.

An event occurs at any number of outdoor or indoor locations. Therefore we need a table for each of the relationships, one to keep the pairs of indoor locations, and event id, and one to pair outdoor locations with event id. Each location added to an event can be accompanied by a short description by the user.

An event may be attended by one or more persons. This relationship is kept in a table event_attendance, with foreign keys to users and events.

A User can subscribe to all events posted by another user. Subscribe2user is a table that keeps this relationship with a foreign key for each user, the pair of which is unique and thus constitutes the primary key.

A User may also subscribe to events of a certain type. Subscribe2type is a table that keeps this relationship with a foreign key for the user and the type e is subscribed to, the pair of which is unique and acts as a primary key.

A user creates Groups of users and they are stored in table user_group, with a foreign key to group owner. The members of the group are stored in another table group members where the combination of two foreign keys (group id and member id) act as the primary key.

Table Name	Attributes
university_person	idUniversityPerson , firstName, lastName, photo, email, status, url, telephone, department (FK), office (FK)
university_person _profile	username , password, person (FK)
out_location	idout_location , geometry_type, geometry_coordinates, attribute, featureCode
building	code , name, description, photo, out_location (FK)
building_floor	idbuilding_floor , floor_num, floorplan, building_id (FK)
in_location	idin_location , geometry_type, geometry_coordinates, attribute, code, floor (FK)
department	iddepartment , department_name_short, department_name_long, department_information, department_url, department_president (FK)
lab	idLab, labLongNAme, labShortName, labLogo, labDescription, labUrl, labInLocation(FK), belongs2department(FK)
memberOflab	idmemberOfLab, status, person(FK), lab(FK)
dept_housed_ in_building	department (FK), building (FK)
event	idevent , event_title, event_type, event_start, event_end, event_description, event_attendance_limit, event_visibility, date_created, all_day_event, event_creator (FK)
event_visibility	user_profile (FK), event_id (FK)
event_inloc	inlocation(FK), event(FK), comment
event_outloc	outlocation(FK), event(FK), comment
event_attendance	event (FK), profile(FK)
subscribed2type	profile(FK), type
subscribed2user	subscriber(FK), subscribedTo (FK)
user_group	idgroup , groupName, groupOwner(FK)
group_member	group (FK), member (FK)

Table 1 Relational Schema of the database, description of all the relations and their attributes. Primary keys are underlined and foreign keys have an (FK) to their right.

5.4 Summary

In this Chapter we have described the design of the Data Base of the Application. The Data Base supports involved structuring to allow complex points of view and navigations for information mining and information association. We Described the Entity Relationship Model, and the Relational Schema that implements the data base.

Chapter 6 – Web Service Interface

6.1 Introduction

In this chapter we describe the web service interface that the application offers for the functionalities related to places or to the structure of places and people within the University. This service interface is used currently for offering the rich client functionality for visualizing places services and people within the university and offering online and offline access modes to the mobile clients. The service interface that is described in here can be used in the future for building other mobile applications that need this kind of service and build further functionality on top of it.

This chapter describes the different functionalities offered, the APIs used in the form of http calls and the object format transferred as result of the call in the form of JSON objects.

The data retrieved from the application server via the service interface is stored in Client Local Storage without further processing, and reflects the data stored in the server database. JSON objects stored in Local Storage are accessed by their unique keys, given to them by the client side developer.

6.2 Service Interfaces and Message Formats

In the tables below we describe functions that the client developer can use to get information from the application server, that he can process or use as is. We describe the input format, output format and the http call that must be made.

Campus map related features:

Get Building map features	
Functionality	this method returns a JSON object that describes building features on the campus map
Input	
Output	JSON object Example:

```
{
    "type": "FeatureCollection",
    "features": [

        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_13",
                "code": "D4"
            },
            "geometry": {
                "type": "MultiPolygon",
                "coordinates": [
                    [
                        [
                            [
                                24.069516477583264,
                                35.52652270491111
                            ],

                            ...,
                            [
                                24.069516477583264,
                                35.52652270491111
                            ]
                        ]
                    ],
                    [
                        [
                            [
                                24.06951111316508,
                                35.526671142431724
                            ],
                            ...,
                            [
                                24.06951111316508,
                                35.526671142431724
                            ]
                        ]
                    ]
                ]
            }
        },

        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_28",
                "code": "oldLib"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.070144114492233,
                            35.527561761789336
                        ],…,
                        [
                            24.070144114492233,
                            35.527561761789336
                        ]
                    ]
                ]
            }
        }
    ]
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_BUILDING_FEATURES

Get campus parking lots

Functionality	this method returns a JSON object that describes parking lots on the campus map
Input	
Output	JSON object Example:

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_26",
                "code": "P7"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.070144114492233,
                            35.52845673697216
                        ],
                        [
                            24.070192394254946,
                            35.528565879604336
                        ],
                        [
                            24.070085105894062,
                            35.52848293121738
                        ],
                        [
                            24.070144114492233,
                            35.52845673697216
                        ]
                    ]
                ]
            }
        },
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_33",
                "code": "Science Building parking B"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.068320212363492,
                            35.53189029304867
                        ],
                        [
                            24.068288025855313,
                            35.53192521721278
                        ],
                        [
                            24.06796616077356,
                            35.53166328561319
                        ],
                        [
                            24.068320212363492,
                            35.53189029304867
                        ]
                    ]
                ]
            }
        }
    ]
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PARKING_ON_MAP

Get campus bus stops

Functionality	this method returns a JSON object that describes bus stop features on the campus map
Input	
Output	JSON object

Example:

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_18",
                "code": "mpd"
            },
            "geometry": {
                "type": "POINT",
                "coordinates": [
                    24.070975599285717,
                    35.5263829987588
                ]
            }
        },
       …,
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_23",
                "code": "mhxop"
            },
            "geometry": {
                "type": "POINT",
                "coordinates": [
                    24.071098980901137,
                    35.53207582749406
                ]
            }
        }
    ]
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_BUS_STOPS_ON_MAP

Get campus athletic courts

Functionality	this method returns a JSON object that describes athletic courts features on the campus map
Input	
Output	JSON object Example:

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_31",
                "code": "tennis court 1"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.067295608520954,
                            35.52721250047491
                        ],
                        [
                            24.067306337356413,
                            35.52688943240503
                        ],
                        [
                            24.067537007331822,
                            35.52688943240503
                        ],
                        [
                            24.067510185241826,
                            35.52721250047491
                        ],
                        [
                            24.067295608520954,
                            35.52721250047491
                        ]
                    ]
                ]
            }
        },
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_34",
                "code": "tennis court 2"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.06769257545362,
                            35.52720813469948
                        ],
                        [
                            24.06770330428998,
                            35.52690689557772
                        ],
                        [
                            24.06790178775721,
                            35.52690689557772
                        ],
                        [
                            24.06790178775721,
                            35.527216866251614
                        ],
                        [
                            24.06769257545362,
                            35.52720813469948
                        ]
                    ]
                ]
            }
        },
        {
            "type": "Feature",
            "properties": {
                "locid": "outLocId_35",
                "code": "football field 1"
            },
            "geometry": {
                "type": "POLYGON",
                "coordinates": [
                    [
                        [
                            24.067059574126468,
                            35.52763597962516
                        ],
                        [
                            24.06706493854465,
                            35.527387131323245
                        ],
                        [
                            24.06723659992099,
                            35.527391497089695
                        ],
                        [
                            24.067231235503705,
                            35.527640345378074
                        ],
                        [
                            24.067059574126468,
                            35.52763597962516
                        ]
                    ]
                ]
            }
        }
    ]
```

	}
http syntax	http://147.27.41.130:8080/jqCampus/controller?action=GET_SPORTS_ON_MAP

Building floor map feature related calls:

Get all offices on floor	
Functionality	this method returns a JSON object that describes all office locations on a given floor
Input	String floorId
Output	JSON object Example: `{ "floor137OfficeFeatures": { "type": "FeatureCollection", "features": [{ "type": "Feature", "properties": { "locid": "inLocId_83", "code": "137.p40" }, "geometry": { "type": "POLYGON", "coordinates": [[[8.6785714285669, -25.585714285711], [5.271428571424, -27.128571428568], …, [9.1928571428526, -26.421428571426], [8.6785714285669, -25.585714285711]]] } }, { "type": "Feature", "properties": { "locid": "inLocId_112", "code": "137.B71.2" }, "geometry": { "type": "POLYGON", "coordinates": [[[46.221428571428, -25.039285714277], …, [46.221428571428, -25.039285714277]]] } }`

```
{
    "floor137OfficeFeatures": {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_83",
                    "code": "137.p40"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                8.6785714285669,
                                -25.585714285711
                            ],
                            [
                                5.271428571424,
                                -27.128571428568
                            ],
                            …,
                            [
                                9.1928571428526,
                                -26.421428571426
                            ],
                            [
                                8.6785714285669,
                                -25.585714285711
                            ]
                        ]
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_112",
                    "code": "137.B71.2"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                46.221428571428,
                                -25.039285714277
                            ],
                            …,
                            [
                                46.221428571428,
                                -25.039285714277
                            ]
                        ]
                    ]
                }
            }
```

	`]`
	`}`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action=GET_OFFICE_FEATURES&floor=floorId

Get Labs listing on Floor

Functionality	This method returns a JSON object that that describes only basic lab information for all labs on a given floor of a building on campus. The information described includes the lab id, lab location, short and long name.
Input	String floorid

Output	JSON object
	Example:
	`{`
	`    "labs": {`
	`        "id_5": {`
	`            "ilabInLocation": "58",`
	`            "labName": "Analytical & Environmental Chemistry Laboratory",`
	`            "labShortName": "CheEnv"`
	`        },..,`
	`        "id_3": {`
	`            "ilabInLocation": "17",`
	`            "labName": "Software Technology And Network Applications Laboratory",`
	`            "labShortName": "SoftNet"`
	`        }`
	`    }`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABS_ON_FLOOR&floor=floorid

Get all Labs locations on floor

Functionality	this method returns a JSON object that describes all Lab locations on a given floor
Input	String floorId

Output	JSON object
	
	Example:
	
	`{`
	`    "floor137LabFeatures": {`
	`        "type": "FeatureCollection",`
	`        "features": [`
	`            {`
	`                "type": "Feature",`
	`                "properties": {`
	`                    "locid": "inLocId_80",`
	`                    "code": "137.P42"`
	`                },`
	`                "geometry": {`
	`                    "type": "POLYGON",`
	`                    "coordinates": [`
	`                        [`
	`                            [`
	`                                11.442857142852,`
	`                                -44.871428571427`
	`                            ],`
	`                            [`

```
                                7.5857142857091,
                                -33.814285714284
                            ],
                            [
                                8.3571428571377,
                                -45.899999999999
                            ],
                            [
                                11.442857142852,
                                -44.871428571427
                            ]
                        ]
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_113",
                    "code": "137.B73"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                50.207142857143,
                                -20.410714285705
                            ],

                            [
                                78.621428571428,
                                -22.339285714277
                            ],
                            [
                                75.021428571428,
                                -13.082142857134
                            ],
                            [
                                61.778571428571,
                                -17.582142857134
                            ],
                            [
                                50.207142857143,
                                -20.410714285705
                            ]
                        ]
                    ]
                }
            }
        ]
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABS_FEATURES &floor=floorId

Get all amphitheatres and conference halls on floor

Functionality	this method returns a JSON object that describes all conference rooms and amphitheatre locations on a given floor
Input	String floorId
Output	JSON object

Example:

```
{
    " floor141HallFeatures": {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_80",
                    "code": "137.P42"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                11.442857142852,
                                -44.871428571427
                            ],
                            [
                                7.5857142857091,
                                -33.814285714284
                            ],
                            [
                                8.3571428571377,
                                -45.899999999999
                            ],
                            [
                                11.442857142852,
                                -44.871428571427
                            ]
                        ]
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_113",
                    "code": "137.B73"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                50.207142857143,
                                -20.410714285705
                            ],

                            [
                                78.621428571428,
                                -22.339285714277
                            ],
                            [
                                75.021428571428,
                                -13.082142857134
                            ],
                            [
                                61.778571428571,
                                -17.582142857134
                            ],
                            [
                                50.207142857143,
                                -20.410714285705
                            ]
                        ]
                    ]
                }
            }
        ]
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABS_FEATURES &floor=floorId

Get all elevators on floor

Functionality	this method returns a JSON object that describes all elevator locations on a given floor
Input	String floorId
Output	JSON object

Example:

```
{
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_83",
                    "code": "137.p40"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                8.6785714285669,
                                -25.585714285711
                            ],
                            [
                                5.271428571424,
                                -27.128571428568
                            ],
                             …,
                            [
                                9.1928571428526,
                                -26.421428571426
                            ],
                            [
                                8.6785714285669,
                                -25.585714285711
                            ]
                        ]
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_112",
                    "code": "137.B71.2"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                46.221428571428,
                                -25.039285714277
                            ],
                            [
                                50.721428571428,
                                -23.882142857134
                            ],
                            [
                                52.392857142857,
                                -30.182142857134
                            ],
                            [
                                47.25,
                                -31.082142857134
                            ],
                            [
                                46.221428571428,
                                -25.039285714277
                            ]
                        ]
                    ]
                }
            }
        ]
    }
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action=GET_FLOOR_ELEVATORS&floor=floorId

Get all washrooms on floor

Functionality	this method returns a JSON object that describes all washroom locations on a given floor
Input	String floorId
Output	JSON object Example:

```
{
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_80",
                    "code": "137.P42"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                11.442857142852,
                                -44.871428571427
                            ],
                            [
                                7.5857142857091,
                                -33.814285714284
                            ],
                            [
                                8.3571428571377,
                                -45.899999999999
                            ],
                            [
                                11.442857142852,
                                -44.871428571427
                            ]
                        ]
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "locid": "inLocId_113",
                    "code": "137.B73"
                },
                "geometry": {
                    "type": "POLYGON",
                    "coordinates": [
                        [
                            [
                                50.207142857143,
                                -20.410714285705
                            ],

                            [
                                78.621428571428,
                                -22.339285714277
                            ],
                            [
                                75.021428571428,
                                -13.082142857134
                            ],
                            [
                                61.778571428571,
                                -17.582142857134
                            ],
                            [
                                50.207142857143,
                                -20.410714285705
                            ]
                        ]
                    ]
                }
```

	`}`
	`]`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_FLOOR_WC &floor=floorId

Get Persons listing with office on floor	
Functionality	This method returns a JSON object that contains certain information about persons with offices on a given floor. Information includes the person id, first and last name, and location of office.
Input	String floorid

Output	JSON object
	Example:
	`{`
	`    "persons": {`
	`        "id_30": {`
	`            "idUniversityPerson": "30",`
	`            "firstName": "Giorgos",`
	`            "lastName": "Anestis",`
	`            "in_loc": "25"`
	`        },`
	`        ...,`
	`        "id_21": {`
	`            "idUniversityPerson": "21",`
	`            "firstName": "Ioanna",`
	`            "lastName": "Trochatou",`
	`            "in_loc": "2"`
	`        }`
	`    }`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSONS_WITH_OFFICE_ON_FLOOR&floorId = floorid

University directory related calls:

Get all Persons	
Functionality	This method returns a JSON object that describes University Persons as they are described in the database, plus information to preserve associations. The information in contains includes person id , first and last names, status, email, url, department id, department name, telephone, indoor location code, office code and floor.
Input	

Output	JSON object
	Example:
	`{`
	`    "persons": {`
	`        "id_21": {`
	`            "idUniversityPerson": "21",`
	`            "firstName": "Ioanna",`
	`            "lastName": "Trochatou",`
	`            "status": "Graduate",`
	`            "email": "trioann@gmail.com",`
	`            "url": "",`
	`            "department": "1",`
	`            "department_name": "Electronic and Computer Engineering ",`
	`            "tel": "",`
	`            "in_loc": "2",`

	`"office_code": "141.B.70.1", "floor": "141", "office": "2" },…, "id_36": { "idUniversityPerson": "36", "firstName": "Michalis", "lastName": "Zervakis", "status": "Professor", "email": "", "url": "", "department": "1", "department_name": "Electronic and Computer Engineering ", "tel": "", "in_loc": "65", "office_code": "145.A36.2", "floor": "145", "office": "65" } } }`
http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSONSJSON

Get all Persons listing (light)	
Functionality	This method returns a JSON object that describes information needed for directory listing and contains all persons of the University as they are described in the database. The information described includes the person's id, their name, status in the University, and department they belong to.
Input	
Output	JSON object Example: `{ "persons": { "id 12": { "idUniversityPerson": "12", "firstName": "Nikolaos", "lastName": "Agadakos", "status": "Undergraduate", "department_name": "Electronic and Computer Engineering " },...., "id_36": { "idUniversityPerson": "36", "firstName": "Michalis", "lastName": "Zervakis", "status": "Professor", "department_name": "Electronic and Computer Engineering " } } }`
http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSONSJSON_MIN

Get all Departments	
Functionality	this method returns a JSON object that describes University Departments as they are described in the database, plus information to preserve associations
Input	
Output	JSON object

Example:

```
{
    "departments": {
        "id_8": {
            "iddepartment": "8",
            "department_name_short": "ARCH",
            "department_name_long": "Architectual Engineering",
            "department_information": "The establishment of this Department was
approved by the Senate at its session 96/3-11-2002. It comprises four sectors; the
title of the three first of them clearly state their content. More particularly,
these sectors are oriented towards an ecological aspect of the architectural and
urban design, urban and town planning and landscape architecture. This department
takes into serious consideration the relation between closed spaces and their
influence to the environment and ecology of the area. Furthermore, it emphasizes on
the preservation, maintenance-protection and restoration of monuments, ecosystems and
natural sources while at the same time, it incorporates the natural recycling process
into the design and management of space. The fourth sector of the department focuses
on civil engineering issues thus offering courses in construction, material
resistance, material technology etc.",
            "department_url": "http://www.arch.tuc.gr/",
            "department_president": "19",
            "firstName": "Emmanuel",
            "lastName": "Marmaras",
            "person_ids": [
                19
            ],
            "lab_ids": []
        },

        "id_1": {
            "iddepartment": "1",
            "department_name_short": "ECE",
            "department_name_long": "Electronic and Computer Engineering ",
            "department_information": "The Department of Electronic & Computer
Engineering offers a modern, complete programme of undergraduate and postgraduate
studies. It admitted its first students in 1990. Ever since, the department has made
continuous and significant progress, which is shown by the quality of the study
programme, the international recognition of the teaching staff working in the
department, the successful careers of its graduates and the quality of the research
work conducted. Both the educational and the research work conducted at the
department are internationally acknowledged.",
            "department_url": "http://www.ece.tuc.gr/",
            "department_president": "3",
            "firstName": "Minos",
            "lastName": "Garofalakis",
            "person_ids": [
                12,
                18,
                41,
                26,
                33,
                37,
                8,
                21,
                36
            ],
            "lab_ids": [
                4,
                2,
                1,
                7,
                3,
                6
            ]
        },
        …..,

        "id_7": {
            "iddepartment": "7",
            "department_name_short": "SCIENCE",
            "department_name_long": "General Science ",
            "department_information": "The objective of the Department of Sciences is
to provide scientific training in basic disciplines thus offering curriculum support
to the rest of the departments of the Technical University of Crete. At the same
time, the Department promotes research in applied sciences and engineering.",
            "department_url": "http://www.science.tuc.gr/",
            "department_president": "-1",
            "firstName": "",
            "lastName": "",
            "person_ids": [
                35
            ],
            "lab_ids": [
                5
            ]
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_DEPTSJSON

Get all Departments listing (light)

Functionality	This method returns a JSON object that describes information needed for directory listing and contains all departments of the University as they are described in the database. The information described includes the department id, long and short name.
Input	

Output

JSON object
Example:

```
{
    "departments": {

        "id_1": {
            "iddepartment": "1",
            "department_name_short": "ECE",
            "department_name_long": "Electronic and Computer Engineering "
        },
        ...,
        "id_7": {
            "iddepartment": "7",
            "department_name_short": "SCIENCE",
            "department_name_long": "General Science "
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_DEPTSJSON_MIN

Get all Labs

Functionality	This method returns a JSON object that describes all labs of the University as they are described in the database, plus information to preserve associations
Input	

Output

JSON object
Example:

```
{
    "labs": {

        "id_1": {
            "labName": "Laboratory of Distributed Multimedia Information Systems and
Applications ",
            "labShortName": "MUSIC",
            "labDescription": "The TUC/MUSIC Laboratory was established in 1990 in
the Department of Electronics and Computer Engineering of the Technical University of
Crete which is located in Chania, Crete, Greece. The TUC/MUSIC Laboratory is a center
of research, development and teaching in the areas of distributed information
systems, application engineering, computer graphics, and simulation engineering. We
hope that you find the information in this site useful, and remain at your disposal
for any additional information that you may want. ",
            "labUrl": "www.music.tuc.gr",
            "ilabInLocation": "15",
            "floor": "141",
            "belongsToDept": "1",
            "lab_members": {
                "id_30": "employee",
                "id_1": "director",
                …,
                "id_5": "undergraduate",
                "id_32": "graduate",
                "id_21": "graduate"
            }
        },…,
        "id_3": {
            "labName": "Software Technology And Network Applications Laboratory",
            "labShortName": "SoftNet",
            "labDescription": "The Software Technology and Network Applications
Laboratory is a unit of the Electronic and Computer Engineering Department of the
Technical University of Crete located in Chania, Crete. The laboratory is a centre of
research and teaching software systems technology and network applications. The
research and teaching activities of the laboratory include operating and distributed
systems, sensor networks, continuous data streams, large and distributed databases,
and topics in algorithms and complexity. ",
            "labUrl": "",
            "ilabInLocation": "17",
            "floor": "141",
            "belongsToDept": "1",
            "lab_members": {
                "id_24": "employee",
                "id_2": "undergraduate",
                "id_6": "faculty",
                "id_3": "director",
                "id_34": "faculty"
            }
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABSJSON

Get all Labs listing (light)

Functionality	This method returns a JSON object that describes information needed for directory listing of a lightweight object containing all labs of the University as they are described in the database. The information described includes the lab id, long and short name.
Input	
Output	JSON object Example:

```
{
    "labs": {
        "id_1": {
            "labName": "Laboratory of Distributed Multimedia Information Systems and
Applications ",
            "labShortName": "MUSIC"
        },
        ...,
        "id_6": {
            "labName": "Telecom",
            "labShortName": "Telecom"
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABSJSON_MIN

Get all Buildings

Functionality	This method returns a JSON object that describes all buildings of the University as they are described in the database, plus information to preserve associations. Each building object contains building id, name location and description
Input	
Output	JSON object Example:

```
{
    "buildings": {

        "id_L": {
            "buildingName": "Science Building",
            "buildingLocation": "1",
            "buildingDescription": "The Science building houses departments of
Electrical and Computer Engineering, and the Science Department, along with all their
offices and laboratories",
            "buildingFloors": [
                "137",
                "141",
                "145"
            ]
        },...,
        "id_M1": {
            "buildingName": "M1",
            "buildingLocation": "6",
            "buildingDescription": "",
            "buildingFloors": [
                "M1.0",
                "M1.1",
                "M1.2"
            ]
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_BUILDINGSJSON

Get all Buildings listing (light)

Functionality	This method returns a JSON object that that describes information needed for directory listing of a lightweight object containing all buildings of the University as they are described in the database. The information described includes the building id, name and building feature id.
Input	
Output	JSON object Example:

```
{
    "buildings": {

        "id_L": {
            "buildingName": "Science Building",
            "buildingLocation": "1"
        },
        ...,
        "id_M1": {
            "buildingName": "M1",
            "buildingLocation": "6"
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action=GET_BUILDINGSJSON_MIN

Get Person Details

Functionality	This method returns a JSON object that that describes a person with a given id. The information described includes the person id, first and last name, status in the department, email, url department id they belong to, department name they belong to, telephone, location id of their office, office code, office floor.
Input	Int personid
Output	JSON object Example:

```
{
    "persons": {
        "id_1": {
            "idUniversityPerson": "1",
            "firstName": "Stavros",
            "lastName": "Christodoulakis",
            "status": "Professor",
            "email": "stavros@ced.tuc.gr",
            "url": "",
            "department": "1",
            "department_name": "Electronic and Computer Engineering ",
            "tel": "28210 37399",
            "in_loc": "66",
            "office_code": "145.A24",
            "floor": "145",
            "office": "66"
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSON_DETAILS_JSON&pid=personid

Get Department Details

Functionality	This method returns a lightweight JSON object that that describes a department with a given id. The information described includes the department id, short and long name, information, url, president, first and last name, ids of persons that belong to it, and ids of labs that belong to it.
Input	Int deptId
Output	JSON object Example:

```
{
    "departments": {
        "id_1": {
            "iddepartment": "1",
            "department_name_short": "ECE",
            "department_name_long": "Electronic and Computer Engineering ",
            "department_information": "The Department of Electronic & Computer
Engineering offers a modern, complete programme of undergraduate and postgraduate
studies. It admitted its first students in 1990. Ever since, the department has made
continuous and significant progress, which is shown by the quality of the study
programme, the international recognition of the teaching staff working in the department,
the successful careers of its graduates and the quality of the research work conducted.
Both the educational and the research work conducted at the department are
internationally acknowledged.",
            "department_url": "http://www.ece.tuc.gr/",
            "department_president": "3",
            "firstName": "Minos",
            "lastName": "Garofalakis",
            "person_ids": [
                12,
                30,
                24,
                …,
                21,
                36
            ],
            "lab_ids": [
                4,
                …,
                3,
                6
            ]
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_DEPT_DETAILS&did=deptId

Get Lab Details

Functionality	This method returns a lightweight JSON object that contains all details about a given lab
Input	int labid
Output	JSON object Example:

```
{
    "labs": {
        "id_1": {
            "labName": "Laboratory of Distributed Multimedia Information Systems and 
Applications ",
            "labShortName": "MUSIC",
            "labDescription": "The TUC/MUSIC Laboratory was established in 1990 in the 
Department of Electronics and Computer Engineering of the Technical University of Crete 
which is located in Chania, Crete, Greece. The TUC/MUSIC Laboratory is a center of 
research, development and teaching in the areas of distributed information systems, 
application engineering, computer graphics, and simulation engineering. We hope that you 
find the information in this site useful, and remain at your disposal for any additional 
information that you may want. ",
            "labUrl": "www.music.tuc.gr",
            "ilabInLocation": "15",
            "floor": "141",
            "belongsToDept": "1",
            "lab_members": {
                "id_30": "employee",
                "id_1": "director",
                "id_28": "employee",
                "id_7": "graduate",
                …,
                "id_5": "undergraduate",
                "id_32": "graduate",
                "id_21": "graduate"
            }
        }
    }
}
```

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LAB_DETAILS_JSON &lid= labid

Get Building Details

Functionality	This method returns a JSON object that that describes only one building given a building id. The information described includes the building id, name, location, description and floors.
Input	String buildingid
Output	JSON object Example:

```
{
```

	`    "buildings": {`
	`        "id_L": {`
	`            "buildingName": "Science Building",`
	`            "buildingLocation": "1",`
	`            "buildingDescription": "The Science building houses departments of Electrical and Computer Engineering, and the Science Department, along with all their offices and laboratories",`
	`            "buildingFloors": [`
	`                "137",`
	`                "141",`
	`                "145"`
	`            ]`
	`        }`
	`    }`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_BUILDING_DETAILS_JSON& bid=buildingid

Get Lab information Person belongs to

Functionality	This method returns a JSON object that that describes a lab that a person belongs to. The information described includes the lab id, lab location, short and long name, floor it belongs on and person ids of persons that are members of it.
Input	Int personid

Output	JSON object
	Example:
	`{`
	`    "labs": {`
	`        "id 1": {`
	`            "labName": "Laboratory of Distributed Multimedia Information Systems and Applications ",`
	`            "labShortName": "MUSIC",`
	`            "ilabInLocation": "15",`
	`            "floor": "141",`
	`            "lab_members": {`
	`                "id_30": "employee"`
	`            }`
	`        }`
	`    }`
	`}}`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LAB_INFO_FOR_PERSON& pid=personid

Get of Lab members listing

Functionality	This method returns a JSON object that that describes all persons that are members of a given lab. The information described includes the person id, first and last name.
Input	String labid

Output	JSON object
	Example:
	`{`
	`    "persons": {`
	`        "id_30": {`

	`"idUniversityPerson": "30",`
	`"firstName": "Giorgos",`
	`"lastName": "Anestis"`
	`},...,`
	`"id_21": {`
	`"idUniversityPerson": "21",`
	`"firstName": "Ioanna",`
	`"lastName": "Trochatou"`
	`}`
	`}`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSON_INFO_IN_LAB&lid=labid

Get listing of Labs in a Department	
Functionality	This method returns a JSON object that that describes only basic lab information for all labs that belong to a given department. The information described includes the lab id, lab location, short and long name.
Input	int departmentId

Output	JSON object
	Example:
	`{`
	`"labs": {`
	`"id_4": {`
	`"labName": "Digital Image and Signal Processing Laboratory",`
	`"labShortName": "Display"`
	`},`
	`...,`
	`"id_3": {`
	`"labName": "Software Technology And Network Applications Laboratory",`
	`"labShortName": "SoftNet"`
	`},`
	`"id_6": {`
	`"labName": "Telecom",`
	`"labShortName": "Telecom"`
	`}`
	`}`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_LABS_IN_DEPT &did=departmentId

Get listing of Persons in a Department	
Functionality	This method returns a JSON object that that describes only basic person information for all persons that belong to a given department. The information described includes the person id, lab location, short and long name .
Input	int departmentId

Output	JSON object
	Example:
	`{`
	`"persons": {`
	`"id_12": {`
	`"idUniversityPerson": "12",`
	`"firstName": "Nikolaos",`
	`"status": "Undergraduate",`
	`"lastName": "Agadakos"`
	`},...,`
	`"id_36": {`

```
            "idUniversityPerson": "36",
            "firstName": "Michalis",
            "status": "Professor",
            "lastName": "Zervakis"
        }
    }
}
```

http syntax

http://147.27.41.130:8080/jqCampus/controller?action= GET_PERSONS_IN_DEPT &did=departmentId

Get listing of Building options

Functionality

This method returns a JSON object that that describes Buildings of the University that have been mapped to a location on campus, and their name and code.

Input

Output

JSON object

Example:

```
{
    "buildings": [
        {
            "code": "A2",
            "name": "Kostis Nikiforakis Amfitheatre"
        },
        {
            "code": "B1",
            "name": "B amphitheatre complex"
        },
        ...,
        {
            "code": "M4",
            "name": "M4"
        },
        {
            "code": "M5",
            "name": "M5"
        }
    ]
}
```

http syntax

http://147.27.41.130:8080/jqCampus/controller?action= GET_BUILDING_OPTIONS

Get listing of locations inside a Building

Functionality

This method returns a JSON object that that describes indoor locations of a given building with their id, code and functionality.

Input

String buildingId

Output

JSON object

Example:

```
{
    "buildingFeatureOptions": [
        {
            "locationId": "1",
            "locationCode": "141.A21",
            "locationAttribute": "office"
        },
        ...,
        {
            "locationId": "173",
            "locationCode": "145.P59",
            "locationAttribute": "lab"
```

	`}`
	`]`
	`}`

http syntax	http://147.27.41.130:8080/jqCampus/controller?action= GET_BUILDING_FEATURE_OPTIONS&bid=buildingId

Chapter 7 - User Interfaces

7.1 Introduction

In this chapter we describe the basic principles we followed in designing the User Interfaces. The most important of all was remembering to design while keeping user context in mind, and optimizing for small screens.

7.2 User Interfaces

The primary medium we are targeting is the mobile smart phone. Therefore, it is essential we take into account the limited screen real estate, and the fact that user input is based on touch interactions. That being established, we also want the interface to adapt gracefully to all screen sizes, and of course accept traditional input such as mouse and keyboard.

To achieve the above, we have made use of the jQuery Mobile JavaScript library, which provides a unified, HTML5-based user interface system. The library offers touch optimized User Interface components, such as buttons, form elements, lists, menus, toolbars, pages, and dialogs. In addition, it provides an extensive API for event handling and component configuration. Finally, it provides a theme framework, which takes advantage of CSS3 properties, allowing the theme file to be very lightweight and reducing server requests. [http://jquerymobile.com]

One problem a designer has to face when designing for small screens is that less is more. Designers must show on screen the least possible information and user interface components, focus on the user goal they are trying to enable.

Another issue is designing for context. The three major mobile contexts that need be considered are bored, busy and lost.

- **Bored:** there are a lot of people using their smart phones on the couch at home. In this context, the user has a more immersive experience, with a longer user session. Interruptions are still highly likely, and the app must be able to pick up where the user left off. This context applies to users who may be using the application from their home or office, browsing the campus map.
- **Busy: "**running through the airport" scenario. Users are trying to accomplish tasks quickly and reliably with one hand in a hectic environment. It is critical to accommodate the user with huge targets and bold design. This context applies to our users trying to use the person directory for example, or looking for a location on the map. They have a very specific task in mind and are in a hurry to complete it. (get to the amphitheatre where an exam is being held, email or call a colleague etc.)

- **Lost:** Users in transit, in unfamiliar surroundings, or in familiar surroundings, but interested in something unknown. Sketchy connectivity and battery life are big concerns in this context, and a level of offline support should be offered, with sparing use of geolocation and other battery hogs. Users that this context applies to are visitors of the University or even locals to the university, looking to locate themselves within the campus or a building, or browsing directory information.

[Jonathan Stark, April 16, 2012 http://www.netmagazine.com/features/10-principles-mobile-interface-design]

General Guidelines followed in TUC CAMPUS

Navigation: Users have two main concerns when navigating a web application: returning to the page they were, or returning back to base. We provide back and home buttons on all content displaying pages, providing the user with those two options. We find a page title displaying the function of a page is more important that displaying the full navigation, which can be replicated with very few extra interactions

Polish: the personal nature of smart phones means users appreciate the details. No matter how powerful the application, user experience degrades when application interfaces are not delivered with an aesthetic approach.

Control: controls are added at the bottom or the top of page content, preventing slip errors while scrolling through information and separating content from major user actions.

User Control: Users do not like it when an application takes decisions on their part. We allow offline caching of persistent application data, but only when the user requires it.

Visual: The theme we use has contrast, making elements stand out. Colors are definitive of content and actions. Blue is for selectable content that results in navigation, red is used for login/logout and important user confirmations. Yellow is for external links and undoing an action. Green is for success messages and confirming form submissions.

The structure of the user interface.

Figures 31, 32, and 33 display the general directory of the application in the opening page, as well as the route and information provided to the user for accessing the university from different points of access (airport, harbor, city of Chania, highway). On Figure 31, Login button is at the top right of the page, where user is accustomed to finding it, and is a red button, following the custom color for logon/logout buttons. Menu is blue, with explanatory icons. The link to the University home page is in yellow (as all external links in our application)

Figures 34, 35 and 36 display the interface provided to the user for browsing information on top of the campus, and finding information about buildings, parking lots, etc. the user can ask for detailed information about a building and such information will be provided to him.

Figures 36 to 41 display the interface provided to the user for browsing and viewing directory information.

Figure 42 displays the interface provided to the user for location a place of interest on campus and in a building.

Figure 43 and 44 display the interface for logging in and welcoming the user.

Figures 45 to 54 display the interface provided to the user for creating browsing and viewing events.

Figure 55 displays the user interface provided to the user for editing his profile and subscription preferences.

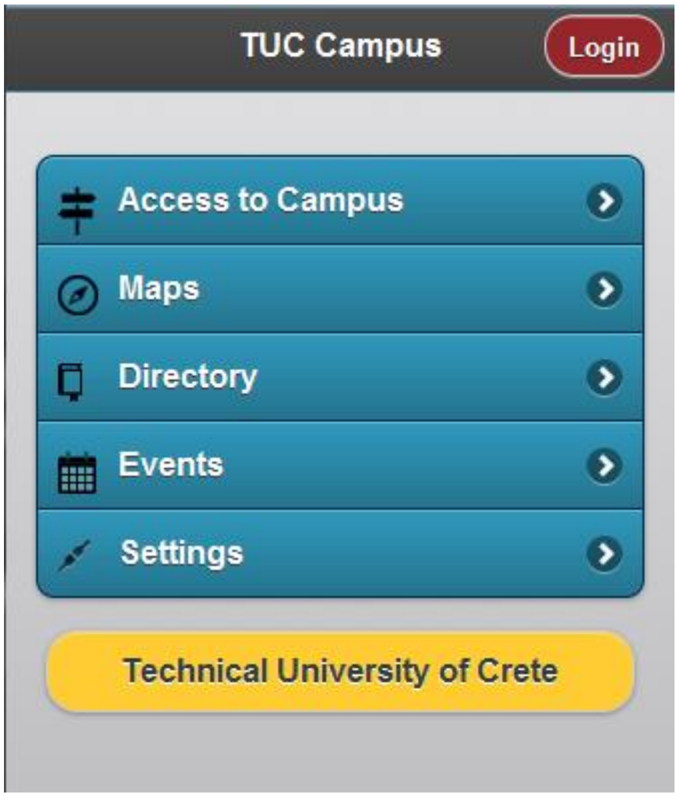
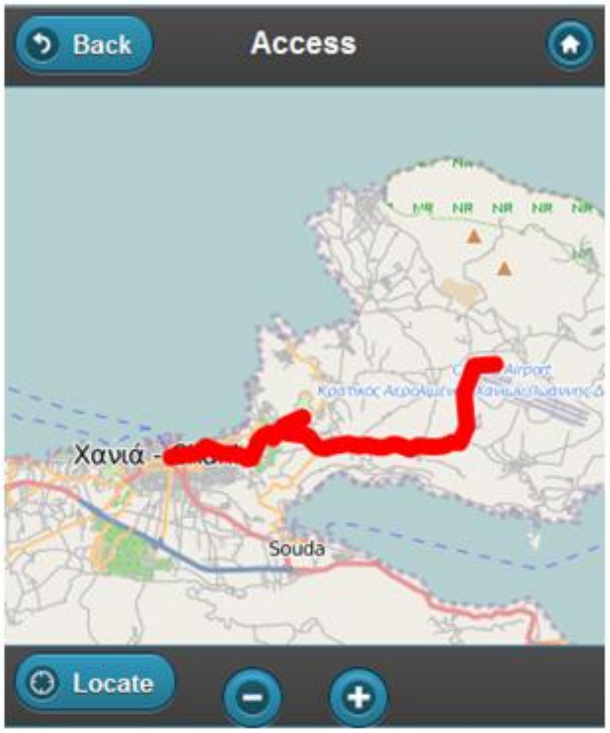


Figure 30 Application home menu.



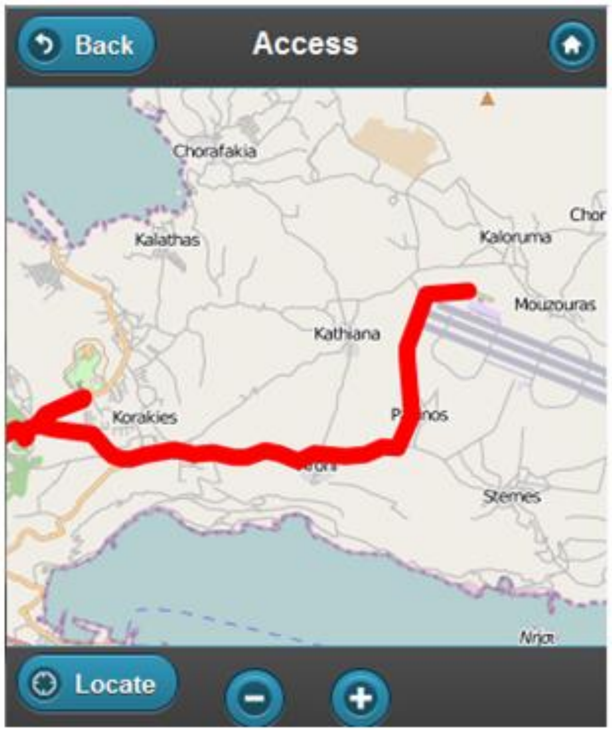
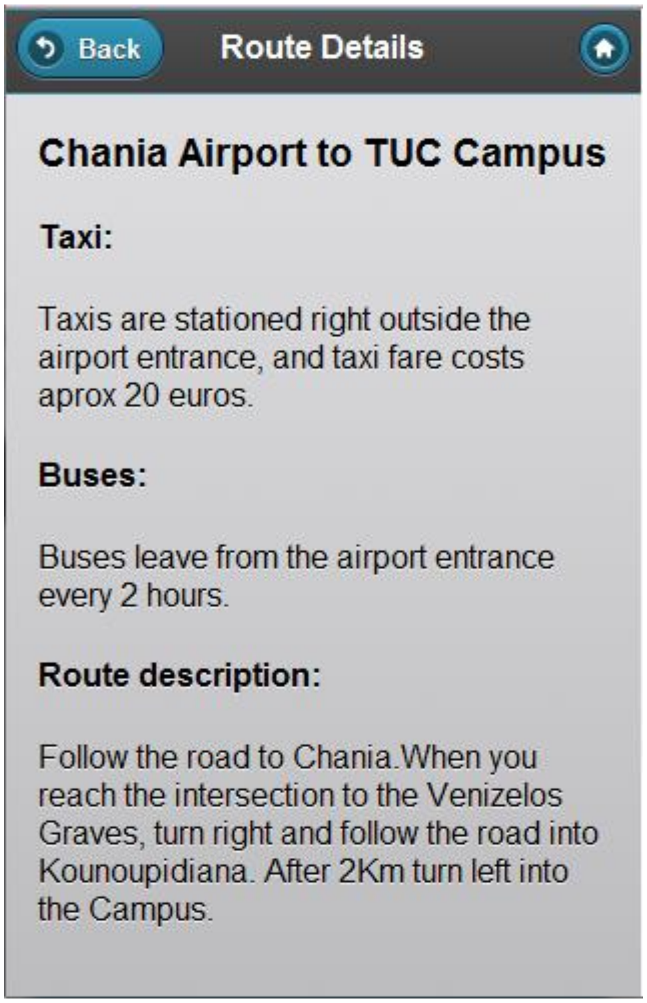


Figure 31 (left) Access to Campus displays map of the area, with interactive lines to display routes, thick enough to react to touch events. Footer of the page has buttons to zoom in and out, and a locate button that triggers the Geolocation function. (right) The same screen zoomed in.



Chania Airport to TUC Campus

Taxi:

Taxis are stationed right outside the airport entrance, and taxi fare costs aprox 20 euros.

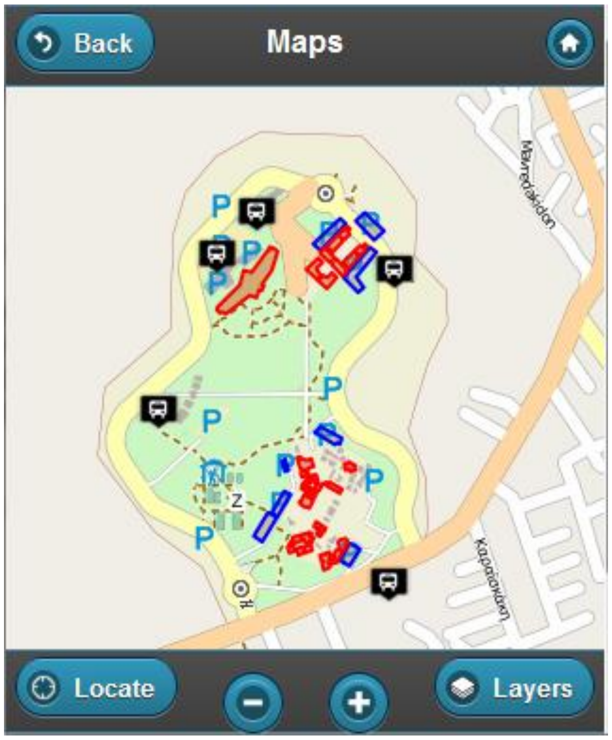
Buses:

Buses leave from the airport entrance every 2 hours.

Route description:

Follow the road to Chania.When you reach the intersection to the Venizelos Graves, turn right and follow the road into Kounoupidiana. After 2Km turn left into the Campus.

Figure 32 Route details that display when user selects an interactive line



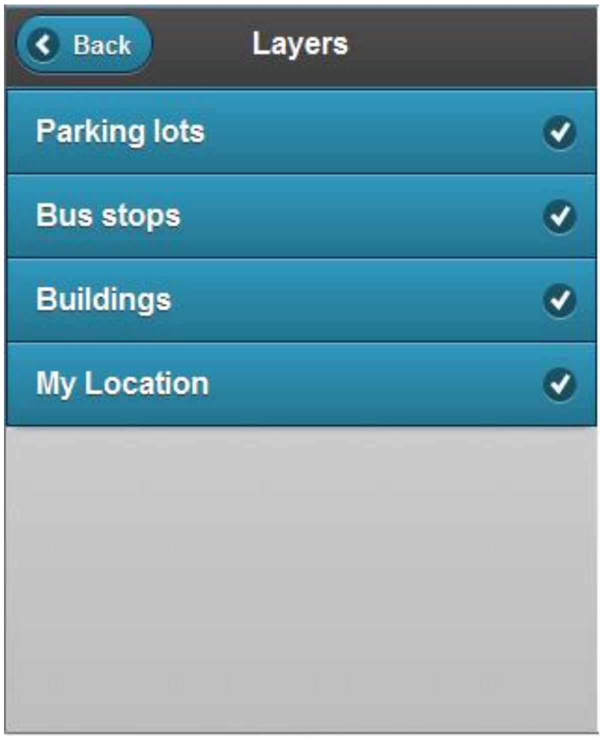


Figure 33 Campus map at default zoom. Map displays bus stops, parking lots and buildings. Geolocation is enabled by clicking Locate button, Map Layers are made active/inactive from Layer menu

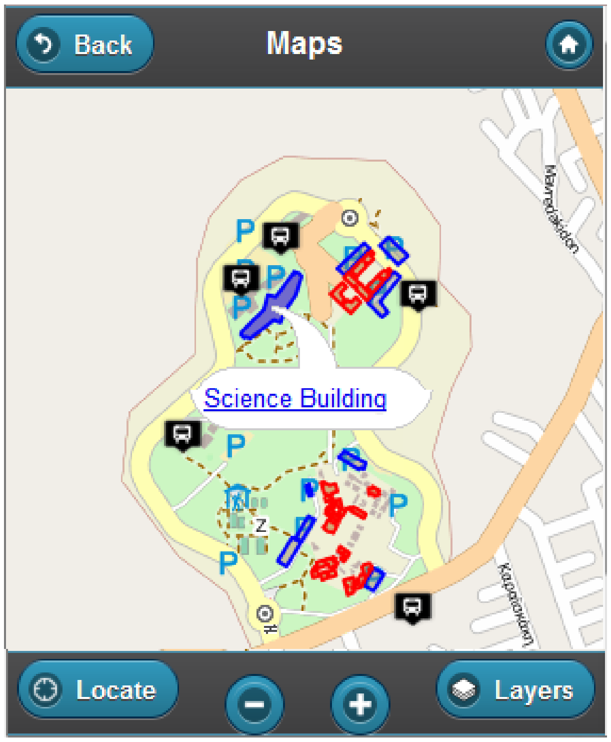
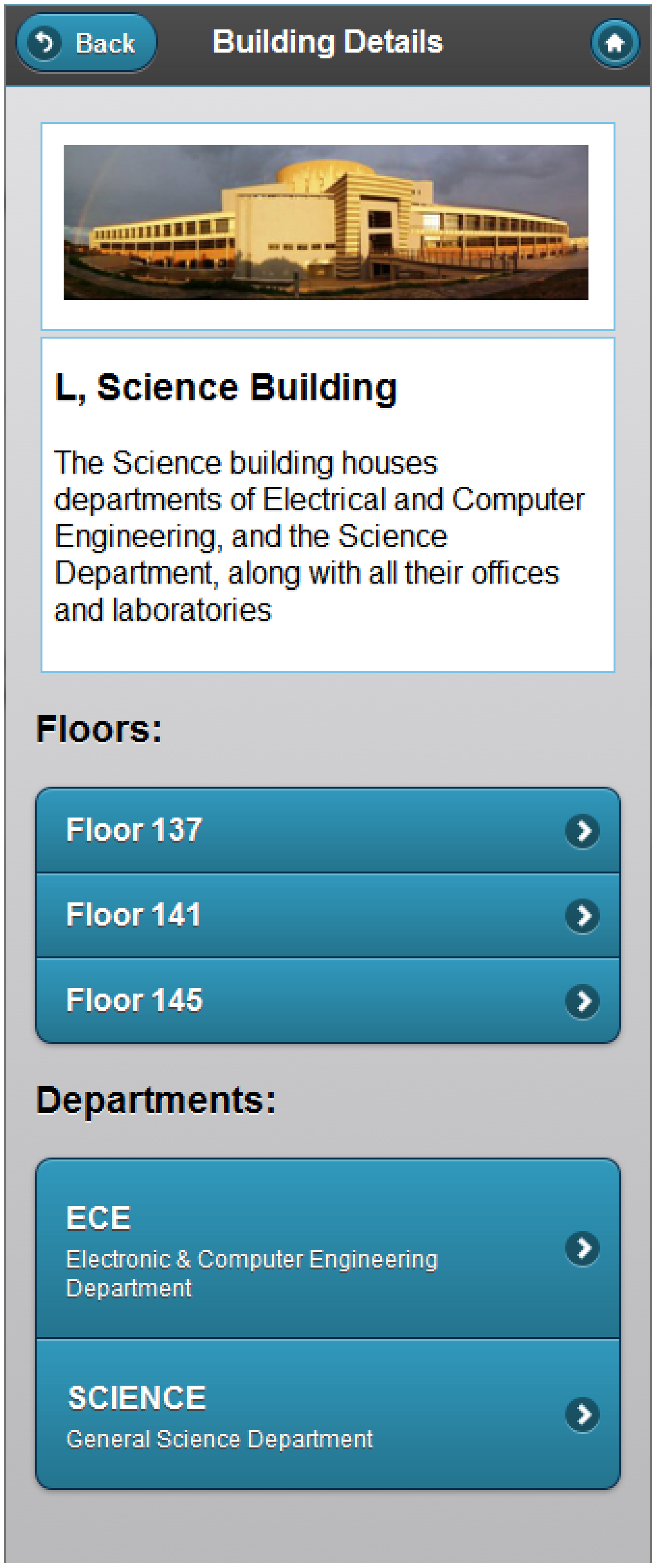


Figure 34 Map features are interactive, and display informative popup on selection.



L, Science Building

The Science building houses departments of Electrical and Computer Engineering, and the Science Department, along with all their offices and laboratories

Floors:

Floor 137

Floor 141

Floor 145

Departments:

ECE
Electronic & Computer Engineering Department

SCIENCE
General Science Department

Figure 35 Building Details Page. User arrives here, either from selecting the link on campus map, or from the building listing in the directory

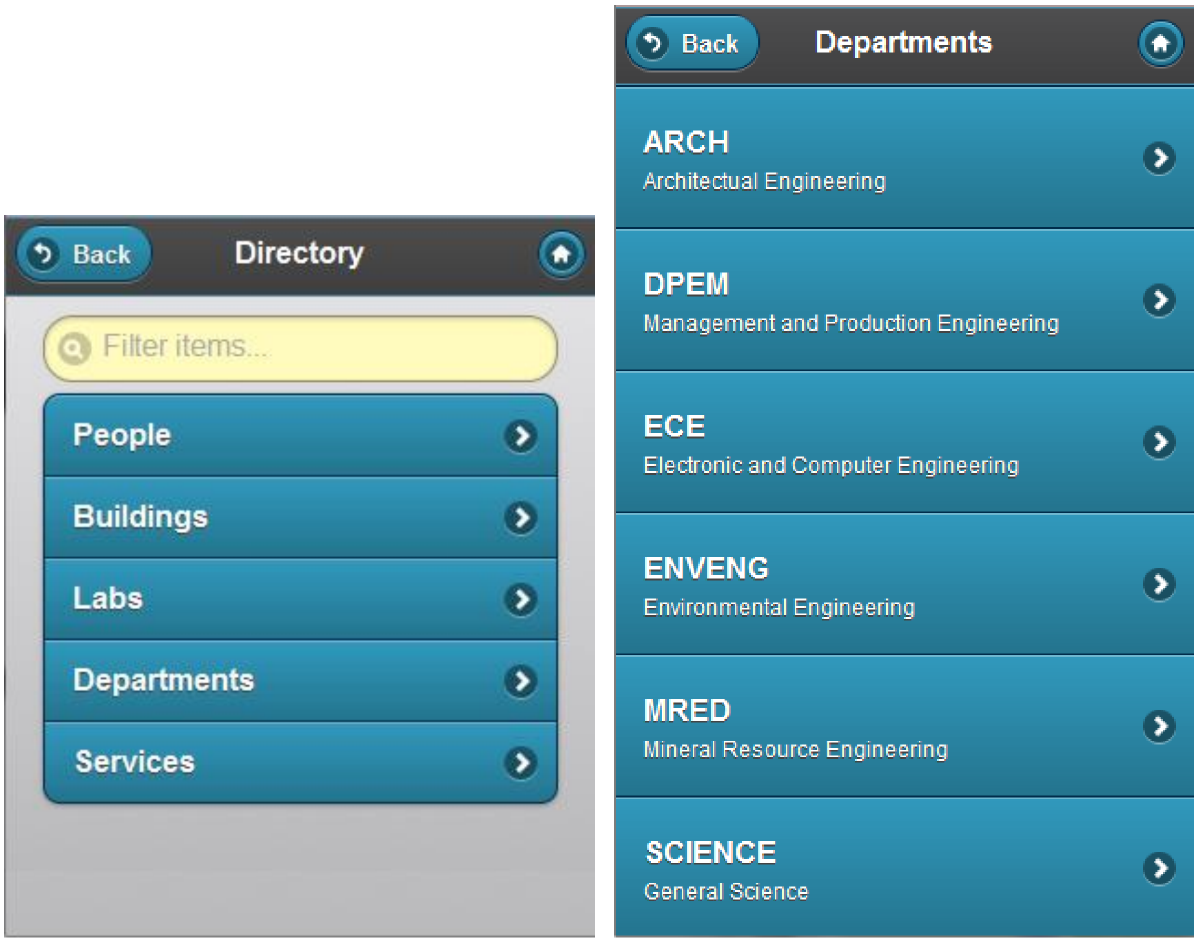
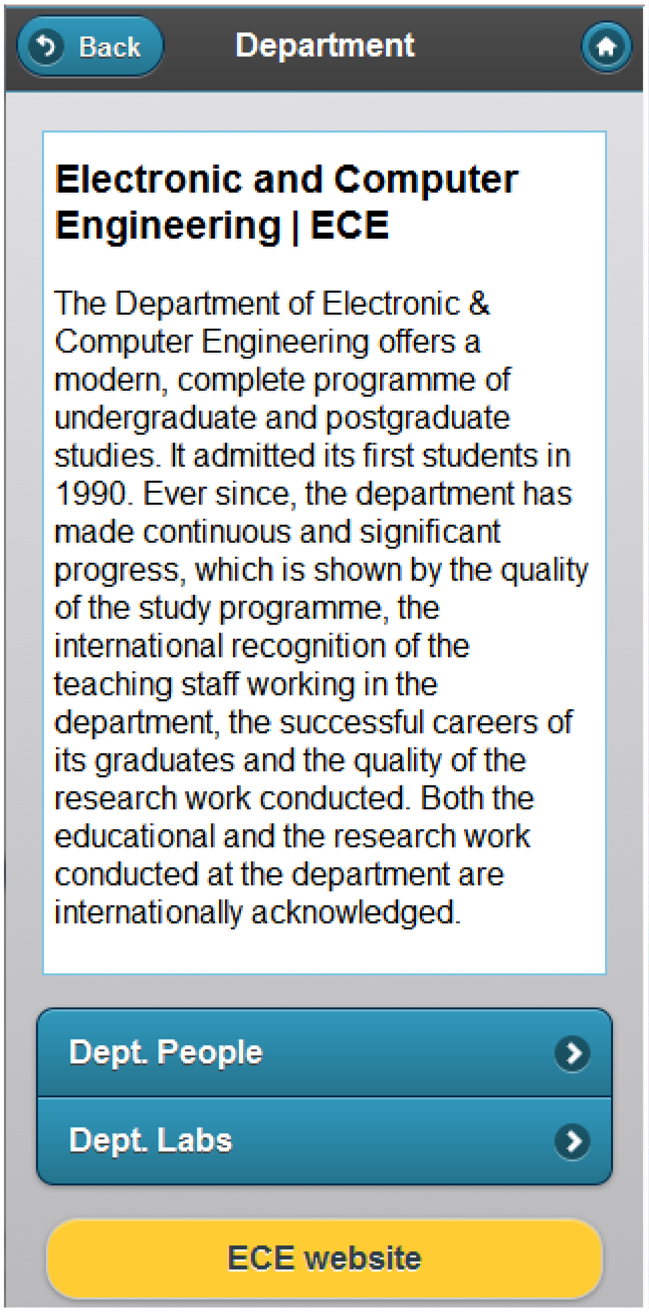
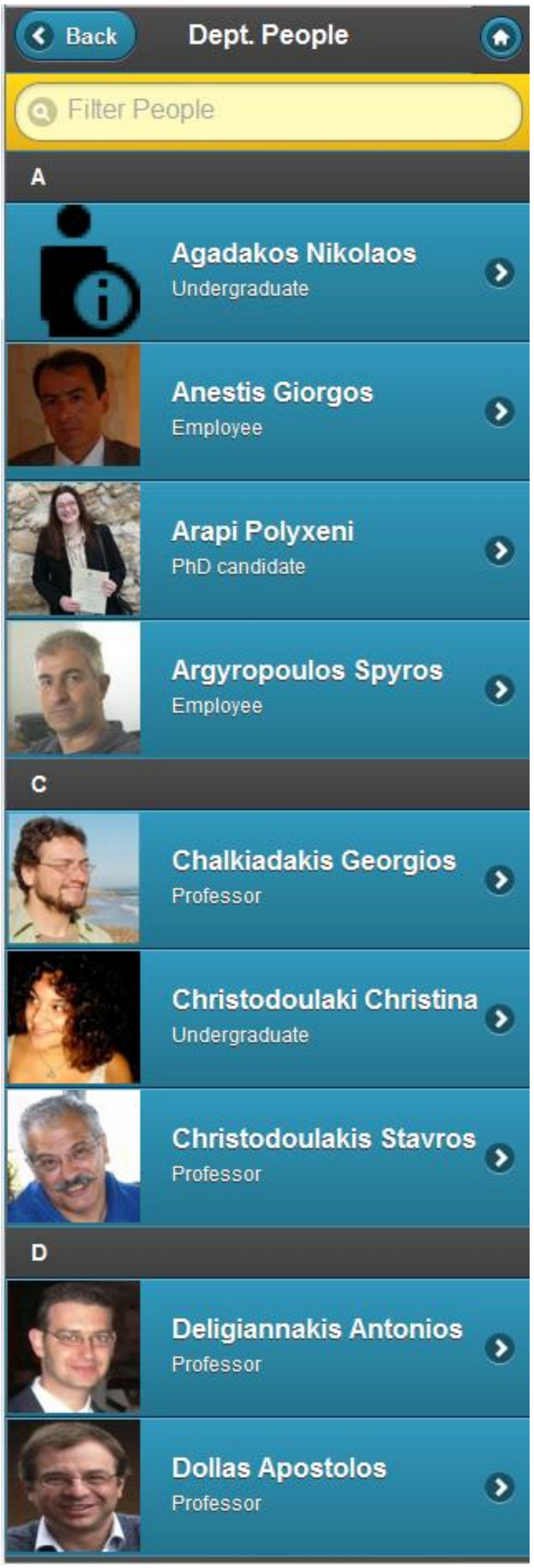


Figure 36 Directory listing, and Departments list



Electronic and Computer Engineering | ECE

The Department of Electronic & Computer Engineering offers a modern, complete programme of undergraduate and postgraduate studies. It admitted its first students in 1990. Ever since, the department has made continuous and significant progress, which is shown by the quality of the study programme, the international recognition of the teaching staff working in the department, the successful careers of its graduates and the quality of the research work conducted. Both the educational and the research work conducted at the department are internationally acknowledged.



A

Agadakos Nikolaos
Undergraduate

Anestis Giorgos
Employee

Arapi Polyxeni
PhD candidate

Argyropoulos Spyros
Employee

C

Chalkiadakis Georgios
Professor

Christodoulaki Christina
Undergraduate

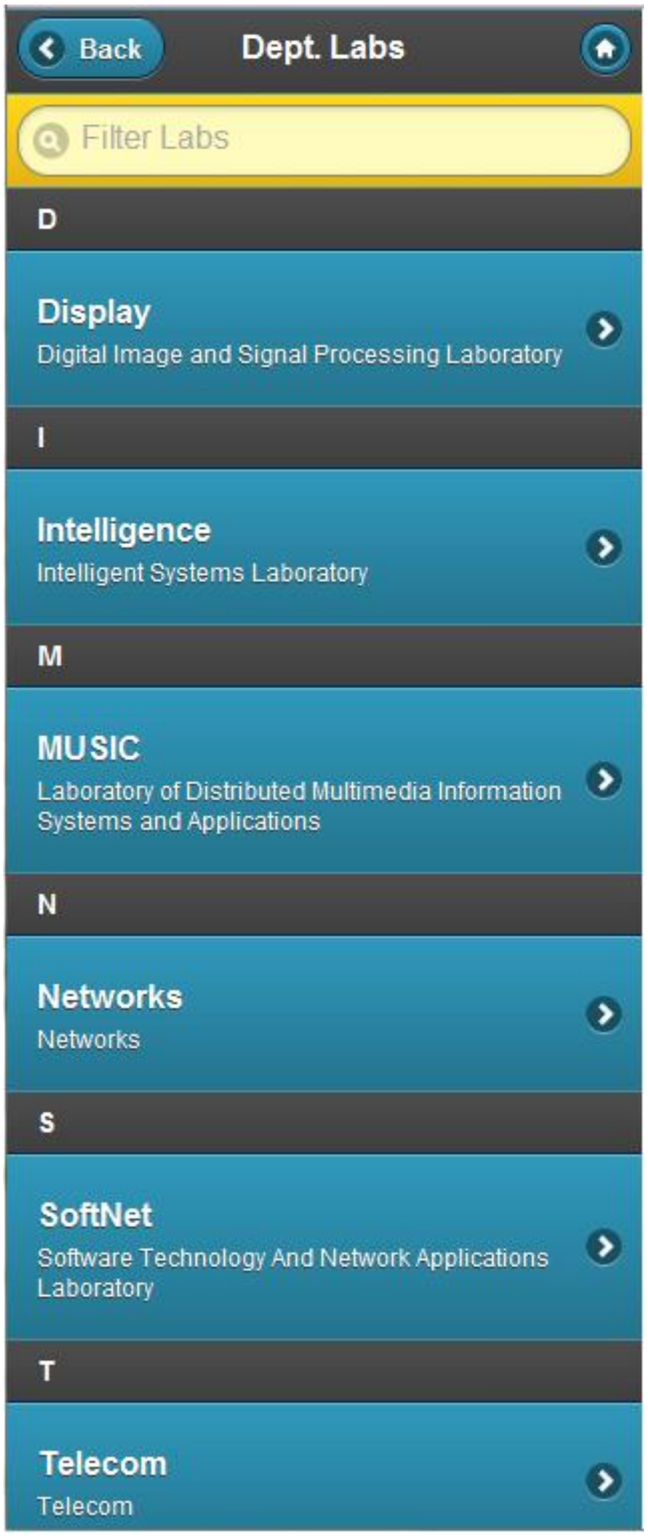
Christodoulakis Stavros
Professor

D

Deligiannakis Antonios
Professor

Dollas Apostolos
Professor

Figure 37 list of People in Department



D

Display
Digital Image and Signal Processing Laboratory

I

Intelligence
Intelligent Systems Laboratory

M

MUSIC
Laboratory of Distributed Multimedia Information Systems and Applications

N

Networks
Networks

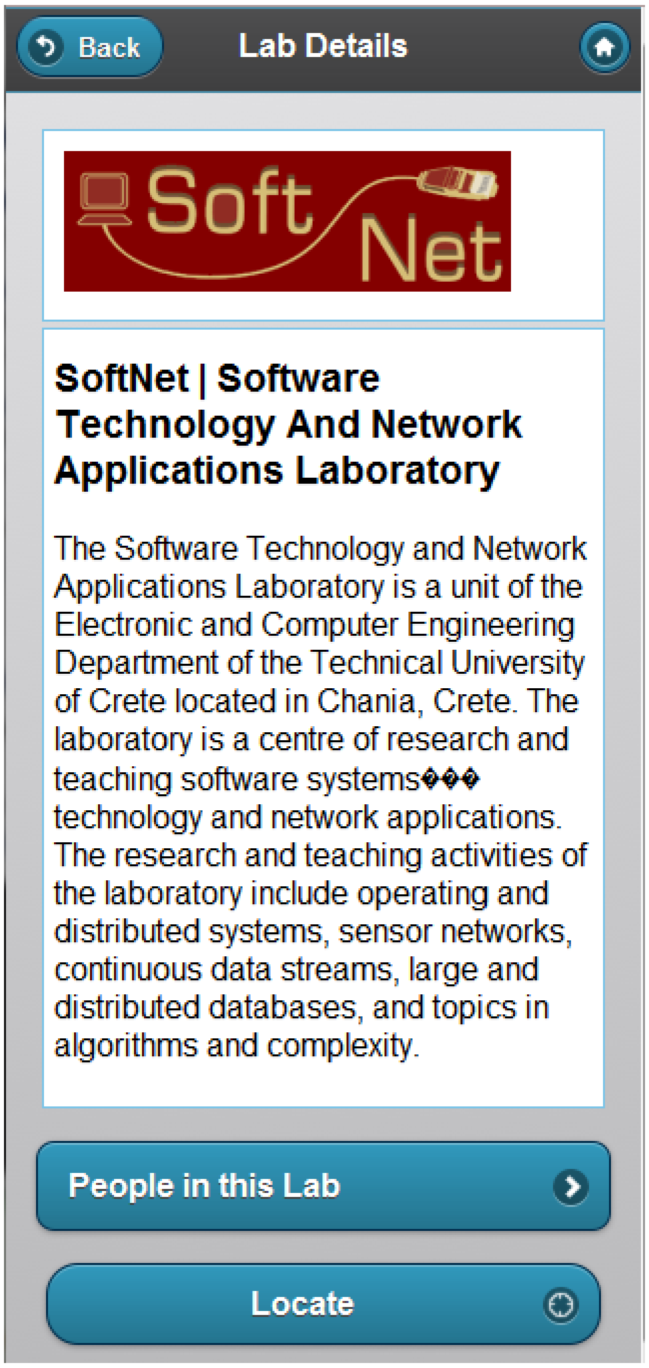
S

SoftNet
Software Technology And Network Applications Laboratory

T

Telecom
Telecom

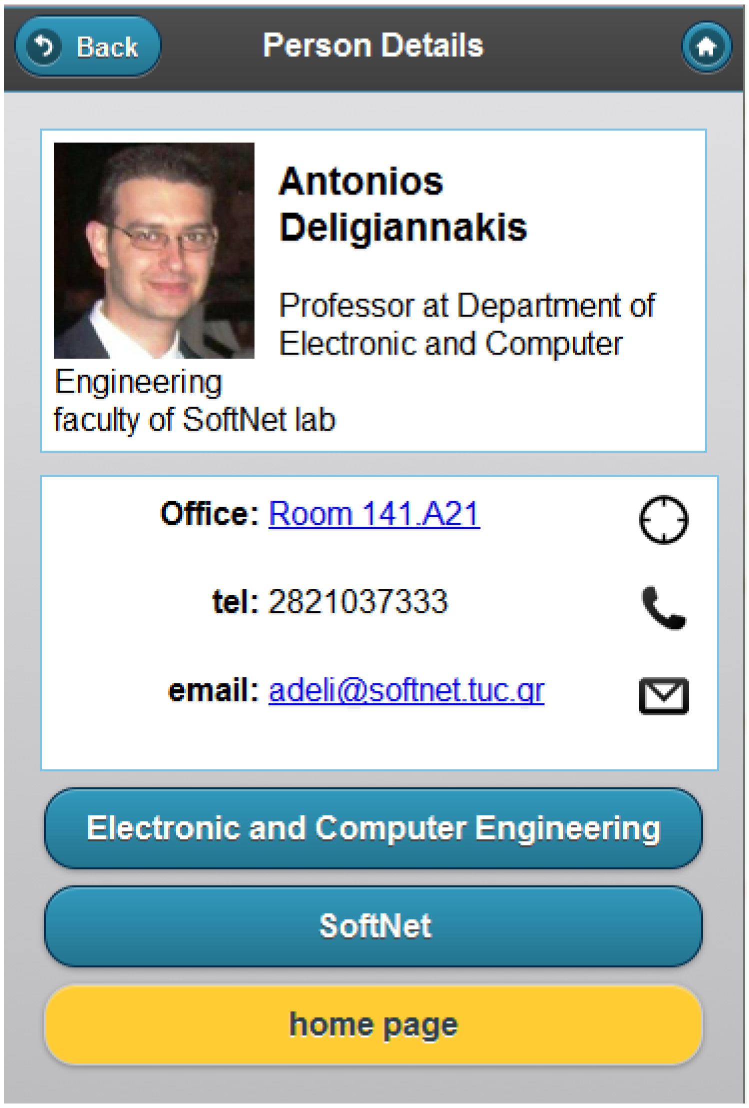
Figure 38 Lab Listing in Department



SoftNet | Software Technology And Network Applications Laboratory

The Software Technology and Network Applications Laboratory is a unit of the Electronic and Computer Engineering Department of the Technical University of Crete located in Chania, Crete. The laboratory is a centre of research and teaching software systems��� technology and network applications. The research and teaching activities of the laboratory include operating and distributed systems, sensor networks, continuous data streams, large and distributed databases, and topics in algorithms and complexity.

Figure 39 Lab details



Antonios Deligiannakis

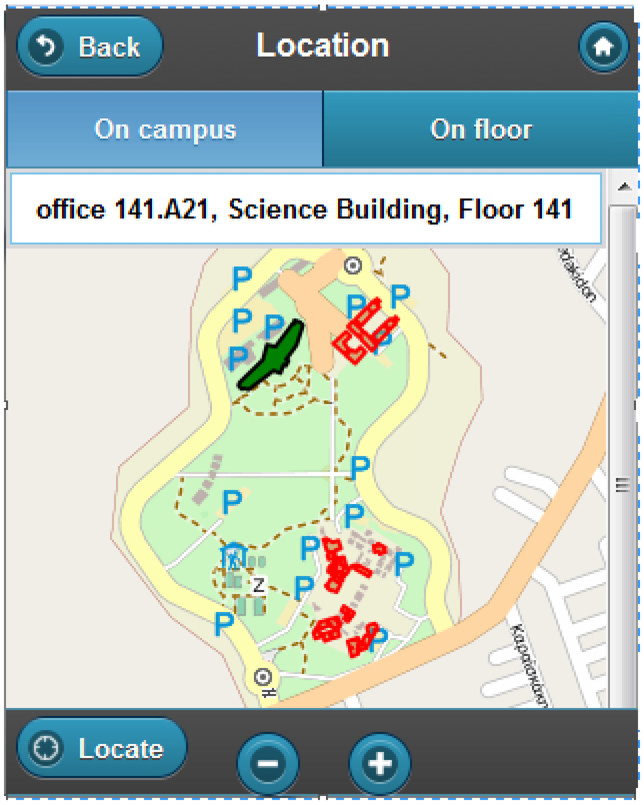
Professor at Department of Electronic and Computer Engineering
faculty of SoftNet lab

Office: Room 141.A21

tel: 2821037333

email: adeli@softnet.tuc.gr

Figure 40 Person details page. Displays user photograph, and function in the University. Also access information (office, telephone and email).



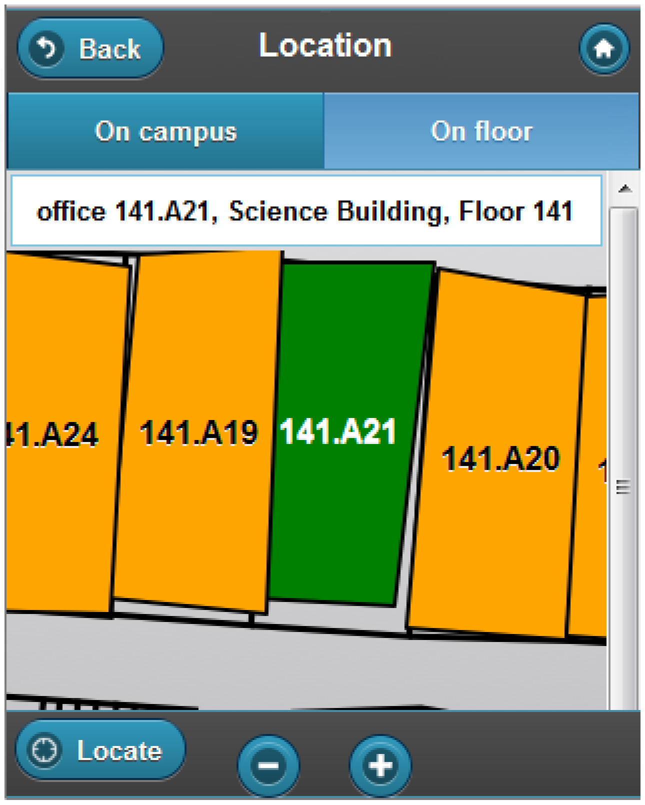
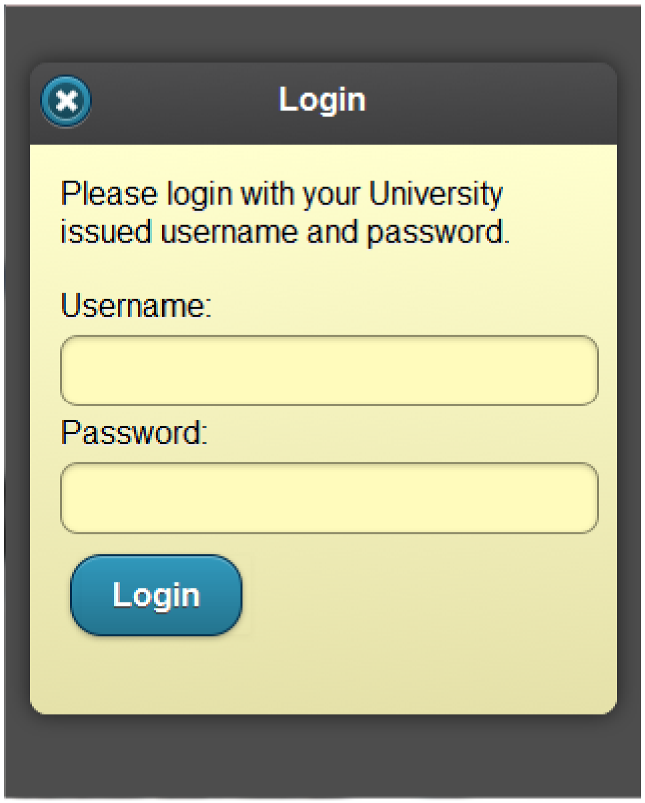


Figure 41 Locate a place of interest on campus, displays a campus map and a floor map view. Maps are zoomed in on the feature, and feature is distinct from other surrounding



Please login with your University issued username and password.

Username:

Password:

Figure 42 Login dialog

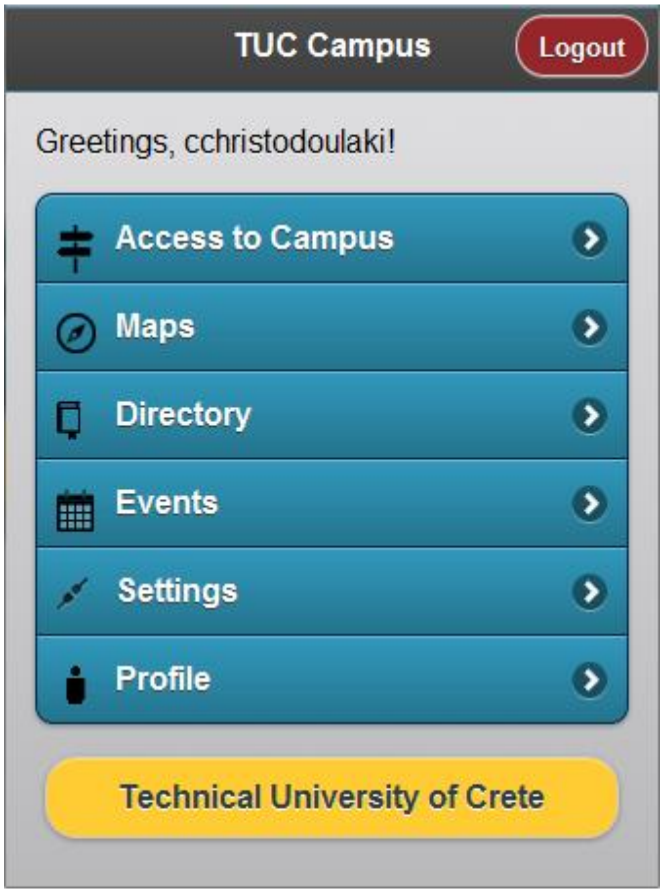
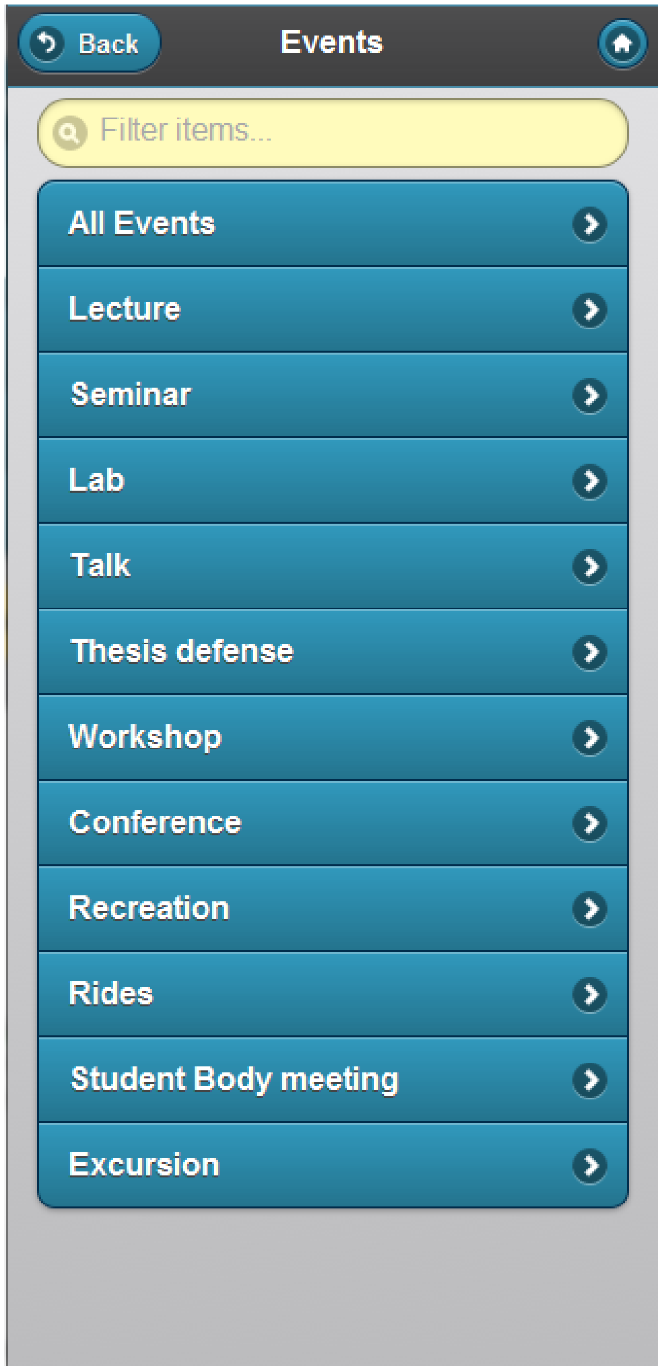


Figure 43 Home screen after login



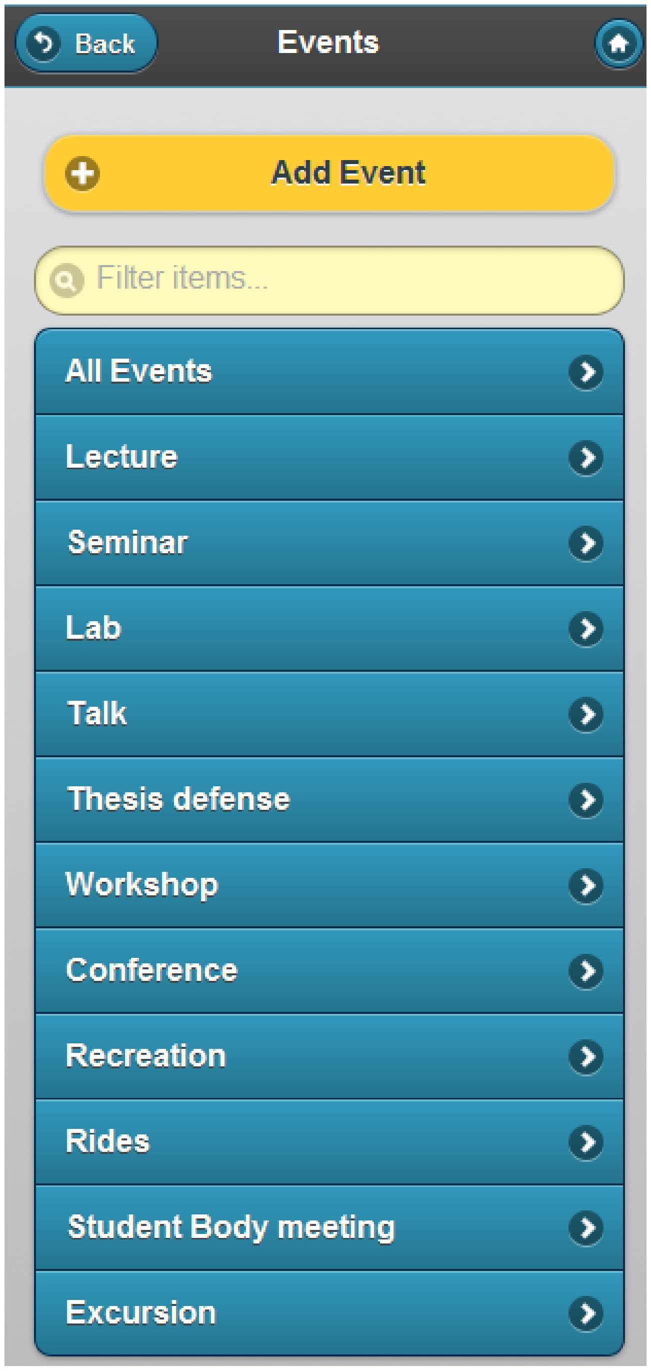


Figure 44 Events page, before and after login

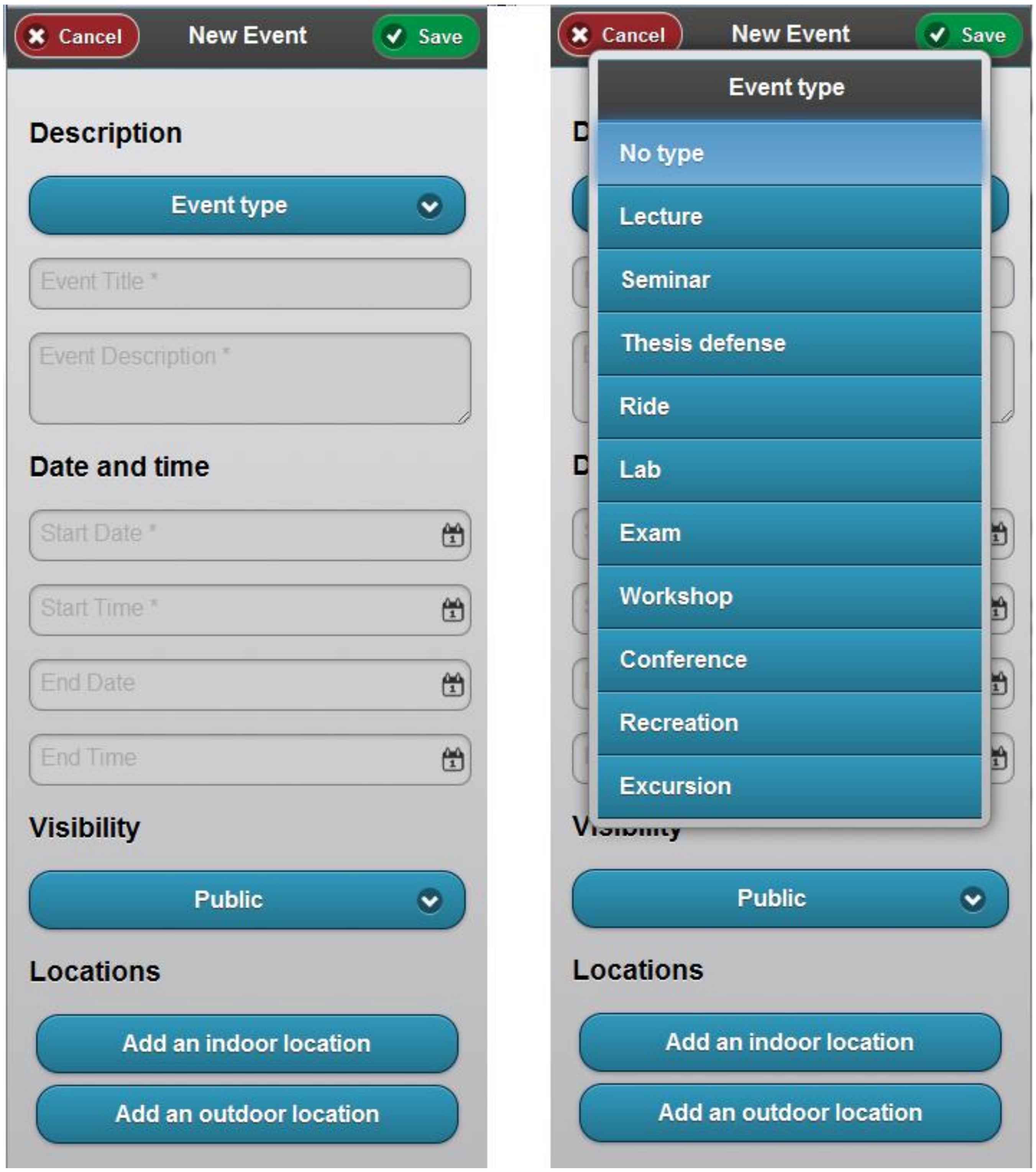
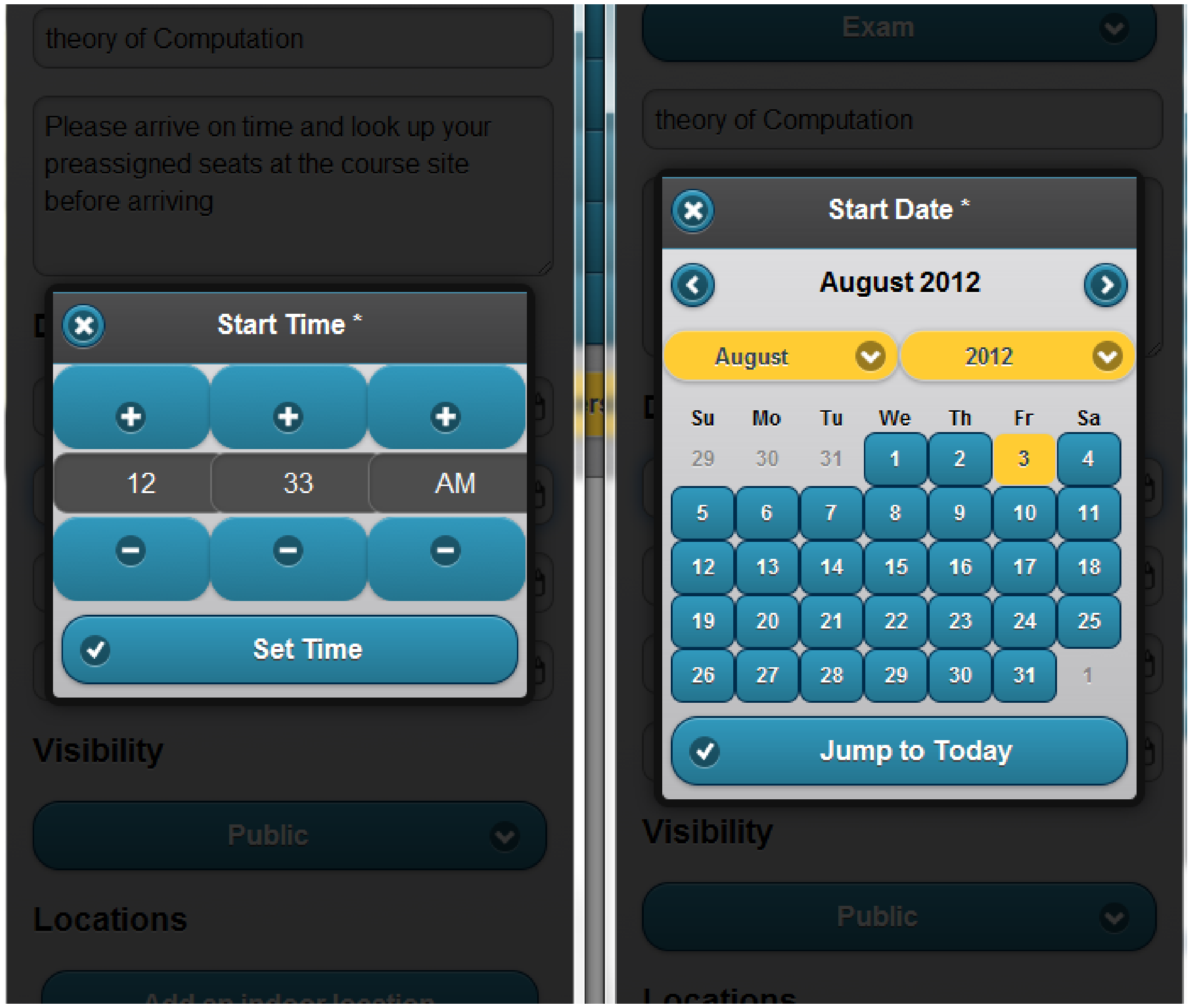


Figure 45 New Event form



theory of Computation

Please arrive on time and look up your preassigned seats at the course site before arriving

12	33	AM

Visibility

Public

Locations

Exam

theory of Computation

August 2012

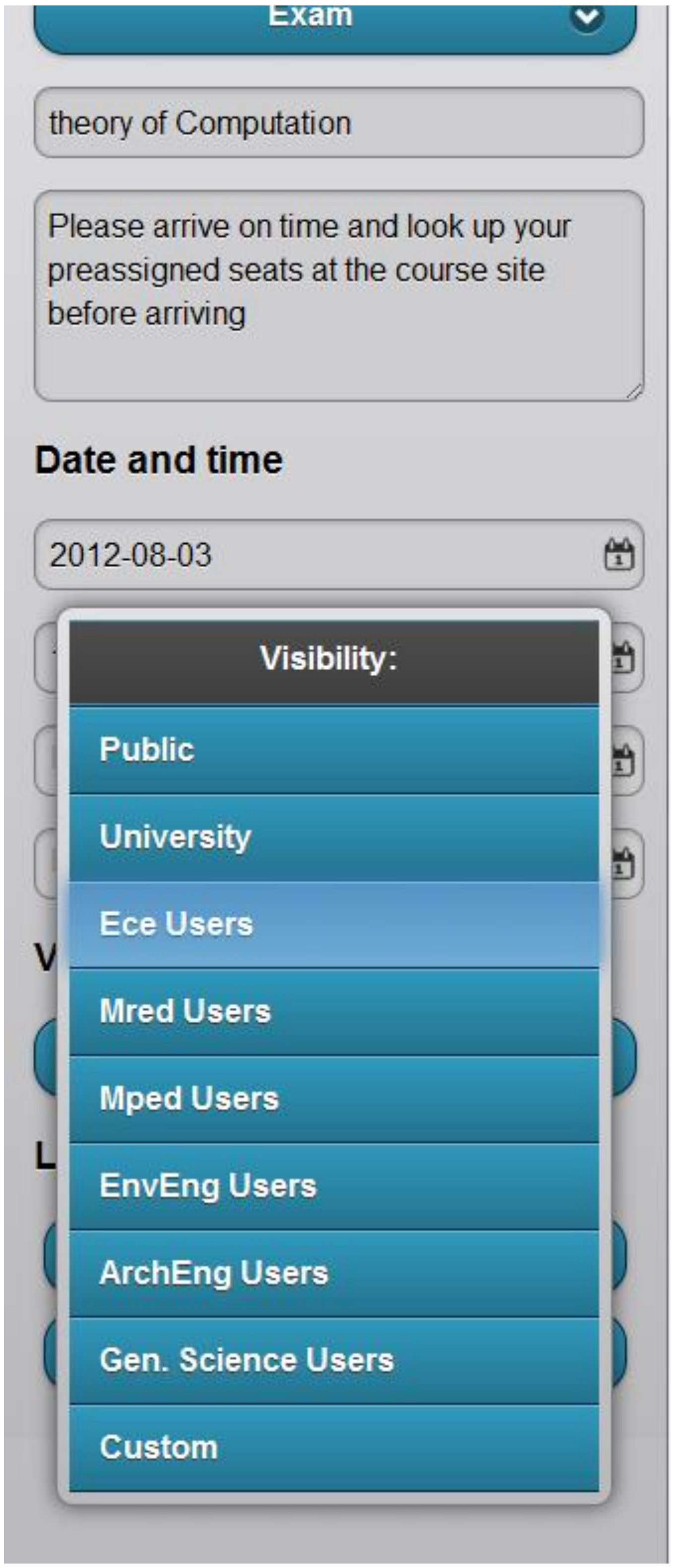
August 2012

Su	Mo	Tu	We	Th	Fr	Sa
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

Visibility

Public

Figure 46 New Event form (cont'd)



Exam

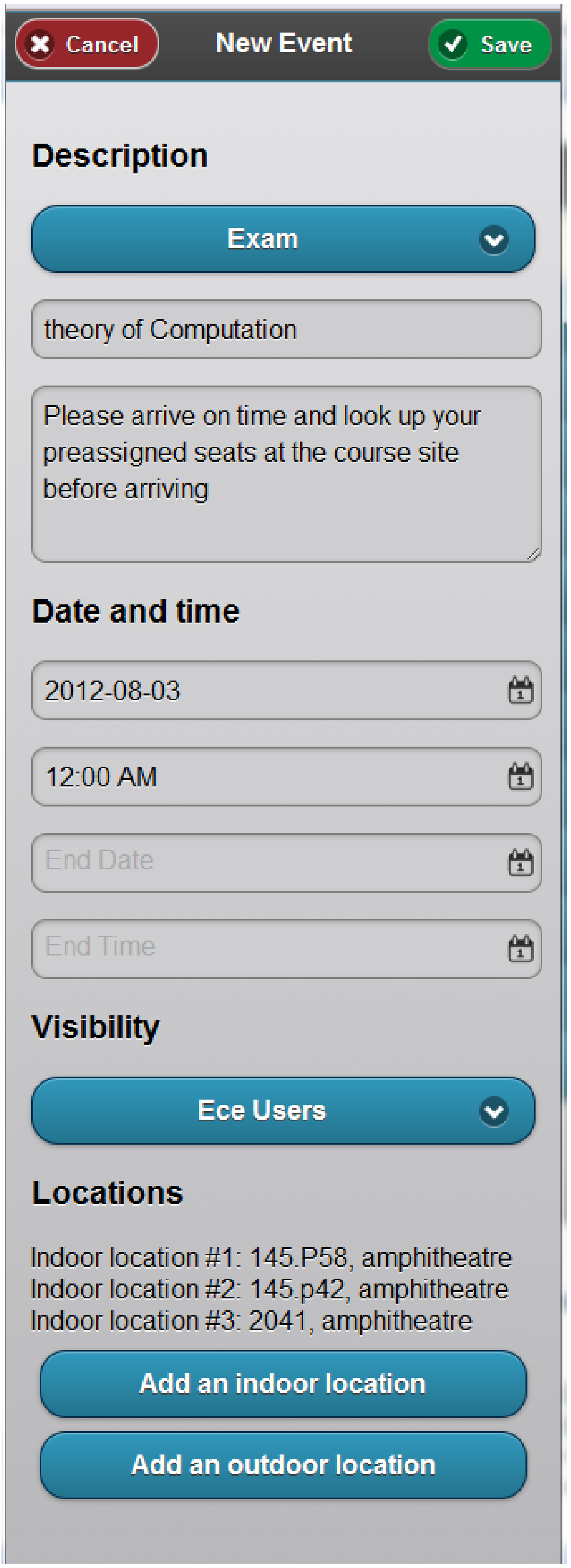
theory of Computation

Please arrive on time and look up your preassigned seats at the course site before arriving

Date and time

2012-08-03

Figure 47 New Event form (cont'd)



Description

Exam

theory of Computation

Please arrive on time and look up your preassigned seats at the course site before arriving

Date and time

2012-08-03

12:00 AM

End Date

End Time

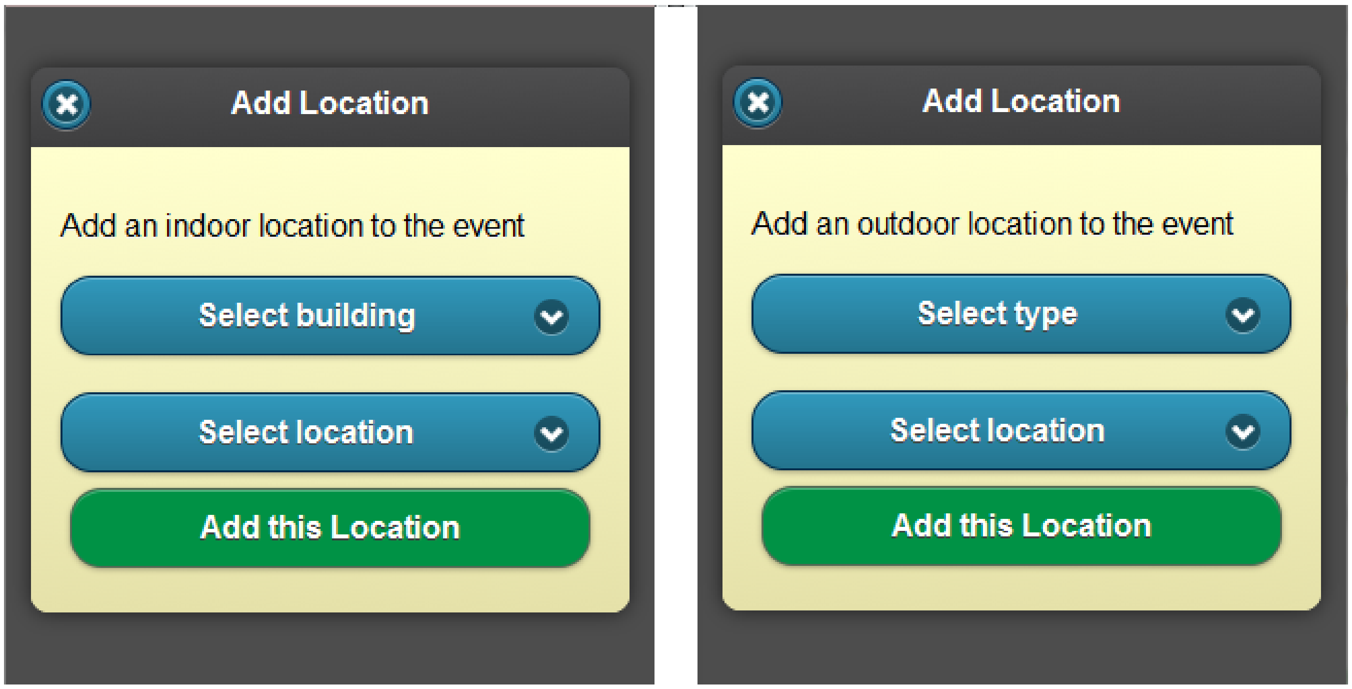
Visibility

Ece Users

Locations

Indoor location #1: 145.P58, amphitheatre
Indoor location #2: 145.p42, amphitheatre
Indoor location #3: 2041, amphitheatre

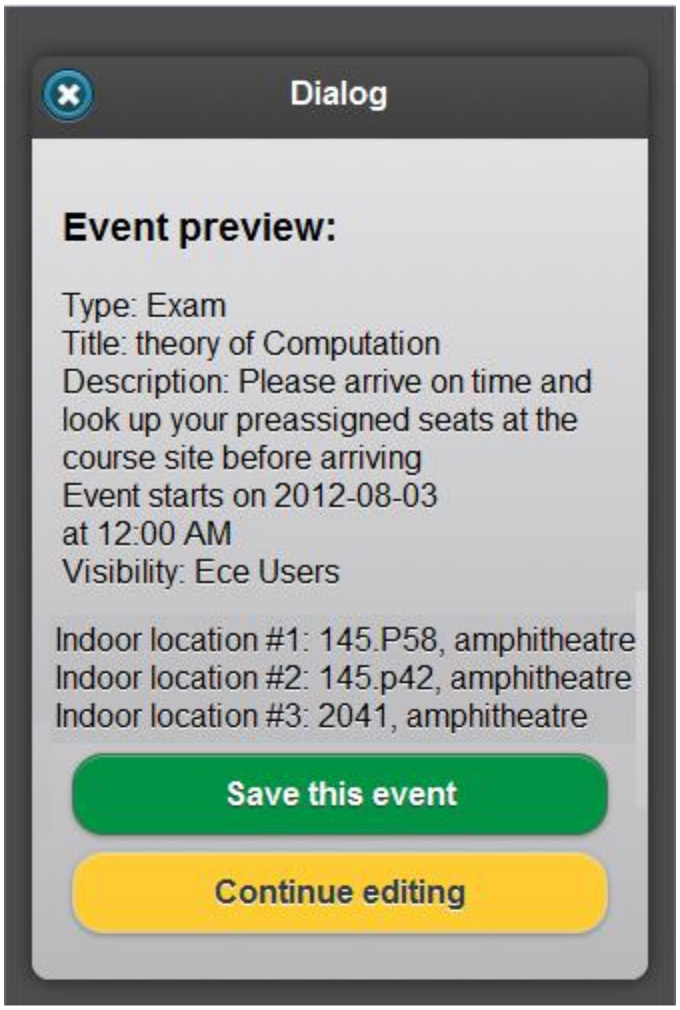
Figure 48 New Event form (cont'd)



Add an indoor location to the event

Add an outdoor location to the event

Figure 49 Add indoor and outdoor locations to event

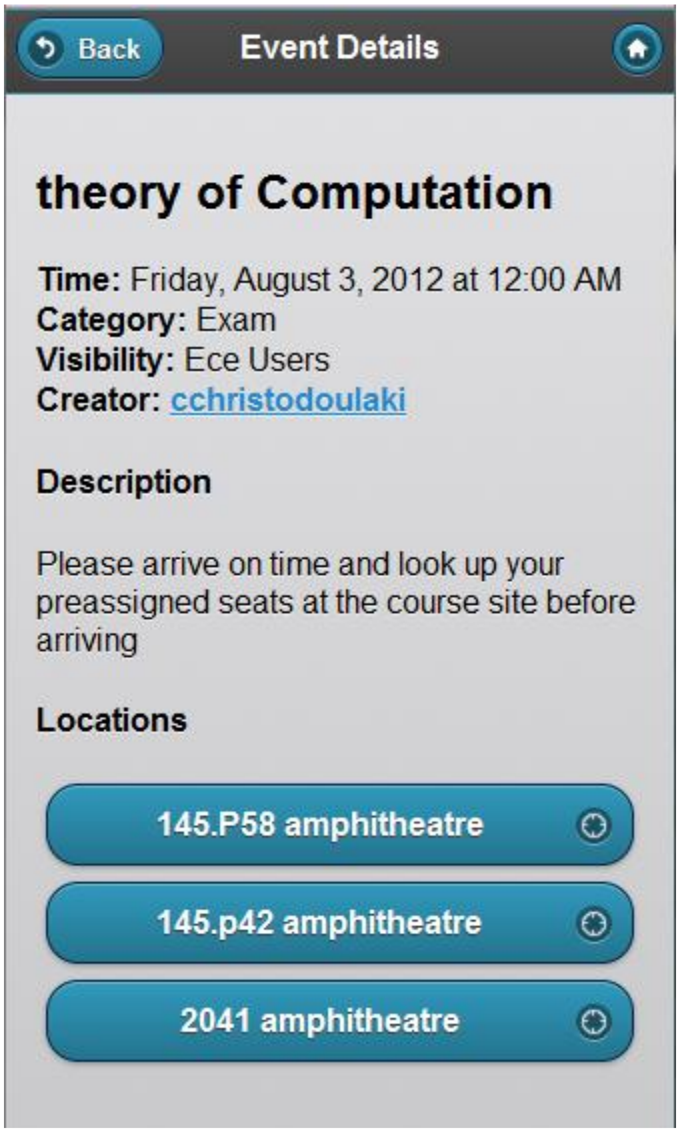


Event preview:

Type: Exam
Title: theory of Computation
Description: Please arrive on time and look up your preassigned seats at the course site before arriving
Event starts on 2012-08-03
at 12:00 AM
Visibility: Ece Users

Indoor location #1: 145.P58, amphitheatre
Indoor location #2: 145.p42, amphitheatre
Indoor location #3: 2041, amphitheatre

Figure 50 new event dialog asking for submit confirmation



theory of Computation

Time: Friday, August 3, 2012 at 12:00 AM
Category: Exam
Visibility: Ece Users
Creator: cchristodoulaki

Description

Please arrive on time and look up your preassigned seats at the course site before arriving

Locations

Figure 51 Event Details

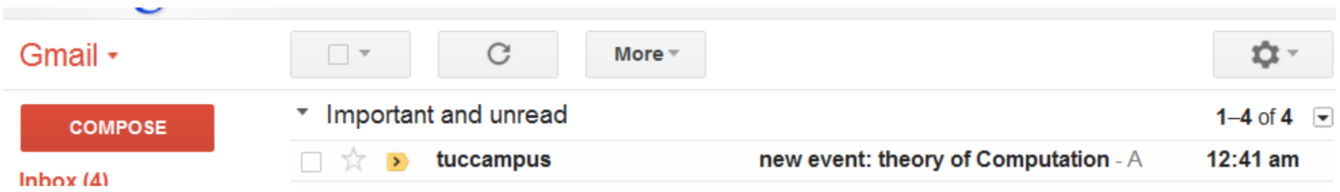
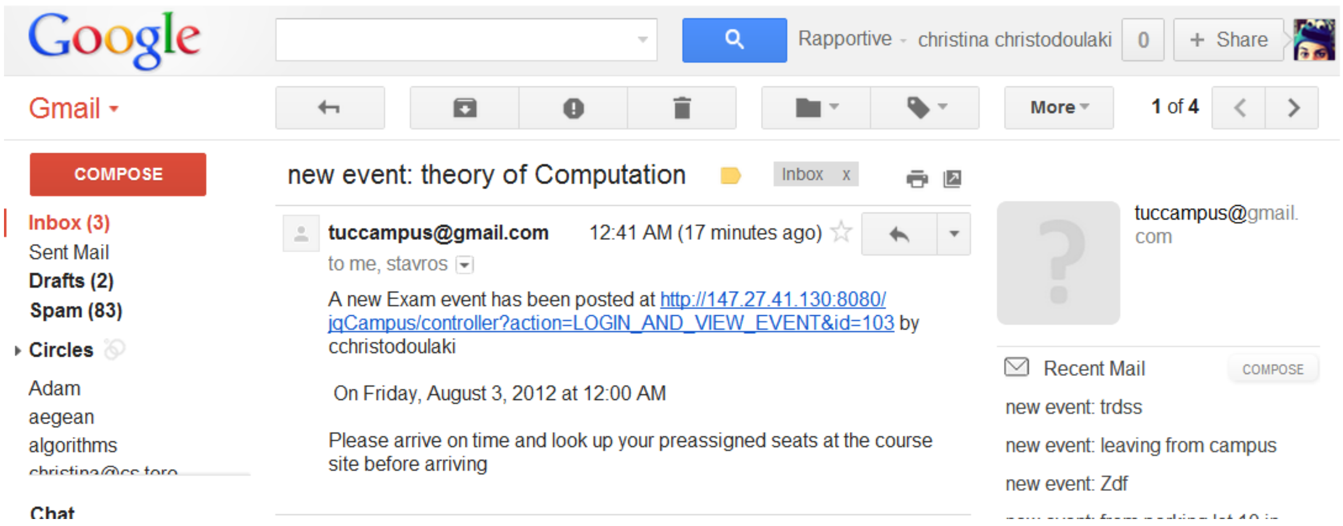


Figure 52 new event notification is sent to subscribers with visibility privileges



new event: theory of Computation

tuccampus@gmail.com 12:41 AM (17 minutes ago)

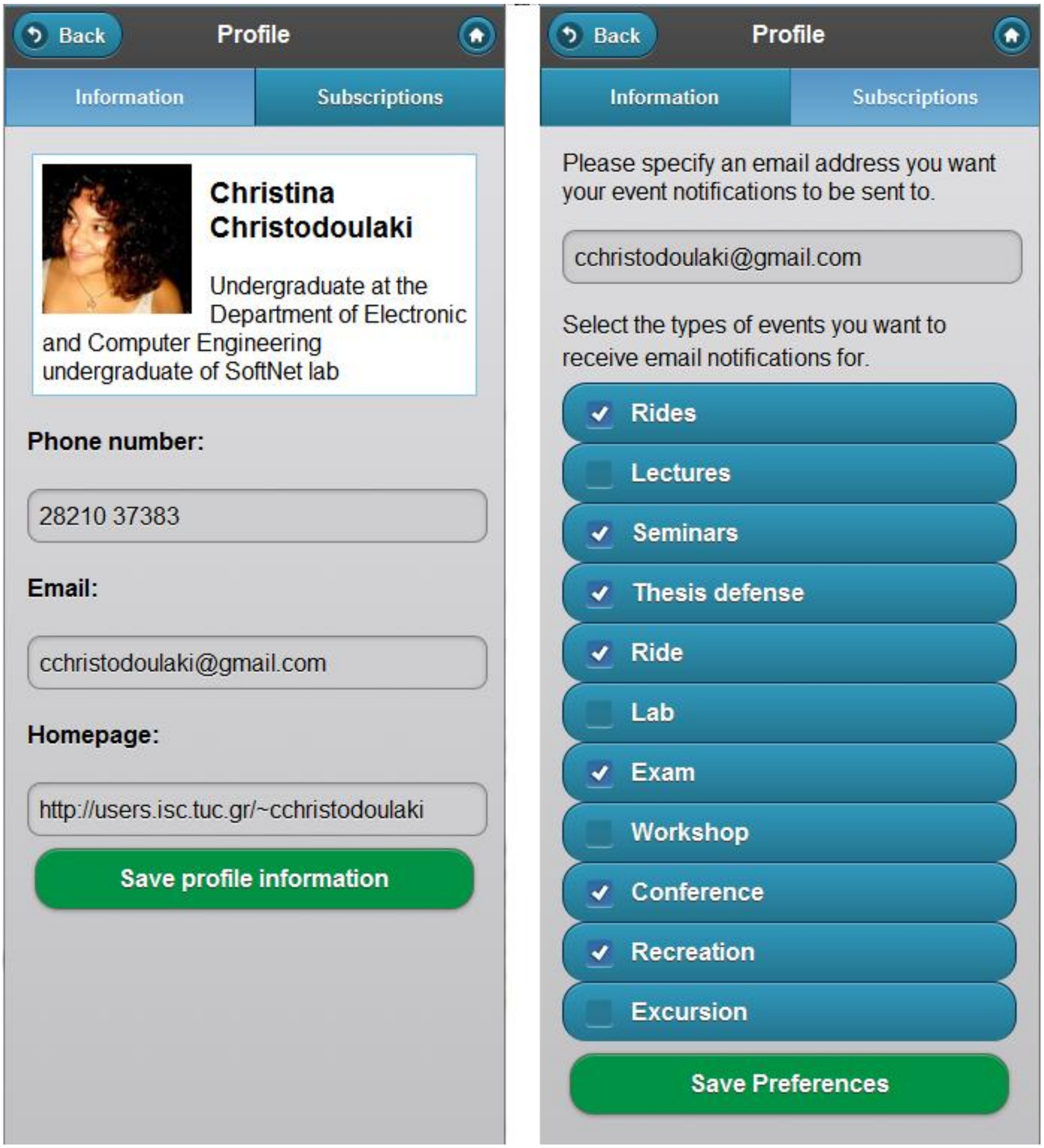
to me, stavros

A new Exam event has been posted at http://147.27.41.130:8080/jqCampus/controller?action=LOGIN_AND_VIEW_EVENT&id=103 by cchristodoulaki

On Friday, August 3, 2012 at 12:00 AM

Please arrive on time and look up your preassigned seats at the course site before arriving

Figure 53 new event subscription email format



Christina Christodoulaki

Undergraduate at the Department of Electronic and Computer Engineering undergraduate of SoftNet lab

Phone number:

28210 37383

Email:

cchristodoulaki@gmail.com

Homepage:

http://users.isc.tuc.gr/~cchristodoulaki

Please specify an email address you want your event notifications to be sent to.

cchristodoulaki@gmail.com

Select the types of events you want to receive email notifications for.

- [✓ Rides]
- Lectures
- [✓ Seminars]
- [✓ Thesis defense]
- [✓ Ride]
- Lab
- [✓ Exam]
- Workshop
- [✓ Conference]
- [✓ Recreation]
- Excursion

Figure 54 Profile Page, Information and Subscriptions

7.3 Evaluation Indoors and Outdoors

The final system was evaluated with two students from a design and development expert group from one of the labs of the University, and on location with two students from the Technical University of Crete. The design and development group evaluated a walkthrough of the application, while the two students evaluated the system with the Think Aloud methodology (They were asked to complete tasks that correspond to each of the User Goals described in Chapter 3, and notes were taken of user comments). The users found the functionality useful, intuitive, and described the experience to be consistent.

7.4 Evaluation and Diverse Mobile Devices

The system targets primarily mobile devices such as smart phones and tablets. Therefore testing was done on diverse mobile devices, such as phones and tablets running iOS and Android, and windows 7. Devices behave similarly throughout user task completion, apart from form element input which renders differently depending on device and browser, and takes into account the user interface of that device. In addition, the interface adapts to the device screen size and orientation. We filmed a 12 minute video of the experience on the above devices, which will accompany this report.

7.5 Summary

Designing interfaces for mobile devices presents a challenge. Not only must the design take context and largely varying screen sizes into account, but also the testing and evaluation must be done on a multitude of devices, and on location and in context where the application is expected to be used.

Chapter 8 - Summary Conclusions

8.1 Summary

We have designed and implemented a mobile information system that provides spatial information about indoor and outdoor locations in the University campus. This information includes buildings, parking spaces, athletic courts, lecture halls, offices, labs, coffee shops, etc. It can associate people and services with these locations. The information system can then be used for indoor and outdoor navigation in the campus.

The location of the mobile phone user outdoors is captured by the GPS capability of the mobile device and is displayed on top of the map so that the user can orient herself with respect to the buildings and other outdoor locations. Interactions with the map objects can display information about the name and functionality of buildings and other outdoor places. Each building has interactive floor maps associated with it. The floor maps include information about offices, lecture halls, labs, etc. The location of the user on the map is captured on request by selecting the room number nearest to the user from a selection list and the location is displayed on top of the interactive floor plan. The user can interact with the interactive floor plan to find out the function of the rooms near him, or the location of the rooms that he wants to visit. The maps and the floor plans can be scaled arbitrarily. The information system can provide information about the location of people (offices of professors, employees, graduate students, etc), displaying the campus map and highlighting the location of the building in which they are located, displaying the floor plan of the floor in which they are highlighting the room in which they are located on top of the floor plan.

By interacting with locations on top of the university map and the floor plans the user can navigate to find more information about the location. For example, by selecting a lab from a floor plan the user can navigate to lab details and find information about the people that are associated with the lab, their room locations, their web site, etc. The information in the data base is very richly structured to allow complex navigations and information mining, including information that is associated with locations in the campus outdoors or the buildings.

The mobile information system can also operate off-line so that its functionality is delivered even in places which may not have (free) internet access and the user does not want to use mobile internet access for downloading large pieces of data such as large maps.

The data generated in the above functionality can be categorized as semi persistent (University Directory and special information), and real time, (events). The mobile information system we designed takes this into account and allows download of semi persistent data, and caching of interface resources, so the functionality concerning the semi persistent data can function as part of a native application, running in the browser.

The mobile information system also manages information about university events such as a lecture, a meeting, an exam, a tennis game within the campus premises, etc. Events are associated with space and time, and therefore are associated with places on the interactive campus map and interactive floor diagrams. The information system organizes the events according to their time of occurrence, and the users can browse in this list to find events of interest. University users can be registered users and they can have profiles describing their interests in events. The system will only notify them for events that interest them.

The university community is structured into sub-communities based on their characteristics. Such sub-communities are for example professors, professors of the Electronic and Computer Engineering Department, graduate students, undergraduate students, university personnel, lab members of a certain lab, etc. These communities have different interests and they may have to be notified for different types of events. The originator of the event can restrict, if he wants, the visibility and the notification of events to certain sub-communities of the university. For example, an open meeting with a visiting professor who has interests related to web engineering and is going to take place between 2 and 4 in the departmental meeting room for ECE is an event that may interest only the professors and the graduate students of the ECE department. The system allows the user that creates the event to specify that the community that will be notified for the meeting is the community of professor and graduate students of ECE. Our user can add indoor and outdoor locations to events, by progressively narrowing his selection (by type and building). System displays available locations and once added to the event, they contain a link to the campus and floor map, outlining the location on the map.

More informal community events can also be created. For example, a graduate student may want to notify the students of the lab that he belongs to that he will be departing from the university to the city of Chania with his car that is parked at a particular location in the campus, at a certain time, inviting interested fellow students for a ride.

The application developed is a Web Application accessible from mobile devices. It is a Rich Media Web Application since it is heavily focusing on interactive graphic content presentations and asynchronous communication protocols (AJAX). It also allows contextualized and personalized information access, where the context involves information relevant to time, space, and community.

In contrast to existing Rich Media mobile internet applications which are native applications developed for particular platforms, the application developed is a completely Web Application developed on top of the developing HTML 5 standards. The project showed the feasibility of developing large scale, complex, Rich Media, interactive Web Applications for mobile devices. This approach has a major advantage in comparison to developing native mobile applications which have high costs of development, maintenance and time to market, due to development in different platforms. Our application is accessible by all the major platforms of mobile devices.

We have presented the detailed analysis of requirements, storyboarding, paper prototyping of user interfaces, evaluation of user interfaces using heuristic and think aloud methodologies, data base design and relational implementation, patterns used for the mobile web application development, the resulting Web Server and Web Client architectures which utilize local storage and caching in the mobile Web Client, asynchronous communication and synchronization with the Web Server, and capability for off-line operation. We have also presented the technologies and tools used for the development, as well as the difficulties found in the development process, and the solutions adopted.

8.2 Contributions

We have designed and implemented a mobile information system that provides spatial information about indoor and outdoor locations on the University Campus, capable of working as an offline application. Furthermore, we have generalized and provided a model of an information system that describes an institution spread out on many locations on a campus, such as a University.

We have also provided a model of organizational context, composed of communities, events, time, space, and personal preferences, in the everyday life of an institution. We have shown how to use rich media such as graphics, maps, diagrams and geolocation to present the spatial context in the user interface.

Overall, we show the feasibility of building a platform independent, offline capable, web application, suitable for mobile devices, and propose the system architecture, architectural patterns for building such a system.

References

[Brown et al. 2003] k. Brown, G. Craig, G. Hestler, R. Stinehour, W. Pitt, M. Weitzel, J. Amsden, P. Jakab, D. Berg: Enterprise Java Programming with IBM WebSphere, IBM Press,2003.

[Buxton 2007] B. Buxton: Sketching User Experiences, Elsevier, 2007.

[Delgado 2011] E. Delgado: Offline as your world used to be, Kuala Lumpur, Malaysia, http://devfest-html5-offline.appspot.com/#1 , 2011.

[Fielding 2000] R. Fielding. Architectural Styles and The Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm .

[Freeman and Robson 2011] E. Frieman and E. Robson: Head First HTML5 Programming, A Learner's guide to building web apps with JavaScript, 2011.

[Abbott and Fisher 2011] M. Abbott and M. Fisher: Scalability Rules, 50 Principles for Scaling Web Sites, Addison Wesley, 2011.

[Alonso et al 2004] G. Alonso, F. Casati, H. Kuno, V. Machiraju: Web Services, Concepts, Architectures and Applications, Springer 2004.

[Alur et al 2008] D. Alur, J. Crupi, D. Malks: Core J2EE Patterns, Best Practices and Design Strategies, Second Edition, Prentice Hall, 2008.

[Bas et al. 2003] Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 2nd ed*. Addison-Wesley Professional, 2003.

[Beyer and Holtzblatt 1998] H. Beyer and K. Holtzblatt: Contextual Design, Defining Customer Centric Systems, Morgan Kaufman, 1998.

[Brown and Wilson 2011] A. Brown and Gwilson (editors): The Architecture of Open Source Applications, ISBN 978-1-257-63801-7, 2011.

[Gamma et al 1995] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Professional, 1995.

[Gartner 2010] http://www.fiercemobilecontent.com/story/gartner-forecasts-mobile-web-access-will-surpass-pcs-2013/2010-01-13

[Gordon 2011] I Gordon: Essential Software Architecture, Springer, 2011.

[Gualtieri 2011] M. Gualtieri: Mobile Application Design Best Practices, Forrester report, http://www.forrester.com/Mobile+App+Design+Best+Practices/fulltext/-/E-RES59132?docid=59132 April 2011.

[Fowler 2003] M. Fowler: “Patterns of Enterprise Application Architecture”, Addison Wesley, 2003.

[Kerpen 2011] D. Kerpen: “Likeable Social Media, Mc Graw Hill, 2011.

[King 2008]A. King: Website Optimization, O’Reilly, 2006.

[Lidwell et al 2003] W. Lidwell, K. Holden, J. Butler: Universal Principles of Design, Rockport publishers, 2003.

[Microsoft 2009] Microsoft: Microsoft Application Architecture Guide, Second Edition, 2009.

[Moggridge 2007] B. Moggridge: Designing Interactions, MIT, 2007.

[Nielsen 1993] J. Nielsen: Usability Engineering, Academic Press, 1993.

[Nielsen and Mack 1994] J. Nielsen and R. Mack: Usability Inspection Methods, John Wiley and sons, 1994.

[Norman 2004] D. Norman: Emotional Design, Basic, Books, 2004.

[Potasso et al 2008] C. Potasso, O Zimmermann, F. Laymann: RESTfull Web Services vs. “Big” Web Services: Making the Right Architectural Decision, WWW 2008 / Refereed Track: Web Engineering - Web Service Deployment , pp806-814.

[Raman 2009] T.V. Raman: Toward 2^w, Beyond Web 2.0, CACM 52, 2, February 2009.

[Shaw and Garlan 1996] M. Shaw and D. Garlan: Software Architecture, Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[Smitt and Simpson 2011] C. Scmitt and K. Simpson: HTML5 Cookbook, O’Reilly, 2011.

[Snyder 2003] C. Snyder: Paper Prototyping, Morgan Kaufman, 2003.

[Taylor et al. 2010] R. Taylor, N. Medvidovic, E. Dashofy: Software Architecture, Foundations, Theory, and Practice, John Wiley and Sons, 2010.