Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**(Ελληνικά)** **Στοχευμένες καμπάνιες χρησιμοποιώντας τις δυνατότητες των κινητών συσκευών** **(Αγγλικά)** **Targeted Campaigns Using the Power of Mobile Devices**
Ονοματεπώνυμο Φοιτητή	**Ιωάννης Κοζομπόλης**
Πατρώνυμο	**Χρήστος**
Αριθμός Μητρώου	**ΜΠΣΠ 15036**
Επιβλέπων	**Ευθύμιος Αλέπης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης   **05/2018**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)   (υπογραφή)   (υπογραφή)

Αλέπης Ευθύμιος   Βίρβου Μαρία   Πατσάκης Κωνσταντίνος
Επίκουρος Καθηγητής   Καθηγητής   Επίκουρος Καθηγητής

Table of Contents

Abstract

We all know about the advertisement campaigns or simple campaigns. In our days, it is the most profitable way for promotional activities, sharing content and attracting interest. The main idea is a series of messages that aim to convince and urge the users. The dissertation focuses on the power of mobile devices and how that can be exploited in order to design targeted campaigns. A mobile device is possible to provide useful information about user activities, interests and location. Imagine a mobile application that has the ability to recognize the exact location every time a user accesses the location service. Furthermore, think how easily a mobile application could access personal data, activities, education and interests from social media such as Facebook. There is a way to incorporate all of these features into one large company mobile application. The application will save this information in a database. An admin tool will manage this data in order to design targeted campaigns for these users. By having a lot of information, we could increase the effectiveness of the campaigns. Also, the campaigns will not be disturbing and they will be appealing to the right users.

Introduction

2.1 What a campaign is and how could it be implemented by using information technology?

A campaign is defined as a series of coordinated activities, such as public speaking and demonstrating, designed to achieve a social, political or commercial goal. By using information technology, it is possible to gather useful information. The information could be analyzed and studied in order to give a system the ability to recognize future activities. Furthermore, a campaign is able to target a specific segment, depending on the collected information. Imagine a target group deriving from data extracted by a specific location, specific studies and working experience. Moreover, information technology could help on scheduling and automating campaigns with dynamic content.

The current dissertation describes a method to collect specific information from the users using mobile devices. Specifically, a mobile application has been developed, that contains features and services, which are collecting the specific data. In addition, a web application has been developed, which is capable of creating specific segments. Those segments can then be targeted by campaigns. The campaigns can be scheduled on specific date/time and a certain text can be sent to the mobile users who are using the mobile application. The users are able to receive the specific content via push notifications. In other words, the current dissertation implements the main idea of campaigns; a series of messages that share a single idea and theme, which form an integrated communication. Those targeted campaigns are using the power of mobile devices. Such campaigns, are more efficient and may eliminate significantly the possible disturbance caused to the end users.

2.2 Gathering information through the use of mobile devices

Every person has at least one mobile device. Mobile devices are used for communication. However, the evolution of the smartphones has changed our life. Such devices contain many sensors and useful applications. The sensors offer us information such us user location, environment temperature, acceleration and lighting. Additionally, the option of developing a mobile application is provided; an application that is able to access information from the user's social media accounts. The social media accounts can provide useful information such as origin, working experience, personal data and interests.

According to these possibilities, a mobile application has been developed, which exploits these features. Specifically, the application runs a service that recognizes the location coordinates every time the GPS is enabled. Moreover, scheduled jobs save the last location in a remote database. The application has been implemented with Facebook login. The Graph API is used in order to access the useful information that is mentioned above. In conclusion, it is communication provided via push notifications. This communication channel serves the campaigns' interaction with the end users.

2.3   Designing Campaigns with the use of a web application

A web application is an application that is running on a web server. The web server can be accessed from the internet. So, the clients can interact with the web application directly from the internet. Such applications consist of web services. A web service helps the application to communicate with other applications. By developing web APIs, it is possible to achieve the interaction between a mobile application and web application. Furthermore, web applications are able to provide high performance, complex services, data protection, storage data, dynamic operations and remote access using a simple web browser.

According to the above flexibility, a web application has been implemented. The application provides web services, which have the ability to save data in a database, authenticate the users and send push notifications. An admin tool is available as web page. The administrator can use filters to create specific segments from registered users. The segments are created by the useful information that is collected from the mobile application. The administrator can use each segment in order to design a different campaign so as to aim specific users. Another feature is the campaign creation and configuration. The administrator can schedule a campaign to run at specific date/time using a specific segment of users in order to send a specific message via push notification.

Field Review

3.1   Hiscox's B2B Location-Based WiFi Campaign

Hiscox is an international specialty insurance company. The company has launched a location-based WiFi campaign in the United Kingdom that combined mobile display ads with an existing outdoor board campaign.

When businesspeople logged on to public WiFi in the vicinity of one Hiscox's outdoor boards, they were greeted by a digital ad on their browser start-up screens that correlated with the existing outdoor boards in the immediate vicinity. In other words, if they logged on to the internet while sitting at a restaurant in Covent Garden, the ads on the start-up screen matched the ads on the nearby outdoor boards. Both the internet ads and outdoor boards were specifically designed for people who were located near Covent Garden. The result was a double impact: first on the outdoor boards that were strategically located near the WiFi hot spots, and the second on the start-up screen that greeted people accessing the Internet using their smartphones, laptops, or tablet computers.

How did the campaign perform? Like gangbusters. The click-through rates on the location-based WiFi campaign were five times higher than the company's average rate for traditional online display ads.

3.2   Paramount's Shrek Display Ad Campaign

Over 50 mission people visit Yahoo!'s mobile home page each month, which is about 1.5 million visits per day. Regarding the high volume of traffic decided the creation of a "rich-media" campaign around the release of the latest Shrek movie.

Visitors to Yahoo!'s mobile home screen were able to see Shrek's head along the bottom of their smartphone screens. When the people touched the top of Shrek's head, the character would pop up and fill the whole screen. If they tapped Shrek's head again, they would be redirected to the mobile microsite where they could buy tickets or see more information about showtimes.

3.3   Campaign tool – TargetEveryOne

TargetEveryOne is a cloud-based admin tool that could be easily used from large scale companies. The companies could design scheduled campaigns and communicate via major channels such as social networks, SMS, email, mobile apps and QR codes. The idea is to recruit valuable customers into a database, add segments filtered by valuable contacts based upon their preferences and interaction behaviors over time, create digital marketing campaigns using drug-and-drop action based interactive modules, choose the preferred communication channel and analyze statistics to help you facilitate successful marketing strategies.
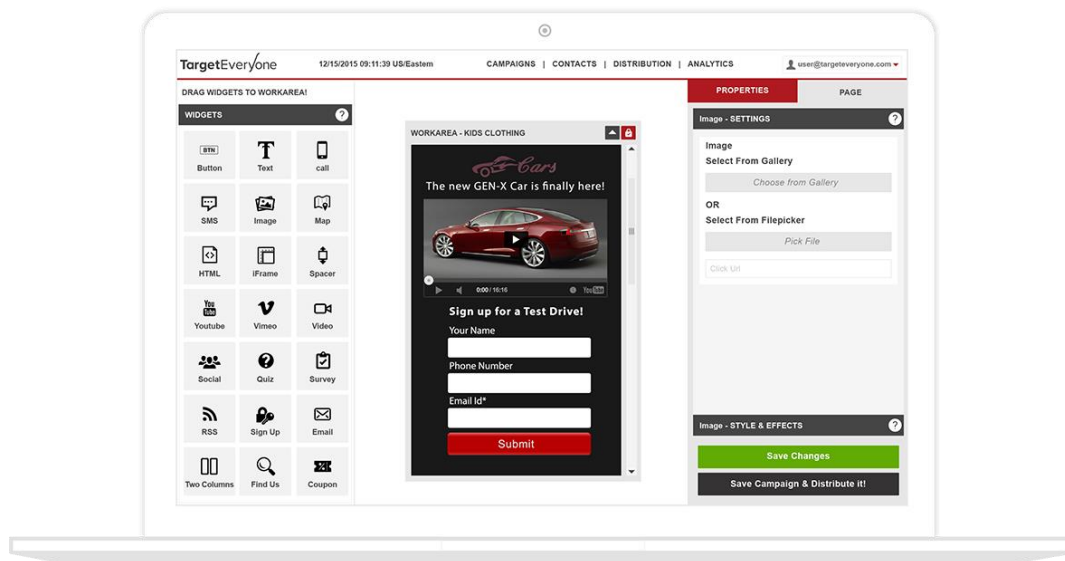
3.4   Campaign tool – Autopilot

Autopilot is a cloud-based tool for design targeted campaigns. The tool is able to create campaigns that could support marketing journeys. The campaigns could be designed with multiple flows, which could interact with user inputs, in order to decide dynamically the next marketing actions. The campaigns could use communication channels such us emails, Heads-up messages, SMS and Postcards. All contacts are imported and organized into lists, smart segments and folders. So, the target segment is created dynamically and matches any combination of field values and behaviors given.
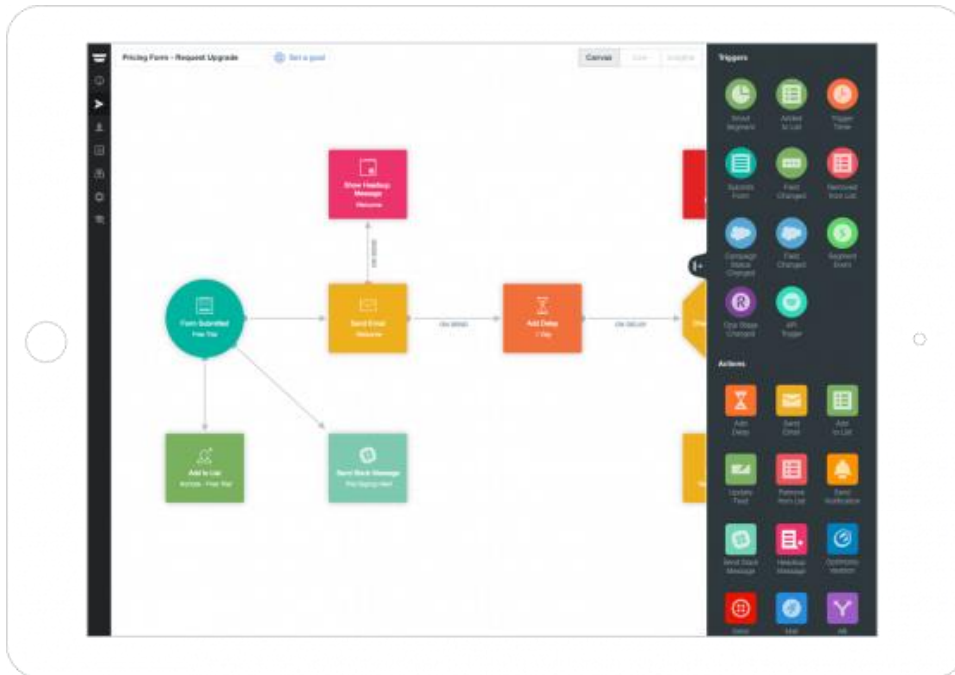
Presentation and Application Use

4.1   General Description

The implementation includes two different applications, an android application and a web application.

The android application is a simple native application that includes custom login, registration flow, Facebook login, retrieval of Facebook data features and custom location-service. The android application does not provide any further functionality. Imagine that it is a commercial mobile application of a large-scale company, which integrates all the above login flows and the services that have been implemented. Using the custom location-service the application can update the user's last known location. The service has been designed in order to retrieve the user's location whenever the GPS is enabled. A scheduled job updates the last known location in a remote database using a web service. Using the Facebook login, the application is able to retrieve a lot of information regarding the user from the Facebook Graph API. Various data from Facebook login are recorded into a database using different web services.

The web application includes web services and an admin tool web page. A lot of web services are used by the android application in order to save useful data on the database. The admin tool is a web page where the user has the ability to design segments to target specific audience. The segments are designed via filters that access the information from every user in the database. The admin tool provides the possibility of creating simple notification campaigns as well. This operation aims to schedule simple notifications for android mobile devices, which have the android application installed. The simple notification campaigns are configured for specific user target groups (segments) and specific execution time.

4.2   **Android Application**

The android application can be installed on android smartphones. As soon as the user opens the application from the menu, the login screen appears. The application offers two login methods, the custom login and Facebook login. For custom login, the user provides a valid username & password and then clicks on the login button. For Facebook login, the user clicks on the Facebook login button. If the user accesses the application for the first time, and thus is not registered, he can register through a custom account by selecting the "Don't have an account? Register Here" option. Alternatively, he can simply continue by pressing the Facebook login button and complete the registration automatically.
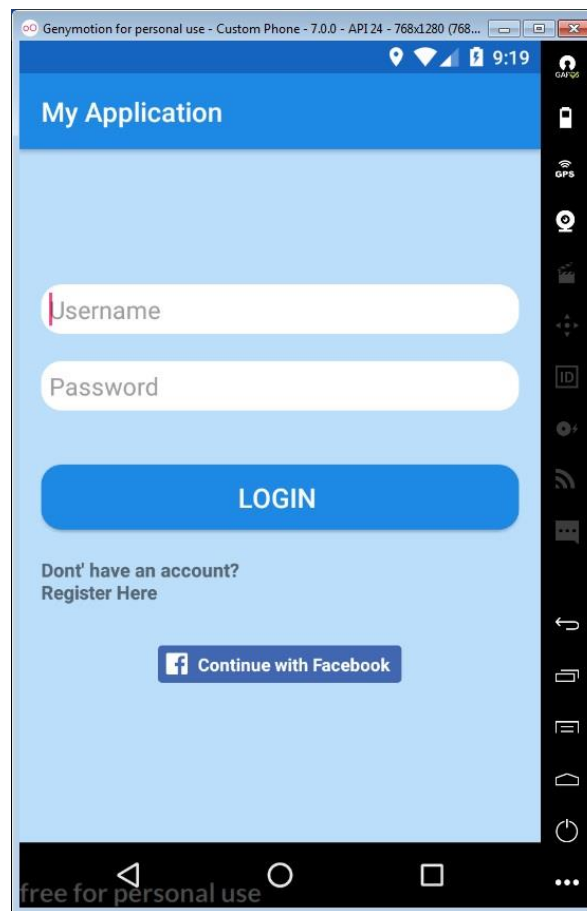
Figure 1: Android Application - Login screen

The custom login method includes security checks, which return error messages when the user is trying to login with invalid credentials. In this case, a message appears on the bottom of the screen with the following text "Invalid username or password".
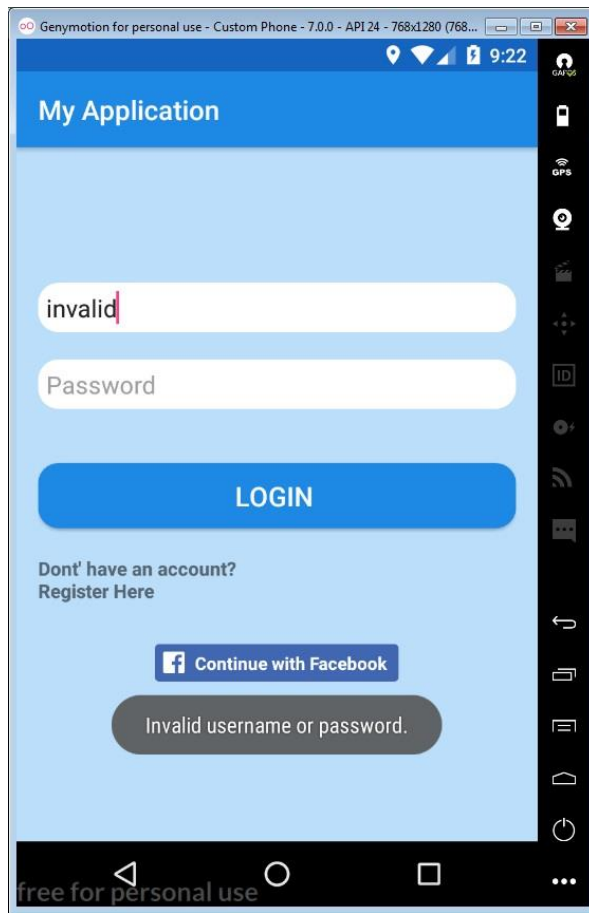
Figure 2: Android Application - Login screen invalid username

From that point, the user has to proceed with the creation of a new custom account by pressing "Don't have an account? Register Here". He will then be redirected to the registration page, where he has to fill in the fields with a valid username and password. By clicking on the "NEXT" button, the system sends a request to a web service in order to validate that the provided username has not been used by another user. If the provided username is unique, then the user is redirected to the second registration screen, else an error message informs the user that the current username already exists in the database.
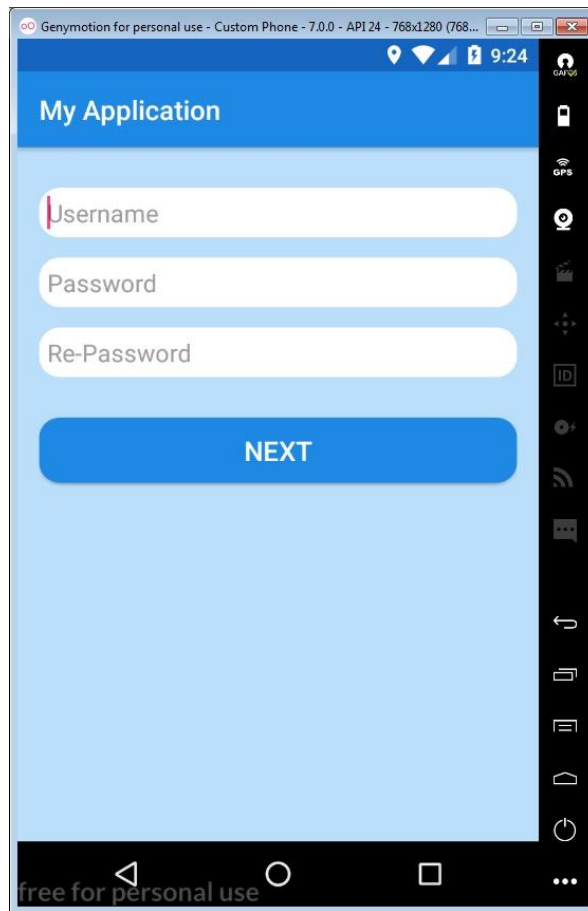
Figure 3: Android Application - Registration screen 1

The error message "The username exists" appears, in case the user is trying to register with an existing username, already stored in the database.
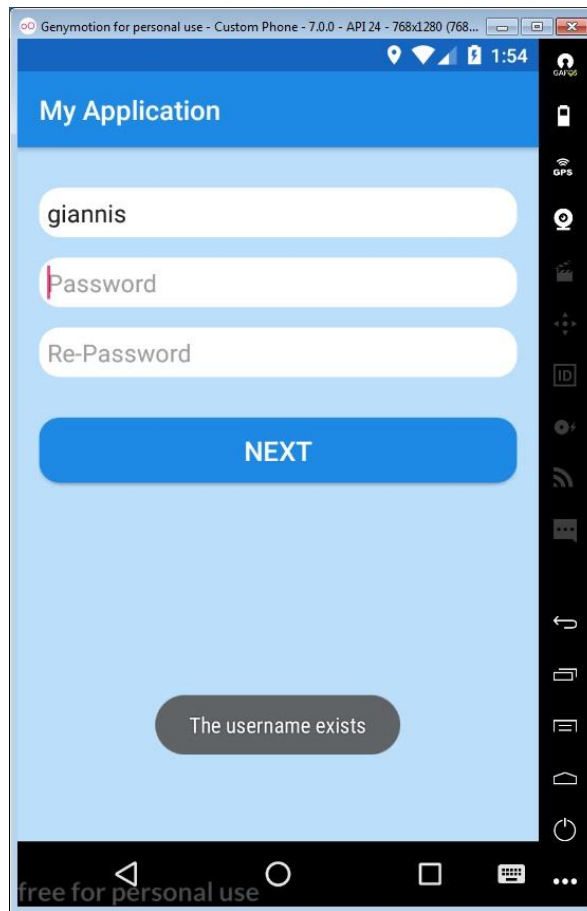
Figure 4: Android Application - Registration screen 1, username exists

The second registration screen contains input fields where the user can fill in a few personal data. The input fields are name, surname, email and mobile phone number. The user then clicks on the register button in order to complete the registration. The user is then redirected to the home screen of the application and his credentials along with the rest of the information given are stored in the database.
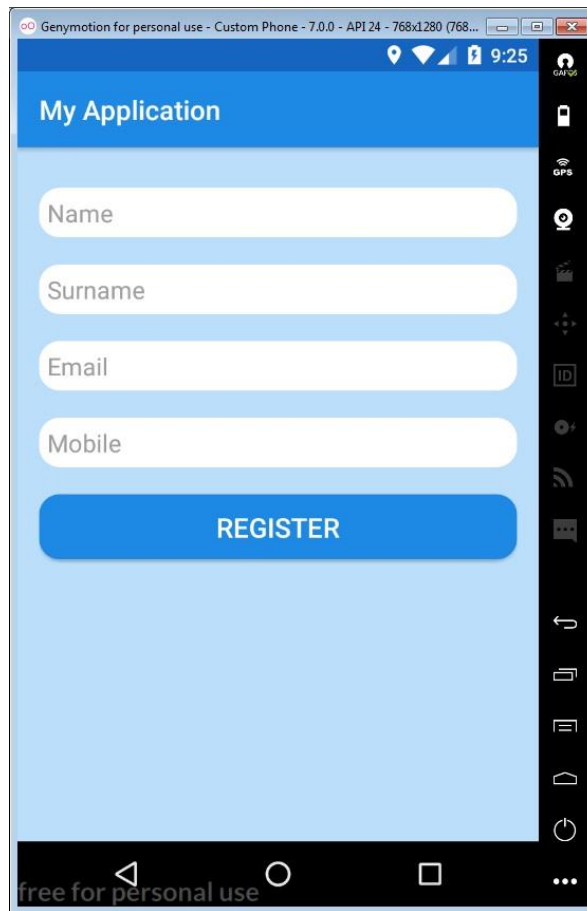
Figure 5: Android Application - Registration screen 2

    The user may also register or login automatically from the login screen by selecting the Facebook login button. By clicking on that button Facebook relative interface opens on the screen. The user completes his Facebook credentials and presses "Login". Then a second page of Facebook interface asks for the user's consent regarding the retrieval of personal data through his Facebook profile.
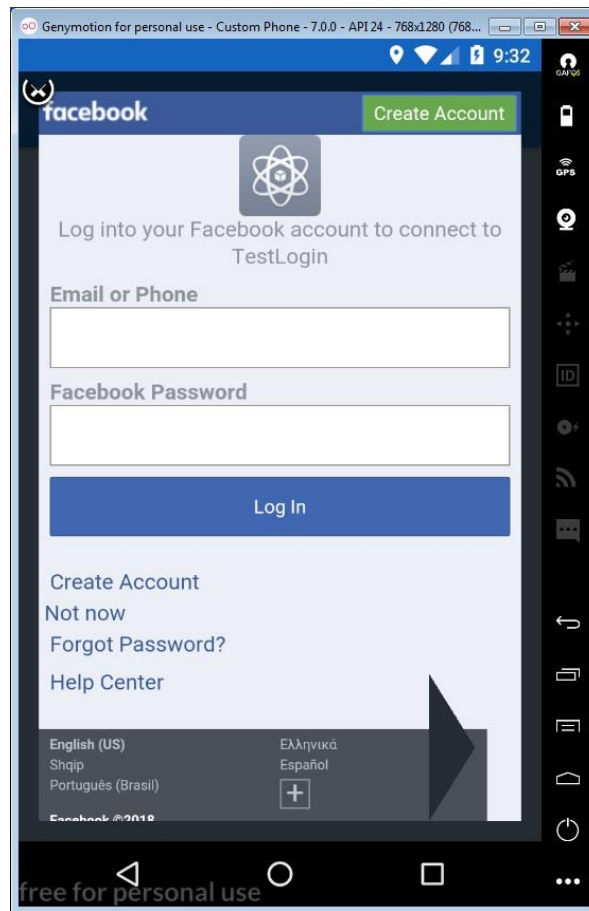
Figure 6: Android Application - Facebook login
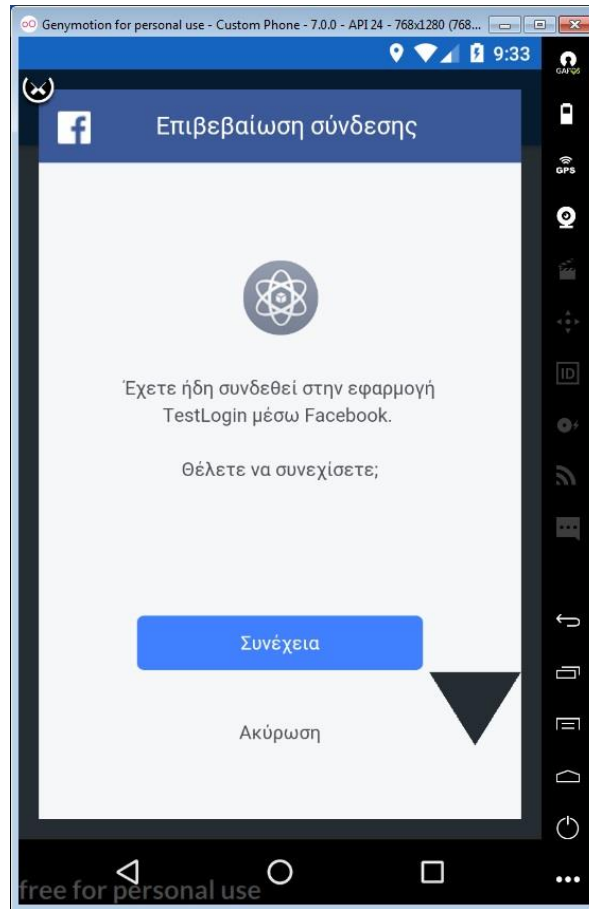
Figure 7: Android Application - Accept Facebook Application

Whether the user loges in either via custom login, or Facebook login, he is finally redirected to the home screen. The home screen is basically empty for the purposes of this dissertation. As discussed above, the application contains only those features that will be integrated in a commercial application, in order to provide the implemented services. So, imagine that the home screen contains content such as a commercial message.
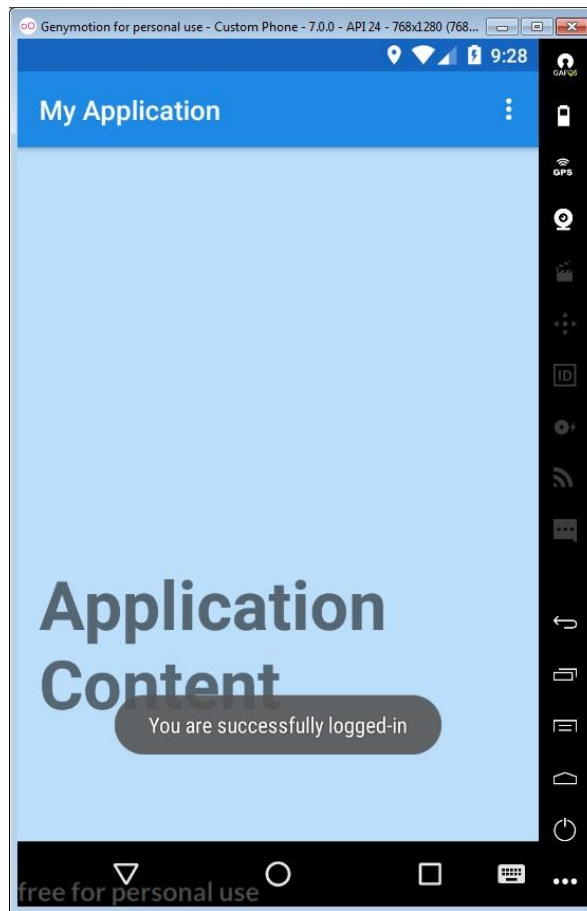
Figure 8: Android Application - Central screen

An option menu has been implemented on the home screen. The available options are:

- ◦ "Show Personal Data", which returns the registration info of the user
- ◦ "Get Last Location", which returns the last known location that is detected from the custom location service
- ◦ "Enable Location Detection", which enables the custom location service
- ◦ "Disable Location Detection", which disables the custom location service
- ◦ "Logout", which logs out the user

Below you may see the implemented menu of the mobile application:
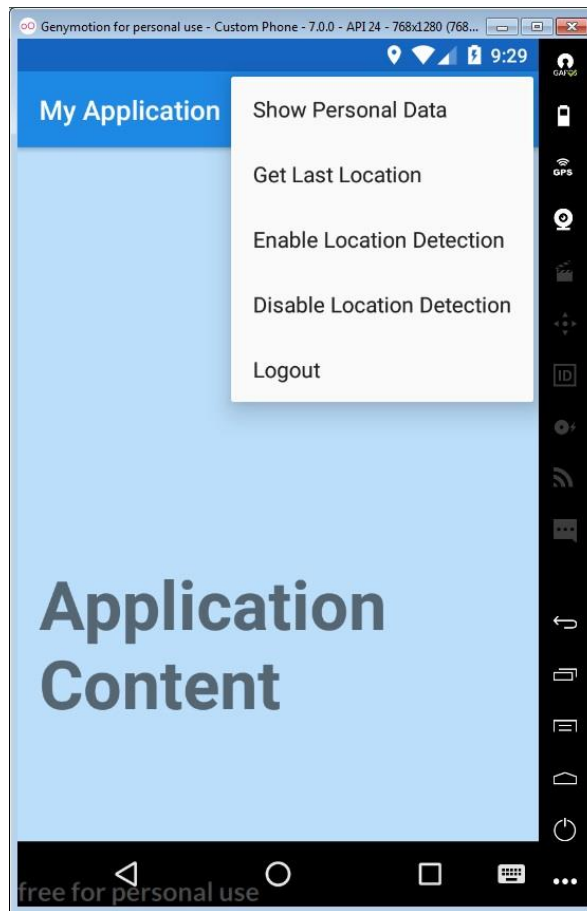
Figure 9: Android Application - Options menu

Below you can see the personal information, which is handled from the custom login registration:
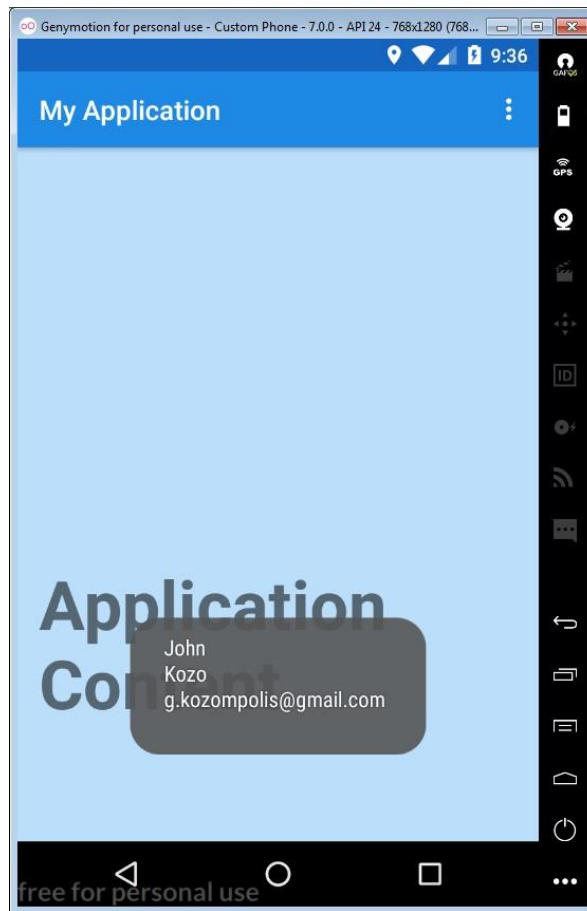
Figure 10: Android Application - Show registration data

Below you can see the last known location, which is handled from the custom location service:
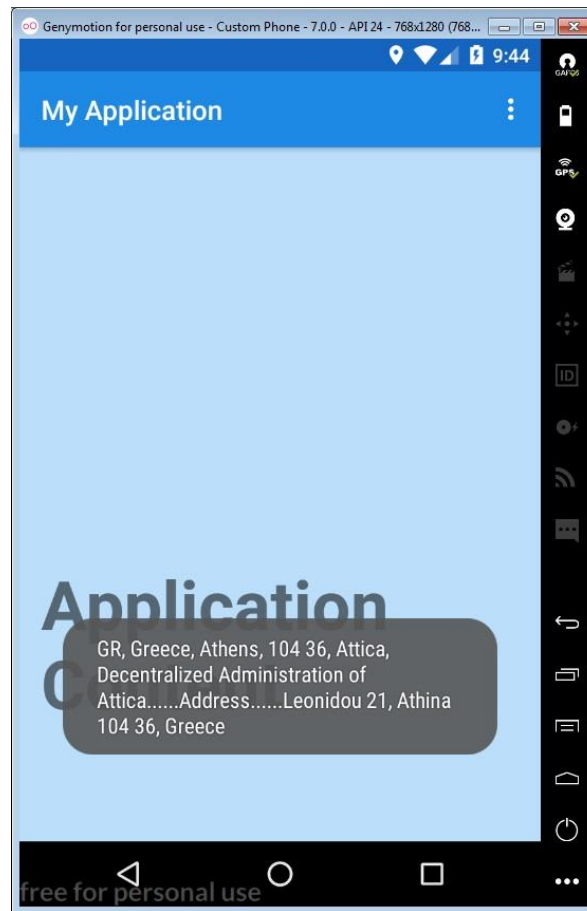
Figure 11: Android Application - Get Last Location

Below you can see how the application informs the user that the custom location service starts:
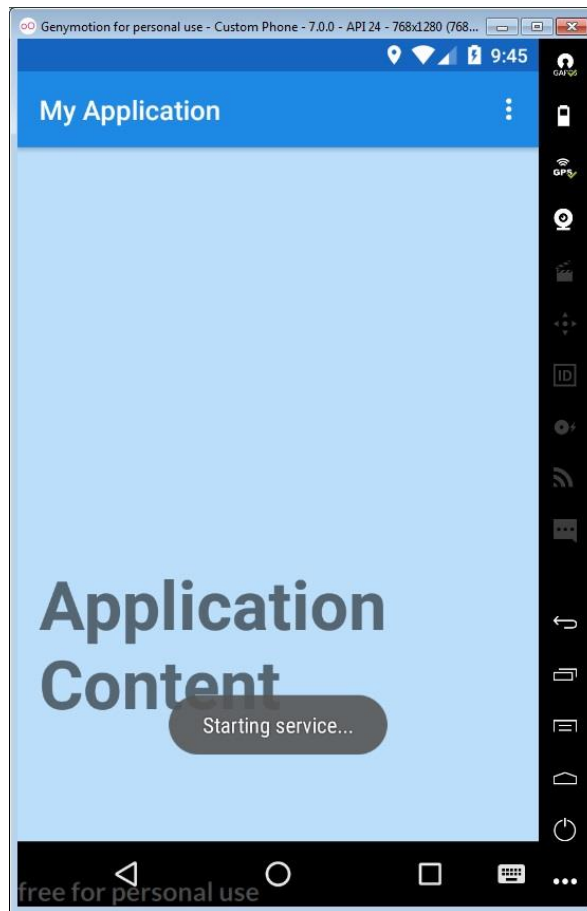
Figure 12: Android Application - Start Service

Below you can see how the application informs the user that the custom location service stops:
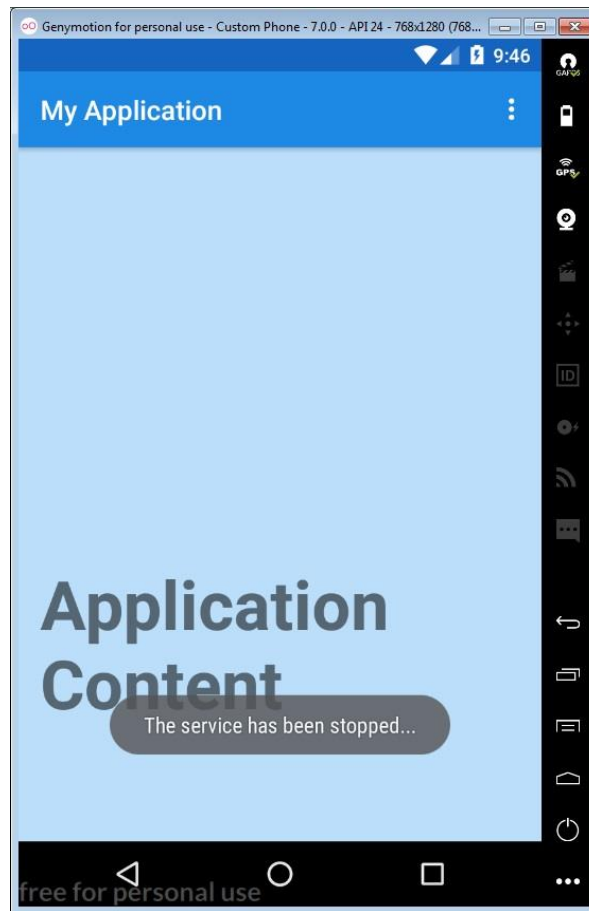
Figure 13: Android Application - Stop service

4.3   **Web Application – Admin Tool**

The web application consists of an admin tool and many web services. The admin tool is a dynamic web page. It provides a section for segment creation with the use of dynamic filters and another one for campaign configuration.

On the right side of the page you may find the main menu, which helps the user navigate through the admin tool. The user can navigate to the segmentation section, in order to create segments by clicking on the "Segments" option on the right menu. Below you can see the segmentation page. The page contains a list with the available segments. By pressing "New Segment" button, the user has the ability to create a new segment. The segment is a group of users, which is created by using data that are collected from the android application. The segmentation is necessary for the campaign configuration.
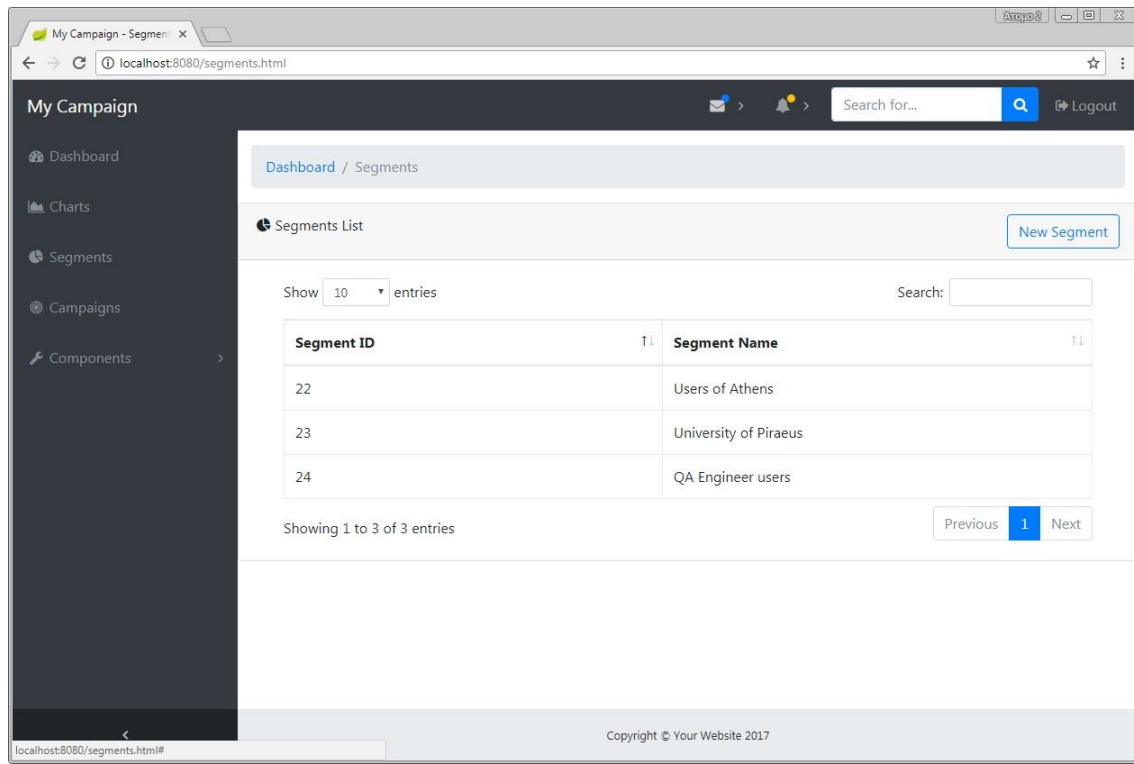
Figure 14: Admin Tool - Segmentation page

The user can configure a segment at the "New Segment" section. The first step of the configuration is to fill in the name and the description of the segment. The "Target Location" section contains a filter that helps the user choose a specific location. The filter consists of five fields:

- Country
- Department
- Prefecture
- City
- Postal Code

The user can select or deselect items in every step. The selected options return dynamically new choices in every step.

e.g. "If the user selects Greece as a country option, the next step filter will contain departments only from Greece".

The "Target Work Info" section contains filters such as specific employers and work positions, from which the user can form target group. The filter consists of two fields:

- Employer
- Position
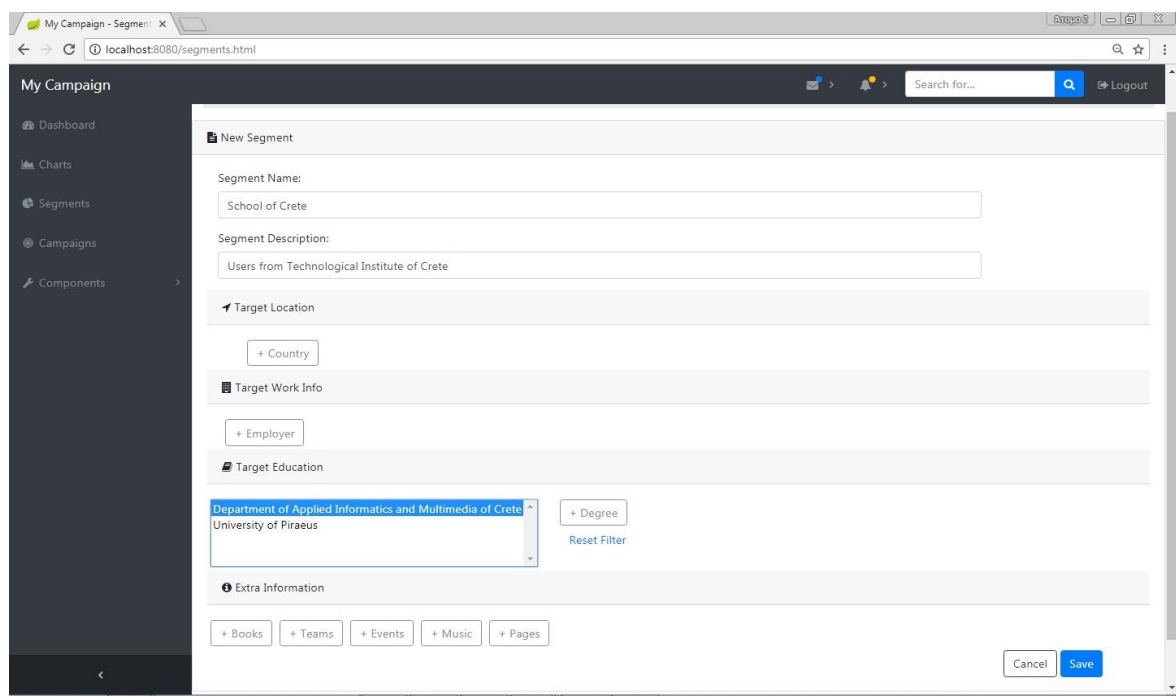
The "Target Education" section contains filters from which the user can select specific education experience. The filter consists of three fields:

- School
- Degree
- Year

The "Extra Information" section contains 5 simple filters:

1. Books: It is used to recognize users who prefer certain books
2. Teams: It is used to filter users who support a certain sports team
3. Events: It is used to filter users who are interested in certain events
4. Music: It is used to filter users who listen to a certain kind of music
5. Pages: It is used to filter users, who like certain pages on Facebook

All the filters can be used in order to create a segment. In multiple selection, the segment is created with logic AND. Below you can see a configured segment that contain users from the Department of Applied Information and Multimedia of Crete.

Figure 15: Admin Tool - New Segment

Below is a configured segment that includes users who live in Peloponnese:
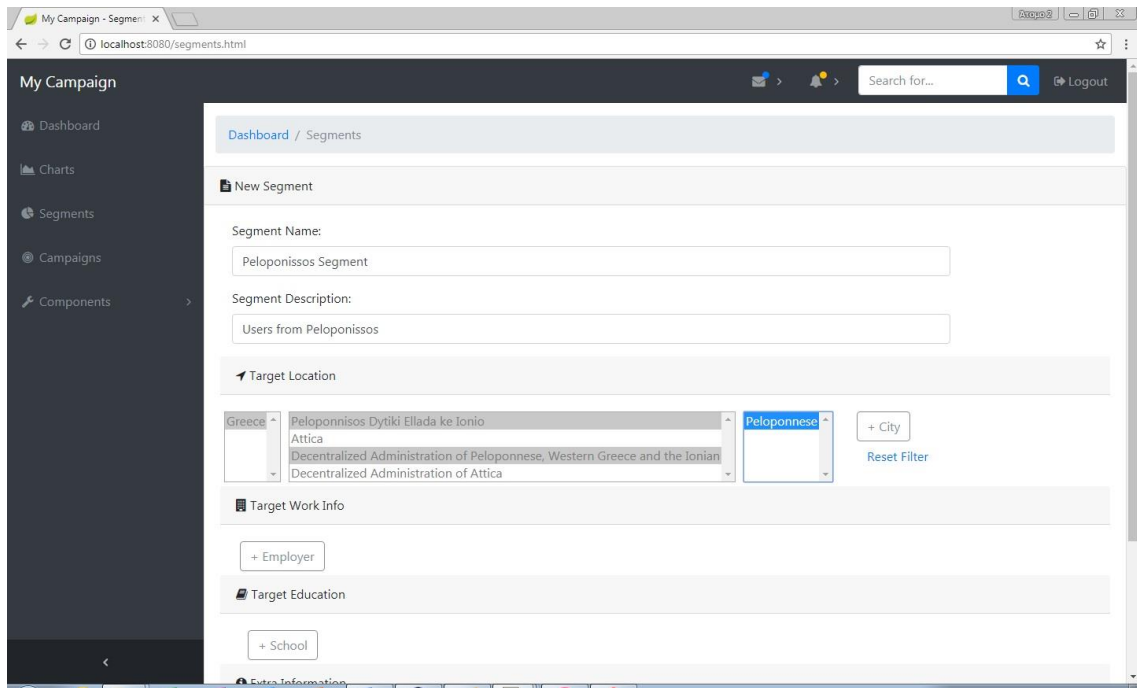
Figure 16: Admin Tool - Peloponnese segment

If the name and description input fields are completed, the segment can be saved right after the filter selection. If the segment has been saved successfully the success message "Success! The segment has been tested successfully" will be displayed.
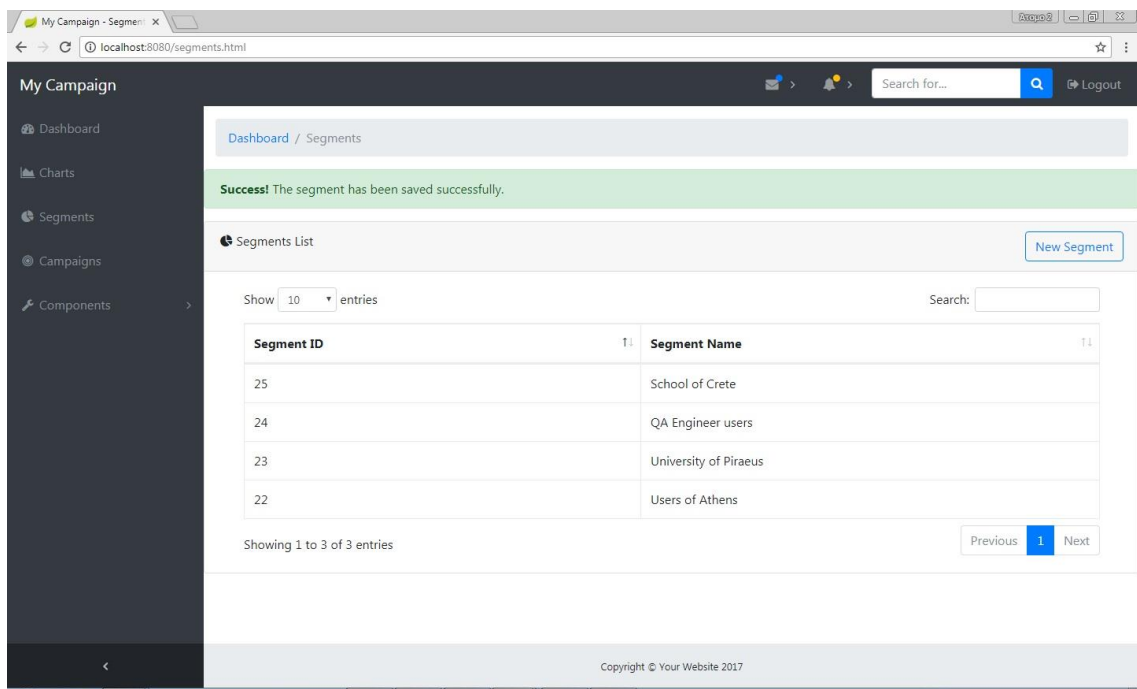
Figure 17: Admin Tool - Segment success message

The campaigns are configured on the campaigns section. The campaigns section displays a list of the configured campaigns. The user can edit only inactive campaigns. The "Activate" button enables the campaign to be executed on the configured time. The details button shows the configuration details of the campaign. The "New Campaign" button contains a dropdown sub-menu with options for different types of campaigns. Only the "Simple Notification" campaign has been implemented within this dissertation.
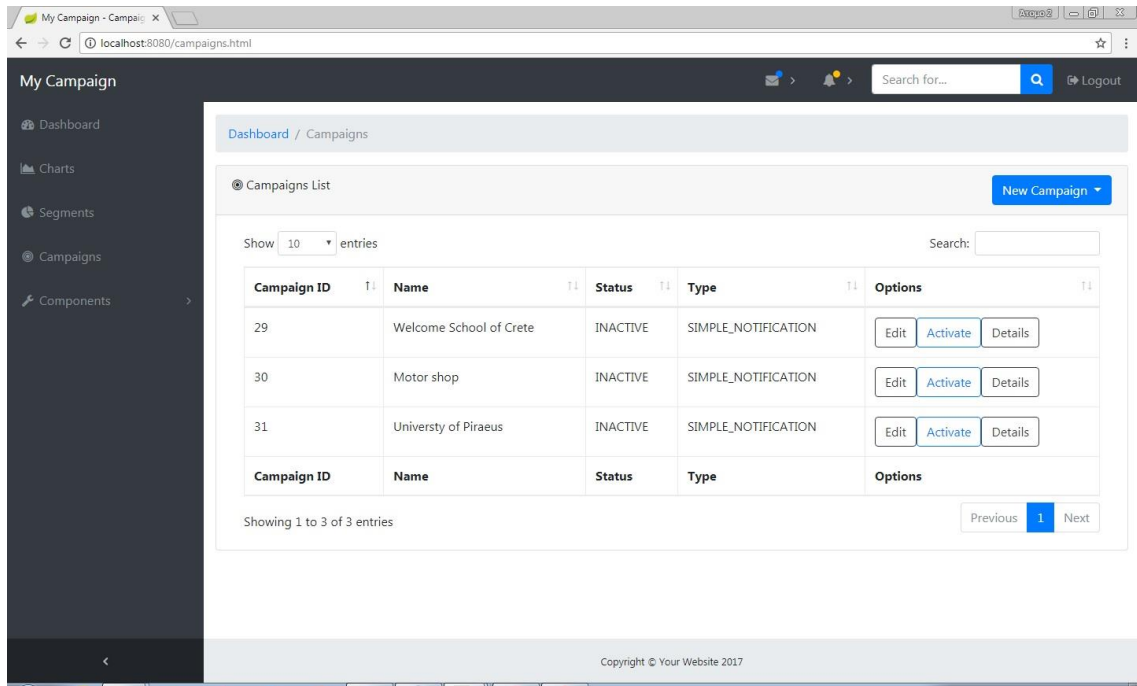
Figure 18: Admin Tool - Campaigns Page

By clicking on the "New Campaign" button and then chosing "Simple Notification", the user can create a new campaign of this type. The "Simple Notifications" campaigns are configured to send a simple notification on a specific time to a specific segment.
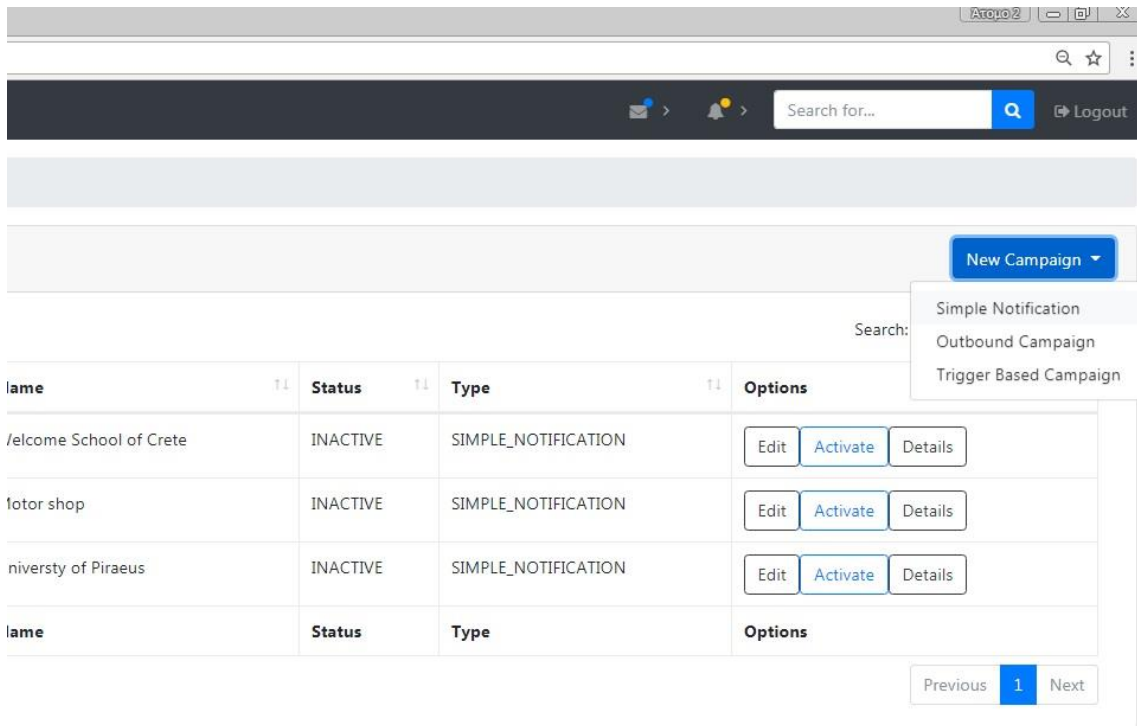
Figure 19: Admin Tool - New Campaign button

When creating a new "Simple Notification" campaign, the user should fill in the necessary input fields. The name and the description are required fields. The name must be a unique value:
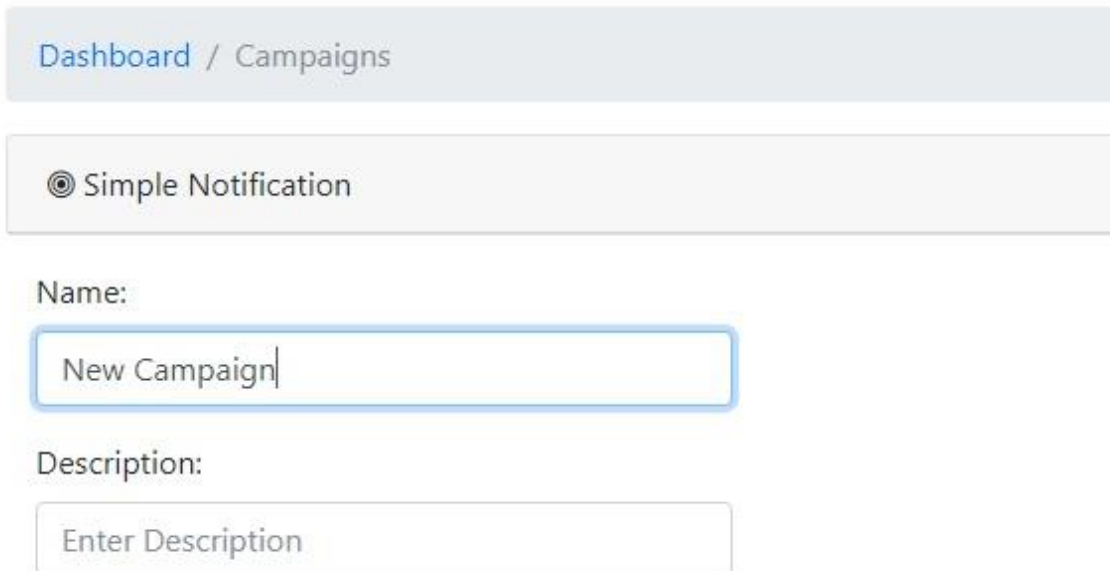
Figure 20: Admin Tool - Campaign name and description

The next configuration step is to select the specific segment that the campaign will target. The segment is selected from a dropdown list in the "Select Segment" section:

Figure 21: Admin Tool - Select campaign's segment

The next configuration step is to select the execution time that the campaign will run. In the "Execution Time" section the user should choose the preferred date and time from the dropdown menu:

Figure 22: Admin Tool - Campaign's execution time

The last step is to fill in the message title and the message body inputs fields in the "Notification Content" section. These inputs include the notification content which is received by the targeted mobile devices:

Figure 23: Admin Tool - Campaign's Notification Content

On the campaigns page, on the right of every configured campaign, the user can select the "Details" button. This action shows all the details of the campaign configuration. Below you can see the details of a configured campaign:
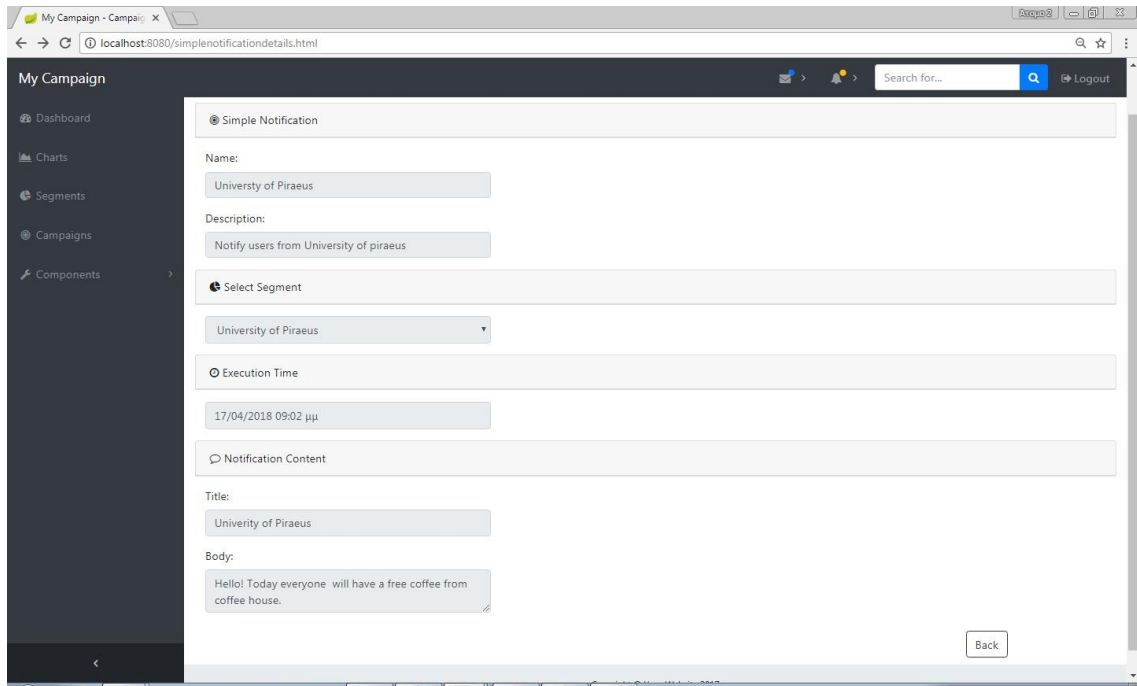
Figure 24: Admin Tool - Campaign Detail Section

When the configuration is completed the user can activate a campaign by pressing the "Activate" button on the right of the campaign name, within the campaign list. Then all the users who are included in the chosen segment receive the notification on their mobile devices.
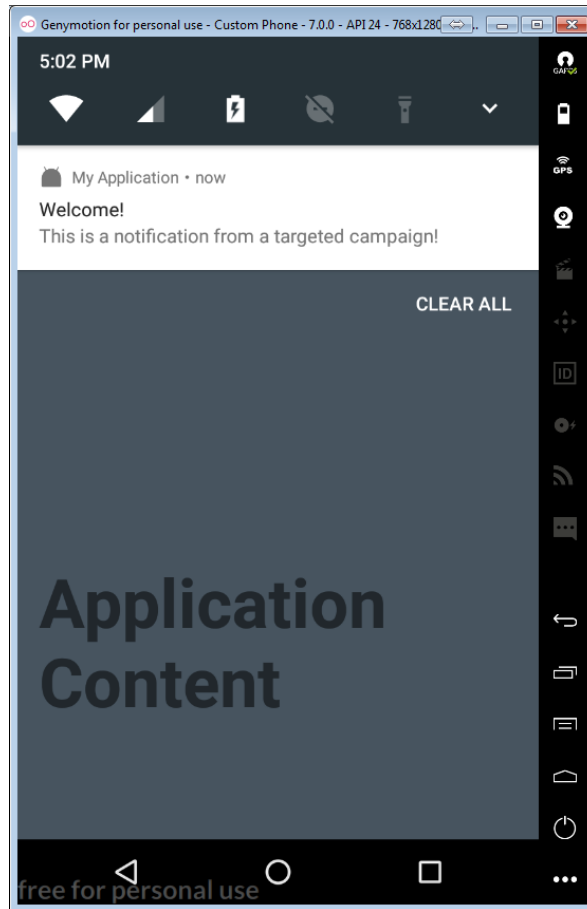
Figure 25: Android Application - Received Notification

System Architecture

5.1   Implementation structure

The implementation consists of two applications which communicate with each other. An android application has been implemented in order to run on mobile devices and a web application has implemented so as to provide specific web services and an admin tool.

The android application is installed on mobile devices with android operating system. This application uses the internet connection provided by Wi-Fi network or mobile network, in order to communicate with the web application. Through the internet connection, the android application is able to access all the required web services. The application requires internet

connection to operate. A service has been implemented to run in background mode and recognize the activation of the GPS sensor, whenever this may happen. On a separate time lap, the service saves the last known location locally. A scheduled job uses a web service, which uploads the last location on the database, as soon as the internet connection is established again. The application has also been designed to support Facebook login. The application exploits the Facebook graph API in order to access user Facebook profile and Facebook activity information. Thereafter, it requests specific web services in order to upload this information on the remote database. Every mobile device, that has this application installed, is connected with the firebase cloud messaging service.

The web application has been installed on a web server, which is accessible via internet connection. The web application communicates with the database directly through the network. All web services are implemented on this application. Also, the web page of the admin tool is hosted on the web server. The web page can be accessed from a simple web browser over the internet. The cloud messaging service is used from the web application in order to send simple notifications to the mobile devices with the android application installed .
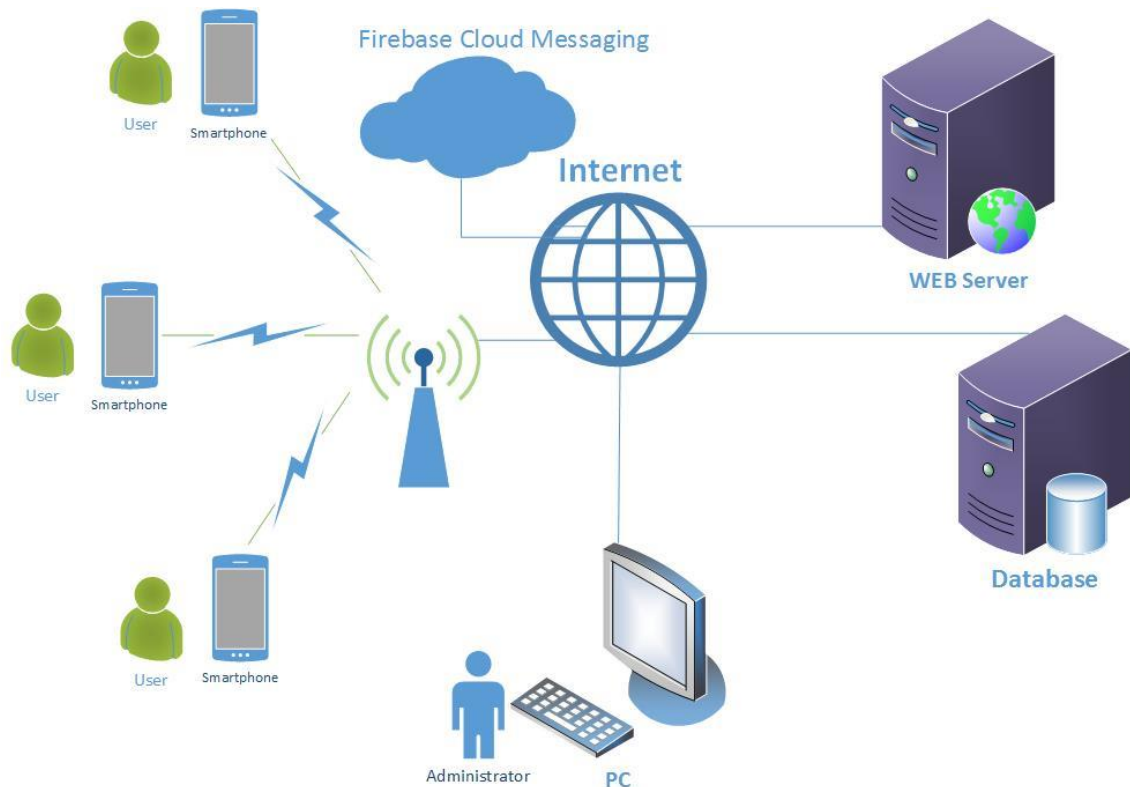
Figure 26: Implementation Structure - Implementation Diagram

5.2   **Development and Design Tools**

The tools that were used in the whole development are listed below:
- Netbeans IDE 8.2
- Android Studio

- Microsoft Visio
- Git
- Genymotion
- pgAdmin 4
- Postman
- Google Chrome – Developer Tools

5.2.1   **Netbeans IDE 8.2**

Netbeans is an integrated development environment (IDE) for Java. Netbeans allows applications to be developed from a set of modular software components called modules. Netbeans runs on Microsoft Windows, macOS, Linux and Solaris. In addition to Java development it has extensions for other languages like PHP, C, C++ and HTML5, Javadoc and Javascript.

Netbeans IDE has been used in order to write and compile java code of the Web Application. Netbeans provides a lot of useful frameworks (like maven), web servers (like Tomcat) and Database Services (like JDBC drivers) which are useful for the implementation. The IDE helps the developer to write clear and structured java code. The NetBeans Editor indents lines, matches words and brackets, and highlights source code syntactically and semantically. It lets you easily refactor code, with a range of handy and powerful tools, while it also provides code templates, coding tips, and code generators.

Keeping a clear overview of large applications, with thousands of folders and files, and millions of lines of code, is a daunting task. Netbeans IDE provides different views of your data, from multiple project windows to helpful tools for setting up your applications and managing them efficiently, letting you drill down into your data quickly and easily, while giving you versioning tools via Subversion, Mercurial, and Git integration out of the box.
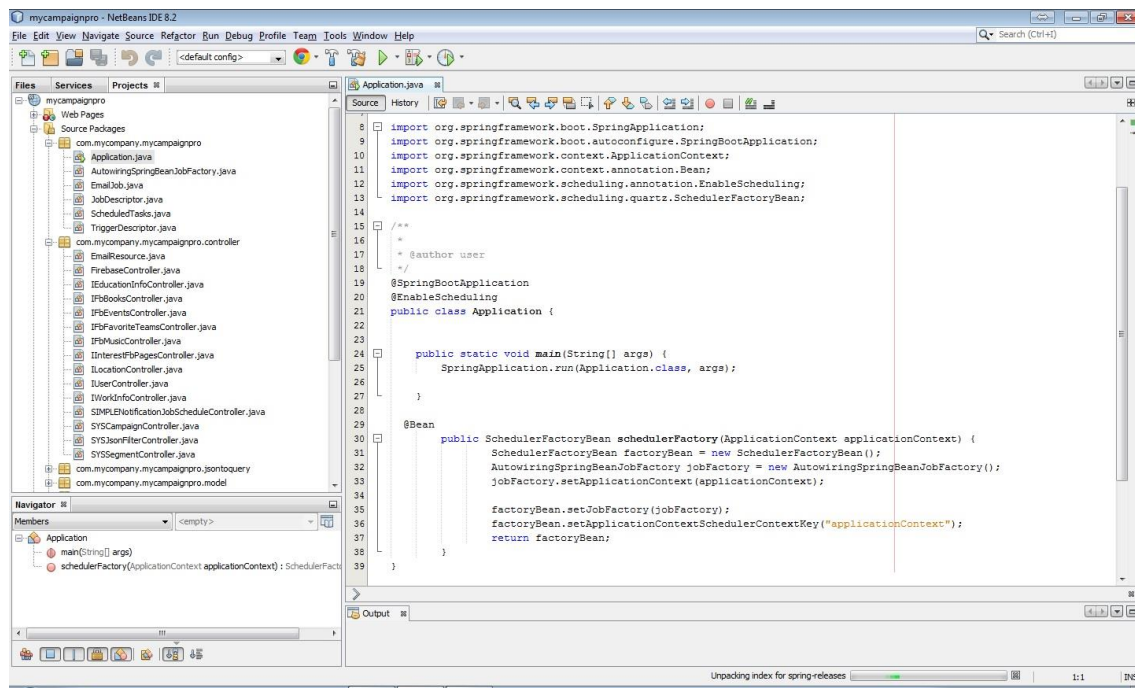
Figure 27: Implement Tools - Netbeans

5.2.2   Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for Windows, macOS and Linux based operating systems. It is replacement for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

The Android Studio provides code editors that help the developer write correct and structured Java code for the Android Application. Android Studio uses [Gradle](#), an advanced build toolkit, to automate and manage the build process, while allowing you to define flexible custom build configurations. The developer has the ability to define many dependencies in order to include a lot of libraries which are going to be used.

Also, the IDE provides a layout editor where the designer can quickly build different views by dragging UI elements into a visual design editor or write the layout in XML by hand. On the design editor, it is possible to present the view of different devices and versions, offering the designers the opportunity to resize the layout so as to be sure that every view works well on the majority of popular screen resolutions.
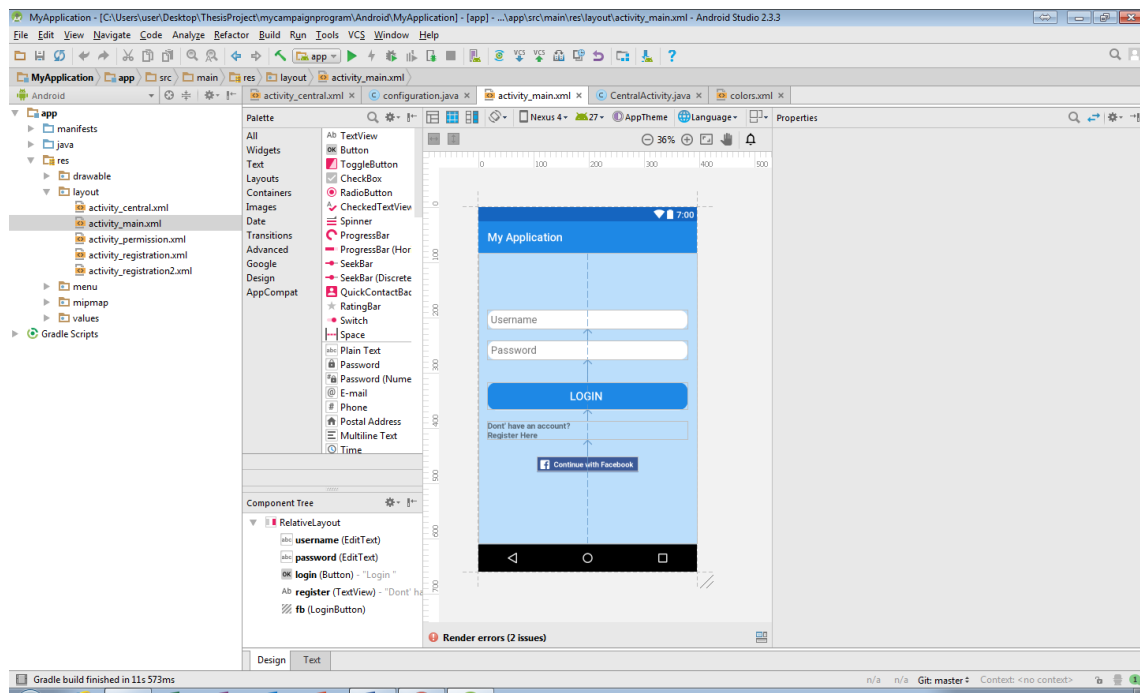
Figure 28: Implementation and design tools - Android Studio, layout editor

5.2.3   Microsoft Visio

Microsoft Visio is a diagramming and vector graphics application and is part of Microsoft Office family. This program supports many different types of diagrams. The diagrams apply to many different specialties. One of them is the software technology. Visio supports all the types of UML diagrams. A software engineer can design all the UML diagrams using the Visio. The UML diagrams help in the analysis of the use cases and the efficient design of the application. With the use of Visio, a software engineer can design the below UML diagrams:

- Class diagrams
- Object diagrams
- Component diagrams

- Deployment diagrams
- Use case diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams
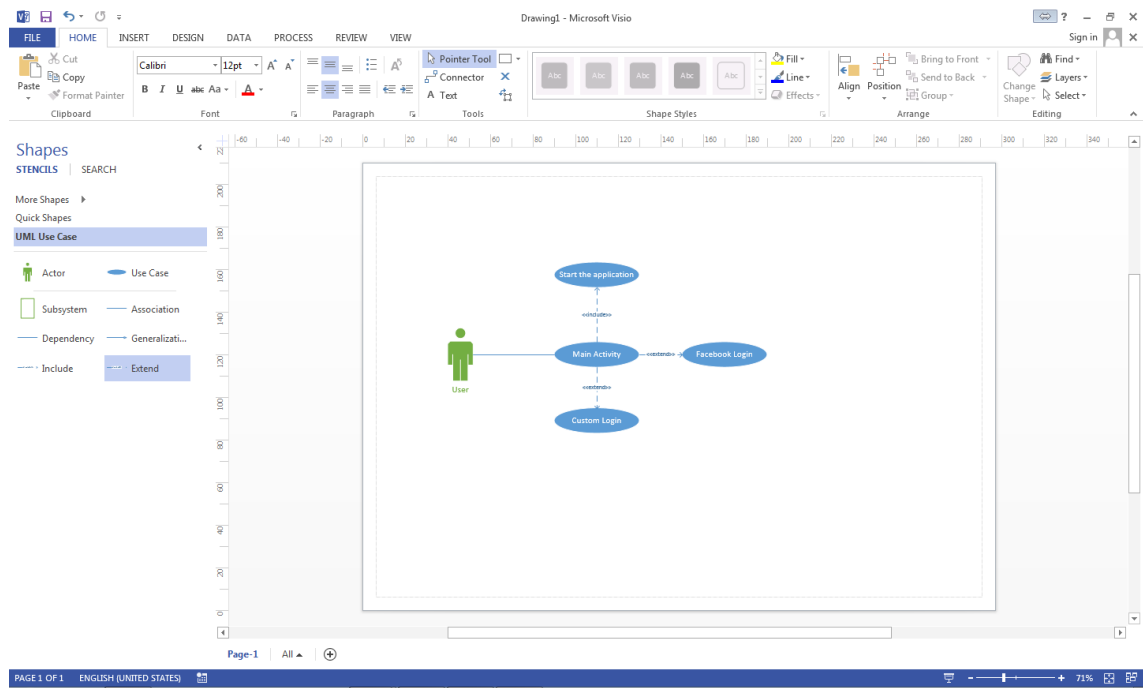
Figure 29: Implement and Design Tools - Microsoft Visio, use case diagram

5.2.4   Git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system it is aimed at speed, data integrity and support for distributed, non-linear workflows.

Git provides developers with the ability to save their work in separate versions. This is helpful because many times a specific change or a new feature in the code might affect the rest of the project's functionality. Git offers the option code revert in case a fallback to a previous version is required. Furthermore, two or more developers can work together on a project. The only needed action is to pull the last changes of the code, before the start of their work, and push the new changes at the end of their coding work. Git provides a mechanism which avoids conflicts on the committed code in case of two developers changing the same part of code.

For this dissertation a repository has been created on https://github.com.
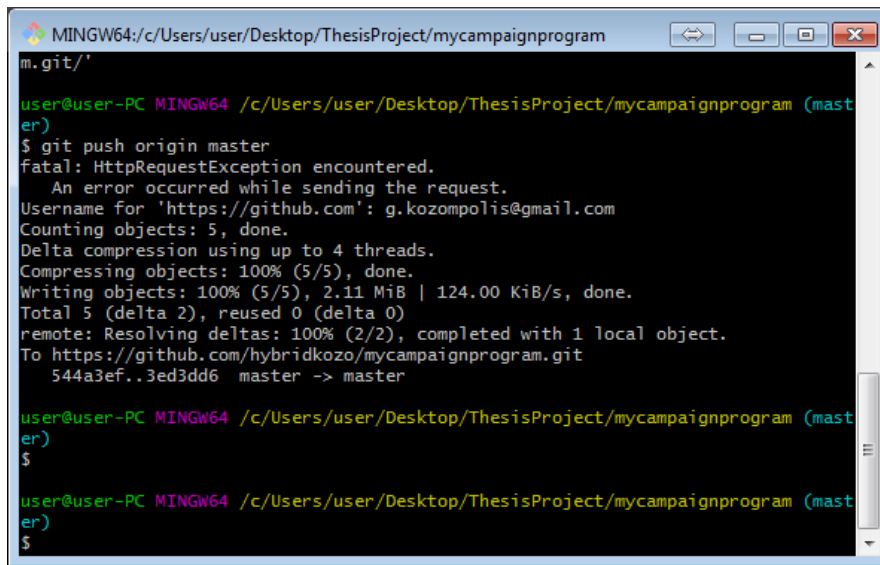
Figure 30: Implement and Design Tools - Git

5.2.5   **Genymotion**

Genymotion is an Android emulator, which supports a lot of mobile devices and Android OS versions. The emulator is used in order to run the developed applications and test their functionalities in multiple devices and OS versions. The emulation includes all the existing sensors from real devices like the location sensor.

The developer, or tester, can select the preferred device and OS version or create a custom device with a specific OS version. He just chooses the configured device and starts the virtual machine. Then he can install the APK and start the application from the virtual machine.
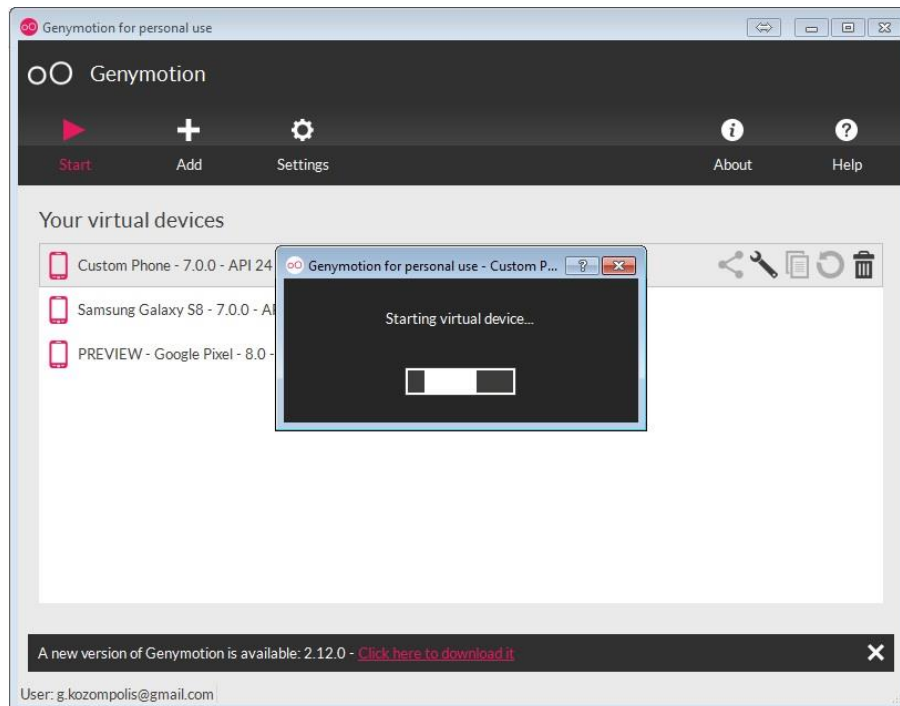
Figure 31: Implement and Design Tools - Genymotion

5.2.6   **pgAdmin 4**

pgAdmin is a management tool for PostgreSQL and derivative relational databases. It may be run either as web or desktop application. The desktop application has been used on windows for this dissertation.

pgAdmin application helps a database administrator connect and manage postgreSQL databases. It is possible to design new tables, columns and relations from the user interface without writing any SQL script at all. The administrator can access or update all the tables, the data, the relations, the sequences and everything else with high-level user experience.
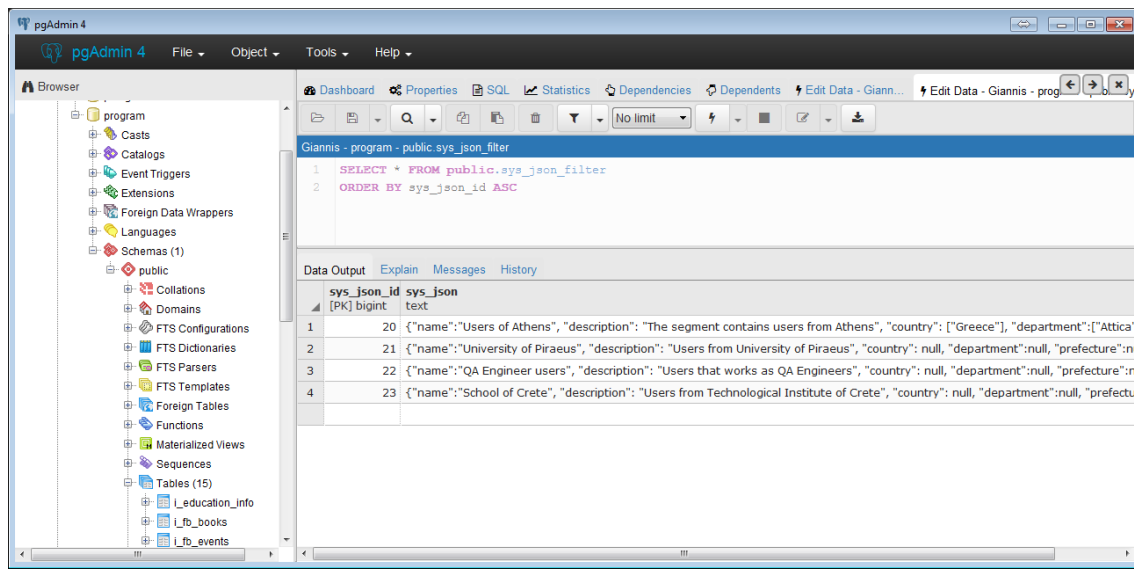
Figure 32: Implement and Design Tools - pgAdmin

5.2.7   **Postman**

It is a Google Chrome plug-in for API development & testing. You can build, test, and document your APIs faster and share your work with other team members. The main features of Postman are:

- Managing API collections, environments, tests and sharing
- Import & export API collections
- Extensive team collaboration tools
- Maintain history of API requests
- Extended API documentation & monitoring features

Postman was useful in order to test the APIs that served the communication between the android application and the web application.
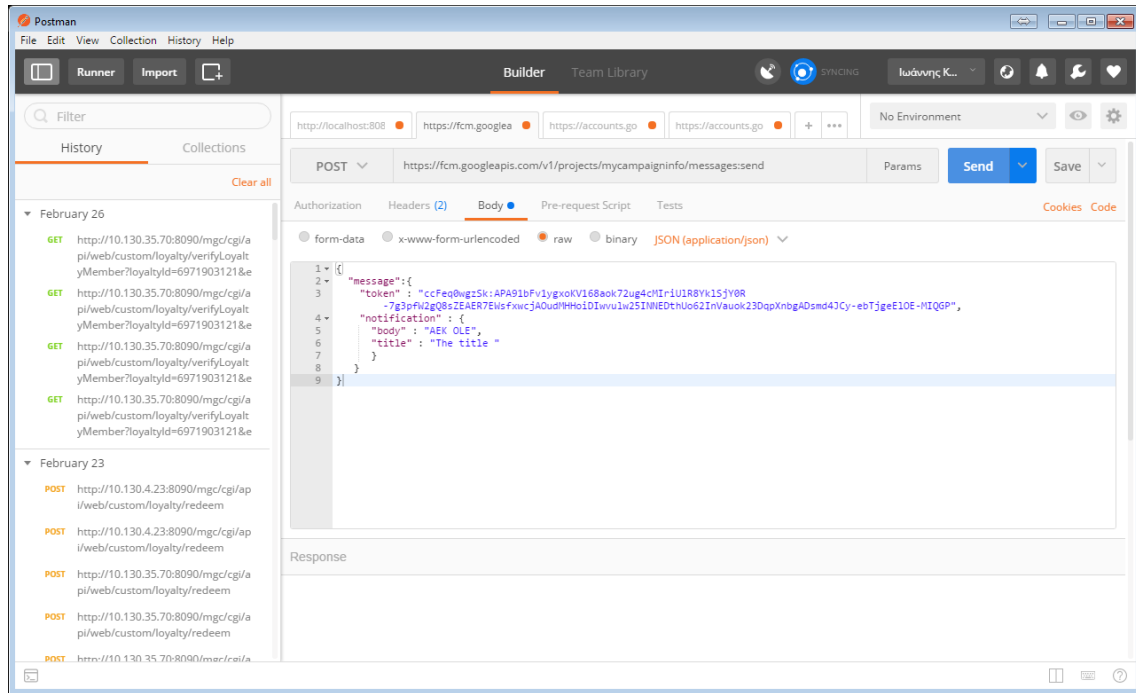
Figure 33: Implement and Design Tools - Postman

5.2.8   **Google Chrome – Developer Tools**

The Chrome Developer Tools (DevTools for short), are a set of web authoring and debugging tools built into Google Chrome. DevTools provide to web developers advanced access into the internals of the browser and their web application. Use the DevTools to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization.

The Dev Tools were used for inspecting HTML DOM elements and styles, checking errors on console and Javascript debugging.
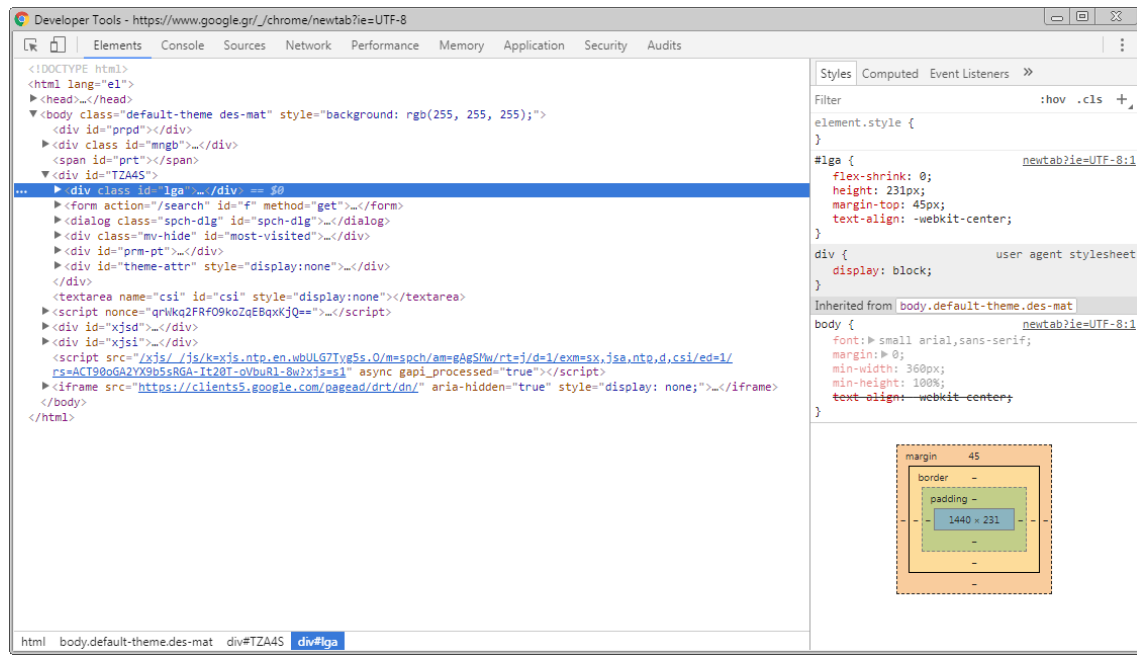
Figure 34: Implement and Design Tools - Chrome Developer Tools

5.3   Programming languages and Frameworks

5.3.1   Java

Java is a general-purpose computer-programming language that is class-based, object-oriented and specifically designed to have as few implementation dependencies as possible. The compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java Virtual machine (JVM) regardless of computer architecture. Reported that 9 million developers use the Java programming language and is one of the most popular programming languages particularly for client-server web applications. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

Java has been designed to be portable. This means that all programs written in Java, must run similarly on any combination of hardware and operating system with adequate runtime support. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to architecture-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be executed by a virtual machine (VM) written specifically for the host hardware.

The Java language has been used for both programming the android application and the web application of this project. The android application's graphic interface has been programmed on a basis of following the user's interactions and the necessary REST APIs called, so as to establish the communication with the web application, services, jobs and user authentication functionalities.

From the web application side, Java has been used in order to program the web the REST APIs, controllers, repositories and services.

5.3.2   **JavaScript**

JavaScript is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm. The JavaScript is one of the three core technologies of World Wide Web with HTML and CSS. This language enables interactive web pages and thus is an essential of web applications. The vast majority of websites use it and all major web browser have a dedicated JavaScript engine to execute it. Initially the JavaScript designed to run on client-side (web browsers) but now is embedded to run for server-side purposes.

This language has been used for the client side of the admin tool. By using JavaScript, the elements change dynamically according to current user interactions. JavaScript has the ability to change the HTML DOM file elements of a web page after user interactions that might take place, handled events and/or programmatic actions. Moreover, JavaScript is used to request data from the web services and then process the response on client-side view, by changing dynamically the web page content.

5.3.3   **Bootstrap**

Bootstrap is a free and open source front end development framework for the creation of websites and web apps. The Bootstrap framework is built on HTML, CSS, and JavaScript to facilitate the development of responsive, mobile-first sites and apps.

Responsive design makes it possible for a web page or app to detect the visitor's screen size and orientation and automatically adapt the display accordingly. The mobile first approach assumes that smartphones, tablets and task-specific mobile apps are employees' primary tools for getting work done and addresses the requirements of those technologies in design.

The web page of the admin tool has been designed with the use of a free Bootstrap Template. The framework offers a responsive template with high user experience. By using the Bootstrap template, the buttons, the inputs, the forms, the tables and the navigation menu are more effective without the need of specific CSS configurations and JavaScript scripts for more animation effectiveness.

5.3.4   **Maven**

Maven is a build automation tool which is used primarily for java projects. The maven describes how software is built and which dependencies are used. An XML file is used to define how the project is being built, the dependencies on other external modules and components, the build order, directories and the required plug-ins.

At the build process maven can download dynamically Java libraries and Maven plug-ins from one or more repositories such as Maven 2 Central Repository and store them in a local cache. The dependencies are defined in XML file named POM (Project Object Model). The POM file also configured with project name and plugin configuration. One can define a compiler-plugin that will use Java 1.5 for compilation.

Maven has been used in the web application in order to define the project build process. The POM file is inserted in the home directory of the project. All the needed plugins, dependencies, repositories and project information contained in the POM file. The maven facilitates the implementation by organizing the whole built process in a file. The developer can define a dependency for a specific library in three lines. The maven dynamically downloads the current library in the local cache on the built process. If the developer needs to update a library with new version, the only thing is needed, is to change the dependency version definition and re-built the project.

5.3.5   **Spring Framework**

Spring is an open source framework which simplifies Java development. Spring was created to address the complexity of enterprise application development and make it possible to use plain-

vanilla JavaBeans to achieve things that were previously only possible with EJB. But Spring's usefulness isn't limited to server-side development. Any Java application can benefit from Spring in terms of simplicity, testability, and loose coupling.

To back up its attack on Java complexity, Spring employs four key strategies:

- Lightweight and minimally invasive development with POJOs
- Loose coupling through DI and interface orientation
- Declarative programming through aspects and common conventions
- Eliminating boilerplate code with aspects and templates

The Spring framework supports dependency injection. Dependency injection is a technique whereby one object supplies the dependencies to another object. A dependency is an object that can be used (service). An injection is the passing of a dependency to a depended object(client) that would use it. The service is made part of the client's state. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

The Spring Framework adopts annotations in order to configure the behaviors which are provided. The Spring Framework was largely controlled through XML configuration. Below is a set of annotations that have been used for this dissertation:

- @Autowired
- @EnableAutoConfiguration
- @SpringBootApplication
- @RestController
- @RequestMapping
- @RequestBody
- @RequestParam

The dependency injection design pattern has been applied on the implementation. Specifically, the annotation @Autowired used directly on properties.

The @EnableAutoConfiguration annotation is usually placed on the main application class. The annotation implicitly defines a base "search package". This annotation tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.

The @SpringBootApplication annotation is used on the application class while setting up a Spring Boot project. The class that is annotated with the @SpringBootApplication must be kept in the base package. The one thing that the @SpringBootApplication does is a component scan. But it will scan only its sub-packages. As an example, if you put the class annotated with @SpringBootApplication in com.example, then @SpringBootApplication will scan all its sub-packages, such as com.example.a, com.example.b, and com.example.a.x.

The @RestController annotation is used at the class level. The annotation marks the class as a controller where every method returns a domain object instead of a view. By annotating a class with this annotation, you no longer need to add @ResponseBody to all the RequestMapping methods. It means that you no long use view-resolvers or send HTML in response. You just send the domain object as an HTTP response in the format that is understood by the consumers, like JSON.

The @RequestMapping annotation is used on both the class and method level. This annotation is used to map web requests onto specific handler classes and handler methods. When @RequestMapping is used on the class level, it creates a base URI for which the controller will be used. When this annotation is used on methods, it will give you the URI on which the handler methods will be executed. From this, you can infer that the class level request mapping will remain the same whereas each handler method will have their own request mapping.

The @RequestBody annotation is used to annotate request handler method arguments. The annotation indicates that a method parameter should be bound to the value of the HTTP

request body. The HttpMessageConveter is responsible for converting from the HTTP request message to object.

The @RequestParam annotation is used to annotate request handler method arguments. Sometimes you get the parameters in the request URL, mostly in GET requests. In that case, along with the @RequestMapping annotation, you can use the @RequestParam annotation to retrieve the URL parameter and map it to the method argument. The@RequestParam annotation is used to bind request parameters to a method parameter in your controller.

Below you can see a code sample which uses the annotation described above. The class IUserController has been implemented to response on requests that concern user data operations like return list with all users, return a user with specific username, register a user and validate user account. The annotation @RestController defines that the class is a controller, inside which, methods that will reply on specific HTTP requests will be defined. The @Autowired annotation applies dependency injection, injecting the dependencies from the objects I_User_Repository and UserServiceImpl. The @RequestMapping annotation is used on all methods to defile the URI that every method will response. The @RequestParam is used in the method arguments to handle the parameters from HTTP GET requests and the @RequestBody annotation is used in method arguments, in order to handle the request body from HTTP POST requests:

```java
@RestController
public class IUserController {
  @Autowired
  I_User_Repository iUserRepository;
  @Autowired
  UserServiceImpl userServiceImpl;

  @RequestMapping("/getAllUsers")
  public List<I_User> getAllUsers(){

    return iUserRepository.findAll();
  }

   @RequestMapping("/getByUsername")
  public ResponseObject getUserByUsername(@RequestParam String username){
    I_User i_User=iUserRepository.find(username);

    if (i_User==null){

      ResponseObject responseObject = new ResponseObject("fail", i_User);
      return responseObject;
    }else {
      ResponseObject responseObject = new ResponseObject("success",i_User);
      return responseObject;
    }
  }
  @RequestMapping("saveUserCredentials")
  public String saveUserCredentials(@RequestParam String username, String password){
    iUserRepository.save(new I_User(username, password));
    return "Done";
  }

  @RequestMapping("/validateUsername")
  public String validateUsername(@RequestParam String username){
    return userServiceImpl.getUsernameValidation(username);
  }

  @RequestMapping("/validatePassword")
  public String validatePassword(@RequestParam String password){
    return userServiceImpl.getPasswordValidation(password);
  }

  @RequestMapping("/validateAccount")
  public ResponseObject validateAccount(@RequestBody I_User i_User){
    return userServiceImpl.validateAccount(i_User.getI_username(),i_User.getI_password());
  }

  @RequestMapping("/register")
  public ResponseObject registerUser(@RequestBody I_User i_User){
    iUserRepository.save(new
    I_User(i_User.getI_first_name(),i_User.getI_last_name(),i_User.getI_mobile_number(),i_User.getI_username(),i_U
    ser.getI_password(),i_User.getI_email_address(),i_User.getI_register_type()));
   return new ResponseObject("success",i_User);
  }
 }
```

43

5.3.6   Android SDK

The Android SDK (software development kit) is a set of development tools used to develop applications for Android operating system. The Android SDK is updated for every version of Android OS and includes the following components:

- Platform-tools
- Build-tools
- SDK-tools
- The Android Debug Bridge (ADB)
- Android Emulator

Google updates the Android OS with new versions (KitKat, Lollipop, Marshmallow, Nougat, Oreo, Android P). When a new version is released, Google also updates the Android SDK version to supports the features for the new Android OS version. So, the developers should use the corresponding Android SDK, according what Android OS the application is implemented to run.

However, there are many different programming languages and IDEs that can be used to create an Android Application, the SDK is a constant. SDK provides a selection of tools required to build Android apps or to ensure the process goes as smoothly as possible. The Android Studio bundled with the Android SDK. The installation of the Android SDK was the first thing before the start of the implementation. The Android Studio provides interface which allows the developer to download and install the needed Android SDK version.

The Android Studio uses the Android SDK in order to provide many SDK-tools, which are necessary to create the APK – turning the Java program into an Android app that can be launched on a phone. The Android Device Monitor is a tool on Android studio that allow to check status about the Android device.

The Build tools were once categorized under the same heading as the Platform tools but have since been decoupled so that they can be updated separately. As the name suggests, these are also needed to build your Android apps. This includes the zipalign tool for instance, which optimizes the app to use minimal memory when running prior to generating the final APK, and the apksigner which signs the APK for subsequent verification.

The Platform tools are more specifically suited to the version of Android that you want to target. Generally, it is best to install the latest Platform tools, which will be installed by default. After first installation though, you need to keep your Platform-tools constantly updated.

The Android Debug Bridge (ADB) is a program that allows the communication with any Android device. It relies on Platform-tools in order to understand the Android version that is being used on said device and hence it is included in the Platform-tools package. ADB can be used to access shell tools such as logcat, to query your device ID or even to install apps.

The Android emulator is what lets the apps testing and the apps monitoring on a PC, without necessarily needing to have a device available. To use this, is needed also to get an Android system image designed to run on PC hardware. The Android Virtual Device manager can be used in order to choose the preferred Android OS to emulate, along with the device specifications (screen size, performance etc.).

5.4   Database Architecture

5.4.1   PostgreSQL

PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from

other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications.

PostgreSQL was designed to run on UNIX-like platforms. However, PostgreSQL was then also designed to be portable so that it could run on various platforms such as Mac OS X, Solaris, and Windows. Below is an inexhaustive of various features found in PostgreSQL, with more being added in every major release.

Data Types

- Primitives: Integer, Numeric, String, Boolean
- Structured: Date/Time, Array, Range, UUID
- Document: JSON/JSONB, XML, Key-value (Hstore)
- Geometry: Point, Line, Circle, Polygon
- Customizations: Composite, Custom Types

Data Integrity

- UNIQUE, NOT NULL
- Primary Keys
- Foreign Keys
- Exclusion Constraints
- Explicit Locks, Advisory Locks

Concurrency, Performance

- Indexing: B-tree, Multicolumn, Expressions, Partial
- Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Bloom filters
- Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
- Transactions, Nested Transactions (via savepoints)
- Multi-Version concurrency Control (MVCC)
- Parallelization of read queries
- Table partitioning
- All transaction isolation levels defined in the SQL standard, including Serializable

Reliability, Disaster Recovery

- Write-ahead Logging (WAL)
- Replication: Asynchronous, Synchronous, Logical
- Point-in-time-recovery (PITR), active standbys
- Tablespaces

Security

- Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
- Robust access-control system
- Column and row-level security

Extensibility

- Stored procedures
- Procedural Languages: PL/PGSQL, Perl, Python (and many more)
- Foreign data wrappers: connect to other databases or streams with a standard SQL interface
- Many extensions that provide additional functionality, including PostGIS

Internationalisation, Text Search

- Support for international character sets, e.g. through ICU collations

- Full-text search

5.4.2   **Database Schema**

The database schema consists of fourteen tables and its relationships. The tables are used to store data from both the android and web application. The tables below are used to store user data like account information, information retrieved from Facebook and location sensor data:

- I_USER
- I_EDUCATION_INFO
- I_FB_BOOKS
- I_FB_EVENTS
- I_FB_FAVORITE_EVENTS
- I_FB_MUSIC
- I_INTEREST_FB_PAGES
- I_WORK_INFO
- I_LOCATION

The below tables are used to save information about the segmentation and campaign configuration:

- SYS_SEGMENT_CLIENT
- SYS_SEGMENT
- SYS_JSON_FILTER
- SYS_CAMPAIGN
- SYS_FLOW_NODE
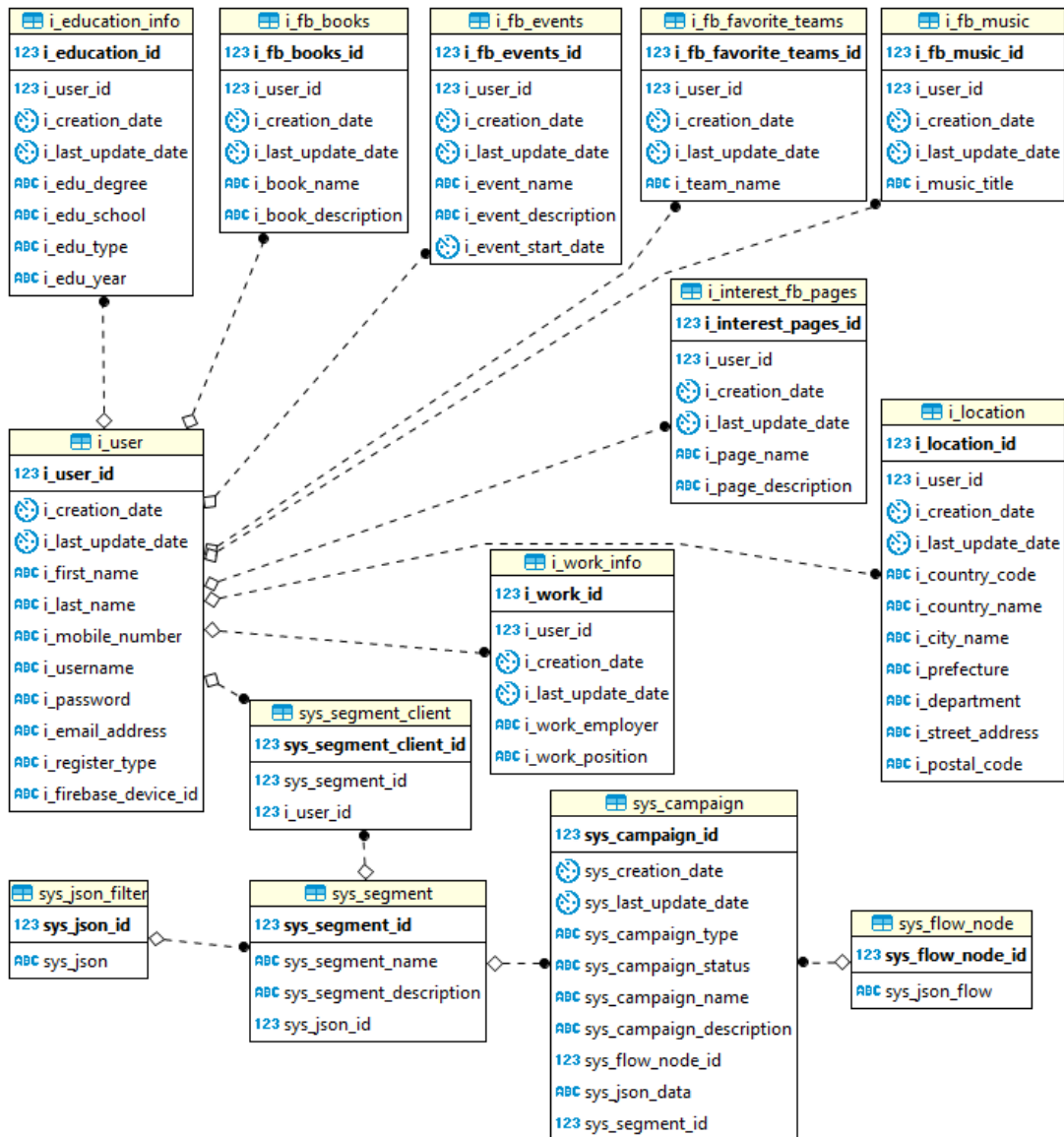
Figure 35: Database Architecture – Schema

The table I_USER has "one to many" relationship with the tables I_EDUCATION_INFO, I_FB_BOOKS, I_FB_EVENTS, I_FB_FAVORITE_EVENTS, I_FB_MUSIC, I_INTEREST_FB_PAGES, I_WORK_INFO, I_LOCATION and SYS_SEGMENT_CLIENT. This table holds user's account credentials (username and password), personal information (name, surname, mobile, email), register channel and unique firebase device id.

The table I_EDUCATION_INFO has "many to one" relationship with the table I_USER. This table holds information about user's education, which is retrieved from Facebook Graph API.

The table I_FB_BOOKS has "many to one" relationship with the table I_USER. This table holds user's favorite books information, which is retrieved from Facebook Graph API.

The table I_FB_EVENTS has "many to one" relationship with the table I_USER. This table holds the events that the user prefers, which are retrieved from Facebook Graph API.

The table I_FB_FAVORITE_TEAMS has "many to one" relationship with the table I_USER. This table holds user's favorite sport-teams information, which is retrieved from Facebook Graph API.

The table I_FB_MUSIC has "many to one" relationship with the table I_USER. This table holds user's favorite music, which is retrieved from Facebook Graph API.

The table I_INTEREST_FB_PAGES has "many to one" relationship with the table I_USER. This table holds user's favorite Facebook pages information, which is retrieved from Facebook Graph API.

The table I_WORK_INFO has "many to one" relationship with the table I_USER. This table holds user's work experience, which is retrieved from Facebook Graph API.

The table I_LOCATION has "many to one" relationship with the table I_USER. This table holds the last known user's location, which is retrieved from GPS sensor on mobile device.

The table SYS_SEGMENT_CLIENT has "many to one" relationship with the table I_USER and "many to one" relationship with table SYS_SEGMENT.

The table SYS_SEGMENT has "many to one" relationship with the table SYS_JSON_FILTER. This table holds the name, description and the json filter that will be used.

The table SYS_JSON_FILTER holds information in json type that describes the segment. This json is processed from the system in order to create the SQL query which selects the targeted users.

The table SYS_CAMPAIGN has "many to one" relationship with the table SYS_SEGMENT. This table holds information about the campaign name and description, campaign type, campaign status and data like execution time, notification title and body saved in JSON file.

5.5   Use Case Diagrams

At this point you may take a closer look at the diagrams that represent the use cases of this project. The first diagram describes the possible scenarios deriving from the user's navigation within the android application. The second diagram describes how a user creates a new segment within the admin tool of the web application and the third one describes how a campaign is configured within the same tool.
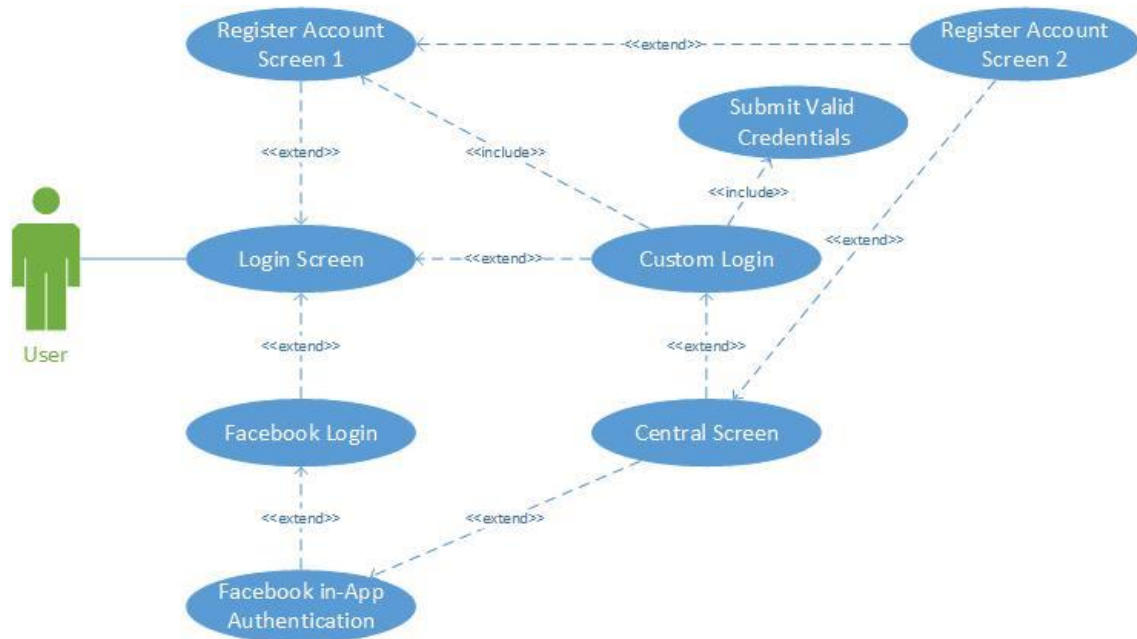
Figure 36: Use Case Diagram for android application navigation
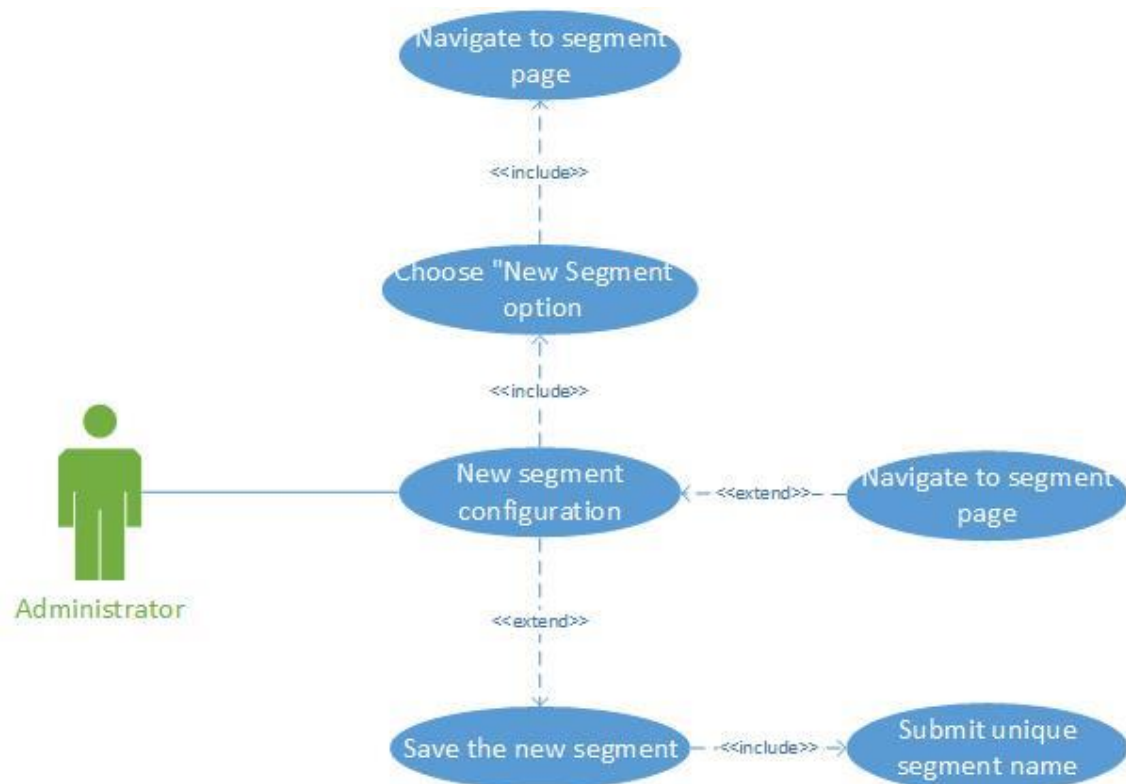
Figure 37: Use Case Diagram for new segment configuration
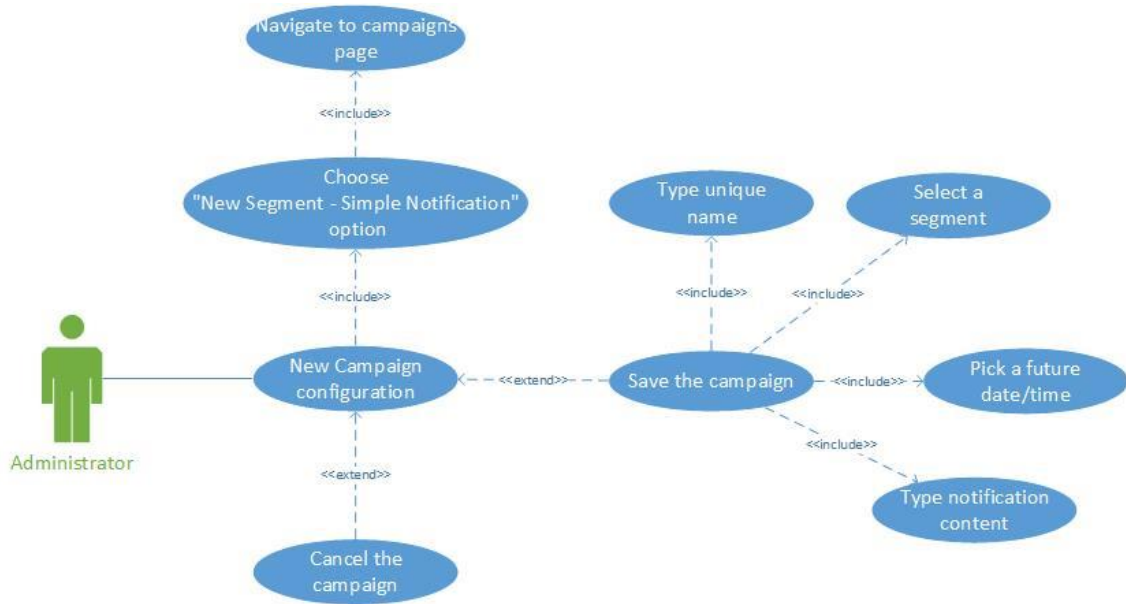
Figure 38: Use Case Diagram for new campaign configuration

5.6   Sequence Diagrams

The first of the following diagrams describes the entire flow of Facebook login that is used on the android application. The second diagram describes the complete flow of the creation of a new segment within the admin tool. The last diagram describes the flow of the creation, activation and final execution of a campaign.
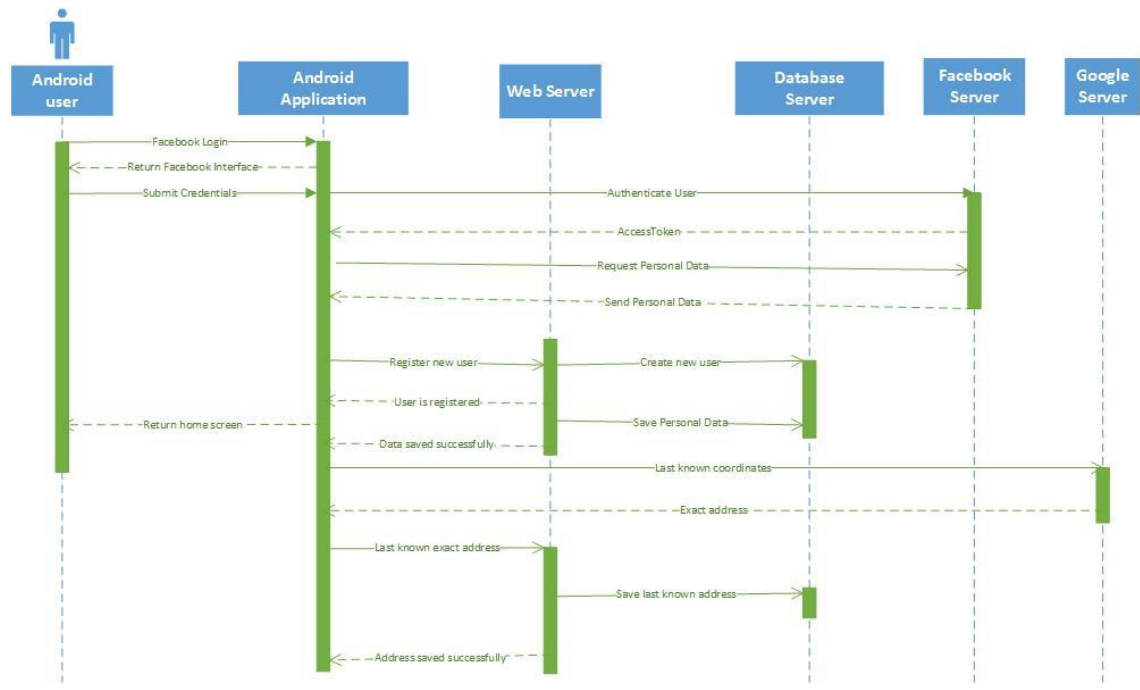
Figure 39: Sequence Diagram for Facebook login
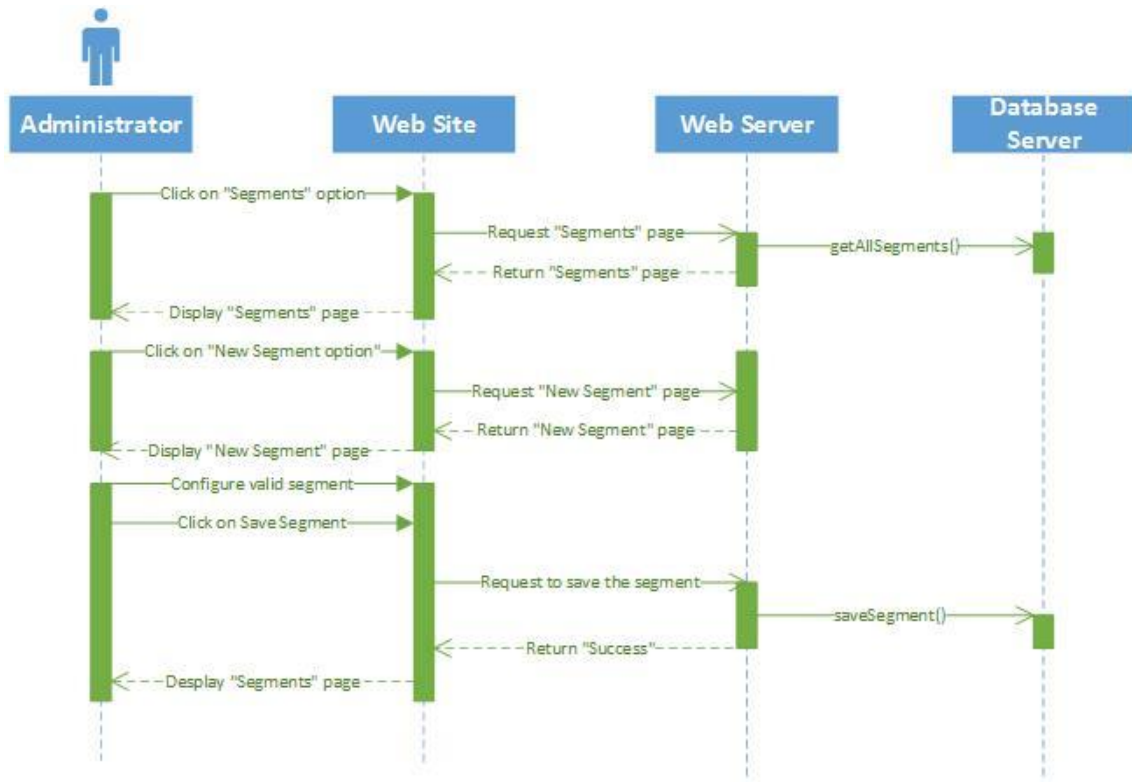
Figure 40: Sequence Diagram for new segment
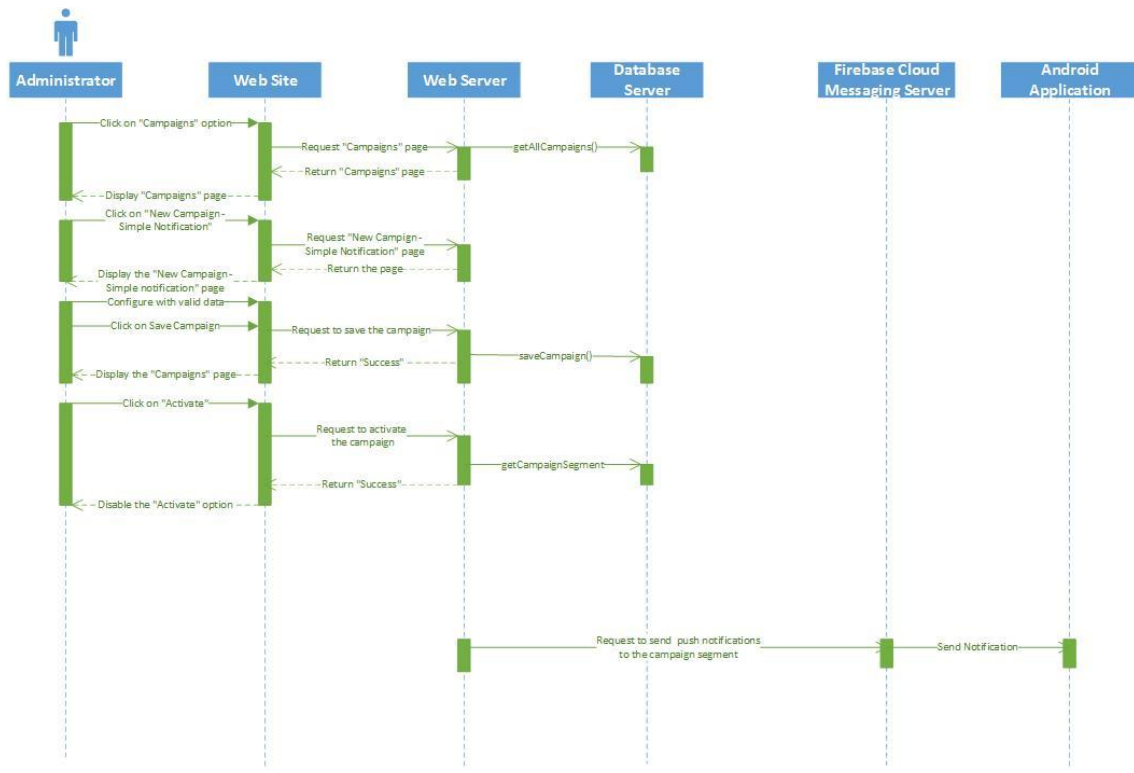
Figure 41: Sequence Diagram for new campaign

5.7   Important Code Sections

5.7.1   Custom location service on the android application

The custom location service has been implemented on the android application. The implemented service is running in the background of the application and it is started after the user's successful login. This service recognizes when the GPS sensor is enabled on the mobile device. Then holds the last known coordinates whenever the GPS sensor recognizes changes on location. Every ten seconds the service saves locally the last recognized coordinates, while the GPS sensor is on. The service stops automatically this process when the GPS sensor is off. An alarm has been implemented to run every day and schedule a job, which uploads the last known user's location on the remote database, when the mobile device has access on the internet via REST API, in order for the web application to be able to process the last known user's locations.

The above described functionality has been implemented in the GPSService class of the android application source code. This class extends the IntentService class that provides a straightforward structure for running an operation on a single background thread. This allows it to handle long-running operations without affecting your user interface's responsiveness. Also, an IntentService is not affected by most user interface lifecycle events, so it continues to run in circumstances that would shut down an AsyncTask.

```
public class GPSService extends
IntentService{
        ……
        ………
        …………
```

In GPSService class methods that overrides the methods onStartCommand(Intent intent, int flags, int startId) and onDestroy() are defined. The GPSService class should also be defined in the manifest file of the android application as a service:

```
<service android:name=".services.GPSService"
```

The onStartCommand method is executed when the service is started. The method first of all, checks if the permissions for the GPS sensor are granted. If the permissions are not granted, nothing is going to happen and a message informs the user that he does not have permissions. If the permissions are granted, then the method initializeLocationManager() is called, a method which provides access to the system location services. These services allow the application to obtain periodic updates of the device's geographical location. The initialized location manager uses the requestLocationMethod in order to get the last location from the GPS provider, every LOCATION_INTERVAL time, every LOCATION_DISTANCE meters and using a defined listener (The Location Listener is described next). Then if location listener exists, the method searchNewLocation is called, which sets a handler every ten seconds to save the coordinates on Shared Preferences (Application Cache)

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {

    Toast.makeText(getApplicationContext(),"Starting
service...",Toast.LENGTH_LONG).show();
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED&&ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

        Toast.makeText(getApplicationContext(),"Permissions are not
granted...",Toast.LENGTH_LONG).show();
        sharedPreferences =
getSharedPreferences(GpsPREFERENCES,Context.MODE_PRIVATE);

    }else {

        initializeLocationManager();
        sharedPreferences =
getSharedPreferences(GpsPREFERENCES,Context.MODE_PRIVATE);

        try {
            mLocationManager.requestLocationUpdates(
                    LocationManager.GPS_PROVIDER, LOCATION_INTERVAL,
LOCATION_DISTANCE,
                    mLocationListeners[0]);
        } catch (java.lang.SecurityException ex) {
            Log.v(TAG, "fail to request location update, ignore", ex);
        } catch (IllegalArgumentException ex) {
            Log.v(TAG, "gps provider does not exist " + ex.getMessage());
        }
    }
    if(mLocationListeners[0]!=null){
        searchNewLocation();
    }

    return Service.START_NOT_STICKY;
}
```

The onDestroy method is executed when the GPSService is no longer used and it destroys it. The stopSearchNewLocation and stopLocationListeners methods are called when the service

is destroyed. The method stopSearchNewLocation() stops the handler, which is running every 10 seconds and saves locally the last known location. The method stopLocationListeners removes all location updates for the specified location listener:

```java
private void stopSearchNewLocation(){
    handler.removeCallbacksAndMessages(null);
}

private void stopLocationListeners(){
    Log.v(TAG, "onDestroy");
    super.onDestroy();
    if (mLocationManager != null) {
        try {
            mLocationManager.removeUpdates(mLocationListeners[0]);
        } catch (Exception ex) {
            Log.v(TAG, "fail to remove location listners, ignore", ex);
        }

    }
}
```

The Location Listener is defined as an inner class of the GPSService class with the name LocationListener and implements the android.location.LocationListener. The inner class implements methods which override the methods   onLocationChanged onProviderDisabled, onProviderEnabled and onStatusChanged. The onLocationChanged method handles location changes, whenever those may happen, and saves the new location. The onProviderDisabled method recognizes every time that the GPS sensor is disabled, in order to stop the configured handler, which saves locally the last known location. The method onProviderEnabled recognizes when the GPSSensor is enabled in order to start the handler, which saves locally the last known location:

```java
    private class LocationListener implements android.location.LocationListener
{

  public LocationListener(String provider)
  {
    Log.v(TAG, "LocationListener " + provider);
    mLastLocation = new Location(provider);
  }

  @Override
  public void onLocationChanged(Location location)
  {
    Log.v(TAG, "onLocationChanged: " + location);
    mLastLocation.set(location);
  }

  @Override
  public void onProviderDisabled(String provider)
  {
    Log.v(TAG, "onProviderDisabled: " + provider);
    stopSearchNewLocation();
    Toast.makeText(GPSService.this, "Stop searching for new locations...", Toast.LENGTH_SHORT).show();
  }

  @Override
  public void onProviderEnabled(String provider)
  {
    Log.v(TAG, "onProviderEnabled: " + provider);
    searchNewLocation();
    Toast.makeText(GPSService.this, "Start searching for new locations...", Toast.LENGTH_SHORT).show();
  }

  @Override
  public void onStatusChanged(String provider, int status, Bundle extras)
  {
    Log.v(TAG, "onStatusChanged: " + provider);
```

The above functionality describes the custom service location which recognizes, whenever the GPS Sensor is enabled and, every 10 seconds, saves locally the last known location. However, this functionality is not enough in order for the Android Application to update the remote database with these data. For this reason, a job has been implemented, which is running every day and updates the remote database when the internet connection is established.

The JobScheduler API is used in order to schedule the job, when the Internet connection is established. The job is scheduled when a receiver handles broadcast intents. So, the class JobReceiver has been implemented to extend the BroadcastReceiver. The onReceive method is executed, when broadcast intents are handled for the JobReceiver class. Then the job is scheduled to run using the JobScheduler API:

```java
    public class JobReceiver extends BroadcastReceiver {
    JobScheduler jobScheduler;

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Alarm is running....", Toast.LENGTH_LONG).show();
        jobScheduler = (JobScheduler) context.getSystemService(context.JOB_SCHEDULER_SERVICE);
        jobScheduler.schedule(new JobInfo.Builder(1,
            new ComponentName(context, JobServiceLocation.class))
            .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)
            .build());
```

The class JobServiceLocation has been implemented to execute the scheduled job. The class overrides the method onStartJob, which checks if the location coordinates have been saved locally and the user is in sign-in mode, in order to update the remote database. The last known location is updated on the remote database by requesting a specific web service. The method InsertLocation requests the web service with the appropriate parameters:

```java
@Override
public boolean onStartJob(JobParameters jobParameters) {

    sharedPreferences = getSharedPreferences(GPSService.GpsPREFERENCES,
Context.MODE_PRIVATE);
    sharedPreferences2 = getSharedPreferences(MainActivity.MyPREFERENCES,
Context.MODE_PRIVATE);
    Toast.makeText(getApplicationContext(),"The job has been
started",Toast.LENGTH_LONG).show();
    // Start task to pull work out of the queue and process it.
    if(sharedPreferences.getString("latitude",
null)!=null&&sharedPreferences2.getString("username",null)!=null){

InsertLocation(APIURL(getAPIParameters(Double.valueOf(sharedPreferences.getString("la
titude", null)), Double.valueOf(sharedPreferences.getString("longitude", null)))));
    }
    return false;
}
```

Below is the InsertLocation method which uses the Mysingleton design pattern in order to request via GET method:

```java
    private void InsertLocation(String URL){

  JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.GET, URL, null, new
Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
      try {
        JSONObject jsonObject = response.getJSONObject("object");

      } catch (JSONException e) {
        e.printStackTrace();
      }
    }
  }, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
      Toast.makeText(getApplicationContext(),"Something went wrong",Toast.LENGTH_LONG).show();
    }
  });

  Mysingleton.getInstance(getApplicationContext()).addToRequestque(jsonObjectRequest);
```

The above method takes the requested URL as an argument. Since, the web service handles data like city, country, postal code, prefecture and department, the location coordinates should be converted to an address with these information and then these data should be passed on to the web service. For these reason, the getAPIParameters method is used in order to convert the coordinates into a real address, which is then saved in a array of strings. Then a method named APIURL accepts as argument this array and convert it to the requested URL, which is passed on to the InsertLocation method. The getAPIParameters method uses the Geocoder class from the Android framework location APIs, which can convert a geographic location into an address. The address lookup feature is also known as reverse geocoding:

```java
private String[] getAPIParameters(double lat, double lon) {

    String errorMessage, TAG = "GET ADDRESS: ",CountyCode= null, CountryName= null,
FeatureName= null, PostalCode= null, SubAdminArea = null, AdminArea= null,Address =
null;
    Geocoder geocoder = new Geocoder(getApplicationContext(), Locale.getDefault());
    List<Address> addresses = null;

    try {
        addresses = geocoder.getFromLocation(lat, lon, 10);
    } catch (IOException ioException) {
        // Catch network or other I/O problems.
        errorMessage = "The service is not available";
        Log.e(TAG, errorMessage, ioException);
    } catch (IllegalArgumentException illegalArgumentException) {
        // Catch invalid latitude or longitude values.
        errorMessage = "Invalid latitude used...";
        Log.e(TAG, errorMessage + ". " +
                "Latitude = " + lat +
                ", Longitude = " +
                lon, illegalArgumentException);
    }
    // Handle the case where no address is found.
    if (addresses == null || addresses.size() == 0) {
            errorMessage = "The address is not found...";
            Log.e(TAG, errorMessage);
    } else {
        Address address = addresses.get(4);
        Address address1 = addresses.get(0);
        Address address2 = addresses.get(0);
        CountyCode = address.getCountryCode();
        CountryName = address.getCountryName();
        FeatureName = address.getFeatureName();
        PostalCode = address1.getPostalCode();
        SubAdminArea = address.getSubAdminArea();
        AdminArea = address.getAdminArea();
        ArrayList<String> addressFragments = new ArrayList<String>();
        // Fetch the address lines using getAddressLine,
        // join them, and send them to the thread.
        for (int i = 0; i <= address2.getMaxAddressLineIndex(); i++) {
            addressFragments.add(address2.getAddressLine(i));
        }
        Address = addressFragments.get(0);
        Log.i(TAG, "The address has been found...");

    }
    String[] APIParameters = {CountyCode, CountryName, FeatureName, PostalCode,
SubAdminArea, AdminArea, Address};
    return APIParameters;
}

private String APIURL(String[] APIParameters){

    String URL = localURL + "saveLocation?" + "i_username=" +
sharedPreferences2.getString("username", null) + "&i_country_code=" +
APIParameters[0] + "&i_country_name=" + APIParameters[1] + "&i_city_name=" +
APIParameters[2] + "&i_prefecture=" + APIParameters[4] + "&i_department=" +
APIParameters[5] + "&i_street_address=" + APIParameters[6] + "&i_postal_code=" +
APIParameters[3];
    Log.v("CreateURL", URL);

    return URL;
}
```

Last but not least is how the JobReceiver handles a broadcast intent in order to schedule the job. This functionality has been implemented using the AlarmManager, which provides access to the alarm services. These allows you to schedule your application to be executed on some point in the future. When an alarm goes off, the Intent that had been registered for it, is broadcast by the system, automatically starting the target application if it is not already running. Registered alarms are retained while the device is asleep (and can optionally wake the device up if they go off during that time) but will be cleared if it is turned off and rebooted.

The class AlarmBroadcastReceiver uses the AlarmManager in order to create and send broadcast intents on specific dates/times. The startAlert method starts an alarm to send a broadcast intent every day. The class extends BroadcastReceiver in order to start the alarm after the device reboot.

```java
public final  class AlarmBroadcastReceiver extends BroadcastReceiver {

public static AlarmManager alarmManager;
PendingIntent pendingIntent;

@Override
public void onReceive(Context context, Intent intent) {
    startAlert(context);

    }

public void startAlert(Context context) {

    Intent intent = new Intent(context, JobReceiver.class);
    pendingIntent = PendingIntent.getBroadcast(context,123456,intent,0);
    alarmManager = (AlarmManager) context.getSystemService(context.ALARM_SERVICE);
    alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
SystemClock.elapsedRealtime() + 10000, 1000 * 10, pendingIntent);
    }

public void stopAlert(Context context){

    alarmManager.cancel(pendingIntent);

    }
```

5.7.2   Retrieve Facebook Data in Android Application

Let's assume that the user decides to use the login via Facebook. The application exploits the Facebook graph API and retrieves useful information for the web application. In this section is described how the application retrieves education information from the user and saves this in the remote database. By the same way, other information is retrieved from the user's Facebook profile.

Different methods have been implemented in the class facebookFunction, which are used to retrieve all the useful information from the user's Facebook profile. The method saveUserEducationInfo has been implemented to retrieve education information. GraphRequest class, provided from Android SDK, provides support for integrating with Facebook Graph API. This class contains the method newMeRequest, which is used to fetch the user data for the given access token. The response is a JSONObject which is processed in order to retrieve information regarding the user's degree, school, graduation year and type of education. The retrieved information is encoded into UTF8 format so as to avoid special characters issues in the requested URL. The saveData method is called in order save the information in the remote database.

```java
public void saveUserEducationInfo(AccessToken accessToken, final Context context) {

    GraphRequest request = GraphRequest.newMeRequest(accessToken,
            new GraphRequest.GraphJSONObjectCallback() {
                @Override
                public void onCompleted(JSONObject object, GraphResponse response) {

                    try {

                        String degree, school, type , year, URL;

                        if (object.has("education")) {
                            JSONObject deg, sch, yr,obj;
                            JSONArray jsonArray = object.getJSONArray("education");
                            for (int i = 0; i < jsonArray.length(); i++) {
                                 obj = jsonArray.getJSONObject(i);
                                if (obj.has("degree")) {
                                   deg = obj.getJSONObject("degree");
                                }else{
                                    deg = null;
                                }
                                if (obj.has("school")) {
                                    sch = obj.getJSONObject("school");
                                }else {
                                    sch = null;
                                }
                                 if (obj.has("year")){
                                     yr = obj.getJSONObject("year");
                                 }else {
                                     yr = null;
                                 }
                                if (obj.has("type")) {
                                    type = URLEncoder.encode(obj.getString("type"),"utf-8");
                                }else {
                                    type = null;
                                }
                                if(deg!=null){
                                    degree = URLEncoder.encode(deg.getString("name"),"utf-8");
                                }else{
                                    degree = null;
                                }
                                if(sch!=null) {
                                    school = URLEncoder.encode(sch.getString("name"),"utf-8");
                                }else{
                                    school = null;
                                }
                                if(yr!=null) {
                                    year = URLEncoder.encode(yr.getString("name"),"utf-8");
                                }else{
                                    year=null;
                                }
                                URL = URL_SAVE_EDUCATION_INFO + "?i_username=" + username +
"&i_edu_degree=" + degree + "&i_edu_school=" + school + "&i_edu_type=" + type + "&i_edu_year=" + year;
                                saveData(URL,methodGET,context,"The Education Info saved
successfully...","Something went wrong on saving Education Info...");
                            }

                        }else {

                            Log.v("FACEBOOK_FUNCTIONS", "No Education Info found from facebook....");
                        }

                    } catch (JSONException e) {
                        e.printStackTrace();
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }
            });
    Bundle parameters = new Bundle();
    parameters.putString("fields", "education");
    request.setParameters(parameters);
    request.executeAsync();

}
```

The saveData method accepts the requested URL, the method type, the success message and the fail message as arguments. By Using MySingleton design pattern, a web service is requested in order to save the user's education info in the remote database. On the response, the application handles and logs the status. If any education information exists in the user Facebook account, then the API returns the success message, else the API returns the fail message.

```java
public void saveData(String URL, int method, final Context context, final String successMessage, final String failMessage) {

    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(method, URL, null, new
Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            try {
                String status = response.getString("status");
                if (status.equals("success")){
                    Log.v("FACEBOOK_FUNCTIONS", successMessage);
                }else {
                    Log.v("FACEBOOK_FUNCTIONS", failMessage);
                }

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.v("ON_RESPONSE_VOLLEY: ", "Something went wrong on response... please try again..");

        }
    });

    Mysingleton.getInstance(context).addToRequestque(jsonObjectRequest);

}
```

5.7.3   Save Location - Web Service

The previous section describes how the Android Application requests web services in order to save the location information in the remote database. Now it is time to describe how the web service handles this kind of requests and sends the response back. The web application has been implemented with Spring Boot and runs on Tomcat Server, which is accessible via Internet connection.

All requests are handled by controllers, which have been implemented in the web application. The class ILocationController is a controller and includes the method saveLocation. This method handles requests in order to save location information for a specific user. This method handles GET requests under the "/saveLocation" URI with parameters i_username, i_country_code, i_country_name, i_city_name, i_prefecture, i_department, i_street_address and i_postal_code. The i_username is used in the method "find" from I_User_Repository in order to actually find the corresponding user in the database and return his data as I_User object. The method "save" from I_Location_Repository is used to save the location in the database. In the end, the web service responses with one ResponseObject object, which contains the response status and the corresponding location object. The client handles the response object in JSON format.

```
    @RequestMapping("/saveLocation")
    public ResponseObject saveLocation(@RequestParam String i_username, String
i_country_code, String i_country_name, String i_city_name, String i_prefecture, String
i_department, String i_street_address,String i_postal_code){

        I_User i_User = i_User_Repository.find(i_username);
        I_Location i_Location;
        i_Location_Repository.save(i_Location = new I_Location(i_User.getI_user_id(),
        i_country_code, i_country_name, i_city_name, i_prefecture, i_department,
        i_street_address, i_postal_code));

        return new ResponseObject("Success",i_Location);
```

The I_User and I_Location are objects that maps the corresponding tables in the database using the Java Persistence API (JPA). The I_Location_Repository and the I_User_Repository are interfaces which extends the CrudRepository. CrudRepository is a Spring Data interface. Spring provides CrudRepository implementation class automatically at runtime. It contains methods such as save, findById, delete, count etc. If we want to add extra methods, we need to declare it in our interface. The repositories are used in our controller with dependency injection without needed to initiate them by using the annotation @Autowired:

```
@RestController
public class ILocationController {

    @Autowired
    I_Location_Repository
i_Location_Repository;

    @Autowired
    I_User_Repository i_User_Repository;
......
.................
.........................
................................
```

In the I_User_Repository, the find method has been declared to return a user, which corresponds to a specific username. The @Query annotation defines the query that will be used in order to find the corresponding user:

```
public interface I_User_Repository extends CrudRepository<I_User, Long>{

  @Override
  List<I_User> findAll();

  @Query("SELECT p FROM I_User p WHERE LOWER(p.i_username) = LOWER(:username)")
  public I_User find(@Param("username") String username);

}
```

The I_Location_Repository is used to save a location object in the database. The method save is provided by default from the GrudRepository. So, the I_Location_Repository contains only other methods, which are used to interact with the database for other functionalities such as find the locations for a specific user:

```java
public interface I_Location_Repository extends CrudRepository<I_Location, Long>{

    @Override
    List<I_Location> findAll();

    @Query("SELECT p FROM I_Location p WHERE p.i_user_id = :i_user_id")
    public List<I_Location> findByUserId(@Param("i_user_id") Long i_user_id);

    @Query("SELECT DISTINCT p.i_country_name FROM I_Location p WHERE
p.i_country_name!='null' and p.i_country_name is not null")
    public List<String> getDistinctCountyNames();

     @Query("SELECT DISTINCT p.i_department FROM I_Location p WHERE
p.i_department!='null' and p.i_department is not null")
    public List<String> getDistinctDepartmentNames();

     @Query("SELECT DISTINCT p.i_prefecture FROM I_Location p WHERE
p.i_prefecture!='null' and p.i_prefecture is not null")
    public List<String> getDistinctPrefectureNames();

    @Query("SELECT DISTINCT p.i_city_name FROM I_Location p WHERE p.i_city_name!='null'
and p.i_city_name is not null")
    public List<String> getDistinctCityNames();

     @Query("SELECT DISTINCT p.i_postal_code FROM I_Location p WHERE
p.i_postal_code!='null' and p.i_postal_code is not null")
    public List<String> getDistinctPostalCodes();

    @Query("SELECT DISTINCT p.i_department FROM I_Location p WHERE p.i_department is
not null and p.i_country_name in(:data)")
    public List<String> returnDistinctLocationDepartmentViaFilter(@Param("data")
List<String> data);

    @Query("SELECT DISTINCT p.i_prefecture FROM I_Location p WHERE p.i_prefecture is
not null and p.i_department in(:data)")
    public List<String> returnDistinctLocationPrefectureViaFilter(@Param("data")
List<String> data);

    @Query("SELECT DISTINCT p.i_city_name FROM I_Location p WHERE p.i_city_name is not
null and p.i_prefecture in(:data)")
    public List<String> returnDistinctLocationCityViaFilter(@Param("data") List<String>
data);

    @Query("SELECT DISTINCT p.i_postal_code FROM I_Location p WHERE p.i_postal_code is
not null and p.i_city_name in(:data)")
    public List<String> returnDistinctLocationPostalViaFilter(@Param("data")
List<String> data);

}
```

5.7.4 Segment Filters

The Segment filters have been implemented in the "new segment" page of the admin tool. These filters are used to select under which conditions the users will be included in the specific segment. The filters are composed from different categories, but here only the target location category will be explained. The target location information is composed from different subcategories (country, department, prefecture, city and postal code). So, the administrator can select every time data from one subcategory. The next subcategory list will expand only corresponding data of the section, according to the selected data from the preview subcategory.

Below is the JavaScript section which is used to return the available countries. The XMLHttpRequest object can be used to exchange data with a web server behind the scenes. The countries are requested by using specific URL and GET method. Then the response is handled in a JSON format which is read in a "for" loop, in order to build the html section which is the inserted to the web page:

```
addLocationCountryNames: function(locationCountryId){
      var xhttp = new XMLHttpRequest();
         xhttp.onreadystatechange = function() {
               var data,x = "",i,flen;
               if (this.readyState === 4 && this.status === 200) {
                    data = JSON.parse(this.responseText);
                    flen = data.length;

                     if (data!==null){
                         for (i=0;i<flen;i++){
                             x+= "<option>" + data[i] + "</option>"
                         }

                      document.getElementById(locationCountryId).innerHTML = x;
                  }

            }
         };
                   xhttp.open("GET","http://localhost:8080/getDistinctCountyNames",
true);
xhttp.send();
}
```

Below is the JavaScript section which is used to handle the "on click" event of the country button. The code changes the web page by hiding the country button, showing the country select list, displaying the department button and a reset filter button:

```
selectCountry: function(){

    var countrySel = document.getElementById('countrySelect');
    var countryButton = document.getElementById('countryButton');
    var departmentButton =
document.getElementById('departmentButton');
    var resetButton = document.getElementById('resetLocationButton');

    countrySel.style.display='block';
    countryButton.style.display='none';
    departmentButton.style.display='block';
    resetButton.style.display='block';
```

After the changes from the above code section, the user can choose the available countries. Then, if he clicks on the department button, the mechanism returns the departments of the selected countries. Below is the JavaScript code which disables the department button, presents a "select" list with departments only from the selected values and presents the prefecture button:

```
selectDepartment: function(){
    var countrySel = document.getElementById('countrySelect');
    var departmentSel = document.getElementById('departmentSelect');
    var departmentButton = document.getElementById('departmentButton');
    var prefectureButton = document.getElementById('prefectureButton');
targetlocation.addFilterValues('departmentSelect',targetlocation.getSelectValues(countr
ySel),'http://localhost:8080/getDistinctDepartmentNames','http://localhost:8080/getDist
inctLocationDepartmentViaFilter');
    countrySel.disabled = true;
    departmentSel.style.display = 'block';
    departmentButton.style.display = 'none';
    prefectureButton.style.display = 'block';
`
```

In the below JavaScript code the function addFilterValues is called with four arguments. The departments select list id, a list with selected items from country select list, the request URL, which returns all the departments and the request URL, which returns the departments from the selected countries. The purpose is to check if the user has selected or no any countries. If there are no countries selected, the first request URL is used in order to fill the department select list. In the opposite case, the second request URL with the selected items is used to request the appropriate web service. Then, the web service uses the selected countries in order to return only the corresponding departments which exist in the database:

```javascript
addFilterValues: function(itemId,selectedItems,URL,FILTER_URL){
    if (selectedItems === null){
        var xhttp = new XMLHttpRequest();
                xhttp.onreadystatechange = function() {
                    var data,x = "",i,flen;
                    if (this.readyState === 4 && this.status === 200) {
                        data = JSON.parse(this.responseText);
                        flen = data.length;

                        if (data!==null){
                            for (i=0;i<flen;i++){
                             x+= "<option>" + data[i] + "</option>";
                            }
                            document.getElementById(itemId).innerHTML = x;
                        }
                    }
                };
                xhttp.open("GET", URL, true);
                    xhttp.send();
                }else{
                    var xhr = new XMLHttpRequest();

                        xhr.open("POST", FILTER_URL, true);
                        xhr.setRequestHeader("Content-type",
"application/json");

                        xhr.onreadystatechange = function () {
                            var data,x,z,final,i,flen;
                            if (xhr.readyState === 4 && xhr.status === 200) {

                                data = JSON.parse(xhr.responseText);
                                flen = data.length;

                                 if (data!==null){
                                    for (i=0;i<flen;i++){
                                        x+= '<option value="' + data[i] +
'">' + data[i] + '</option>';
                                    }

document.getElementById(itemId).innerHTML = x;
                                }
                            }
                        };

                        xhr.send(selectedItems);
                    }
    }
};
```

In the same way the prefecture, city and postal code filters are expanded. But let's see how the web service handles the requests for the department select list and how it responses with the appropriate data. In the ILocationController class the methods getDistinctDepartmentNames and getDistinctLocationDepartmentViaFilter have been designed to handle the above requests. The getDistinctDepartmentNames uses the getDistinctDepartmentNames from

I_Location_Repository so as to return all the departments. The getDistinctLocationDepartmentViaFilter handles a post request with country list in JSON format. Then it uses the returnDistinctLocationDepartmentViaFilter with the list of the selected countries as argument from I_Location_Repository in order to return the corresponding departments:

```java
@RequestMapping("/getDistinctDepartmentNames")
 public List<String> getDistinctDepartmentNames(){
     return i_Location_Repository.getDistinctDepartmentNames();
 }

@RequestMapping(value = "/getDistinctLocationDepartmentViaFilter", method =
RequestMethod.POST)
public List<String> getDistinctLocationDepartmentViaFilter(@RequestBody List<String>
string){

     return i_Location_Repository.returnDistinctLocationDepartmentViaFilter(string);
 }
```

Below you can see how the method returnDistinctLocationDepartmentViaFilter in the I_Location_Repository interface is defined:

```java
    @Query("SELECT DISTINCT p.i_department FROM I_Location p WHERE p.i_department is
not null and p.i_country_name in(:data)")
    public List<String> returnDistinctLocationDepartmentViaFilter(@Param("data")
List<String> data);
```

5.7.5 Saving New Segment

The segments are necessary for the web application and they are used by the admin tool. Specifically, the campaigns use the segments in order to target specific users. The administrator uses the filters which are described above in order to select specific criteria. Then, the criteria are saved in the database in JSON format. The criteria in JSON format are processed before the campaign execution, in order to create one SQL Script, which is used to select the target users.

Below is the JavaScript function which is called when the user saves a segment. The function recognizes all the text inputs and selected lists. The data are handled from the inputs simply and the selected data from the selected lists are handled by using the getSelectValues function. All the data are saved in JSON format and the callTheBackToSave function is called to save the segment in the end:

```javascript
onSave: function(){

    var name = document.getElementById('segmentName');
    var description = document.getElementById('segmentDescription');
    var country = document.getElementById('countrySelect');
    var department = document.getElementById('departmentSelect');
    var prefecture = document.getElementById('prefectureSelect');
    var city = document.getElementById('citySelect');
    var postal = document.getElementById('postalSelect');
    var employer = document.getElementById('employerSelect');
    var position = document.getElementById('positionSelect');
    var school = document.getElementById('schoolSelect');
    var degree = document.getElementById('degreeSelect');
    var year = document.getElementById('yearSelect');
    var books = document.getElementById('bookSelect');
    var teams = document.getElementById('teamSelect');
    var events = document.getElementById('eventSelect');
    var music = document.getElementById('musicSelect');
    var pages = document.getElementById('pageSelect');

    var js='{"name":';
    if (name.value==='' || name.value === null){
       js+= '""' + ', "description": ';
    }else{
        js+= '"' + name.value + '"' + ', "description": ';
    }
    if (description.value === '' || description.value === null){
        js+= '""' + ', "country": ';
    } else{
        js+= '"' + description.value + '"' + ', "country": ';
    }

    js+= targeteducation.getSelectValues(country) + ', ';
    js+= '"department":' + targeteducation.getSelectValues(department) + ', ';
    js+= '"prefecture":' + targeteducation.getSelectValues(prefecture) + ', ';
    js+= '"city":' + targeteducation.getSelectValues(city) + ', ';
    js+= '"postal":' + targeteducation.getSelectValues(postal) + ', ';
    js+= '"employer":' + targeteducation.getSelectValues(employer) + ', ';
    js+= '"position":' + targeteducation.getSelectValues(position) + ', ';
    js+= '"school":' + targeteducation.getSelectValues(school) + ', ';
    js+= '"degree":' + targeteducation.getSelectValues(degree) + ', ';
    js+= '"year":' + targeteducation.getSelectValues(year) + ', ';
    js+= '"books":' + targeteducation.getSelectValues(books) + ', ';
    js+= '"teams":' + targeteducation.getSelectValues(teams) + ', ';
    js+= '"events":' + targeteducation.getSelectValues(events) + ', ';
    js+= '"music":' + targeteducation.getSelectValues(music) + ', ';
    js+= '"pages":' + targeteducation.getSelectValues(pages) + '}';
```

The function callTheBackToSave accepts as argument the segment information as JSON format. Then, it requests a web service to save the new segment using a POST method and including the JSON data. If the response has success status, the segment is saved successfully. If the response status is "fail", the error message with the error description is presented. Below is the function which requests and handles the response:

```javascript
callTheBackToSave: function(json){
        var data;
        var xhr = new XMLHttpRequest();
        var url = "http://localhost:8080/saveNewSegment";
        xhr.open("POST", url, true);
        xhr.setRequestHeader("Content-type", "application/json");
        xhr.onreadystatechange = function () {

        if (xhr.readyState === 4 && xhr.status === 200) {

            data = JSON.parse(xhr.responseText);
            if(data.status==='success'){

                var successVal = document.getElementById('successMessage');

                document.getElementById('errorMessage').style.display='none';
                successVal.innerHTML = '<strong>Success! </strong> The segment has
been saved successfully.';
                successVal.style.display='block';
                segments.getAllSegments();
                document.getElementById('newSegment').style.display='none';
                document.getElementById('segmentList').style.display='block';


            }else{
                var errorVal = document.getElementById('errorMessage');
                errorVal.style.display='block';
                document.getElementById('successMessage').style.display='none';
                errorVal.innerHTML = '<strong>Warning!</strong>' +
data.object.description;
            }

                    }
                };


                    xhr.send(json);
    }
```

On the other hand, a specific web service handles this request, processes the JSON data and responses appropriately. A method named saveNewSegment which accepts the JSON data as argument has been implemented in the SYSSegmentController class. Firstly, it reads the name and description of the segment from the JSON format and then it checks whether it is null or if the same name exists in the database, so as to provide response with fail status and relative error message. If the name is not null and the name does not exist in the database, then the JSON data are saved in the SYS_JSON_FILTER array and the segment is saved in SYS_SEGMENT array. With this flow, the segment is saved successfully and can be used by any campaign:

```java
    @RequestMapping("/saveNewSegment")
    public ResponseObject saveNewSegment(@RequestBody String json) throws
JSONException{
        String name,description;
        JSONObject jsonObject = new JSONObject(json);
        if(jsonObject.get("name")==null){
          name = null;
        }else{
            name = (String) jsonObject.get("name");
        }

        if(jsonObject.get("description")==null){
          description = null;
        }else{
          description = (String) jsonObject.get("description");
        }
        if("".equals(name)){
        return new ResponseObject("fail",new errorMessage("Invalid Name","Missing the
name",""));

        }else if (sys_Segment_Repository.findSegmentByName(name)!=null){
          return new ResponseObject("fail",new errorMessage("Name Exists","Please give
another name.",""));
        }else{
            Sys_Json_Filter newJsonFilter = new Sys_Json_Filter(json);
            sys_Json_Filter_Repository.save(newJsonFilter);
            Sys_Segment newSegment = new
Sys_Segment(name,description,newJsonFilter.getSys_json_id());
        sys_Segment_Repository.save(newSegment);

             return new ResponseObject("success",newSegment);
        }
```

5.7.6   Saving New "Simple Notification Campaign"

The administrator is able to configure, save and activate a campaign to send a simple notification to a specific target group. The campaign can be saved, when the name and description are filled in for the campaign, a unique segment is selected, an execution time is picked and title & body of the notification are filled in for this simple notification. All the inputs should be filled in or selected, in order for the campaign to be saved successfully. Also, the execution date/time should be a future date/time.

The function onSave has been implemented in the "campaign.js" file and is defined in the "edit" namespace. Firstly, it reads all the filled in and selected values. In a second phase, it checks if the name, description, datetime, notification title and notification body have not been filled in, so as to enable the corresponding error messages. If this information is filled in, then the function builds the JSON, which contains all the configured information and calls the callTheBackToSave function, which accepts the JSON as argument:

```javascript
    onSave: function(id){
        var js="";
        var name = document.getElementById('campaignName').value;
        var description = document.getElementById('campaignDescription').value;
        var segmentObject = document.getElementById('selectSegment');
        var segmentId = segmentObject.options[segmentObject.selectedIndex].value;
        var datetime = document.getElementById('notificationDatetime').value;
        var notificationTitle = document.getElementById('notificationTitle').value;
        var notificationBody = document.getElementById('notificationBody').value;
        var errorMessageName = document.getElementById('errorMessageName');
        var errorMessageDescription =
document.getElementById('errorMessageDescription');
        var errorMessageDatetime = document.getElementById('errorMessageDatetime');
        var errorMessageNotificationTitle =
document.getElementById('errorMessageNotificationTitle');
        var errorMessageNotificationBody =
document.getElementById('errorMessageNotificationBody');
        errorMessageName.style.display='none';
        errorMessageDescription.style.display='none';
        errorMessageDatetime.style.display='none';
        errorMessageNotificationTitle.style.display='none';
        errorMessageNotificationBody.style.display='none';

        if(name===""){
            name=null;
            errorMessageName.style.display='block';
        }

        if(description===""){
            description=null;
            errorMessageDescription.style.display='block';
        }
        if(datetime===""){
            datetime=null;
            errorMessageDatetime.style.display='block';
        }

        if (notificationTitle===""){
            notificationTitle=null;
            errorMessageNotificationTitle.style.display='block';
        }
        if (notificationBody===""){
            notificationBody=null;
            errorMessageNotificationBody.style.display='block';
        }

      if(name!==null && description!==null && datetime!==null &&
notificationTitle!==null && notificationBody!==null){
            js+='{';
            js+='"campaignName" : "' + name + '",';
            js+='"campaignDescription" : "' + description + '",';
            js+='"segmentId" : "' + segmentId + '",';
            js+='"datetime" : "' + datetime + '",';
            js+='"notificationTitle" : "' + notificationTitle + '",';
            js+='"notificationBody" : "' + notificationBody + '",';
            js+='"campaignId" : "' + id + '"';
            js+='}';

             edit.callTheBackToSave(js);
        }
      }
```

The function callTheBackToSave has implemented in campaigns.js and is defined in the "edit" namespace. This function requests a web service in order to save the configured campaign. If the campaign name is not existed, the response has success status, else the status is fail with an error massage. According the response, the user is redirected to the campaigns page or presents all the available error messages in order the user corrects the wrong inputs:

```javascript
    callTheBackToSave: function(json){
        var data;
        var xhr = new XMLHttpRequest();
        var url = "http://localhost:8080/saveEditCampaign";
        xhr.open("POST", url, true);
        xhr.setRequestHeader("Content-type", "application/json");
        xhr.onreadystatechange = function () {

        if (xhr.readyState === 4 && xhr.status === 200) {

            data = JSON.parse(xhr.responseText);
            if(data.status==='success'){
                window.location.replace("/campaigns.html");

            }else{
                var errorVal =
document.getElementById('errorMessage');
                errorVal.style.display='block';
                errorVal.innerHTML = '<strong>Warning!</strong>' +
data.object.description;
            }
        }
                            };
                            xhr.send(json);
```

But let's see what happens on the other side. A web service handles the request and responses appropriately. The method saveNewCampaign in SYSCampaignController has been implemented to handle such requests. Firstly, it reads all the values from the requested JSON, then it checks if another campaign with the same name exists in the database in order to response with the corresponding error message. If the campaign has a unique name, then the campaign is saved to the relative SYS_CAMPAIGN array in the database:

```java
    @RequestMapping("/saveNewCampaign")
    public ResponseObject saveNewCampaign(@RequestBody String json) throws
JSONException{
        String name,description,datetime,segmentId,title,body;
        JSONObject jsonObject = new JSONObject(json);
        name = (String) jsonObject.get("campaignName");
        description = (String) jsonObject.get("campaignDescription");
        segmentId = (String) jsonObject.get("segmentId");
        datetime = (String) jsonObject.get("datetime");
        title = (String) jsonObject.get("notificationTitle");
        body = (String) jsonObject.get("notificationBody");

        if(sys_Campaign_Repository.getCampagnByName(name)!=null){
            return new ResponseObject("fail",new errorMessage("Invalid Message","The
Campaign name exists",""));
        }else{
            Sys_Campaign sys_Campaign = new
Sys_Campaign("SIMPLE_NOTIFICATION","INACTIVE",name,description,json,Long.valueOf(seg
mentId));
            sys_Campaign_Repository.save(sys_Campaign);
            return new ResponseObject("success",sys_Campaign);
        }
        }
```

5.7.7   **Activate New Simple Notification Campaign**

All the campaigns are in draft state after the saving that has been described above. Every saved campaign needs activation in order to be executed. The user can see the campaigns in draft mode in the campaigns section. The campaigns that are in draft mode can be activated by clicking on the "Activate" button.

The "Activate" button calls a function named "activate" from the campaigns.js file, which is defined in the campaign namespace. The function checks if the execution date/time of the configured campaign is on future date/time. If the execution date/time belongs to a past date/time, the campaign is not activated. If the date/time belongs to a future date/time, the execution is scheduled by calling the triggerSimpleNotification function and the campaign's state changes from draft to activated:

```javascript
activate: function(id){

    var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
                var campaignTime,campaignTimeMili, sysdate, sysdateMili;
                if (this.readyState === 4 && this.status === 200) {
                    campaignTime = new Date(this.responseText);
                    campaignTimeMili = Date.parse(campaignTime);
                    campaignTimeMili += 100000;
                    sysdate = new Date();
                    sysdateMili = Date.parse(sysdate);
                    if(sysdateMili > campaignTimeMili){
                        document.getElementById('errorMessage').innerHTML = '<strong>Warning!</strong>' + 'Invalid
excecution date';
                        document.getElementById('errorMessage').style.display = 'block';
                    }else{

                        campaign.triggerSimpleNotification(campaignTime,id);
                    }
                }
            };
            xhttp.open("GET", "http://localhost:8080/getStartDateOfCampaign?id=" + id, true);
            xhttp.send();
    }
```

The triggerSimpleNotification function is defined in the campaign namespace of campaigns.js file as well. This function is responsible of building a JSON file and requesting a web service in order to schedule the execution of the campaign. The request contains a JSON file which provides information about the type of the campaign, the name of the scheduled job, the group that the job consists of and the execution date/time that the job should be executed. How the request handles and schedules the job is described in the next section. Below is the function which is used for the request:

```
    triggerSimpleNotification: function(date,id){
        var month = parseInt(date.getMonth());
        month++;
        var json = '{ "name": "Simple Notification ' + id + '", "id" : ' + id + ',
 "triggers":[ { "name": "Simple Notification ' + id + '", "group":
 "simplenotification", "cron": "0 ' + date.getMinutes() + ' ' + date.getHours() + ' '
 + date.getDate() + ' ' + month + ' ? ' + date.getFullYear() + '"} ]}';
        alert(json);
        var xhr = new XMLHttpRequest();
        var url =
 "http://localhost:8080/simple/notification/api/v1.0/groups/simplenotification/jobs";
        xhr.open("POST", url, true);
        xhr.setRequestHeader("Content-type", "application/json");
        xhr.onreadystatechange = function () {

        if (xhr.readyState === 4 && xhr.status === 200) {

            alert("Campaign activated!!!");
            campaign.changeActivateStatusOnDatabase(id);
        }
        };
        xhr.send(json);
        campaign.changeActivateStatusOnDatabase(id);
    }
```

5.7.8   Send a Simple Notification

The above section describes how the campaign is scheduled and started in order to send a simple notification. Now it's time to describe how the web service, which handles this request and schedules the campaign, is implemented. The request is handled by using a controller and which then calls a service, which schedules the campaign to run on the configured date/time by using the Quartz Job Scheduling Library.

First of all, let's discuss about this library. Quartz is a richly featured, open source job scheduling library that can be integrated within virtually any Java application - from the smallest stand-alone application to the largest e-commerce system. Quartz can be used to create simple or complex schedules for executing tens, hundreds, or even tens-of-thousands of jobs; jobs whose tasks are defined as standard Java components that may execute virtually anything you may program them to do.

The Scheduling Process needs first to define the job and tie it with a class using the JobDetail class. Then, it needs to define the trigger by using the Trigger class, which is used to fire the created job. The scheduler is used with these defined objects in order to schedule and run the job. The service which creates a new scheduled job in order to execute a simple notification campaign is described below.

The method createJob in the SIMPLENotificationJobScheduleController is used to handle the incoming requests when a campaign is activated. The method handles the group value from the requested URI and the SimpleNotificationDescriptor object, which is passed as a JSON body on the request. Then the createJob method, with these arguments, is called from the simpleNotificationService class:

```
@PostMapping(path = "/groups/{group}/jobs")
public ResponseEntity<SimpleNotificationDescriptor> createJob(@PathVariable String
group, @RequestBody SimpleNotificationDescriptor descriptor) {
                return new ResponseEntity<>(simpleNotificationService.createJob(group,
descriptor), CREATED);
        }
```

The createJob method in the SimpleNotificationService class defines the JobDetail object. The JobDetail object contains execution information about what the job is going to do. Within JobDetail object, a SimpleNotificationJob object is also defined, which implements the Job interface and defines the steps that will do the job after the execution. The createJob method also defines a set of triggers, which contain a set of dates/times that the job will be executed on. The triggers are built with information from SimpleNotificationDescriptor object which is passed as argument. Then the set of the triggers is created. In the end, the SheduleJob method from Scheduler class is used in order to schedule the job:

```
public SimpleNotificationDescriptor createJob(String group, SimpleNotificationDescriptor descriptor) {
        descriptor.setGroup(group);
        JobDetail jobDetail = descriptor.buildJobDetail();
        Set<Trigger> triggersForJob = descriptor.buildTriggers();
        log.info("About to save job with key - {}", jobDetail.getKey());
        try {
                scheduler.scheduleJob(jobDetail, triggersForJob, false);
                log.info("Job with key - {} saved sucessfully", jobDetail.getKey());
            } catch (SchedulerException e) {
        log.error("Could not save job with key - {} due to error - {}", jobDetail.getKey(), e.getLocalizedMessage());
            throw new IllegalArgumentException(e.getLocalizedMessage());
          }
        return descriptor;
    }
```

Let's see the overridden execution method, which describes the job steps. The job contains an id, which refers to a specific campaign id. The campaign data are read from SYS_CAMPAIGN table by using this id. The segment data is read from SYS_SEGMENT table, by using the segment id, which is provided from the SYS_CAMPAIGN table. The segment criteria are saved on JSON format in the SYS_JSON_FILTER table, which are then retrieved by using the segment id from SYS_SEGMENT table. Since all the needed information is collected, the method starts to build the notification. Firstly, it retrieves the title and the body that will be used on the notification from the SYS_CAMPAIGN table. The SQL Script that will create the target group is created by using the method getSegmentUsers from JsonToQuery class, which accepts as argument the segment criteria in JSON format. The SQL Script is used to retrieve the list with the users and then the notification is sent to the whole target group by using the sentMessageToUser method from Firebase_Send_Message_Repository. These methods send requests to the Firebase Cloud Messaging server, in order for the notifications to be delivered to the targeted mobile devices:

```java
@Override
    public void execute(JobExecutionContext context) throws JobExecutionException {

        log.info("Job Simple Notification triggered");
                JobDataMap map = context.getMergedJobDataMap();
                Long id = map.getLong("id");
                Sys_Campaign sys_campaign = sys_Campaign_Repository.findOne(id);
                Sys_Segment sys_Segment =
sys_Segment_Repository.findOne(sys_campaign.getSys_segment_id());
                Sys_Json_Filter sys_Json_Filter =
sys_Json_Filter_Repository.findOne(sys_Segment.getSys_json_id());
                JsonToQuery jsonToQuery = new JsonToQuery();
        try {
            JSONObject jsonCampaignData = new
JSONObject(sys_campaign.getSys_json_data());
            title = jsonCampaignData.getString("notificationTitle");
            body = jsonCampaignData.getString("notificationBody");
        } catch (JSONException ex) {
            Logger.getLogger(SimpleNotificationJob.class.getName()).log(Level.SEVERE,
null, ex);
        }

        try {
            this.query = jsonToQuery.getSegmentUsers(sys_Json_Filter.getSys_json());
            System.out.println(this.query);

        } catch (JSONException ex) {
            Logger.getLogger(SimpleNotificationJob.class.getName()).log(Level.SEVERE,
null, ex);
        }

     String jdbcUrl= "jdbc:postgresql://localhost:5432/program";
     String username= "postgres";
     String password= "giannis2669";

    try (Connection conn = DriverManager.getConnection(jdbcUrl, username, password);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(this.query);) {

     int row = 0;
     while (rs.next()) {


firebase_Send_Message_Repository.sentMessageToUser(firebase_Repository.getAccessToken
(), rs.getString(11),title , body);

     }
    } catch (SQLException e) {
      e.printStackTrace();
    }    catch (IOException ex) {
            Logger.getLogger(SimpleNotificationJob.class.getName()).log(Level.SEVERE,
null, ex);
        }


    }
```

Conclusion and Future Extensions

The number of network connected devices per person around the world from 2003 to 2020, is rising rapidly. Most of them are mobile devices, which can be used to handle specific information about user location, interests, needs, sports, education, work, music and preferred events. This is an opportunity for us to create more targeted campaigns, exploiting the rapidly rising usage of mobile devices.

However, the user's personal information must be handled with respect to the user rights. The personal information must be handled only after the user's permission. The implemented applications in this dissertation are based on personal data. The android application uses the location service in order to report the last known user location. Also, the android application exploits the Facebook login so as to handle personal information. For this reason, the application asks the user's permissions, before the usage of location service and the usage of Facebook data are enabled.

Over the years, the advertisement campaigns have increased. The majority of people is tired of reading or watching all these campaigns, because most of them do not seem interesting to them. So, they try to avoid advertisement campaigns from other common communication channels by using specific ad-blockers. The simple notification campaigns are different. They are using the mobile devices in order to target only specific users based upon their interests. The users benefit from this way of advertisement because they are not receiving useless information on a daily basis.

At the moment the campaigns are limited only on the "simple notification campaigns" type that sends a simple notification. The next move is to implement in-out campaigns, which will be able to interact with the users. These campaigns may communicate with the user in order to recognize which communication channel is preferred. Also, the campaigns may invoke the users to do an action, such as visiting a specific page in order to give them a discount coupon. Generally, the campaigns will have flows and take decisions according to the user's interaction.

In conclusion, an algorithm could be implemented in order to optimize the filter operation of the segmentation. LinkedIn could be used as well in order to handle education and work experience data better. Additionally, the admin tool could be upgraded to support draws, which will assign prizes to the users in order to intrigue them to continue using the android application and interact with the campaigns.

Bibliography

[1] Craig, Walls., 2015. Spring in action. 4th edition. USA: Kindle Edition

[2] Jeanne, Hopkins., Jamie, Turner., 2012. Go Mobile: Location-Based Marketing, Apps, Mobile Optimized Ad Campaigns, 2D Codes and Other Mobile Strategies to Grow Your Business. Canada: Wiley

[3] Amuthan G., 2014. Spring MVC Beginner's Guide . UK: Packt Publishing Ltd

[4] https://developer.android.com/training/scheduling/alarms

[5] https://developer.android.com/guide/components/services

[6] https://developer.android.com/guide/components/broadcasts

[7] https://developer.android.com/topic/performance/scheduling

[8] https://developers.facebook.com/docs/android/graph

[9] https://developer.android.com/training/location/display-address

[10] https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[11] https://projects.spring.io/spring-boot/