TRABAJO FIN DE GRADO

Título:	Diseño e implementación de un sistema de evacuación de edificios en Google Glass
Título (inglés):	Design and implementation of a building's emergency evacuation system on Google Glass
Autor:	Jesús Manuel Sánchez Martínez
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Tomás Robles Valladares
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Amalio Francisco Nieto Serrano

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF A BUILDING'S EMERGENCY EVACUATION SYSTEM ON GOOGLE GLASS

Jesús Manuel Sánchez Martínez

Julio de 2015

Resumen

Esta memoria es el resultado de un proyecto cuyo objetivo ha sido implementar un sistema de evacuación de edificios en caso de que tuviese lugar alguna emergencia.

Para hacer esto se ha implementado un servidor que será el encargado de mandar las notificaciones a los dispositivos conectados en caso de que exista la necesidad de evacuar el edificio en cuestión. El usuario de las Google Glass verá la notificación y posteriormente podrá acceder a la aplicación de evacuación.

Este servidor está compuesto por diferentes módulos que se encargan tanto de trabajar en la autenticación por parte del propio servidor y del dispositivo del usuario como de mandar al servidor de Google las notificaciones para que éste posteriormente las reenvíe a las Google Glass.

Posteriormente se ha diseñado una aplicación que consiste en la visualización del mapa del edificio. Esta aplicación permite, a través del acceso a la ubicación del usuario y de algoritmos de navegación, indicarnos el camino a la salida del edificio.

Por último, se han recogido las diferentes pruebas realizadas para comprobar el correcto funcionamiento del sistema.

Palabras clave: Google Glass, Indoor, Maps, Navigation, Routing, Android, Java

Abstract

This thesis is the result of a project whose objective is to develop and deploy a building's evacuation system on Google Glass in case there is an emergency in the building.

To do so, a server has been deployed in order to send the notifications to the linked devices should it exists the need to evacuate the building. The Google Glass's user will be able to see the alert and to access the evacuation application afterwards.

This server is composed of several modules that will work both for authenticating the server's requests and the user's device and for sending the notifications to the Google's server for this later to forward it to the Google Glass.

To continue, a Google Glass application has been developed. This application will allow the user to see the building plan along with its indoor location, and thanks to several navigation algorithms, the route to the nearest exit.

Finally, we gathered the extracted conclusions plus some tests done to check the correct performance of the system.

Keywords: Google Glass, Indoors, Maps, Navigation, Routing, Android, Java

Agradecimientos

Gracias a mis padres por motivarme a empezar esta carrera y apoyarme durante estos años hasta acabarla.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Normally, visitors or even everyday workers are not aware of the different areas or zones of the building itself. Therefore, moving in case of emergency or just trying to go from one place to another can become a big trouble.

Whereas it is very common to use navigation systems in cars to reach designated locations, indoor navigation systems are quite hard to find. Nowadays, some buildings with complex infrastructures are starting to use them, such as airports, hospitals, universities...

Recent technologies can provide a solution to this problem, thanks to electronic devices we can always be connected to the internet and therefore we can have access to our location and depending on the case, even to the building's fingerprints.

In this project, I will try to come up with a solution based on real-time network positioning. To do so, a server sending the emergency notification will be needed along with an application where the user will be able to see the fingerprints of the university and establish the shortest route to the exit.

1.2 Project goals

Along with the introduction of new electronic devices comes the adaptation of systems and applications already accomplished in other devices. In this project I will adapt an indoor navigation system to a new electronic device, the Google Glass.

This main goal includes some smaller tasks such as:

- Implementation of a notification launcher server linked to Google Glass.
- Design of a building fingerprint.
- Customization of the fingerprint using the routing tool.
- Setting WiFi access points with the measurement tool.
- Development of the Google Glass's application to visualize the routes.
- Testing in order to debug the system.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is the following: ***Chapter 1*** explains the context in which this project is developed. Moreover, it describes the main goals to achieve in this project. ***Chapter 2*** provides a description of the main technologies on which this project relies. ***Chapter 3*** explains the complete architecture and all the components and modules of the Evacuation System. ***Chapter 4*** describes how the map has been designed and implemented. ***Chapter 5*** provides an overview of the most important use cases. And finally ***Chapter 6*** joins the conclusions drawn from this project, problems faced and suggestions for a future work.

CHAPTER 2

Enabling Technologies

Before designing the application and the system environment, this chapter gives insight into techniques the used in this project. First of all, the Google Glass device along with its development features will be explained in section 2.1. Secondly, the technology that has made possible location tracking in section 2.2. Finally, the technologies that have been used to work with indoor mapping in section 2.3.

2.1 Google Glass

Google Glass is a type of wearable technology with an optical head-mounted display. It has been developed by Google with the mission of producing a mass-market ubiquitous computer. It also counts with a touch pad on the right part of the Glass that lets the user control the system with his finger. According to the concept of hands-free device, you can also use the glasses with voice commands.

In addition, the device has a built-in battery, speakers so that the user is able to listen to voice commands and a front camera to allow the user to take photos or record a video.

The different components and their position are shown in the following figure.

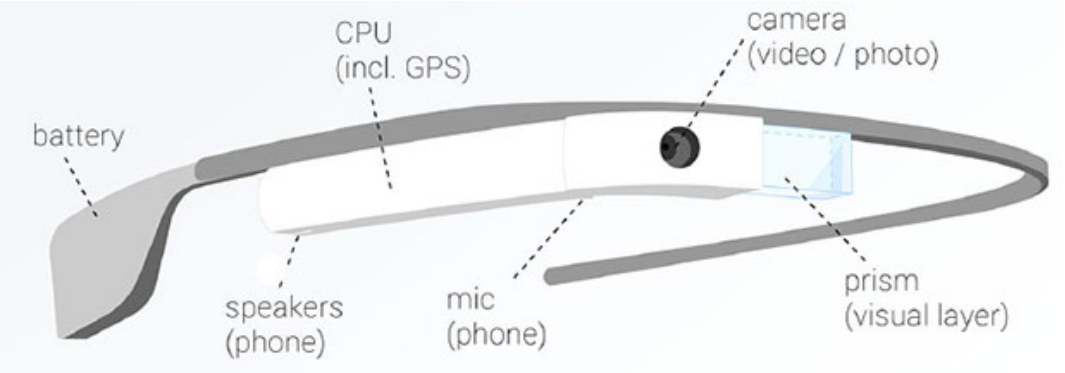

Figure 2.1: Google Glass Components

The Glass's interface is simple. It is based on a timeline sequence where the user can see the notifications sorted by date. The oldest ones are on the extreme right whereas the most recent ones are on the center. The device also counts with a standard block screen showing the actual time and a settings screen. The user can navigate through this interface swiping the finger on the touch pad. The next figure can help to understand the interface.

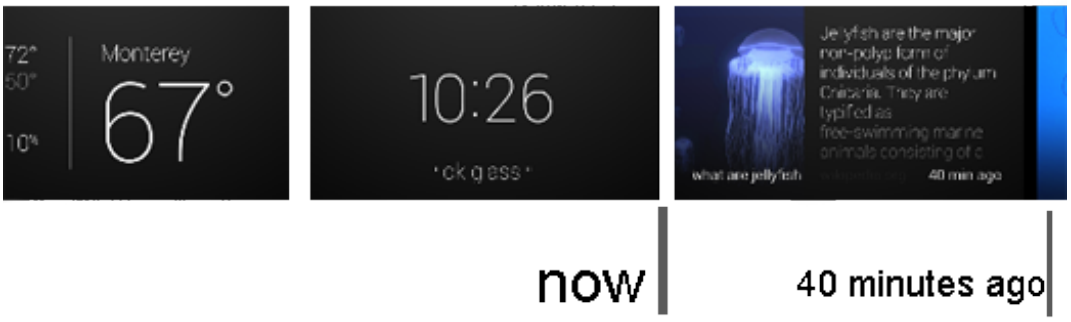

Figure 2.2: Timeline interface

The notifications shown in the timeline interface are called *Static cards*. There are two main ways to develop in Google Glass, Static cards through Mirror API and Immersive Applications [1]. The first one consists of launching notifications from a server to the device, while the second one is applications based and running on the device. In the following sections these two ways will be explained.

2.1.1 Mirror API

The Google Mirror API allows the developer to build web-based services that interact with Google Glass. It provides this functionality over a cloud-based API and does not require running code on Glass. To be able to use this API, a server with the logic of the application is needed. The following diagram shows the use case of sending a card from the web-based service to the user's device.

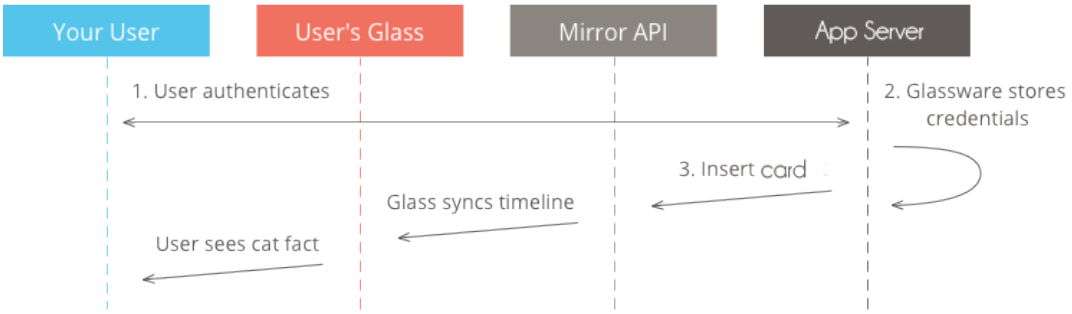

Figure 2.3: Card insertion

1. Users subscribe by authenticating with OAuth 2.0 in the web-based service.

2. The web-based service stores an index of users and their credentials.

3. The server publishes a new *Static Card*. It does this by iterating through all stored users and inserting a timeline item into their timelines.

In the next section the properties of the *Static Cards* will be explained.

2.1.1.1 Static Cards

Static cards are placed on the right of the Glass clock by default and display information relevant to the user at the time of delivery.

However, they do not require immediate attention and users can choose to read or act on the card at their own leisure. Glass may play a notification sound to alert users when Glassware inserts static cards into the timeline. The older static cards stored in the timeline will shift to the right and disappear from the timeline after 7 days or when 200 cards are newer.

Static cards can be written in HTML along with some CSS labels to create a more stylish card. Besides, they can have attachments such as images or videos.

The web-based service can also deliver pre-rendered map images in timeline cards by

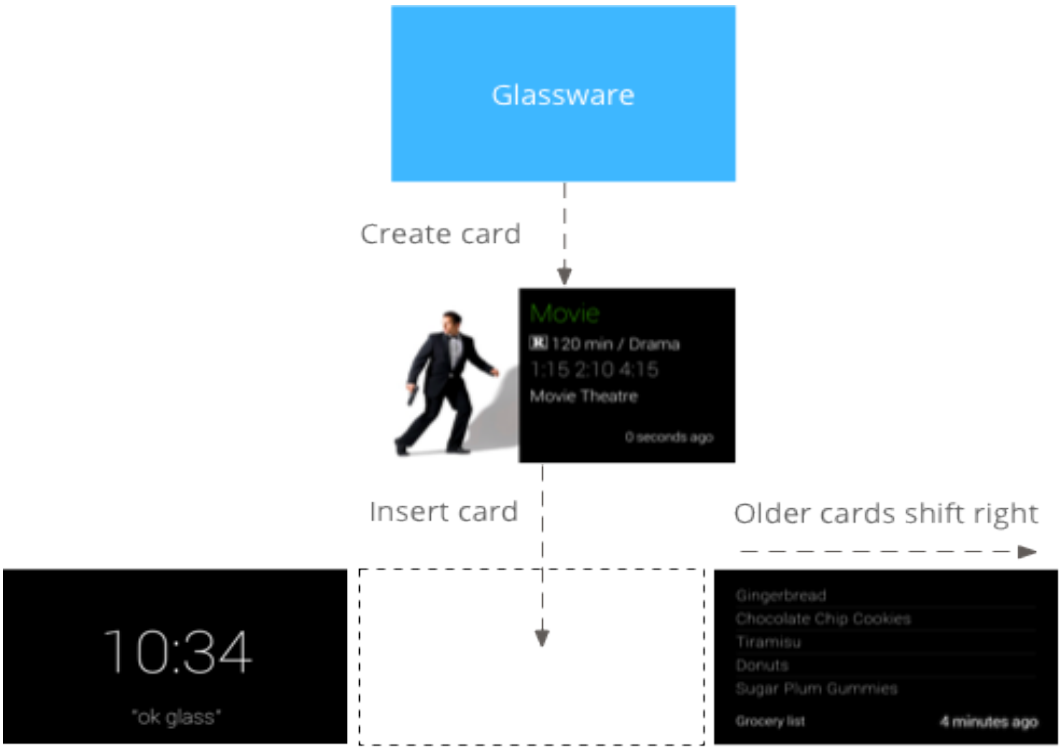

Figure 2.4: Static card timeline

giving the Mirror API the coordinates to draw. The Google Mirror API can render maps and overlay markers and lines to signify important places and paths using a limited version of the standard Google Maps API.

Another feature is the menu items that allows users to request actions that are related to the timeline card, and comes in two types: built-in menu items and custom menu items. On one hand built-in menu items provides access to special functionalities provided by Glass, such as reading a timeline card aloud, navigating to a location, opening a web page, sharing an image, or replying to a message. On the other hand, custom menu items allows the application to expose behaviour that is specific to the Glassware, and can also provide a menu item icon to match the particular branding. In this project custom menu items will not be used.

Static cards are great for delivering periodic notifications to users when important things happen. Mirror API static cards can also start live cards or immersions. This allows the developer to create hybrid interactions that use static cards as notifications and a live card or immersion for a more interactive experience. This type of experience will be the one used for this project, joining the advantages of static cards and immersion activities.

Here is an example:

Figure 2.5: Static card example

2.1.2 Glass Development Kit (GDK)

The Glass Development Kit (GDK) is an add-on to the Android SDK that lets the developer build Glassware that runs directly on Glass written in native code. By using this kit the developers can create two types of Glassware, Live Cards or Immersions.

2.1.2.1 Live Cards

Live cards appear in the present section of the timeline and display information that is relevant at the current time. However unlike static cards, live cards do not persist in the timeline and users explicitly remove them after they have finished using them.

Users typically start live cards by speaking a voice command at the main menu, which starts a background service that renders the card. Afterwards, they can tap the card to show menu items that can act on the card, such as removing it from the timeline like it has been explained in Static cards section.

Another benefit of live cards is that they are well suited for user's interfaces that require real-time interaction with users and real-time updates to the UI.

The timeline still has control over the user experience due to the fact that it is not an immersive experience, so actions on the touch pad like swiping forward or backward will change to other timeline item instead of acting on the live card itself. In addition, the screen will not be active at all times; it will turn on and off based on how the system behaves.

However, live cards have access to several of the same features that an immersion activity does, such as sensor or GPS data. Using these features, the developer can create experiences while the user stays in the timeline, being able to do other things.

The main use case of live cards is to deliver periodical information to Glass when users

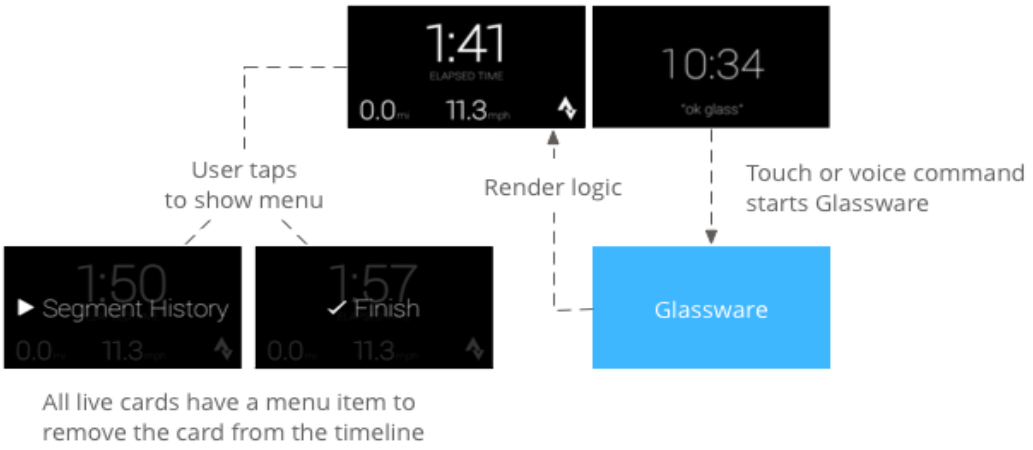

Figure 2.6: Live card timeline

are actively engaged in a task. For example, checking their time while running every few minutes or controlling a music player when they want to skip or pause a song.

In this project Live Cards will not be used because they do not provide a complete immersive experience and this will be required in our project.

2.1.2.2 Immersions

Immersions give the developer more ways to deal with user input and create user interfaces. This allows the developer to create custom experiences. However it involves a higher effort because it is based on programming native code.

Immersions display outside of the timeline, giving complete control over the user experience from the time Glass launches the immersion. To help its understanding we can compare them with the ordinary applications that can be used in devices like smartphones.

They can be built using standard Android activities, layouts, UI widgets, and the rest of the Android platform. Afterwards, the GDK is used to integrate specific features of Glass into the experience such as integrating voice commands, built-in menus, Glass-styled cards, and more. When immersions start, the system switches from the timeline to the activity appearing in full screen aspect. When users want to finish an immersion, they back out by swiping down.

Immersions are designed to create experiences that require continuous user attention along with having access to all the features the device can provide. By using them, we can create a specialized user interface that will appear outside the timeline, so users can go

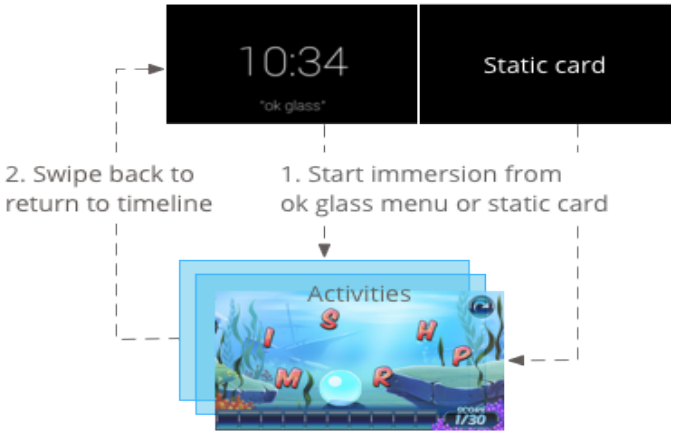

Figure 2.7: Immersion Pattern

deeper into a customized experience. Immersions are also necessary when the application needs to overwrite timeline-specific functionality such as forward and backward swipes and controlling the screen timeout. In this project we will create an immersion application so that the user can visualize the map and navigate through the building.

2.2 Location Tracking of Mobile Devices

The increase of the number of location-based services over the last years has lead to the need for better positioning methods of mobile devices. In the following sections I will explain the different technologies that can be used for positioning.

2.2.1 GPS

Global Positioning System (GPS) is the leading technology to determine locations on mobile devices. Nowadays, almost every electronic device is capable of working with GPS signals. GPS is a freely accessible system based on satellites. To determine a position the GPS receiver needs a line of sight to four or more satellites. For this reason, GPS only works outdoors.

Therefore, this method is not useful for this project due to the fact that it is based on indoor mapping.

2.2.2 WiFi

While the GPS method for positioning works great outdoors it is not usable indoors because it needs a line of sight to at least four satellites. The location tracking method based on mobile phone network is also not suitable due to its accuracy which is from 50 up to 300 meters and that Google Glass does not use mobile phone network.

To determine a usable indoor location estimate, different algorithms based on Wi-Fi technology can be used. In the following section we will explain the chosen one.

Radio Frequency Fingerprinting [2]

To create radio frequency (RF) fingerprints for different points of the area where the location should be tracked, we will need a physical walk through the building with special spectrum analysis units with the purpose of gathering the datasets and feature sets that can be used to generate a location fingerprint database. A fingerprint identifies locations by measures of the radio frequency setting, which is created by the wireless network access points. [3]

Some vendors include management systems for these fingerprints. These systems have the ability to compute fingerprints for every point of the area with sophisticated interpolation algorithms based on the measured fingerprints. The device sends the current RF fingerprint of its environment to a server in order to determine its position. The server compares this real-time fingerprint with the ones already measured stored in the database and computes a position based on the fingerprints which are similar to it.

The benefit of this system is that it also takes environmental effects like reflections on walls or other objects into account. Due to its advantages and the restriction of being indoor mapping this will be the method used for tracking the user while using our system.

2.3 Indoor Maps

There are many available tools to work with outdoor maps in every device, but not so many provide indoor maps as well. Because of the nature of this project, indoor mapping will be needed, therefore, in the following section the different possibilities and technologies will be described.

2.3.1 Google Maps Indoor

Even though they are not quite common yet, Google Maps provides indoor maps since the end of 2011 [4]. Instead of just showing the outdoor map with few details, when the user is viewing the map and zooms in, detailed floor plans automatically appear when indoor map data is available. Here is an example:

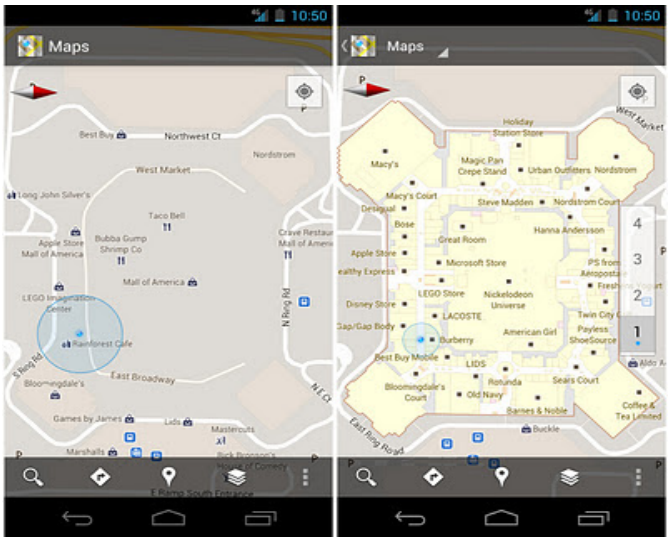

Figure 2.8: Mall of America in Minneapolis before and after, with a floor selector

Some buildings, mainly in USA, already implement this possibility such as airports or big retailers. In Spain we can find some examples like Plaza de Toros de Las Ventas or Madrid-Barajas airport.

The main disadvantage for this project would be the design of the university map and its publishing by Google that would delay the process of this project.

2.3.2 Leaflet

Leaflet is a modern open-source JavaScript library for mobile-friendly interactive maps. [5]

It works efficiently across all major desktop and mobile platforms out of the box, taking advantage of HTML5 and CSS3 on modern browsers while still being accessible on older ones. There are some developments that, combining Leaflet.js and MapBox.js, are trying to create indoor mapping like *Indoor.io* [6]

The main disadvantage of using this tool in the project is that Google Glass does not support JavaScript even though the MIT is developing a JavaScript environment, called

WearScript [7] . This environment includes some features such as a Playground, a method to send Static Cards to the Google Glass... However, this tool is still in development process and does not provide enough possibilities to be used in this project by now.

2.3.3 Private tools

Some enterprises have developed their own APIs to work with indoor mapping in Android devices. These companies provide their SDK to develop other applications using their products. Some of them are: InfSoft, SPREO or indoors.rs

The following table exposes the details of the enterprises that have been taken into account:

Vendor-Feature	Indoor Maps	Design	MMT	AR	GG Project	Free SDK
InfSoft	Yes	Good	Good	Yes	No	No
SPREO	Yes	Good	Unknown	Yes	Yes	No
indoo.rs	Yes	Medium	Good	No	Yes	Yes

All of these products count with measurement and management tools providing an easy interface to work with. The measurement tool is needed to be able to record the radio frequency fingerprints. [1]

The main disadvantage of InfSoft and SPREO API's is the premium access to their SDK, and not being able to use it without paying a certain amount of money. There are few differences between them, the most important being the augmented reality feature.

Therefore, the chosen one for this project is Indoo.rs due to its advantages.

2.4 Conclusions

In this chapter we have introduced some of the technologies which are part of the Google Glass project.

To sum up, in this project I will be using Google's Mirror API to send the notifications (static cards) to Glass. Then, I will use the Immersion type inside the GDK to develop the app that will guide the user through the building. To locate the user, WiFi radio frequency

[1] Check the section 2.2.2 for more information

fingerprinting will be the chosen technology. Eventually, to represent and deal with the maps, including routing and navigation, I will use Indoo.rs API.

CHAPTER 3

Architecture

In this section, we will explain the architecture of this project, including the design phase and implementation details. First of all, in the overview we divide the system into several modules to help its understanding. Afterwards, every module will be clarified in detail.

3.1 Overview

The project is composed of the following modules:

- Notification server: We need a server to be able to send the alert notifications (static cards) to the Google Glass device. The responsible of the building will access a graphic interface.

- Social simulator: In order to recreate the human behaviour, we will use this tool to establish a destination.

- GDK App: The Google Glass native application that the client will use to be guided through the building. Using Indoo.rs API, it will process the routing algorithm and

afterwards represent the path to the destination point. Therefore, this module is split into 2 sub-modules:

- Routing
- Map viewer

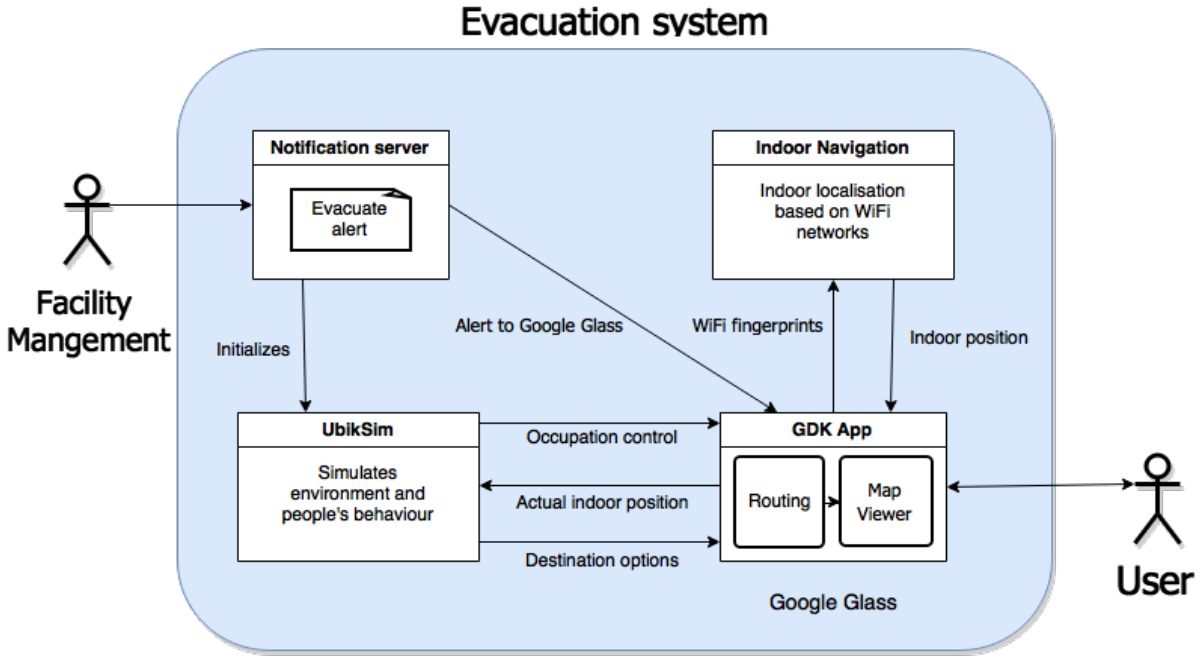

Figure 3.1: Architecture

3.2 Notification Server

The main goal of this module is to send the alert notification to Google Glass. To do so, Google's Mirror API [8] is needed. For this web-based service, we have built a server using the Google's Quick Start Project [9]. This project is available in several programming languages such as Java, PHP or Python. We have chosen Java because our knowledge of this language.

The process to use it is the following:

1. Navigate to link. An OAuth 2.0 permission request screen appears.
2. You will be asked to grant the project access to the Google account which has to be linked with the Google Glass device in order to be synchronized.

3. By clicking the launch alert notification, the server will use the Mirror API to send the static card to Google Glass.

4. In the device's timeline, the user will be able to see the new notification which will be linked to the GDK to continue the evacuation process.

The sequence process would look like this:

Figure 3.2: Process sequence

3.2.1 Authentication

Every HTTP request sent from the web application to Mirror API needs to be authorized. Mirror API uses "Bearer Authentication", which means that it is needed to provide a token with each request. Token is sent by the Google API using OAuth 2.0 protocol. [10]

1. Once the user has logged in with his credentials, the application will send a request to Google API, and the user will be presented with a consent screen generated by the Google API.

2. If access permissions are granted to the web application, Google API will issue a token that the server will use for calling the Mirror API

3.2.2 Graphic Interface

The person who launches the alerts will be asked to introduce his credentials and accept some permission requests. Once he has done it, the dashboard will appear showing the button to send the notification starting the evacuation process.

All this logic has been implemented using Java Servlets, HMTL5, CSS and JavaServer Pages (JSP).

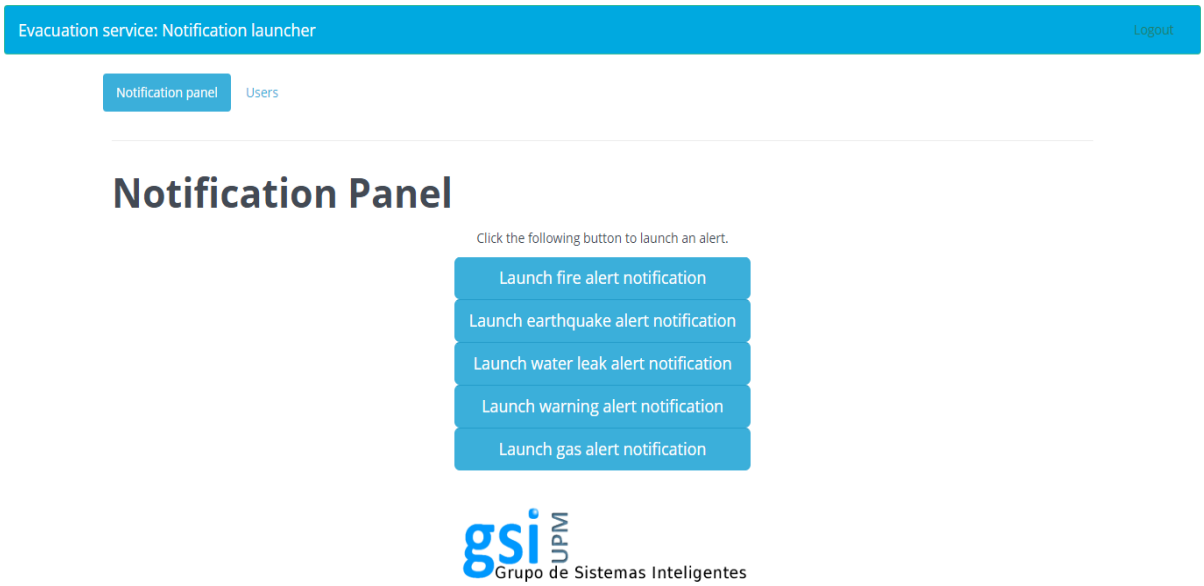

Figure 3.3: Dashboard

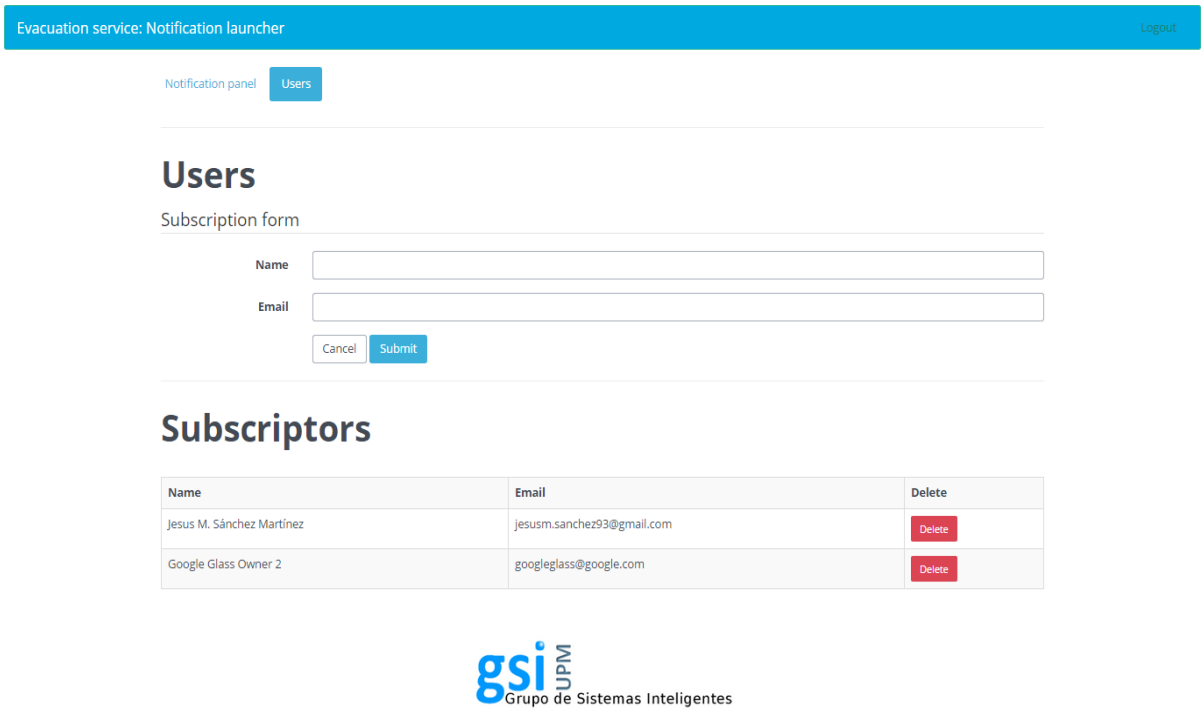

Figure 3.4: User creation

3.2.3 Database

To help the users, we have included a MongoDB database in which we will have access to all the userID interested in receiving our notifications. Every time a user submits his data, it is stored following this model:

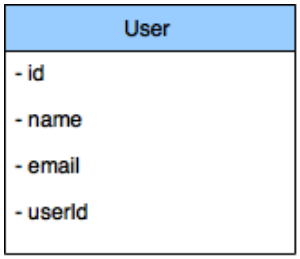

Figure 3.5: User model

The id field is just for internal purpose, the name and email fields are used to help the identification of each user. The most important field is userId, it is the id that Google assigns to every Gmail account and the one that is required to generate the credential needed to send the notification alert to Google Glass.

3.2.4 Notifications

To alert the user about the emergency, we use Static Cards (section 2.1.1.1). Five types of cards have been designed depending on the type of emergency: Fire, Earthquake, Water leak, Gas leak and Warning.

The cards used in this project are shown below:

Figure 3.6: Fire card

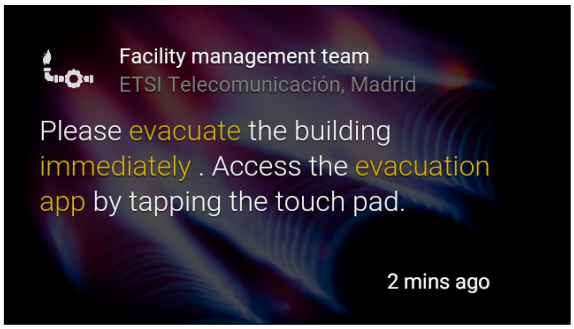

Figure 3.7: Gas leak card

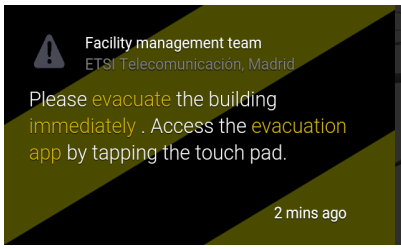

Figure 3.8: Warning card

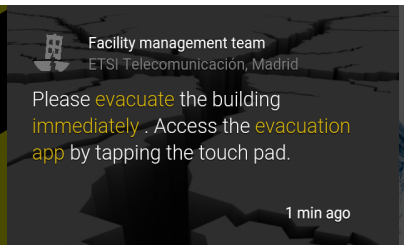

Figure 3.9: Earthquake card

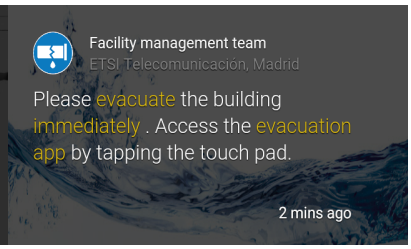

Figure 3.10: Water leak card

These cards are written in HTML, language that Google Glass can read and represent on the screen, here is an example of the code:

```
<article class='author'>
    <img src='http://i60.tinypic.com/693plj.jpg' width='100%' height
        ='100%'> <!-- Background -->
       <div class='overlay-full'/>
          <header>
            <img src='http://i60.tinypic.com/28wjiwh.png'/> <!-- Icon
                header -->
            <h1>Facility management team</h1>
            <h2>GSI, Madrid</h2>
          </header>
          <section>
            <p class='text-auto-size'> Please <span class='yellow'>
                evacuate</span> the building <span class='yellow'>
                immediately </span>. Access the <span class='yellow'>
                evacuation app </span>by tapping the touch pad.</p>
          </section>
</article>
```

All these Static Cards count with the following built-in actions:

- Read aloud: By accessing this action, the user is able to hear "Evacuation alert" warning him about the kind of notification.

- Delete: The user is also capable of deleting the card from the timeline if he wants to.

- Open website: This is the most interesting action. It is what we use to connect the alert to the GDK app. By accessing this action the user will execute a customized url (android scheme) opening the evacuation app.

3.2.5 Social simulator interaction

At the same time that we control the notifications we will control the simulation. Clicking any of the buttons from the Dashboard will execute several requests to the social simulator website. These requests are:

- Pause: We need to send this request to make the simulator create the environment, I will explain this need in the following section.
- Play: This request will forward the simulator to an initial position.
- Create emergency: To simulate a real situation we set a random emergency from the beginning.

3.3 UbikSim (Social simulator)

We will be using an adapted version of the social simulator UbikSim 2.0 [11] to recreate the human behaviour inside a building. For that purpose, the map of the university will be modelled and represented on this tool. UbikSim will have the duty to send the coordinates of the destination, in most of the cases, an exit and represent the actual position of the users.

UbikSim is a framework used to develop social simulation which emphasizes the construction of realistic indoor environments, the modelling of realistic human behaviours and the evaluation of Ubiquitous Computing and Ambient Intelligence systems. UbikSim is written in Java and employs a number of third-party libraries such as SweetHome3D and MASON. Our implementation consists of a console that will let us launch the simulation as well as a map in 3D or 2D where we will able to see the position of all the persons. To represent a real situation we will include some other agents apart from the real users of the system.

Once it is launched, to create the environment it is needed to change the status of UbikSim to *Paused*. To make it progress it is only needed to press or access the play action.

The GDK app will interact with UbikSim sending and retrieving data using a WebApp provided by UbikSim authors. The most important features are:

- Progress control: Including play, pause and stop actions.

- People: This action returns a JSON that contains the information about all the agents and their position.
- Update position: This action is complement to People feature. We can create or update the position of an agent by providing 3 parameters: id, x position and y position. The returned message is a JSON which contains the information about the distance of the exits.
- Emergency: This feature lets us create a random emergency and sets a goal exit for the virtual agents.

For more information about this point, see section 3.6.

3.4 GDK Application

Once the app is loaded, the user will see the building footprint as well as his location pointed with a circle. The route will be calculated using the indoo.rs API when the social simulator answers with the routes information.

3.4.1 Accessing the app

The user can access the app in three different ways:

- Static card: Tapping the notification card will show the option "Open website" which will take the user to the app.
- Voice menu: Google Glass counts with a built-in voice menu that will be displayed if the user says "ok glass". Inside that menu our app is shown and is accessible by saying "Evacuation app".
- Tactile menu: A menu will appear if the user taps on the touchpad while being in the "home screen", our app will be one of the options.

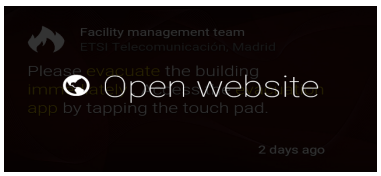

Figure 3.11: Static card

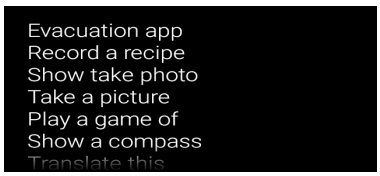

Figure 3.12: Voice menu

Figure 3.13: Tactile menu

3.4.2 User interface

To help the understanding, I attach a screenshot of the main activity below.

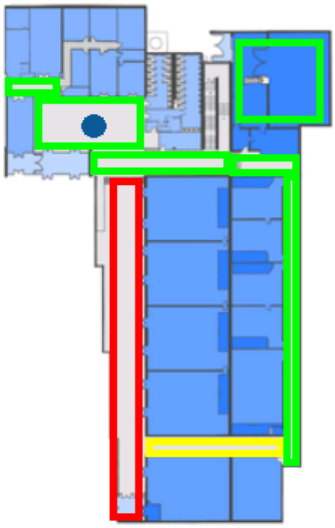

Figure 3.14: Google Glass app screen

The user can see his location pointed with a circle as well as the route chosen towards an exit. To let the user know whether an area is crowded or empty, we draw several coloured rectangles whose colour depends on the number of persons (virtual agents) that are in that specific zone.

3.4.3 Menus and user control

In order to let the user have a more important role in the system, we include the feature of choosing the type of route that he wants. Following the philosophy of Google and its hands-free usage of Google Glass I have implemented two types of menu:

3.4.3.1 Voice control

The user can access the routes menu by saying "Ok glass". Afterwards, a submenu will be displayed showing the routing options: "Take me to the closest exit", "Take me to the safest exit" or "Take me to the least crowded exit".

Figure 3.15: Voice control menu

3.4.3.2 Gesture control

This menu can be accessed using gestures on the Google Glass touchpad, more precisely by a long press. As it can be appreciated in the following figure, the user can scroll through the same options that in the voice control to choose the one he wishes.

Figure 3.16: Tactile menu

Some other gesture options are available:

- Tap: Tapping the touchpad will make the window scroll down.
- Swipe up: It is the opposite action of the one above, it scrolls up de window.
- Swipe right: This gesture increases the zoom of the map.
- Swipe left: It is the opposite action, it reduces the zoom of the map.
- Swipe down: Like any other Google Glass native app, it can be shut down by swiping down.

3.4.4 Structure

In the following section I will explain how the app is structured and what are the methods and algorithms used to achieve our goals. The app is based on a single activity, which represents the building fingerprint, user's position and route path. There are several global variables such as *closestExit*, *safestExit*, *leastCrowdedExit*, that store the value of each type of exit depending on the user, or the constant *zones* which contains the number of areas to analyse.

3.4.4.1 onCreate

First of all, the gesture detector is initialised, voice commands and keep screen always on are enabled as well. Secondly, the app downloads the map information from indoo.rs API (see section 3.4.5) using an API-Key and a building id. Finally the map is drawn on the Google Glass screen.

3.4.4.2 Position updated

This method is executed every time the user's position changes and receives as a parameter the new coordinate. We use this method to move the map in case the user is out of range. Afterwards, we need to convert the coordinate from the app fingerprint to a UbikSim coordinate. To do so we have to add a specific offset to each x and y coordinate, invert the x axis and swap both coordinates.

After this is done, we call the AsyncTask *UpdatePosition()* providing the new coordinate (see section 3.4.4.3).

Regarding the routing part, we evaluate the choice of the user about the type of exit where he wants to go, the end of the route is set and the path is drawn.

Finally we call the class *Surface Overlay*(see section 3.4.4.3) to refresh the occupancy rectangles.

3.4.4.3 Update Position AsyncTask

This private class extends an AsyncTask, therefore, it implements *doInBackground* and *onPostExecute* methods. The app will execute the long requests in the doInBackground method, these requests are: a HTTP request to UbikSim webapp to make the simulation

move one step forward and the download of a JSON. By downloading this JSON we refresh the position of the user in the simulation and retrieve the distance towards all of the exits.

The JSON follows this structure:

```
{
  "YARD1": {
    "distance": 120,
    "position": "(27,84)",
    "distanceToEmergency": 114,
    "loadOfExit": 31
  },
  "YARD2": {
    "distance": 340,
    "position": "(272,135)",
    "distanceToEmergency": 177,
    "loadOfExit": 36
  },
  "YARD3": {
    "distance": 186,
    "position": "(116,167)",
    "distanceToEmergency": 18,
    "loadOfExit": 34
  }
}
```

In *onPostExecute* method the app will read the JSON and store the three possible exits (closest, safest or least crowded one).

3.4.4.4 Surface overlay

This class is in charge of drawing the rectangles that represent the number of persons in each area. Before programming the class, we need to know the vertexes of every figure that will be drawn, these points are stored in an array.

This class is composed of three methods: *initialize*, *paint* and *getColor*

- Initialize: This method is executed every time we create an object of this class. We use it to initialize the paint object and to start the *UpdateOccupancy AsyncTask* (see section 3.4.4.5).
- Paint: This method receives as a parameter the canvas to draw on. To begin with, it is necessary to convert all the coordinates to absolute canvas coordinates. Once it is

done, we draw the rectangles using the method *drawRect*. To check what color needs to be drawn, we call the method *getColor*.

- getColor: We use it to check the array where the number of persons of each area are stored. It receives the zone whose occupancy needs to be checked. It returns the color red, yellow or green depending on the number of persons-

3.4.4.5 Update occupancy AsyncTask

Like in *UpdatePosition AyncTask* the app downloads a JSON in the *doInBackground* method. This JSON is read and analysed in *onPostExecute*. This analysis consists on reading the parameter *room* of each agent and increasing the number of persons on each zone if that room matches the selected zones. Here is an example of this JSON:

```
{
  "a77": {
    "positionY": 133,
    "room": "099.G7",
    "positionX": 158
  },
  "a79": {
    "positionY": 182,
    "room": "062.0",
    "positionX": 102
  },
       .....
       .....

  "a75": {
    "positionY": 35,
    "room": "047.0",
    "positionX": 47
  },
  "a74": {
    "positionY": 100,
    "room": "040.0",
    "positionX": 79
  }
}
```

Finally we update the values of the array used to colour the rectangles.

3.4.5 Indoo.rs API

The indoo.rs API provides a basic interface to connect the library [12]. The following figure illustrates the process:

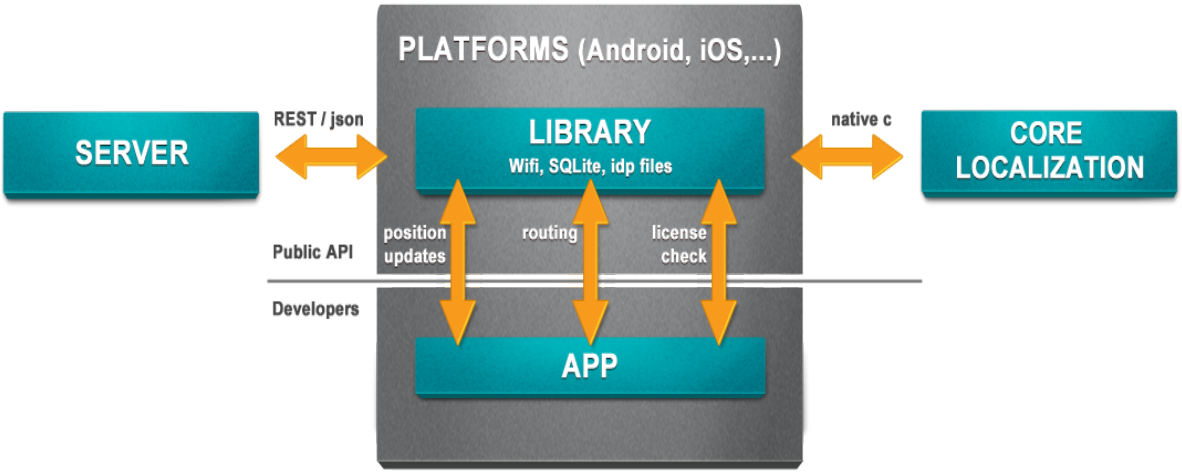

Figure 3.17: Overview of indoo.rs API

Once the application is launched, it will send a request to the indoo.rs server to obtain the building map providing the Indoor.rs API-Key and the Building-ID. The API also allows storing the map in the local storage in case the user does not have an internet connection to load it every time he opens the application, the SDK will check if there is a matching map included in your app every time it is requested to load a building. Afterwards, when the building is loaded, the API will keep updating the position of the user while it moves. In addition, we will draw a route from the current position, which will be in constant updating, to the destination of the user.

The API also provides methods to know when the user has left the building, changed floor or entered some delimited zones.

3.5 Android app

The android app has been developed due to the similarities with the GDK app. The only difference is the user interaction. Obviously, the user control cannot be the same in a device like Google Glass than in a smartphone. Therefore, this android app counts with an *Action Bar* in which features like *Routing* and *Information* are available.

3.5.1 User interface

When the user opens the app, he will be asked to set the type of route that he wishes to follow (closest, safest or least crowded). Afterwards the map will be shown along with the user position and the route. The user is always capable of changing his preference by clicking the button in the *Action Bar*. An information activity explaining a resume of the project is also included. In the following figures these features can be seen:

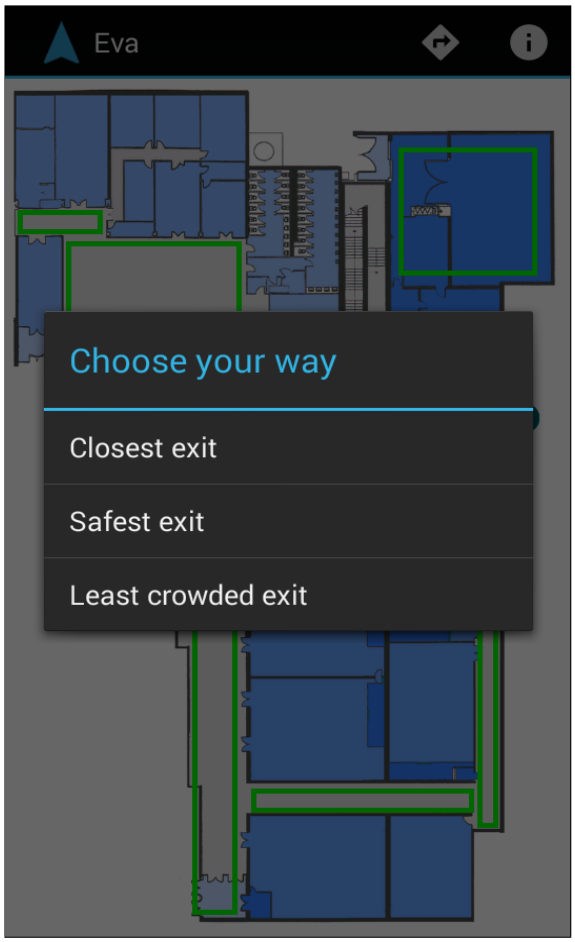

Figure 3.18: Choose route dialog

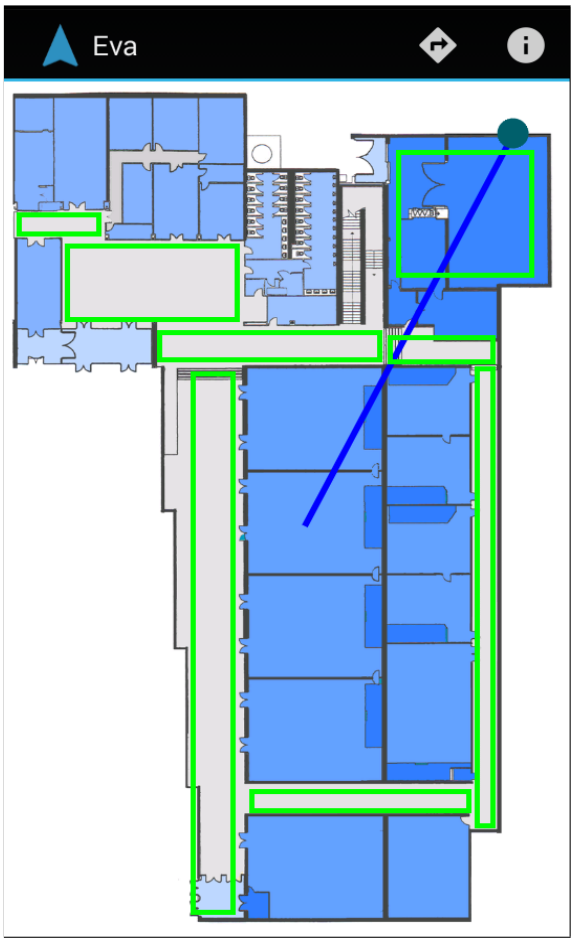

Figure 3.19: Android map

The appearance in a tablet looks like this:

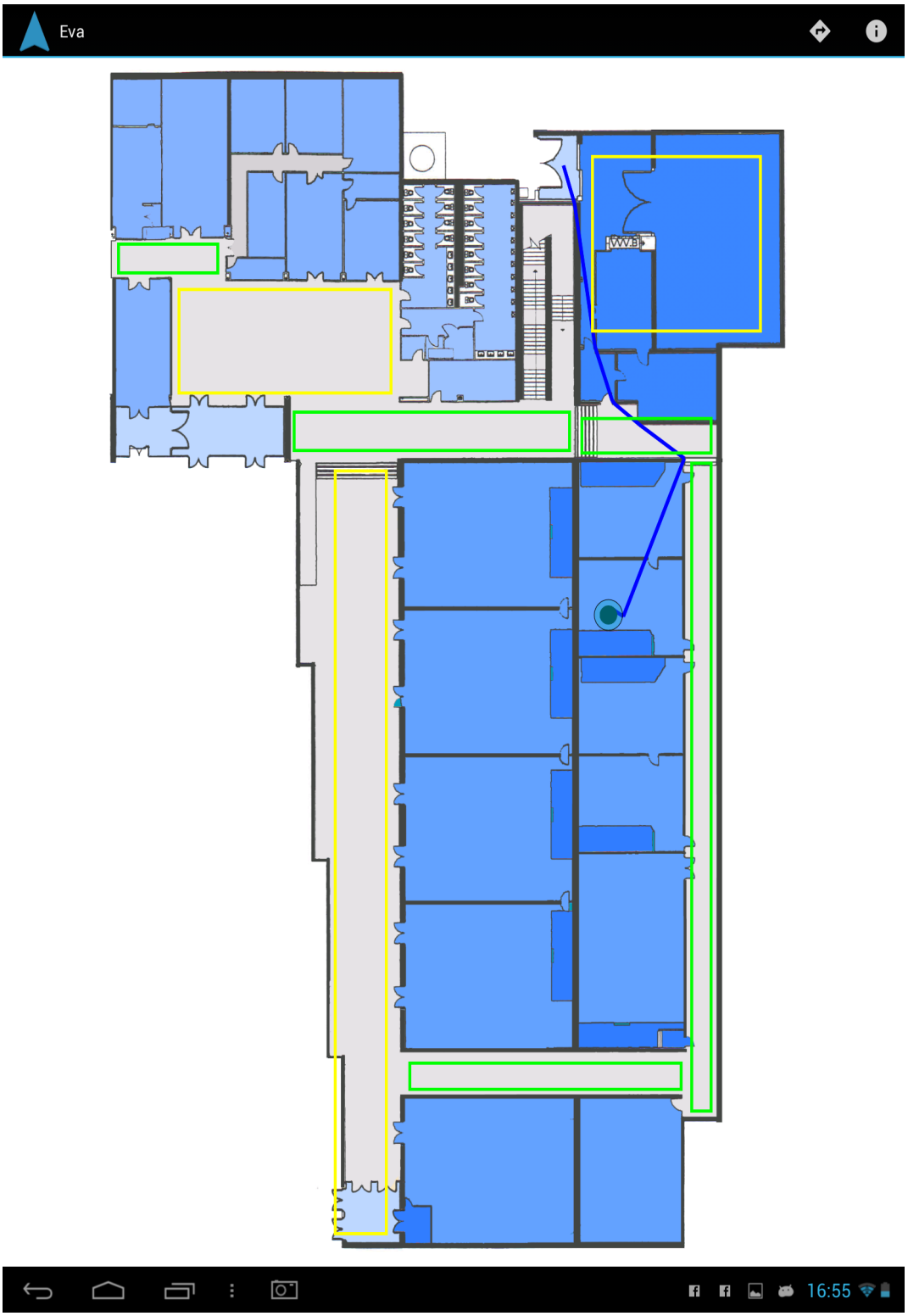

Figure 3.20: Tablet appearance

3.6 Social simulator interaction

Due to the importance of the communication part of the project I will explain it in depth in this section to help the understanding.

The interaction with the social simulator will consist in sending the current position of the user so that it can be represented in the map simulation and receiving the destination of the user and the occupancy of the zones. The whole communication process can be resumed in the next figure.

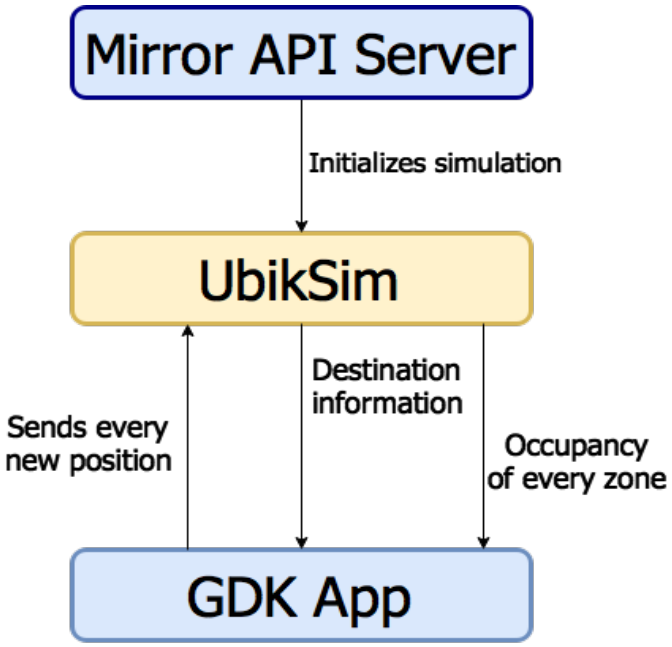

Figure 3.21: Communication process

3.6.1 Mirror API Server interaction

To start with, the simulation needs to be created, this will be done by the Mirror API Server. The facility manager will activate the simulation by launching the first emergency alert to the Google Glass users. The same button that sends the alert to the Google Glass will send a request to the server to initialize the simulation and set the random emergency spot.

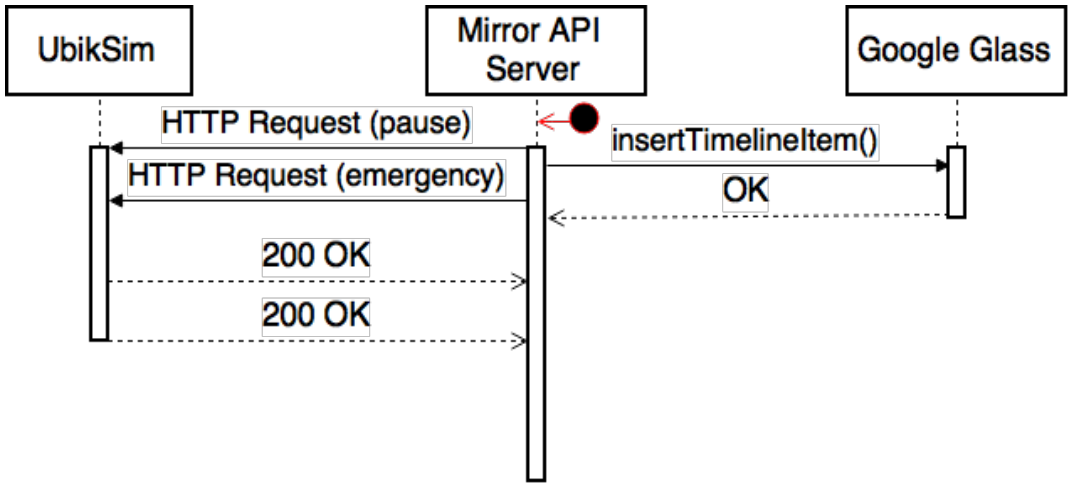

Figure 3.22: Mirror API server communication process

3.6.2 GDK/Android app interaction

When the user starts the evacuation app, it will communicate to the simulator the user's id and position by a HTPP request to the Webapp service of UbikSim. The answer is a JSON that contains the information about all of the exits. Moreover, the app will download another JSON from a different route in the webapp. This last JSON contains the information about the occupancy of all the zones analysed.

The whole process will be repeated every time the user moves and changes position.

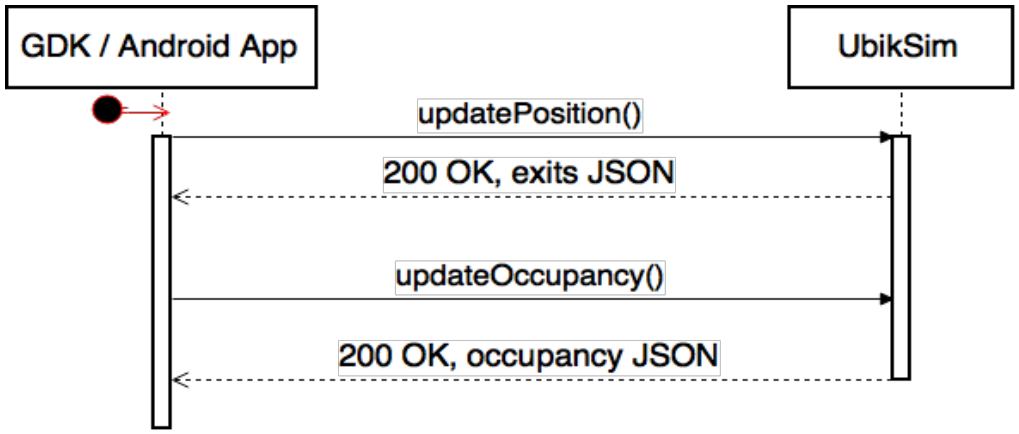

Figure 3.23: GDK/Android app communication process

3.7 Conclusions

In this chapter I have explained the features and objects that this project has along with their communication and relationship.

To sum up, this project counts with a Notification Server that stores the Google Glass users who want to receive the emergency alerts, sends the notifications and activates the simulation. It also includes the adapted webapp version of UbikSim. We spoke about the main feature of this project, the Google Glass app along with the android app and their communication with the simulation as well.

CHAPTER 4

Map Developing

In this chapter we will explain what tools and processes are needed to create the map fingerprint with all the data content.

4.1 Introduction

The map displayed in the app is not a simple JPEG file, therefore, in this chapter I will explain what is the process that let us display a map with information about walls and WiFi spots stating from a PNG file.

This process is mainly done with a tool provided by indoo.rs enterprise called *Measurement Tool*. The next sections will explain step by step the used features of this tool.

4.2 Measurement Tool

This tool is available for free after registering in the indoo.rs webpage [13]. All the features available are described in the following document [14]. However in this chapter I will only explain the relevant ones for this project.

4.2.1 Creating a new building

Just clicking *New building* will make another window appears, where we will have to name the building and optionally set the location and rotation to enable the orientation feature in the map.

Then we have the possibility to add more than one floor, but in this project we will use only one. Finally we will select the PNG file to be loaded.

The last step is to adjust the scale that can be done by selecting two points in the map and writing the real measure.

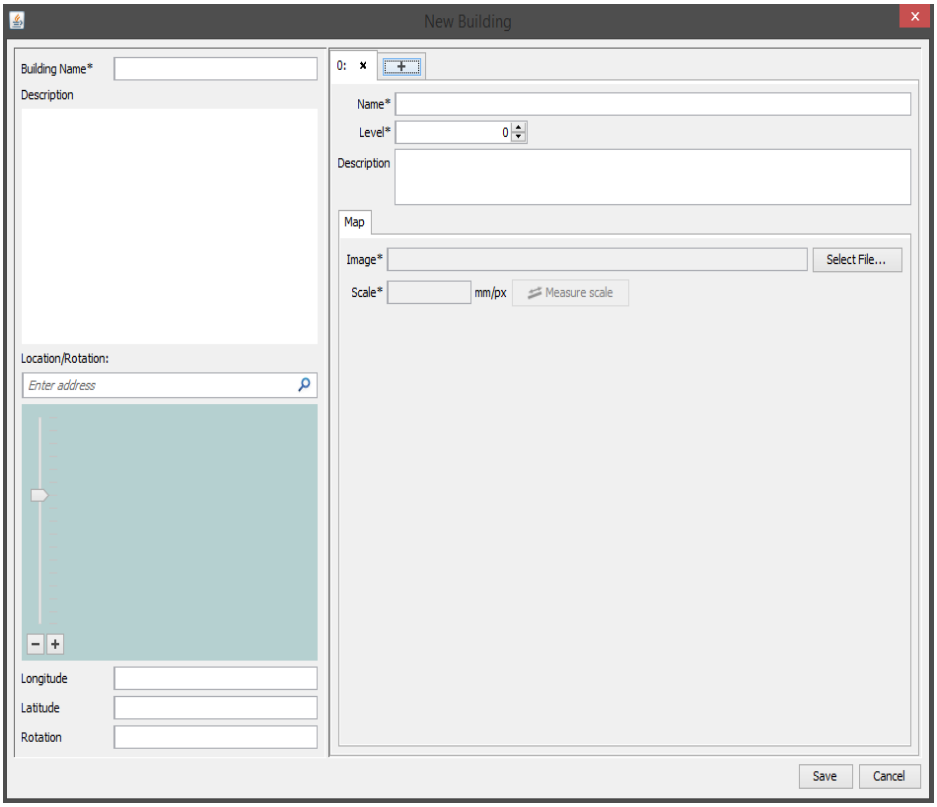

Figure 4.1: New building window

4.2.2 Drawing the walls

This process is simple, we will have to click over the map drawing lines to point where the walls are placed. The result looks like this: (walls are coloured in red)

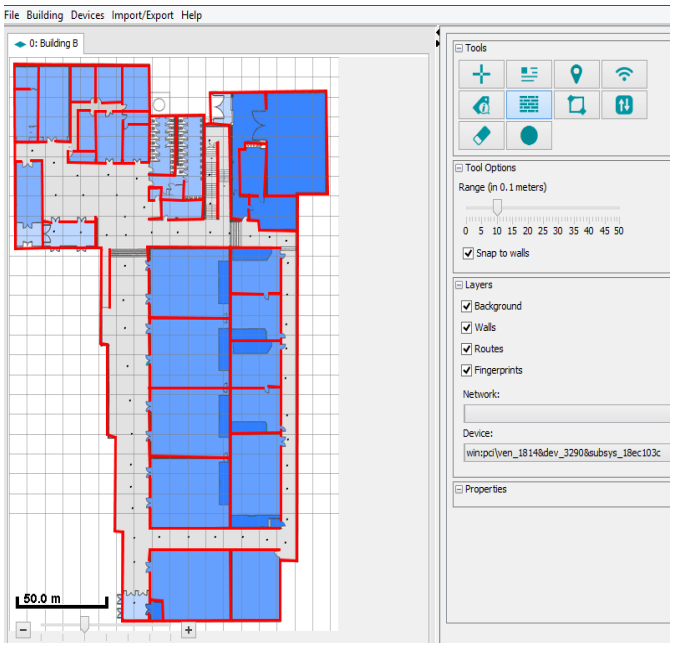

Figure 4.2: Drawing walls

4.2.3 WiFI Measure points

To record the WiFi signal in some points of the building we just need to click over the map while we are in real in the same position. I have used the same WiFi adapter that my laptop has to measure the spots. Bluetooth signals can be recorded as well, but as described in the previous chapters, we will not be using this technology. The result looks like this, every point is one measure spot:

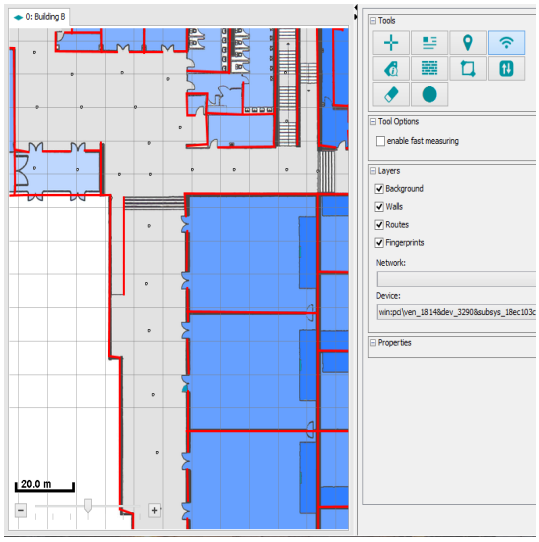

Figure 4.3: WiFi Measure points

4.2.4 WiFI Heatmap

A different feature that can be interesting is the WiFi heatmap, as we have recorded the WiFi signals around the building, we can see where the WIFI signal of each SSID is stronger.

For example we can see the coverage of the GSI-WIFI-B-205 or one of the DIT WiFi

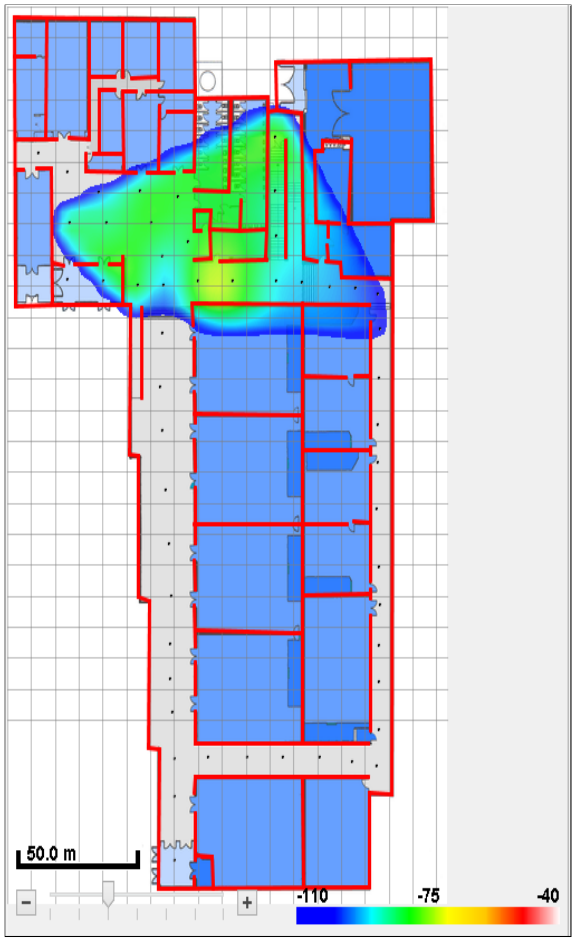

Figure 4.4: GSI WiFi

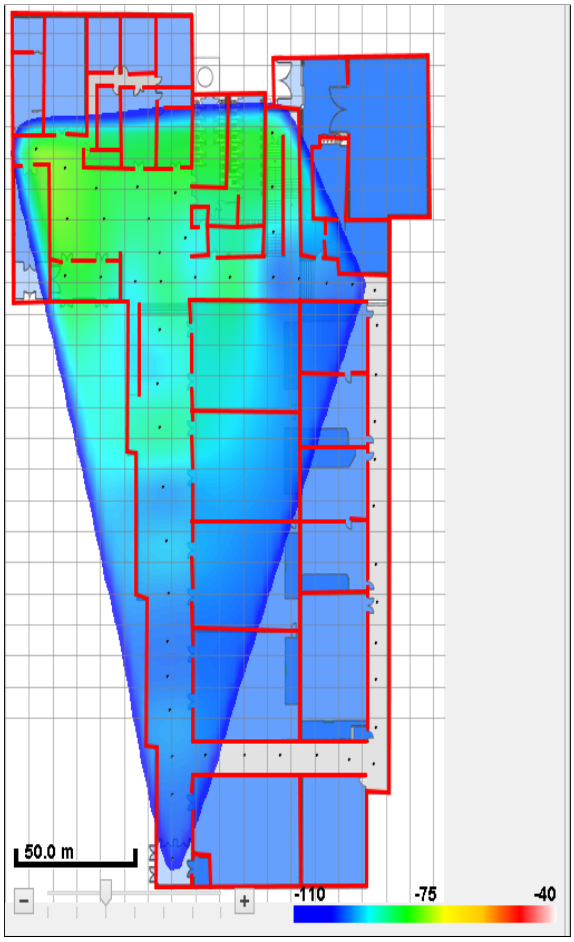

Figure 4.5: DIT WiFi

4.3 Conclusions

In this chapter we have seen how the map is developed using the Measurement Tool from indoo.rs. By uploading a PNG file we can set the scale, walls and WiFi sampling. Thanks to these features we can be located and routed to an exit using the evacuation app. Moreover it offers the possibility to see in a heatmap the strength of every WiFi signal.

CHAPTER 5

Case study

This chapter describes the scenario where our system could be used along with the solutions offered to the user, covering all the features of this project.

5.1 Problem and scenario

The problem we are facing is the ignorance of the building in an emergency situation. The person can be lost or confused inside the building losing important time to get out of the building. An example could be a visitor in an airport, hospital or university.

To solve this problem, we will imagine an scenario where the user is somewhere inside the B building of the School of Telecommunications Engineering of the Polytechnic University of Madrid. The goal will be to evacuate the building as fast as possible letting the user be part of the evacuation.

5.2 Google Glass user

The user will receive a notification in the Google Glass, alerting him about the emergency and urging him to leave the building, offering to use the evacuation app as well.

Figure 5.1: Notification

The user can access the app by tapping on the notification, through the voice menu or tactile menu. Every action will lead the user to the map view. The map will appear right from the beginning. The default destination will be the closest exit. However if the user wants to change the destination he can do it by saying "ok glass" or doing a long tap on the tactile side piece of the Glass.

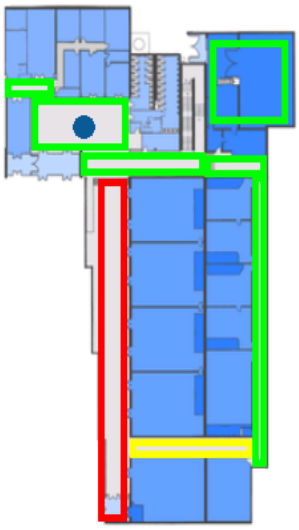

Figure 5.2: Google Glass app screen

The user can control the map by swiping forwards or backwards to increase or decrease the zoom. To scroll up the map he can swipe up. The opposite can be done by a simple

tap. The swipe down is reserved for closing the app. The user's position and the route will be refreshed every time the user moves in the building.

The feature of changing the route type is available during the whole use period of the app.

5.2.1 Enabling notifications

If the user wants to receive in his Google Glass the notifications about the emergency situations, he will access to the server.

First of all he will have to log in with the Gmail account that is linked with the Google Glass. He will be asked to accept some permissions. Secondly, he will have to head to the users part of the welcome page, where the user will see the user creation field set.

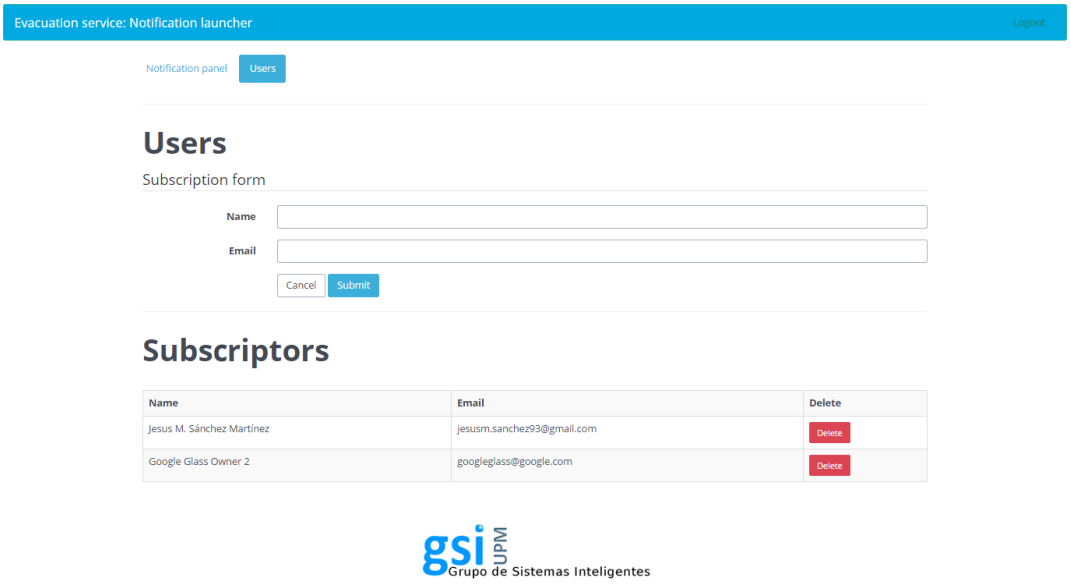

Figure 5.3: User creation

Finally, the user will fill the fields of name and email, for identifying purposes. By clicking on the Submit button the user will be stored in the database.

5.3 Smartphone/Tablet user

The user can be launching the app in a smarthphone or tablet device as well. However the access could not have been done by the voice menu or a static card because these features are not available in devices like smartphone or tablets.

The map will not appear right from the beginning like in the Glass device. A dialogue will appear before asking the user to set the route preference: closest, safest or least crowded one. After the user selects one of these options, the map will be available. He can zoom or move it with simple movements like in any other map such as Google Maps, etc. The feature of changing the route type is always available by pressing the Action Bar icon.

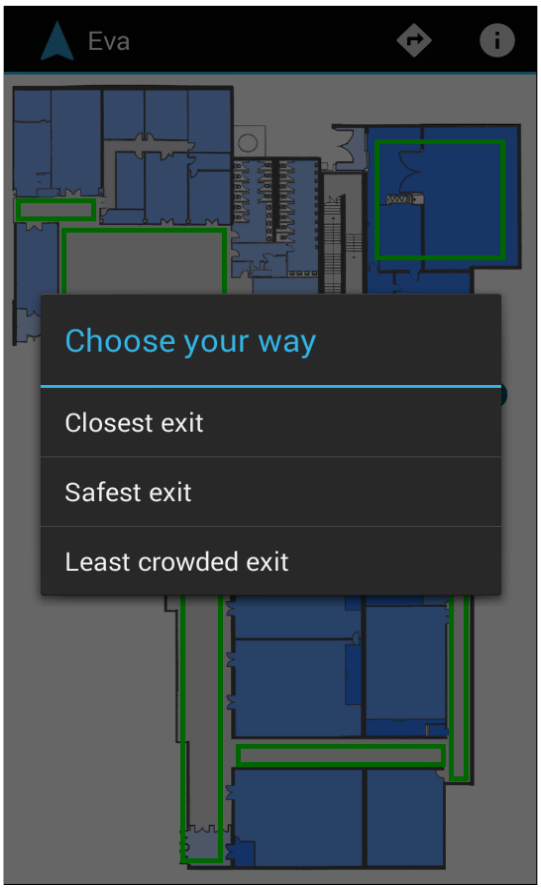

Figure 5.4: Choose route dialog

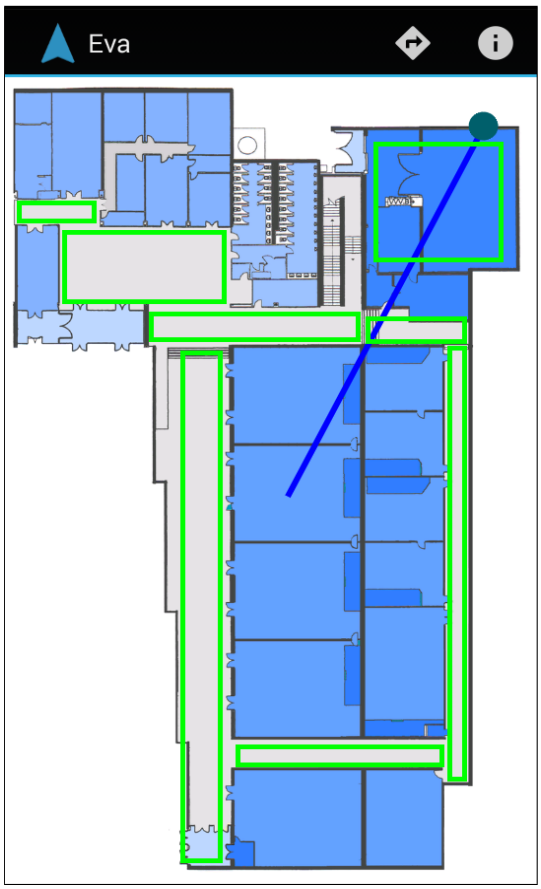

Figure 5.5: Android map

5.4 Facility manager

The facility manager will want to warn as many users as he can about the emergency. With that purpose he can access the Mirror API server. He will see the following webpage:

Just by clicking one of the buttons all the users that have previously register on the system will receive the alert.

The server might return an error if the social simulation is not set, but the notification will be sent anyway.

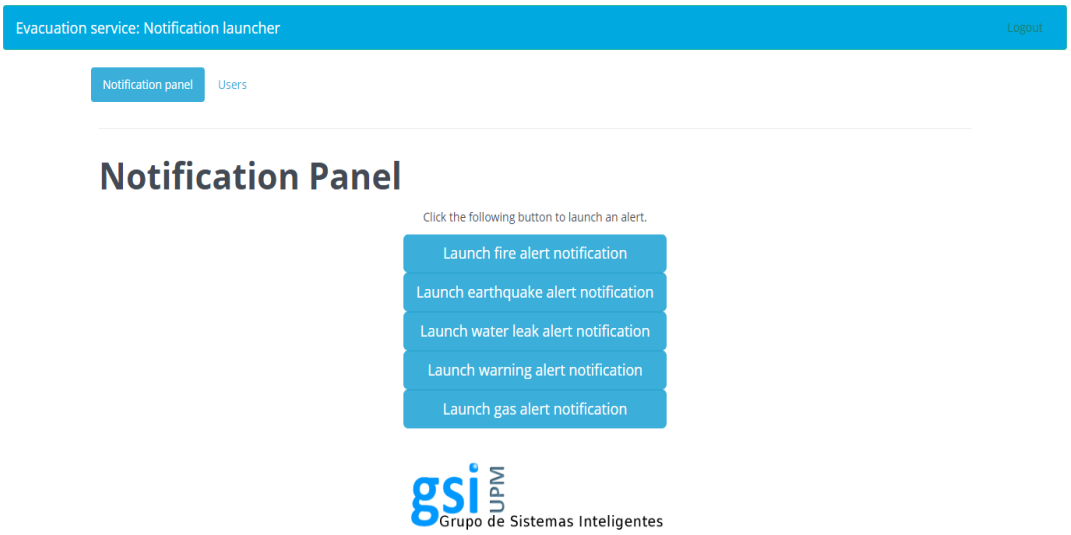

Figure 5.6: Dashboard

5.5 Conclusions

In this chapter we have seen the possibilities and features of this project as well as the problem that we are trying to solve. We have explained how in a real scenario the system could be used, including the points of view of each agent that could take part in the evacuation.

To sum up, we offer an app to help the user evacuating the building, through the closest, safest or least crowded exit, depending on his decision. This app is available in Google Glass and an adapted version in android smartphones and tablets. We have also developed a notification server to alert the user about the emergency.

CHAPTER 6

Conclusions and future work

In this chapter we will describe the conclusions extracted from this thesis, problems, achievements and suggestions about future work.

6.1 Conclusions

In this project we have created a building's emergency evacuation system on Google Glass and smartphones to make the process of evacuating a building easier. When visiting new buildings we are not aware of the positions of each and every exit, this system can help users that do not know the building in depth to exit in a faster way.

This system has 3 components, the main one being the Google Glass app. Whether this app or the smartphone app will lead the user towards an exit. To let the user have a more interesting role in the simulation he will be able to change the type of route that he wants to follow. We have also implemented a server to take advantage of the new features of Google Glass along with a MongoDB database to store all the users of the system.

In the following sections I will describe in depth the achieved goals, the problems faced and some suggestions for a future work.

6.2 Achieved goals

In the following section I will explain the achieved features and goals that are available in this project.

Developing a Google Glass app This was the main goal of the project, we had to develop an app in a device under development. This means that few examples are available to get to know the technology or even to start from them. Despite this, this goal has been accomplished and the app is completely functional.

Sending alerts to Google Glass The feature of alerting the Google Glass user about the emergency in the building is really interesting. Thanks to Google's Mirror API server this goal has been achieved and we can send customised notifications to the user.

Locating the user using WiFi signals The indoor positioning was crucial. We needed to be able to locate the user, GPS positioning is not available indoors, therefore we had to find another technology. Indoo.rs API let us achieve this goal using WiFi signal positioning.

Drawing a route towards an exit In order to guide the user inside the building, we needed to overlay a path on the building fingerprint. Using the indoo.rs API we have achieved it.

Giving to the user the option to choose To make the system more participative, it was interesting to include a feature which lets the user have a more important role. This was accomplished by creating an interactive menu in which the user can choose the type of route to follow.

Synchronising with a social simulator To recreate the human behaviour, it was essential to include a framework that can simulate persons in the building. By using the adapted webapp version of UbikSim we have achieved this.

Drawing occupancy areas Letting the user know where most of the persons were was a really interesting feature for the system. This way the user can realise what areas will be crowder that others. This has been accomplished using UbikSim as well.

6.3 Problems faced

During the development of this project we had to face some problems. These problems are listed below:

- Google Glass device: The main device where this project will be executed is Google Glass. This device is still under development, this fact means that it is still a limited hardware, therefore problems such as overheating or processing delays are unavoidable.
- Connection problems: The goal building is the building B of the Technical School of Telecommunication Engineering and there is not a single stable WiFi network that we can use for the communication from the app to UbikSim. For this reason, some connection problems can appear when moving inside the building due the connection loss.
- Location problems: We are using WiFi positioning because it was easier and faster than any other method. This technology is not 100% accurate, therefore some inaccurate positions might appear.
- Indoo.rs API: Because we are using the free plan of this API, we can only display the map in 5 different devices and only one building. For this reason the API limits our system.
- Virtual reality: This was an interesting feature that we wanted to include at the beginning of the project, but looking all the available options, none of them provided virtual reality for free.
- userId storing: In the database we store the userId, an unique id linked to every Gmail account that is used to identify the Google Glass user. However, this userId is not constant, during the development of this project the userId that we are using has changed a couple of times. To solve this it is necessary to delete and create again the user in the web service.

6.4 Future work

In the following section I will explain the possible new features or improvements that could be done to the project.

Google Glass redesign The actual Google Glass model is still a prototype, some months ago Google seemed to have cancelled the project. However, some other news point to the contrary [15] [16]. According to these news an improved model is up to come out, with an appearance redesign and a technical improvement. It would be interesting to adapt this app to the new model.

Bluetooth beacons Even though we have chosen WiFi positioning, Bluetooth positioning is available as well. This method is supposed to be more accurate than WiFi positioning but a higher initial effort is required because some extra devices, called Bluetooth beacons, are needed. These beacons would have to be placed all around the building displayed. A possible improvement could be changing the positioning technology to this one.

Show emergency In the current version the user can not see where is the emergency in the building. An interesting new feature could be the possibility of displaying the emergency on the map allowing the user to see it. This could be done by requesting to UbikSim the emergency position, converting the coordinates and drawing it on the map.

Bibliography

[1] E. Redmond, *Programming Google Glass: Build Great Glassware Apps with the Mirror API and GDK*. The Pragmatic Programmers, 2015.

[2] C. Janssen, "Radio frequency fingerprinting," http://www.techopedia.com/definition/9078/radio-frequency-fingerprinting-rf-fingerprinting, accessed April 07, 2015.

[3] J. T. S. Yi Han, Erich P. Stuntebeck and G. D. Abowd, "A visual analytics system for radio frequency fingerprinting-based localization," Georgia Institute of Technology, Tech. Rep., 2009, http://www.cc.gatech.edu/ stasko/papers/vast09-rf.pdf.

[4] Google, "Google maps indoors information on google's official blog," http://googleblog.blogspot.ca/2011/11/new-frontier-for-google-maps-mapping.html, accessed March 23, 2015.

[5] V. Agafonkin, "Leaflet.js info website," http://leafletjs.com/, accessed March 23, 2015.

[6] "Indoor.io website," https://indoor.io/, accessed March 23, 2015.

[7] MIT, "Wearscript documentation website," http://www.wearscript.com/en/latest/, accessed March 23, 2015.

[8] Google, "Google's mirror api documentation," https://developers.google.com/glass/v1/reference/, accessed March 26, 2015.

[9] G. Inc., "Google's quickstart project documentation," https://developers.google.com/glass/develop/mirror/quickstart/index, accessed March 26, 2015.

[10] D. Selmanovic, "Google's mirror api explanation," http://www.toptal.com/google-glass/mirror-api-google-glass-for-web-developers, accessed March 26, 2015.

[11] E. Serra, "Ubiksim 2.0 documentation," https://github.com/emilioserra/UbikSim/wiki, accessed March 30, 2015.

[12] Indoo.rs, "Indoo.rs api documentation," https://my.indoo.rs/javadoc/, accessed March 30, 2015.

[13] indoo.rs, "Measurement tool," https://my.indoo.rs/indoors/rest/download/free/mmt-installer-x64.exe, accessed July 2, 2015.

[14] Indoo.rs, "Measurement tool guide," https://my.indoo.rs/javadoc/mmt_guide/, accessed July 2, 2015.

[15] T. V. Nathan Ingraham, "The next google glass is coming soon," http://www.theverge.com/2015/4/24/8494399/eyewear-maker-luxottica-says-the-next-google-glass-is-coming-soon, accessed June 30, 2015.

[16] W. Cade Metz, "Sorry, google glass isn't anywhere close to dead," http://www.wired.com/2015/02/sorry-google-glass-isnt-anywhere-close-dead/, accessed June 30, 2015.