Bachelor's Thesis

Geolocalization and routing in complex multi-floor environments

UC Odisee

Department of Engineering Technology - Electronics and Information Technology,

Specialisation Information Technology

Joachim Cardoen

2018

I want to thank my foster mother, Miek Roels for her devotion to guide me through the process of writing this thesis. I want to thank her and both Nathan Verhoeven, one of my closest friends, as well as Cedric De Roover (Senior Consultant Manager at IBM) to correct my thesis where necessary. I also want to thank my mentor Sven Sanders for answering my e-mails day and night whenever I had got stuck.

Abstract

The aim of this project is to realize a mobile application for patients of the hospital CHC Saint-Jean in Liege that will enable them to register, view appointments and find their way in the hospital. First the existing application for consulting scheduled appointments and registration is reworked to a native iOS mobile application (mobile app in short). This feature is extended with the option to view the information of the hospital and set reminders for any appointment in the near future. Second an indoor location framework is added to the reworked mobile app that shows the route to the place of appointment. After implementing the geolocation and routing, an optimization method is implemented: the ant colony optimization algorithm. The ant colony algorithm determines the shortest path to the place of appointment based on some factors such are: amount of visitors in some corridors and hallways and the amount of stairs a patient has to climb (a patient's mobility). Using the mobile app the hospital can improve their safety procedures by notifying users of the application of any problems in the hospital, these problems can be but are not limited to: fire, malfunctioning of the elevator and electrical outage.

Contents

Chapter 1

Summary

Chapter 2

Project Specification

2.1 Project Description

2.1.1 Technical Design Specs

The communication with the hospital happens with a server provided by IBM and the hospital's API. This means that the mobile device interacts with a intermediary server from IBM which in its turn communicates with the API of the hospital. This model is an example of a highly reusable architecture. If another hospital needs to be attached to the IBM server, only a small 'translator' for the endpoints of the additional hospital's API needs to be created whilst the structure of the IBM server remains the same.

2.1.2 Features

The main features of the project are specified below [4]:

1. Login with hospital provided credentials;
2. Synchronization of appointments with the hospital;
3. Ability to set reminders for an appointment;
4. See the hospital's location (and venues) as well as contact details;
5. Allow geolocalization inside the hospital;
6. Provide feedback after an appointment;
7. Localization in French, English and Dutch;
8. Available on both iOS and Android
9. Distributed in the Apple Store and Google Play Store;

2.2 Development Guidelines

To attain uniformity in the codebase of iOS and Android a 'Development Guidelines' document is written, this document can be found as an appendix.

Chapter 3

Proof of Concept

3.1 User Interface

Considering the application is only a proof of concept (PoC) there is almost no focus on the user experience (UX) nor on the user interface (UI). To at least give a slight indication of what information needs to be displayed where, a UI mock-up is created in Adobe Xd, Adobe Xd is a lightweight, rudimentary visual editor that enables designers to quickly develop and share interactive prototypes. A few example screens of the UI prototype can be found below.

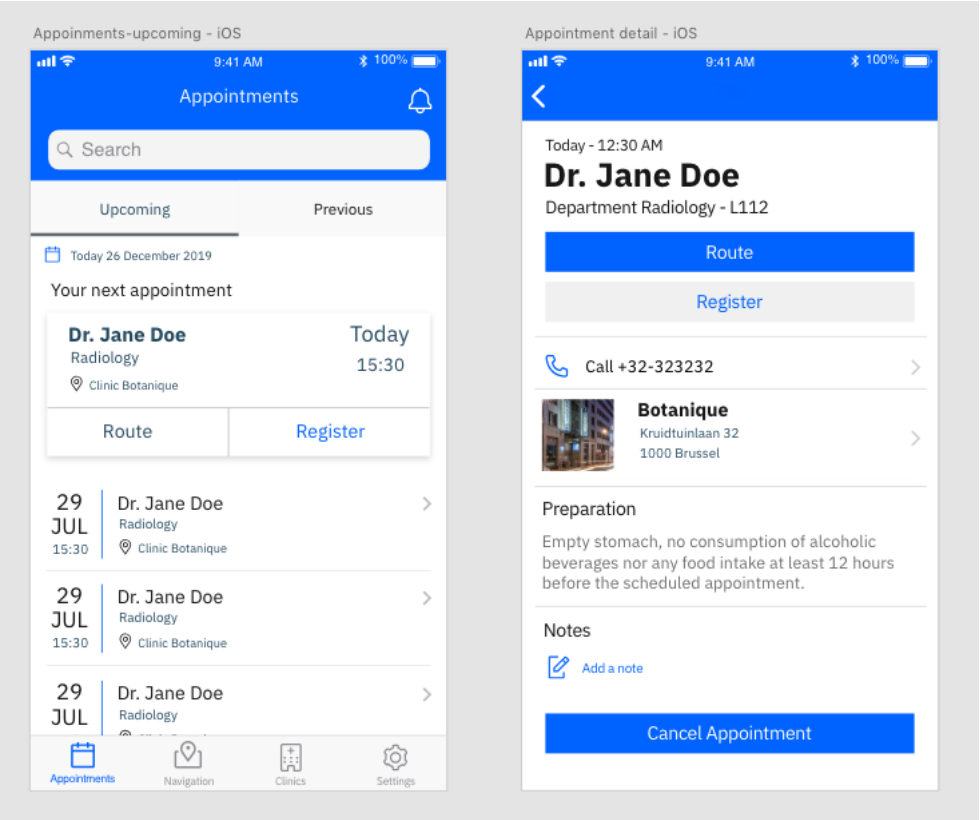

Figure 3.1: User interface of the appointments and detailed view for iOS

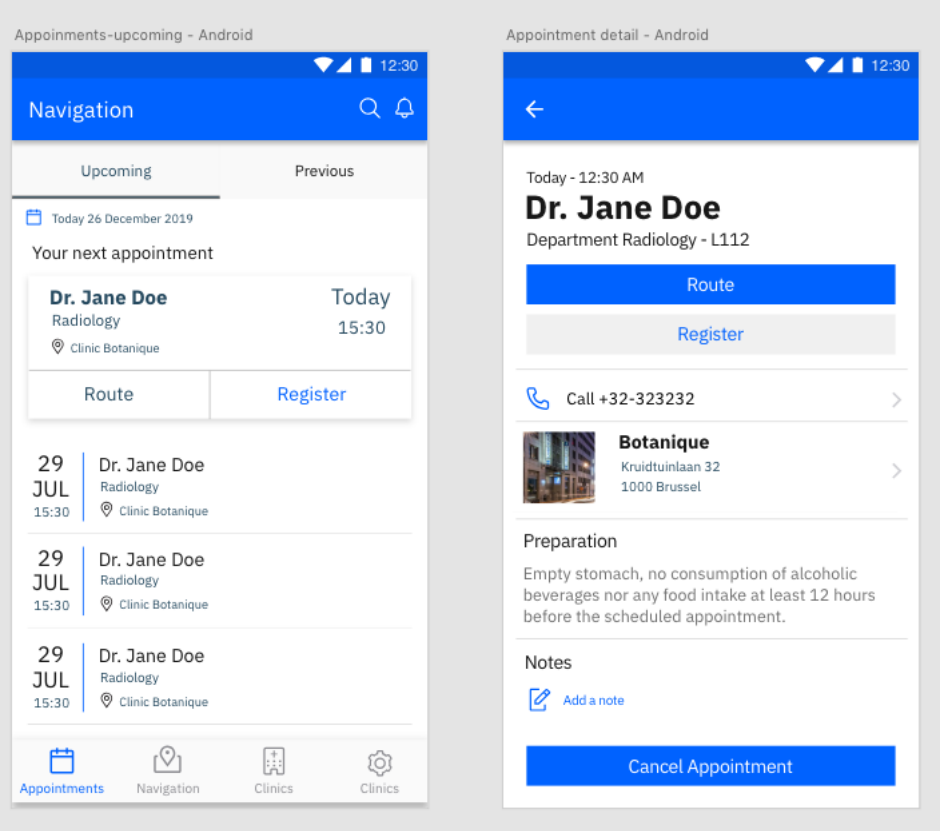

Figure 3.2: User interface of the appointments and detailed view for Android

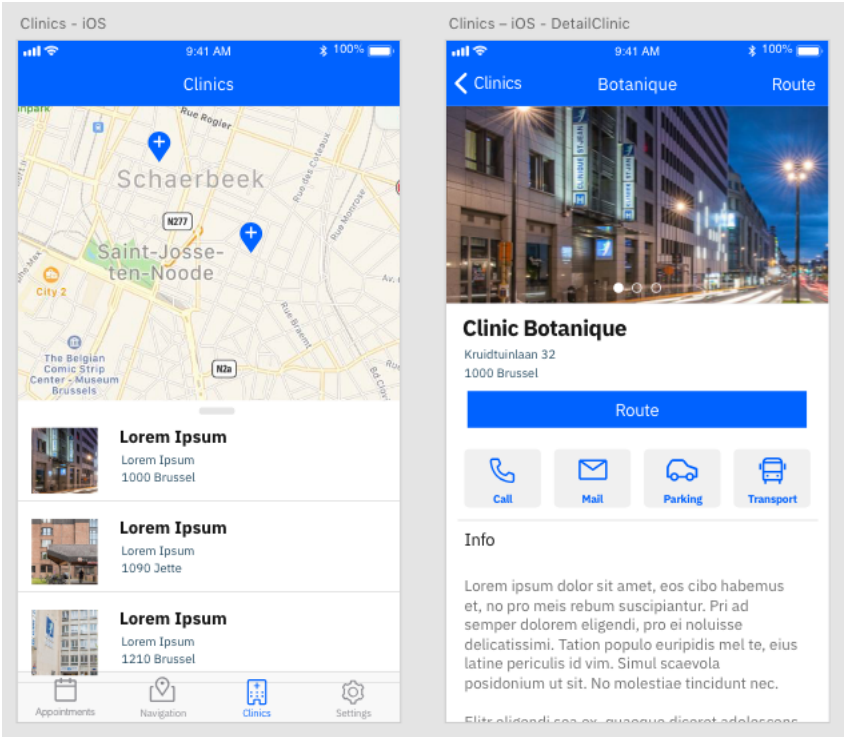

Figure 3.3: User interface of the hospital venues and detailed view for iOS

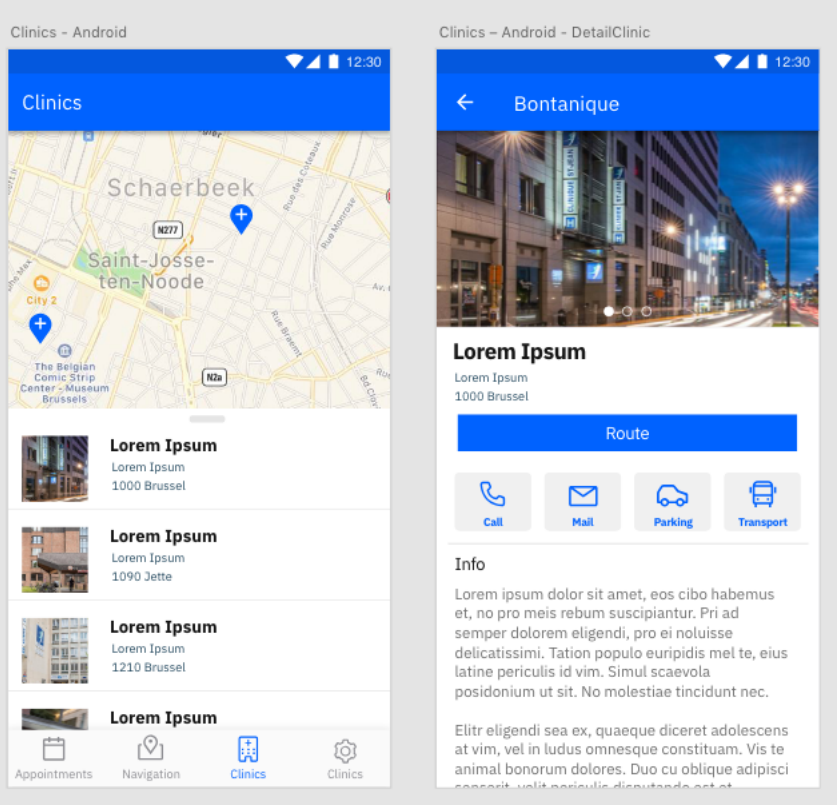

Figure 3.4: User interface of the hospital venues and detailed view for Android

3.2 Architecture

The emphasis of the PoC is on developing it in such a way that it should be easy to re-implement the application elsewhere. The PoC is developed in the two current formats for mobile development: iOS and Android. This bachelor's thesis will cover the implementation of the Android architecture.

3.2.1 General Architecture

Entities

The specific entities used throughout this application are:

- appointment;
- hospital;
- venues - the different locations of a hospital;
- address;
- additionalinformation - meta for a hospital and venues;
- department;

UML Diagram

3.3 Database Communication

3.3.1 Testing API

To aid in testing the database a test API is programmed using a Node.js framework: Express. it is a very simple tool to create web APIs ready for consumption [3]. The testing API is structured according to the entities specified in the UML diagram with the corresponding relations. For this specific application there a couple of endpoints exposed for requests:

- GET /appointments - this returns a JSON array containing test appointments;
- GET /hospital - this returns information about the hospital such as address, contact details and venues;
- GET /doctors - this fetches all the doctors present in the hospital records;
- GET /departments - this returns all the departments available in the hospital and its venues;

Faker

Instead of using ad random numerical combinations or lorem ipsum texts, a library called Faker is used to generate different random values such are names, addresses, e-mail addresses and phone numbers. Faker is available for almost every general purpose language and is easy to use. It is always easier to work with representative data than it is to work with 'lorem ipsum' or '123456789' [2].

3.4 Android Architecture

3.4.1 Separation of concern: Dependency Injection

3.4.2 Dependency Injection: Restaurant Analogy

3.4.3 Types of Dependency Injection

3.4.4 Type used in the mobile application

3.4.5 Dagger2 - Dependency Injection Library

3.4.6 Kotlin Language

3.4.7 Lifecycle Events

For the duration of the runtime of the mobile application (from the moment the app is opened until it is closed) some events occur that are typical for an Android mobile application. A brief summary of these events is listed below (in chronological order) [1]:

- onCreate() - When the activity is launched (This can happen after the onDestroy() event);
- onStart() - When the activity is visible to the user;
- onResume() - When the user returns to the activity after an onPause() event occurs;
- onPause() - When activity is no longer visible;

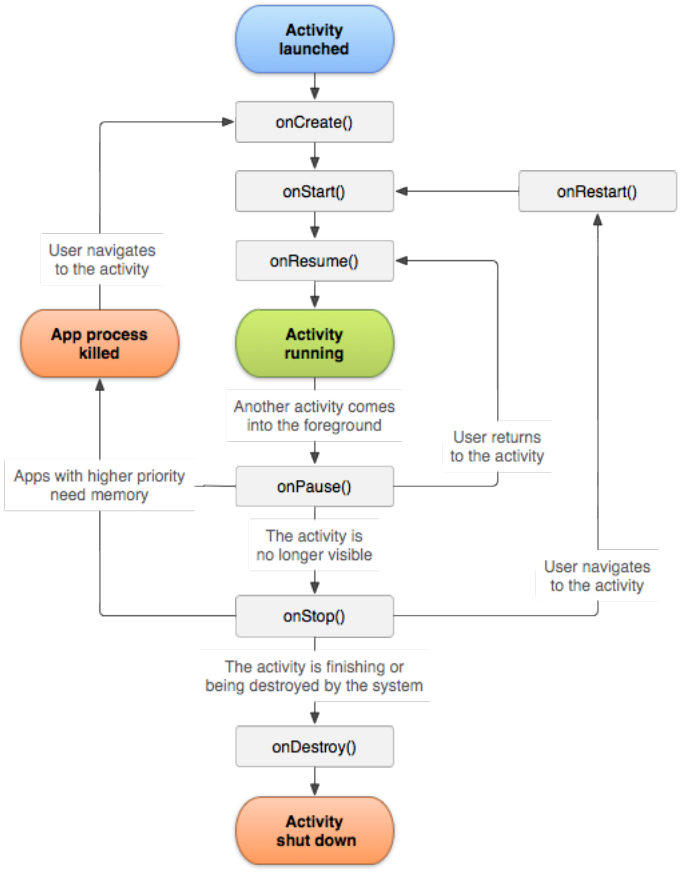

Figure 3.5: Android Activity Lifecycle Schematic [1]

- onStop() - When the activity is finished or destroyed;
- onRestart() - When the activity is restarted after a stoppage;
- onDestroy() - When the activity is shut down;

The lifecycle of an activity is an important factor to take into account whilst the application is being developed. This means a certain level of persistency is required for an optimal user experience.

Bundles & Saved State

The way in which the onCreate() method is implemented allows a developer to declare a Bundle, which is an object that contains key-value pairs, that is used to restore an activity's previous state. If no such state exists then the Bundle will be equal to null. The Bundle object that is passed to an activity in the onCreate() method should only contain specific information such as user interactions: form fields, position on the screen and sometimes navigational properties. The main usage for this technology is when an activity gets paused or stopped, this means the OS (operating system) can freely destroy any activities [5].

3.4.8 SQLite

Another way to persist data throughout the lifecycle of an application is to use the (smart)phone's local storage. Each application can create a new local, file-based database using SQLite. SQLite is a transactional and serverless db (database), which means it is optimal for storing user-specific data. The

fact that it is indeed a transactional db means that upon failing to save data it will rollback the entire operation [6].

3.5 Testing Application Programming Interface

3.6 Tools and frameworks used

3.7 MapWize

MapWize is a service that digitalizes architectural plans and makes them interactive. The generated map from MapWize is the one used throughout the development of the application for hospital CHC Saint-Jean in Liege. The main reason for using this service is to take care of the digital mapping of the hospital itself, which is not a task that should be completed by developers. The digital map will be used to show the routes from point A to point B, applied to this case: from the hospital's registration office to the place of appointment.

3.7.1 MapWize SDK

MapWize provides developers with a ready-made SDK for both iOS and Android. This SDK covers all important methods to

3.8 Cisco Connectected Mobile Experiences integration

The manner in which the position of a patient is retrieved is based on the nearest WiFi router of Cisco.

3.8.1 Function of Cisco CMX

3.9 IndoorLocation Framework

Chapter 4

Conclusion

Appendix A

Development Guidelines

Appendix B

Development Guidelines

Appendix C

Development Guidelines

Bibliography

[1] Android Developer. *Transformations — Android Developers*. 2019. URL: https://developer.android.com/reference/android/arch/lifecycle/Transformations (visited on 03/06/2019).

[2] Daniele Faraglia. *Welcome to Faker's documentation! — Faker 1.0.2 documentation*. 2014. URL: https://faker.readthedocs.io/en/master/ (visited on 03/08/2019).

[3] Express. *Express - Node.js web application framework*. 2019. URL: https://expressjs.com/ (visited on 03/08/2019).

[4] IBM. *Med App Project Specification*. 2018.

[5] James Halpern. *Android – SavedInstanceState Bundle FAQ*. 2012. URL: https://content.pivotal.io/blog/android-savedinstancestate-bundle-faq (visited on 03/06/2019).

[6] TutorialsPoint. *SQLite Tutorial*. 2019. URL: https://www.tutorialspoint.com/sqlite/ (visited on 03/06/2019).