



**Screen missing patients using smart and innovative  
solutions to eradicate tuberculosis**

EPCON at Blended-AIM

2019 / 2020

**Diogo Capela**



# **Screen missing patients using smart and innovative solutions to eradicate tuberculosis**

EPCON at Blended-AIM

2019 / 2020

**1171316**

**Diogo Capela**



**Degree in Software Engineering**

June 2020

ISEP Supervisor: **Nuno Escudeiro**

External Supervisor: **Vincent Meurrens**



# Acknowledgment

I would like to thank professores Ana Barata for promoting the Blended-AIM program to the students and professor Nuno Escudeiro for all the support and availability provided during its execution. To Vincent Meurrens, the CEO of EPCON, who accompanied us through the time of the project and attended all our weekly meetings brainstorming ideas, testing features and solving problems. To the professionals at EPCON, mainly Matthys Potgieter, Fábio Neves and Anet Potgieter who worked alongside us to make the project go live. And to all my colleagues and team mates, Wajdan Ali, Jannik Timmer, Frauke de Vry, Robin Verbeelen, Euan Barrett, Sivan Mustafa, Irini Tsitsekidi, Yamilla Cachuela, Narcisa Duarte and Anthony Onimisi for the greatest companionship in this project. Finally, I want to thank the university (ISEP) and the software engineering department (LEI) for the conditions that were provided for the realization of this project.



# Contents

<b>Abstract</b>	<b>IX</b>
<b>List of Figures</b>	<b>XI</b>
<b>Glossary</b>	<b>XIII</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Organization description . . . . .	1
1.2 Problem description . . . . .	1
1.3 Approach . . . . .	2
1.4 Contributions . . . . .	2
1.5 Work planning . . . . .	3
1.6 Technologies used . . . . .	3
<b>2 State of the art</b>	<b>5</b>
2.1 Similar solutions . . . . .	5
2.2 Web application development context . . . . .	6
2.3 Mobile application development context . . . . .	8
<b>3 Solution analysis and design</b>	<b>11</b>
3.1 Problem domain . . . . .	11
3.2 Functional requirements . . . . .	12
3.3 Domain modeling . . . . .	14
<b>4 Solution development</b>	<b>15</b>
4.1 Implementation description . . . . .	15
4.2 Backend architecture . . . . .	16
4.2.1 User authentication . . . . .	17
4.2.2 Background services . . . . .	18
4.3 Web client structure . . . . .	18

4.4	Android client structure . . . . .	25
4.5	Tests . . . . .	28
4.6	Releases and deployment . . . . .	29
4.6.1	Version 0.1.0 . . . . .	30
4.6.2	Version 0.2.0 . . . . .	30
4.6.3	Version 0.3.0 . . . . .	31
4.6.4	Version 1.0.0 . . . . .	31
<b>5</b>	<b>Conclusion</b>	<b>33</b>
5.1	Objectives achieved . . . . .	33
5.2	Limitations and future work . . . . .	33
5.3	Final assessment . . . . .	34
	<b>Bibliography</b>	<b>35</b>



# Abstract

This report aims to document the project carried out in the 2020 edition of the Blended-AIM program, within the scope of the PESTI curricular unit, in the third year of the degree in software engineering of ISEP.

The main focus of the project was to develop on top of existing AI technology to help EPCON screen missing patients using smart and innovative solutions to eradicate tuberculosis.

**Keywords (Subject):**

Computer Aided Diagnostics, Tuberculosis

**Keywords (Technology):**

JavaScript, Node.js, Express, PostgreSQL, React, Redux, Android, Java, Kotlin, HTML, CSS



# List of Figures

2.1	Web application framework popularity by GitHub stars . . . . .	7
2.2	Mobile hybrid framework popularity by GitHub stars . . . . .	9
3.1	EPCON demo web application . . . . .	11
3.2	Use case diagram . . . . .	12
3.3	Domain model . . . . .	14
4.1	System architecture . . . . .	15
4.2	Database schema . . . . .	16
4.3	Backend server architecture . . . . .	17
4.4	Web client structure . . . . .	19
4.5	Redux store . . . . .	20
4.6	Homepage of the web application . . . . .	21
4.7	Empty screenings list . . . . .	22
4.8	Submit a screening page . . . . .	23
4.9	Screening details page . . . . .	24
4.10	Android client structure . . . . .	26
4.11	Android client authentication . . . . .	27
4.12	Submit a screening using the Android app . . . . .	27
4.13	Unit tests . . . . .	29



# Glossary

<b>AI</b>	Artificial intelligence
<b>API</b>	Application programming interface
<b>CAD</b>	Computer aided diagnostics
<b>CD</b>	Continuous development
<b>CET</b>	Central European time
<b>CI</b>	Continuous integration
<b>CRUD</b>	Create, read, update and delete
<b>CSS</b>	Cascading style sheets
<b>DAO</b>	Data access object
<b>DOM</b>	Document object model
<b>FTP</b>	File transfer protocol
<b>GIS</b>	Geographic information system
<b>GPS</b>	Global positioning system
<b>HTML</b>	Hypertext markup language
<b>HTTP</b>	Hypertext transfer protocol
<b>JDK</b>	Java development kit
<b>JSON</b>	JavaScript object notation
<b>JWT</b>	JSON web token
<b>ML</b>	Machine learning
<b>MVP</b>	Minimum viable product
<b>NGO</b>	Non-governmental organisation
<b>NPM</b>	Node package manager
<b>OEM</b>	Original equipment manufacturer
<b>ORM</b>	Object-relational mapping
<b>PNG</b>	Portable network graphics
<b>PWA</b>	Progressive web application
<b>REST</b>	Representational state transfer
<b>SDK</b>	Software development kit
<b>TB</b>	Tuberculosis
<b>UI</b>	User interface



# 1. Introduction

The proposed challenge by EPCON was to develop on top of an already existing system that uses AI technology to help doctors and other healthcare workers screen patients in risk of contracting TB. This project from EPCON was focused on computer-aided diagnosis, specifically for tuberculosis, and for under-developed world regions.

## 1.1 Organization description

EPCON is a small startup company based in Antwerp, Belgium that since 2018 focuses on identifying, monitoring and evaluating epidemic outbreak regions and finding those in need of help through technology and innovation.

Through their partner network, that consists of a vast list of international companies, NGOs and local communities that have proven track records and unique expertise in the field of GIS, AI and healthcare, they consolidate expertise in terms of health data and commit to further innovate and develop technologies to help epidemic control and fight against TB.

By doing this, they aggregate real-time data and combine it with contextual geographical information like population density, disease co-infection indexes, migration ratios and others. And then, using their distributed Bayesian reasoning models, they analyse cause-effect patterns and use neural networks to screen chest x-rays at large scale for computer-aided diagnostics. [1]

## 1.2 Problem description

One of the already built technologies of EPCON consists of a small web application that connects to their internal API allowing users to upload a chest x-ray photo and receiving a computer generated estimation on how likely is the x-ray to indicate that the patient could have tuberculosis.

After some testing with real doctors in developing countries, EPCON asked the users for some feedback about the usability of the application and they replied with some issues that concerned their use of it.

One of the biggest complaints was referent to the upload process of the photos of x-rays where the process was taking too long and when the connection was unstable it would fail the request and stop uploading.

The objective was to build a wrapper system around EPCON's machine learning API to provide users with a better experience, performance and increased features. So, the main goals of this project were:

{itemize

Build an RESTful API wrapper around EPCON's API with user authentication and persistent data

Build a web client with focus on user experience and performance

Build a mobile client with focus on user experience and performance

### **1.3 Approach**

We were 11 students from 7 different countries working together remotely for approximately 3 months. We have a first initial meeting during one week in Belgium where we had the chance to meet each other, to get to know the people behind the company and to be elucidated about more specific details on the project we were about to develop.

### **1.4 Contributions**

The main beneficiaries from the development of this project would be the local communities in developing countries where the tuberculosis incidence is still very high and where EPCON mostly focuses their efforts. EPCON uses their technologies and expertise in function of health and commit themselves to further innovate and develop technologies in function of epidemic control and the fight against tuberculosis.

In order to do so, they partner up with some third parties like RiskScape, in Johannesburg South Africa, which combines spatial analysis and risk modelling to identify unique locations and make them addressable [2], CognitiveSystems, located in Cape Town, also in South Africa, where they work with biologically inspired computing [3] and Imec, in Belgium, where people work to bring advanced diagnostics and precision medicine [4]. Imec was also the startup company which provided the space and material for our first meeting in Belgium.



EPCON also partners up with some NGOs like the Stop TB Partnership where under their guidance they follow international standards on quality, best practices and sustainability and partner with the public and private sector to ensure interoperability and a patient centric approach in the fight against tuberculosis [5].

## **1.5 Work planning**

We decided to have a weekly meeting, every Thursday at 20:00 CET using an online video-conference platform provided to us by the company. Where we discussed all the topics and issues we had during the previous week and all the tasks and user stories we were about to develop on the next week. Vincent, the CEO of EPCON, would join this meeting every two weeks.

In order to communicate with each other we used Slack as it had a free-tier option and it allowed us to communicate, send files and make voice calls.

To manage our workflow and write down all the user stories and their respective status we decided to use Trello, a web-based Kanban-style list-making application. [6] We first thought about using Jira, an agile project management software developed by Atlassian, but, as we had some team members who were not very experienced with scrum-based workflow and software development in general, we decided to go with Trello as it seemed a much more simpler, user-friendly option which could fulfill all our organizational needs.

In order to build software in a continuous development fashion we need a version control system where we could track version changes of our code repository. Nowadays, it is almost industry standard to use Git as a version control software, but there are other options like Mercurial or SVN, so we decided to use Git. As a Git platform we used GitLab, as it offered free private repositories and free CI/CD tools.

## **1.6 Technologies used**

As described in the previous sections, to the development of this project it was necessary to use different frameworks, technologies and tools related to team work and software development. Here is a detailed list of each of them:

### **Programming languages:**

{itemize

Node.js

JavaScript

HTML

CSS

Kotlin

Java

Bash

**Frameworks:**

{itemize

React

Express

Bootstrap

**Software:**

{itemize

Visual Studio

Postman

Visual Paradigm

**Tools:**

{itemize

Trello

Google Drive

Slack

Zoom

These were the technologies that helped the development of the project in a sustainable way. It is important to note that all technologies, frameworks and tools were defined at the first meeting in Belgium.

## 2. State of the art

Computer aided detection (CAD) is a clinically proven technology that increases the detection of clinical signs and diseases by assisting the doctors in decreasing observational oversights. The more recent clinical introduction of CAD to assist doctors in the detection of pulmonary signs and diseases will likely be followed by the development, clinical trials validation, regulatory approval and commercialization of a variety of CAD applications in diagnostic imaging. [7]

These kinds of software are being used in all fields of medicine but because of the nature of the data itself it is more used in areas where the clinical diagnosis is mostly made by using visual tools, like radiology for example. The computer analyses the image and symptomatic data and outputs a likelihood percentage to aid the doctor in his/her diagnosis.

Nonetheless, making a diagnosis based on images of a patient is often not an easy task. The idea of using computers to help, and possibly improve the interpretation of medical images is therefore very appealing. The first reference to the term computer-aided diagnosis that I have found is in a paper from 1963 by G.S. Lodwick, in *Investigative Radiology* 1(1):72-80 entitled "Computer-aided diagnosis in radiology. A research plan." [8]

### 2.1 Similar solutions

There are many companies investing in this field of aided diagnostics, but when analyzing it just for the ones who are focusing on tuberculosis, we can say that EPCON has two main competitors in the market:

- **Qure.ai** from the United States, which has a product named **qXR** that detects abnormal chest X-rays, then identifies and localizes 15 common abnormalities and screens for tuberculosis. It is used in public health screening programs and was trained with over a million curated X-rays and radiology reports, making it hardware-agnostic and robust to variations in X-ray quality. [9]
- **Delft Imaging** from the Netherlands, that developed **CAD4TB** in order to help

(non-expert) readers detect tuberculosis more accurately and cost-effectively through the speed of digital X-rays combined with deep learning and remote expertise. [10]

All these technologies have a big focus on computer aided-diagnosis, however, we were not developing the CAD AI system ourselves, but rather using the one already built and tested provided by EPCON. Our job was to build an application wrappers around it to make it user friendly and usable for the web and for mobile.

## 2.2 Web application development context

The way we develop web application has come along way since its beginnings.

During the **early 90s** was the era of the static HTML pages, where web pages were text documents and only later it was possible to add styles, images, audio and video files.

Around **1995** the JavaScript language was presented, which made the web pages faster and added the possibility of having dynamic client content and elements on the web pages.

In the year **1996** Flash was introduced by Macromedia, which allowed to enrich the web pages with interactive animations using a programming language called ActionScript. This allowed for an explosion of interactive web video games on the world wide web.

Around **2006** there started to be a shift from static to dynamic web applications with the introduction of jQuery, which made DOM manipulation way easier and browser consistent and with the introduction of Ajax technology, which enabled the client (in this case the browser) to make asynchronous requests to REST APIs. During this time it was also introduced the the notion of responsive web design, where developers would build applications that would scale depending on the window width, making it available for smaller screens on mobile and larger screens on desktop, writing the application just once, instead of having two separate versions for mobile and desktop users.

During **2011** there was a big leap forward for web development because of the introduction of the HTML5 spec. It improved widely the standards, supporting most types of multimedia to be present on the web and allowing to create web applications that are

independent from browsers and platforms. The introduction of HTML5 marked the beginning of the decline in the use of Flash for interactive content on web pages. Around this time some web application frameworks started to become popular and widely used, most notably Backbone and Ember.js.

From around **2016** frameworks like Angular, React and Vue started to emerge and the concept of progressive web application was born. Using the latest browser APIs, like notification, geo-location, and service workers, web applications could behave almost identical to native apps.

Nowadays the development of client-side web applications is dominated by **React** and **Vue**, followed by **Angular**. Concepts like server-side rendering, static HTML generation and posterior hydration with JavaScript and offline usage using service workers have been widely adopted and standardized.

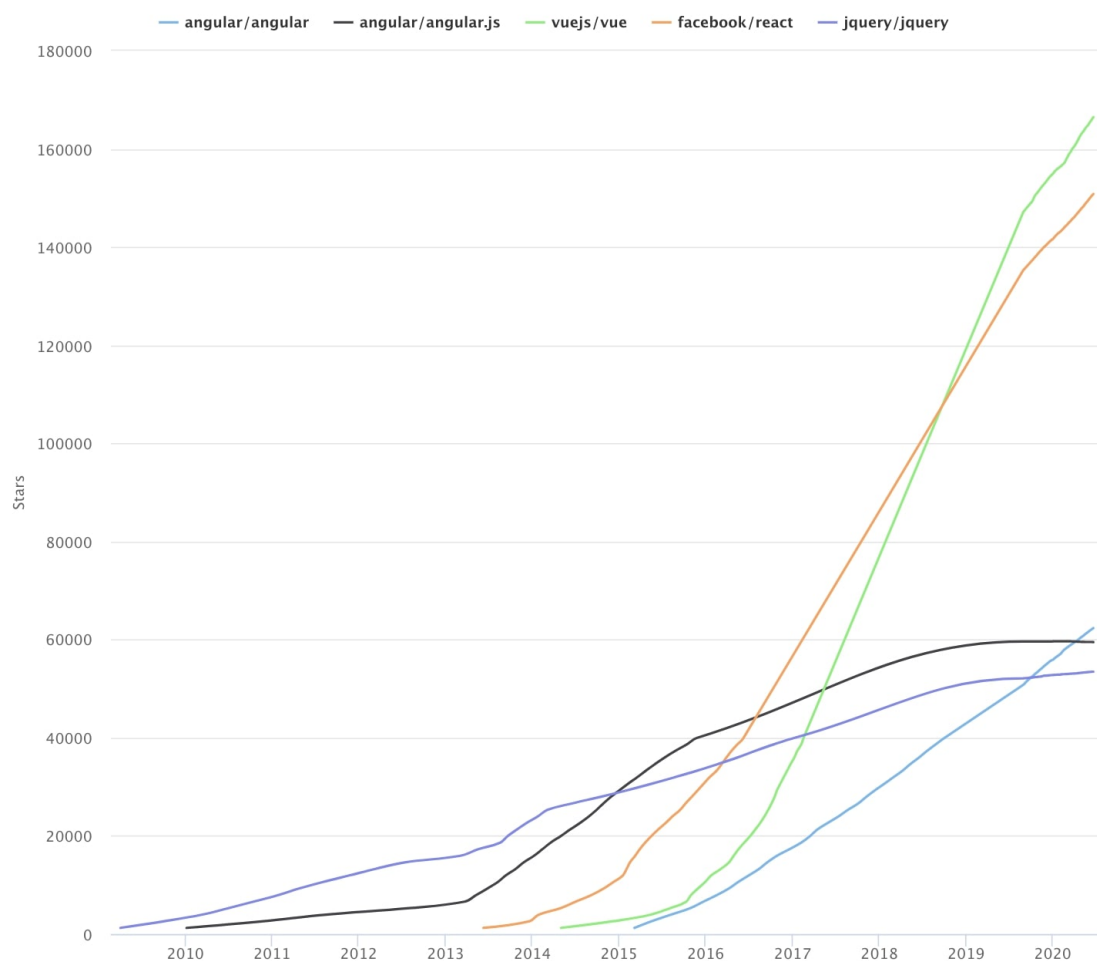


Figure 2.1: Web application framework popularity by GitHub stars

## 2.3 Mobile application development context

The state of the development of mobile applications has been evolving since its conception and it still didn't grew to a completely mature point as progresses are being made still every single day.

In the context of mobile applications we can divide them into two distinct groups: **native applications** and **hybrid applications**.

A **native application** is a software which has been developed to perform some specific task on particular environment or platform, built using a SDK for a certain software framework, hardware platform or operating system. For example, Android applications are build using the Java development kit and Kotlin or Java, iOS applications using the iOS SDK, Swift and Objective-C and Windows applications using the .NET frameworks and the C language.

Operating System	Frameworks and Languages
Android	JDK, Kotlin, Java
iOS	iOS SDK, Swift, Objective-C
Windows	.NET, C

An **hybrid application** has some similarities to the native apps but also some differences. It can be downloaded from the platforms app store just like native apps do, it can get access to most of the native platform features and it's performance can become close to the native app. The major difference is that with an hybrid app the developers can write code just once and publish it in all platforms. The are advantages and disadvantages in deciding to build an hybrid app instead of a native app.

### Advantages of hybrid apps:

- Typically faster to develop
- Unified development of having a single code base
- Less expensive to build and maintain
- Ability to target a wider user pool without much effort

### Disadvantages of hybrid apps:

- Hybrid UI feels less responsive to users
- Indirect access to device hardware and software APIs, usually accessed through framework wrappers
- Worse performance

As of today, there are three popular frameworks for creating hybrid mobile applications: **Ionic**, **React Native** and **Flutter**.

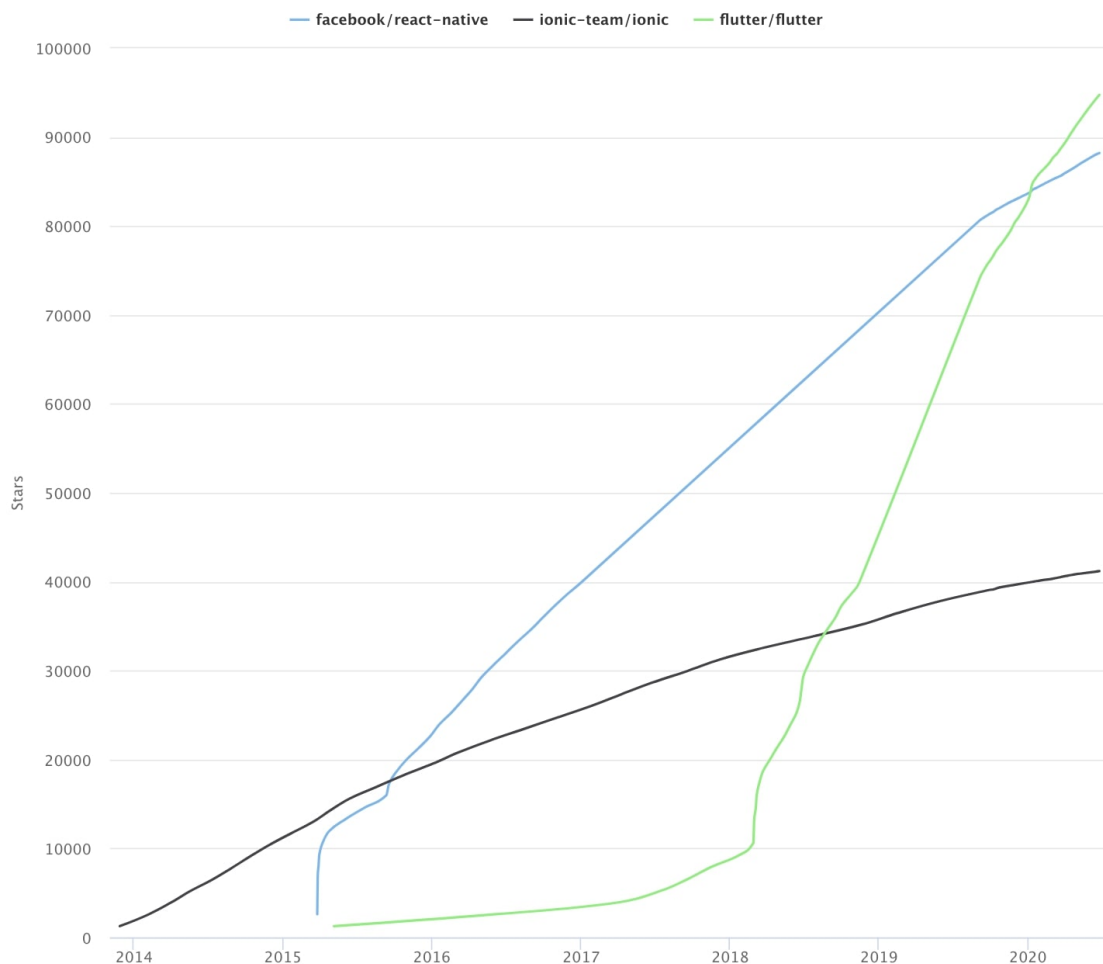


Figure 2.2: Mobile hybrid framework popularity by GitHub stars

**Ionic** is a framework that allows developing applications for iOS and Android. It embeds the application inside a browser without the navigation options (or WebView) that can be run as a standalone application on any platform. [11]

**React Native** uses a very different approach to browser-based described above. Instead of rendering content in a `WebView` or the mobile web browser, the content is rendered with native OEM components provided by the platform. JavaScript is still used for the application logic to be able to re-use the same logic across platforms, and the rendering, which is the most performance-sensitive part, is done with native components for increased performance. [12]

**Flutter** is the latest and more trending nowadays. It was developed by Google and uses a programming language named Dart, which is also maintained by Google. It provides very good and fast tools to work, build and run the code and it maintains solid consistency between all platforms. [13]



### 3. Solution analysis and design

EPCON is a startup company that works in multiple fields of digital social or medical software like epidemics and NGO support. This project from EPCON was focused on computer-aided diagnosis, specifically for tuberculosis, and for under-developed world regions.

#### 3.1 Problem domain

We were told about the problem and all the details when we first met with EPCON in Belgium. During this meeting they showed us their API and a demo web application they had built that connected to this API. It allowed users to upload a pulmonary X-ray image and to get an output image with the areas where signs were found and a likelihood of tuberculosis. [14]

See the screenshot of the app below:

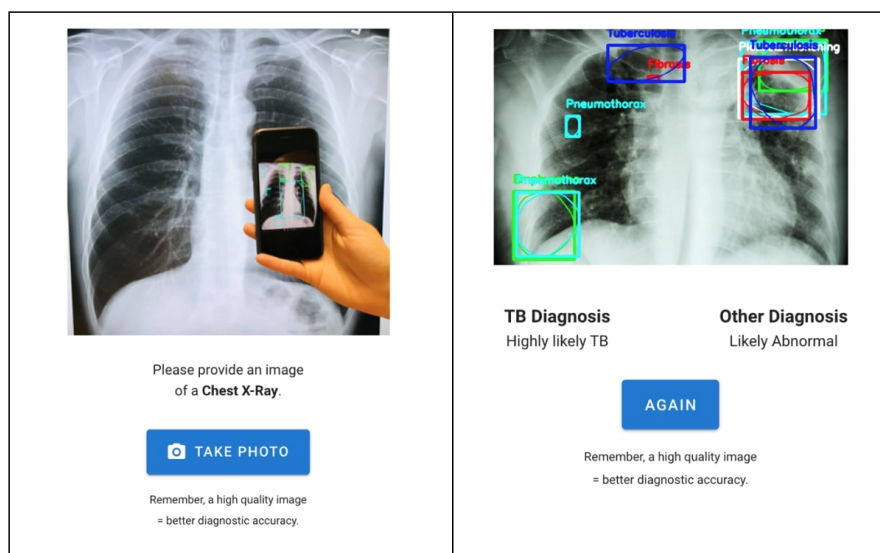


Figure 3.1: EPCON demo web application

This demo application had three main problems:

- It connected directly with the artificial intelligence API, which made all the requests very slow

- It didn't use the latest symptoms payload they have added to the API
- It didn't persist data

So, our project was to build a system that would consume this machine learning API they have created that addressed all the listed problems and more.

EPCON wanted the app to be free to use for anyone, but they also wanted to have a registration system for doctors who wanted to persist all their screenings and patients. The app should allow doctors to manage patients and screen their respective X-rays.

## 3.2 Functional requirements

One of our first approaches during the first meeting was to try to define user-stories and write all of them down.

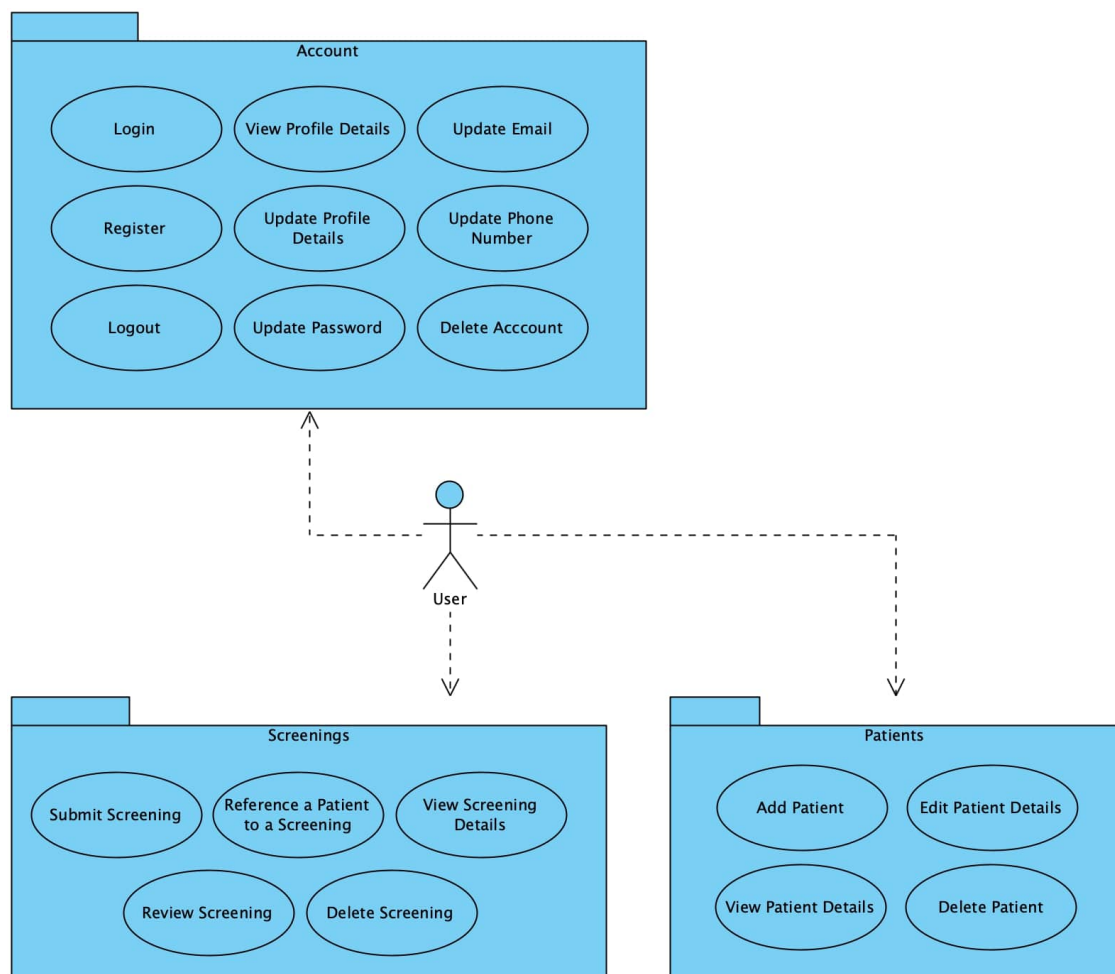


Figure 3.2: Use case diagram

Here are the use cases we collected:

- A user can register using email, password and phone number
- A user can login using email and password
- A logged user can logout
- A logged user can view his/her profile
- A logged user can edit his/her profile (name, city, country and bio)
- A logged user can edit his/her password
- A logged user can edit his/her phone number
- A logged user can edit his/her email address
- A logged user can delete his/her account
- A user can submit a screening for diagnosis
- A user can view the diagnosis of a specific screening
- A user can view the list of his/her screenings
- A user can review a diagnosed screening
- A user can delete a specific screening
- A logged user can add a patient
- A logged user can view the list of his/her patients
- A logged user can view a specific patient details
- A logged user can edit a patient details(name, sex and year of birth)
- A logged user can delete a specific patient
- A logged user can reference a patient when submitting a screening
- A user can read the version changelog
- A user can read the about statement
- A user can read the terms and conditions
- A user can read the privacy policy
- A user can read the cookie policy

### 3.3 Domain modeling

The first step was to identify the different domains we would be working with. This was a time-consuming process because all of us were completely new to this computer-aided diagnosis area and also because all of us came from different countries and were speaking a non-native language. Nevertheless, we came up with these three different domains:

- User
- Patient
- Screening

A **user** is anyone who is using the app. The app is targeted to be use by doctors, health-care professionals or medical students, but it is free and open to anyone, so we thought it would be better to abstract from the concept of doctor.

A **patient** is a record added and persisted by a user and can be referenced on a screening.

A **screening** is an attempt to diagnose an X-ray pulmonary image.

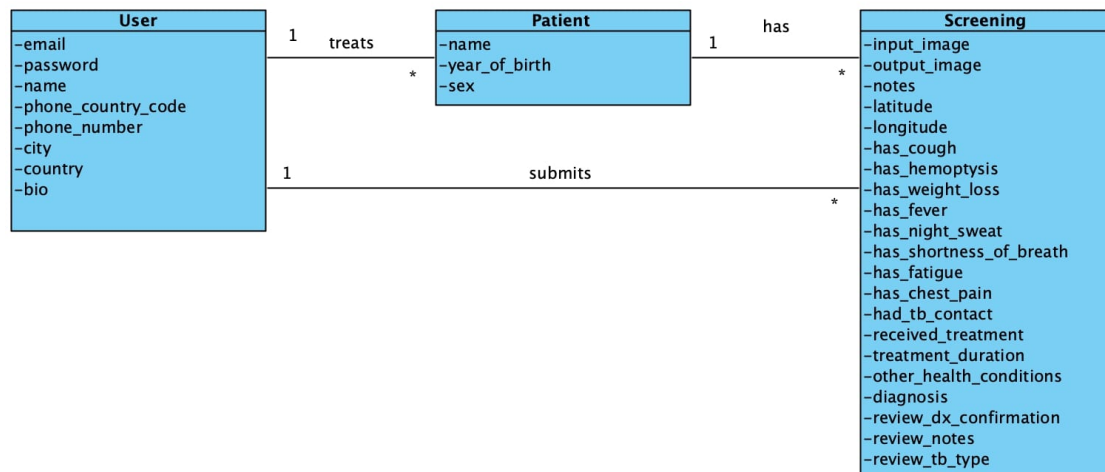


Figure 3.3: Domain model

## 4. Solution development

EPCON had an already built API which provided computer-aided diagnostics. This API accepted an x-ray photo encoded using base 64 and returned a diagnosis estimation. This is a computer-intensive process and it usually took between 10 to 40 seconds to respond. The point was to create a backend RESTful API to hide complexity and manage those heavy requests and then develop clients that would connect to this facade API.

This new API would also deliver features that were not present on the EPCON API, like user authentication, patient and screening data storage.

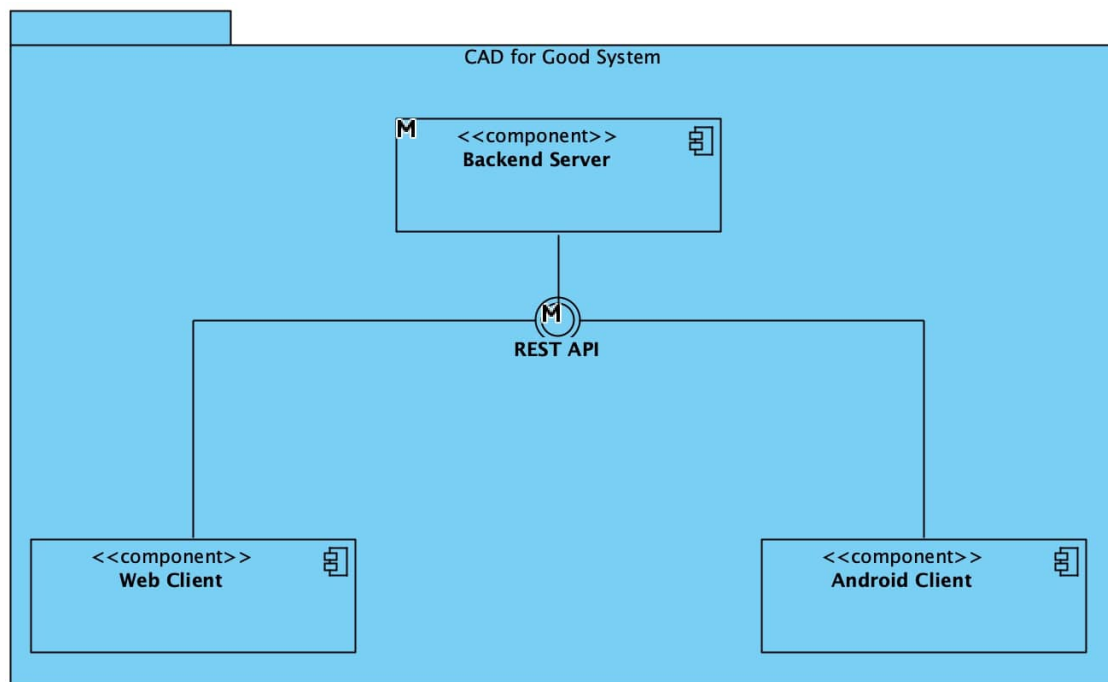


Figure 4.1: System architecture

The objective was to build clients with a focus on user-experience and which felt faster than directly requesting the EPCON servers.

### 4.1 Implementation description

To decide which technologies and frameworks we would use in order to develop the whole system we started by gathering information about the technical skills and previous experiences each of us had. We strongly thought about it and tried to not be influenced by the framework hype that exists nowadays. Many times during software

development people put too much emphasis on the technology rather than the solution. We all agreed that we would go with the technologies that we felt comfortable with and not the ones that seemed better.

After this information gathering we decided to go with Node.js and Express for the back-end API, JavaScript and React for the web client and Kotlin and Java for the Android client.

For designing the database schema we had to make some interesting decisions. One of the use cases required that the diagnose screening feature worked for both registered and unregistered users. For unregistered users the screening feature needed to be somehow connected to the specific device but EPCON also wanted to have the screening data persisted. To achieve this we added a device hash attribute to the Screening on the database schema and on the clients (web and Andorid) we would generate a unique hash and send it to the API on every diagnose screening request. This way we could merge all the local device screening with the ones persisted when an unregistered user finally decided to register.

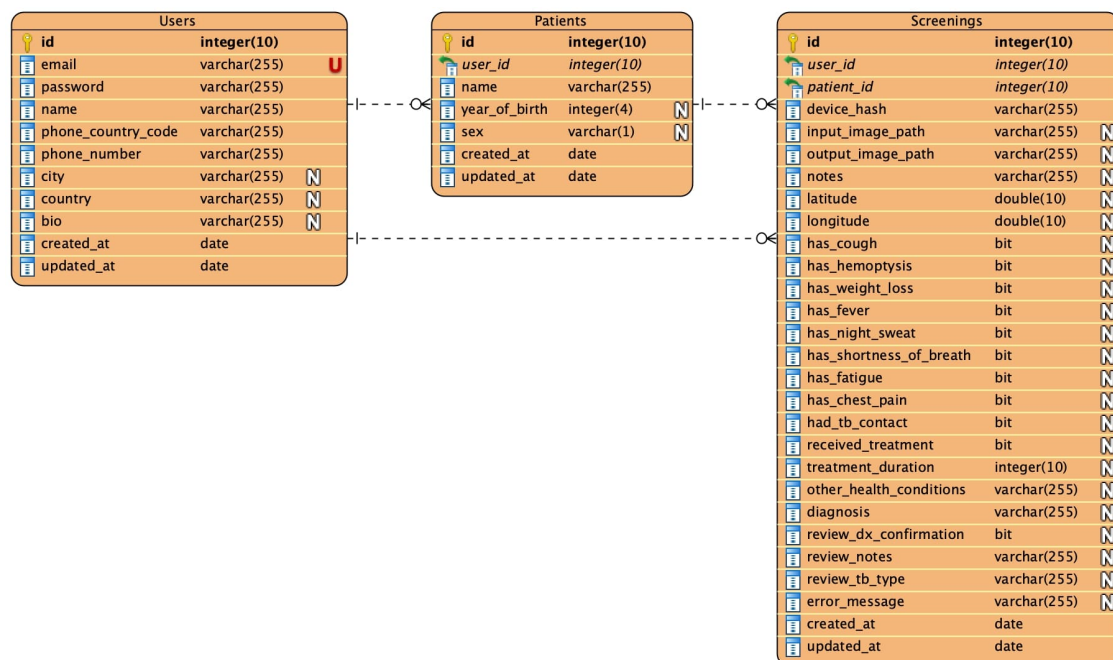


Figure 4.2: Database schema

## 4.2 Backend architecture

Our back-end was built using Node.js, a JavaScript run-time engine for server-side applications. To manage our API routes we used a popular Node.js framework named

Express which made the endpoint development easier for us because of the framework API and nice online documentation.

We structured our code using different components that internally worked together. When a client would request our sever API the request would be caught by our routes components, then it would pass by our middleware functions (different middlewares depending on the endpoint), then it would pass to our controllers and then, depending on the use case, hitting our database using our data access functions or generating some background service throught the use of events.

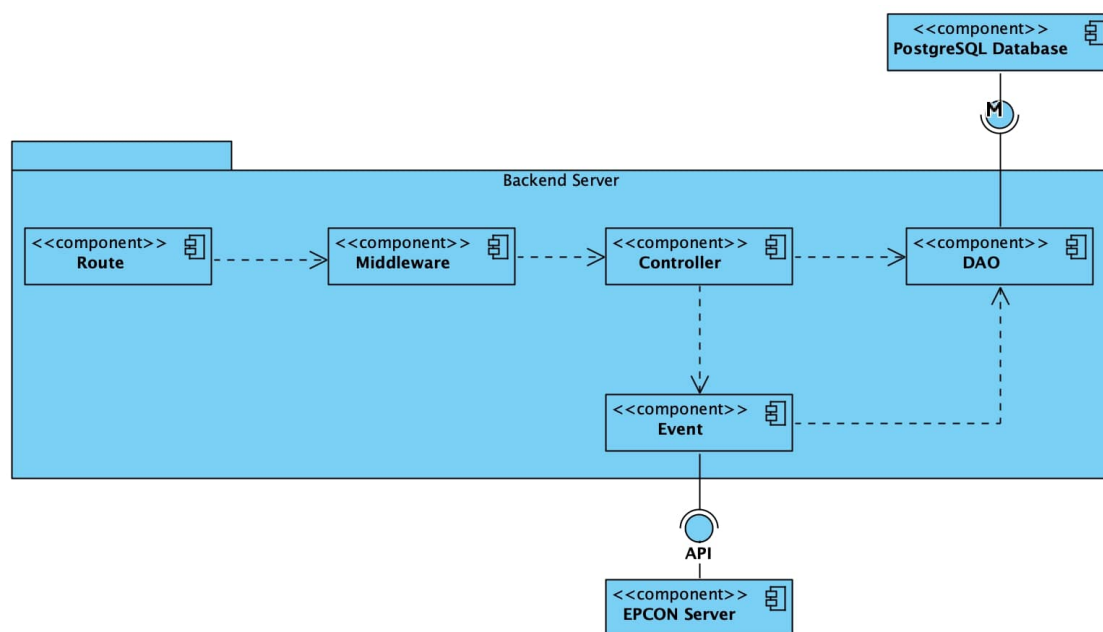


Figure 4.3: Backend server architecture

While designing and building this API, we had to make some interesting decisions. Here I will explain two of them in greater detail.

#### 4.2.1 User authentication

To build our user authentication system with security on our minds we have created 2 different endpoints, one to register using email and password and another to login, also using email and password. When a user registers we check if the email address is already persisted in our database and if not we process to encrypt the password using a gold-standard open source password encryption npm module named bcrypt and then save the user on the database.

When the user logs in we compare the plain-text password to the encrypted one on

the database using a bcrypt function and if it returns true we generate a JSON web token with the user id and return it to the client. Then client can decode the token but cannot modify it. This way, every time the client wants to request a private endpoint, it sends the token as a bearer token in the HTTP headers and the server can then decode the token, get the user id, and check if it was really that logged user who sent the request. This HTTP headers checking is done using our custom middleware functions.

#### **4.2.2 Background services**

One other challenge that we faced was the communication with EPCON API. To get diagnostics data from EPCON machine learning model, we have to first make a request to EPCON's authentication server, with a special username and login they provided for this project, get an authentication token (valid for 2 hours) from it, convert the multi-part-form JPEG or PNG image into base64 and then request an extremely slow-to-respond endpoint from EPCON bio-metrics server. This made the request to our endpoint extremely slow also, so we had to do it differently. When a user would submit a screening, it would save the initial details to the database and start a background service using a Node.js event. Then we would respond back to the client with the initial data stored about the screening, but with the diagnosis data still empty. And then, in the background service we would request the EPCON authentication server, get the token and then convert the images and request the bio-metrics server to get the diagnosis data. After that we would update that specific screening on the database.

After we updated the screening on the database we were thinking about using websockets to inform the user that the screening was updated with the diagnosis data, but, because of our lack of time, we didn't yet implement this feature, so right now, we show the user information that says the screening is still processing and ask the user to refresh the page to fetch new data. We use this short polling technique while we don't have this web-socket feature proper implemented yet.

### **4.3 Web client structure**

Our web application was written using JavaScript and React. We bootstrapped our project using create-react-app which is a command line tool built by Facebook to people set up a modern React web app [15], and Bootstrap which is the world's most popular front-end open source framework to quickly design and customize responsive mobile-first sites, featuring SASS variables and mixins, responsive grid system, extensive pre-



built components, and powerful JavaScript plugins [16].

The overall component schema of the web application can be seen in the image below.

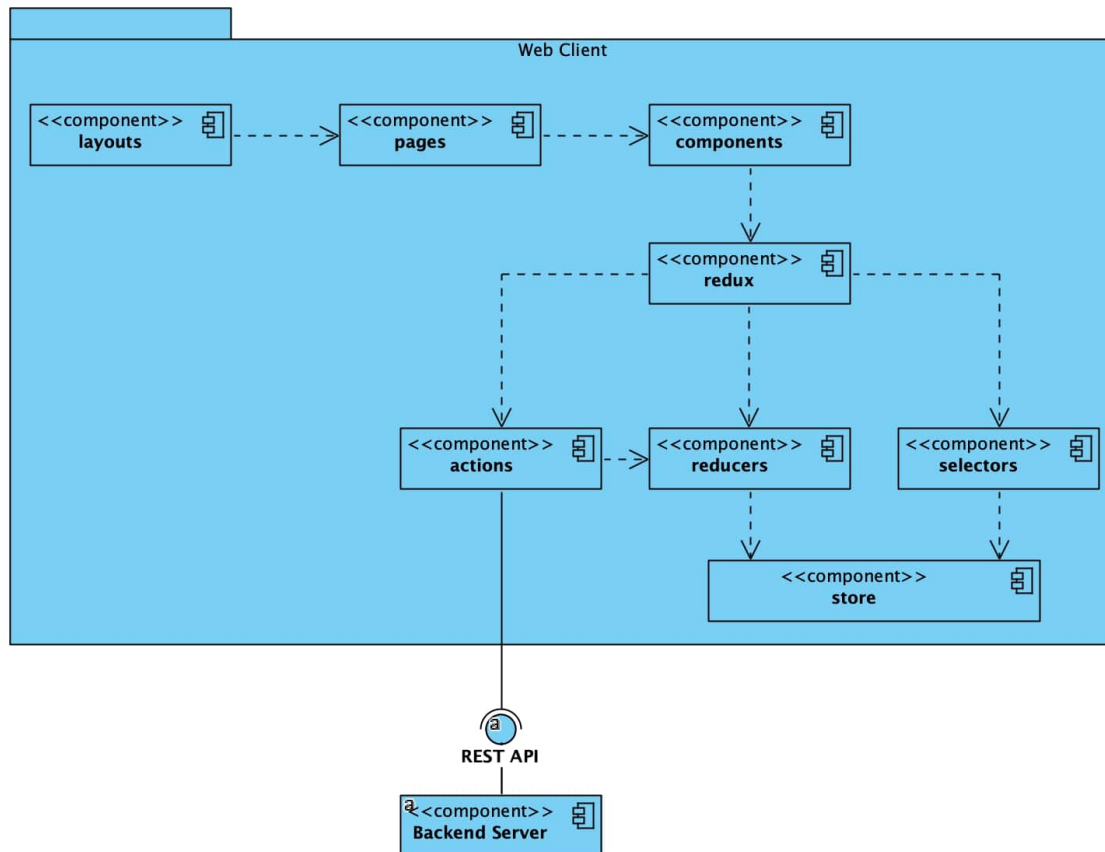


Figure 4.4: Web client structure

The visualization part is composed of pages, composed by a layout and many components.

If a component needs to post some data through the API of our backend server it needs to use our Redux layer. The component uses this layer by calling action functions, which are going to hit the API and call different action types which activate different reducers that will mutate the Redux store state.

If a component needs to fetch some data from the API it also needs to pass by the Redux layer, this time by using selector functions, which will return a specific piece of state to that component.

Below we can see a screenshot of our Redux store state at a given point in time. It holds information about the user profile, loading and error states and a memory cache of all the patients and screenings that were already fetched from the API.

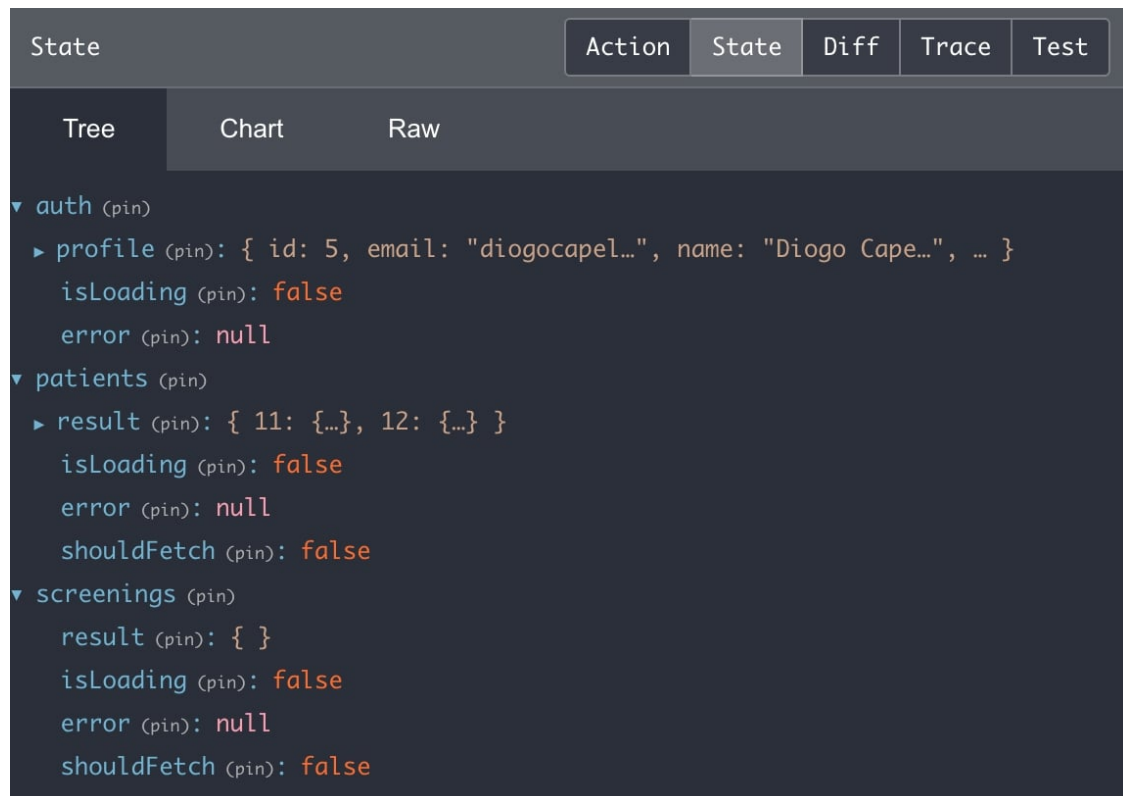


Figure 4.5: Redux store

To style the website to match EPCON design needs we decided to use a SCSS, a CSS tool that give us more options when writing styles for our layouts and components.

In the figure below we can see a screenshot of the homepage of the web application.

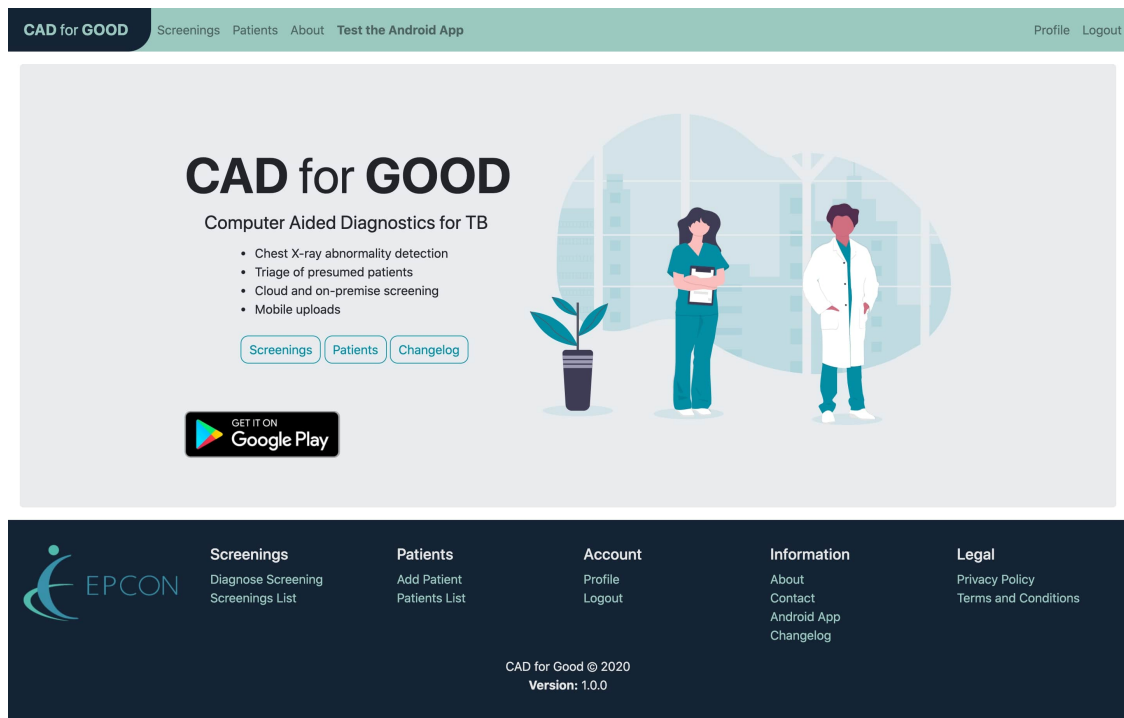


Figure 4.6: Homepage of the web application

The screening process starts by the client generating an unique device hash so every request do diagnose a screening can be traced back to the device.

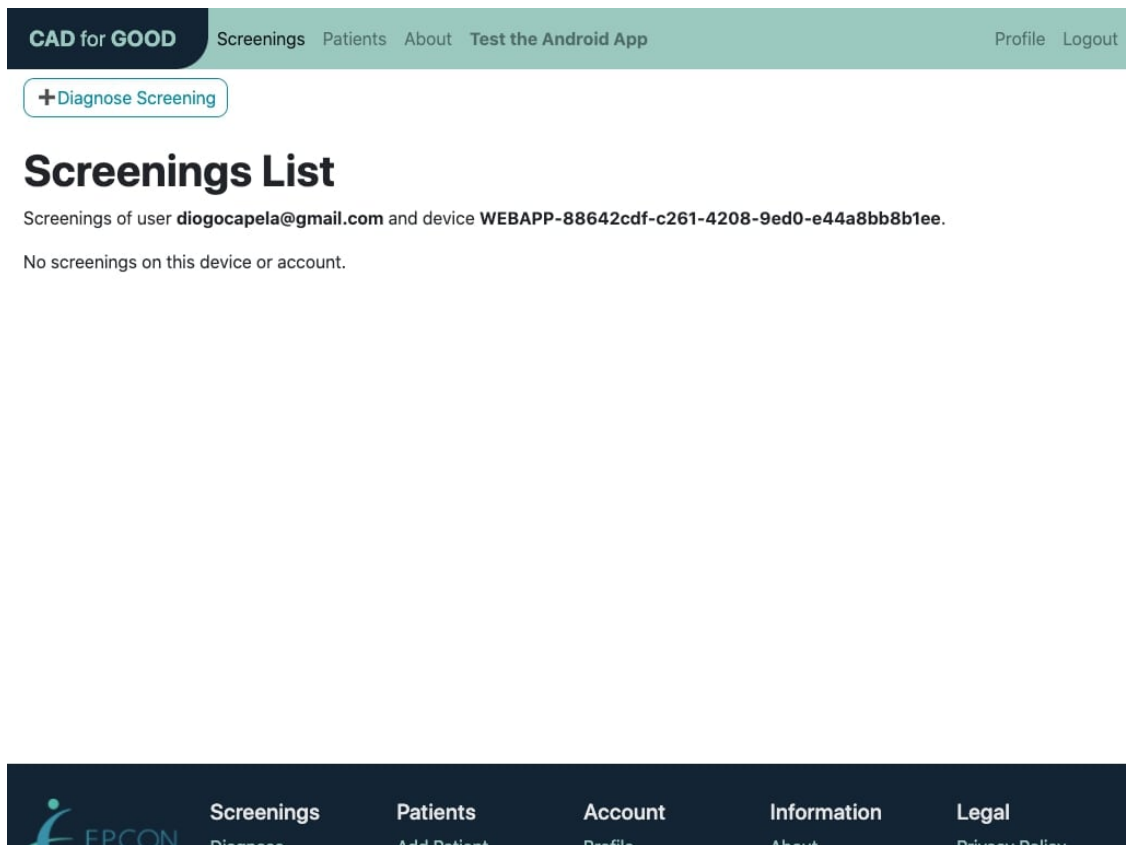



Figure 4.7: Empty screenings list

After we click on diagnose screening we are taken to a page where we can upload the X-ray picture and fill all the symptoms and additional details.

**CAD for GOOD** Screenings Patients About Test the Android App Profile Logout

## Add Screening

1. Upload Chest X-Ray



Choose file xray-00010.jpg

2. Location (optional)

3. Reference a Patient (optional)

4. Symptoms (optional)

5. Clinical History (optional)

Diagnose Screening

Figure 4.8: Submit a screening page

When we submit a screening the server is running it as a background service, so we should wait on the client side for it to be completed. After the screening has diagnostic data from the back-end we can view it in the screening details page you can see below:

## Screening Details

Status: Completed

Reviewed: False

### 1. General Data

ID: 18

Device Hash: WEBAPP-88642cdf-c261-4208-9ed0-e44a8bb8b1ee

User ID: 4

Patient ID:

Latitude: 41.1479853

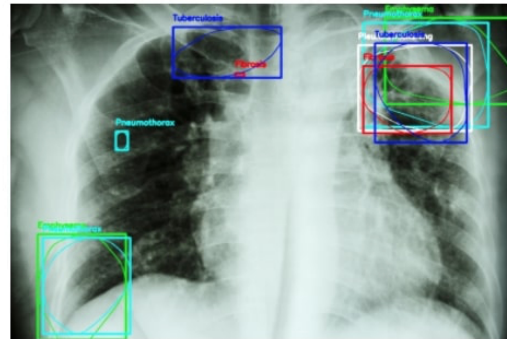
Longitude: -8.6270735

### 2. Chest X-Ray

Input:



Output:



### 3. Symptoms

### 4. Clinical History

### 5. Diagnosis

Abnormal: Likely Abnormal

Tuberculosis: Highly likely TB

- Mass has a occurrence of 21.84%
- Hernia has a occurrence of 15.32%
- Nodule has a occurrence of 20.88%
- Effusion has a occurrence of 1.19%
- Fibrosis has a occurrence of 62.10%
- Emphysema has a occurrence of 86.39%
- Pneumonia has a occurrence of 2.42%
- Silicosis has a occurrence of 0.30%
- Atelectasis has a occurrence of 1.24%
- Cardiomegaly has a occurrence of 2.28%
- Infiltration has a occurrence of 10.77%
- Pneumothorax has a occurrence of 57.74%
- Tuberculosis has a occurrence of 98.48%
- Consolidation has a occurrence of 1.36%
- Pleural\_Thickening has a occurrence of 47.06%

Figure 4.9: Screening details page

## 4.4 Android client structure

To build the Android client we use the standard development tools: Android Studio and the JDK for Android. We decided to build a native application because most of our team was familiar with Java, the programming language used by Android Studio and the Android API.

We divided our components into layouts, composed by activities and fragments, both of them with logic glued to the respective controllers. And then this controllers would connect to our repositories which saved data locally using a SQLite database, or communicated directly with our back-end API.

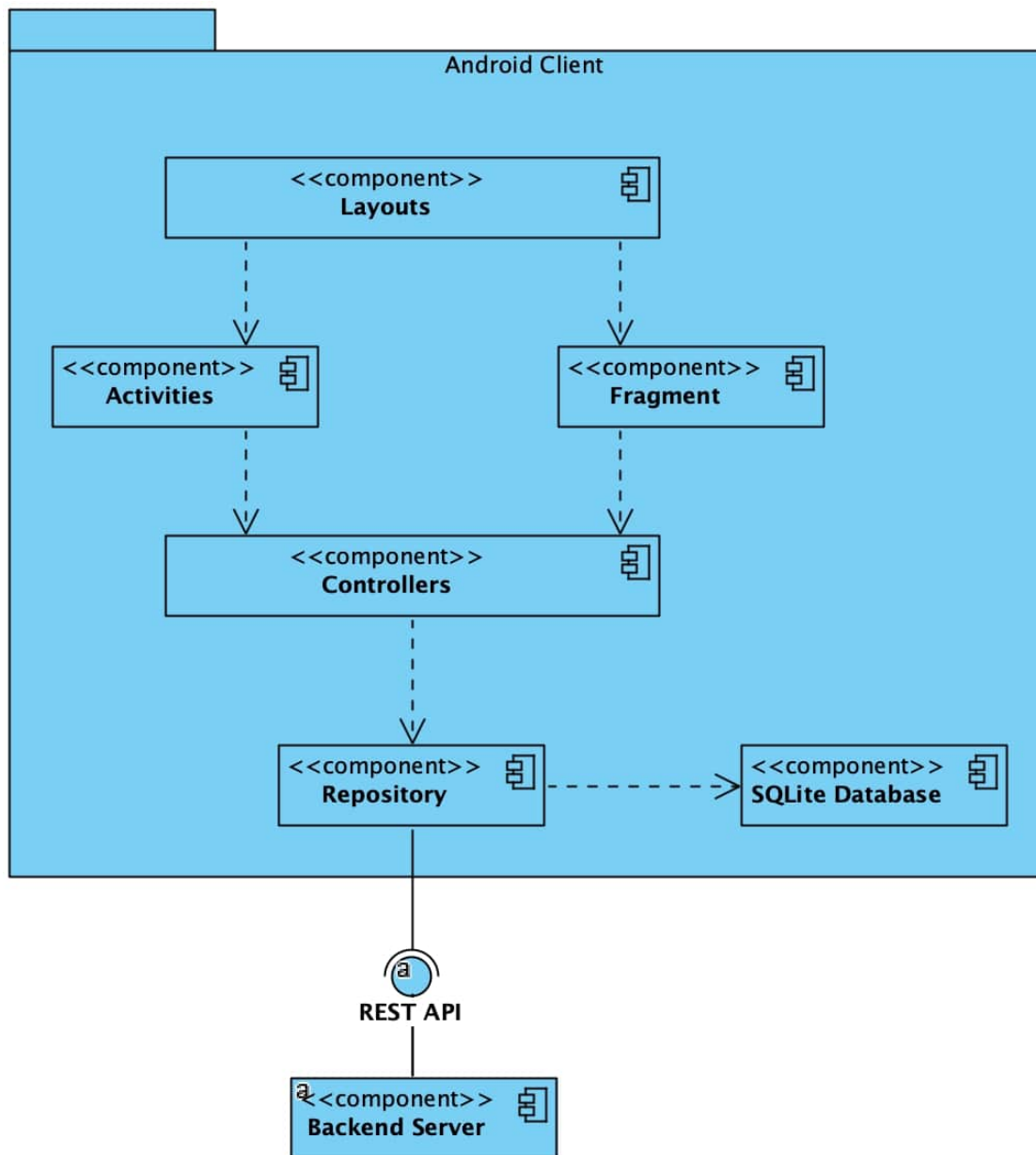


Figure 4.10: Android client structure

Below are some screenshots of the authentication layout we have at the time of this report.



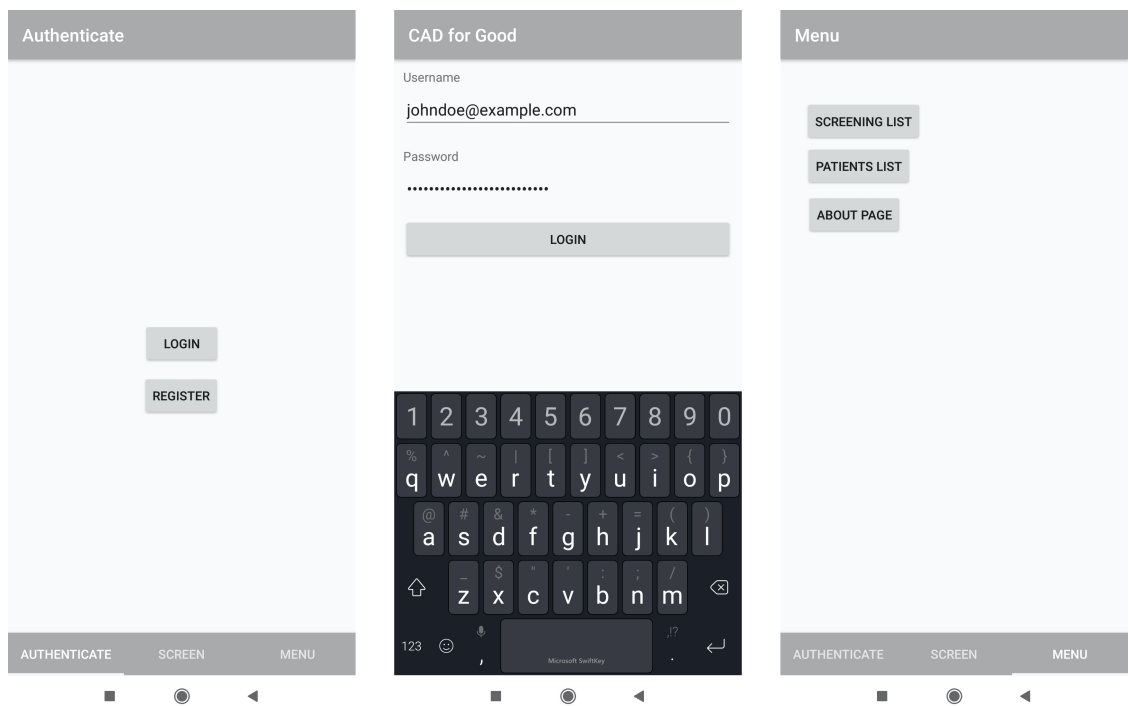


Figure 4.11: Android client authentication

And below you can find the MVP layout for the screening submission process on our mobile app.

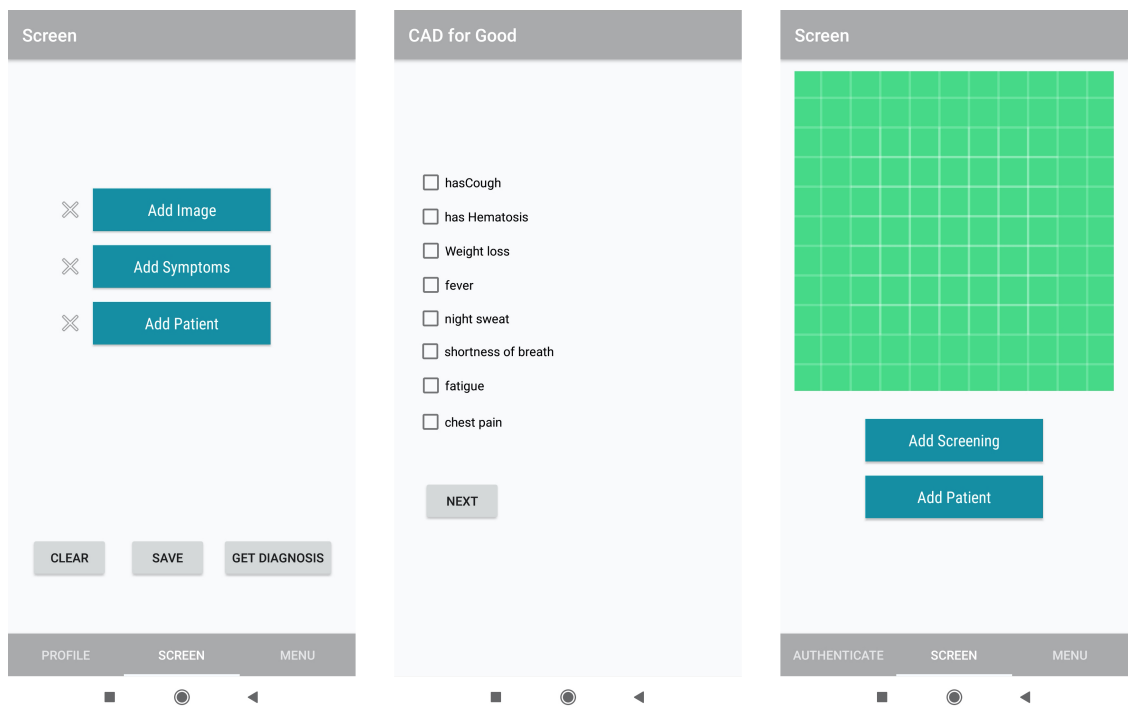


Figure 4.12: Submit a screening using the Android app

## 4.5 Tests

We believe that we can only make sure that our code works if we test it. In this way, we have written unit tests for most of our API endpoints, where we would tests success cases and also error cases.

After some discussion about it, we decided to test only the back-end server and to not do front-end testing in our web and Android clients. We didn't had the needed time to spend writting tests for all of our three components, so we decided to do it for the most important one, which was the back-end API. Below you can view a screenshot of some of the unit tests we wrote:

```
Test all routes
Authentication:
  Register
    ✓ should register successfully (253ms)
    ✓ should not allow register without an invalid email address (6ms)
  Login
    ✓ should login successfully (581ms)
    ✓ should fetch the correct user with the correspondent auth token successfully (341ms)
    ✓ should not login with a wrong password (190ms)
    ✓ should not login a non-existent user (115ms)
Users:
  Update Profile
    ✓ should update a user profile successfully (278ms)
  Update Email
    ✓ should update a user email successfully (226ms)
  Update Password
    ✓ should update a user password successfully (582ms)
  Update Phone Number
    ✓ should update a user phone number successfully (240ms)
Patients:
  Fetch Patients
    ✓ should fetch the current user patients successfully (115ms)
    ✓ should fetch a specific patient of the current user successfully (112ms)
    ✓ should not fetch the current user patients without the auth correspondent token (3ms)
    ✓ should not fetch a specific patient without the auth correspondent token (2ms)
  Add Patient
    ✓ should add a new patient successfully (225ms)
    ✓ should not allow to add a patient without a authorization token (4ms)
    ✓ should not allow to add a patient without name (2ms)
  Update Patient
    ✓ should update a patient successfully (231ms)
    ✓ should now allow to update a patient of another user (112ms)
  Delete Patient
    ✓ should delete a patient successfully (345ms)
    ✓ should not allow to delete a patient of another user (114ms)
Screenings:
  Fetch Screenings
    ✓ should fetch all anonymous screenings of a device successfully (113ms)
    ✓ should fetch all screenings of a user successfully (114ms)
    ✓ should not allow to fetch a screening of another user (116ms)
  Review Screening
    ✓ should review a positive screening successfully (261ms)
    ✓ should review a negative screening successfully (251ms)
    ✓ should not allow to review a screening of another user (111ms)
    ✓ should not allow to review a screening with invalid data (112ms)
  Delete Screening
    ✓ should delete a screening successfully (456ms)
    ✓ should not allow to delete a screening of another user (114ms)

Test Suites: 1 passed, 1 total
Tests:       30 passed, 30 total
Snapshots:   0 total
Time:        8.776s
Ran all test suites.
```

Figure 4.13: Unit tests

## 4.6 Releases and deployment

During our first face-to-face meeting in Belgium we decided that we would release and deploy the system as a whole and not the specific components (web client, mobile client and API). So instead of releasing by component we would release by feature, where we would develop the back-end and the front-end simultaneously.

We didn't agree on a fixed schedule of releases, so we would be releasing and documenting the changes every time we would feel that we had a good amount of issues delivered on that release. During our development time, we released three private alpha versions and a public open-source beta version. Below is our release management change-log:

#### **4.6.1 Version 0.1.0**

We released the alpha version 0.1.0 on 21 March 2020 with the following features:

- Register
- Login
- Logout
- View the user profile
- Diagnose a screening
- View the screenings list
- View the details of a specific screening
- Add a patient
- View the patients list
- View the details of a specific patient
- View the about page
- View the privacy policy page

#### **4.6.2 Version 0.2.0**

Our second deployment, the alpha version 0.2.0 was released on the 8th of April of the same year with the following changes:

- Edit the user profile (name, city, country and bio)
- Edit the user password
- Edit the user phone number
- Edit the user email address

- Reference a patient when diagnosing a screening
- View the output image of a screening
- Edit a patient details (name, sex and year of birth)
- View breadcrumbs for better navigation
- View the terms and conditions page
- View the cookie banner
- Multiple bug fixes

#### **4.6.3 Version 0.3.0**

Our third and last alpha deployment, the alpha version 0.3.0 was released on the 10 June 2020 with the following changes:

- Send location data (latitude and longitude) when diagnosing a screening
- Properly view the diagnosis result of a screening
- Properly validate all input fields
- Refactored the style of the application
- Add contact page and form
- Multiple bug fixes

#### **4.6.4 Version 1.0.0**

Two days after our third release, on 12 June 2020, we released our first beta version 1.0.0, with some small bug fixes and deployed all our code on GitHub public as open source software using the MIT license.



## 5. Conclusion

### 5.1 Objectives achieved

The objectives we set during our first meeting were mainly this three:

- Build an RESTful API wrapper around EPCON's API with user authentication and persistent data
- Build a web client with focus on user experience and performance
- Build a mobile client with focus on user experience and performance

Our project was ambitious, we were 11 students from different countries and communicating in English (which for most of us, including myself, was not our native language), some more experienced than others in terms of working remotely, agile methodology and software development.

We successfully developed the REST API with persistent data and an authentication system and the two clients (web and Android). As of this moment, there are still bugs and issues to work on and the Android app was not yet released on the Google Play Store. Nevertheless, I think we mostly achieved the objectives we have defined at the beginning of the project.

### 5.2 Limitations and future work

During the execution and development of this project we have faced multiple challenges, some more with more impact than others. Here are some issues we have faced:

- Cultural differences between team members
- Working remotely on different time-zones
- Experience gap in terms of agile software development

Most of the difficulties we faced were related to team-work issues. All of us were working remotely, in different countries, some in a very different time-zone which created barriers of communication. Another issue we faced was very different ranges of technical knowledge in software development tools and processes. Some of our team members were

not familiar with popular programming languages like Java or JavaScript, others had not yet been introduced to versioning tools like Git and most were not familiar with the processes that software development takes, like writing user-stories, following a backlog and working following agile methods.

We have recently published all our code with a permissive open source license on GitHub. So, as of today anyone in the world can start contributing for the project. We plan on continuing to develop the system until September, where our main focus will be on finishing the Android app and bug fixing some issues on the server and on the web client.

### **5.3 Final assessment**

Having the opportunity to work on this project was a pleasure for me. It strengthened my knowledge both in a technical perspective and a team-work related view. I've learned a lot about APIs and the Android SDK and working with such a culturally diverse team made me rethink on how much communication is important.

I feel that EPCON was appreciative of our work and recognized our efforts and in the end we effectively build a tool that was asked for us. Overall it was an amazing experience that I think everyone will take some knowledge home.



# Bibliography

- [1] “Epcon.” [Online]. Available: <https://www.epcon.ai>
- [2] “RiskScape | Actuarial and geographical information solutions in the financial and utilities sectors.” [Online]. Available: <http://riskscape.pro/>
- [3] “Cognitive Systems – Bridging The Gap Between People and Things.” [Online]. Available: <http://www.cognitivesystems.ai>
- [4] “Imec R&D, nano electronics and digital technologies.” [Online]. Available: <https://www.imec-int.com/en/home>
- [5] “Stop TB Partnership | Home Page.” [Online]. Available: <http://www.stoptb.org>
- [6] “Trello.” [Online]. Available: <https://trello.com>
- [7] R. A. Castellino, “Computer aided detection (CAD): An overview,” pp. 17–19, 2005.
- [8] G. S. Lodwick, “Computer-aided diagnosis in radiology: A research plan,” *Investigative Radiology*, vol. 1, no. 1, pp. 72–80, 1966.
- [9] “Qure.ai | Artificial Intelligence for Radiology.” [Online]. Available: <http://qure.ai>
- [10] “CAD4TB | Delft Imaging.” [Online]. Available: <https://www.delft.care/cad4tb>
- [11] “ionic-team/ionic: Build amazing Native and Progressive Web Apps with web technologies. One app running on everything .” [Online]. Available: <https://github.com/ionic-team/ionic>
- [12] “facebook/react-native: A framework for building native apps with React.” [Online]. Available: <https://github.com/facebook/react-native>
- [13] “flutter/flutter: Flutter makes it easy and fast to build beautiful apps for mobile and beyond.” [Online]. Available: <https://github.com/flutter/flutter>
- [14] “Biometrics.” [Online]. Available: <https://biometrics.riskscape.pro/beta/epcon>
- [15] “facebook/create-react-app: Set up a modern web app by running one command.” [Online]. Available: <https://github.com/facebook/create-react-app>

- [16] "Bootstrap · The most popular HTML, CSS, and JS library in the world." [Online].  
Available: <https://getbootstrap.com>