



معهد مصر العالي للتجارة و الحاسبات بالمنصورة



Gibson

Medical Solution

Department: Computer Science
Graduation project 2019 | Team: cs_11

Team members

Loay Alaa Sadek

Khaled Abdelmoaty Alam

Eslam Elhosiny

Nurhan Mohamed Nosier

Michael Mealad

Shaimaa Akram Wafaa

Supervised by

Prof. Dr. Hesham Arafat Ali

Eng. Fernando Romany

Content

Chapter(1) : Introduction

- 1.1 Introduction about project
- 1.2 The purpose of the project and its advantages
- 1.3 Problems and solutions

Chapter(2) : Tools

- 2.1 Programs and tools
 - 2.1.1 Languages used
 - 2.1.2 Applications used
 - 2.1.3 Api's and Web services used
- 2.2 Required hardware components
- 2.3 Similar examples
- 2.4 Who is your customer
- 2.5 ECG guide

Chapter(3) : Components and FlowCharts

- 3.1 Components and screens
- 3.2 Flowcharts

Chapter(4) : Functions and References

- 4.1 Project functions
- 4.2 Future work
- 4.3 References

Chapter 1

Introduction

1.1 Introduction about project

Project name: Gibson Medical Solution

Project type: Mobile application

Platforms: Cross platform | Android & Ios

The word “**Gibson**” is a surname of English origin.

Also there are some medical centers in USA called Gibson

Gibson is a mobile application used to provide various medical services and help all of the patients, injureds, doctors and ambulance drivers.

Gibson will help:

Patients:

There is a separate version for patients, patients can search for doctors easily on map, read bio of doctor, find the nearest doctor to him and patients will be able to book an appointment at doctor, watch his/her medicine and medicine history.

Injureds:

Also with patients application, if someone had an accident, he/she can easily use application just few clicks-and fast urgent request will be sent automatically to the nearest available ambulance on the map from the current location of the injured person, using auto send request mechanism in case an ambulance driver canceled the request another new request sent to the second nearest available ambulance to the injured person, if second person canceled then send to the third nearest available ambulance and so on..

Doctors:

There is a separate version for doctors, doctors will be able to add their info(bio, location, experiences, phones, emails...) on the map which will allow patients to find them easily, getting more patients also the application will help doctors to export QR codes for medicine which will be very helpful for both patient and pharmacist. The application also will help doctors view and manage bookings easily.

Ambulance Drivers:

There is a separate version for ambulance drivers, ambulance drivers will receive injureds and patients requests in real time.

The application provides direction service which allow ambulance driver to know the shortest path on map from his/her location and the destination location(injured or patient) so he/she can reach position as fast as possible easily without wasting time.

1.2 The purpose of the project and its advantages

The main purpose of the project is to provide electronic medical services at the highest level that serve all of doctors, patients and emergency drivers using mobile application.

The most prominent features are:

- Processing in the realtime(since we use firebase realtime database and google apis)
- Pick the nearest available ambulance which is faster and easier than call someone for help.
- Send emergency requests automatically to other nearest ambulances cars in case nearest ambulance driver canceled request order.
- Using QR codes for medicine helps patients to get his/her correct medicine easily and that is better than using regular medicine papers(sometimes pharmacist can't read paper correctly)

1.3 Problems and solutions

Patients:

Problem: Patients don't have knowledge about medicine names in medicine papers and sometimes doctor handwriting is not clear enough.

Solutions: Using QR codes refers to medicine info and all related information.

Pharmacist:

Problem: Pharmacist similar to patients sometimes can't read medicine names in medicine papers and because of that some doctors handwriting are not clear.

Solutions: Using QR codes refers to medicine info and all related information, easy to scan, fast to finish patient orders.

Injureds:

Problem: Injureds maybe had a bad accident and not able to call someone and clear his situation and ask someone to help.

Solutions: Use emergency part in the application, just by about 3 clicks he can make automatic urgent emergency requests to all nearest ambulances can and requesting continue until some ambulance accept his request, otherwise(ambulance driver cancel injured request for any reason, any request will be sent to other nearest ambulance.. and so on)

Doctors:

Problem: New doctors and non-popular doctors can't get clients(patients) easily and always they need secretary to manage bookings, and some doctors handwriting is not clear.

Solutions: Doctors will be able to add all their info and bio on the map, able to accept bookings themselves will be able to see all patient info and have very good booking managing control, also export medicine papers as QR code.

Ambulance drivers:

Problem: drivers sometimes can't understand injureds real locations, or take very long time to reach injureds locations.

Solutions: By using the application the driver will get the real current location of the injured person, also will see the best shortest path direction on the map which enable ambulance driver to reach injured location very fast.

Chapter 2

Tools

Section 1: Main work

We can divide our application's main work into main four parts:

- Learning android development basics (took three weeks)

Before developing our application we needed to read about android because knowledge about android developing was not sufficient enough to start developing the project.

- Optical Character Recognition “OCR” (took a month)
 - We started in reading about OCR in general and how it works, so that we can use this technique in our project.
 - Converting the Tesseract OCR engine from C language into java language, to use it because the our developing language for android is java
 - Testing the OCR engine
- Server developing Using PHP language : (took a week)
- Developing the rest of our application services (took 2 months)
- Application final testing and interface (took one week)

2.1 Programs and tools :

Applications languages and API's used

2.1.1 Languages used:

JavaScript: was created by Brendan Eich in 1995 during his time at Netscape Communications. It was inspired by Java, Scheme and Self.

Netscape, for a time, made the best browser in the world and enjoyed market dominance.

In late 1995, when Microsoft cottoned-on to the competitive threat the Web posed, the Internet Explorer project was started in an all-out attempt to wrestle control of the emerging platform from Netscape.

In so doing Microsoft became a mortal threat, compelling Netscape to respond. First, they started a standardization process to prevent Microsoft gaining control of the JavaScript language. Second, they partnered with Sun to leverage their shared interest in breaking the Microsoft monopoly.

Sun began development of Java in 1990 in an attempt to write a language for “smart appliances”. This approach floundered and in 1994, Sun regrouped and set sights on the Web as the delivery platform of choice.

So the Netscape/Sun partnership meant Sun acquired the use of a competitive browser and a delivery system for their strategic technology.

Netscape, on the other hand found a powerful ally against Microsoft. They also aimed to out-manoeuvre Microsoft by being the official browser of the highly anticipated platform that was Java.

Brendan Eich has said that with Sun on board, they decided to surf the tidal wave of hype surrounding Java and position JavaScript as the companion language to Java, in the same way Visual Basic was to C++. So the name was a straightforward marketing ploy to gain acceptance.

Netscape's Mocha (later JavaScript) aimed to turn the web into a full-blown application platform. Furthermore, when used together with their LiveWire application server product, it would enable isomorphic development, with the same language used on both client and server.

If this sounds familiar, it is because this was exactly what Sun was attempting to pull off with Java. At the time however, the Web was very limited when compared to Java; for example, drawing pixels was not possible in JavaScript as it is now with canvas. So Sun (erroneously, I believe) never saw the language as a competitor and the alliance held.

Unfortunately for JavaScript, its early market positioning outlived its usefulness and later became a brake on market acceptance as it emerged as a viable technology in its own right.

So JavaScript was conceived of as a scripting language for the Web for both client and server side. It was then quickly re-positioned as a Web “companion” for Java. Unfortunately, together with the speed of its creation, this meant the inclusion of a number characteristics that would later be ridiculed. Such as:

- automatic semicolon insertion (ASI)
- automatic type coercion when using common operators like lack of block scoping
- lack of classes
- lack of dedicated modularisation capability
- unusual inheritance (prototypical)

As we will see in later posts, many of these criticisms were often expressions of unfamiliarity or syntactic “taste” by programmers more familiar with other languages.

React Native: is quite fascinating: what started as Facebook's internal hackathon project, in the summer of 2013, has since become one of the most popular frameworks. The first public preview was in January of 2015 at Reactjs Con. In March of 2015, Facebook announced at F8 that React Native is open and available on GitHub.

After a little over a year, React Native's growth and adoption rate doesn't show any signs of slowing down. The statistics on Github repository are impressive: 1002 contributors committed 7,971 times in 45 branches with 124 releases, and it's 14th most starred repository on GitHub. Plus, it's constantly updated; React Native is following a two-week train release, where a Release Candidate branch is created every two weeks.

React Native was backed up by two tech behemoths at this year's F8 conference: both Microsoft and Samsung committed to bringing React Native to Windows and Tizen. In the near future, we can expect more Universal Windows Platform and Smart TV apps to be built with React Native.

With the support of both community and tech giants, it's not a surprise that React Native is a trending topic and framework. For the first time in the past 12 months, search terms for React Native surpassed iOS and Android development according to Google Trends.

Sometime in November 2015, we started to think about the future of Shoutem platform. After 5 years on the market, supporting thousands of individuals, SMBs and big enterprises building their applications, we knew we needed to take our application creator platform to another level. We faced two major problems: technology limitations and customer demand.

With our current platform technology based on HTML5, Javascript, Cordova, and other solutions we knew cross-platform could never achieve the level of performance of native applications.

On the other hand, customers wanted greater customization freedom and more powerful applications. If you add the rise of SDKs, frameworks for web developers, high global competition, and automated application maintenance that will drive mobile application development prices down, it was the perfect time for a change.

React Native was an obvious choice for multiple reasons. It solved our current architectural problems while allowing us to achieve a level of performance that is indistinguishable from native applications built with Java or Objective-C.

Another reason is that React Native is a cross-platform solution: we're planning to open our platform to other developers in order to supercharge development and build beautiful native apps for iOS and Android. Furthermore, React Native is super web developer friendly and doesn't require learning native iOS and Android languages, or native APIs –learn once, write anywhere.

Tools used :-

Firestore:-

its first product was the **Firestore** Realtime Database, an API that synchronizes application data across iOS, Android, and Web devices, and stores it on **Firestore's** cloud. The product assists software developers in building real-time, collaborative applications. ... In October 2014, **Firestore** was acquired by **Google**.

The **Firestore** Realtime **Database** is a cloud-hosted **database**. Data is stored as JSON and synchronized in realtime to every connected client. ... Try Cloud Firestore, the latest realtime, scalable NoSQL **database** from **Firestore** and Google Cloud Platform.

Firestore is a fully managed platform for building iOS, Android, and web apps that provides automatic data synchronization, authentication services, messaging, file storage, analytics, and more. Starting with **Firestore** is an efficient way to build or prototype mobile **backend** services.

Expo:-

Expo is a toolchain built around **React Native** to help you quickly start an app. It provides a set of tools that simplify the development and testing of **React Native** app and arms you with the components of users interface and services that are usually available in third-party **native React Native** components.

Running your **React Native** application. Install the **Expo** client app on your iOS or Android phone and connect to the same wireless network as your computer. On Android, use the **Expo** app to scan the QR code from your terminal to **open** your project. On iOS, follow on-screen instructions to get a link.

According to the **expo.io** site, **Expo** is “a free and open source toolchain built around React Native to help you build native iOS and Android projects using JavaScript and React.” **Expo** is becoming an ecosystem of its own, and is made up of 5 interconnected tools: XDE: the **Expo** Development Environment.

2.1.2 Applications used:

- Visual studio code
- adobe xd
- Expo

2.1.3 API's and Web services used :

- **Google cloud Api**
- **Expo.io**
- **Google maps**
- **Firebase**
- **Google directions**

2.2 Required hardware components:

- **Mobile**
- **laptop**

1. HARDWAR

Computer hardware are the physical parts or components of a computer, such as the monitor, keyboard, computer data storage, graphic card, sound card and motherboard.

By contrast, software is instructions that can be stored and ran by hardware.

EG: Processor, Keyboard, Mouse, Printer, Screen, etc..

2. SOFTWARES

Software can be defined as programmed instructions stored in the memory of flash drives of computers for execution by the processor.

EG: Operating system, firmware, Drivers, Third party softwares(Vlc, skype.

Benefits:

- Eased the accessibility of doctors' booking to be only a click away. You access our website or mobile application to choose your doctor by specialty, geographic area, insurance purveyor and fees. With more than 100,000 patients' reviews and ratings available on the platform, users can readily determine the doctor with the best medical service and the least waiting time. Once the doctor is chosen, the patient can either book automatically from the website, the mobile app, or contact the call center.
- provide medical services and help all of patient ,doctor ,injureds and ambulance drivers.
- fast solution to people who making accident.
- Easy to contact the patient with Dr.
- Make booking more easy for patient.
- Pick nearest available ambulance car which is faster and easier than call someone for help.
- is an Egyptian online medical care scheduling service , providing a free of charge medical search platform for end users by integrating information about medical practices and doctors' individual schedules in a central location.
- Send emergency requests automatically to other nearest ambulances cars in case nearest ambulance driver canceled request order.
- Using QR codes for medicine helps patients to get his/her correct medicine easily and that is better than use regular medicine papers(sometimes pharmacist can't read paper correctly).

2.3 Similar examples:

vezeta :-

Business model

Vezeeta provides a scheduling system on a paid subscription basis for medical personnel. The scheduling system can be accessed by subscribers both as an online service and via the deployed office calendar software, or integrated with their websites. The subscriber's schedules are available to the patients.

The end user-searchable database includes specialties, range of services, office locations, photographs, personnel educational background and user-submitted reviews. Users can review doctors' schedules and make appointments for specific time slots.

Vezeeta.com is the Middle East Healthcare industry-leading digital startup that emerged in Cairo, Egypt. We are pioneering the shift to automated physician, clinic and hospital bookings. In only 2 years we have become the go-to marketplace with comprehensive access to healthcare providers of every discipline across a multitude of insurance purveyor and specialist networks using our proprietary digital cloud based solutions.

Vezeeta eased the accessibility of doctors' booking to be only a click away. You access our website or mobile application to choose your doctor by specialty, geographic area, insurance purveyor and fees. With more than 100,000 patients' reviews and ratings available on the platform, users can readily determine the doctor with the best medical service and the least waiting time. Once the doctor is chosen, the patient can either book automatically from the website, the mobile app, or contact the call center.

Vezeeta currently operates in Egypt, Jordan, Lebanon and Dubai with plans to penetrate North Africa and Saudi region by 2018.

Availability

The service was initially launched in 2011 as a way to schedule ambulances.[1] Initially limited to Cairo, it expanded to cover 15% of MENA population across 8 cities, and is used by more than 2,000,000 people per month. The service may be used as Android iOS, or

web application, as well as supporting Arabic and English language versions with the launch of Vezeeta en français soon. The service is now available in 4 MENA countries, Egypt, Jordan, Lebanon and KSA.

As of September 2018, the company had raised over \$22.5 million in venture capital from investors.[2]

References

"Vezeeta, the leading MENA healthcare startup, secures \$12M Series C led by STV". techcrunch.com. 2018-09-19. Retrieved 2019-01-30.

"Vezeeta raises \$12 million Series C led by STV for its healthcare platform". menabytes.com. 2018-09-19. Retrieved 2019-01-30.

Electronic health record

An electronic health record (EHR), or electronic medical record (EMR), is the systematized collection of patient and population electronically-stored health information in a digital format.[1] These records can be shared across different health care settings. Records are shared through network-connected, enterprise-wide information systems or other information networks and exchanges. EHRs may include a range of data, including demographics, medical history, medication and allergies, immunization status, laboratory test results, radiology images, vital signs, personal statistics like age and weight, and billing information.[2]

A decade ago, electronic health records (EHRs) were touted as key to increasing of quality care. Today, providers are using data from patient records to improve quality outcomes through their care management programs. Combining multiple types of clinical data from the system's health records has helped clinicians identify and stratify chronically ill patients. EHR can improve quality care by using the data and analytics to prevent hospitalizations among high-risk patients.

EHR systems are designed to store data accurately and to capture the state of a patient across time. It eliminates the need to track down a patient's previous paper medical records and assists in ensuring data is accurate and legible. It can reduce risk of data replication as there is only one modifiable file, which means the file is more likely up to date, and decreases risk of lost paperwork. Due to the digital information being searchable and in a single file, EMRs are more effective when extracting medical data for

the examination of possible trends and long term changes in a patient. Population-based studies of medical records may also be facilitated by the widespread adoption of EHRs and EMRs.

Advantages:-

- Eased the accessibility of doctors' booking to be only a click away. You access our website or mobile application to choose your doctor by specialty, geographic area, insurance purveyor and fees. With more than 100,000 patients' reviews and ratings available on the platform, users can readily determine the doctor with the best medical service and the least waiting time. Once the doctor is chosen, the patient can either book automatically from the website, the mobile app, or contact the call center.
- is an Egyptian online medical care scheduling service , providing a free of charge medical search platform for end users by integrating information about medical practices and doctors' individual schedules in a central location.
- vezeeta.com is the leading digital healthcare booking platform and practice management software in MENA. We are pioneering the shift to automated physician, clinic and hospital bookings making healthcare easily accessible in the region.
- With over 200,000 verified reviews, patients are able to search, compare, and book the best doctors in just 1 minute. Doctors also provide Patients with seamless healthcare experiences through our clinic management software.
- We operate in Egypt, KSA, Levant and UAE. We strive to lead every aspect of the healthcare industry and continue to launch products that have positive impact on people's lives.

Abstract:-

2.4 Who is your customer?

Patients, Healthcare Providers (Doctors in hospitals, clinics and polyclinics), Insurance Purveyors and Pharma Companies

What problem does this idea/product solve or what market need does it serve?

Vezeeta is the leading digital booking platform that is pioneering the shift to automated physician, clinic and hospital bookings. You access our website or mobile application to choose your doctor by specialty, geographic area, insurance provider and fees. With more than 100,000 patients' reviews and ratings available on the platform, users can choose the doctor with the best medical service and the least waiting time. Once the doctor is selected, the patient can either book automatically from the website, the mobile app, or contact the call center. Doctors also benefit from Vezeeta Care – our leading practice management software – that aligns and manages clinics' schedules and medical records in a manner that helps to improve the overall healthcare experience.

What attributes will make this idea/product successful? Why do you believe that those features will create success?

Vezeeta success comes from its unique ability to address real, major problems faced by patients when trying to access doctors in the region. Choosing the right doctor is a tough decision and patients can spend significant time and effort researching in order to get the best medical service. Also, the wide variance between the quality and cost of doctors is another problem. However, the major challenge is the actual booking process – patients can spend up to 4 to 5 hours on the phone just trying to get an appointment. Vezeeta has managed to overcome all these pains by offering verified reviews on our platform for all subscribed doctors highlighting their price points and expected waiting times to ease choice and empower patients.

Explain how you (your team) will execute to make this idea/product successful? What gives you (your team) an advantage over others already in the market or new to this market?

All our solutions are built on Microsoft technologies and we employ the best talent to ensure continual product innovation and market success.

yodawy:-

- Yodawy – The #1 Pharmacy Delivery Application is a user-friendly application enabling you to easily order any medicine online from your nearest pharmacy and get it immediately delivered to your doorstep.
- You can do that by searching through a comprehensive, up-to-date, online medical database or leave it up to the experts to decipher that hand-written prescription by simply uploading images of your paper prescription or even medicine packaging.
- You can also receive medical e-prescriptions from your doctor after visiting their clinic or hospital. As for the cherry on top.
- Yodawy even links up to your insurance provider online! If your insurance provider is part of our partner network we will submit a claim on your behalf every time you order an eligible drug from our online medical store.
- Once your claim is approved, we send your order immediately to the nearest pharmacy for pickup or delivery for a hassle-free ordering experience.
- No more filing those annoying insurance claim papers!

Easy Drugs :-

Easy Drugs is an index for drugs and other medical products existing in the Egyptian market. Easy Drugs is backed with the most trusted, accurate, and up-to-date medicine data source in Egypt, which has been popularly used years ago in Egypt's medical sectors

- Dawaa . Medications data is maintained and updated frequently to always keep the application accurate, and up-to-date.

You can easily search medicines or active ingredients, write only as little as 3 letters and let Easy Drugs remind you with the full medicine or active ingredient name.

Browse drugs by medical categories, choose main category, such as: anemia, then subcategory, such as: iron preparations and browse products in Egypt's medical market under this group.

Browse all medicine companies that has products in the Egyptian market, use the fast alphabetical indexer to fast scroll to the company you want, and browse its products. Browse all active ingredients, see drugs under specific active ingredient.

Detailed view of drug information, including its name, form, company, and - most importantly – price in Egyptian pounds.

Know what other categories a specific drug belongs to, Easy Drugs show you all categories for a specific drug, so you know all its possible uses.

Know drug alternatives through its active ingredients. Easy Drugs support both Arabic and English user interface according to your mobile current language.

Your medical information, enriched, accurate, and up-to-date.

Medscape:-

- Medscape is the leading online global destination for physicians and healthcare professionals worldwide, offering the latest medical news and expert perspectives.
- point-of-care drug and disease information , and relevant professional education and CME/CE. The Medscape application is designed to provide a personalized experience so that you can easily access what you need, when you need it.
- What's NEW? * Discover the latest clinical news and perspectives in your specialty with our tailored newsfeed.
- Receive alerts about FDA approvals, conference news, late-breaking clinical trial data, and more.
- Access Reference articles, News, and CME/CE courses with our modern taskbar.
- Customize your homescreen, save your favorite clinical tools, and bookmark articles you like or want to check out later.
- **Additional Key Features:** Look up the most current prescribing and safety information on 8500+ prescription and OTC drugs, herbals, and supplements.
- Check out other useful resources, including our drug interaction checker, specialty-specific calculators, pill identifier, and more.
- Find essential procedures in your field with 6200+ Reference articles.
- Earn free CME/CE credits and ABIM MOC points on-the-go, and monitor your progress with our built-in Activity Tracker.
- Access the largest network for physicians and medical students with Consult.

2.5 ECG Guide:-

The most comprehensive ECG application in the iPhone App Store - over 200 examples of common and uncommon ECGs iPad version? bit.ly/ECGiPad or search iTunes "ECG Guide iPad" Praise for the ECG Guide:

"The best ECG application currently available."

-iMedicalApps.com "Core Medical App."

- Georgetown Medical School. "This brand-new gem features concise explanations with ample high-quality ECG examples..."

- American College of Emergency Physicians "...a 5 star application"

- User review "...a must have application for doctors, residents and medical students."

- User review Succinct yet authoritative, this ECG reference is a critical companion for practicing physicians, medical trainees, nurses and paramedics. It is designed as a learning tool for those still training as well as a reliable reference for more experienced interpreters of ECGs.

Features:

- Full reference text that includes in-depth information on topics such as: - Approach to ECG Interpretation - Analysis of rate, rhythm, axis, P wave, QRS complex, ST segment, T wave and QT interval

- Ventricular hypertrophy and atrial enlargement - Assessment of ischemia - Approach to arrhythmia

- Bradyarrhythmias and tachyarrhythmias, Heart block, and much more... (content based on national guidelines 2009 AHA/ACCF/HRS Recommendations for the standardization and interpretation of the electrocardiogram)

- The largest ECG library available on the iPhone with over 200 high resolution sample ECGs with common and complex findings for easy reference

- ECG Interpreter - Stepwise assistance with ECG interpretation. Helps create differential diagnosis based on simple to assess ECG features.

- Over 100 multiple-choice quiz questions to test your knowledge - Rapid Reference section
- one tap takes you to 11 summary pages used to confidently interpret ECGs
- Clear descriptions of both common and complex arrhythmias and ECG findings
- Review etiology and differential diagnosis of common and complex arrhythmias and ECG findings
- Pediatric values included Compiled and written by Jason Andrade, MD, FRCPC.

Chapter 3


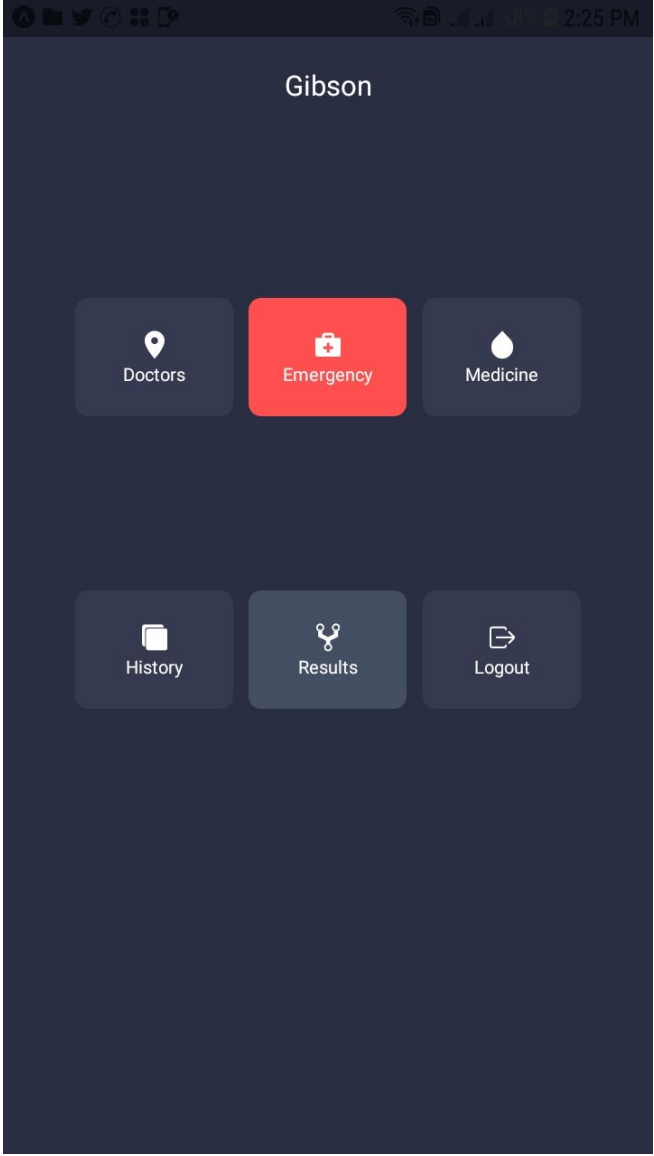
Components, Screens and FlowCharts

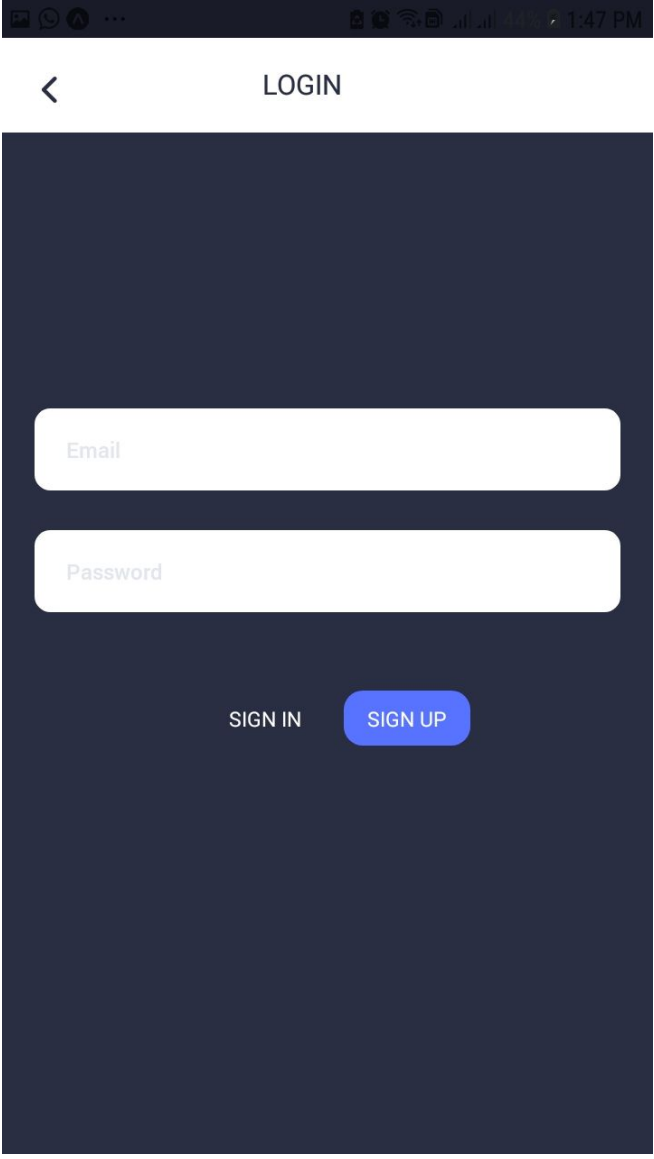
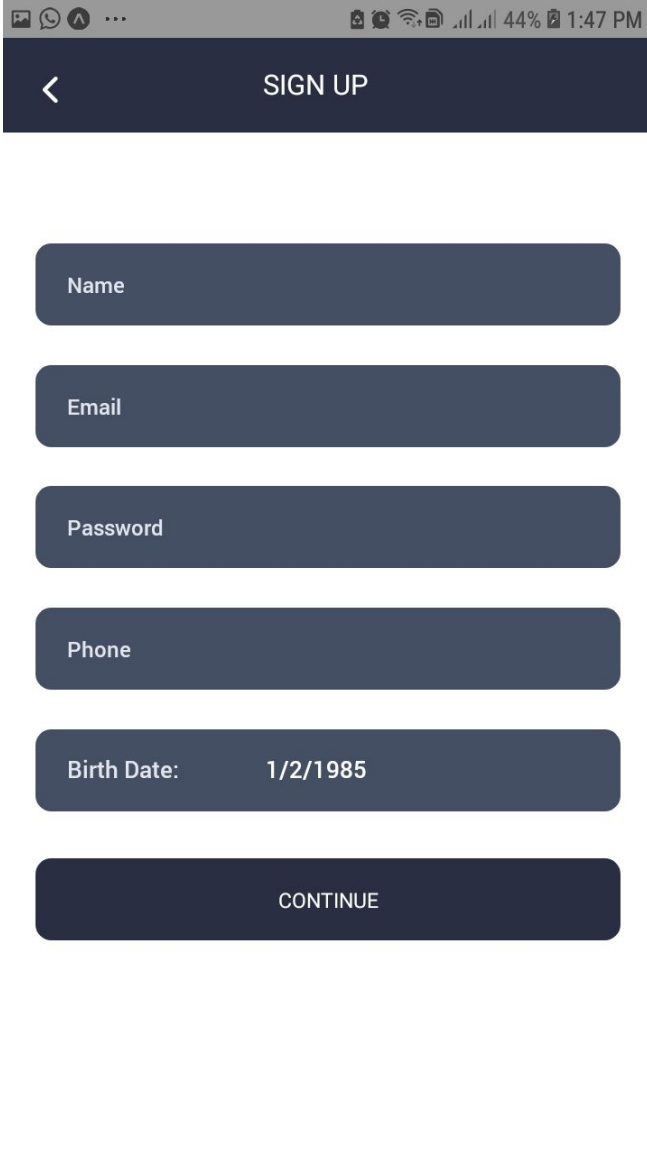
3.1 Components and screens

Patient app

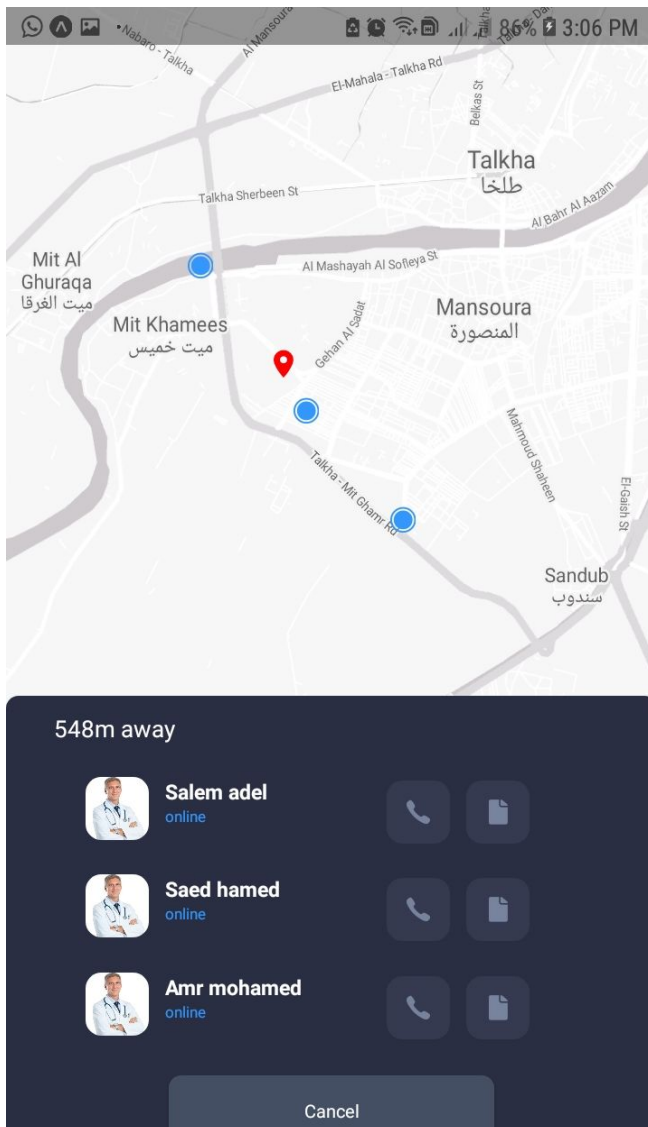
Screens:

- Welcome screen
- Login screen
- Register screen
- Home screen
- Doctors screen
- Emergency screen
- Medicine screen
- History screen
- Test Results screen
- History of doctor screen
- History of diagnose screen

1. Welcome screen	2. Home screen
	
Very first screen appears to patient After click logo redirect to login screen	Main home screen contains 6 icons for 6 screens: doctors, emergency, medicine, results, profile.

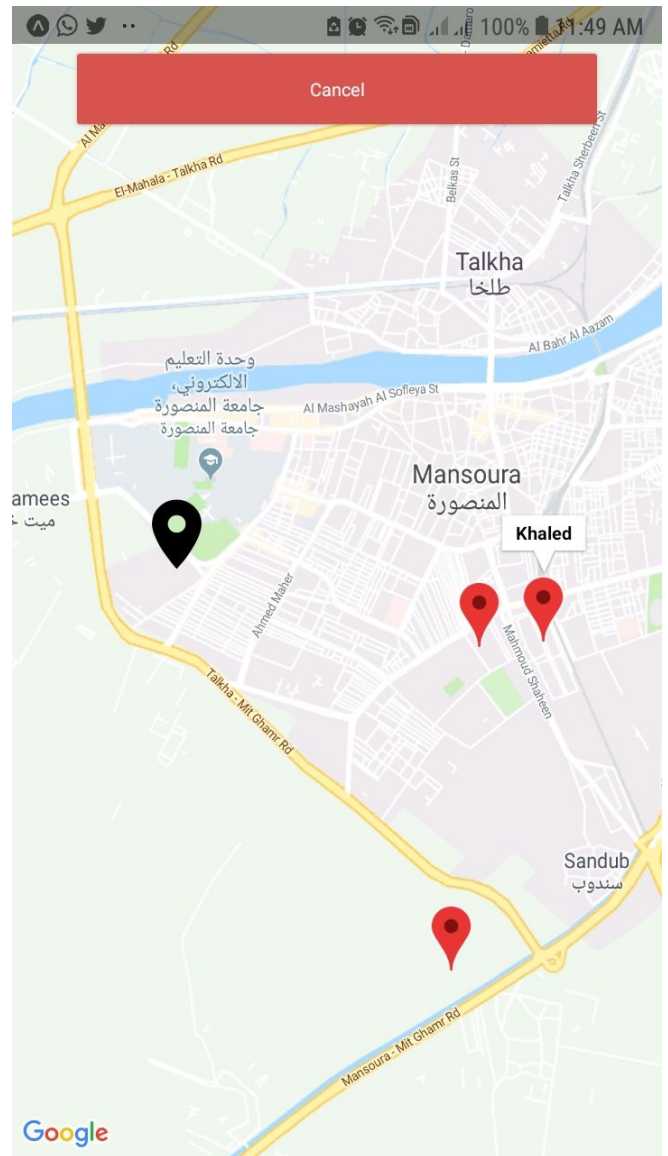
3. Login screen	4. Register screen
	
<p>Allow user to sign in is using his/her email and password</p> <p>After click signin redirect to home screen</p> <p>After click signup redirect to register screen</p>	<p>Allow patient to register on app, asking for patient details: name , email, password, phone, birthdate..</p> <p>After click continue redirect to home page</p>

5. Doctors screen



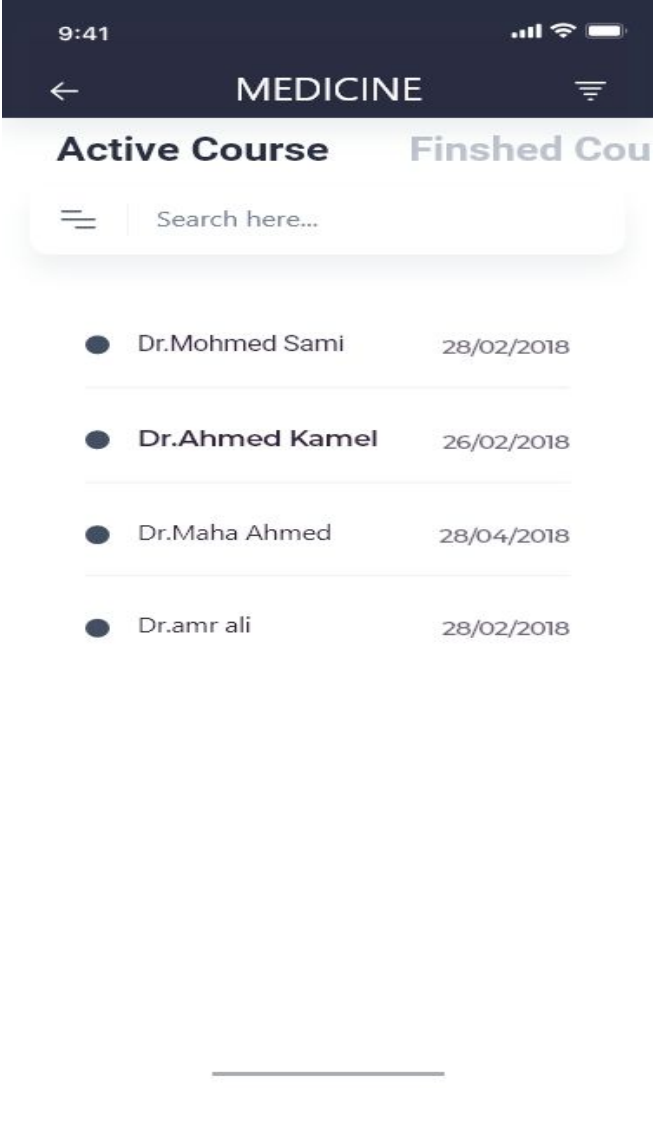
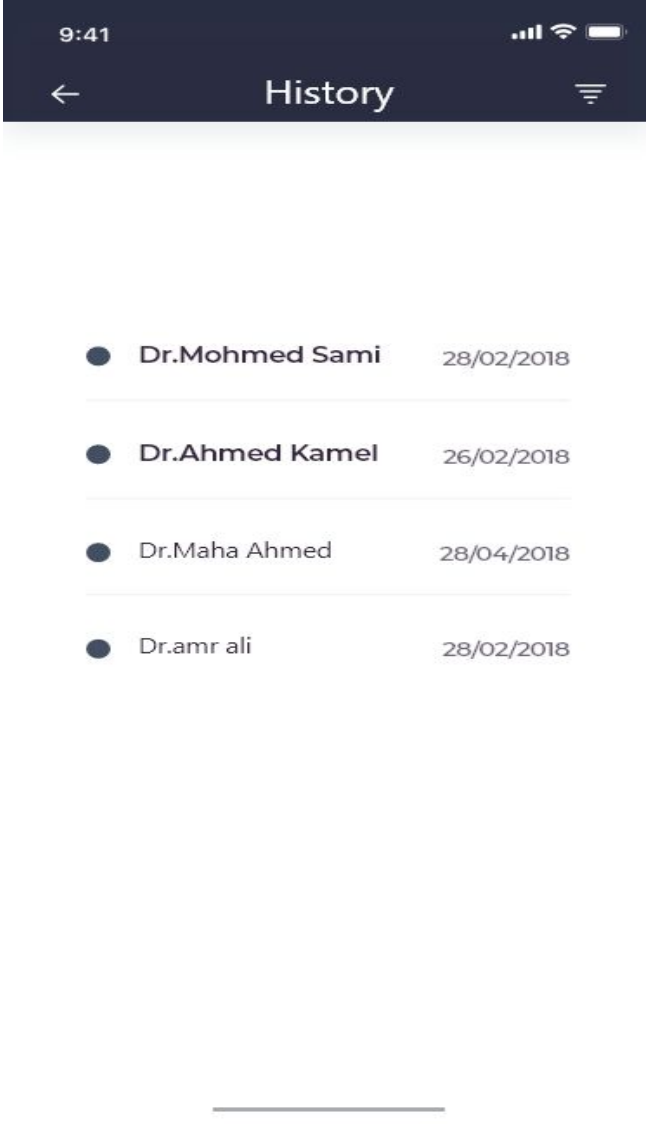
Show all doctors in patient region, patient can call doctor by click photo icon, show doctor info or book by clicking paper icon.

6. Emergency screen

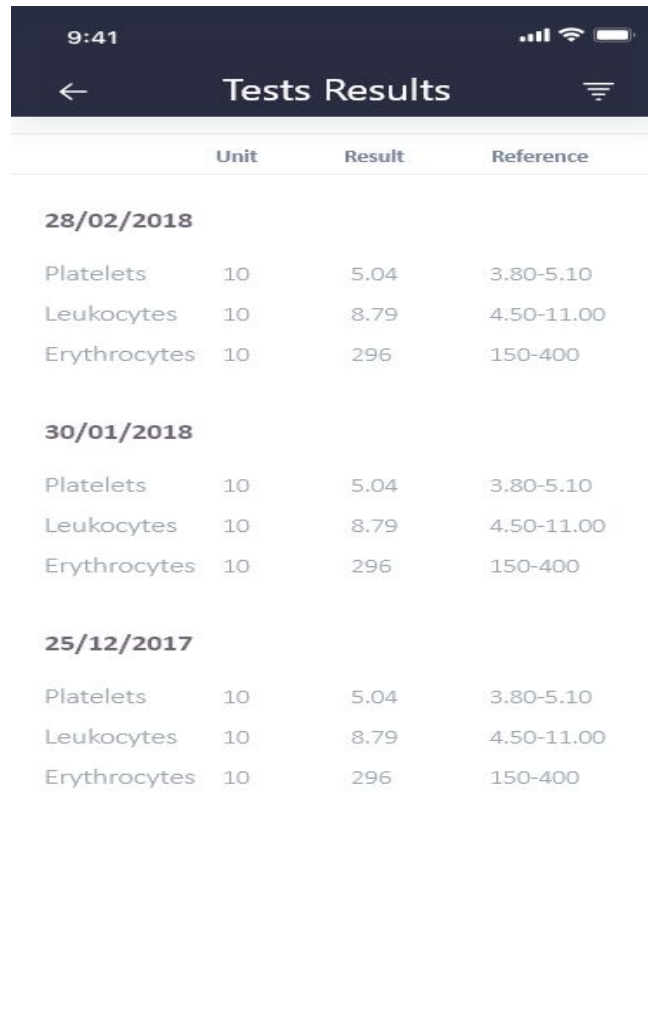


Request nearest ambulance automatically, in case an ambulance not available or canceled request, new request send to other nearest cars.

Injured person can cancel emergency request by click cancel button

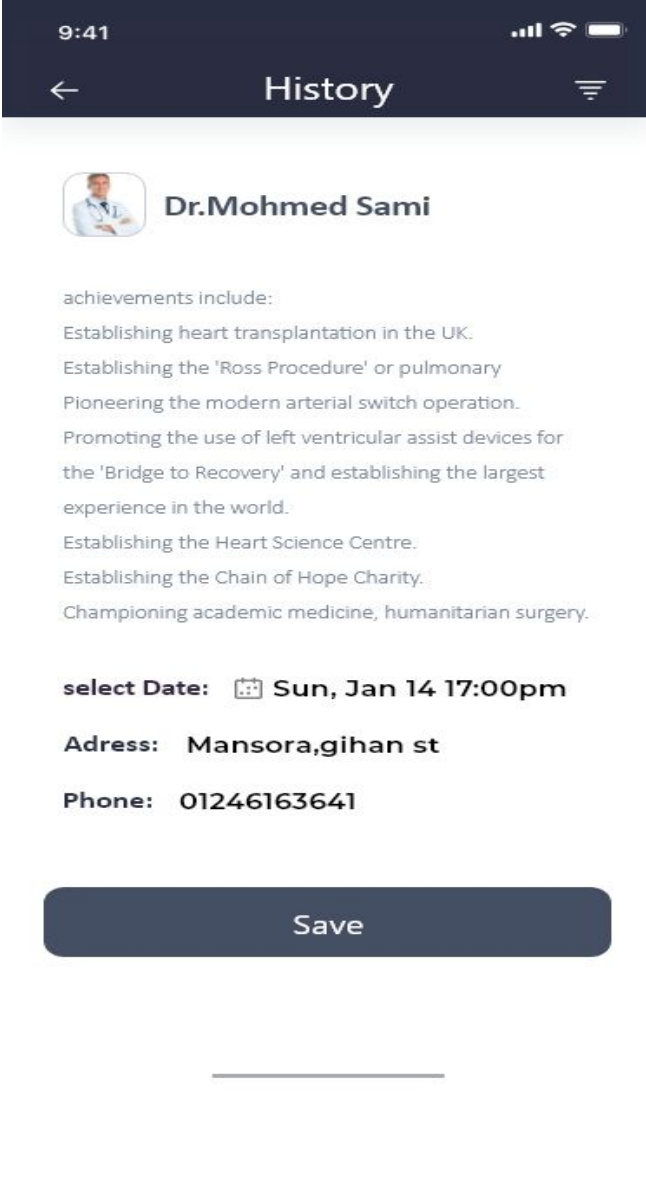
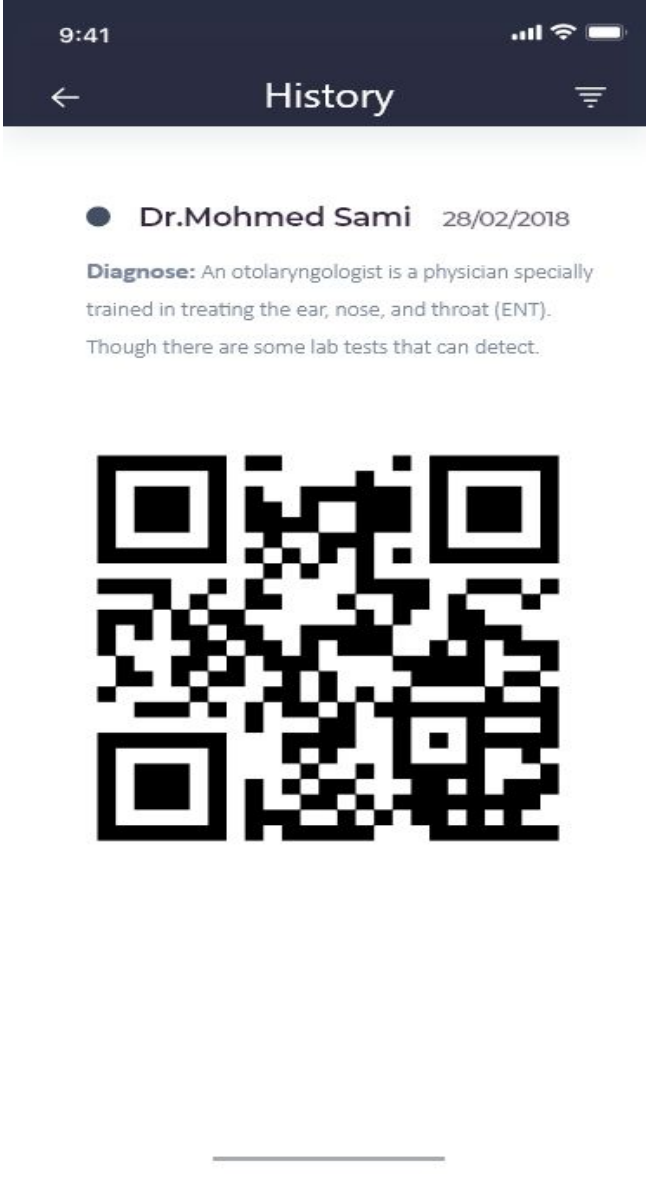
7. Medicine screen	8. History screen
	
<p>Show all patient medicines courses</p> <p>Click active tab to see current active medicine</p> <p>Click finished tab to see current finished medicine</p>	<p>Show patient booking history details</p>

9. Test Results screen



9:41 Tests Results			
	Unit	Result	Reference
28/02/2018			
Platelets	10	5.04	3.80-5.10
Leukocytes	10	8.79	4.50-11.00
Erythrocytes	10	296	150-400
30/01/2018			
Platelets	10	5.04	3.80-5.10
Leukocytes	10	8.79	4.50-11.00
Erythrocytes	10	296	150-400
25/12/2017			
Platelets	10	5.04	3.80-5.10
Leukocytes	10	8.79	4.50-11.00
Erythrocytes	10	296	150-400

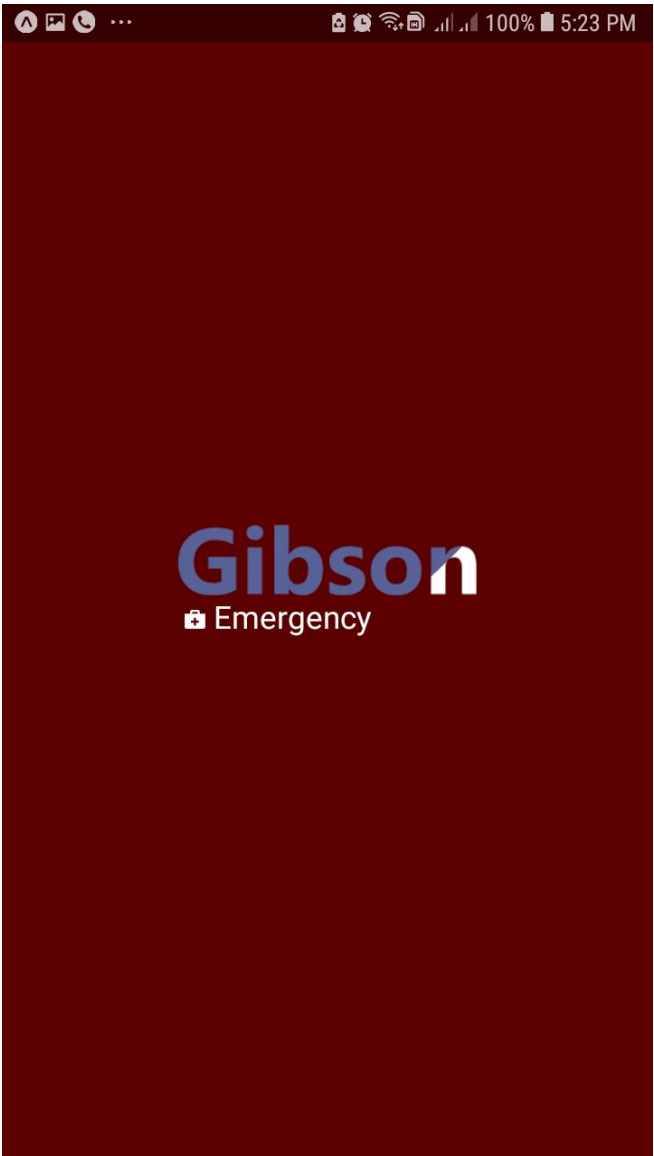
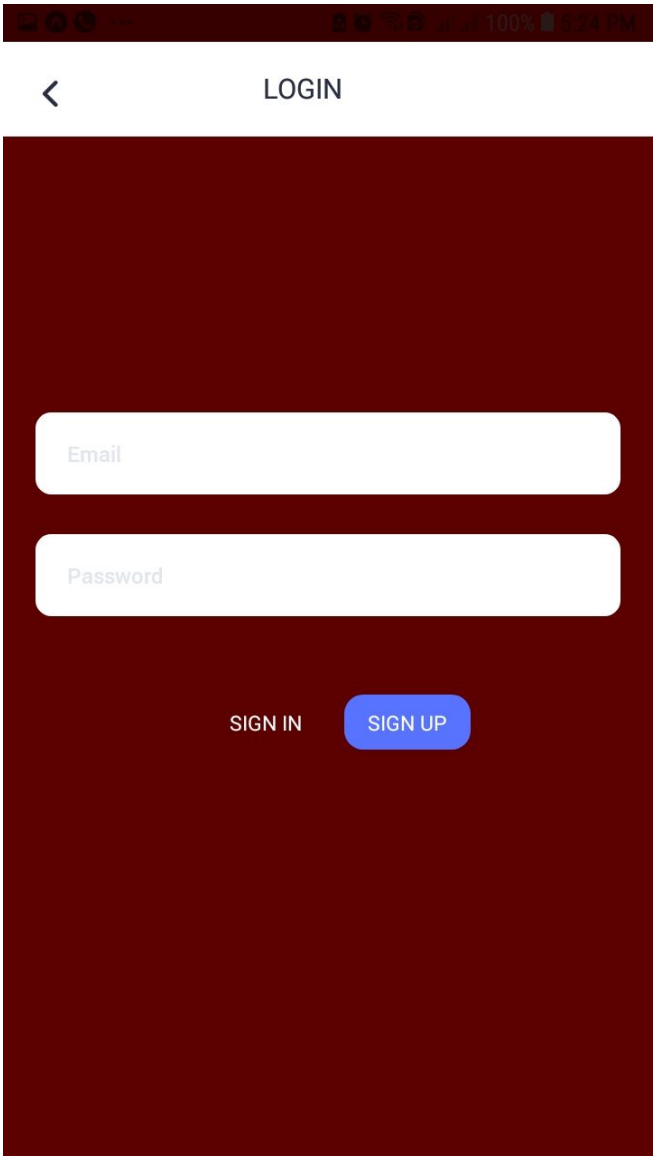
Patient analysis details

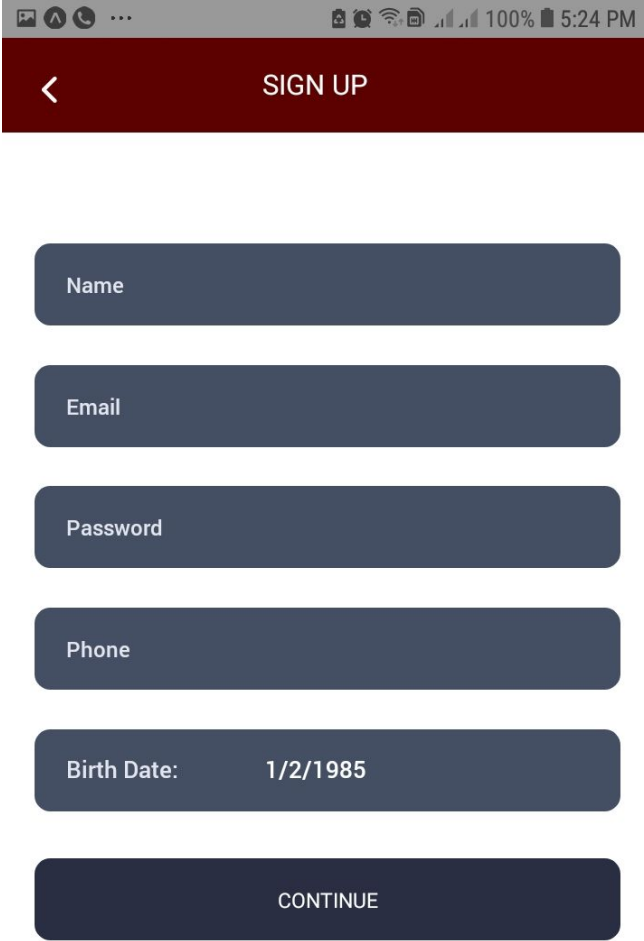
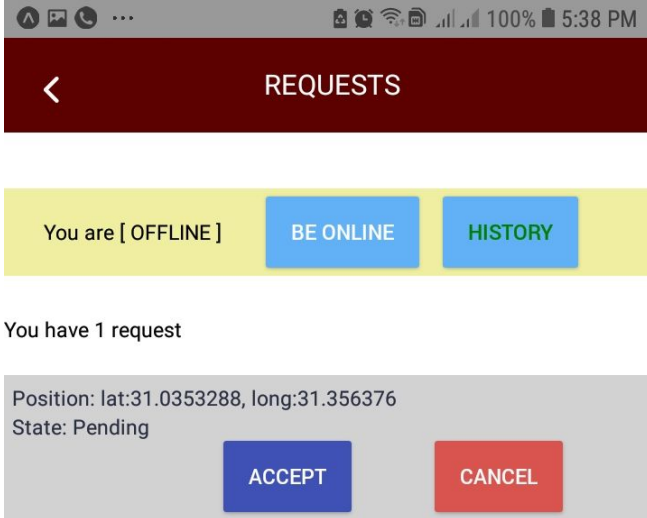
10. History of Doctor screen	11. History of Diagnose screen
	
Show all details about doctor	Show patient booking diagnose history details

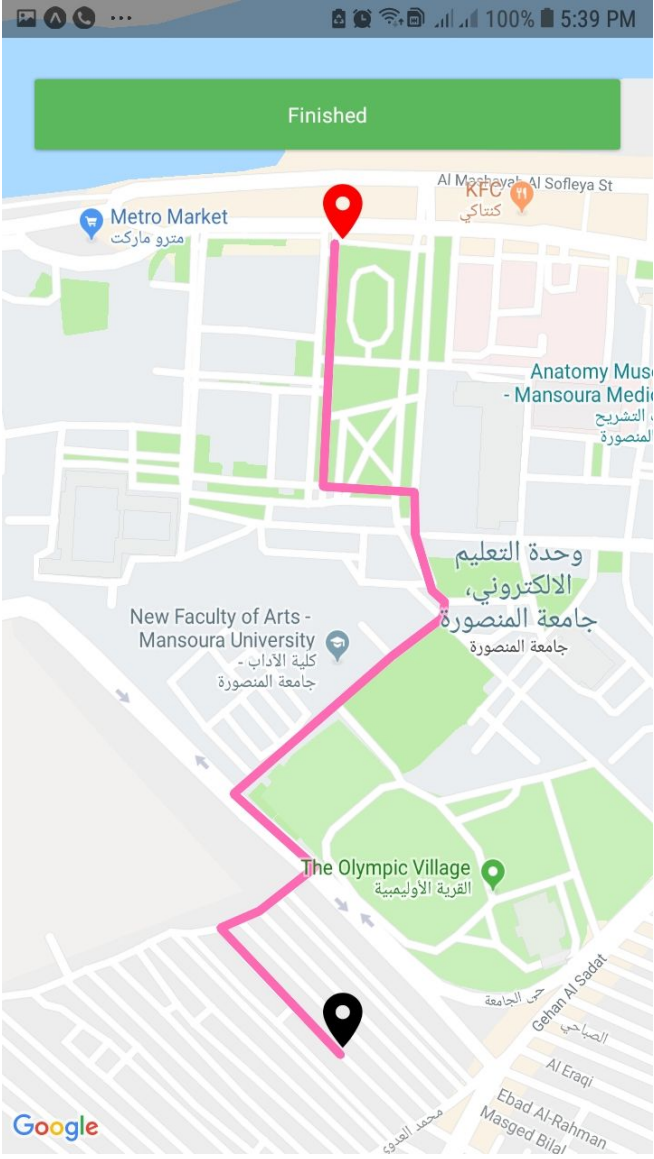

Emergency app

Screens:

- Welcome screen
- Login screen
- Register screen
- Home screen
- Request screen
- History screen

1. Welcome screen	2. Login screen
	
<p>Very first screen appears to ambulance driver</p> <p>After click logo redirect to login screen</p>	<p>Allow ambulance driver to signin is using his/her email and password</p> <p>After click signin redirect to home screen</p> <p>After click signup redirect to register screen</p>

3. Register screen	4. Home screen
	
<p>Allow ambulance driver to register on app, asking for patient details: name , email, password, phone, birthdate, and his/her current location</p> <p>After click continue redirect to home page</p>	<p>Show active ambulance driver requests, switch button to toggle driver availability state.</p> <p>Driver can accept request by click accept button, cancel request by click cancel button, see old requests history by click history button</p>


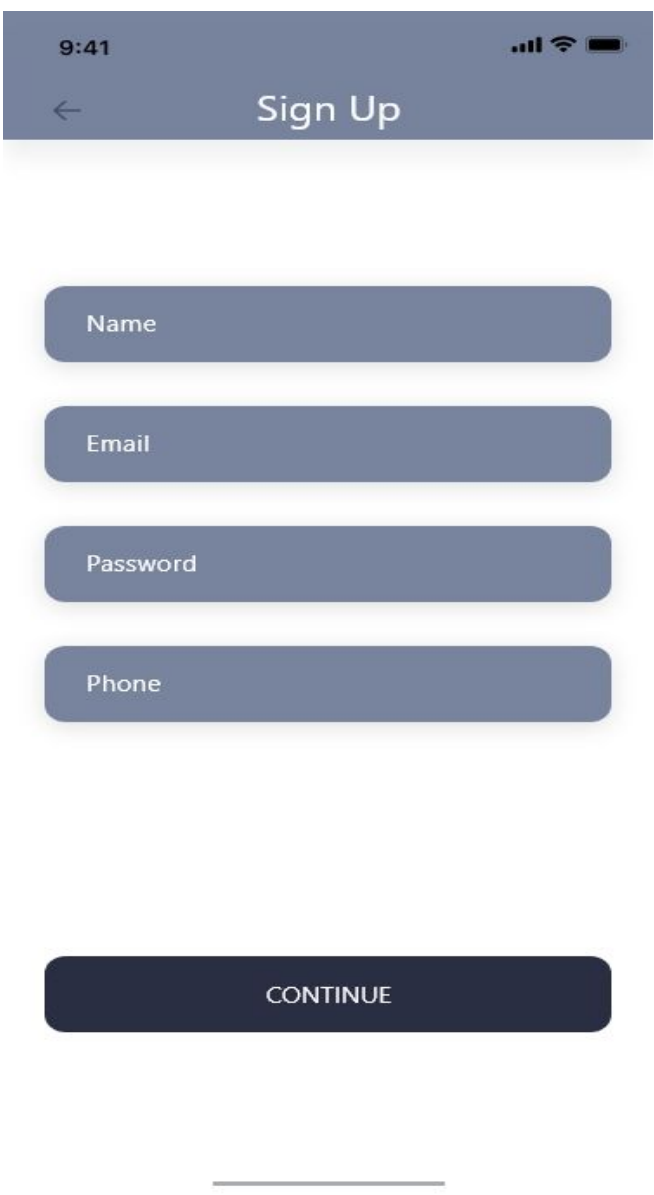
5. Request screen	6. History screen
	
<p>Show direction with shortest path from ambulance car and injured person.</p> <p>Black pin(current driver location) Red pin (target injured location)</p>	<p>Show old history requests to ambulance driver.</p>

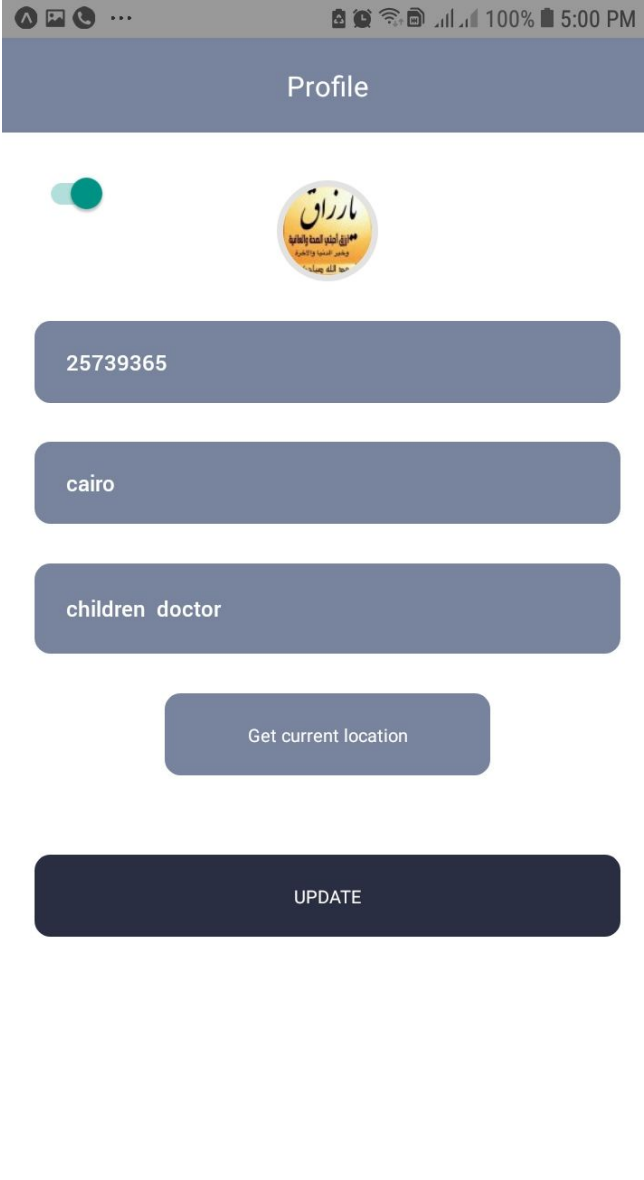
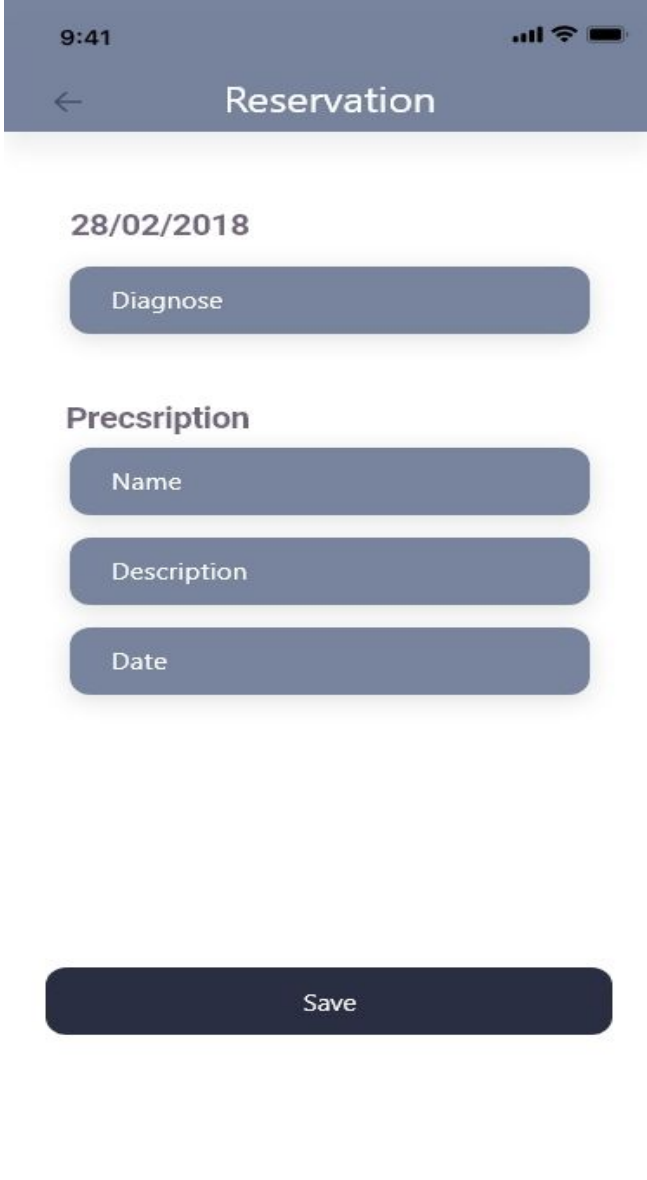
Doctor app

Screens:

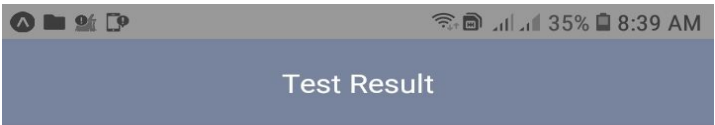
- Welcome screen
- Home screen
- Login screen
- Register screen
- Reservations screen
- Profile screen
- Test Result screen

1. Welcome screen	2. Home screen
	
<p>Very first screen appears to user</p> <p>After click logo redirect to login screen</p>	<p>Main home screen contains 3 icons for 3 screens: reservation, results, profile.</p>

3. Login screen	4. Register screen
 <p>The login screen has a dark blue background. At the top, there is a status bar with the time 9:41, signal strength, Wi-Fi, and battery icons. Below the status bar is a header with a back arrow and the word 'LOGIN'. The main content area contains two white input fields for 'Email' and 'Password'. At the bottom, there are two buttons: 'SIGN IN' and 'SIGN UP' (which is highlighted in blue).</p>	 <p>The register screen has a white background. At the top, there is a status bar with the time 9:41, signal strength, Wi-Fi, and battery icons. Below the status bar is a header with a back arrow and the text 'Sign Up'. The main content area contains four dark blue input fields for 'Name', 'Email', 'Password', and 'Phone'. At the bottom, there is a large dark blue button labeled 'CONTINUE'.</p>
<p>Allow doctor to signin is using his/her email and password</p> <p>After click signin redirect to home screen</p> <p>After click signup redirect to register screen</p>	<p>Allow doctor to register on app, asking for doctor details: name , email, password, phone..</p> <p>After click continue redirect to home page</p>

5. Profile screen	6. Reservations screen
	
<p>Show doctor profile info and allow to update profile, address, bio and location.</p>	<p>Show all patients bookings for the doctor and allow to see all details of each booking in details</p>

9. Test Results screen



03/24/19

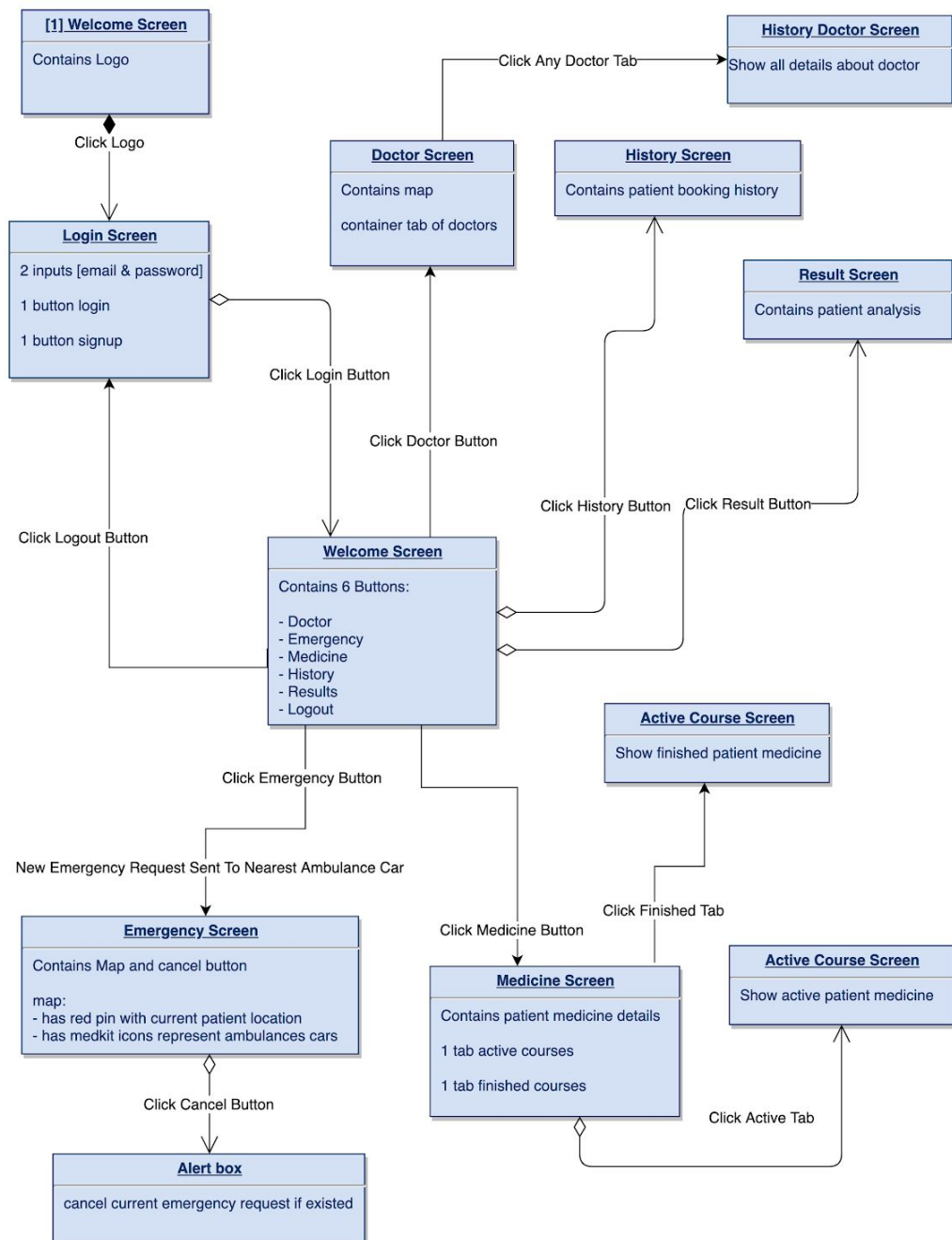
	Unit	Result	Refrence
Kentucky	Mn	1	1-4
Lotus	Nm	13.0	9.0-16.0

Patients analysis details

3.2 FlowCharts

Patient flowchart

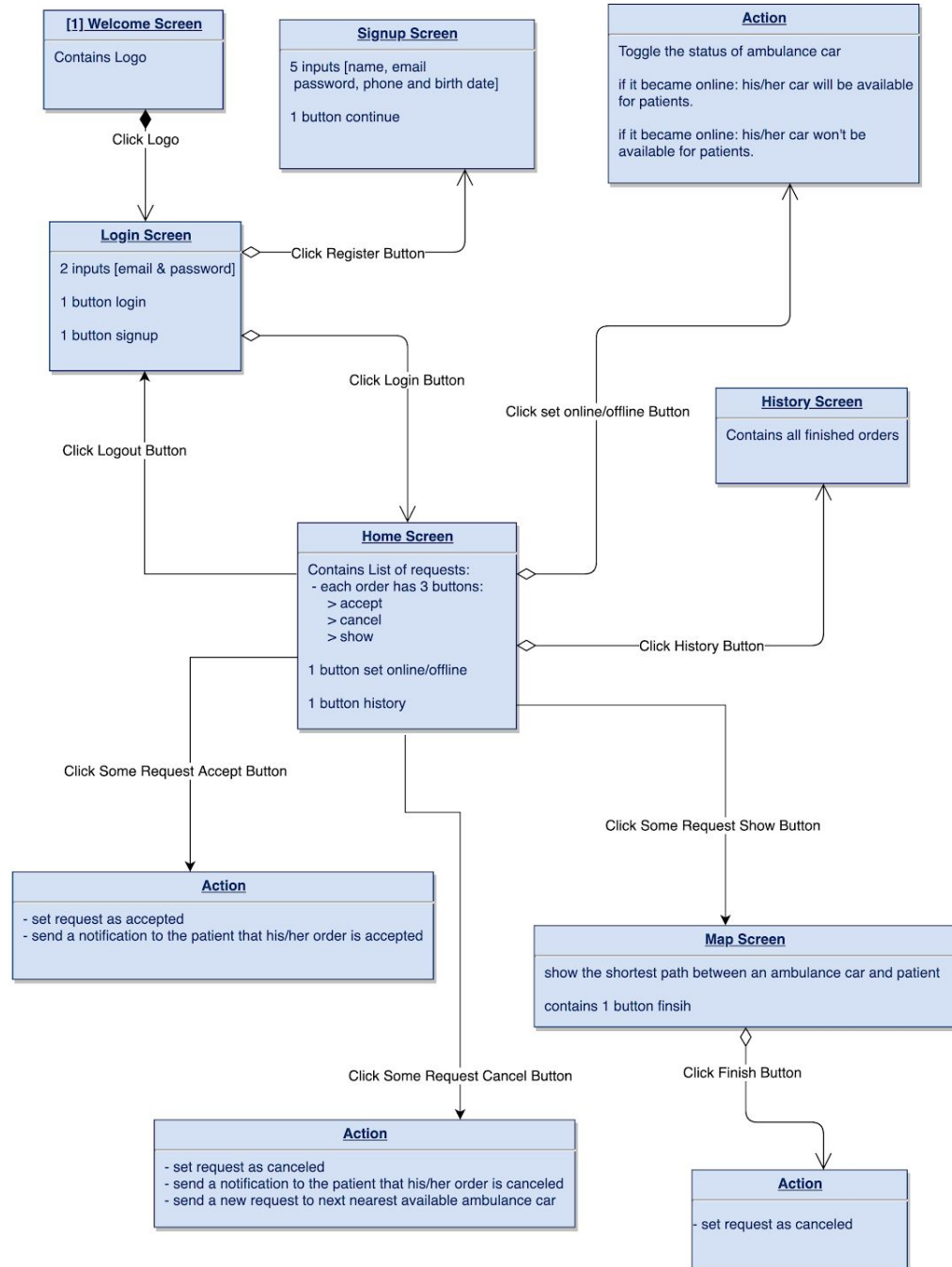
Patient Application



Emergency flowchart

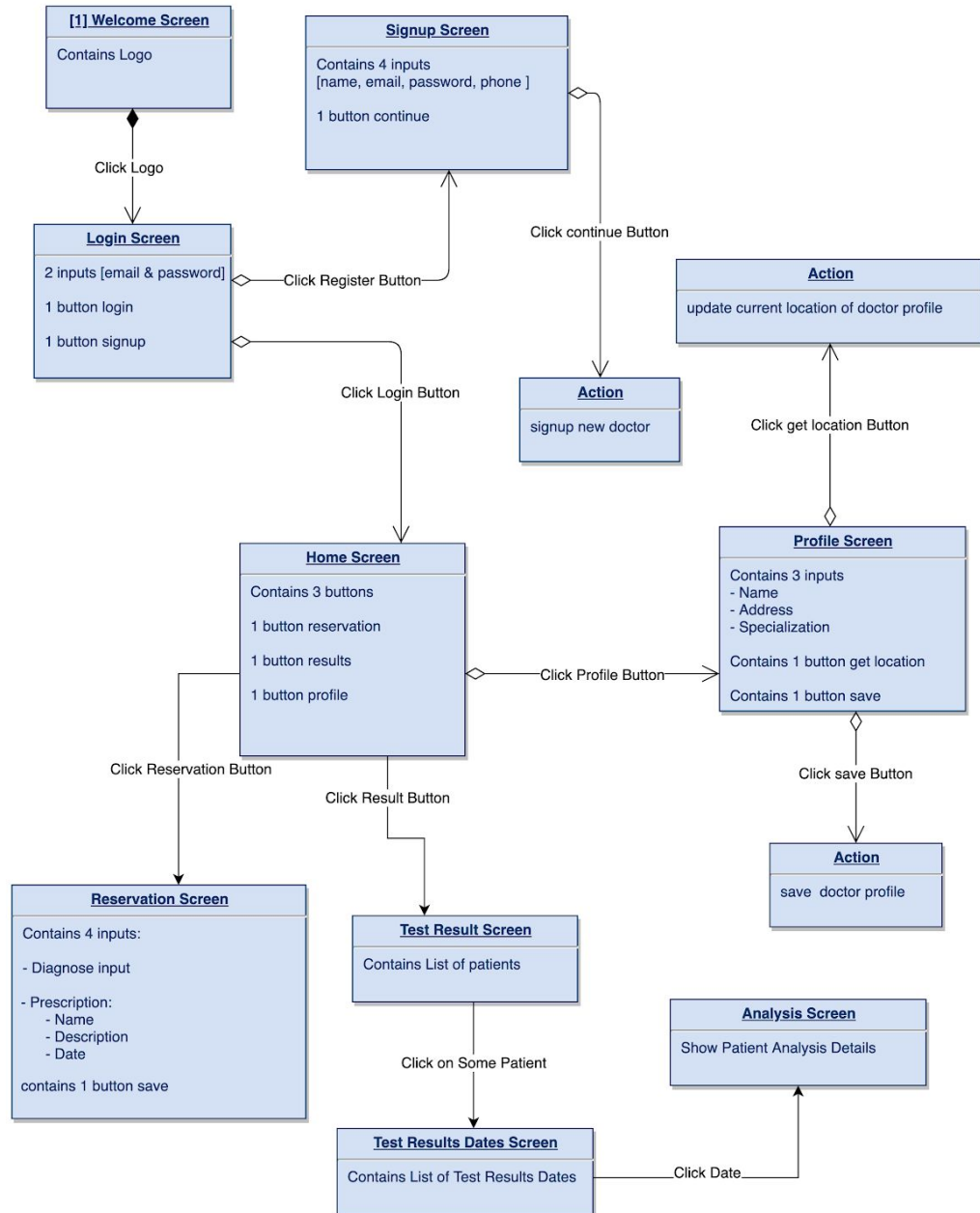
Emergency Application

1



Doctor flowchart

Doctor Application



Chapter 4

Functions and References

4.1 Project Functions:

Patient Application Functions

1. handleLogin()

Login function by the email and password.

```
handleLogin = () => {
  const { email, password } = this.state
  return firebase.auth().signInWithEmailAndPassword(email, password).then(() => {
    const resetAction = StackActions.reset({ index: 0,
      actions: [ NavigationActions.navigate({ routeName: 'HomeNav' }) ]
    })
    this.props.navigation.dispatch(resetAction);
  }).catch(err => { this.refs.toast.show(err.toString(), 1000, () => {
    // something you want to do at close });
  })
}
```

2. handleRegister()

Register new patient in the authentication and in the database.

```
handleRegister = () => {
  const { email, password, name } = this.state
  return firebase.auth().createUserWithEmailAndPassword(email, password).then(data => {
    Toast.show({ text: 'success, please wait', buttonText: "Okay",
      duration: 2500, type: 'success'
    })
    data.user.updateProfile({ displayName: name, })
    this.handleUserData(data.user.email)
  }).catch((err) => {
    Toast.show({
      text: 'something went wrong!',
      buttonText: "Okay", duration: 2500, type: 'danger'
    })
  })
}
```

3. getNearby()

Get the nearest doctors on the map from the database

```
getNearby() {
  const { coordinate } = this.state
  var firebaseRef = firebase.database().ref('Doctors');
  var geoFire = new GeoFire(firebaseRef);
  var geoQuery = geoFire.query({
    center: [coordinate.latitude, coordinate.longitude],
    radius: 2
  })
  let nearby = []
  geoQuery.on("key_entered", (key, location, distance) => {
    nearby.push(key)
    this.setState({ nearby, distance: distance * 1000 })
  })
  geoQuery.on('ready', () => {
    if (nearby.length !== 0) {
      this.getData()
    }
  })
}
```

3. getData()

Get the nearby doctor data

```
getData() {
  const db = firebase.firestore().collection('Doctors')
  let refs = []
  this.state.nearby.map(item => {
    refs.push(db.doc(item).get())
  })
  const data = Promise.all(refs)
  data.then(res => {
    let data = []
    res.map(snap => {
      data.push(snap.data());
    })
    this.setState({ data });
  })
}
```

4. handlebook()

Book the selected doctor

```
handleBook = () => {
  const { date, doctor, doctorEmail } = this.state
  let usr = firebase.auth().currentUser
  return firebase.firestore().collection('Booking').add({
    patient: usr.email,
    patientName: usr.displayName,
    Createdate: new Date().getTime(),
    date: date,
    doctor: doctor.email,
    doctorName: doctor.name,
    prescription: "",
    diagnose: "",
    finished: false
  }).then(() => {
    this.props.navigation.goBack()
  })
}
```

5. requestData()

Get the nearest emergency car from the database

```
requestData = () => {
  const { nearby } = this.state
  const db = firebase.firestore().collection('Emergency')
  let refs = []
  nearby.map(item => {
    refs.push(db.doc(item.key).get())
  })
  let data = Promise.all(refs)
  data.then(res => {
    let data = []
    res.map(snap => {
      if (snap.data() !== undefined && snap.data().state) {
        data.push(snap.data())
      }
    })
    this.handleRequest(data.reverse()[0])
  })
}
```

6. handleRequest()

Add new request to the selected car

```
handleRequest(data) {  
  firebase.firestore().collection('Requesting').add({  
    Date: new Date().getTime(),  
    carId: data.carId,  
    location: this.state.coordinate,  
    patient: firebase.auth().currentUser.email,  
    finished: false  
  }).then((doc) => {  
    this.setState({ requestId: doc.id })  
  })  
}
```

7. handleCancel()

Cancel the request to the car in the database

```
handleCancel = () => {  
  if (this.state.nearby.length != 0) {  
    firebase.firestore().collection('Requesting').doc(this.state.requestId).delete().then(() => {  
      this.props.navigation.goBack();  
    })  
  }  
}
```

8. gerDataTestResult()

Get the test results of the patient from the database

```
getDataTestResult={()=>{  
  let user=firebase.auth().currentUser.email  
  firebase.firestore().collection('TestResult').where('patient','==',user).orderBy('date').get().then(snap=>{  
    let data=[]  
    snap.forEach(doc=>{  
      data.push(doc.data())  
    })  
    this.setState({data:data.reverse(),loaded:true})  
  })  
}
```

Get the current active medicine for the patient

```
componentDidMountActiveMedic() {  
  let user = firebase.auth().currentUser.email  
  firebase.firestore().collection('Booking').where('patient', '==', user).get().then(snap => {  
    let data = []  
    snap.forEach(doc => {  
      if (doc.data().prescription.length != 0) {
```

```

    data.push(doc.data())
  }
})
this.setState({ data, loaded: true })
})
}

```

9. componentDidMountFinishedMedic()

Get the finished medicine for the patient

```

componentDidMountFinishedMedic() {
  let user = firebase.auth().currentUser.email
  firebase.firestore().collection('Booking').where('patient', '==', user).get().then(snap => {
    let data = []
    snap.forEach(doc => {
      if (doc.data().prescription.length !== 0) {
        data.push(doc.data())
      }
    })
    this.setState({ data })
  })
}

```


Doctor Application Functions

1. handleLogin()

Login function by the email and password.

```
handleLogin = () => {
  const { email, password } = this.state
  return firebase.auth().signInWithEmailAndPassword(email, password).then(() => {
    const resetAction = StackActions.reset({
      index: 0,
      actions: [
        NavigationActions.navigate({ routeName: 'HomeNav' })
      ]
    })
    this.props.navigation.dispatch(resetAction);
  }).catch(err => {
    this.refs.toast.show(err.toString(), 1000, () => {
      // something you want to do at close
    });
  })
}
```

2. handleRegister()

Register new doctor in the authentication and in the database.

```
handleRegister = () => {
  const { email, password, name } = this.state
  return firebase.auth().createUserWithEmailAndPassword(email, password).then(data => {
    Toast.show({
      text: 'success, please wait',
      buttonText: "Okay",
      duration: 2500,
      type: 'success'
    })
    data.user.updateProfile({
      displayName: name,
    })
    this.handleUserData(data.user.email)
  }).catch((err) => {
    console.log(err);
    Toast.show({
      text: 'something went wrong!',
      buttonText: "Okay",
      duration: 2500,
      type: 'danger'
    })
  })
}
```

```

})
}

```

3. componentDidMount()

Get the booked patient for the doctor by the email.

```

componentDidMount() {
  firebase.auth().onAuthStateChanged(user => {
    firebase.firestore().collection('Booking').where('doctor', '==', user.email).get().then(snap => {
      let data=[]
      snap.forEach(doc => {
        data.push(doc.data())
      })
      let result = data.filter(function (a) {
        return !this[a.patientName] && (this[a.patientName] = true);
      }, Object.create(null));
      this.setState({data:result,loaded:true})
    })
  })
}

```

4. handleData()

Add patient diagnose on picked request and the patient prescription.

```

handleData = () => {
  const { data, diagnose, key, name, desc, date, prescription } = this.state
  if (name.length == 0 && desc.length == 0) {
    firebase.firestore().collection('Booking').doc(key).update({
      diagnose: diagnose,
      finished: true,
      prescription: prescription,
    }).then(() => {
      this.props.navigation.goBack()
    })
  } else {
    let items = prescription
    let item = { name: name, desc: desc, date: date }
    items.push(item)
    this.setState({ prescription: items, name: "", desc: "" })
  }
}

```

5. handleDataView()

Get the patient previous Test results or add new Test result.

```

handleDataView() {
  const { name, unit, refrence, result, data, date, patient } = this.state
  if (name.length != 0 && result.length != 0) {
    let item = { name: name, unit: unit, refrence: refrence, result: result }
    let items = data
    items.push(item)
  }
}

```

```

        this.setState({ data: items, name: "", refrence: "", result: "", unit: "" })
    } else if (data.length !== 0 && date.length !== 0) {
        firebase.firestore().collection('TestResult').add({
            patient: patient,
            date: date,
            data: data,
            doctor: firebase.auth().currentUser.email
        }).then(() => {
            this.props.navigation.goBack()
        })
    }
}

```

6. handleUpdate()

Update the doctor information's in the profile page.

```

handleUpdate = async () => {
    const { key, coordinate, phone, bio, address, thumbnail, newThumbnail } = this.state
    if (newThumbnail !== null) {

        firebase.firestore().collection('Doctors').doc(key).update({
            coordinate: coordinate,
            address: address,
            phone: phone,
            bio: bio,
            thumbnail: newThumbnail
        }).then(() => {
            this.props.navigation.goBack()
        })
    } else {
        var firebaseRef = firebase.database().ref('Doctors');
        var geoFire = new GeoFire(firebaseRef);
        geoFire.set(key, [coordinate.latitude, coordinate.longitude])
        firebase.firestore().collection('Doctors').doc(key).update({
            coordinate: coordinate,
            address: address,
            phone: phone,
            bio: bio,
            thumbnail: thumbnail
        }).then(() => {
            this.props.navigation.goBack()
        })
    }
}

```

7. selectPhoto()

Open the gallery to pick image as profile picture then call other functions.

```

selectPhoto = async () => {
    const status = await Permissions.getAsync(Permissions.CAMERA_ROLL);
    if (status) {

```

```

const result = await ImagePicker.launchImageLibraryAsync();
if (!result.cancelled) {
  this.setState({ thumbnail: result.uri })
  this.shrinkAsync(result.uri).then(res => {
    this.uploadPhoto(res.uri).then(save => {
      this.setState({ newThumbnail: save })
    })
  })
}
};

```

8. shrinkAsync()

Comprise the image size to save database storage.

```

async shrinkAsync(uri) {
  const img = await ImageManipulator.manipulateAsync(uri, [{ resize: { width: 500 } }], {
    compress: 0.5,
  });
  return img
}

```

9. uploadPhoto(uri)

Upload the selected photo to firebase storage for the doctor profile.

```

async uploadPhoto(uri) {
  return new Promise(async (res, rej) => {
    var name = uuid.v1();
    const blob = await this._urlToBlob(uri)
    const ref = firebase.storage().ref(name);
    const unsubscribe = ref.put(blob).on(
      'state_changed',
      state => {
        this.setState({ percent: ((state.bytesTransferred / state.totalBytes) * 100).toFixed() })
      },
      err => {
        unsubscribe();
        rej(err);
      },
      async () => {

        unsubscribe();
        const url = await ref.getDownloadURL();
        res(url);
      },
    );
  });
}

```

10. toggleState(state)

Change the doctor status from online or offline.

```
toggleState(state) {  
  let emRef = firebase.firestore().collection('Doctors').doc(this.state.key);  
  emRef.update({ 'state': state })  
}
```

Emergency Application Functions

1. **handleLogin()**

Login function by the email and password.

```
handleLogin=()=>{
  const {email,password} = this.state
  return firebase.auth().signInWithEmailAndPassword(email,password).then( ()=>{
    this.props.navigation.replace('Home')
  }).catch(err=>{
    this.refs.toast.show(err.toString(), 1000, () => {
      // something you want to do at close
    });
  })
}
```

2. **handleRegister()**

Register new ambulance driver in the authentication and in the database.

```
handleRegister=()=>{
  const {email,password} =this.state
  return firebase.auth().createUserWithEmailAndPassword(email,password).then(data=>{
    Toast.show({
      text: 'success, please wait',
      buttonText: "Okay", duration: 2500, type:'success'
    })
    this.handleUserData(data.user.email)
  }).catch((err)=>{
    let error=err.code.split('/')
    Toast.show({
      text: error[1],
      buttonText: "Okay", duration: 2500, type:'danger'
    })
  })
}
```

3. componentDidMount()

The first function that run on open , get the current user email then get the current position of the car and update the request.

```
async componentDidMount() {  
  let email = firebase.auth().currentUser.email;  
  
  console.log(email);  
  
  let coordinate = {};  
  await navigator.geolocation.getCurrentPosition(pos => {  
    coordinate = {  
      latitude: pos.coords.latitude,  
      longitude: pos.coords.longitude,  
      latitudeDelta: 0.015,  
      longitudeDelta: 0.0121  
    };  
    this.setState({ coordinate, email });  
    this.checkUser();  
    console.log("geo..");  
    let interval = setInterval(() => {  
      this.updateRequests();  
    }, 4000);  
    this.setState({ interval });  
  });  
}
```

4. checkUser()

Check if the user logged in status to continue or return to login

```
checkUser() {  
  let interval = this.state.interval;  
  let x = firebase  
    .firestore().collection("Emergency")  
    .where("email", "==", this.state.email)  
    .get() .then(snap => {  
    let usr = null;  
    snap.forEach(doc => {  
      usr = doc.data();  
    });  
    if (usr == null) {  
      alert("You are not an Emergency worker!");  
      clearInterval(interval);  
      this.setState({ interval: null });  
      Firebase .auth() .signOut()  
        .then(() => this.props.navigation.navigate("Login"));  
    } else {  
      let user = usr;  
      this.setState({ user });  
      this.updateRequests();  
    }  
  });  
}
```


5. updateRequests()

Get the patients requests from the database by the user email.

```
updateRequests() {  
  let ref = firebase.firestore().collection("Requesting");  
  ref.orderBy("date");  
  ref  
    .where("employerUsername", "==", this.state.user.email)  
    .get() .then(snap => {  
      let data = [];  
      snap.forEach(doc => {  
        data.push({ ...doc.data(), ...{ key: doc.id } });  
      });  
      let state = data.state;  
      this.setState({ data });  
    });  
}
```

6. updateRequests()

Change the status of the car from online or offline on click.

```
toggleState() {  
  if (this.state.user.name == undefined) return;  
  
  let emRef = firebase  
    .firestore() .collection("Emergency") .doc(this.state.user.name);  
  emRef.get().then(snap => {  
    let state = !snap.data().state;  
    this.state.user.state = state;  
    this.setState({ user: this.state.user });  
    emRef.update({ state: state });  
  });  
}
```

7. handleAccept()

Accept the request and send the response to the database.

```
handleAccept(item) {  
  let ref = firebase  
    .firestore()  
    .collection("Requesting")  
    .doc(item.key);  
  ref.get().then(snap => {  
    ref.update({ accepted: true, canceled: false });  
    alert("Request Accepted!");  
  });  
}
```

8. handleCancel()

Cancel the current request in the database.

```
handleCancel(item) {  
  let ref = firebase  
    .firestore()  
    .collection("Requesting")  
    .doc(item.key);  
  ref.get().then(snap => {  
    ref.update({ finished: true, accepted: false, canceled: true });  
    alert("Request Canceled!");  
  });  
}
```

9. getActiveRequests()

Get all the unfinished or waiting request from the database.

```
getActiveRequests() {  
  let ret = [];  
  if (this.state.data == undefined || !this.state.data.length)  
    return [];  
  
  this.state.data.map((item, i) => {  
    if (!item.finished && !item.cancel) ret.push(item);  
  });  
  return ret;  
}
```

10. finishRequest()

Set the request state as finished and push it to history.

```
finishRequest(){  
  let docRef =  
    firebase.firestore().collection('Requesting').doc(this.state.target.key);  
  docRef.get().then(function (doc) {  
    console.log(doc.data())  
    docRef.update({'finished': true, 'accepted': true, 'canceled': false})  
    alert('Finished!')  
  });  
}
```

4.2 Future Work

- Use finger ring button to send emergency requests.
- Add chat option between doctors and patients as part of gibson application.
- Create gibson web application and desktop application.
- Add charts and more analytics details for patients
- Add search for nearest available pharmacy that has all patient medicines feature



4.3 References

React Native: <https://facebook.github.io/react-native/docs/getting-started.html>

Firebase Cloud Firestore: <https://firebase.google.com/docs/firestore>

Firebase Authentication: <https://firebase.google.com/docs/auth>

Firebase Realtime Database: <https://firebase.google.com/docs/database>

GeoFire for JavaScript:

<https://firebaseopensource.com/projects/firebase/geofire-js>

GeoFire API: <https://github.com/firebase/geofire-js/blob/master/docs/reference.md>

Google maps: <https://developers.google.com/maps/documentation>

Google Directions: <https://developers.google.com/maps/documentation/directions/start>

Expo : <https://docs.expo.io/versions/latest>

Ionicons: <https://ionicons.com>

Material Design: <https://material.io/tools/icons/?style=baseline>

Visual Studio Code: <https://code.visualstudio.com/docs>

Stackoverflow: <https://stackoverflow.com/questions/tagged/react-native>