** **

**Information Systems Department**

**Faculty of Computers and Artificial Intelligence**

**Cairo University**

**AMUN MR**

|  |  |  |
| --- | --- | --- |
| **Names** | **IDS** | **Department** |
| **Ahmed Bassem Fouad Hussein** | **20160008** | **IS-DS** |
| **Abdelrahman Magdy Hassan Ismail** | **20160134** | **IS-DS** |
| **Ibrahim El Sayed Ibrahim El Sayed** | **20160003** | **IS-DS** |
| **Eslam Hamada Mohamed Mahmoud** | **20160053** | **IS-DS** |
| **Omar Mokhtar Hassan Mohamed** | **20160165** | **IS-DS** |

**Supervised by**

**Dr. Galal Hassan**

**TA. Dina Amr**

**Academic Year 2019-2020**

Table of Contents

[Chapter 1: Introduction 3](#_Toc48104244)

[1.1 Background and Motivation 3](#_Toc48104245)

[1.2 Problem Statement 4](#_Toc48104246)

[1.3 Scope 4](#_Toc48104247)

[1.3.1 In-Scope 4](#_Toc48104248)

[1.3.2 Out of Scope 5](#_Toc48104249)

[1.4 Project Objectives 6](#_Toc48104250)

[1.5 Methodology 6](#_Toc48104251)

[1.5.1 Methodology Selection 8](#_Toc48104252)

[1.6 Solution Statement 9](#_Toc48104253)

[1.7 Limitations 9](#_Toc48104254)

[1.8 Related Work 10](#_Toc48104255)

[1.9 Storyboard 11](#_Toc48104256)

[1.10 Stakeholders 12](#_Toc48104257)

[Chapter 2: Requirements Analysis 13](#_Toc48104258)

[2.1 Context Model 13](#_Toc48104259)

[2.1.1 Context Diagram 14](#_Toc48104260)

[2.1.2 System Architecture Diagram 15](#_Toc48104261)

[2.2 Functional Requirements 16](#_Toc48104262)

[2.3 Non-Functional Requirements 18](#_Toc48104263)

[2.3.1 Usability 18](#_Toc48104264)

[2.3.2 Reliability 18](#_Toc48104265)

[2.3.3 Security 18](#_Toc48104266)

[2.3.4 Performance 19](#_Toc48104267)

[2.3.5 Availability 19](#_Toc48104268)

[2.4 Use Case Models 19](#_Toc48104269)

[2.4.1 Actors 19](#_Toc48104270)

[2.4.2 Use Case Diagrams 20](#_Toc48104271)

[2.4.3 Use Case Description 22](#_Toc48104272)

[2.5 Domain Model 22](#_Toc48104273)

[2.5.1 Domain Model Diagram 23](#_Toc48104274)

[2.5.2 Domain Model Description 24](#_Toc48104275)

[2.6 Sequence Models 24](#_Toc48104276)

[2.6.1 Patient Sequence Diagrams 25](#_Toc48104277)

[2.6.2 Patient Activity Diagram 30](#_Toc48104278)

[2.7 Entity Relationship Diagram 32](#_Toc48104279)

[2.8 Prototype 34](#_Toc48104280)

[Chapter 3: Design 35](#_Toc48104281)

[3.1 Tools Used 35](#_Toc48104282)

[3.1.1 MongoDB 35](#_Toc48104283)

[3.1.2 Firebase 36](#_Toc48104284)

[3.1.3 Flutter 36](#_Toc48104285)

[3.1.4 NodeJS 36](#_Toc48104286)

[3.1.5 Postman 36](#_Toc48104287)

[3.1.6 Vue.js 37](#_Toc48104288)

[3.1.7 GitHub 37](#_Toc48104289)

[3.2 Design Validation and Verification 37](#_Toc48104290)

[3.2.1 Design Validation 37](#_Toc48104291)

[3.2.2 Design Verification 39](#_Toc48104292)

[3.4 Component Diagram 40](#_Toc48104293)

[Chapter 4: Testing 41](#_Toc48104294)

[4.1 Test Plan 41](#_Toc48104295)

[4.1.1 Interface Testing 41](#_Toc48104296)

[4.1.2 API Testing 43](#_Toc48104297)

[4.1.3 System Testing 52](#_Toc48104298)

[4.2 Planning 56](#_Toc48104299)

[4.3 Team Roles 57](#_Toc48104300)

[Chapter 5: Evaluation 58](#_Toc48104301)

[5.1 Project Overview 58](#_Toc48104302)

[5.2 Approach Followed 58](#_Toc48104303)

[5.3 Possible future work 58](#_Toc48104304)

[5.4 Achievements Made 59](#_Toc48104305)

[5.5 Lessons Learnt 59](#_Toc48104306)

[References 60](#_Toc48104307)

[Appendix 61](#_Toc48104308)

[Appendix A 61](#_Toc48104309)

# Chapter 1: Introduction

In our country, it is complicated to keep hold of medical records, they may get damaged or lost over long periods of time, even if they aren’t lost, it is a burden to carry medical records around while going to a doctor’s appointment. It is becoming clear that a new way of handling patient’s medical records is needed in our community. In the current model, most records are stored on paper, even patients’ previous visits are stored on papers, which is in our time with many different technologies available it is not an efficient way to store medical records. Smart phones offer advanced technologies with functionalities similar to a personal computer. While offering a standardized platform for application developers, a smart phone performs as complete operating system software. With smart phones being widely spread, nearly at the hands of each human being, it is beneficial to create a system that holds patients’ medical records. Which helps patients store their medical vitals or medical reports on smart phones. It decreases interaction between patients and doctors, the system proposed removes the burden of carrying a ton of medical reports while going to an appointment, which makes visits to doctors easier. The proposed system also stores patients’ visits to medical facilities in a centralized database, with the details of the visit. This project aims to make the whole process of a doctor’s appointment easier on both sides, the patients’ side; as it holds all medical records. The doctors’ side; easier for doctors to navigate through patient’s medical history.

## 1.1 Background and Motivation

In Egypt, the number of patients increases every year excessively along with their medical records, so they need a service to facilitate their interactions with the doctors and laboratories. As the usage of information systems and computer science are growing in the medical field, so we need to store medical records and medical information digitally and securely, instead of hardcopies. Medical records are going to be stored in a centralized database, as centralized databases are more secure and reduces conflicts, which separates top management that process the data, lower-level managers that use data inappropriately. There will be two applications; one for patients, the other is for doctors, medical facilities and clerks. The patients’ application is where the electronic medical records of the patients appears. The other application is where medical records are written and edited.

## 1.2 Problem Statement

In a patient’s life there is a lot of medical records that may be carried with him in every visit to the doctor includes (Medical history reports, Radiographs, X-Rays, etc.) which makes excess visits to the medical facilities. Medical records hardcopies are not centralized in one place and may be lost or damaged through the years, so there should be a centralized system that collects medical records securely and provide analysis for the data entered.

## 1.3 Scope

### 1.3.1 In-Scope

* We aim to develop two mobile applications; one for doctors and the other for patients, both applications will be supported on iOS and Android platforms.
* Our applications will allow medical records to be available on mobile phones, synchronized with doctor’s or hospital’s existing patient management database.
* Laboratory clerks will be able to send radiographs and other medical tests to patients via mobile phone.
* Clinic clerks will be able to add and delete medical records and history.
* Doctors will be able to update records.
* Doctors will be able to view patients connected with them.
* Patients will be able to add reminders for prescriptions or appointments.

### 1.3.2 Out of Scope

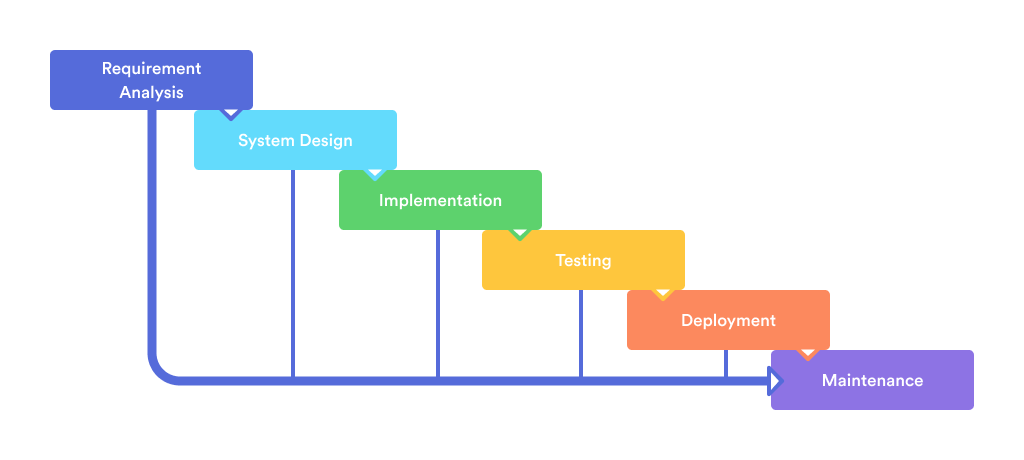
* This system isn’t concerned with the billing system of doctors or medical facilities.
* This system isn’t concerned with the appointment between doctor and patient.
* This system deals with a specific criterion of medical facilities and doctors.
* This system doesn’t support old operating systems.
* This system isn’t concerned with the height or weight changes for the user, it only adds blood pressure and glucose measurements for the patient’s profile.

## 1.4 Project Objectives

Our application is aiming to:

* Help patients and doctors manage medical records digitally.
* The application decreases the interaction between doctors and patients.
* The application centralizes medical records in one place to be managed easily.
* Tracking patients’ measured data over certain periods of time (Cholesterol, blood sugar, blood pressure)
* Helps doctors viewing their patients’ medical records.
* Helps Medical Facilities to track patient’s visits
* Eases the storage of medical records.

## 1.5 Methodology



This project will use Waterfall model, the waterfall model divides software development process into separate phases. The outcome of each phase is the input for the next phase. Waterfall model allows for departmentalization and control. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

1. Requirement analysis:
   * In this stage we define the expectations of the users for the application mainly the patients and the doctors. This will be achieved by conducting surveys. Then, the requirements will be analyzed as they are coming from different sources and backgrounds. Finally, all the requirements will be merged to satisfy all the users. At the end of this stage, the requirements will be documented in different formats such as use case models and storyboard.
2. System design
   * This stage is the bridge that takes us from requirements to solutions, its objectives is to define and organize the components of the final solution as it serve as the blueprint for the next stage(implementation stage).In this stage, the design of application, databases and interfaces(both user and system) are identified. Mostly, the interface of our mobile application will be designed along with the sequence diagrams and class diagrams.
3. Implementation
   * In this stage, the design that is identified in the previous stage is implemented through transforming each use case into a working functional application, the applications are implemented using a cross-platform mobile development framework meaning that the source code will be the same for both smart phones operating systems we will be talking about it in the next chapters.
4. Testing
   * Once the system has been developed, it will be tested to ensure that it is working as expected without any defects. A test plan will be followed and conducted. Different test cases will be executed according to two types; interface testing, API testing.
5. Deployment
   * Both applications will be added on Google play store and iOS “App store”, and a website will be developed. Some additional test cases are executed related to system testing.

### 1.5.1 Methodology Selection

We used UML as object-oriented design methodology to provide a clear way to visualize the design of the system. Good object-oriented design helps in coding and testing phases by first determining the functionality of the system without consideration of implementation constraints, and then consider how to make specific solutions to these abstract requirements, and refine them into detailed designs and codes by constraints such as technology and budget.

Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs.

We chose waterfall methodology as it was appropriate for this project for the following reasons;

1. Requirements are very well documented, clear and fixed
2. Phases are processed and completed one at a time
3. Well understood milestones
4. Easy to arrange tasks.

## 1.6 Solution Statement

The solution architecture consists of; two applications which can be used on both of the most used operating systems, iOS and android and a website for clerks and medical facility administrator. An application for patients that contains the patient medical status and other medical information that concerns this patient, and an application that helps doctors and medical facilities; Enter medical records for patients, track patients’ health record. This solution aims to centralize the medical records to make doctor appointments easier, reducing interactions between patients and medical facilities.

## 1.7 Limitations

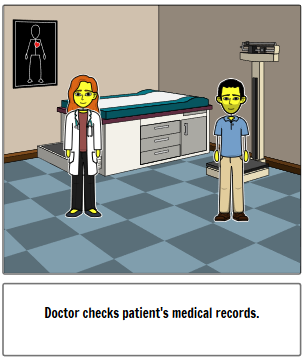
System limitations represent the environment surrounding the system in which system administrators have no control and it may affect the system performance. They can be summarized as follows:

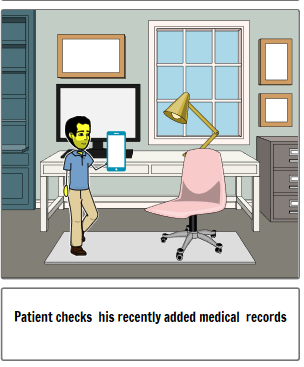
* The application deals with a specific criterion of patients and hospitals.
* Patients can only access the application through smartphones.
* The application needs internet connection to perform some features.
* Clerks cannot remove medical records; they can only add them to a patient’s profile.
* Medical facility administrators cannot delete patient’s visit.
* Old operating systems aren’t supported.

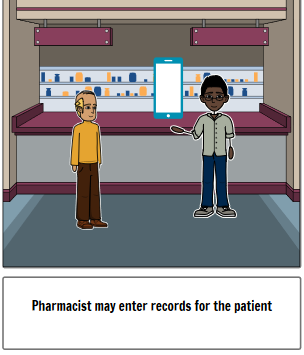
## 1.8 Related Work

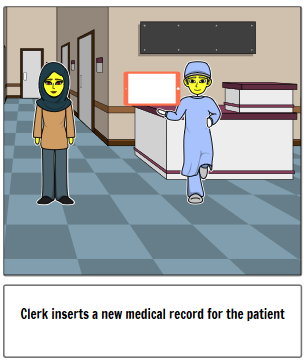
1. Health, Medical Record Organizer
   * Provides intuitive UI.
   * Has a Sign-Up problem.
   * Saves any file type, giving the user 2 GB for free.
   * Copies data to cloud, ensuring that the patient never loses his health records.
2. Doctor At Work – Patient Medical Records
   * A patient electronic medical record, patient appointment tracker, biller.
   * Documents history of examination, diagnosis and treatment of patients.
3. Medical Records App
   * Easily save patient records, patient history.
   * Exports medical data to excel sheet.
   * PDF reports about blood sugar, blood pressure.
4. Medical Records
   * Saves history of visits to the doctors, recommendations, referrals.
   * The application has technical support.
   * Reminder about time of drug taking, keeping history of drugs taken.
   * Registering health rates measurements.
5. GenieMD
   * Tracks, stores, shares your health profile.
   * Medication Tracker.
   * Tracking Vitals, from blood pressure to glucose levels.
   * Fitness Aid, keeps track of exercise routines added by the user.
   * Emergency Contacts, contact friends and family when most needed.

## 1.9 Storyboard









## 1.10 Stakeholders

* Doctors
* Medical Facility Administrator
* Patients
* Pharmacist
* Medical Clerk

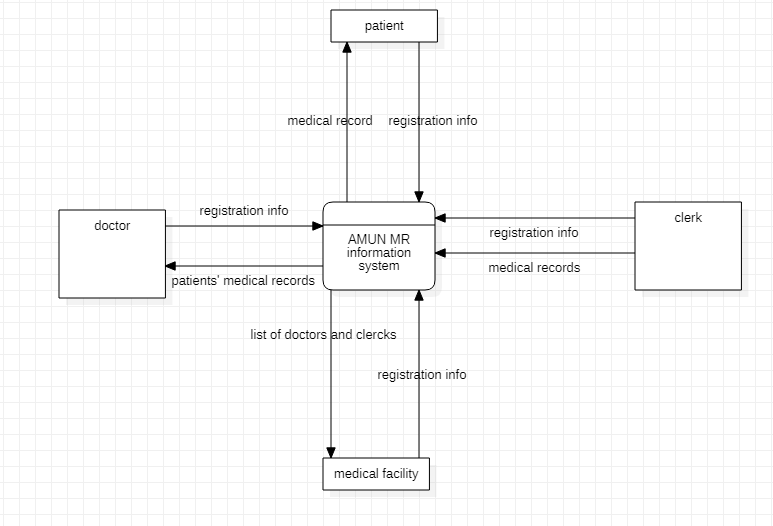
# Chapter 2: Requirements Analysis

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. Requirements analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users. In this chapter, the requirements analysis of the system is explained in full details. This chapter describes the overview of the system architecture. The system architecture is explained in diagrams; use case models, context models, class diagrams, entity relationship diagrams and sequence diagrams. Requirements analysis provide a checklist of requirements, provide a high-level description of the proposed system.

## 2.1 Context Model

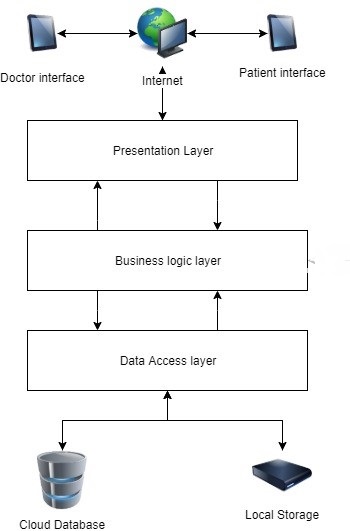
A context model defines how context data are structured and maintained, it aims to produce a formal or semi-formal description of the context information that is present in a context-aware system. Context diagrams shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities. Context models shows the scope and boundaries of a system to stakeholders or developers of the system.

### 2.1.1 Context Diagram



### 2.1.2 System Architecture Diagram

System architecture diagram is used to clarify and communicate ideas about the system structure.



## 2.2 Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Function name | Function description |
| P.1 | Register | Patients registers their information in the system and enter their personal information. |
| P.2 | Add patient medical information | Patients should be able to add their medical information such as blood pressure, blood types, weight and other information. |
| P.3 | Update information | Patients updates their personal information that are already added. |
| P.4 | View record | Patients view their medical records and history that are entered by clerks along with all details such as the exact time the record was created and the corresponding doctor and the medical facility he visited. |
| P.5 | View doctors | Patient may view doctors that they are connected with or created a link with and their contact information. |
| P.6 | Create Link with Doctor | Patients can create a link with a certain doctor.  By creating a link, the doctor is added to the patients’ doctor list.  The connection can be created by scanning a QR code from the patients’ smartphone. |
| P.7 | Delete account | Patients have the option to delete their account along with it, their medical record permanently this action cannot be undone. |
| P.8 | View medical facilities | Patients may view the medical facilities they scanned, and also may view doctors by visiting the medical facility’s page. |
| P.9 | Delete link with doctor | Patients may delete link with their doctor thus making doctors unable to view or update the patients’ medical record. |
| P.10 | Add Medical Record | Patients may add medical records to his profile. |
| P.11 | Add Blood Pressure and Blood Glucose | Patients may add blood tests which can be done in pharmacies. |
| D.1 | Register | Doctors register their information in the system.  The system will always require a verification to prove doctors’ identity. |
| D.1.1 | Add personal information | Doctors add their personal information such as name, age, profession and contact details. |
| D.2 | View Records | Doctors may view the medical record and history of patients that they are connected with. |
| D.2.1 | View Patient Records | View a certain patient’s record. |
| D.3 | Delete Account | Doctors may delete their account permanently from the system. This action cannot be undone. |
| D.4 | View Medical Facilities | Doctors may view all the medical facilities they are a member of. |
| C.1 | Add to patient’s record | Clerks may add the medical record and history of a patient. When a record is added the corresponding patient will be notified immediately. |
| M.1 | Register | Medical facilities may register in the system. |
| M.1.1 | Add medical facility information | Medical facilities should add their information such as its address, type (clinic, hospital or a laboratory) along with other important information, and an ID is automatically assigned to them, generating a unique QR code for each medical facility. |
| M.2 | Add doctor | Medical facilities can add doctors thus making them a member of this facility and of course with the knowledge with doctor and his consent. |
| M.3 | Add clerks | Medical facilities are able to add clerks thus making them members of this facility. Adding a clerk will require the clerk information, an ID and password is assigned by the medical facility administrator. |
| M.4 | View members | Medical facilities are able to view their members (clerk or doctor). |
| M.4.1 | View doctors | Medical facilities are able to view the doctors that are members of this facility. |
| M.4.2 | View clerks | Medical facilities are able to view the clerks that are members of this facility |
| M.4.3 | View Patients | Medical facilities are able to view the patients that are connected with it through the doctors that are members of this facility. |
| M.5 | Delete facility | Medical facilities may delete their accounts from the system. This action will also delete all the clerks that are members of this facility and this action cannot be undone. |
| M.6 | Remove Doctor | Medical facilities are able to remove the doctors that are members of the facility thus making them no longer members. |
| M.7 | Remove clerk | Medical facilities are able to remove a clerk that is a member of the facility thus deleting the corresponding ID and password. |

## 2.3 Non-Functional Requirements

### 2.3.1 Usability

Users are able to use the system and website easily as the interface is friendly which is easy to deal as it includes many functionalities represented in a few buttons.

### 2.3.2 Reliability

Robustness: If one of the website's pages has a bug or a problem while using it, it will not affect the other website pages and they will work, same with the application, if it has a problem it will not affect the website.

Safety: Each competent authority has its own application and the administrator is responsible for it, and no one can access it without his permission.

### 2.3.3 Security

User’s data is stored in a table different than other tables, so no one can see user’s personal information, only the user, even in medical records only the medical information is available using API requests.

Password Requires a minimum length, and a special character.

### 2.3.4 Performance

Response time: Website response is as soon as possible to the citizen’s problem and serves him/her during the estimated time.

Throughput: Web application will serve the user during estimated time that been calculated since the problem happened.

### 2.3.5 Availability

The website for medical facilities and clerks is available 24/7.

## 2.4 Use Case Models

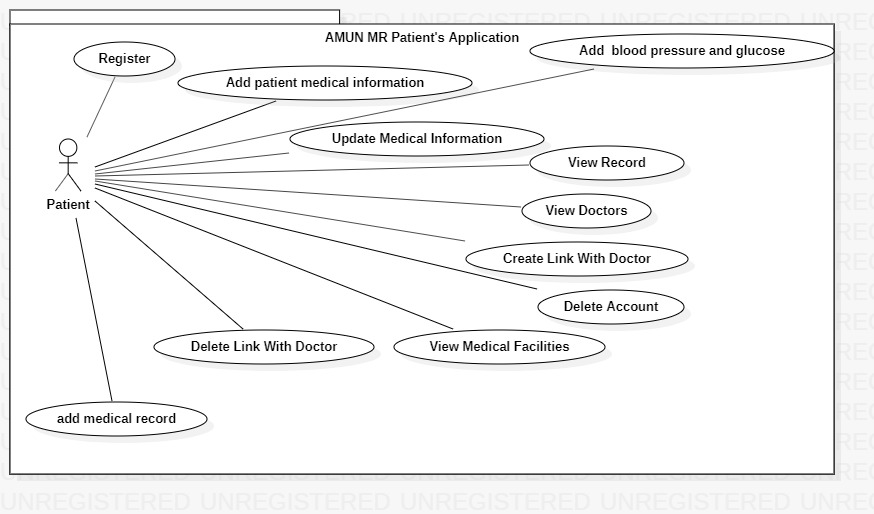
In this section, the use case model is shown to describe the interactions between actors and the system, use case is a list of actions or event steps typically defining the interactions between an actor and a system to achieve a goal. The actor can be a human or other external system.

### 2.4.1 Actors

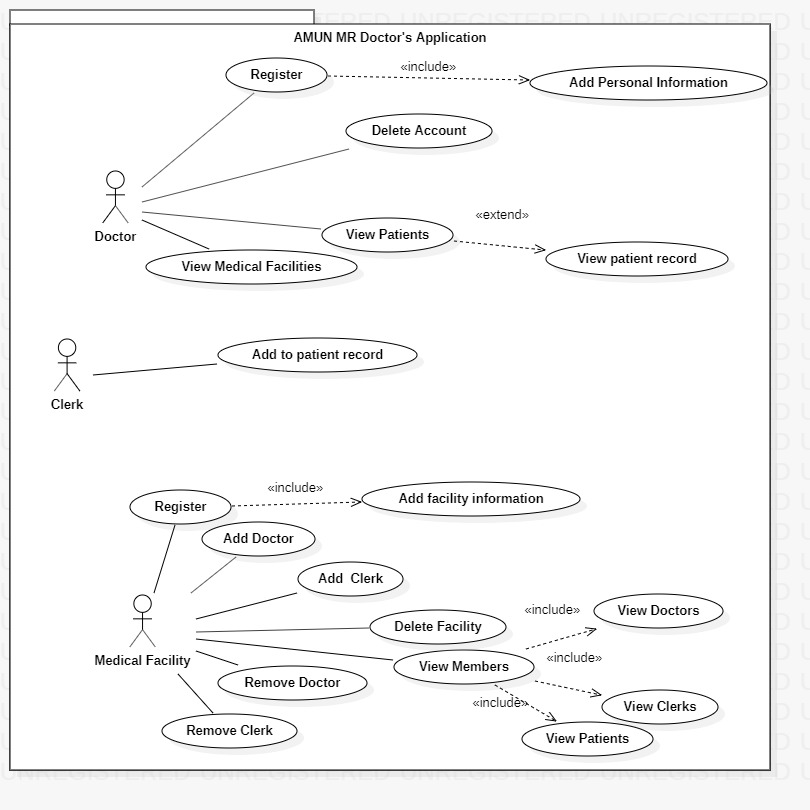
* Patient
* Doctor
* Clerk
* Medical Facility

### 2.4.2 Use Case Diagrams

#### Patient Use Case Diagram



#### Doctor-Medical Facility-Clerk Use Case Diagram



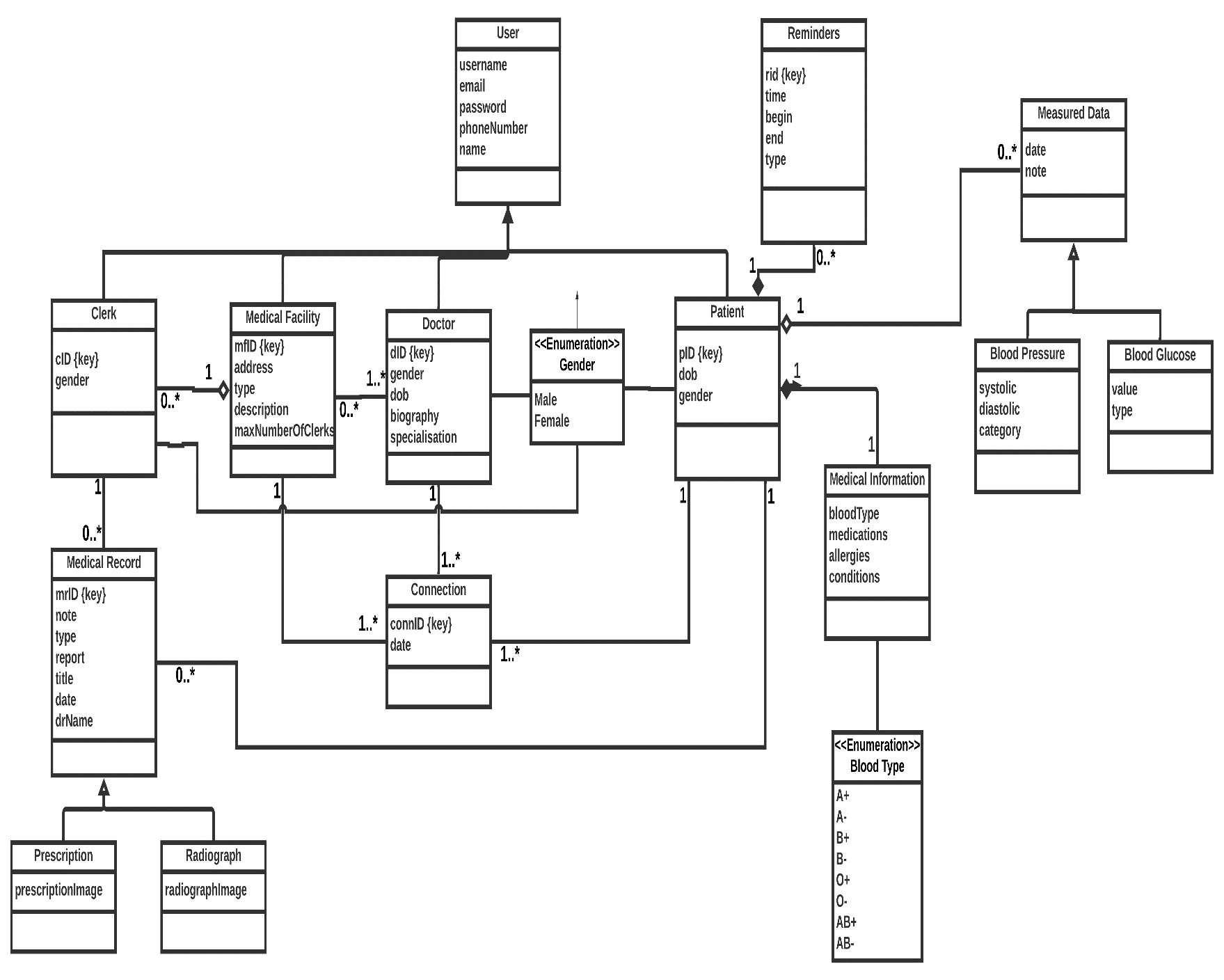
### 2.4.3 Use Case Description

A use case description consists of tables that; each table contains an ID for the use case, use case name, actors, pre-conditions, post-conditions, flow of events and exceptions. This table is available in appendix A.

## 2.5 Domain Model

A domain model is a system of abstractions that describes selected aspects of a sphere of knowledge, influence or activity. The model can then be used to solve problems related to that domain. The domain model is a representation of meaningful real-world concepts pertinent to the domain that need to be modeled in software. Domain models facilitate object-oriented design to represent application’s business objects.

### 2.5.1 Domain Model Diagram



### 2.5.2 Domain Model Description

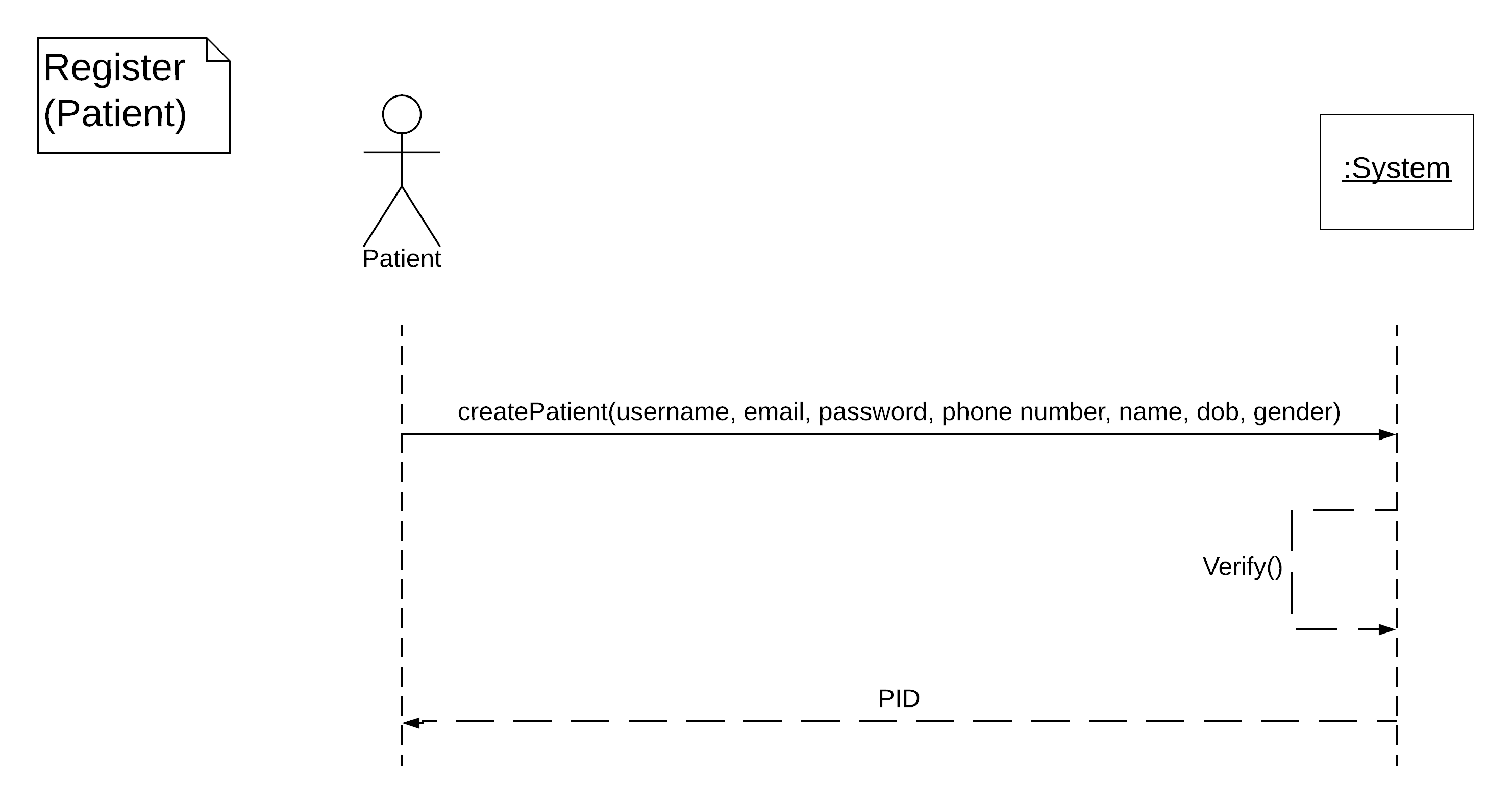
|  |  |
| --- | --- |
| Class Name | Description |
| User | A super class that encapsulates user’s data. |
| Doctor | A sub class of User that encapsulates doctor’s data. |
| Clerk | A sub class of user that encapsulates clerk’s data, also this class is aggregated by medical facility. |
| Medical Facility | A sub class of user that encapsulates medical facilities’ data, has a list of doctors and patients connected to it. |
| Patient | A sub class of user that encapsulates patient’s data. |
| Reminders | A class that adds reminders to patient’s profile, aggregated by patient. |
| Measured Data | A class that adds measured data to patient’s profile, measured data are tests that can be done outside a medical facility. |
| Blood Pressure | This class is responsible for capturing patient’s blood pressure and storing it in patient’s profile. |
| Blood Glucose | This class is responsible for capturing patient’s blood glucose and storing it in patient’s profile. |
| Medical Information | This class encapsulates patient’s medical data such as blood type, list of medications, list of allergies and personal conditions. |
| Medical Record | A class that encapsulates medical records data, medical records can only be added by clerks then stored at patient’s profile. |
| Prescription | A sub class of medical record. |
| Radiograph | A sub class of medical record. |
| Connection | A class that connects between medical facility, doctor, patient. |

## 2.6 Sequence Models

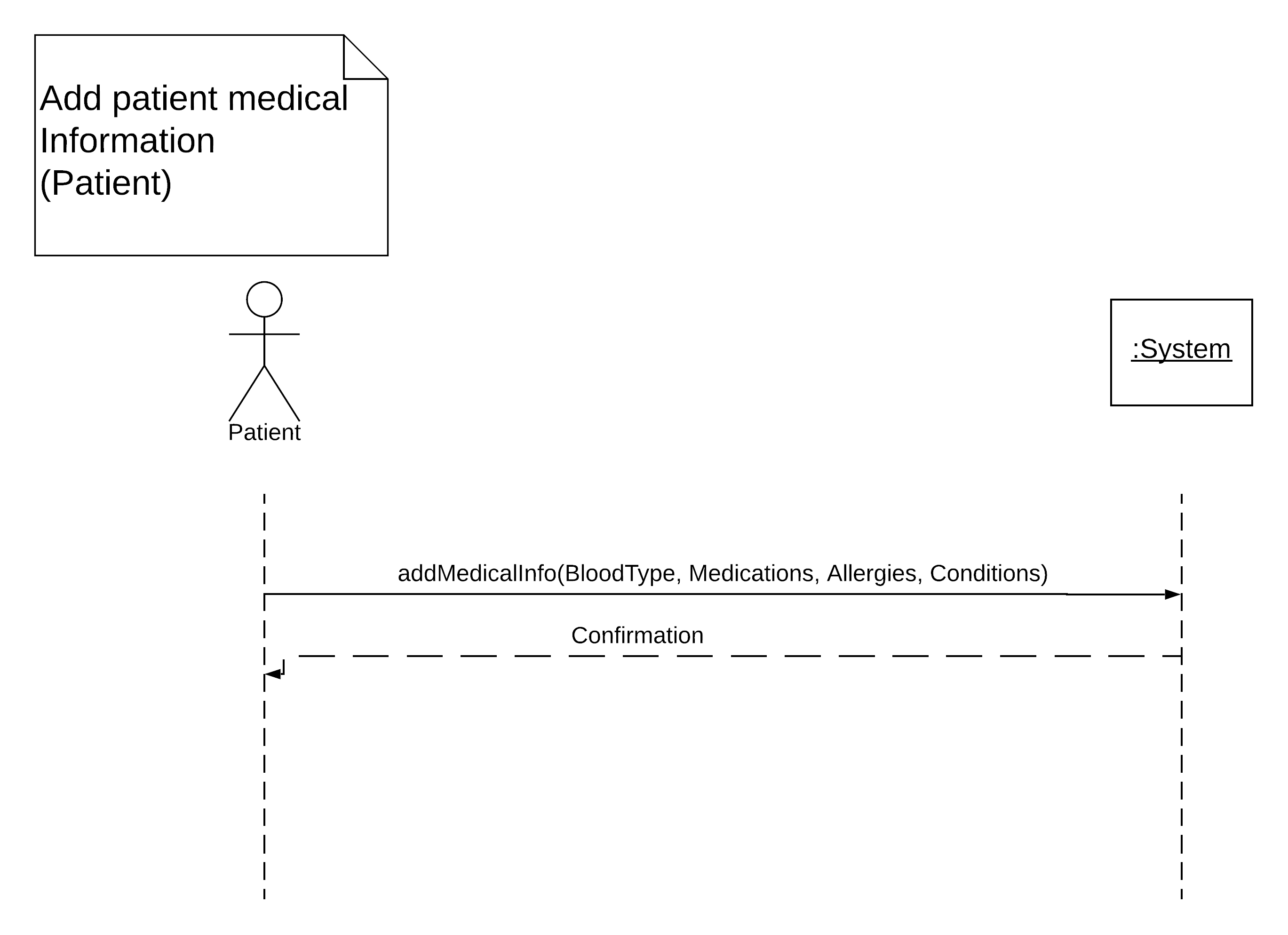
Sequence diagram simply depicts interaction between objects in a sequential order. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems. It captures the interaction that take place in a collaboration that either realizes a use case or an operation. Sequence diagrams are easy to read and also allows reverse engineering.

### 2.6.1 Patient Sequence Diagrams

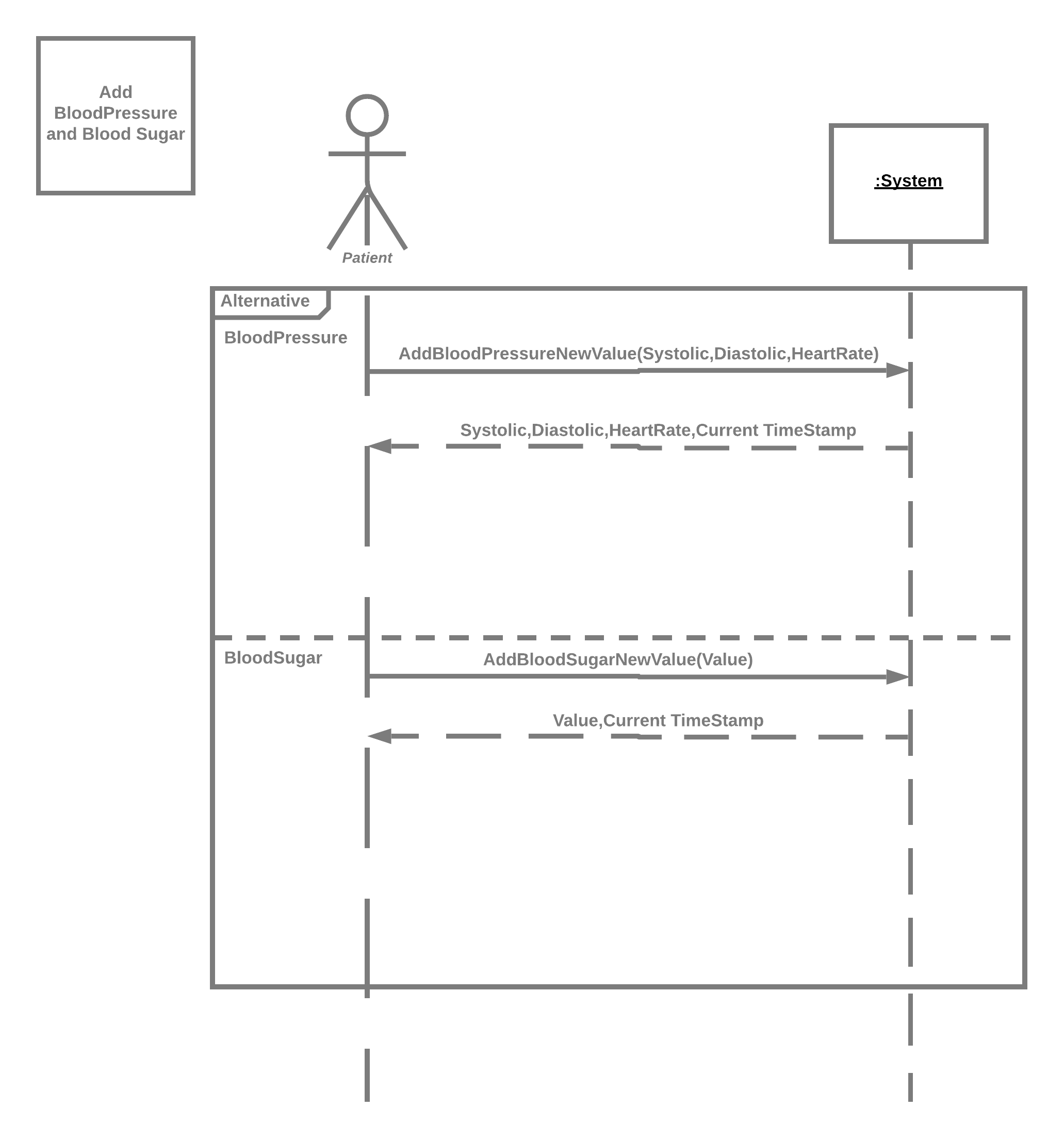
* Register



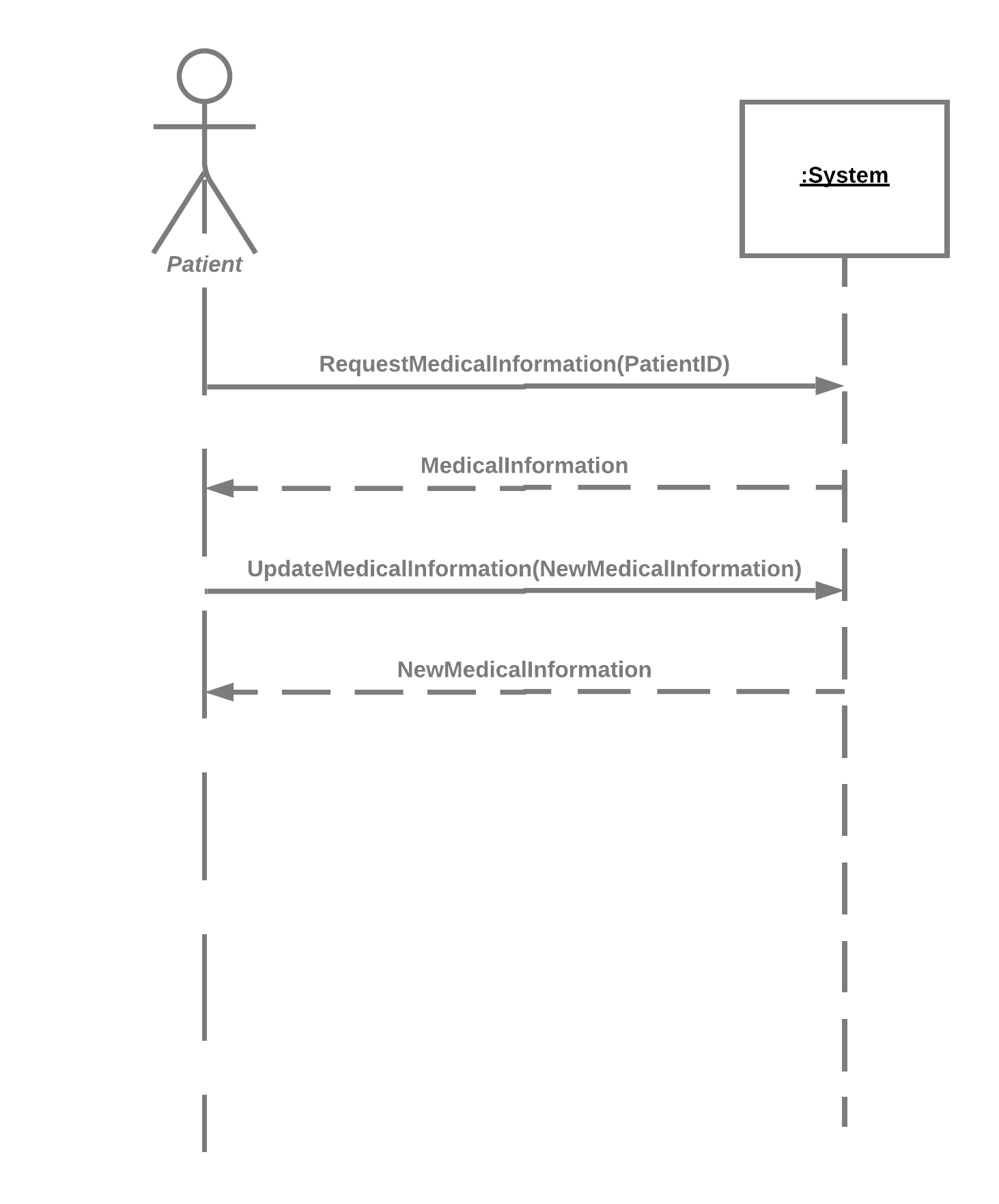
* Add Patient Medical Information



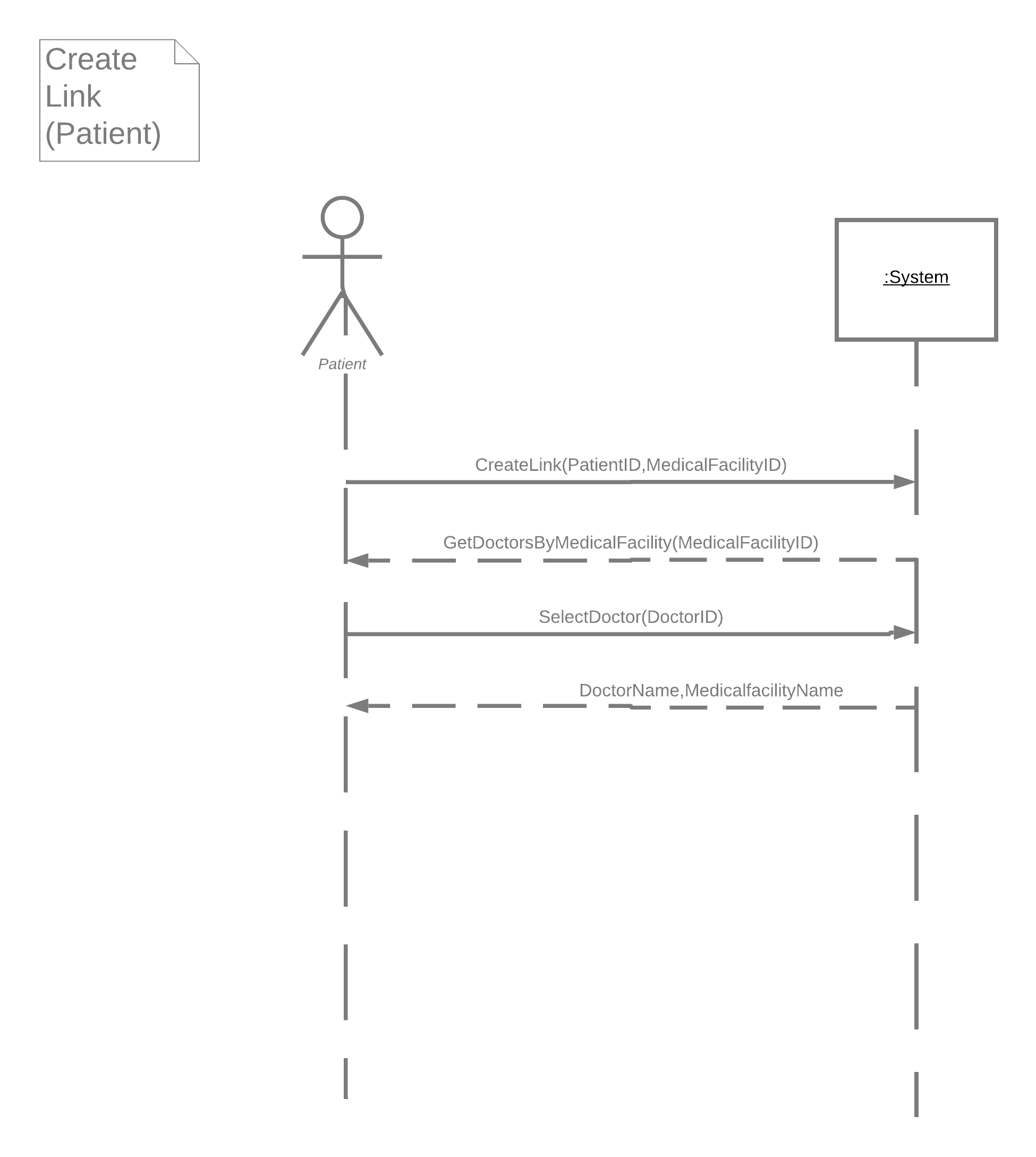
* Add Blood Tests



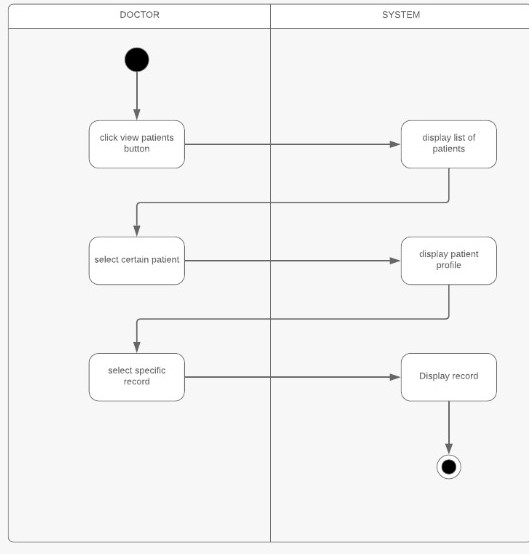
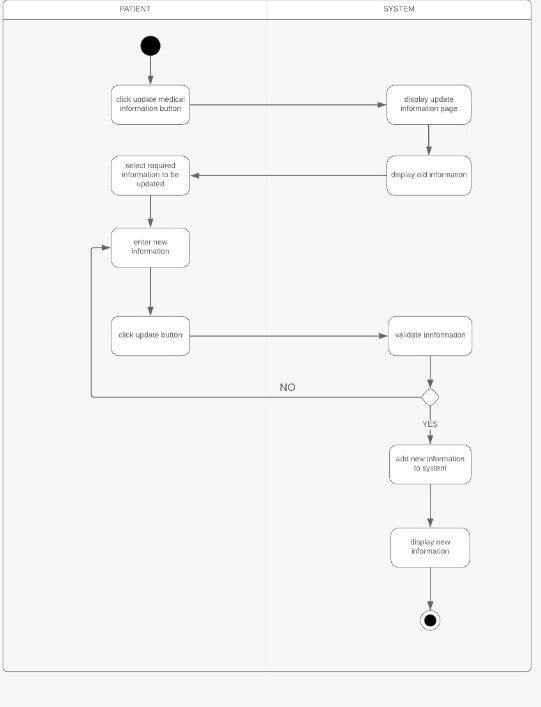
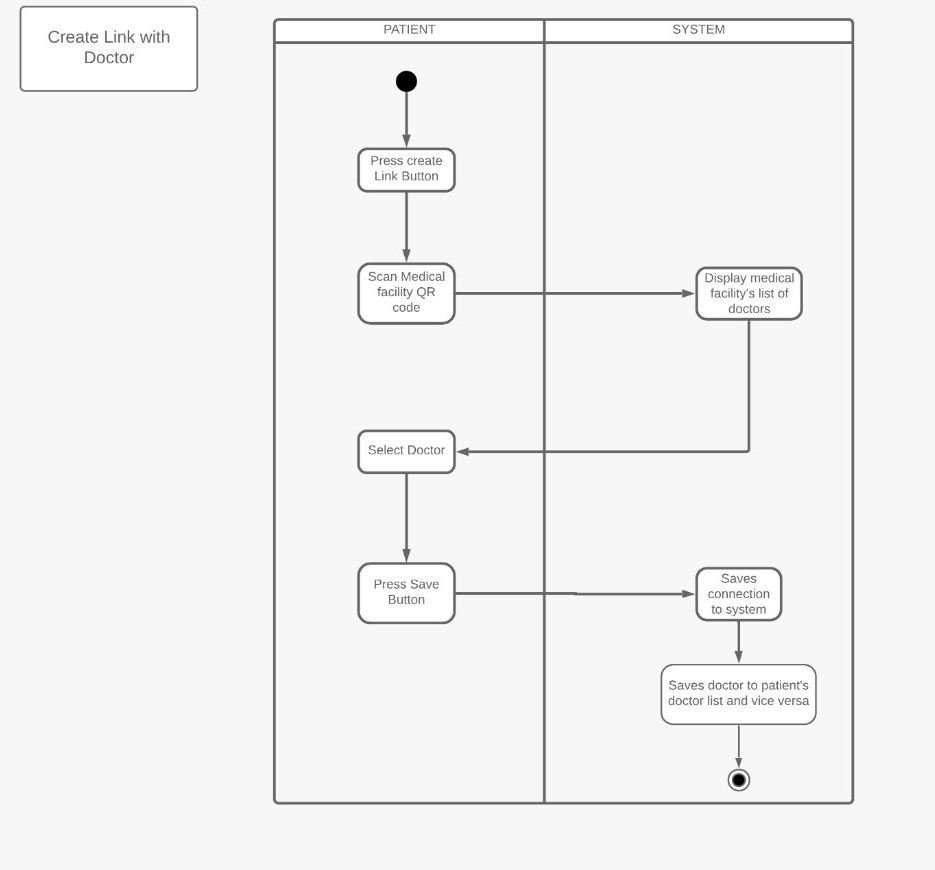
* Update Medical Information



* Create Link

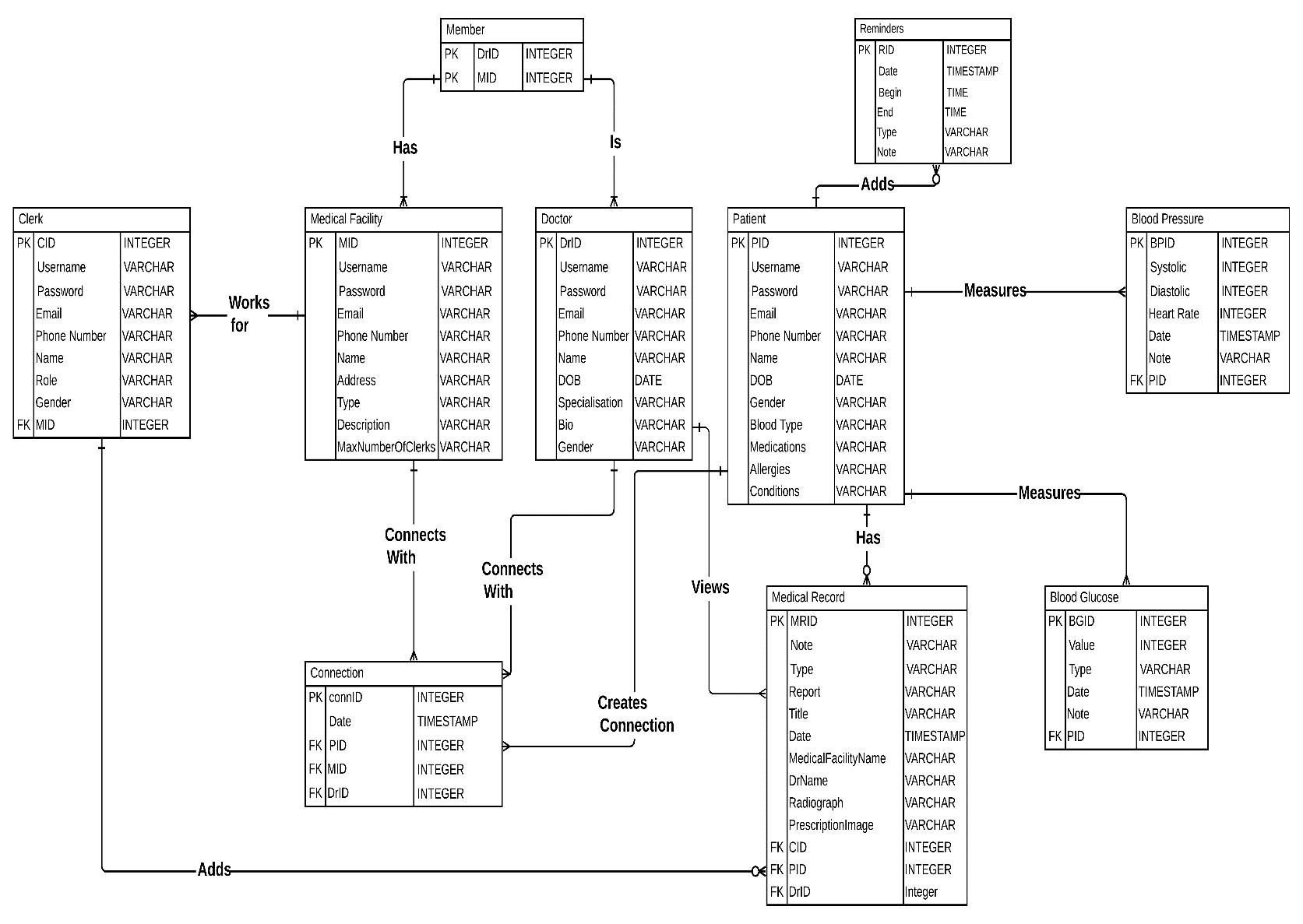


### 2.6.2 Patient Activity Diagram



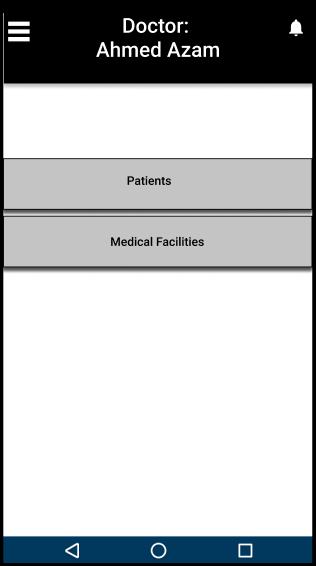
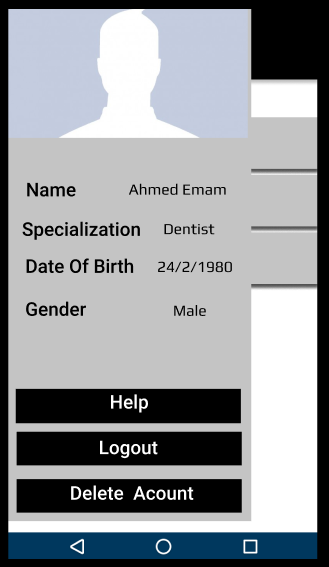
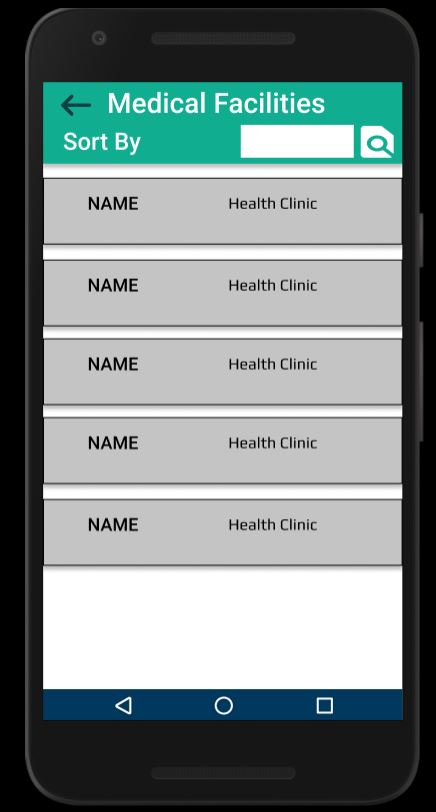
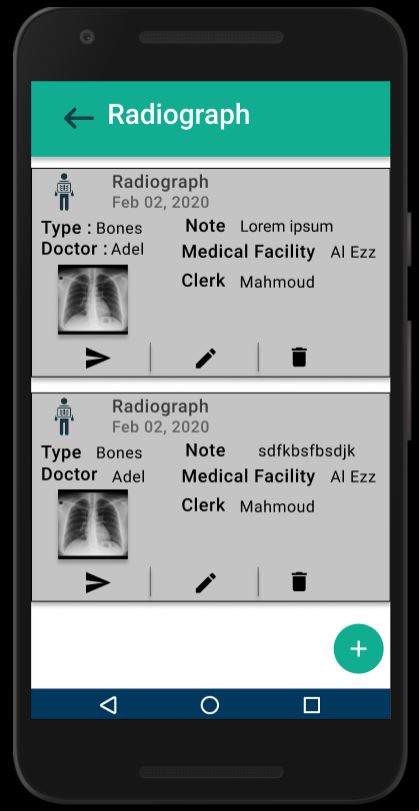
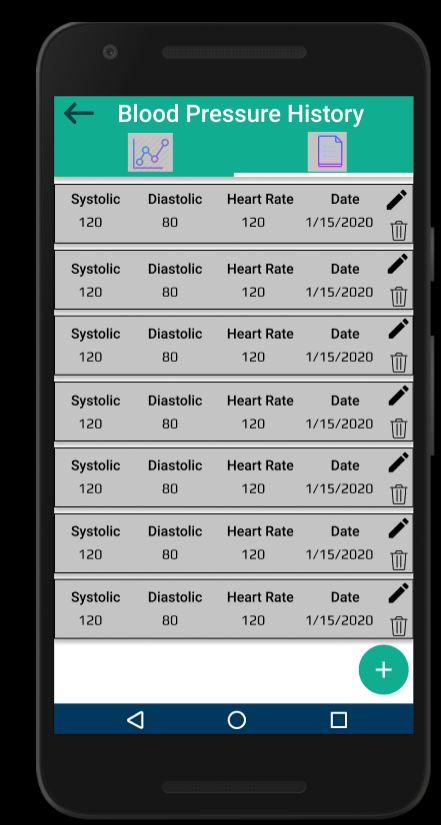
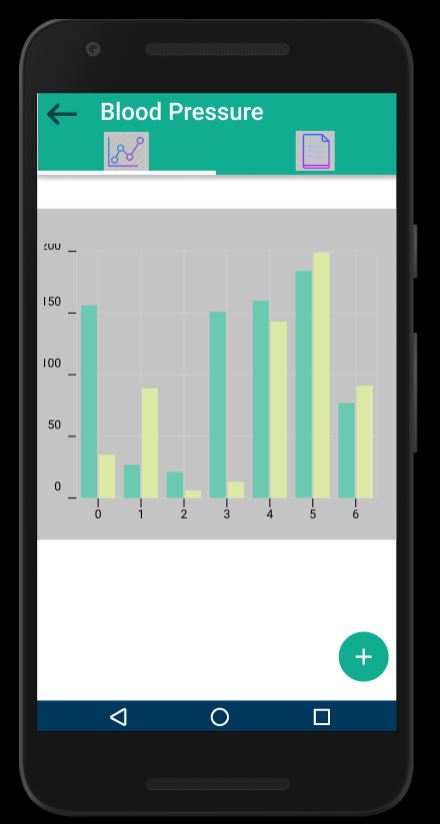
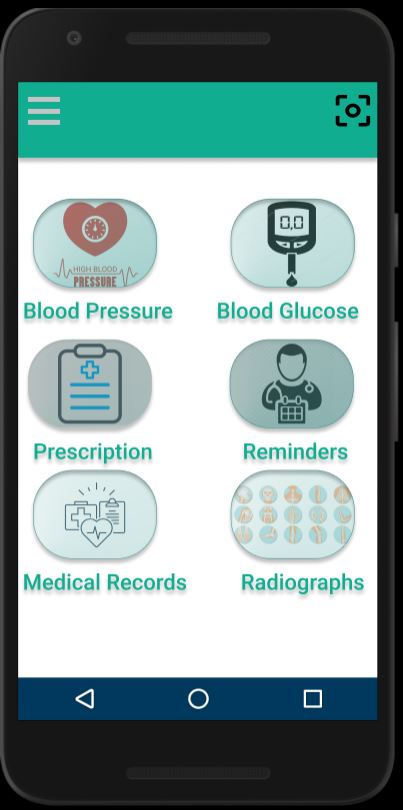
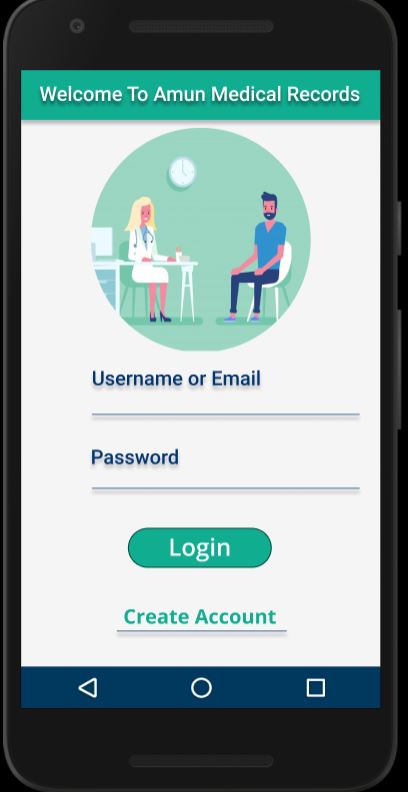
## 2.7 Entity Relationship Diagram

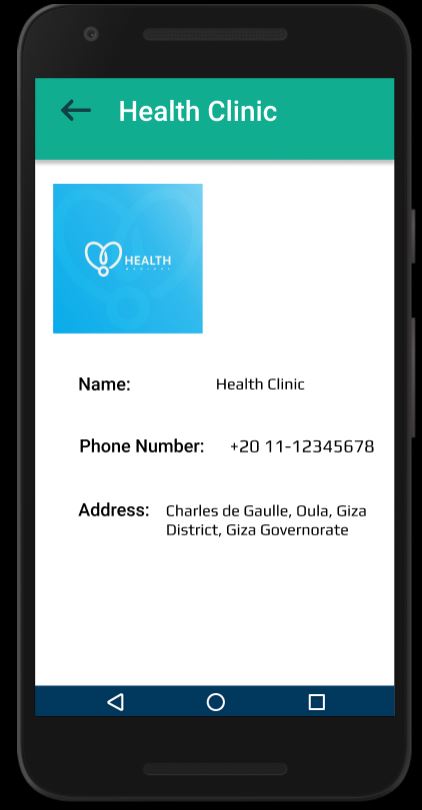
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used to sketch out the design of a database. ER model is very simple because if we know relationship between entities and attributes. It is an effective communication tool for database designer.



## 2.8 Prototype

Prototyping is the first phase in which you can actually interact with your creation, even if only slightly. Prototypes allow you to explore the UI, pinpointing which elements work best, and predicting usability problems before they become problems.





# Chapter 3: Design

In this chapter, A transition is made from conceptual models to physical models, we will describe how we implemented the application and the tools used to guide developers in this project.

## 3.1 Tools Used

In this section, a description of each tool used to implement this project.

### 3.1.1 MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program. MongoDB stores data in flexible, JSON-like documents, indexing, and real time aggregation provide powerful ways to access and analyze data. It was used in this project to create the database system that is used across the application.

### 3.1.2 Firebase

Firebase is a mobile and web application computing and development platform developed by Google. It is used in this project to store files in the database such as radiographs.

### 3.1.3 Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android and iOS from a single codebase, it was used to develop this project’s mobile application.

### 3.1.4 NodeJS

NodeJS is an open-source, cross-platform, JavaScript runtime environment (Framework) that executes JavaScript code outside a web browser, NodeJS allows the creation of web servers using JavaScript.

### 3.1.5 Postman

Postman is a collaboration platform for API (Application Programming Interface) development. Postman's features simplify each step of building an API and streamline collaboration so you a better API can be created. It was used in this project to test API’s requests.

### Vue.js

Vue.js is an open-source model–view–viewmodel JavaScript framework for building user interfaces. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages. It was used in this project to create the session and send requests as POST, GET and PATCH, a library called axios was used to send requests to the API.

### 3.1.7 GitHub

GitHub provides hosting for software development and version control using Git. In this project, GitHub was used to help the team collaborate, organize and improve the applications’ code.

## 3.2 Design Validation and Verification

Design validation is a testing process by which the system is proven that it works for the end user as intended, while design verification is where design is tested to show if the system’s outputs match design inputs. Based on our requirement analysis it has shown that there is a difficulty in interaction between doctors and patients when it comes to explaining the patient’s medical problems, also carrying past medical records may become a burden if there are many medical visits. In the proposed system’s design, we solved many of the issues to tackle this problem.

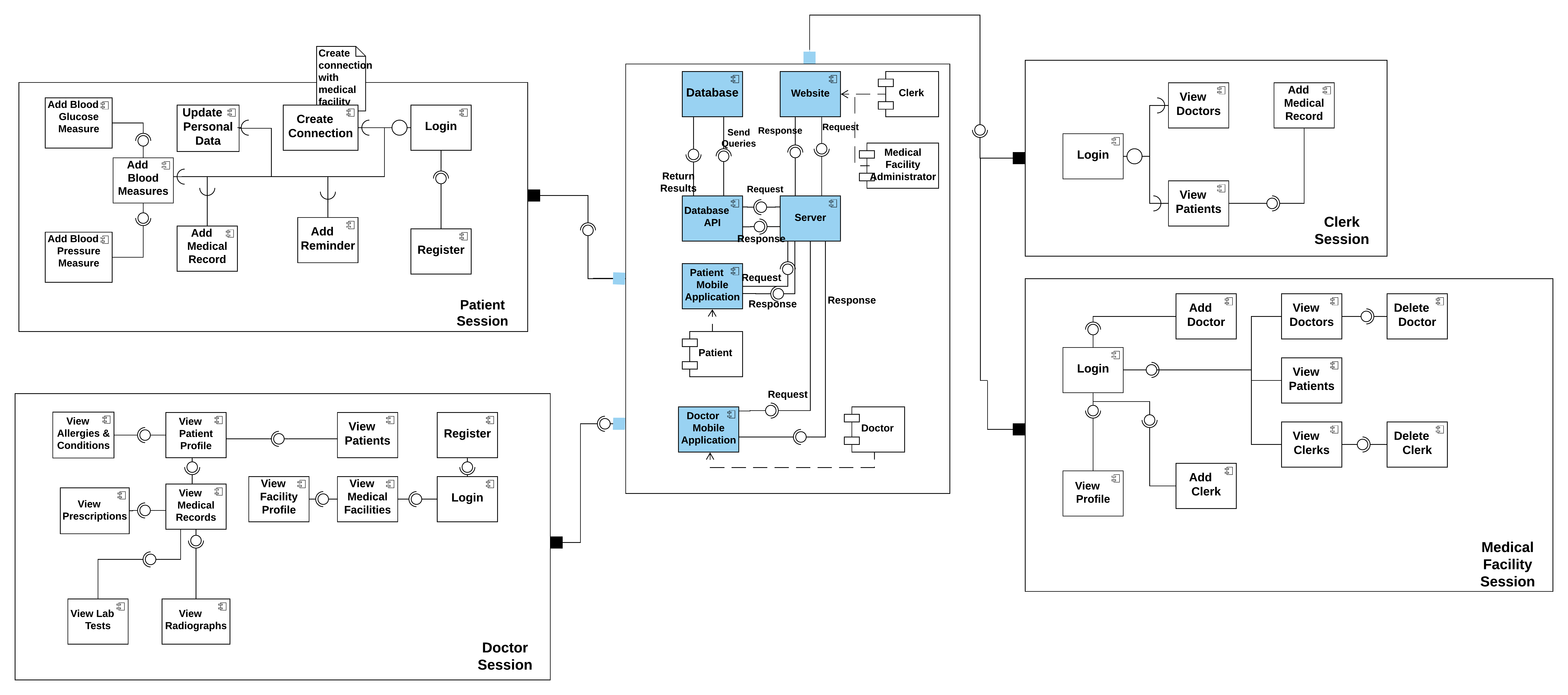
### 3.2.1 Design Validation

|  |  |
| --- | --- |
| **User’s Requirements** | **Validation Test** |
| * Patient wishes to show a doctor his past medical issues. | * System’s design proposes a connection system between patients, doctors and medical facilities. |
| * Patient decided not to see this doctor. | * Patient may delete connection with doctor to hide his medical information from him. |
| * Patient wishes to add to his medical records but has no connection. | * System provides an option where medical record can be inserted in patient’s profile, which will be added when patient has internet connection |
| * Patient wishes to view doctors that he is connected with. | * Patient’s application provides a view doctors button which views all doctors this patient had visited. |
| * Doctor wants to view a patient’s medical profile even after the visit. | * If the patient didn’t delete the connection, the system views all patient’s this doctor is connected with. |
| * A doctor doesn’t have time to add medical records. | * The system provides an option for clerks to add medical records instead of doctors, also the patient may add the medical record. |

### 3.2.2 Design Verification

|  |  |
| --- | --- |
| **Product Requirement** | **Verification Test** |
| * Clerk/Medical Facility Administrator tries to access a page without logging in. | * System doesn’t allow accessing its website without a valid session. |
| * Patient may want to edit his profile. | * System allows edit for a patient’s profile, except for some fields as name, email and username. |
| * Medical facility administrator inserted wrong clerk’s information. | * System prevents edit for clerks, administrator should delete clerk then try to insert again. |
| * Medical facility administrator wishes to remove a doctor that isn’t working in this facility anymore. | * System provides the option view current facility doctors and the option to delete his profile from this facility. |
| * Patient wishes to view current medications. | * System provides a list of medications which can be edited freely by patients. |

## 3.4 Component Diagram

Component diagram is a special kind of diagram in UML, it describes the components used to make those functionalities.

# Chapter 4: Testing

## 4.1 Test Plan

### 4.1.1 Interface Testing

Interface Testing is a kind of software testing which verifies the communication between two different software systems. It checks the authentication of the connection established. Interface testing is needed due to the following reasons:

1. To ensure that the end-users or customers should not encounter any problem while using a particular software.
2. To identify which application areas are usually accessed by end-users and to check if they are user-friendly.
3. To verify the security requirements while communication propagates between the systems
4. To check if a solution is capable of handling network failures between an application server and website
5. To make sure that the software is proficient in error handling.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Status** | **Comments** | **Users** |
| Accessing a page without logging in. | Failed | While testing the web application we found that the session wasn’t working so we added a function to check for session if it is available, defect was solved. | * Clerk * Medical Facility Administrator |
| Generating QR code for medical facility | Success | Medical facility administrator may generate a unique QR code for his facility using the assigned id by converting the string to QR code. | * Medical Facility Administrator |
| Signing in with an invalid username or password to check the error message | Success | Error message appears as an alert for the user. | * Clerk * Medical Facility Administrator |
| Adding a medical record with an image to the API | Failed | Image was being uploaded from user’s system to the website correctly, but the POST request had to be changed to accept images.  Defect Solved | * Clerk |
| Creating an account on patient’s application | Failed | Email maximum length was set to 15 characters, which may not serve all users, email’s maximum length was increased to solve this defect. | * Patient |
| Edit Information in doctor’s application | Failed | The button had a non-existing route so it caused an error, it was solved by sending the right route.  Defect Solved | * Doctor |

### 4.1.2 API Testing

API testing is a software testing type that validates Application Programming Interfaces (APIs). The purpose of API Testing is to check the functionality, reliability, performance, and security of the programming interfaces. In API Testing, instead of using standard user inputs(keyboard) and outputs, you use software to send calls to the API, get output, and note down the system's response.

#### Requirements:

* NodeJS
* MongoDB

#### Request Types:

* Post
* Get
* Patch
* Delete

#### Requests:

All responses are in JSON (JavaScript Object Notation) format, it contains in most of the requests; status which is true or false, if the request is true it is denoted by response code:200 OK, else it prints an error message which contains the reason why a request was refused.

* **Patient Requests**

1. *Patient Registration*

**URL:** {domainname}/patients

**Method:** POST

**Parameters:** email, firstName, lastName, password, username, mobile, birthDate, gender.

**Success:** returns newly created user and assigns id for him/her*. (Code: 200)*

**Error:** email/username isn’t available, missing parameters. *(Error Code: 400)*

1. *Patient Login*

**URL:** {domainname}/patients/auth

**Method:** POST

**Parameters:** email/username, password

**Success:** returns user’s information and session’s token. *(Code: 200)*

**Error:** Invalid username/email or password

*(Error Code:400)*

1. *Add Facility Patient connection*

**URL:** {domainname}/facilities-patients

**Method:** POST

**Parameters:** patientId, medicalFacilityId, doctorId

**Success:** returns id of the connection between the parameters, patientId, medicalFacilityId, doctorId and date in which the connection was created. *(Code: 200)*

**Error:** Missing Parameter *(Error Code: 400)*

using wrong authorization token. *(Error Code:403)*

1. *Search Doctor*

**URL:**

**Method:** GET

**Parameters:**

**Success:** returns doctor’s id that the user searched for. *(Code: 200)*

**Error:** this doctor doesn’t exist. *(Error Code: 400)*

1. *View Doctors*

**URL:** {domainname}/facilities-patients/doctor

**Method:** GET

**Parameters:** -

**Success:** returns a list that contains; id of the connection between doctor and patient and doctor’s information *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

1. *Get Profile Data*

**URL:** {domainname}/patients/

**Method:** GET

**Parameters:** -

**Success:** returns patient’s profile which contains medical records, tests added, and credentials. *(Code: 200)*

**Error:** Forbidden Access *(Error Code:403)*

1. *Update Data*

**URL**: {domainname}/patients/

**Method:** PATCH

**Parameters:** mobile, birthDate, password, allergies, conditions, medications, bloodType, gender

**Success:** returns updated user data. *(Code: 200)*

**Error:** updates forbidden data such as email, firstName, lastName, username *(Error Code: 400)*

1. *Add Blood glucose measure*

**URL:** {domainname}\blood-glucose\

**Method:** POST

**Parameters:** value, type, note, date(ISO 8601 format)

**Success:** returns id of the document, parameters inserted, createdAt, patientId *(Code: 200)*

**Error:** date format is wrong. *(Error Code: 400)*

1. *Get Blood glucose measure*

**URL:** {domainname}\blood-glucose\

**Method:** GET

**Parameters:** -

**Success:** returns a list of all blood glucose measures done by this patient. *(Code: 200)*

**Error:** Forbidden Access *(Code: 403)*

1. *Add Blood pressure measure*

**URL:** {domainname}/blood-pressure/

**Method:** POST

**Parameters:**  an array that contains systolic and diastolic, date, note.

**Success:** id of the document, patientId, date created. *(Code: 200)*

**Error:** missing parameters, wrong date format. *(Error Code: 400)*

1. *Get Blood pressure measure*

**URL:** {domainname}/blood-pressure/

**Method:** GET

**Parameters:** -

**Success:** returns a list of all blood pressure measures done by this patient. *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

* **Doctor Requests**

1. *Doctor Registration*

**URL:** {domainname}/doctors

**Method:** POST

**Parameters:** firstName, lastName, username, password, email, mobile, birthDate, gender, specialization, bio

**Success:** generates doctorId, and returns parameters entered by doctor. *(Code: 200)*

**Error:** email/username isn’t available, doctor didn’t enter required parameters (bio is optional), password is less than 8 chars. *(Error Code: 400)*

1. *Doctor Login*

**URL:** {domainname}/doctors/auth

**Method:** GET

**Parameters:** username/email and password

**Success:** returns doctor’s data and session’s token.

(*Code: 200)*

**Error:** Invalid username/email or password

*(Error Code: 400)*

1. *View Patients Profiles*

**URL:** {domainname}/patients/

**Method:** GET

**Parameters:** -

**Success:** returns a list of all patients connected with this doctor, and all their medical records. *(Code: 200)*

**Error:** Wrong authorization token *(Error Code: 403)*

1. *Update Doctor’s Data*

**URL:** {domainname}/doctors/:id

**Method:** PATCH

**Parameters:** password, birthDate, bio, specialization, gender

**Success:** returns updated doctor’s data. *(Code: 200)*

**Error:** tried to update forbidden data such as username, email, firstName, lastName. *(Error Code: 403)*

1. *View Facilities*

**URL:** {domainname}/medical-facilities

**Method:** GET

**Parameters:** -

**Success:** returns a list of all facilities connected with this doctor *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

* **Facility and Clerk Requests**

1. *Facility Registration*

**URL:** {domainname}/medical-facilities

**Method:** POST

**Parameters:** name, username, password, email, mobile, address, type, description, maxNumberOfClerks

**Success:** generates medicalFacilityId, and returns parameters entered by facility’s administrator.

*(Code: 200)*

**Error:** missing required parameters (Error Code: 403)

1. *Facility Login*

**URL:** {domainname}/medical-facilities/auth

**Method:** POST

**Parameters:** (email or username), password

**Success:** returns true, medical facility data and session’s key *(Code: 200)*

**Error:** “Invalid Username/Password” *(Error Code: 400)*

1. *Add Doctor*

**URL:** {domainname}/facilities-doctors

**Method:** POST

**Parameters:** doctorId, medicalFacilityId

**Success:** returns id of the connection, doctor’s id medical facility’s id and date the connection was created.

*(Code: 200)*

**Error:** Duplicate, invalid doctor/medical facility ID

*(Error Code: 400)*

1. *Add Clerk*

**URL:** {domainname}/clerks

**Method:** POST

**Parameters:** firstName, lastName, username, password, email, mobile, role, gender

**Success:** returns newly created clerk and assigns id for him/her. *(Code: 200)*

**Error:** missing any of the parameters, unavailable username/email *(Error Code:403)*

1. *View Patients*

**URL:** {domainname}/facilities-patients/patient

**Method:** GET

**Parameters:** -

**Success:** returns a list of all patients who created a connection with this medical facility, id of the connection and the date this connection was created *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

1. *View Clerks*

**URL:** {domainname}/clerks?medicalFacilityId={:id}

**Method:** GET

**Parameters:** -

**Success:** returns a list of all clerks that this medical facility registered. *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

1. *Clerk Login*

**URL:** {domainname}/clerks/auth

**Method:** POST

**Parameters:** email/username, password

**Success:** returns true, clerk’s data and session’s key *(Code: 200)*

**Error:** Invalid Username/email or password

*(Error Code: 400)*

1. *View Doctors*

**URL:** {domainname}/facilities-doctors

**Method:** GET

**Parameters:** -

**Success:** returns a list of all doctors connected with this medical facility and id of the connection between them.

*(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

1. *Delete Doctor*

**URL:** {domainname}/facilities-doctors/{:id}

**Method:** DELETE

**Parameters:** -

**Success:** deletes connection *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

1. *Delete Clerk*

**URL:** {domainname}/clerks/{:id}

**Method:** DELETE

**Parameters:** -

**Success:** deletes connection *(Code: 200)*

**Error:** Forbidden Access *(Error Code: 403)*

* **Medical Record Requests**

1. *Create Medical Record*

**URL:** {domainname}/medical-records

**Method:** POST

**Parameters:** title, notes, date, patientId, doctorId, medicalFacilityId, clerkId, type, radiograph, enteredBy

**Success:** adds medical record to database, adds the radiograph to firebase’s local storage and generates medical record ID. *(Code: 200)*

**Error:** missing any of the required data, adding a duplicate *(Error Code: 400)*

Forbidden Access *(Error Code: 403)*

1. *Delete Medical Record*

**URL:** {domainname}/medical-records/{:id}

**Method:** DELETE

**Parameters:** -

**Success:** delete’s medical record and removes the image from firebase’s local storage.

**Error:** id doesn’t exist *(Error Code: 400)*

Forbidden Access *(Error Code: 403)*

1. *View Medical Record*

**URL:** {domainname}/medical-records

**Method:** GET

**Parameters:** -

**Success:** Returns id of the medical record, created by whom, date it was created at, and the medical record parameters.

**Error:** id doesn’t exist *(Error Code: 400)*

Forbidden Access *(Error Code: 403)*

### 4.1.3 System Testing

#### Functional Testing

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Status** | **Comments** | **Users** |
| Add Medical Record | Success | - | * Patient * Clerk |
| Add Clerk | Success | - | * Medical Facility |
| View Patient’s Profile | Success | This function is available from the doctor’s app. It views patient’s medical information. | * Doctor |
| View Medical Facility Profile | Success | The profile contains medical facility’s information and the generated QR Code. | * Medical Facility Administrator |
| Add Blood Glucose/Pressure Measure | Success | - | * Patient |
| Add Doctor | Success | Medical Facility connects to doctor through doctorID | * Medical Facility Administrator |
| Create Doctor’s Account | Success | - | * Doctor |
| View Clerks | Success | Medical facility administrator may view clerks, that the administrator adds | * Medical Facility Administrator |
| Add Clerk | Success | Medical facility may add clerks up to the available maximum number | * Medical Facility Administrator |
| Delete Clerk | Success | A clerk’s profile can be deleted through the view clerks method as it views all clerks connected to this medical facility. | * Medical Facility Administrator |

#### Non-Functional Testing

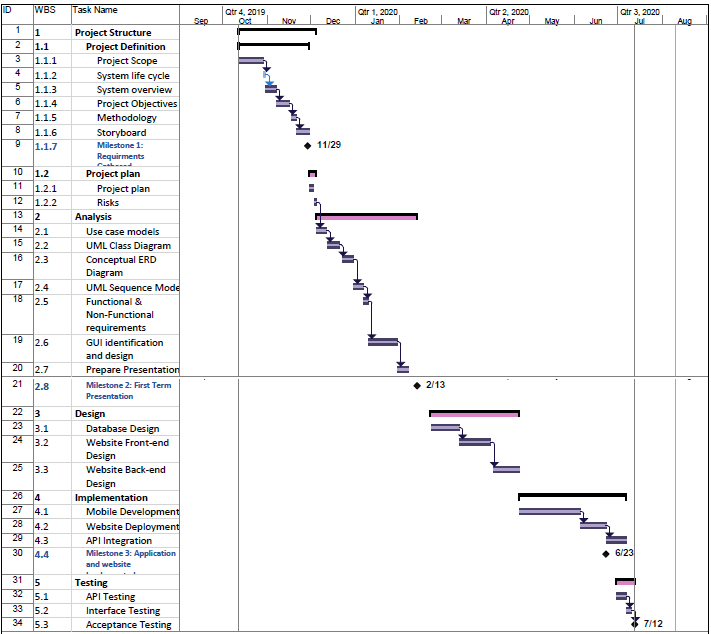
* **Security Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Test Result** | **Comments** | **Users** |
| * Access a page without logging in. | Failed | The system’s website routes users that aren’t logged in to a page informing him that it is forbidden to access it. | * Clerk * Doctor * Medical Facility |
| * Entering Password with more than 8 with a special character | Success | As password needs to contain more than 8 letters and a special character | * Clerk * Patient * Medical Facility * Doctor |
| * Viewing connected patient through doctor’s application | Success | The application views only the connected patients | * Doctor |
| * Viewing connected doctors through patient’s application | Success | The application views only the doctors that the patient relates to. | * Patient |

* **Interoperability Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Test Result** | **Comments** | **Users** |
| Adding a medical record from clerk’s web application and viewing it from patient’s application | Success | As the web application sends a request to the system’s API, if the request is accepted it is uploaded to the database. | * Clerk * Patient |
| Connecting a doctor to a medical facility through the web application, then check the doctor’s application for connected facilities. | Success | - | * Doctor * Medical Facility |
| Deleting a clerk from medical facility, then trying to login with this clerk’s credentials | Failed | Since the clerk is removed, his credentials are removed from the database | * Clerk * Medical Facility |
| Adding a medical record from patient’s application then viewing it from the doctor’s application | Success | as soon as the patient’s insert a medical record data is available for doctor to access it within a few seconds | * Patient * Doctor |
| Removing connection with a doctor through patient’s application, then accessing this patient’s medical record through this doctor. | Failed | after a patient terminates a connection with a specific doctor, this doctor cannot access this patient’s records | * Doctor * Patient |

## 4.2 Planning



## 4.3 Team Roles

|  |  |
| --- | --- |
| Abdelrahman Magdy | * Application Developer * Prototype Creator * Requirements Analyst |
| Omar Mokhtar | * Application Developer * Requirements Analyst * Back-end Developer * Database Designer |
| Ibrahim El Sayed | * Front-End Developer * Prototype Creator * Requirements Analyst * Tester |
| Eslam Hamada | * Requirements Analyst * Tester * Technical Writer * Back-End Developer |
| Ahmed Bassem | * Requirements Analyst * Technical Writer * Tester * Back-End Developer |

# Chapter 5: Evaluation

## 5.1 Project Overview

In summary, two applications were released; one for doctors and the other is for patients, doctors’ application is for viewing patients’ medical records, patients’ application is for updating his medical profile, adding reminders for medications and adding measurements as blood glucose and blood pressure. Also, a web app was deployed for medical facilities and clerks, which helps medical facilities track their patients’ visits, adding clerks and doctors to their database.

## 5.2 Approach Followed

The main idea of this project is to reduce the burden that patients’ carry with them while going to a doctor’s visit, as patient’s may carry a lot of radiographs and other lab tests which may be damaged or lost, the application makes it easier for patient’s to save their data on a cloud based system.

The methodology used for this project was waterfall method as it eased for the team to understand milestones and when milestones where reached, and also helped in documenting results easily.

## 5.3 Possible future work

1. A new application for clerks may be developed, so that they can add medical records to a visiting patient’s profile from a mobile phone or a computer.
2. After collecting enough data from patients, some analysis may be performed based on age and other measurements such as blood pressure by age, which couldn’t be calculated at this time due to insufficient data.
3. A feature may be added to locate nearby medical facilities by patients’ location in addition to doctors’ availability.
4. A feature may be added for patient’s application to view doctor’s available at medical facilities and doctors time schedule.

## 5.4 Achievements Made

During this project, a web application was created using one of the modern tools which is vue.js and more advanced tools to create an easy user interface for users. Also, two applications were created using flutter which is a new software development tool. The application achieved its goals by making it easier for the patients to deal with doctors easily and reduced the problem of losing medical records and historical prescriptions.

## 5.5 Lessons Learnt

During this project, our team learnt;

1. How to create a plan and distribute tasks upon the team to optimize time usage.
2. Technical writing that helped writing this documentation and communicating with our supervisors.
3. We also developed new programming skills as flutter, vue.js, node.js
4. How to create HTTP requests and develop a RESTful API
5. To collaborate through GitHub in which we learned how to use Git commands.
6. How to distribute tasks and verify tasks by another team member.

# References

1. ***Perforce.* [Online] https://www.perforce.com/blog/alm/design-verification-validation-medical-device.**

**2. John W. Satzinger, Robert B. Jackson, Stephen D. Burd. *Systems Analysis and Design in a Changing World.* s.l. : CENGAGE Learning, 2015.**

**3. Bruegge, Bernd and H. Dutoit, Allen. *Object-Oriented Software Engineering Using UML, Patterns, and Java.* Munich, Germany : Pearson, 2010. Vol. 3.**

**4. *Figma.* [Online] www.figma.com.**

**5. *Lucid Chart.* [Online] Lucid Software Inc. www.lucidchart.com.**

**6. El Masri, Ramez and Navathe, Shamkant B. *Fundamentals of Database Systems.* s.l. : Addison Wesley, 2011.**

**7. *Stack Overflow.* [Online] www.stackoverflow.com.**

**8. *Creately.* [Online] www.creatly.com.**

**9. *Simple Programmer.* [Online] www.simpleprogrammer.com.**

**10. *Science Direct.* [Online] www.sciencedirect.com.**

**11. *AltexSoft.* [Online] www.altexsoft.com.**

**12. *Storyboard That.* [Online] https://www.storyboardthat.com/.**

**13. *Tutorials Point.* [Online] https://www.tutorialspoint.com/.**

**14. *Flutter.* [Online] www.flutter.dev.**

# Appendix

## Appendix A

#### Patient Use cases

|  |  |
| --- | --- |
| Use Case ID | P.1 |
| Use Case Name | Register |
| Actors | Patient |
| Pre-Conditions | * User doesn’t have an account. |
| Post-Conditions | * User’s credentials are stored in database. * User is redirected to the home page. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses register button. | 1. System redirects to register page. | | 1. User enters his information and presses submit. | 1. System checks for validity of the user’s information, and adds it to the database. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User presses register button. | 1. System redirects to register page. | | 1. User enters his information and presses submit. | 1. System checks for availability of the user’s information, and finds it unavailable. | | 1. User tries again till entered information is available. | 1. System confirms and adds it to database. | |

|  |  |
| --- | --- |
| Use Case ID | P.2 |
| Use Case Name | Add patient’s medical information |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * Patient’s medical information is added to the system. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User enters his medical information. | 1. System stores medical information in the user’s profile. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | P.3 |
| Use Case Name | Update Information |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * User’s personal or medical information is updated. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User views his profile, user presses ‘Edit Information’ button. | 1. System views user’s personal information and medical information. | | 1. User edits his information then presses ‘Submit’. | 1. User’s information is updated in the system. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User views his profile, user presses ‘Edit Information’ button. | 1. System views user’s personal information and medical information | | 1. User enters his information and presses submit. | 1. System checks for availability of the user’s information, and finds it unavailable. | | 1. User tries again till entered information is available. | 1. System confirms and updates it in database. | |

|  |  |
| --- | --- |
| Use Case ID | P.4 |
| Use Case Name | View Record |
| Actors | Patient |
| Pre-Conditions | * User is registered into the system. * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses ‘View Record’ in his homepage. | 1. System shows all medical records added by the user along with his medical information | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | P.5 |
| Use Case Name | View Doctors |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses ‘View Doctors’ button. | 1. System shows two lists; 2. A list that contains doctors connected with this user. 3. A list that contains other doctors. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | P.6 |
| Use Case Name | Create Link with Doctor |
| Actors | Patient, Doctor |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * User is connected with a doctor. * Doctor can view this user’s profile. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User Presses ‘Connect with Doctor’. | 1. System views the QR code associated with the user. | | 1. Doctor scans the QR code. | 1. System adds the doctor to the ‘Connected list’ associated with the user. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User Presses ‘Connect with Doctor’. | 1. System views the QR code associated with the user. | | 1. Doctor can’t scan the QR Code |  | |

|  |  |
| --- | --- |
| Use Case ID | P.7 |
| Use Case Name | Delete Account |
| Actors | Patient |
| Pre-Conditions | * User has an account. |
| Post-Conditions | * User account is removed from the database. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses ‘Delete Account’ | 1. System asks for confirmation that the user wants to delete his account. | | 1. User confirms account deletion. | 1. System signs the user out, then removes the account from the database. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | P.8 |
| Use Case Name | View Medical Facilities |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. Patient presses ‘View Medical Facilities’. | 1. System show 2 lists; 2. Medical Facilities visited by the user. 3. Medical Facilities that the user hasn’t visited. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | P.9 |
| Use Case Name | Delete link with doctor |
| Actors | Patient |
| Pre-Conditions | * User is connected to a doctor. |
| Post-Conditions | * User isn’t connected with this doctor. * Doctor cannot view this user’s medical profile. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User views all doctors he/she is connected with. | 1. System shows all doctors. | | 1. User selects the doctor that he wants to delete his link with. | 1. System views this doctor’s information. | | 1. User selects delete link. | 1. System removes this doctor from the user’s connected list. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to view all doctors. | 1. System shows all doctors. | | 1. User isn’t connected with any doctors. |  | |

|  |  |
| --- | --- |
| Use Case ID | P.10 |
| Use Case Name | Add Medical Record |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * Medical record is added to the database and reflects on the application |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add medical record | 1. System asks for essential information. (Type, Note, Title, Doctor Name, Medical Facility Name) | | 1. User enters data correctly | 1. System adds medical record to patient’s profile | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User presses add medical record button. | 1. System asks for essential information. (Type, Note, Title, Doctor Name, Medical Facility Name) | | 1. User doesn’t enter data correctly. | 1. System rejects user’s order, and asks to fill required information. | |

|  |  |
| --- | --- |
| Use Case ID | P.11 |
| Use Case Name | Add Blood Pressure and Blood Glucose |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses add blood test button. | 1. System shows the form of blood tests. (Blood pressure or blood glucose as the user chooses.) | | 1. User adds required information based on the test. | 1. System accepts data and adds it to database. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User presses add blood test button | 1. System shows the form of blood tests. (Blood pressure or blood glucose as the user chooses.) | | 1. User adds wrong information. | 1. System refuses the order, and asks user to fill required information. | |

#### Doctor Use Cases

|  |  |
| --- | --- |
| Use Case ID | D.1 |
| Use Case Name | Register |
| Actors | Doctor |
| Pre-Conditions | * User doesn’t have an account |
| Post-Conditions | * Account is created and doctor is redirected to the home page |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses ‘Sign up’. | 1. System shows sign up page. | | 1. User enters his information. | 1. System checks the data and asks the user to validate that he is a doctor. | | 1. User sends validation. | 1. System confirms the user and, adds the user’s information to the database. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User presses ‘Sign up’. | 1. System shows sign up page. | | 1. User enters his information. | 1. System checks the data and asks the user to validate that he is a doctor. | | 1. User cannot validate that he is a doctor. | 1. System cancels the request. | |

|  |  |
| --- | --- |
| Use Case ID | D.1.1 |
| Use Case Name | Add Personal Information |
| Actors | Doctor |
| Pre-Conditions | * User pressed the register button |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User enters the required information. 2. (email, username, password) | 1. System Checks for availability. | |  | 1. System confirms account creation. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User enters the required information. 2. (email, username, password) | 1. System Checks for availability. | |  | 1. Credentials already exists | | 1. User tries again till credentials are available | 1. System accepts. | |

|  |  |
| --- | --- |
| Use Case ID | D.2 |
| Use Case Name | View patients |
| Actors | Doctor |
| Pre-Conditions | * User is logged in |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to view patients connected with him. | 1. System views patients connected in a list. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | D.2.1 |
| Use Case Name | View patient records |
| Actors | Doctor |
| Pre-Conditions | * User is logged in. * User viewed all patients. |
| Post-Conditions |  |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to view a patient’s medical record. | 1. System views this patient’s medical record | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | D.3 |
| Use Case Name | Delete Account |
| Actors | Doctor |
| Pre-Conditions | * User has an account. |
| Post-Conditions | * User account is deleted. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to delete account. | 1. System asks for confirmation. | | 1. User confirms. | 1. System logs out this user, then deletes account from database. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to delete account. | 1. System asks for confirmation. | | 1. User cancels deletion request. |  | |

|  |  |
| --- | --- |
| Use Case ID | D.4 |
| Use Case Name | View medical facilities |
| Actors | Patient |
| Pre-Conditions | * User is logged in. |
| Post-Conditions |  |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to view medical facilities connected on the application | 1. System views two lists; 2. Medical facilities of users who are connected to them. 3. Medical facilities users aren’t connected to them. | |
| Exceptions | - |

#### Clerk Use Cases

|  |  |
| --- | --- |
| Use Case ID | C.1 |
| Use Case Name | Add patient record |
| Actors | Clerk |
| Pre-Conditions | * User is signed in to the medical facility account. * User is connected to the same medical facility as the clerk. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a record to patient’s account. | 1. System checks if user is connected to the same medical facility. | |  | 1. System allows user to add a record. | | 1. User adds record. | 1. System stores record in database, and adds it to patient’s electronic medical record. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a record to patient’s account. | 1. System checks if user is connected to the same medical facility. | |  | 1. User isn’t connected with this patient. |  |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a record to patient’s account. | 1. System checks if user is connected to the same medical facility. | |  | 1. System allows user to add a record. | | 1. User adds record, but misses required information. | 1. System doesn’t allow record addition. | |

#### Medical Facility Use cases

|  |  |
| --- | --- |
| Use Case ID | M.1 |
| Use Case Name | Register |
| Actors | Medical Facility |
| Pre-Conditions | * User hasn’t registered before. |
| Post-Conditions | * User enters facility information |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to register a medical facility. | 1. System asks for required credentials. | | 1. User adds required credentials. | 1. System saves these credentials. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to register a medical facility. | 1. System asks for required credentials. | | 1. User adds required credentials. | 1. Credentials already exist. | |

|  |  |
| --- | --- |
| Use Case ID | M.1.1 |
| Use Case Name | Add facility Information |
| Actors | Medical Facility |
| Pre-Conditions | * User hasn’t registered before |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User added required credentials, and wants to add facility information. | 1. System adds credentials to database, and redirects this user to medical facility home page. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User added required credentials, and wants to add facility information. | 1. Facility credentials already exist. | |

|  |  |
| --- | --- |
| Use Case ID | M.2 |
| Use Case Name | Add Doctor |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * Doctor is added to the medical facility |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add doctor. | 1. System asks for doctor’s ID. | | 1. User enters doctor’s ID. | 1. System adds doctor to this medical facility. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a doctor to medical facility. | 1. System asks for doctor’s ID. | | 1. User sends doctor’s id | 1. ID doesn’t exist. | |

|  |  |
| --- | --- |
| Use Case ID | M.3 |
| Use Case Name | Add Clerk |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a clerk to medical facility. | 1. System asks for clerk’s credentials. | | 1. User adds clerk’s credentials | 1. System adds clerk to medical facility. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a clerk to medical facility. | 1. System sends a message that this medical facility exceeded maximum numbers of clerks |  |  |  | | --- | --- | | User Action | System Response | | 1. User wants to add a clerk to medical facility. | 1. System asks for clerk’s credentials. | | 1. User adds clerk’s credentials | 1. Credentials already exists. | | 1. User keeps trying till credentials are available. | 1. System adds clerk to medical facility. | |

|  |  |
| --- | --- |
| Use Case ID | M.4 |
| Use Case Name | View Members |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses view all members button. | 1. System shows 3 options; 2. Patients 3. Doctors 4. Clerks | | 1. User selects on of the options appeared |  | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | M.4.1 |
| Use Case Name | View Doctors |
| Actors | Medical Facility |
| Pre-Conditions | * User Facility is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses view all doctors button, it only views doctors connected to this medical facility. | 1. System views a list of all doctors connected to this medical facility. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | M.4.2 |
| Use Case Name | View Clerks |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses view all clerks, it only views clerks connected to this medical facility. | 1. System views a list of all clerks connected to this medical facility. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | M.4.3 |
| Use Case Name | View Patients |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | - |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses view all patients button, it only views patients connected to this medical facility. | 1. System views a list of all patients connected to this medical facility. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | M.5 |
| Use Case Name | Delete Facility |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * Medical Facility is deleted. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User presses delete account button. | 1. System asks for confirmation. | | 1. User confirms his action. | 1. System logs out this user, then removes this account from database. | |
| Exceptions | - |

|  |  |
| --- | --- |
| Use Case ID | M.6 |
| Use Case Name | Remove Doctor |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in. |
| Post-Conditions | * The removed doctor cannot interact with this facility. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wishes to remove a doctor’s account from a certain facility. | 1. System asks for doctor’s ID. | | 1. User enters doctor’s ID. | 1. System removes doctor from this facility. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wishes to remove a doctor’s account from this facility. | 1. System asks for doctor’s ID. | | 1. User cancels deletion process. |  | |

|  |  |
| --- | --- |
| Use Case ID | M.7 |
| Use Case Name | Remove Clerk |
| Actors | Medical Facility |
| Pre-Conditions | * User is logged in |
| Post-Conditions | * The removed clerk cannot interact with this facility. |
| Flow of Events | |  |  | | --- | --- | | User Action | System Response | | 1. User wishes to remove a clerk’s account from this facility. | 1. System asks for clerk’s ID. | | 1. User enters clerk’s ID. | 1. System removes clerk from this facility. | |
| Exceptions | |  |  | | --- | --- | | User Action | System Response | | 1. User wishes to remove a clerk’s account from this facility. | 1. System asks for clerk’s ID. | | 1. User enters invalid ID | 1. System informs user that this ID doesn’t exist | |