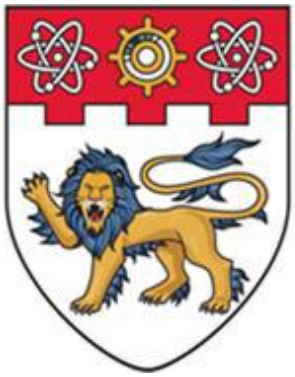

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

System Requirement Specification for QMePls

By Team Titans

Date: 9th September 2021

Seow Jing Hng Aloysius
Jacob Law Zhen
Jolene Tan
Soh Qian Yi
Samuel Lee Si En
Zeta Chua Hui Shi

Team Titans
School of Computer Science and Engineering

Revision History

Name	Date	Changes	Version
Sameul Lee	21/09/2021	Background and description	V 1.0
Qian Yi	22/09/2021	UML, system investigation, operational requirements	V 1.1
Aloysius Seow	22/09/2021	Constraints (5.1-5.5), input requirements	V 1.1
Jolene Tan	22/09/2021	Format, Output requirements, process requirements,UML,	V 1.1
Zeta Chua	22/09/2021	Architecture	V 1.1
Jacob Law	22/09/2021	Architecture	V1.1

1 Introduction

1.1 Purpose

The purpose of this document is to describe accurately the requirements for the QMePls software application. It will explain the purpose and features of the software.

1.2 Document Conventions

The document is based on the IEEE template for System Requirement Specification Documents. Different sections are numbered according to the flow of the document. Bold-faced texts are used for section and subsection headings, while italicised texts represent comments.

1.3 Intended Audience and Reading Suggestions

This document is mainly intended for project managers, developers and testers of the application, and users who would be using this application for their learning.

Section 1 provides an overview of the scope of the software and the purpose of this software, which can be read and understood by all stakeholders. The remaining sections contain interface descriptions, functional and non-functional requirements, as well as technical feasibility and constraints, which are intended for project managers, developers, and testers to read and understand in-depth about the application.

2 Problem Statement

Waiting times at clinics are increasing, especially with the additional precautions that are being taken due to COVID-19 where a clinic in Sengkang has an average waiting time of 86 minutes 95% of the time (Hirschmann, 2021).

When waiting times are over an hour, it could be too much of a discomfort for patients. Being able to book a queue slot beforehand or checking the current queue status would bring about convenience and extra comfort for people.

3 Overview

3.1 Background

With the rise of COVID-19 cases, we are faced with adapting to the “new normal”. In the face of longer waiting times due to increased safety measures and people regularly going to the clinics for PET (Pre-Event Testing), clinic queue lengths and waiting times have taken a surge upwards.

There is a need to reduce queue lengths to abide with the COVID number restrictions. Furthermore, it is also physically difficult for a person feeling unwell to queue for long periods of time.

Due to the reasons stated above, and to increase efficiency in clinics, we have developed an application to combat and resolve these issues - thus making the lives of people going for testing and those feeling unwell more convenient and efficient.

3.2 Overall Description

QMePls as the name suggests, is a central queuing system that links both clinics and patients together. This system consists of 2 main parts, namely the patient application and the clinic application.

The patient application allows users to find clinics near their area and view queue timings and lengths. An interested user can then add themselves to a queue in a particular clinic while also logging their symptoms which could be shown to doctors during their visit.

The clinic application allows the clinic staff to manage queues and bookings. This consists of viewing current queues, stopping new bookings and also pushing certain late patients down the queue stack.

The frontend is done through XML while the backend consists mainly of Java. The database being used is Firebase which allows for real time changes, a crucial feature for an ever changing queue based system.

4 Investigation & Analysis Methodology

4.1 System Investigation

‘QMePls’ application allows Patient Users to add themselves into the clinic queue. At the same time, Clinic Staff Users are able to skip to the next patient, or dequeue a patient. This information would be stored and updated into Firebase. Subsequently, an appropriate feedback would be displayed to the user as a pop-up, as well as a notification.

4.2 Analysis Methodology

4.2.1 Feasibility Study and Requirement Elicitation

Gather a team of skilled and ambitious people with knowledge or keen interest developing and implementing an android application that can be updated constantly to suit the needs of Patients Users and Clinic Staff Users. Additionally, organise regular meetings to ensure that everyone is on the same page.

Conduct interviews with Singaporeans of different age groups to get a better understanding of the minimum requirements of the application based on their feedback and concerns.

Carry out a Risk Management Plan to outline the best solution(s) based on the results of the interview. It is essential to collect feedback and opinions from the interviewees in determining the current situation as well as future system updates.

4.2.2 System Analysis and Requirement Specification

The Analysis methodology will embody the business analysis, requirement analysis, data analysis, process analysis, and application architecture:

- **Business analysis**: The budget requirement and sponsorship details will be discussed in the project proposal. Kindly refer to the same for further elaboration.

- **Requirement analysis**: User requirement definition, functional and security requirement

- **Data analysis**: Involve data collection process, data validation, data storage, manipulation and retrieval

- **Process analysis:** Data/process flow analysis, process decomposition and system interfaces

- **Application architecture**: The application's user interface is made to be as intuitive and engaging as possible. The user interface design follows our aim of creating an application that does not require any prior training to navigate the features.

4.2.3 Object-orientated design using UML

A detailed object-oriented design for the entire system will be developed. UML will be used for the graphical representation and documentation of the design. At its core, Patient Users will join a clinic's queue, and this information will be sent to Firebase. In addition, the system will allow Patient Users to enter their symptoms, and to look for nearby clinics with the shortest queues. The system will be secured with a Patient User's gmail and password, as well as the Clinic Staff User's unique credentials.

4.2.3.1 Use Case Diagram

Figure 1 : Use Case Diagram

4.2.3.2 Use Case Descriptions

4.2.3.2.1 Login

Use Case ID:	UC-1		
Use Case Name:	Login		
Created By:	Aloysius	**Last Updated By:**	
Date Created:	16 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, Clinic Staff User, User Database
Description:	Login to Patient User account or Clinic Staff account
Preconditions:	● User must have Internet Connection ● User must allow Live Location Permission
Postconditions:	● Patient User logged into application with patient user interface and personal information ● Clinic Staff User logged into application with clinic staff user interface
Priority:	High
Frequency of Use:	Frequently
Flow of Events:	1. Patient User shall enter the application and select the domain as ‘Patient’ 2. Patient User shall login using gmail, facebook, email or phone number. 3. System verifies Patient User information 4. System loads patient user interface
Alternative Flows:	AF-S1: Clinic Staff User shall enter the application and select the domain as ‘Clinic’: 1. Clinic Staff shall enter their given email and password to login 2. System verifies Clinic Staff information 3. System loads clinic staff user interface

	AF-S2: For new Patient Users 1. Patient User shall enter their existing gmail, facebook, email account or phone number 2. System will prompt Patient Users to enter their name 3. System shall register the new Patient User in the User Database 4. System loads patient user interface
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	● Patient Users already have a gmail account ● Clinic is registered under our system
Notes and Issues:	-

4.2.3.2.2 Book an Appointment

Use Case ID:	UC-2		
Use Case Name:	Book an Appointment		
Created By:	Aloysius	**Last Updated By:**	
Date Created:	16 Sept 2021	**Date Last Updated:**	

Actor:	Patient User
Description:	Starts the clinic appointment booking process
Preconditions:	● Patient User must have Internet Connection ● Patient User must allow Live Location Permission
Postconditions:	Patient User added to Queue Management System of selected clinic
Priority:	High

Frequency of Use:	Occasionally
Flow of Events:	1. Patient User shall click on the ‘Book’ Button 2. System directs to UC-3
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	
Notes and Issues:	-

4.2.3.2.3 Symptom Selector

Use Case ID:	UC-3		
Use Case Name:	Symptom Selector		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	16 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, Symptoms Database
Description:	Patient User selects their current symptoms from a list of symptoms in the Symptoms Database
Preconditions:	Patient User must have Internet Connection
Postconditions:	Patients will have the selected symptoms added to Health Report Card (Ongoing) after booking a clinic appointment
Priority:	Medium
Frequency of Use:	Occasionally

Flow of Events:	1. Patient User shall enter a symptom into the search bar 2. System will display if the symptom is in the Symptom Database. 3. User shall select the symptom 4. System shall record the symptom selected. 5. User shall click on the 'Next' Button to move to UC-4
Alternative Flows:	AF-S1: Patient User shall click on the 'Next' Button to move to UC-4 without selecting symptom AF-S2: Symptom not in the Symptom Database 1. System must display an appropriate message.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	Patient Users are not required to select a symptom in order to move to UC-4

4.2.3.2.4 Clinic Map

Use Case ID:	UC-4		
Use Case Name:	Clinic Map		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, User Database
Description:	Patient User must be able to see the top 3 nearest clinics based on current location

Preconditions:	● Patient User must have Stable Internet Connection ● Patient User must allow Live Location Permission
Postconditions:	Users shall be informed of ● Nearest 3 clinic's information
Priority:	High
Frequency of Use:	Occasionally
Flow of Events:	1. System fetches nearest 3 clinics from Patient User's current location, from the User Database 2. System fetches Google Map data from Google Map API. 3. System will display the Markers of the nearest 3 clinics on Google Maps 4. System will display all Buttons on the screen. 5. Patient User shall click on one of the 3 clinic Markers 6. System shall display the information of the selected clinic 7. Patient User shall click on the 'Book' Button to enter the clinic's Queue Management System
Alternative Flows:	AF-S5: User shall input a clinic name in the search bar 1. System shall display the information of the clinic being searched 2. Patient User shall click on the 'Book' Button to move to enter the clinic's Queue Management System
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

4.2.3.2.5 View Health Report Card (History)

Use Case ID:	UC-5
Use Case Name:	View Health Report Card (History)

Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, User Database
Description:	Displays the clinic information for all the past appointments that Patient User has booked
Preconditions:	● Patient User must have Internet Connection ● Patient User must have booked an appointment before
Postconditions:	Patient User shall be able to view the clinic information and past symptoms selected for all the past appointments that they have booked
Priority:	Low
Frequency of Use:	Rarely
Flow of Events:	1. Patient User shall click on the ‘Health Report Card (History)’ Button. 2. System will display the clinic information and symptoms selected for every appointment booking Patient User has made before. 3. Patient User shall click on the ‘Back’ Button to return to the profile page.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	Patient User must have booked an appointment and finished the appointment with a clinic before
Assumptions:	
Notes and Issues:	-

4.2.3.2.6 View Health Report Card (Ongoing)

Use Case ID:	UC-6		
Use Case Name:	View Health Report Card (Ongoing)		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, User Database
Description:	Displays the clinic information and Patient User's queue number for the ongoing appointment that Patient User has booked
Preconditions:	Patient User must have Internet Connection
Postconditions:	Patient User shall be able to view the clinic information and queue number of the ongoing appointment for the clinic booked
Priority:	Mid
Frequency of Use:	Occasionally
Flow of Events:	1. Patient User shall click on the 'Health Report Card (Ongoing)' Button. 2. System will display the clinic information (Clinic name, Clinic Address, and number of patients before Patient User) 3. Patient User shall click on the 'Back' Button to return to the profile page.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	Patient User must already have booked an appointment with a clinic
Assumptions:	-

Notes and Issues:	-

4.2.3.2.7 Add a Patient to Queue

Use Case ID:	UC-7		
Use Case Name:	Add a Patient to Queue		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Clinic Staff User, User Database
Description:	Clinic Staff User can add a walk-in Patient into the clinic's Queue Management System
Preconditions:	Walk-in Patient must register for appointment at the counter
Postconditions:	The clinic's Queue Management System accurately reflects the number of patients waiting in queue.
Priority:	High
Frequency of Use:	Occasionally
Flow of Events:	1. Walk-in Patient registers for an appointment at the clinic counter 2. Clinic Staff User shall click on the 'Add Patient to Queue' Button 3. Clinic Staff User shall enter the Walk-in Patient's name 4. System adds Walk-in Patient into the Queue Management System for the clinic to the back of queue
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-

Assumptions:	-
Notes and Issues:	-

4.2.3.2.8 Next Patient

Use Case ID:	UC-8		
Use Case Name:	Next Patient		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Clinic Staff User, User Database
Description:	Clinic Staff User can move up the queue after the first patient in the clinic's Queue Management System has finished their visit to the clinic.
Preconditions:	There are patients waiting in the Queue Management System
Postconditions:	The clinic's Queue Management System shall be updated
Priority:	High
Frequency of Use:	Frequently
Flow of Events:	1. Patient finishes their visit to the clinic 2. Clinic Staff User shall click on the 'Next Patient' Button 3. System shall prompt the Clinic Staff User to confirm action 4. Clinic Staff User shall confirm to call on 'Next Patient' 5. System shall remove the first patient in the clinic's Queue Management System
Alternative Flows:	AF-S4: Clinic Staff User shall cancel request to call on 'Next Patient'
Exceptions:	-

Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

4.2.3.2.9 Skip Patient

Use Case ID:	UC-9		
Use Case Name:	Skip Patient		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Clinic Staff User, User Database
Description:	Clinic Staff User can move up the queue in the event the first patient in the clinic's Queue Management System does not show up for their appointment
Preconditions:	There are patients waiting in the Queue Management System
Postconditions:	The clinic's Queue Management System shall be updated
Priority:	High
Frequency of Use:	Frequently
Flow of Events:	1. First patient in the clinic's Queue Management System does not turn up for the appointment after a stipulated time 2. Clinic Staff User shall click on the 'Skip Patient' Button 3. System shall prompt the Clinic Staff User to confirm action 4. Clinic Staff User shall confirm to call on 'Skip Patient'

	5. System shall move the first patient in the clinic's Queue Management System to third in queue to postpone their appointment
Alternative Flows:	AF-S4: Clinic Staff User shall cancel request to call on 'Next Patient'
Exceptions:	-
Includes:	-
Special Requirements:	SR-S5: If the same patient did not show up for their appointment the second time when it is their turn again, the patient will be removed from queue once Clinic Staff User clicks 'Skip Patient' again
Assumptions:	-
Notes and Issues:	-

4.2.3.2.10 Receive Notification

Use Case ID:	UC-10		
Use Case Name:	Receive Notification		
Created By:	Aloysius Seow	**Last Updated By:**	
Date Created:	17 Sept 2021	**Date Last Updated:**	

Actor:	Patient User, User Database
Description:	Notification will be sent to the Patient User when they are next in line (2nd in Queue Management System)
Preconditions:	User must have Internet Connection
Postconditions:	Patient User notified that they are next in line
Priority:	Moderate

Frequency of Use:	Occasionally
Flow of Events:	1. Clinic Staff User shall on the ‘Next Patient’ or ‘Skip Patient’ Button 2. Patient User moved up to 2nd in queue in the clinic’s Queue Management System 3. System will send a notification to inform the Patient User that they are next in line
Alternative Flows:	AF-S1: Another Patient User in the clinic’s Queue Management System cancels their booking 1. Patient User moved up to 2nd in queue in the clinic’s Queue Management System 2. System will send a notification to inform the Patient User that they are next in line
Exceptions:	-
Includes:	
Special Requirements:	-
Assumptions:	User must be active in the application
Notes and Issues:	-

4.2.4 Prototyping

QMePls will adopt the Agile methodology in implementing its rapid prototype. The prototype will be working in a tested environment while making use of licensed softwares and the free tiers of the proprietary softwares that is planned to be used during the production phase of the project. The ideal prototype environment would be running the application on Android Studios. The database will be in the free tier of the Firebase services. All of which is to act as a demonstration and proof of concept only.

5 Constraints

5.1 Scalability

QMePls will be scalable as it is using the Firebase. The increase in the amount of data required to be stored in Firebase will be based on the number of app users registering an account as well as the number of clinics registered under our system. Firebase however has a limit of 1 GiB (gigabytes) of stored data for the free pricing, & would require additional costs in order to store data beyond that size.

5.2 Proprietary Hardware and Software

QMePls requires Google's Firebase for its persistent data storage for Patient User accounts as well as clinic data, while Firestore is required to store the exhaustive list of available symptoms for our system. This therefore means that there is no requirement for proprietary hardware since the proprietary software will be sufficient to store the data we need.

5.3 Project Schedule

The timeframe of this project is within the range of 9 to 10 weeks after the acceptance of the proposal and the commencement of the software development life cycle. By the end of this 10 weeks, all of the core functionality of QMePls should have been complete.

6 Operational Requirements

6.1 Help Desk Support

Systems have a 24 x 7 access to email automated response assistance for frequently asked questions regarding the technical nature of QMePls such as sluggish system response, incompatible browser features, application errors, bugs, system downtime inquiries and account lock-out assistance, etc. An automated email service is preferred as it costs less than traditional support services and has immediate response times to answer user queries.

6.2 Application Services and Technical support

All QMePls services are expected to have a 24 x 7, 99.99% uptime throughout the year. Any updates to the application should be pushed out on Google Play Store while others are serviced for maintenance or updated to newer versions of the software should there be any. The application developers will have access to the full source code to address bugs and perform enhancements to the system if deemed necessary. Maintenance personnel in the post production phase will then have access to the full application and logging services via administrator roles but not to the source code. Should there be any issues found in the logging, the maintenance personnel shall contact the development team if necessary.

6.3 Administration Features

Since QMePls is an application that has 2 login domains, patient and clinic staff, there are varying levels of system access and functional authority. Each patient's access is limited to his/her own registration records while each clinic's access is limited to its own queue information. Only authorised system administrator(s) has access to all patient registration records and clinic queue information.

6.4 System hardware fail over and routine back up

Given that the backend system to QMePls resides in the localhost and which codebase is maintained and backed up on an online Github repository during the development phase, a quick cloning of the project from Github and setting of the backend should occur to continue development. During the production phase and beyond, QMePls would run on Google Cloud Platform. Google Cloud has a reliable fall-back system in play, using Actifio to provide Cloud Backup, Disaster Recovery, Migration, and Database Cloning.

6.5 Audit Trail

The system keeps an audit trail for documents and errors that occur within it. All documents (database entries) are time stamped when created and errors are logged with timestamps as well for tracing purposes.

7 Functional Requirements

7.1 Login/User Authentication

7.1.1 Users must be able to select a domain - Clinic or Patient.
7.1.2 Patient Users must be able to register and create an account via gmail.
7.1.3 Patient Users must be able to log in with registered credentials.
7.1.4 Clinic Staff Users must be able to log in with fixed credentials.
7.1.5 System must retrieve login credentials information from the User database.
7.1.6 System must be able to authenticate users.

7.2 Book Appointment System

7.2.1 Patient User shall follow a pipeline to book an appointment.
- 7.2.1.1 Patient User shall click on the ‘Book’ Button to start the pipeline process.
- 7.2.1.2 Patient User shall input their symptoms.
  - 7.2.1.2.1 Selected input Symptoms shall be stored in the system.
- 7.2.1.3 Patient User shall select a nearby clinic or input a clinic into the search bar.
  - 7.2.1.3.1 Patient User must be able to view the number of people in queue in each displayed clinic.
  - 7.2.1.3.2 Patient User shall click on the ‘Queue’ Button to be added into the queue.

7.3 Queue Management System

7.3.1 System must retrieve the queue information from the User database.
7.3.2 System will send a notification to inform the Patient User when they are next in line.
7.3.3 Patient User must be able to be added into the Queue.
7.3.4 Clinic Staff User must be able to add a walk-in patient.
7.3.5 Clinic Staff User must be able to skip to the next patient.

7.4 Symptom Selector

7.4.1 System must be able to retrieve symptoms from the Symptom Database.
7.4.2 Patient User shall key in experienced symptoms in the symptom search bar.
7.4.3 Patient User shall click on chosen symptom from fuzzy search list to log into the Health Report Card (Ongoing).
7.4.4 System must be able to store these selected symptoms into an array list to display in the Health Report Card.
7.4.5 Patient User may choose to skip this step and proceed to Clinic Map.

7.5 Notification System

7.5.1 System must be able to retrieve data from the User database.
7.5.2 System will send a notification to inform the Patient User when they are next in line.

7.6 Clinic Map

7.6.1 System must retrieve the clinic information from the User database.
7.6.2 System shall display 3 nearest clinics from the Patient User, indicated by red-coloured markers on the Clinic Map.
7.6.3 System shall display the ‘Current Location’ Marker of Patient User.
7.6.4 System displays all 3 clinic’s information as a bottom navigation drawer on click.
- 7.6.4.1 System shall display the Clinic's name.
- 7.6.4.2 System shall display the Clinic’s address.
- 7.6.4.3 System shall display the Clinic’s telephone number.
- 7.6.4.4 System shall display the Clinic’s opening and closing hours.
- 7.6.4.5 System shall display the number of people in the Clinic’s queue.

7.6.5 Patient User shall select a clinic and click on the ‘Queue’ button.
- 7.6.5.1 System shall display a disclaimer pop-up to advise Patient Users if they miss their queue.

7.6.6 Patient User may input a Clinic Name into the search bar.
- 7.6.6.1 System shall display an ‘Input Location’ Marker on the map, indicated by blue-coloured markers on the Clinic Map.
- 7.6.6.2 System shall display 3 nearest clinics based on the Input Location.

8 Non-Functional Requirements

8.1 Performance

8.1.1 System should send Patient Users a notification within 5 seconds when they are second/first in the ‘booking’ queue.

8.1.2 Pages must take less than 30 seconds to load their respective information.

8.1.3 The system must respond to a user input within 10 seconds.

8.2 Usability

8.2.1 Users should be able to learn how to use the software with ease within 5 minutes.

8.2.2 The application can be used by literate users of all ages.

8.2.3 Rate of error by users is very low due to low requirements for user input that may cause error.

8.3 Reliability

8.3.1 The Queue Management System will be updated in real-time, making the queue numbers accurate and free of errors.

8.4 Supportability

8.4.1 The system should be functional in any touch screen device with Android operating system (Android 10 and above).

8.5 Maintainability

8.5.1 The system allows new clinics to be registered into our system by providing each new clinic with a Clinic Admin account

8.5.2 The system allows clinics to be unregistered by removing the Clinic Admin account from our server’s database.

8.6 Extensibility

8.6.1 The system allows easy extension of new functions through our design considerations

8.6.1.1 Observer Pattern allows for listening to new changes of an object's state (Refer to A.1.1).

8.6.1.2 Singleton Pattern allows for only one instance of a class to exist (Refer to Section A.1.2).

8.6.1.3 Single Responsibility Principle (SRP) allows easy implementation of new features in the future (Refer to Section A.1.4) .

9 Input Requirements

9.1 Patient User & Clinic Staff User Login credentials

Each patient user will have a unique 'uid' assigned to them when they first register for an account in the system. This 'uid' maps to the Patient User's name and Health Report Card (History). Clinic Staff accounts will similarly have a unique 'uid' for each clinic registered in the system.

9.2 List of Symptoms

The exhaustive list of symptoms will not be fully provided to the Patient Users, but the input symptom search bar has the 'fuzzy search' function to help narrow down symptom searches based on the Patient User's input.

9.3 Action Codes

All other actionable codes such as booking appointment procedures and checking of Health Report Cards is available under the Use Case Description to assist users.

10 Process Requirements

10.1 Firebase Transaction

Firebase is used as our database of choice. The NoSQL database is hosted on a cloud, thus, every query or request is pointed that way. For developmental purposes, the database has allowed connection from any IP addresses should they have admin credentials. This brings smoother developmental processes to data related functions in the backend as we do not need to worry about security mechanisms first and can proceed with development regardless. Thus, offsetting some development time in the early phases of the project. Subsequently, when the security mechanisms are in play, the cluster will shift towards a development-production hybrid authentication system to also allow authenticated users to access the database alongside admins.

10.2 Data Integrity

Firebase database does not have built in data integrity functions. Thus, the team shall enforce the data integrity formulas themselves and perform data integrity checks with complex queries. Furthermore, transactions are committed when completed and rollback if unfinished or time-out.

10.3 Data Validation

Data error from the user's end and back-end must be properly handled. Firebase allows developers to write rules to enforce data validations by restricting writes based on the new data being written. Thus, there will be data validation done when user input is required, and error-handling routines as part of the whole system.

10.4 Performance

The QMePls systems should aim to allow 100 concurrent users with 99.99% availability throughout the year with a 24/7 uptime consistency period. The QMePls system should load the client-side front end within a time period of 10 seconds to minimize click-away prior to the loading of the QMePls mobile application.

10.5 Data Repository

The system will maintain several data repositories across a typical data flow lifecycle. The most identifiable one would be the Firebase database for persistent data storage. Another would be the mobile application local storage allows the browser to store simple client-side information, for example, symptoms selected could be persisted throughout several sessions of accessing QMePls.

11 Output Requirements

11.1 Transaction Summary and Confirmation

A multitude of output is generated from the Firebase backend when users interact with it. For the case of response generated by the Firebase backend, the team has decided to log them by printing them into the console during the developmental phase and to create a logging function for the production phase of the application. The logs shall be in the JSON format for ease of use in querying and analysing the request/response and transactional data.

11.2 Exception Report

Exception reporting will be done using the Firebase Crashlytics SDK. By default. Crashlytics automatically collects crash reports for all app users. Custom keys and logs will be added by the team to record non-fatal exceptions as well as the events leading up to a crash. This will be displayed on the Firebase console at the end of every application run for review by the team. These exception reports could be further organised into a text file daily and sent to the emails of the administrators.

11.3 Dialog Confirmation

After a successful account creation, the system’s backend service will generate an account creation confirmation and welcome message. The user will then be redirected to the homepage of QMePls and can start using the application.

12 Hardware Requirements

12.1 Network

The device must be connected to the Internet upon launching of the application. The Firebase database can be accessed via HTTP(S) calls made by an authenticator instantiated in Android Studios.

12.2 Client Devices

The clients will be mobile devices that run on Android 10 operating system. Samsung devices preferred for optimal application functionality. Devices must support touch screen functions, be able to connect to the Internet and have GPS location services enabled.

12.3 Production Supported Systems

Should the product enter the production phase, its backend servers and services will be set up in the widely available, scalable and affordable Google Cloud Platform Services. The system's database will enter the Firebase paid services.

13 Software Requirements

13.1 Client Operating System & Application

- Android Operating System

13.2 Network System

Network software and protocols in order for the system components to communicate:

- TCP/IP

13.3 Mainframe System

- Development: Google Cloud Platform Services and Firebase Community Service
- Production: Google Cloud Platform Services and Firebase Paid Service

13.4 Licenses

Since our software will be using Android Studio for development, we will be required to use the SDK License from Google, as well as the Android Open Source Project (AOSP) as it uses a few open source initiative approved open source licenses.

14 Deployment Requirements

The three-layered architecture of our product is divided into three parts, which are namely the Presentation, App Logic and Data Layer, and are kept separated to ensure low-coupling of the layers and each layer is easily modified by adding or removing classes within the layers. The layered architecture ensures the dependencies are one-directional and therefore only the upper stream layers can make calls to the next downstream or same stream layers. Layered architecture is a style that also strictly follows the Single Responsibility Principle which we have also used in our system as explained in Section 7.1.2. The logical separation of layers in one machine-platform helps minimize the impact of changes to be contained within each layer.

Our application uses Android Studio IDE hence the unidirectional flow starts with the **Presentation layer** which consists of UIs, activities and fragments. This layer is also in charge of receiving user inputs such as search location inputs and user-click buttons. This layer then calls the **App Logic layer**, which consists of application logic classes to process the user inputs and calls the third layer, the **Persistent Data laye**r which contains the entity classes that pull data from our Database. Taking 'Display Clinics' as an example, the user interacts through the clinic button in *ClinicUI* class (Presentation layer) which then calls *ClinicManager* class (App Logic Layer) to retrieve the data of the clinics in the *Clinic* class (Persistent Data Layer) to display back to the user the clinic locations on the map.

Compared to N-tiered architecture, our application focuses on the grouping of related functionalities into distinct layers stacked vertically on each other. Additionally, the communication between layers is done directly within the same machine such as the Firebase in our Database layer can directly communicate with our Persistent Data layer classes, App Logic layer classes and Presentation layer classes. Hence there is no need for the physical separation of layers in different servers or machines like in N-tiered architecture.

The figure below shows the layered architecture diagram that we have chosen for our system.

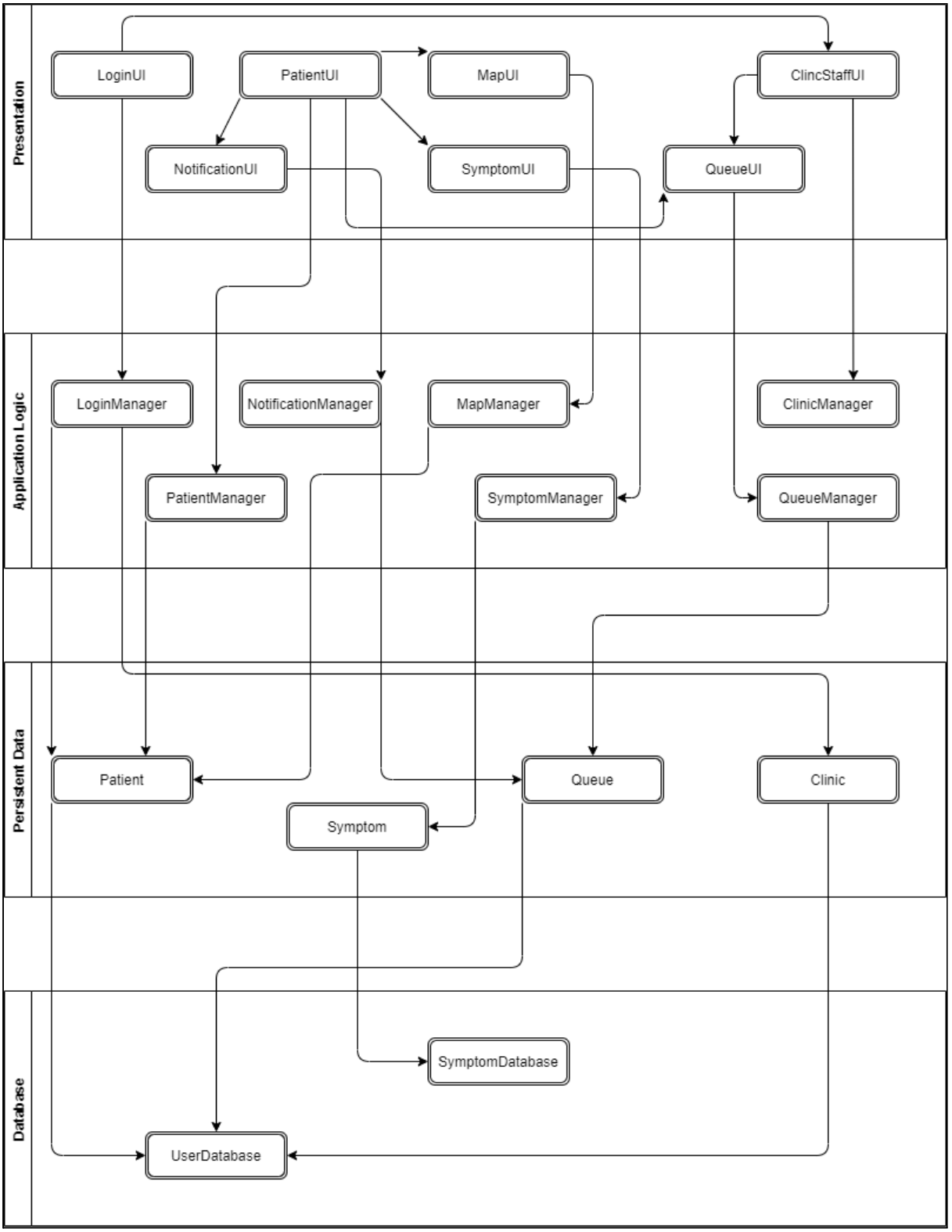

Figure 2: Layered Architecture Diagram

Appendix

Data Flow Diagram

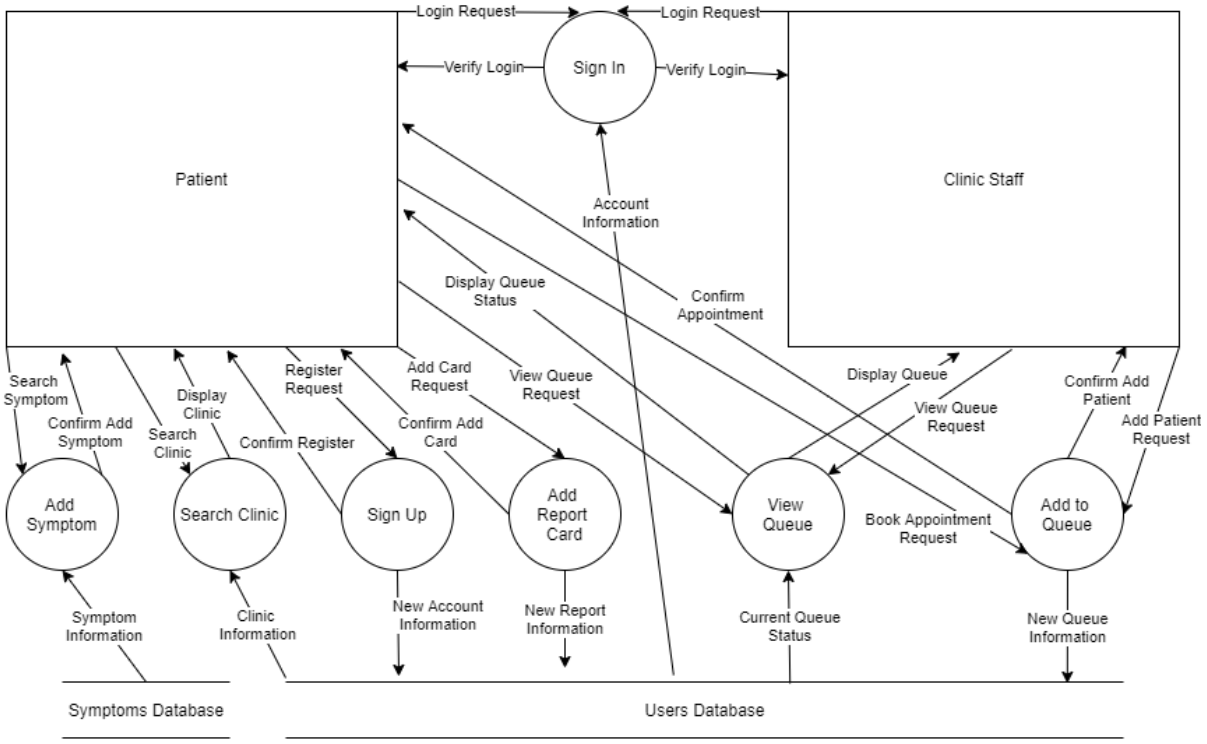

Figure 3: Data Flow Diagram

Dialog Map

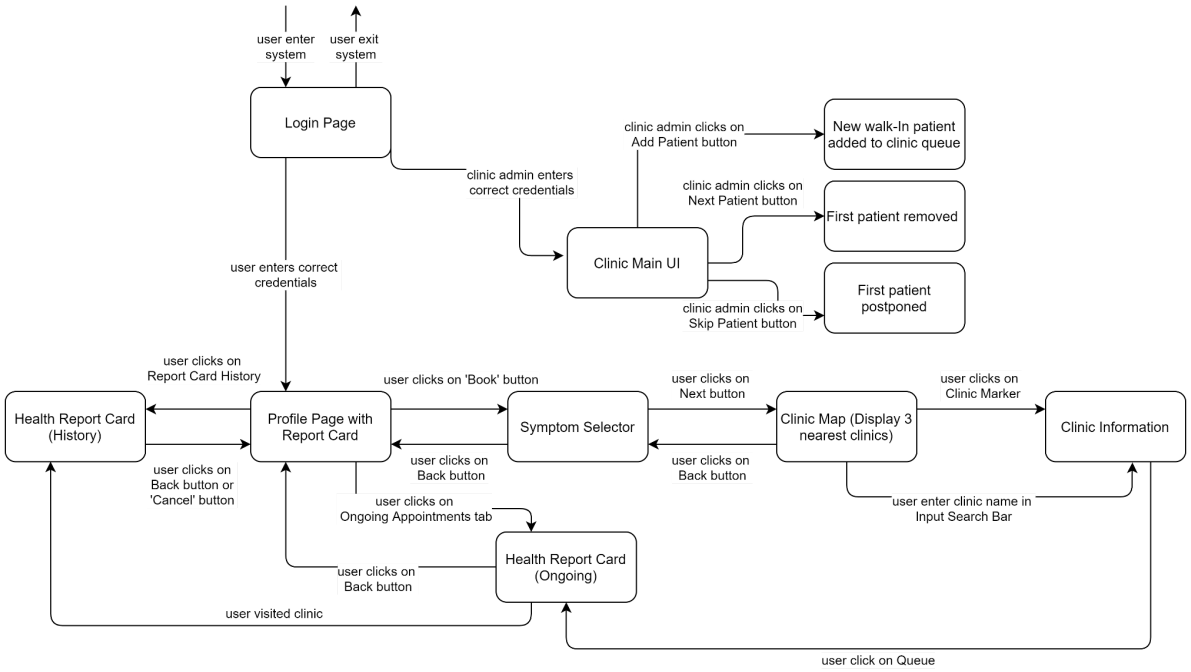

Figure 4: Dialog Map

Class Diagram

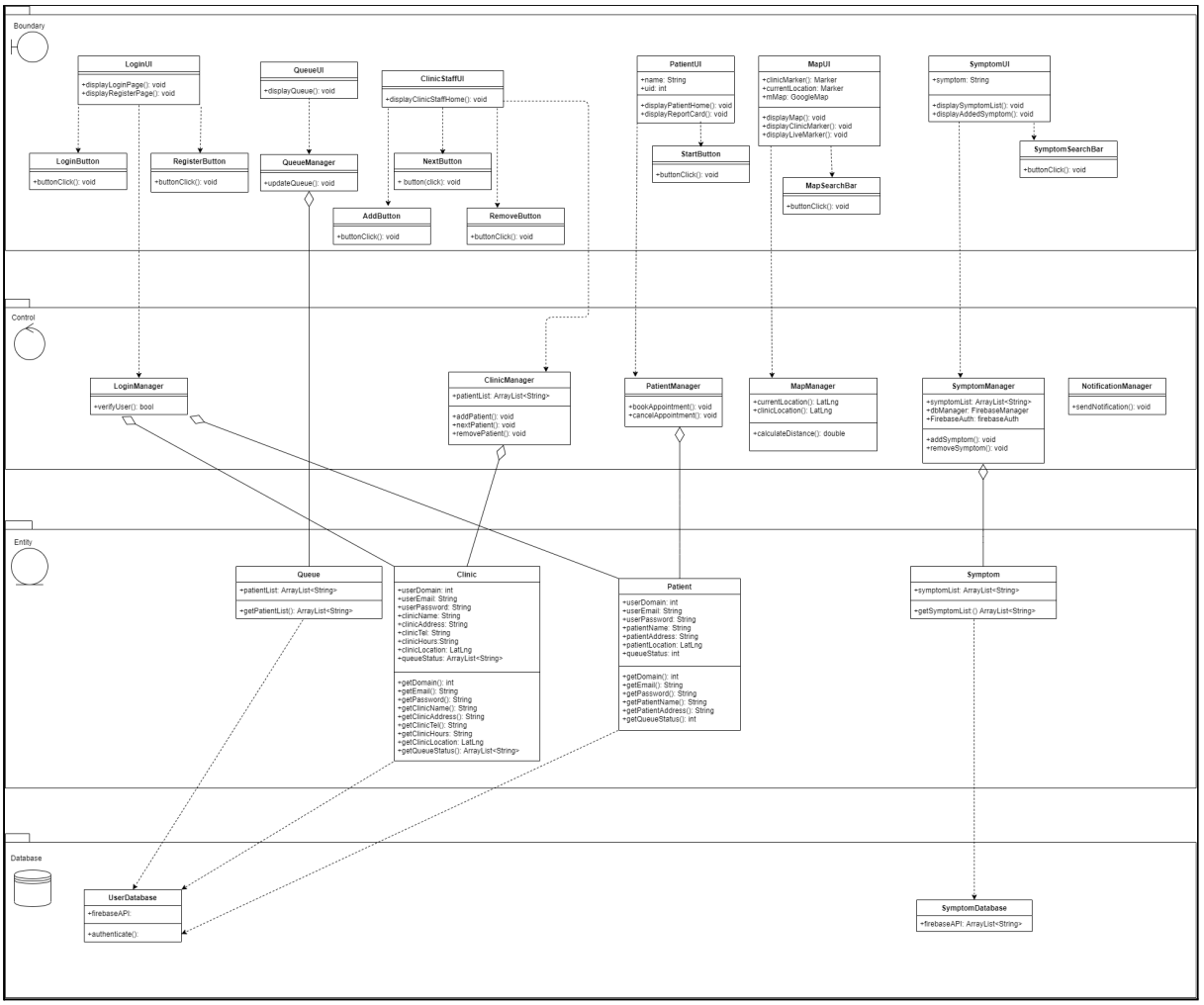

Figure 5: Class Diagram

A. Design Patterns and Practices

A.1. Design Considerations

A.1.1. Observer Pattern

The observer pattern is a means to communicate between different classes and objects by allowing objects to notify other objects about changes in their states. It provides a good solution for designing our application to follow the open/closed design principle, which states that application should be open for extension but closed for modification. The general implementation of the observer pattern is shown in the diagram below.

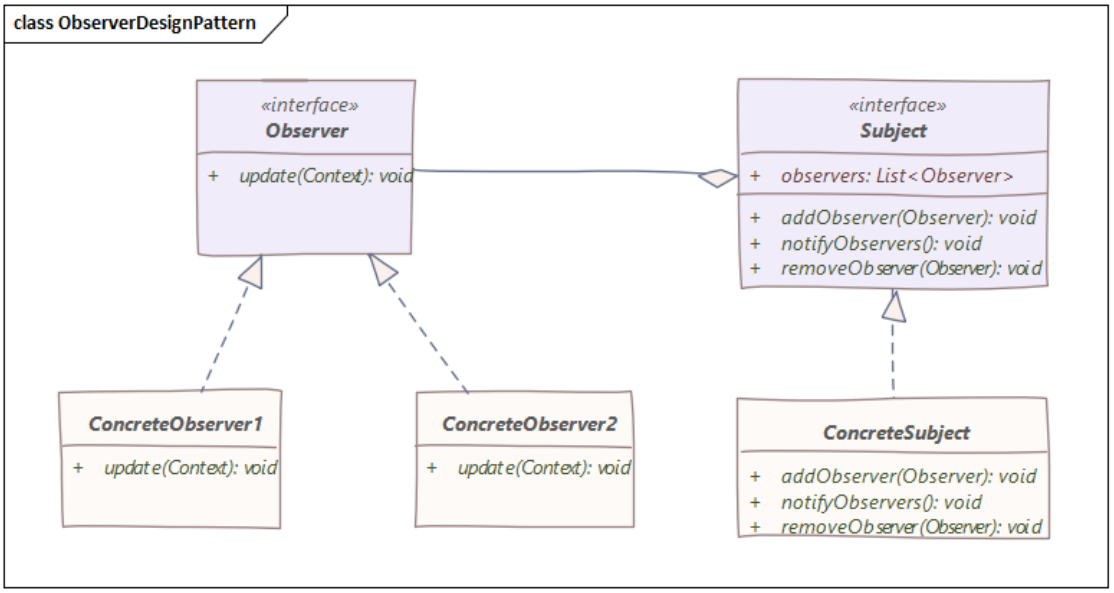

Figure 6: Observer Design Pattern

The various buttons in our application implements the observer pattern. The Button represents the subject and the numerous app Fragments represent the observers. The Fragments are all interested in the Button's state as the Button displays different functionality in every Fragment. The Fragments are given the option to unsubscribe from observing the button at any moment, i.e. we do not want the button to function in a certain way when particular Fragments are not visible.

Our Button implements a few methods. We set the listener method to the Button and everytime we click on it, it will notify the Fragments to know

about the event. Each Fragment only needs to implement one of the Button methods that fits their purpose.

A.1.2. Singleton Pattern

The Singleton design pattern specifies that only a single instance of a class should exist with a glocal access point. In an Android application, there are many objects which only need one global instance as instantiating more than one instance would cause problems like mixing of data, causing confusing results.

Various objects in our application implement the Singleton pattern. For example, the instantiation of our GoogleMap set to Singapore boundaries is done only once in our MainActivity. In this way, the map can be stored as a persistent layer that is passed and displayed with the loading of different classes. This is essential to our application which uses the map as a base for many of our app functions such as the search for clinics and showing the nearest 3 clinics. If we were to instantiate this GoogleMap more than once, it would result in incorrect app behaviour due to resource overuse.

A.1.3. Single Responsibility Principle (SRP)

The basis of SRP states that each responsibility is an axis of change. If a class assumes more than one responsibility, there will be more than one reason for it to change. While designing our application, we adhered to SRP and made each class responsible for only one feature of the application. This ensures that any change to each class will not affect another class, which also allows our application to be open to extension.

This is seen in our Class Diagram (*Figure 4*), one of our functions is to display the nearest 3 clinics from the user on the map, and initially the *MainActivity* class contained all the boundary, control and entity functions for the clinic markers (boundary), logic for calculating distance of clinic from user (control) and clinic information such as the opening hours and phone number (entity). Using the SRP, we have split the function to 3 different classes - boundary class *mapUI,* control class *mapManager* and entity class *Clinic,* all contained respectively in each of the boundary, control and entity packages which follows the layered architecture diagram (Figure 3). This process is repeated for our other functions such as adding symptoms during the book appointment process.

A.1.4. MVC Architecture

The MVC Architecture splits the system into the 3 separate components: **Model** contains the application data and business logic such as *ClinicInfo* class in the entity package, **View** contains the UI logic for everything visible on the screen and user interaction such as *ClinicUI* class (boundary package, *xml* files). **Controller** is the glue between view and model and reacts to the user's input and presents the data requested by the user such as our *ClinicManager* class. The model component has no knowledge of the interface.

MVC has more emphasis on the View Layer and thus MVC allows for multiple Views for each Model which makes the UIs more open and flexible to changes, reusable and extensible. Our project utilizes this such as adding more UI features to the *Boundary* package (View Layer) does not require additional changes to the *Entity* package (Model layer). This ensures maintainability, and reusability of the components as modifications can be easily carried out in each MVC layer individually without affecting the other layers, allowing our application to be more open and flexible to changs.

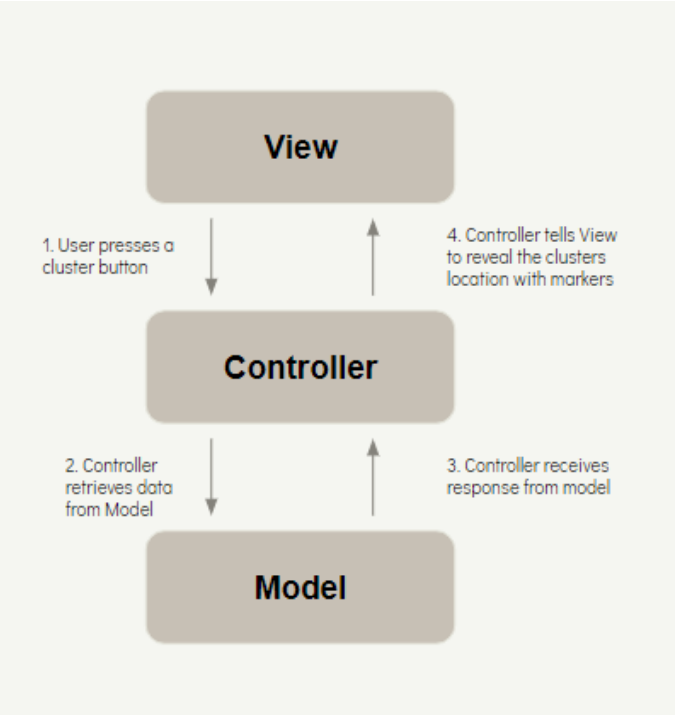

Figure 7: MVC Architecture Diagram

Glossary

Term	Description
Patient User	Refers to users of the patient side application interface, which requires a patient account.
Clinic Staff User	Refers to users of the clinic side application interface, which requires an admin account.
Health Report Card (Ongoing)	A placeholder that stores the Patient User's selected symptoms, currently booked clinic, and the number of patients in front of the user in queue.
Health Report Card (Past History)	A list of placeholders of every booked clinic appointment made by the Patient User, together with the visited clinic info.
Appointment	A session to consult the doctor after booking a slot for a particular clinic using the 'booking' function.
Booking	Refers to when Patient Users made an appointment with a clinic and enters the clinic's queue management system.
Symptom Selector	A list of predefined symptoms for users to select for their appointment.
Clinics	Any clinic registered in the database.
Current Location	Location of the user's device at the point of login that is running the application based on Global Position System (GPS).
Input Clinic	User's desired clinic to visit (entered in the search bar).
Marker	A marker identifies a location on the map.

Clinic Information	Information of a clinic's name, telephone number, address, opening/closing hours & number of patients in queue .
Notification	Refers to the device notification, not the in-application notification.
Clinic Map	Google Map containing location markers of clinics from the database.
Queue Management System	The back-end management system for Clinic Staff Users to manage the patients currently in the queue system for the clinic.
Walk-in Patient	A patient who registers for an appointment at a clinic counter instead of through the application.

References

Hirschmann, R. (2021, April 19). *Singapore: Waiting Time for Consultation in Polyclinics 2021*. Statista. Retrieved September 22, 2021, from https://www.statista.com/statistics/874609/waiting-time-for-consultation-in-polyclinics-singapore/