UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER

CS/IT Honours Final Paper 2020

Title: Software Implementation of Vision Healthcare Management System

Author: Justin Dorman

Project Abbreviation: Vision

Supervisor(s): Aslam Safla

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	20
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	**80**

Software Implementation of Vision Healthcare Management System

Justin Dorman

Computer Science Department
drmjus001@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

This report intends to provide detailed insight into the software development process behind the Vision Health Care Management System. This paper introduces the organisation, Vision Medical Suite (VMS) and identifies the inherent problems with their current paper based information capture system. The aims are subsequently formulated, with a fundamental objective to develop a fully functioning web application that will streamline the operations of VMS. The paper then goes on to compare related systems as well as evaluate various aspects of the system and associated technologies. Requirements stipulated by the organisation are discussed and the design of the system is described based off of these requirements. The process of implementation is then detailed to portray how the system was developed using the agile methodology and React, NodeJS and MySQL as the underlying tech stack. Finally, findings from the usability and functional testing procedures are discussed and analysed to ultimately conclude that the system is sufficiently usable and functionally effective. Thus, the project can be deemed successful, meaning that it is ready to be deployed into production and materialise the aims.

KEYWORDS

Healthcare management system, Web application development, Database, Application server, Authentication, Authorisation.

1 BACKGROUND AND INTRODUCTION

1.1 Background

Vision Medical Suite (VMS) is a Non-Profit Company that provides free medical and dental care to patients from their affiliated beneficiaries, being orphanages, frail care centres and homes for the mentally and physically challenged. They do this by engaging healthcare professionals to voluntarily provide their services. VMS is classified as a health care service provider and possesses a strong vision to provide high quality treatment to vulnerable sectors of South African society. Their ultimate purpose is to create a space where high quality care is determined by needs rather than financials [1]. These services are meant to be done by the state; however, the problem lies with long waiting times.

The services that VMS offer are split into two broad categories being sedation clinics and general practices. The sedation clinics are held on a monthly basis with a focus on providing medical and dental care to children and adults that are physically or mentally challenged. These services are provided to patients from the beneficiaries as well as those who are unable to afford health care. While the clinic does allow for walk ins, the vast majority of patients are booked in advance. On average, for each sedation clinic, roughly one hundred patients are attended to. A visit at the clinic comprises of various phases which the patient transitions through. At each phase of the visit, different staff members are actively involved in logging and capturing important information relating to the visit. The general practice services take place on a daily basis. Health workers who have their own private practices volunteer their services to patients from the aforementioned beneficiaries. There are no segmented phases for a general practice visit, but there are a range of information forms to be captured.

1.2 Problem Statement

As it stands, VMS manages all their operations manually through a paper-based information capture system. This brings about major inefficiencies in the fast paced work environment that they operate in, and consequentially reduces their level of performance. The method is cumbersome, outdated and results in increased form capture time, decreased quality of records and limitations with respect to sharing and access of patient data. It becomes increasingly time consuming for staff members to find the correct sheet of paper, physically write out all the information and try keep everything coherently organised. Further inefficiencies are introduced when trying to manage appointments, locate patients' previous visit information and keep track of operations. It is clear that VMS is in need of a digital system to efficiently and effectively keep track of patients' visits and manage operations. VMS is an NPO and hence has a limited budget. They cannot afford to outsource the development of this application and have therefore liaised with the University of Cape Town to assist. As a result, myself and two other students have been allocated the development of this system for our honours project.

1.3 Aim Formulation

We, as a team, have been presented with the following task: Create an online health care management system that streamlines the management of staff, patients and the various processes executed in VMS's operations. This project will expose us to exciting new tech stack and provide us with experience into real-world, full stack development. On top of this, we will be providing a respectable social impact by helping out an incredible NPO that provides free health care to the vulnerable sector of society. We are thus highly motivated to develop the best application we possibly can. The ultimate aim for the project is to develop a fully functional, full stack application that will be used by VMS to increase their efficiency, better manage their operations and ultimately treat more patients. With all this in mind, the specific project aims can be depicted as follows:

• Increase efficiency and efficacy: Through the implementation of the proposed system, this project aims to reduce the amount of time

spent manually capturing data into paper forms. Through this, it is hoped to increase the level of performance and effectiveness of operations.

• Increase quality of data: Paper-based records are often of poor quality as they are prone to errors and untidiness. Digital records hold to a consistent quality, and allow for validation, thereby ensuring greater quality of ongoing records.

• Increase ease of access and analysis: Paper records require physical storage which inevitably restricts sharing of records across departments. Additionally, it increases the complexity for data analysis. Electronic storage will allow for all information to be stored centrally, thereby increasing access, allowing for detailed reports and promoting interoperability.

The end goal for this project is to create a robust and scalable system that is ready for deployment into production. For this to occur, it will have to meet all functional and non-functional requirements and be suitably usable in a fast-paced environment.

1.4 Ethical, Professional and Legal Issues

1.4.1 Ethical

Ethical clearance was obtained from the Faculty of Science Research Ethics Committee to conduct user tests. Prior to each test, participants were informed of the nature of the project and the purpose of the test. Following this, they were asked to sign a written consent form. All data collected was kept anonymised and recordings were kept confidential. There were no risks associated with the testing phase as the usability testing was done over a video conference call in order to avoid the risk of exposure to Covid-19. Testing of the application was done using fake data, and thus there was no danger of being able to identify a person. There are ethical issues concerning testing medical professionals during a pandemic, as it takes valuable time away from them. This was taken into account, and the situation was monitored closely throughout the development. It was eventually decided that the testing of these users should be conducted at the end of development, as Covid cases began to lessen. Additionally, it was ensured that the tests were conducted efficiently, minimising wastage of time.

1.4.2 Professional

The limited budget of VMS has forced the team to undergo the development of the system using free tools and software. When the system is completed and deployed, it will be owned by UCT and Vision Medical Suite. The system will be made live and handed over with a clear database. VMS will then be able to add and manage all data independently. VMS has proposed that after the system is developed, they will pay the team a monthly fee for maintenance and potential evolution of the system. Additionally, they are willing to pay for a business analytics service to generate reports on the data captured in the system.

1.4.3 Legal

The deployed application will be dealing with real, sensitive patient data. The application thus needs to be designed with the assurance of security of information and the appropriate role-based access configuration, in order to comply with Department of Health's requirements around patient data [3]. To do this, the application was implemented using recognised and established security practices. Authentication, authorisation and encryption were implemented, to ensure access control and confidentiality. The system also needs to comply with the POPI act which states how institutions should act when collecting, processing, storing and sharing an entity's personal information, to ensure accountability [2]. In order to comply, the system ensures that information is made available only to authorised users. Additionally, adequate controls and measures were put in place to safeguard patient information.

1.5 Structure of Report

I was involved in the following areas of the project: database creation and management, authentication, authorisation (privacy and confidentiality) and backend development. This report will focus solely on these aspects of the system. To progress, an overview and analysis will be drawn on the aforementioned sections along with its related fields and technologies. Additionally, related open-source projects will be discussed. Following this, an overview of the system design will be portrayed along with a more detailed design into the database and application server. Subsequently, implementation will be described, explaining the methodologies and tech stack used to develop the system, followed by a more technical description of how the features were implemented. Testing methodologies along with its results and findings will then be discussed and analysed, which will allow for the conclusions to be made. Finally, a reflection of the development process will be given along with recommendations for those who wish to replicate such a project.

2 LITERATURE REVIEW

2.1 Type of Application to Be Developed

A web application would be the most feasible and most appropriate choice to develop the Vision Management System. Web applications serve many advantages over desktop applications, including being easy to update, cost effective, compatible across multiple platforms and highly accessible. More specifically, a single-page web application is most ideal, as it is appropriate in VMS's case and serves the advantages over multi-page applications of being fast, responsive, and simplified. [4, 5, 6]

2. 2 Database

Due to the functional capacity of the proposed application, a dynamic web application is required, along with a database. Incorporating a database allows for easy access, high scalability, fast queries, multiple users, flexibility, and less room for error [7]. Having a database is essential for any interactive system being created. It is imperative to follow best practices for a scalable, high performance application. The two appropriate database types for VMS's case are SQL and NoSQL databases. The Vision application requires complex queries, reports and relationships between data types. Despite NoSQL's impressive scalability and high performance, SQL databases offer stability, assurance of data integrity and support for relationships between data items and would thus be a better fit for VMS's requirements. Furthermore, an SQL database will work well with VMS's application, as the data itself is very structured. Finally, the format of how data is transmitted should be decided for standardisation purposes. It is important to decide on a fast, reliable, and easy to use standardised format to interchange data between the client and the server. Compared to XML, JSON format is the better alternative due to its speed of transfer and intuitive nature.

2.3 Application Server

Using REST architecture will bring about the benefits of lightweight web services, simplicity, scalability and universal presence. Compared to SOAP architecture, REST allows for a greater variety of data formats, easy integration with websites and is generally faster as well as uses less bandwidth. [14]

2.4 Authentication and Authorisation

Authentication is vital for securing a web application and should be implemented in the best possible way. Token-based authentication using JSON Web Tokens is a better alternative over cookie-based authentication due to its stateless nature, scalability, high performance, mobile friendliness, and its necessity when using single-page applications. To further increase security, token refreshing should be implemented, and passwords should be stored securely and hashed accordingly. [9,10,11]

Authorisation acts as an extremely important mechanism to introduce privacy and confidentiality in the application. It is especially prevalent for VMS, due to the sensitive patient information and doctor-patient confidentiality that needs to be managed. Identity management should thus be set up so an authenticated user gains all the permissions from their respective role and will only be allowed access to certain resources based on the permissions embedded in such a role [12].

2.5 Related Work

Upon investigating the possibility of using open-source systems and incorporating the functionalities stipulated by VMS, it was found to be counterproductive. The systems OpenMRS and OpenVista were analysed, and it was determined that their services were on a more general management scale and catered for larger organisations with many departments. Conversely, VMS requires a more fine-grained, tailored application to manage their unique services which target specific beneficiaries. With that in mind, manipulating the codebase to incorporate VMS's specific requirements will be inefficient, as a lot of time will go into understanding the system and changing code. Additionally, there are more security risks as the code is made available to the public, making it easier for hackers to find exploits. Developing a system from scratch, tailored specifically for VMS will allow for greater customisation and control over the system. This will improve the efficiency of the system as the exact requirements will be provided and implemented using best practices available. Additionally, it will allow for a better user interface and user experience. The interfaces of the aforementioned open-source systems are not so user-friendly or easy to use, which would make it difficult to use in a fast-paced environment. [13]

3 REQUIREMENTS ANALYSIS AND DESIGN

3.1 Requirements Analysis

Prior to designing the system, requirements analysis had to take place. During this phase, user expectations were defined in terms of functional and non-functional requirements. These requirements define the function of the system from the client's perspective, and establish the system's functionality, constraints and goals [48].

3.1.1 Requirements Specification

The specification phase is where the system's services, constraints and goals are established. The team underwent this process by collaborating directly with the project proposer. Initially, a two hour video conference was held where the proposer outlined all his expectations for the project, including the users of the system, how such users will use the system, and the specific information inherent to the system. The meeting began with the proposer explaining the underlying conceptual model to provide us with a holistic understanding of the what the system should do. Following this he went through an existing hospital management system which is very similar in nature to the proposed system. This provided us with a list of specific functional requirements for the system, along with all the information that should be included. The meeting was recorded, which allowed us to refer back to it in order to coherently outline the specific requirements. Following the agile methodology, further meetings were regularly held to clarify certain requirements and possibly add additional requirements.

3.1.2 Requirements Modelling

The modelling phase incorporated organising the requirements in a systematic and comprehensible manner. To initiate this phase, the team compiled a document with all the screenshots of the example system presented in the initial specification meeting, along with a list of associated requirements. This allowed us to outline detailed requirements for every aspect of the system. Following this, a use case diagram was compiled to provide an overview of the usage requirements for the system (refer to Appendix A to see the use case diagram). This proved to be a useful tool for the implementation as it gave a good idea of the various actions that should be implemented, from a user's perspective. After compiling all the requirements together, one of the team members worked on designing the initial interface prototype of the system. This was designed to be interactive and served as a highly useful tool in visually representing the requirements to the proposer and getting a feel how the user would interact with the system. All the pages of the final interface design of the system can be viewed in Appendix H. The proposer provided feedback which was taken into account in all aspects of the design phase. Making use of the agile, evolutionary prototyping methodology, subsequent prototypes were then produced, incorporating the feedback provided for each iteration. Based off the iterative feedback, improvements were made, or additional features were added until the final design was accepted. An emphasis was placed to ensure the accommodation of changing requirements [49].

3.1.3 System Outcome

Based off the requirements analysis, it was determined that the system should be designed as an interactive application, including functionality to manage processes for both the sedation clinic and general practice. Fundamentally, VMS requires a patient-centric system that can manage appointments, check in patients for a visit, capture their records and manage their transition throughout the visit. It is imperative that the application is as usable and efficient as possible, as VMS operates in a fast paced environment. The application must accommodate two levels of users, namely staff and admin. The staff are effectively the health workers who play the role of actively managing the operations at the clinics and general practices. They are ultimately responsible for controlling the patients' movements and capturing their information. Staff roles include doctor, dentist, dental assistant, anaesthetist, nurse and allied health. The application should allow these users to

capture necessary information associated with a patient's visit, which then gets stored in a database. For the general practice component, patients are managed individually. To the contrary, the mass sedation clinic requires multiple patients to be managed at once and should thus be split into segmented phases. Each phase should contain a list of patients waiting for the respective activity, ordered their urgency and waiting time. The users should be able to select patients from these lists and capture information relevant to the phase they are in. Once the required information is captured, users should then be able to move the appropriate patient to the next phase, until the visit is complete, and the patient exits the clinic. An admin panel is required, where higher organisational users should be able to manage high level operations relating to the system.

3.1.4 Functional requirements

The specific functional requirements can be drawn from the final, accepted design, summarised as follows:

General system requirements:

The user should have the option to log into the system and toggle between the sedation clinic and general practice components. Available to both components should be the option to register a patient and view a patient's profile, displaying their information in addition to all their past visits. Users should be able to view/edit the patient information as well as each visit's information. On top of all of this, authentication, authorisation, validation and error handling should be incorporated in the system.

Sedation clinic requirements:

The sedation clinic component should contain a dashboard displaying a list of all patients in the system and all visits in the current clinic. This should be visualised in a table format, along with all associated visit and patient information. For the patient list, users should have the option to book an appointment and view the profile for each patient. For the visits list, users should have the same options as above, along with the option to view visit forms, cancel the visit and jump to the current location of the patient in the visit. Furthermore, an appointment manager is required, which should display a list of all upcoming clinics and their associated appointments. Users should have the option to add, edit and remove appointments for a selected clinic as well as have the option to change the clinic capacity. Additionally, an option should be provided to check-in a patient and then start their visit (only if the clinic is scheduled for the current date). For each phase of the clinic, there should exist a table displaying the patients that are waiting in that phase, along with associated visit and patient information. The phases include waiting for triage, waiting for theatre, in theatre, waiting for disposition and waiting for exit. For each patient in the waiting list, an option should be provided to transfer the patients to the next or previous phase which, when complete, should automatically log the time of transfer. Additionally, the user should be able to navigate to the visit forms and capture/edit information inherent to the visit. Included in the forms should be a voice to text option and a search for ICD10 codes. The user should be able to search through all existing ICD10 codes and add them to a table displayed on the respective form capture screen.

General practice requirements:

The general practice component is required to have a dashboard similar to that of the sedation clinic, displaying patients and current day's visits. For the general practice component, every patient gets assigned to one or more staff member. The signed in staff member should thus only be able to view their assigned patients in the table. For each row in the able, the user should have the option to add appointments and view patient profiles. Staff members should only be able to serve their assigned patients in the general practice and should hence be limited to only view and edit their own patients' information. General practice requires an appointment manager, but with a full calendar interface as it is a daily practice. The signed in user should be able to view their appointments for any day as well as add, edit and delete appointments. Additionally, the option should be provided to start a visit for an appointment on the current day. For visits that are started, the user should have the option to navigate to the visit forms specific to the selected general practice visit and capture as well as edit information.

Admin Panel requirements:

The admin panel should be accessible only to authenticated admin users. The panel must include a patient and staff table where the user can view all patients and staff and edit/delete their information. Additionally, admin should be the only users able to register a staff member which, upon success, sends an email prompting the staff member to change their password. A dynamic data component should be provided to allow the admin to view, add, edit and delete dynamic data, including clinic locations, beneficiaries, hospitals and exit locations. A sedation clinic component is required to view all past and upcoming clinics, with the option to edit, delete and end a specific clinic. When a clinic is ended, all remaining patients should get removed from the waiting lists, all remaining appointments should be moved to the next available clinic and the clinic should be marked as complete. Finally, a general practice component should be provided to view all staff members and their assigned patients and provide the option to add and remove links.

3.1.5 Non-Functional requirements

Assurance of data privacy and confidentiality, security of system, high performance, fast response times, seamless UI/UX, accuracy, reliability and compatibility for all screen sizes.

3.2 Design Process

The next step is to design and detail information about the system, to ultimately prepare for the implementation phase.

3.2.1 Overall System Architecture Design

To effectively understand the system and how it works, it is important to provide a brief overview of the system architecture and how the different parts of the system collaboratively function to achieve the aforementioned requirements. The system makes use of a unidirectional, layered architecture, consisting of a frontend as the UI, an API as the business logic and an interactive database as the data layer. The API (Application Programming Interface) is software intermediary developed on the application server, which acts as the middleman of the system. It is responsible for directly communicating with the database and processing the HTTP requests that are fired from the frontend application. Essentially, the API allows the frontend and database to communicate with each other, indirectly. The API directly interacts with the database and gathers the appropriate data to be sent back as a response to the frontend application. Subsequently, the response data gets stored in a global storage state and dealt with to visually present to the user.

JSON is used as the standardised format to interchange data between the client and the server. [45]

The database was designed in accordance with the relational model of data, taking into account normalisation principles. The server was then designed, conforming to a REST (Representational State Transfer) architecture to incorporate universal principles. Finally, the frontend application was designed as a component based system, ensuring code reusability and maintainability. The component tree for the frontend structure can be viewed in Appendix B and the backend structure in Appendix C. As my section of the project deals exclusively with the database and application server, only these sections will be further discussed.

3.2.2 Database Design

Database design is the process of producing a detailed model of a database. Principally, the database design process encompasses designing the logic behind the base structure of the DBMS being used. The most crucial part of the development of a web application is the design of the database, as it provides the foundation for the application to be developed. Small decisions in the beginning have a huge cumulative impact later on in the development [50]. It is therefore imperative for the database design process to align with the agile methodology that was adopted in the requirements analysis phase. The interface design acted as a highly useful representation of the data that is required to be stored in the database. Each prototype in the evolution was thoroughly analysed to determine the specific information that should be incorporated in the database, until the final prototype was accepted.

To visualise the design of the database, an entity-relationship diagram (ERD) was created (see Appendix D). ERD's act as a highly useful framework to create and manipulate abstract and conceptual representations of data. Benefits arising from the use of ERD's include efficient communication, visual representation, easy understanding and high flexibility [15]. For each agile iteration, the data to be stored was gathered and divided into subject-based tables. Tables include patients, staff, admin, sedation visits, general practice visits, appointments, waiting lists and all information inherent to the respective visit, grouped by subject area. A unique primary key was assigned to each table. To maintain consistency, a non-null, auto incrementing integer was used for each table. Using integers as primary keys allows for easy and fast indexing. Following the creation of the tables, relationships were formed. The visits tables (both sedation visits and general practice visits) form the centre of the connection network. A one-to-many relationship exists between patients and visits, as one patient can be in many visits, but each visit can only contain one patient. The same logic applies to the relationship between appointments and patients. To map out the relationship between the information tables and their associated visit (either sedation or general practice), a one-to-one relationship was created. The reason for this is because all information captured for a patient is unique to its associated visit. There is an indirect relationship between the information captured for the visit and the associated patient. A many-to-many relationship was created between the procedure's information table and staff. The reason behind this is because each procedure can have many staff members tagged, and each staff member can be involved in many procedures. The same logic applies to procedures and ICD10 codes. To form the connections, bridge tables were used [16]. The final ERD in Appendix D visually represents the design and structure of the database. It provides a more comprehensive and intuitive understanding of the structure and relationships inherent to the database.

Following the creation of the database structure, normalisation principles were applied to reduce data redundancy and improve data integrity. Where appropriate, more columns were added, new tables were created to form new relationships and large tables were split into smaller tables. Doing so ultimately ensured that the design conforms to a level of normalisation. [17, 18]

3.2.3 Application Server Design

The architectural design of the application server is imperative to viably sustain the project in the long run. A good architecture allows for clean, reusable code and faster development speed. Additionally, it helps avoid repetitions and makes it easier to add new features into the applications [20]. At a fundamental level, the REST architectural style was used to design the application server, as motivated in the literature review section. The server was designed using HTTP (Hypertext Transfer Protocol) as the underlying protocol, where HTTP requests get used for all CRUD (create, read, update and delete) operations. Every request includes all the necessary information for the server to respond. The server is thus stateless, meaning that client context is never stored on the server between requests, ultimately improving scalability. The following constraints were considered as design rules when designing the server: uniform contract between APIs of the server – API's created should be similar in nature; server and client must be independent; no client context shall be stored on the server between requests; well managed caching and finally, layered system implementation. [19]

Once the REST architecture design principles were taken into account, the next step was to design the file structure of the server. The starting file, server.js, which is the core of the application, was created and stored in the root folder, as every other file is dependent on it. Following the creation of the application file, the next most important process was to manage the configuration and connection to the database. A separate config folder was created, with a file containing all the details to create the connection to the database. Another folder for the database was created, with a file that is responsible for creating a singleton instance of the database by exporting a connection [20]. Following this, a routes folder was created, containing all files that define the API. In these files, subject related endpoints, along with its logic, were implemented in each respective route file. Furthermore, a middleware folder was created for any custom middleware file that is required, as well as a shared folder for files used across multiple components. A .env file also was formulated, to store all private, confidential values such as credentials and secrets. This file acts as a secure environment config section for the project, in which only invited collaborators can see. [21, 22, 23, 24, 25]

Through this file structure, the role of each component is clearly defined, emphasising the separation of concerns design principle and adherence to the component based structure. Furthermore, files relating to a single feature are grouped together, allowing for code reuse [22]. The response messages sent back to the frontend also had to be designed appropriately and consistently. Standard REST API response codes were used to identify the specific scenario. In the body of the response, either the requested resource or a customised message about the results was returned. [46]

4 SYSTEM AND IMPLEMENTATION

4.1 Approach

4.1.1 Tech Stack

To begin the implementation of the project, the team made use of modern frameworks and technology in order to efficiently and effectively materialise the requirements and design. To create and manage the relational database, MySQL was used. The reason for using MySQL is because of its scalability, high performance, high availability, strong data protection, management ease, abundance of support and open source freedom [26]. To develop the frontend application, React.js was used. React is an open-source JavaScript library used to build dynamic and engaging web interfaces for single page applications. React was chosen primarily due to its fast rendering allowing for incredibly fast speeds, which is important for any management system. Additionally, React enables reuse of components, provides clean abstraction and incorporates great state management [27]. Node.js was used to develop the application server. Node.js is an open source, server-side platform that executes JavaScript code outside a web browser. It was chosen due to its fast and scalable environment when dealing with multiple client requests. Node.js is highly customisable and the data sync between clients happens very fast. Additionally, React.js and Node.js work very well together, as the React DOM has components designed specifically to work with Node.js. Express was used as the web application framework for Node.js, due to its minimal and flexible nature, along with its robust set of features for web applications [28]. To test the implemented API commands and verify the responses through the implementation of the project, the toolchain Postman was used. Finally, MySQL workbench was used as a visual database design tool to interact directly with the database.

4.1.2 Development Methodology

To manage the software development process, the team made use of the scrum framework, which is a sub-group of agile methodology. The scrum team consisted of the development team, the scrum master and the product owner. The development team was made up of myself and the two other team members, working together to deliver shippable increments at the end of each sprint. The product owner was the proposer of the project, who conveyed the requirements to us. Finally, our project supervisor acted as the scrum master who helped facilitate meetings. There was no team leader, but rather, we worked collaboratively to solve any fundamental issues and problems. [29]

Each sprint was a total of two weeks in length. Prior to each sprint, a thorough planning session took place, where the team would determine which product backlog tasks would be included in the sprint backlog, and who would be responsible for each task. Segmented tasks were assigned, with the assurance that tasks could be completed simultaneously with minimal dependencies on other members' sections. Jira, the task management tool, was used to outline the tasks and track progress. Every workday, the team would engage in a communication meeting where each member would present their progress since the last meeting, planned work before the next meeting and any problems they have experienced. Additionally, a review was held with our supervisor on a weekly basis, where he would monitor our progress and provide feedback [29]. Finally, to allow for collaborative work, and code management, our team made us of Git, the version control software.

4.2 Features Implementation

This section will speak about how the aforementioned functional requirements were implemented, exclusive to my sections, being backend development and database manipulation. The system was implemented in a modular fashion, using best practices wherever possible. The most efficient and effective algorithms and techniques were used wherever possible to ensure scalability and fast response times. Throughout the development, the codebase was constantly commented and refactored, to make future maintainability easier and aid in the intuitive understanding of the implementation.

4.2.1 Environment Set Up

Before feature implementation could begin, the project environment had to be set up and configured. To begin the process, the MySQL database was created and configured appropriately. The database schema was then created, and the SQL code was run to generate all tables and relationships. Upon liaising with ICTS, a virtual machine was set up on the UCT's servers to run the MySQL server instance. Subsequently, the whole team could connect to the same centralised database and co-ordinate activities. Following this, the server set up began. NPM (Node Package Manager), a package manager for JavaScript, was used to aid package installation, version management and dependency management. It was used because of its significantly large software registry [30]. Within the application server, NPM was used to create a package.json file (which holds metadata relevant to the project) and install the framework, Express. The Express application was created in the server.js file, which acts as the middleman for all HTTP requests. Following the creation of the Express application, appropriate middleware was initialised. The body-parser middleware package was used to parse incoming JSON requests to create a new body object before it can be handled. In addition, other middleware was used including helmet: which helps secure HTTP headers returned to the client, cors: which enables cross-domain communication and morgan: which simplifies the process of logging requests to the express application [31, 32, 33]. The route files were then formulated, and router objects were declared and exported within each file. Subsequently, every route was imported in the server file and parsed to the Express application. For each individual route, a path is specified which gets used to fire requests to the respective endpoint.

4.2.2 Database Connection

For the server to communicate and interact with the database, a connection needs to be made. For a large scale application such as this one, it is imperative to use connection pooling. Connection pooling is a data access pattern where database connections are cached so that the connections can be reused with subsequent requests, when required [34]. The pooled connection was made using the MySQL driver which provides a built-in connection pooling feature. Connection pooling allows for multiple queries to be made within a transaction, and for data objects to be shared between subsequent queries. By using connection pooling, the performance of the application is increased significantly. The getConnection() method was used to retrieve a connection from the pool, and connection.release() was used to return a connection to the pool after use. Connections are exported from the pool.js file, which in turn is imported in every route class to allow for communication with the database. [35]

4.2.3 Authentication and Authorisation

As specified in the requirements, only admin users should be able to register new staff users. To implement this, a POST endpoint that is made accessible to only admin users was created. Within this endpoint, an SQL query gets sent to the database which subsequently generates a new tuple in the staff table. The information that gets populated in this tuple comes from the information sent in the body of the request, along with a randomly generated temporary password. Upon success, an email gets sent to the newly registered staff member, prompting them to change their password. The nodemailer module is used to send the email, via a Gmail account. Embedded in the email is a magic link, that navigates the user to a change password page when clicked. The staff member can then change their password, which then updates the respective row in the staff table. The password gets hashed prior to it being saved in the database, making it confidential. The Bcrypt password hashing function is used as it scales with computation power and hashes every password with a salt, thereby making it more secure. When a user signs in, they are located in the database based on the entered username. The entered password is then hashed and compared to the respective password value in the database to determine whether the entered credentials are correct.

As justified previously, the system will make use of JWT's to implement authentication and authorisation. To begin the implementation, a file called auth.js was set up for both normal users and admin users, playing the role of an authentication service. To generate and verify the JWT tokens, the jsonwebtoken module was used. An access token secret which is used to sign the JWT token was then generated using a complex random string. This secret string gets stored in the .env file to keep it confidential. Upon a successful sign in, the access token gets generated and signed with the secret, along with the respective username and role embedded in the JWT. The access token then gets returned to the frontend and is subsequently stored in the local storage of the browser. This token gets sent as a query parameter to the server, along with every request. To manage the verification of these tokens on the server, a middleware function called checkAuthToken was created. Every request to the server gets parsed to this function before the endpoint logic gets computed. This algorithm verifies the token, using the token secret that is stored in the .env file. Once verified, the user object gets attached to the request and calls the next() function to continue with the endpoint logic computation. If the verification fails, a 401 error is sent back to the frontend. Since the authentication middleware binds the user object to the request, the user role can be retrieved and checked, to implement authorisation.

Thus far, if the token had to be stolen, the users account would be compromised. It is therefore imperative to implement token expiration after a specific period of time. On the event of token expiration, a new token needs to be generated so the user can keep their session active. To do this, a refresh token was created on a successful sign in, which then gets used to generate new access tokens. The refresh token is sent back to the frontend along with the access token and gets stored in local storage. The frontend makes a call to the refresh token endpoint just before the access token expires. The server then returns a newly generated access token for the frontend to include in all subsequent requests [36].

Further authorisation constraints are placed in the general practice, where staff members can only access information of their assigned patients. This is done by making use of a bridge table that links patients and staff. For all SQL queries that should adhere to this authorisation constraint, the relationship between patients and staff gets set, and only the table rows with matching keys are returned.

4.2.4 CRUD Operations

Each endpoint in the API retrieves a database connection from the pool. Subsequently, SQL (Structured Query Language) is used to communicate with the database, using the connection established. For every endpoint, the appropriate SQL statements are created and incorporated in the query that is sent to the database. When there is specific data required to be retrieved from or added to a database, a question mark is used as a placeholder in the SQL statement. An array containing all of the user data is then additionally incorporated in the database query, which then populates the respective fields in the SQL statement (in order). Successful actions will return the status code 200 or 201 (when creating new resources), along with the requested resource in the response body. If the request body from the frontend has invalid values or is of the incorrect format, a bad request message gets returned with a status code 400. If there is no token in the request, or if it the verification fails, an unauthorised message gets returned with a status code 401. Finally, if the action fails on the server side, an internal server error message is returned, with a status code 500.

GET endpoints are used for read operations. Within these endpoints, SELECT SQL queries are fired off to the database, fetching the appropriate fields from the appropriate tables. JOIN clauses are used to combine tuples from two or more tables based on a pre-defined relationship. INNER JOINS are used to fetch records that have matching values in both tables and LEFT/RIGHT OUTER JOINS are used to return records when there is a match in either the left or right table. To identify a specific resource in the database, values are passed from the frontend, as parameters in the URL. The API extracts these values and applies them to the respective WHERE clauses in the SQL. When the results are retrieved from the database, they are sent back as an array of JSON objects embedded in the response, along with a status code of 200. GET requests are used to fetch all form information for a particular visit. To do this, queries are processed for each information table, retrieving the appropriate fields to be returned. JOIN clauses are used to manage the relationships, with the visit table being the centre of connection. Each information table contains a foreign key which gets matched to the primary key of the visit table. Hence, the visit id is sent along with the request, which is then used to define the relationship and populate the WHERE clause, to retrieve the required information. The retrieval of information for each table on the frontend works in a similar fashion, with the addition of pagination. To implement pagination, the number of pages required are calculated based on the limit specified by the frontend and the number of elements retrieved from the database. This value is sent back to the frontend which then deals with visualising the page numbers. When the user selects a particular page, the frontend sends the page number that the user selected, along with the table limit. The server then responds with the appropriate segment of results which is determined by the specified page number and limit. Search functionality also uses GET requests. The search function checks whether the appropriate fields contains the search input term that is sent from the frontend as a query parameter. To implement this, an SQL statement is used, selecting the appropriate fields and adding expressions of interest together using CONCAT_WS(). The concatenated value then gets checked against the specified term,

which is encapsulated in the wildcard '%.' The results are then sent back to the frontend in the response. [37, 38 39]

POST endpoints are used for create operations. To validate the input, schema validation is implemented using the powerful and extensive JOI data validation library. Within each route, a schema gets created which encompasses an object containing all the keys of the request body (the JSON object sent from the frontend), along with the associated validation criteria defined. For those inputs where a pre-defined validation method does not exist in the JOI library, a customised regular expression is generated and used. Before firing off the database query, the request body is parsed to the schema, and each object is validated according to the specified criteria. If the validation is successful, the SQL will be sent off to the database, otherwise a bad request error message is returned to the frontend. On successful validation, the retrieved database pool is used to send off the appropriate INSERT SQL queries to the database. Inputted data from the frontend is extracted from the request body and stored in a newly generated array. This array then gets embedded in the database query to populate the respective fields in the SQL statement. Externally defining the array reduces the risk of SQL injection attacks. [37, 39, 40]

PUT endpoints are used for update operations. Before firing off the UPDATE SQL statement to the database, the aforementioned JOI validation technique is utilised. The frontend sends the identifier of the tuple to be updated as a parameter in the URL, along with the appropriate information as an object in the request body. The UPDATE query gets run, and the respective tuple in the database gets updated with the new, inputted information. [37, 39]

DELETE endpoints are used for remove operations. The tuple identifier gets sent as a parameter in the URL, and the DELETE SQL statement then gets fired off to the database, subsequently removing the respective tuple from the database.

4.2.5 Async Functions

Nodejs is asynchronous in nature, which ensures non-block code execution. Asynchronous (async) code executes without having any dependency and sequential order. This ultimately results in faster execution due to the increased efficiency and throughput. It is therefore imperative to implement asynchronous functions to manage any dependencies in the code [41]. Where there are multiple queries required that are dependent on each other, async functions are needed to compute them in a sequential manner. To do this, async functions incorporate the use of promises, which represent the async task that will eventually finish. To work with these promises, the async/await feature is used. The await function is used to pause the code that comes after it when waiting for an async function to finish. When the async function is complete, the await function returns whatever the async function returns, and the code continues to run. Await can only be used inside an async function. GET requests incorporate the use of asynchronous functions, where the response only gets returned to the frontend when the results have been fetched from the database. [42]

Where there are multiple queries required that are not dependent on each other, the promise.all function is utilised. This function takes an array of promises as input. The function is then run and gets resolved when all promises in the array are complete and rejected if any one of them fails. To implement this within the context of endpoint logic, each query is incorporated in an async function, and embodied in a promise. All these query promises are populated in an array and run through the promise.all function. This function computes every query in the array simultaneously. If resolved, a successful response is sent to the frontend and if rejected, an error response is returned. Using this async functionality allows for concurrent operations, immensely improving efficiency. [43]

For any for loop that is required, async functions are needed to sequentially process the logic. To do this, the async.eachSeries function is used. A callback function is included as a parameter and called upon the completion of each iteration, to reiterate the loop [44]. These loops are used when ending a clinic and managing ICD10 codes. For the end clinic functionality, a nested loop is used to iterate through all appointments of a specified clinic (outer loop) and all upcoming clinics (inner loop). The function checks each upcoming clinic whether there is available capacity. If so, the current appointment in the loop will be moved to that particular clinic and the callback function for the outer loop will be called, to jump to the next appointment (iterate the outer loop). If there are no available spots in the clinic, the callback for the inner loop gets called to jump to the next clinic (iterate the inner loop). This will happen until all appointments in the specified clinic are moved to the next available clinic. For ICD10 codes, an array of codes is received from the frontend and looped through using async.eachSeries. For each iteration, an SQL query is fired to the database, to check if the ICD10 code exists in the local database. If so, a connection in the bridge table is made with that particular code, otherwise a new tuple in the ICD10 codes table is created, and a relationship is formed with the newly created code. Following the completion of either one of these operations, the callback function gets called to reiterate the loop. This process happens until all codes in the list are processed in the loop.

5 TESTING METHODOLOGIES

The user testing was split into two phases. The first phase of user testing was conducted after the foundational system was developed, with all core functionality implemented. The second phase was conducted after incorporating the feedback from the first phase. The first phase involved testing six users who worked in the tech industry and or had a background in Computer Science. It was anticipated that this group of users would be able to provide a sound technical analysis and feedback on the UI/UX of the system. The second phase involved testing three users who have had prior experience working with some sort of healthcare management system. Finally, a test was conducted with the project proposer to assess his feedback on the system as a whole. Each session began with a brief explanation on the context of the organisation, overview of the system and the purpose of the usability test. The participants were then guided through the informed consent form and requested to sign it before the test could commence. Each test began with the interviewer providing the user with a series of high level tasks. The type of tasks requested can be seen in Appendix E. The users were asked to think aloud with respect to any thoughts they may have when undergoing the various tasks, to help measure the effectiveness of various aspects of the system. Each test took place through a video call, where the interviewer shared their screen and the user gained remote access to use the system. The session was recorded, and the feedback was analysed to determine the intuitiveness and effectiveness of the system.

In addition to usability testing, functional unit tests were conducted on both the frontend and backend. JEST is a JavaScript testing

framework and was used to test the functionality of the frontend and the backend applications, as they are both coded in JavaScript. With respect to the testing on the frontend, unit tests were written for all actions fired to the API. The tests analysed the effect on the central storage of the application (Redux), to determine if the changes were aligned with what was expected. Similarly, unit tests were written on the backend, to test all significant CRUD endpoints in the API and database. For each unit test, the endpoint was called and the response from the database was compared to the expected response provided in the unit test definition. A test database was used when running the tests, as changes were reflected in the database [47]. The functional tests were conducted throughout the development on the application, to ensure functional effectiveness. After each feature was implemented, a test would be conducted to make sure it is effective.

6 RESULTS, FINDINGS AND DISCUCSSION

In this section, the results of the testing are analysed and discussed. The user tests assessed the usability, intuitiveness and functionality of the system, whilst the functional tests assessed only the functionality of the system. As mentioned previously, the usability tests were segmented into two phases, and will be discussed separately.

6.1 Phase 1 Usability Tests

6.1.1 Login Screen

All the users were able to easily login to the application. Two users identified that there was no feedback when incorrect credentials were entered. Another two users suggested the option to be able to view the password that they entered.

To address these concerns, an error message was displayed, informing the user when they enter in incorrect credentials. Additionally, a clickable eye icon was included allowing the user to make their password visible.

6.1.2 Dashboard

All the users were able to effectively navigate through the actions available in the dashboard. One user suggested it would be ideal if the patient details could be edited on the patient profile screen. Another user suggested that the names in patient pool should be ordered alphabetically. The overall consensus was that the dashboard was intuitive and that it centralised certain actions, increasing its accessibility. The users enjoyed the option to jump to the current location of the patient's visit.

To address the aforementioned issues, the patient details were made editable on the patient profile and the tables were ordered alphabetically by surname.

6.1.3 Sedation Appointments

All the users were able to effectively and easily create an appointment, check-in a patient and start a visit. One user created two appointments for the same patient on the same clinic, which shouldn't be allowed. Two users were unsure that the clinics displayed on the side were supposed to be selected. One user had accidently started a visit and couldn't recover from this error.

To address these issues, it was implemented so that adding two appointments for the same patient on the same clinic will return an error message, and the appointment won't be added. Additionally, the UI of the clinic cards were changed to make it appear clickable. Finally, an option was made available in the dashboard to cancel the visit and move it back to appointments.

6.1.4 Sedation Waiting Lists

All users were able to transfer the patients to the next phase and navigate to the information capture screen.

6.1.5 Information Capture

All users were able to efficiently capture the information for each respective phase. The users seemed to enjoy the interface and thought it was simple and easy to use. Certain users also enjoyed how it initially directed them to the relevant information tab (based on the phase), along with the option to switch between other tabs. An issue came about during two tests where the form information was loading the wrong data. Additionally, one user was unsure when an ICD10 was added, as it reflected behind the modal.

To rectify the information display issue, it was implemented so a spinner gets displayed whilst loading the data. Once complete, the component is displayed with the latest information loaded. For the ICD10 code issue, it was implemented so a tick gets displayed on the respective row when added.

6.1.6 General Practice Appointments

All users were able to add/edit appointments and then start the visit with ease. The users really enjoyed the calendar interface and the way in which the appointments can be added and moved around on the calendar. One user started a visit on a future date, which shouldn't be allowed. Another user suggested it might be nice to differentiate the appointments started from those that haven't, displayed on the calendar.

To address these issues, it was implemented such that the start visit button only gets displayed for appointments on the current date. Additionally, the appointments were changed to be colour coded on the calendar, where appointments are blue initially and get changed to green once the visit is started.

6.1.7 Admin Panel

All users managed to effectively add/edit and remove data as required.

6.1.8 General

A few users were unsure of whether their actions had been successfully completed or not. Additionally, users reported that there were missing confirmation dialogs in the system, which could result in unintentional actions being fired.

To address these issues, feedback was implemented for each action. Upon success of an action, a modal will appear on the bottom right of the screen to indicate the success. If an error had to occur, it will be displayed, informing the user. Additionally, confirmation modals were incorporated for every important action in the system.

6.2 Phase 2 Usability Tests

After incorporating all the above changes in the system, phase two user tests commenced. The three users that were tested in this phase managed to complete all the tasks given to them with ease. Despite the generality of the tasks, each user managed to autonomously

navigate through the system and complete each and every task successfully. Following these user tests, a final demo was conducted with the project proposer to evaluate the system as a whole. The feedback from the demo was highly positive and the proposer confirmed that all the requirements were successfully met. He was very impressed with the system and showed a sense of eagerness to deploy it into production.

6.3 Functional Tests

After each feature was developed, the relevant tests were analysed. If a test had to fail, necessary changes were made in the code until the test became successful. This testing methodology allowed the team to ensure that all features were working effectively, before moving on to implement new features in the backlog. At the end of development, all tests suites for both the frontend and the backend were successful, meaning that the results of the test matched what was expected. The frontend test results can be seen in Appendix F and the backend test results can be seen in Appendix G.

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The agile methodology used to manage the project development was successful with respect to aligning requirements and meeting project deadlines. The iterative approach and constant feedback allowed for change and clarification of requirements. The team was cohesive and worked very well together. Furthermore, the scrum framework along with the project management tools helped to manage collaboration and track progress.

The first phase of user tests surfaced some important UI/UX and functionality issues. It proved to be beneficial testing users with a Computer Science background, as they provided highly useful feedback. This was the ultimate reason for conducting phase one tests. After incorporating the feedback into the system, the second phase produced highly successful results. All users were able to successfully complete all tasks without any issues, concluding that the system is sufficiently usable. Additionally, all the unit tests conducted were successful, concluding that the entire system is functionally effective. Finally, the project proposer was happy with the system and confirmed that it met all requirements, concluding that the system was accepted by the client.

Conducting research into technologies and associated best practices proved to be highly beneficial in the development of the system. The is evident based off the efficient and effective end system that was provably developed. Furthermore, developing the system from scratch appeared to be the best approach in tailoring the system in line with the specific requirements provided. This is evident based off the client's feedback, and the fact that all requirements were successfully met. Additionally, it provided the team with great exposure and proved to be an immense learning experience.

The team was unable to test the system in production because the clinics were not running as a result of the Covid-19 pandemic. Justified predictions on how the system will achieve its aims in production therefore have to be made to prove the systems success. These predictions are based off the testing results. The system proved to be sufficiently usable in a fast paced environment and will thus lead to increased efficiency. Additionally, the system met all functional and non-functional requirements, and will thus be effective in streamlining operations and increasing quality of records. Finally, the system was designed so that all data is centralised, allowing for increased access to records as well as analysis. This being said, if the system had to be deployed, it is predicted to achieve all the aims specified in the introduction, deeming the project an overall success.

7.2 Future Work

The system was designed with the consideration for maintenance and future expansion. The next step for the project is to meet with the VMS board to discuss maintenance and evolution of the system. For further improvement, a general dashboard can be implemented to visualise a range of statistics for the respective clinic. This will allow the staff to track progress of the clinic and make decisions based off an overall analysis. Furthermore, integrated reporting can be implemented to generate graphical representations of all data stored in the database. This will allow VMS to derive insights into the current state of operations and use these insights to derive solutions that ultimately optimise performance.

8 REFLECTION AND RECOMMENDATIONS

The process of developing this application was a tough but rewarding experience. After the initial requirements specification meeting, the team was left feeling overwhelmed with the extensiveness of the requirements. Once we began to coherently conceptualise everything and design the system, we gained a much better understanding of the system and what is expected. It is highly recommended to make use of design tools to detail information about the system, as it significantly speeds up the implementation process and results in greater alignment of requirements. For an extensive system such as this, it is imperative to utilise best practices. This involved many hours of research and discussion. The team leveraged off connections in the software development industry to get advice on important aspects of the system. This proved to be highly beneficial and is recommended to anyone who wishes to replicate this work. The core functionality of the implementation proved to be the biggest challenge of the process, as it involved tapping into new territories. The team made use of popular tech stack for this reason. The abundance of community support significantly aided in conquering these challenges. Once the foundational system was developed, implementation progressed quickly and effectively. Using project management tools to outline tasks and track progress proved to have a significant effect on the efficiency of development. Another major challenge was the fact that the project proposer was extremely busy with the Covid response and was not often available to clarify certain requirements. This hindered the progress of the development as we often had to wait for his response before continuing. It is recommended to try and clarify all requirements early on, in the event of something unpredictable as such.

9 ACKNOWLEDGEMENTS

I would like to acknowledge my team members, Zachary Bresler and Chad Piha for their dedicated commitment and valuable contributions to this project. I would also like to thank our project supervisor, Aslam Safla and second reader, Melissa Densmore for their support and guidance during the project. Additionally, I would like to acknowledge Dr Moosa, who proposed the project and took the time to provide us with all requirements of the system. Finally, a thank you goes out to all the participants take partook in the user evaluations.

REFERENCES

[1] Vision Medical Suite. 2016. About us. Retrieved June 2, 2020 from http://www.visionmedicalsuite.co.za/about-us/

[2] Workpool. 2016. What is POPI? The Protection of Personal Information (POPI) Act explained. Retrieved May 29, 2020 from https://www.workpool.co/featured/popi

[3] Health Professions Council of South Africa. 2016. Confidentiality: Protecting and Providing Information. ACM 16, 1-5

[4] Al-Fedaghi, Sabah. 2011. Developing Web Applications. International Journal of Software Engineering and Its Applications. Article 5. 57-59. DOI: https://doi.org/10.1007/978-1-4302-3531-6_12.

[5] Emily. 2017. Web-Based Applications Offers Far Superior Advantages Over Desktop Ones. (August 2017). Retrieved April 20, 2020 from https://techpatio.com/2017/articles/webbased-applications-offer superioradvantagesdesktop

[6] Paweł Skólski. 2016. Single-page application vs. multiple-page application. (December 2016). Retrieved April 20, 2020 from https://medium.com/@NeotericEU/single-page-application-vs-multiple pageapplication-2591588efe58

[7] Leslie Bloom. 2019. Benefits of Using a Database. (June 2019). Retrieved April 22, 2020 from https://bizfluent.com/facts-4924693-benefits-using-database.html

[8] Sonam Khedar and Swapnil Thube. 2017. Real Time Databases for Applications. Vol. 4. International Research Journal of Engineering and Technology (IRJET) Conference Name: ACM Woodstock conferenceConference Short Name: WOODSTOCK'18Conference Location: El Paso, Texas USAISBN:978-1-4503-0000-0/18/06Year:2018Date:JuneCopyright Year:2018

[9] James Michael Stewart. 2011. *CompTIA Security+™: Review Guide, Second Edition* (2nd. ed.). Sybex.

[10] Raju Raghuwanshi. 2017. JWT (JSON Web Tokens) Are Better Than Session Cookies . (April 2017). Retrieved April 26, 2020 from https://dzone.com/articles/jwtjson-web-tokens-are-better-than-session-cookies

[11] Auth0: Introduction to JSON Web Tokens. Retrieved from https://jwt.io/introduction/

[12] Brian Childress. 2018. Securing Applications with Better User Authorisation (November 2018). Retrieved April 27, 2020 from https://medium.com/capital-one-tech/securing-applications-with-better-user-authorization-625ec07a7001

[13] OpenMRS. 2020. OpenMRS. Retrieved July 11, 2020 from https://github.com/openmrs

[14] Stackify. 2017. SOAP vs REST. (August 2017). Retrieved September 15, 2020 from https://stackify.com/soap-vs-rest/#:~:text=In%20addition%20to%20using%20HTTP,considered%20easier%20to%20work%20with.

[15] EssayCorp. 2017. ER Diagrams and its Benefits. (October 2015). Retrieved September 16, 2020 from https://blog.essaycorp.com/er-diagrams-and-its-benefits/

[16] Nick Mertens. n.d. RDBMS Basics: SQL Database Fundamentals. Retrieved September 16, 2020 from https://www.goskills.com/Development/Resources/RDBMS-basics

[17] Rubel Rana. n.d. *Hospital Management System*. Thesis. Uttara University Uttara, Dhaka.

[18] Microsoft Ignite. 2020. Description of database normalisation basics. (September 2020). Retrieved September 16, 2020 from https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

[19] Ahmet Ozlu. 2018. Mastering REST Architecture. (July 2018). Retrieved September 17, 2020 from https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc

[20] Thiago Pacheco. 2019. Designing a better architecture for a Node.js API. (November 2019). Retrieved September 17, 2020 from https://dev.to/pacheco/designing-a-better-architecture-for-a-node-js-api-24d

[21] Piero Borrelli 2019. The perfect architecture flow for your next Node.js project. (October 2019). Retrieved September 17, 2020 from https://blog.logrocket.com/the-perfect-architecture-flow-for-your-next-node-js-project/

[22] Janishar Ali 2020. Design Node.js Backend Architecture like a pro. (April 2020). Retrieved September 17, 2020 from https://afteracademy.com/blog/design-node-js-backend-architecture-like-a-pro

[23] Ataubu Prince 2018. Design Node.js Backend Architecture like a pro. (June 2018). Retrieved September 17, 2020 from https://dev.to/achowba/build-a-simple-app-using-noe-js-and-mysql-19me

[24] Code for Geek 2015. RESTful API using Node Express and MySQL (May 2015). Retrieved September 17, 2020 from https://codeforgeek.com/restful-api-node-and-express-4/

[25] Bezkoder 2020. Build Node.js REST APIs with Express and MySQL (September 2020). Retrieved September 17, 2020 from https://bezkoder.com/node-js-rest-api-express-mysql/

[26] Tony Branson 2017. The 5 Best Reasons to Choose MySQL (April 2017). Retrieved September 18, 2020 from https://dataconomy.com/2017/04/5-reasons-challenges-mysql/

[27] Simform n.d. Why and Where Should you Use React for Web Development. Retrieved September 18, 2020 from https://www.simform.com/why-use-react/

[28] Justyna Rachowicz. 2017. When, How and Why use Node.js as your Backend. Retrieved September 18, 2020 from https://www.netguru.com/blog/node-js-backend-use-react/

[29] Jodi Lebow. N.d. What is Scrum Methodology. Retrieved September 19, 2020 from https://digital.ai/resources/agile-101/what-is-scrum#:~:text=Scrum%20is%20an%20agile%20project,capability%20every%202%2D4%20weeks

[30] Node. 2011. What is NPM?. (August 2011). Retrieved September 20, 2020 from https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

[31] Express. N.d. CORS. Retrieved September 20, 2020 from https://expressjs.com/en/resources/middleware/cors.html

[32] Express. N.d. Morgan. Retrieved September 20, 2020 from http://expressjs.com/en/resources/middleware/morgan.html

[33] Express. N.d. Writing Middleware for Use in Express Apps. Retrieved September 20, 2020 from https://expressjs.com/en/guide/writing-middleware.html

[34] Baeldung. 2020. A Simple Guide to Connection Pooling in Java. (May 2020). Retrieved September 21, 2020 from https://www.baeldung.com/java-connection-pooling

[35] MySQL Tutorial. 2020. Connecting to the MySQL Database Server from Node.js. Retrieved September 21, 2020 from https://www.mysqltutorial.org/mysql-nodejs/connect/

[36] Janith Kasun. 2020. Authentication and Authorisation with JWTs in Express.js. Retrieved September 21, 2020 from https://stackabuse.com/authentication-and-authorization-with-jwts-in-express-js/

[37] M Fikri Setiadi. 2018. 7 Steps to Create Simple CRUD application using Node.js, Express and MySQL (November 2018). Retrieved September 22, 2020 from http://mfikri.com/en/blog/nodejs-mysql-crud

[38] W3 Schools. SQL JOINS. Retrieved September 22, 2020 from https://www.w3schools.com/sql/sql_join.asp

[39] Zulaikha Geer. 2019. How to Build a CRUD Application using Node.js and MySQL. (May 2019). Retrieved September 23, 2020 from https://medium.com/edureka/node-js-mysql-tutorial-cef7452f2762

[40] NPM. 2020. JOI. (August 2020) Retrieved September 23, 2020 from https://www.npmjs.com/package/joi

[41] Code for Geek. 2016. Asynchronous Programming in Node.js. (April 2016) Retrieved September 23, 2020 from https://codeforgeek.com/asynchronous-programming-in-node js/#:~:text=JavaScript%20is%20asynchronous%20in%20nature,the%20non%2Dblocking%20code%20execution.&text=In%20general%20if%20we%20execute,the%20one%20you%20are%20executing.

[42] JavaScript Info. 2020. Async Await. (August 2020) Retrieved September 24, 2020 from https://javascript.info/async-await

[43] Srebalaji Thirumalai. 2019. All you need to know about Promose.all. (April 2019) Retrieved September 24, 2020 from https://www.freecodecamp.org/news/promise-all-in-javascript-with-example-6c8c5aea3e32/

[44] NPM. 2015. Async-Each-Series. (September 2015) Retrieved September 24, 2020 from https://www.npmjs.com/package/async-each-series

[45] Jan L. Harrington. 2009. Client-Server Architecture. *Relational Database Design. Third Edition.*. Science Direct.

[46] IBM. N.d. REST API response codes and error messages. Retrieved September 25, 2020 from https://www.ibm.com/support/knowledgecenter/SSQP76_8.8.1/com.ibm.odm.dserver.events.ref/topics/ref_dse_restapi_responsecodes_errormsgs.html

[47] JEST. N.d. JEST. Retrieved September 29, 2020 from https://jestjs.io/

[48] Course Hero. N.d. Chapter 4 -Requirements – CS 360. Retrieved October 01, 2020 from https://www.coursehero.com/file/38302656/ch4requirements-pptx/

[49] Shanuj Mishra. 2019. The Importance Of Prototyping In Designing. Retrieved October 02, 2020 from https://uxdesign.cc/importance-of-prototyping-in-designing-7287c7035a0d#:~:text=Following%20are%20the%20fundamental%20reasons,focusing%20on%20important%20interface%20elements.

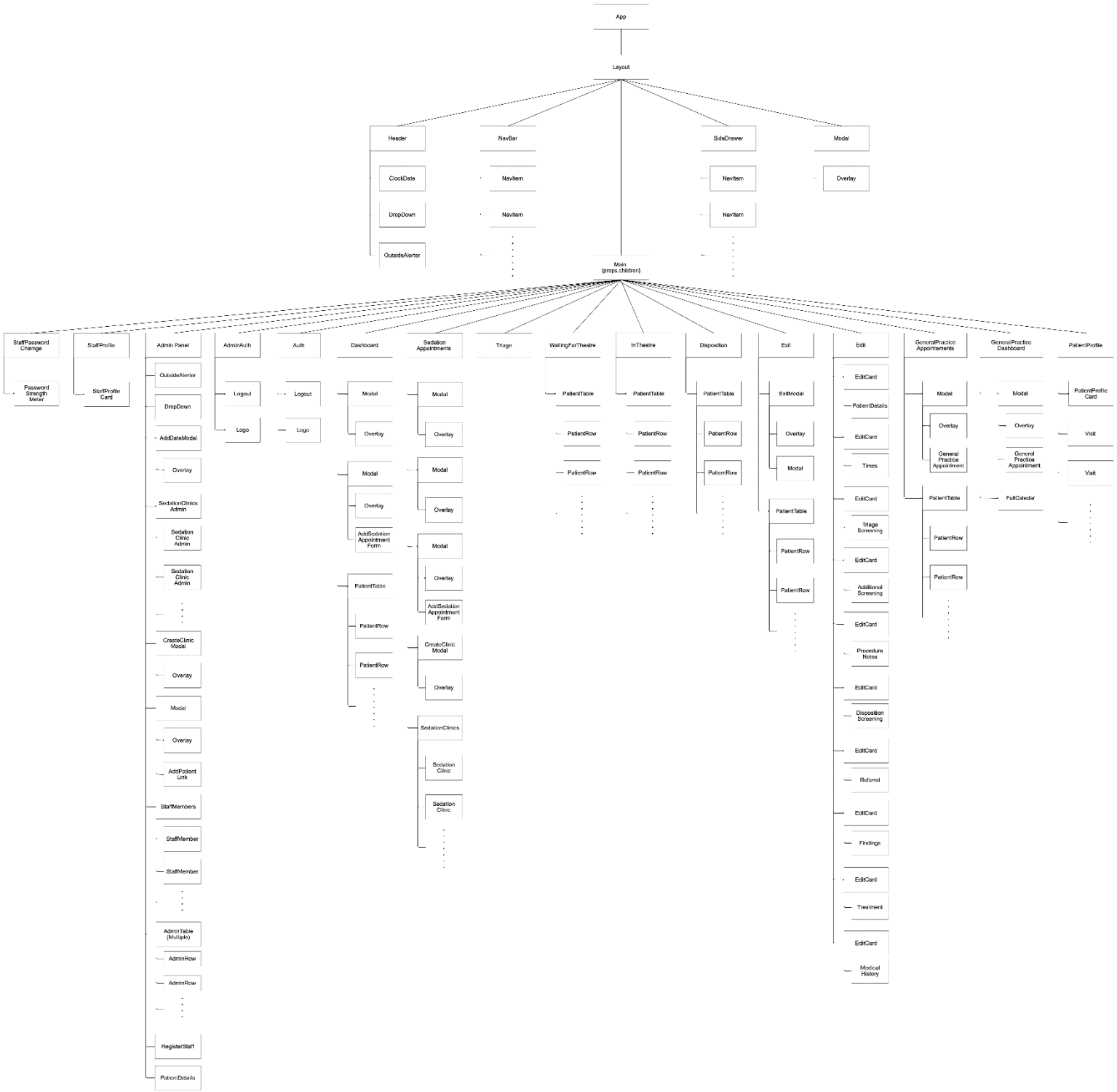
[50] Guru 99. N.d. Database Design Tutorial. Retrieved October 02, 2020 from https://www.guru99.com/database-design.html#:~:text=Database%20Design%20is%20a%20collection,of%20enterprise%20data%20management%20systems.&text=The%20main%20objectives%20of%20database,of%20the%20proposed%20database%20system.

SUPPLEMENTARY INFORMATION

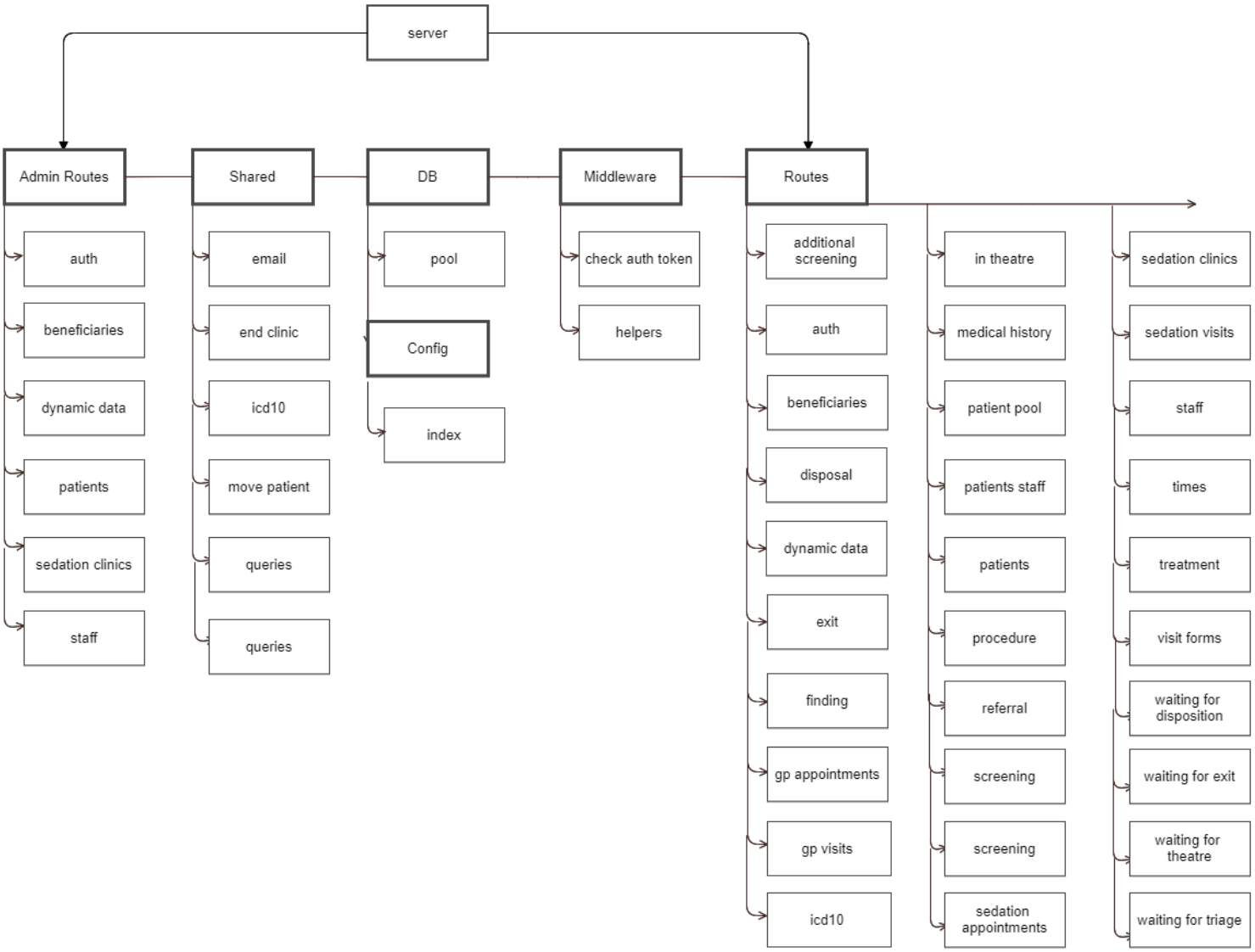
APPENDIX A:

USE CASE DIAGRAM

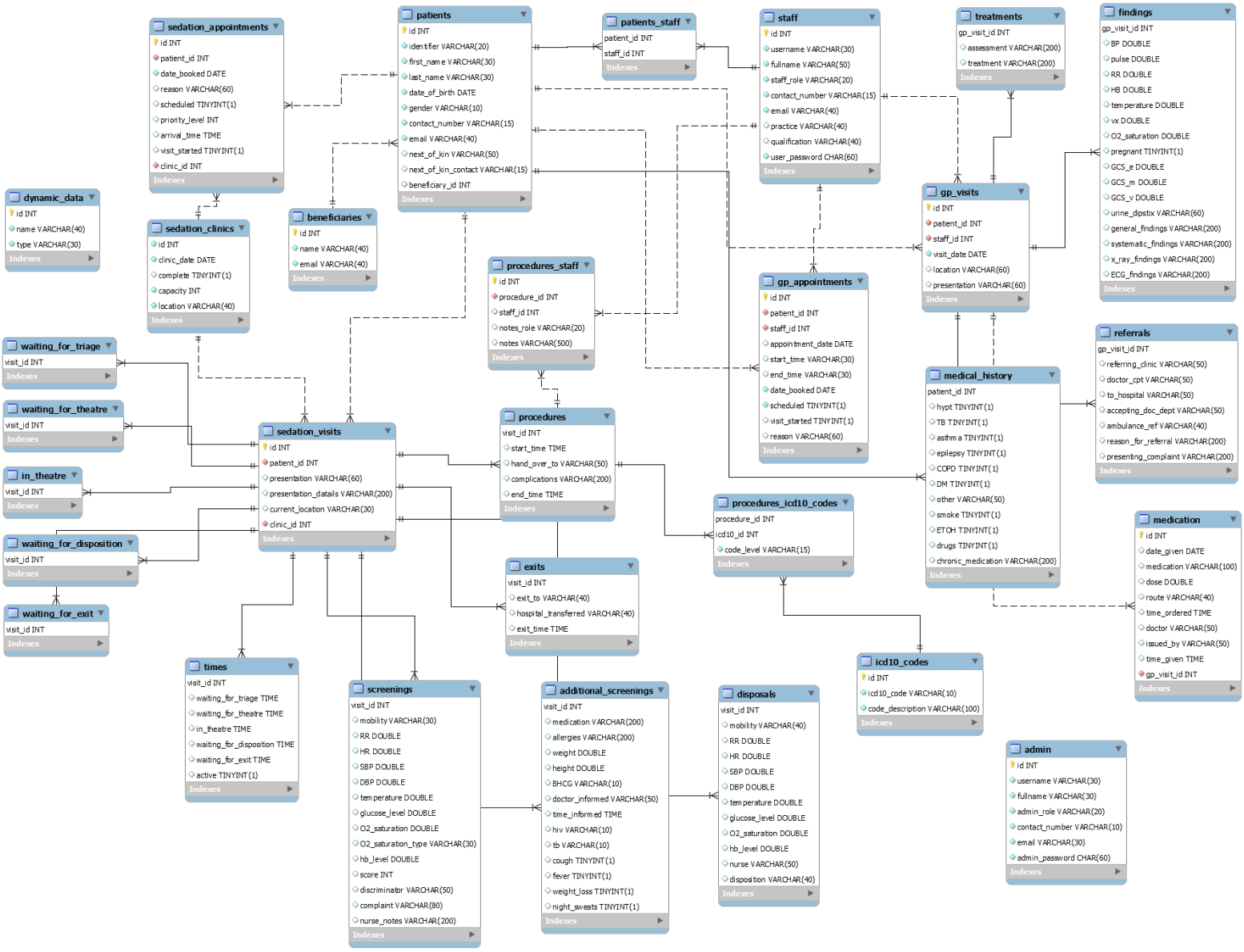
APPENDIX B:
FRONTEND COMPONENT TREE



APPENDIX C:
BACKEND COMPONENT TREE



APPENDIX D:
ENTITY RELATIONSHIP DIAGRAM



APPENDIX E:
USER EVALUATION TASKS

1. Log in to admin panel
2. Register staff member
3. Add dynamic data for everything
4. Add a clinic for today
5. Link patients to yourself
6. Navigate to your email, and change your password
7. Register a patient
8. Create an appointment with recently registered patient
9. If you wanted to increase the capacity of the clinic, what would you do?
10. This new patient walks in for his appointment. Please check him in.
11. The staff is now for him. Begin his visit.
12. Locate the location of the visit
13. Fill in the appropriate information for current phase
14. Once done, transfer the patient to the next phase
15. Repeat until the patient until the patient is ready to be exited
16. When ready, exit the patient from the visit
17. View patient profile
18. Edit info from the visit just ended
19. Redirect to general practice
20. Create an appointment for today
21. Start the visit for that patient
22. Edit all info for that visit
23. Logout

APPENDIX F:

FRONTEND UNIT TEST RESULTS

```
PASS src/store/tests/auth.test.js
PASS src/store/tests/waitingForTheatre.test.js
PASS src/store/tests/gpVisitForms.test.js
PASS src/store/tests/staff.test.js
PASS src/store/tests/beneficiaries.test.js
PASS src/store/tests/patientPool.test.js
PASS src/store/tests/triage.test.js
PASS src/store/tests/visitForms.test.js
PASS src/store/tests/dynamicData.test.js
PASS src/store/tests/disposition.test.js
PASS src/store/tests/sedationClinics.test.js
PASS src/store/tests/sedationAppointments.test.js
PASS src/store/tests/exit.test.js
PASS src/store/tests/gpAppointments.test.js
PASS src/store/tests/inTheatre.test.js
PASS src/store/tests/patient.test.js

Test Suites: 16 passed, 16 total
Tests:       76 passed, 76 total
Snapshots:   0 total
Time:        3.914s
Ran all test suites.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/beneficiaries.test.js
  beneficiary reducer
    ✓ should return the initial state (4ms)
    ✓ fetch beneficiaries
    ✓ add beneficiary (1ms)
    ✓ edit beneficiary (1ms)
    ✓ delete beneficiary

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.73s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/auth.test.js
  auth reducer
    ✓ should return the initial state (1ms)
    ✓ should store the token upon login
    ✓ should delete the token upon logout (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.428s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/staff.test.js
  staff reducer
    ✓ should return the initial state (2ms)
    ✓ fetch staff members
    ✓ edit staff member (1ms)
    ✓ delete staff member
    ✓ add staff patient link
    ✓ delete staff patient link (1ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        2.001s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/sedationAppointments.test.js
  sedation appointments reducer
    ✓ should return the initial state (1ms)
    ✓ add sedation appointment (1ms)
    ✓ delete sedation appointment (1ms)
    ✓ edit sedation appointment

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.546s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/gpAppointments.test.js
  general practice appointments reducer
    ✓ should return the initial state (2ms)
    ✓ add general practice appointment (1ms)
    ✓ delete general practice appointment (1ms)
    ✓ edit general practice appointment

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.531s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/visitForms.test.js
  visit forms reducer
    ✓ should return the initial state (1ms)
    ✓ fetching visit forms
    ✓ edit patient details (1ms)
    ✓ edit triage screening
    ✓ edit times
    ✓ edit additional screening info (1ms)
    ✓ edit procedure information info
    ✓ edit disposition screening (1ms)
    ✓ add icd10 code
    ✓ delete primary icd10 code (1ms)
    ✓ delete secondary icd10 code
    ✓ delete additional icd10 code (1ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        0.646s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/patientPool.test.js
  patient pool reducer
    ✓ should return the initial state
    ✓ register patient (1ms)
    ✓ remove patient from patient pool (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.553s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/dynamicData.test.js
  dynamic data reducer
    ✓ should return the initial state (1ms)
    ✓ fetch dynamic data
    ✓ add dynamic data locations (1ms)
    ✓ add dynamic data hospitals (1ms)
    ✓ add dynamic data exit_to
    ✓ edit dynamic data locations (1ms)
    ✓ edit dynamic data hospitals (1ms)
    ✓ edit dynamic data exit_to
    ✓ delete dynamic data locations (1ms)
    ✓ delete dynamic data hospitals
    ✓ delete dynamic data exit_to (1ms)

Test Suites: 1 passed, 1 total
Tests:       11 passed, 11 total
Snapshots:   0 total
Time:        0.552s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/sedationClinics.test.js
  sedation clinics reducer
    ✓ should return the initial state (3ms)
    ✓ add sedation clinic
    ✓ delete sedation clinic
    ✓ edit sedation clinic

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.175s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/patient.test.js
  patient reducer
    ✓ should return the initial state (1ms)
    ✓ fetch all patient visits (1ms)
    ✓ fetch all patient general practice visits

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.438s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/gpVisitForms.test.js
  gp visit forms reducer
    ✓ should return the initial state (3ms)
    ✓ fetching visit forms (1ms)
    ✓ edit referral information
    ✓ edit finding information
    ✓ edit medical history information (1ms)
    ✓ edit treatment information

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        3.064s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

```
PASS src/store/tests/triage.test.js
  waiting for triage reducer
    ✓ should return the initial state (2ms)
    ✓ move patient to waiting for theatre and remove from waiting from triage reducer (1ms)
    ✓ move patient backward to appointment list and remove from waiting for theatre reducer

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.554s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

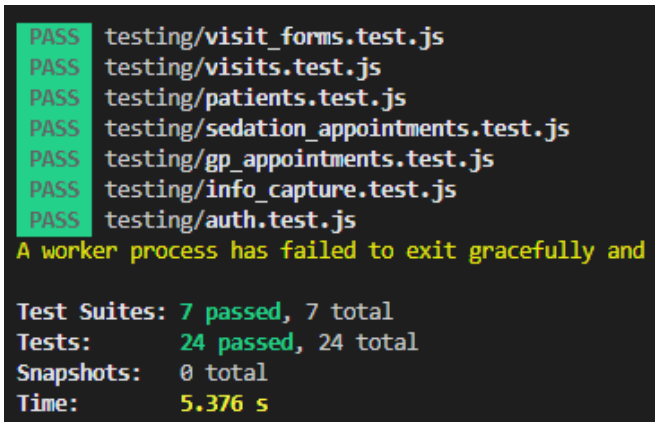
```
PASS src/store/tests/waitingForTheatre.test.js
  waiting for theatre reducer
    ✓ should return the initial state (3ms)
    ✓ move patient to in theatre and remove from waiting for theatre reducer
    ✓ move patient backward to waiting for triage and remove from waiting for theatre reducer (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.953s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

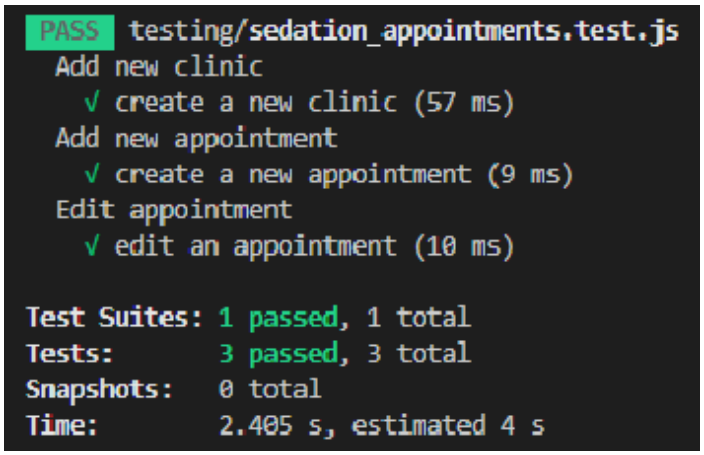
```
PASS src/store/tests/inTheatre.test.js
  in theatre reducer
    ✓ should return the initial state (2ms)
    ✓ move patient to waiting for disposition and remove from in theatre reducer
    ✓ move patient backward to waiting for theatre and remove from in theatre reducer (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.471s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

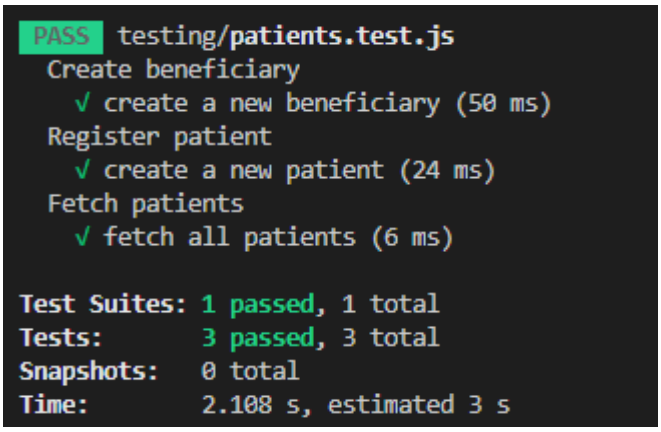
```
PASS src/store/tests/disposition.test.js
  waiting for disposition reducer
    ✓ should return the initial state
    ✓ move patient to waiting for exit and remove from waiting from disposition reducer (1ms)
    ✓ move patient backward to in theatre and remove from waiting for disposition reducer

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.554s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

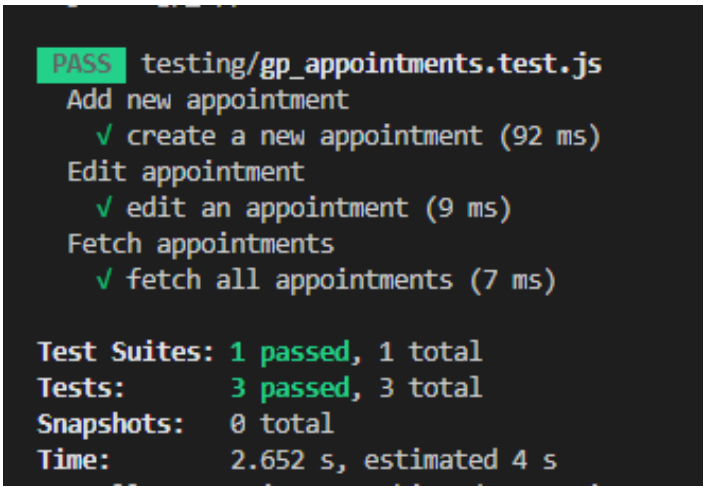
```
PASS src/store/tests/exit.test.js
  waiting for exit reducer
    ✓ should return the initial state (2ms)
    ✓ exit patient from sedation clinic and remove from waiting from exit reducer
    ✓ move patient backward to waiting for disposition and remove from waiting for exit reducer (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.476s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

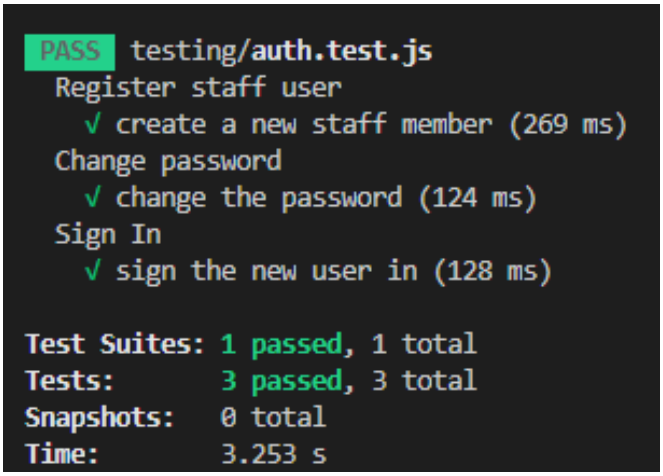
APPENDIX G:
BACKEND UNIT TEST RESULTS

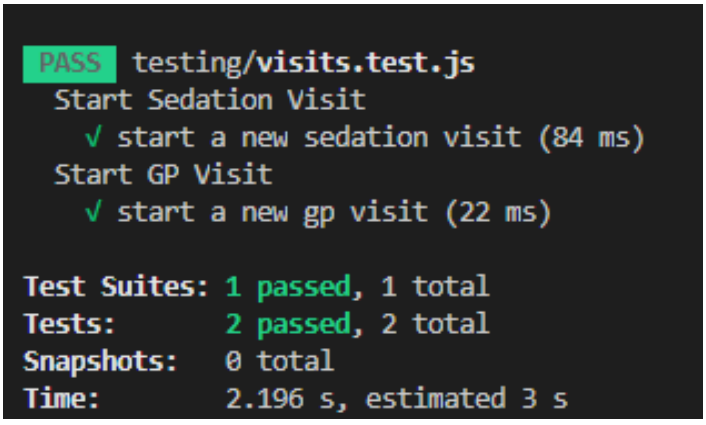












```
PASS testing/info_capture.test.js
  Capture screening
    √ capture screening info (56 ms)
  Additional screening
    √ capture additional screening info (7 ms)
  Procedure
    √ capture procedure info (17 ms)
  Capture disposal
    √ capture disposal info (6 ms)
  Capture exit
    √ capture exit info (4 ms)
  Capture medical history
    √ capture medical history info (4 ms)
  Capture treatment
    √ capture treatment info (18 ms)
  Capture finding
    √ capture finding info (6 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.528 s, estimated 3 s
```

```
PASS testing/visit_forms.test.js
  Fetch Sedation Clinic Forms
    √ fetch sedation clinic forms (50 ms)
  Fetch GP Forms
    √ fetch GP forms (10 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.999 s, estimated 4 s
```

APPENDIX H:
INTERFACE DESIGN

Sedation Clinic:

Figure 1: Application login screen

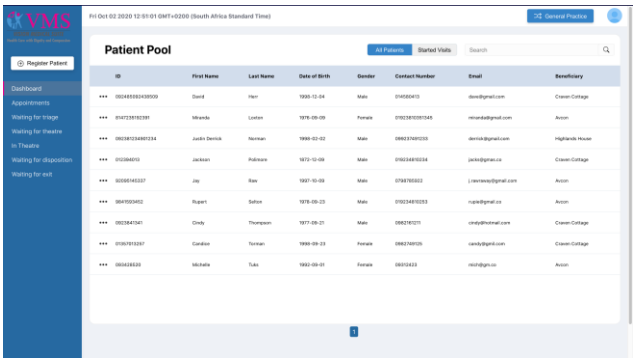


Figure 2: Dashboard

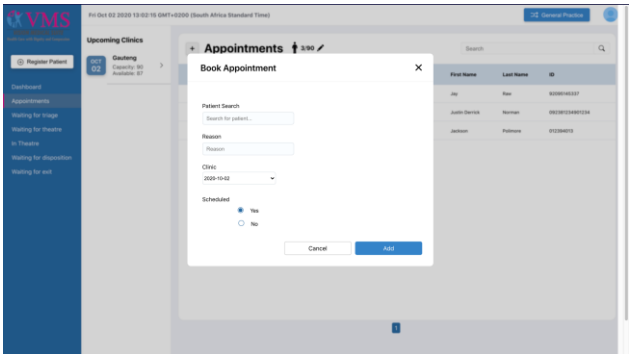


Figure 3: Creating clinic appointment

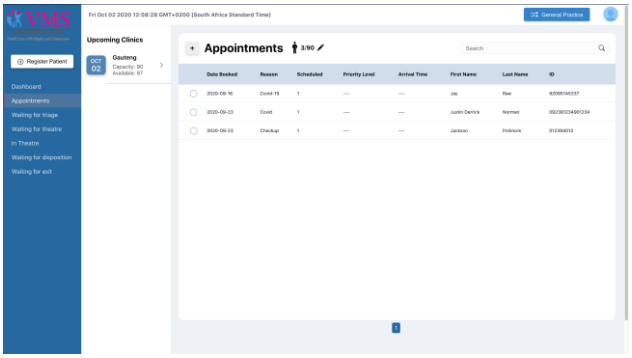


Figure 4: Sedation clinic appointments

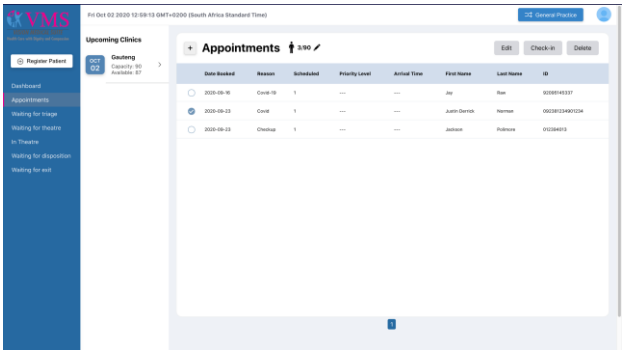


Figure 5: Editing, Checking-in and delete appointment function

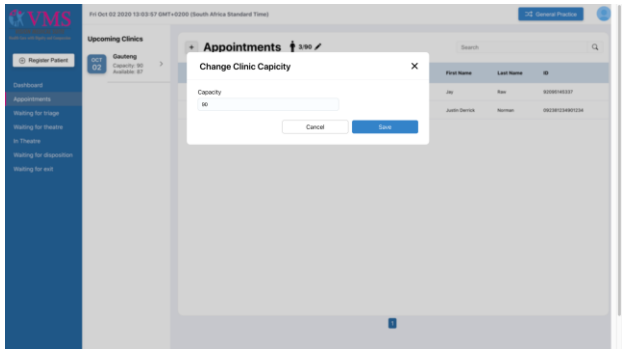


Figure 6: Edit sedation clinic capacity

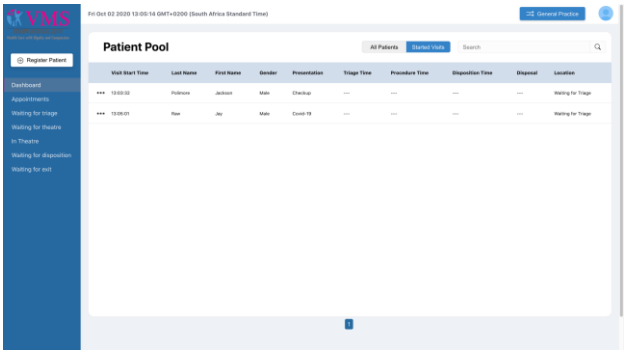


Figure 7: Sedation clinic ongoing visits

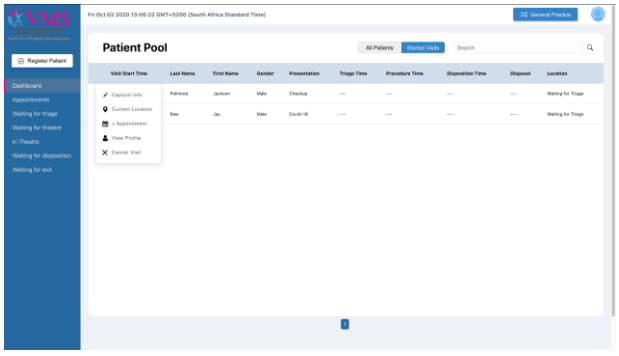


Figure 8: Patient actions

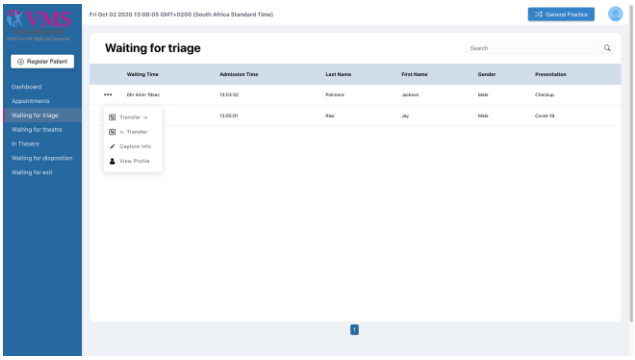


Figure 9: Waiting for triage table with patient actions

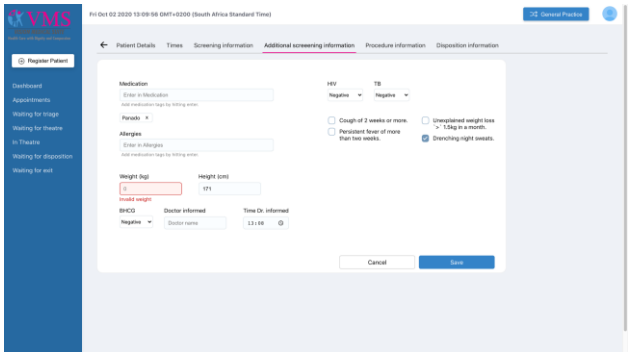


Figure 10: Additional screening form (with validation)

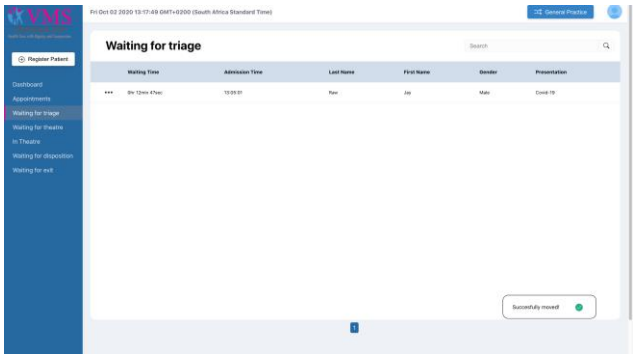


Figure 11: Moving patient between phases feedback modal

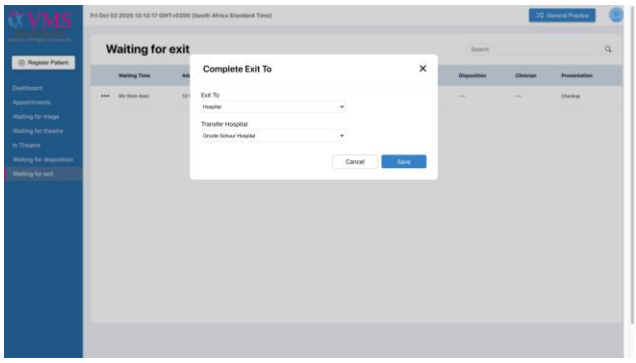


Figure 12: Completing sedation clinic visit

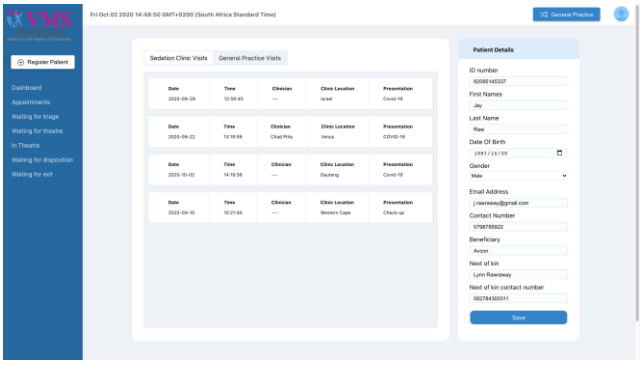


Figure 13: Patient Profile

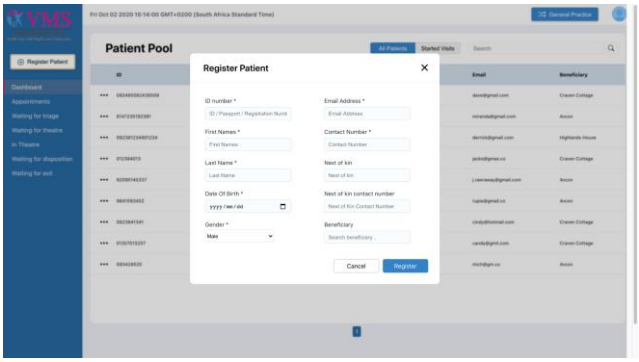


Figure 14: Register Patient

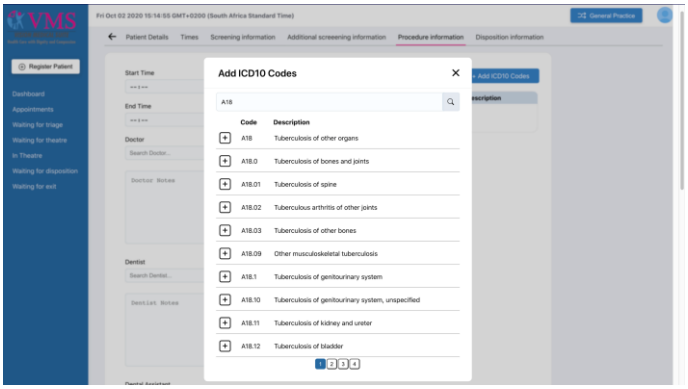


Figure 15: Add ICD10 code modal

General Practice (GP):

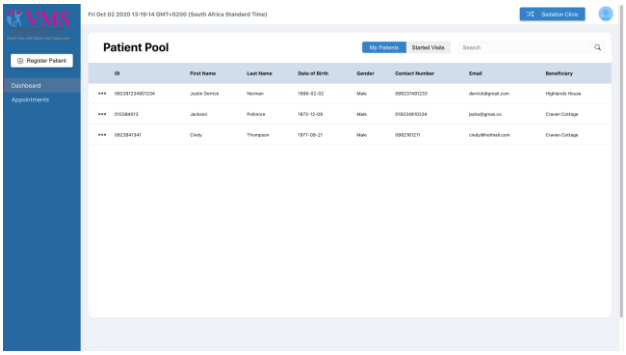


Figure 16: GP Dashboard

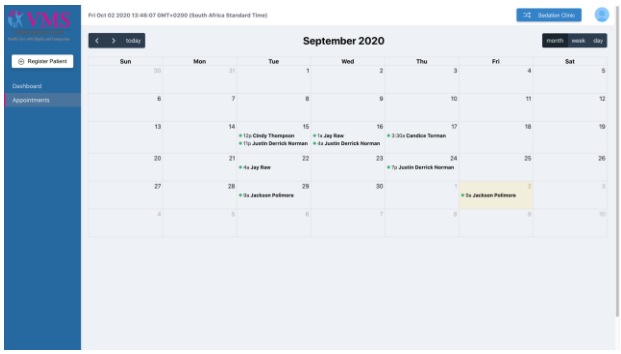


Figure 17: GP appointments

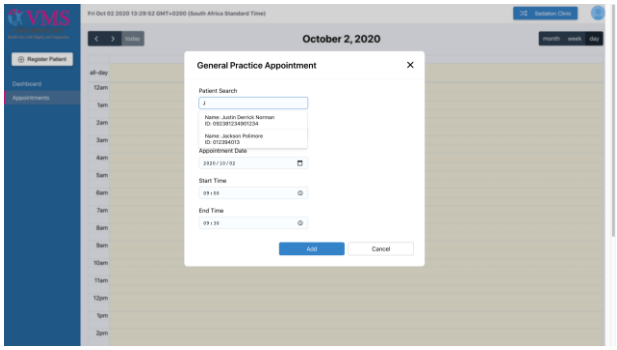


Figure 18: Create GP appointment (with patient list dropdown)

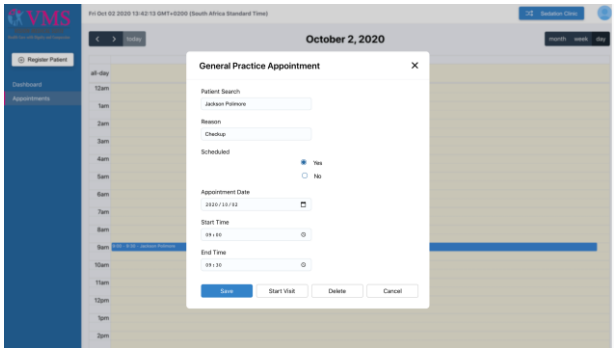


Figure 19: Appointment actions

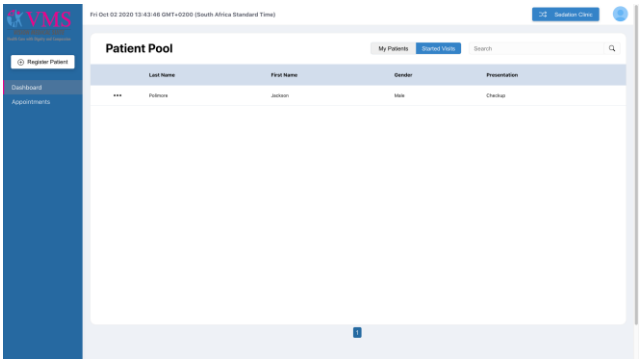


Figure 20: Patients undergoing their appointment

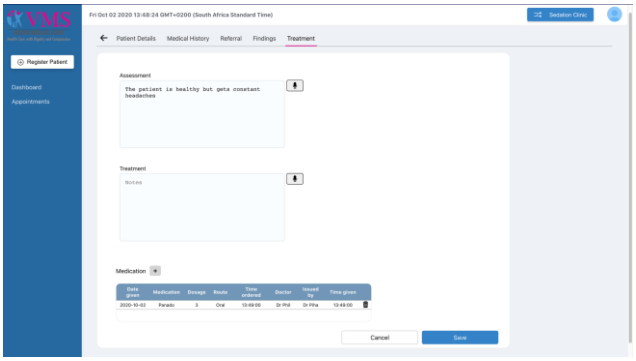


Figure 21: GP Treatment Information Capture (w/ Voice to text)

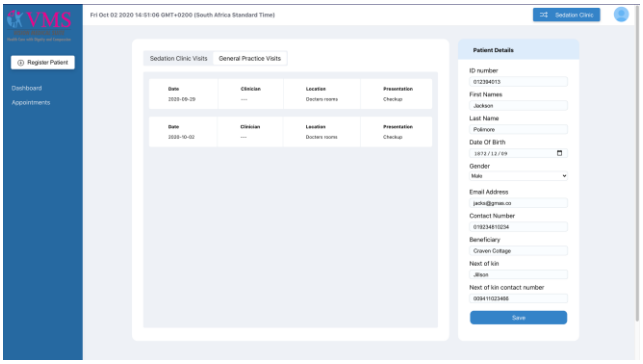


Figure 22: GP patient profile

Admin Panel

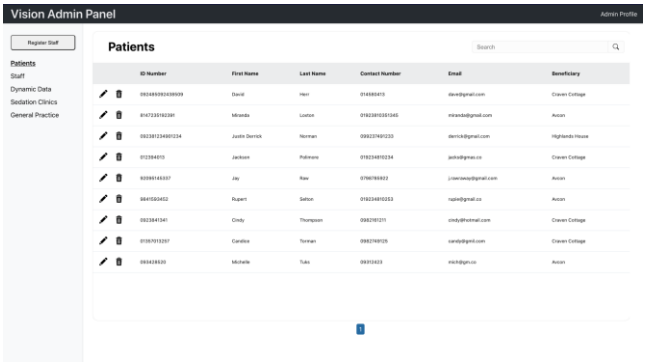


Figure 23: List of all patients

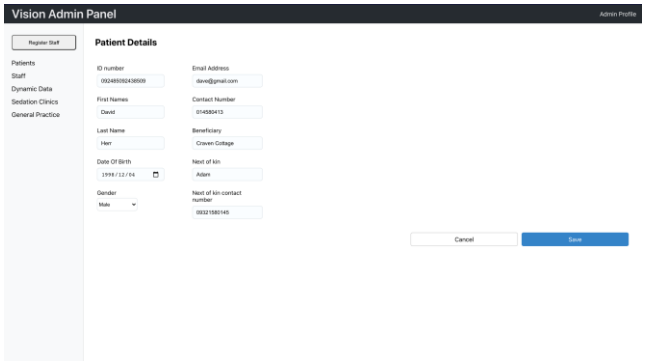


Figure 24: Editing patient details

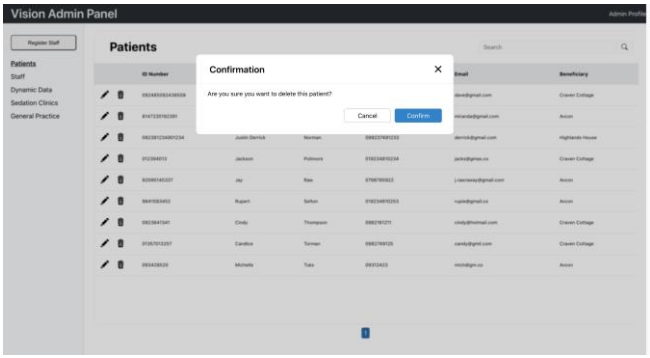


Figure 25: Confirmation modal

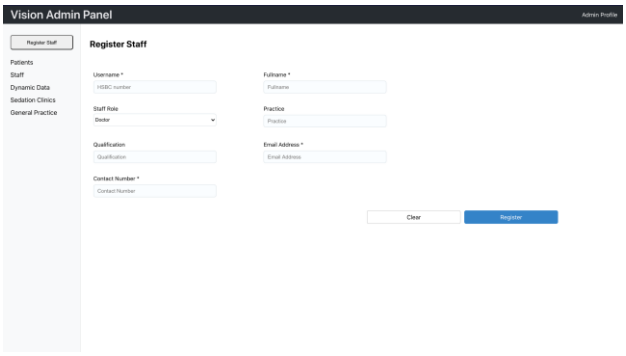


Figure 26: Register staff

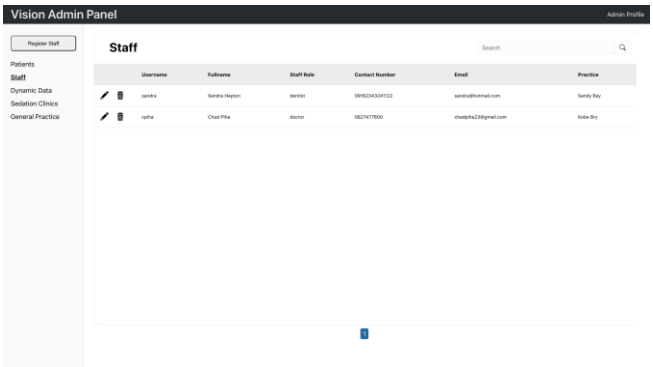


Figure 27: List of all staff members

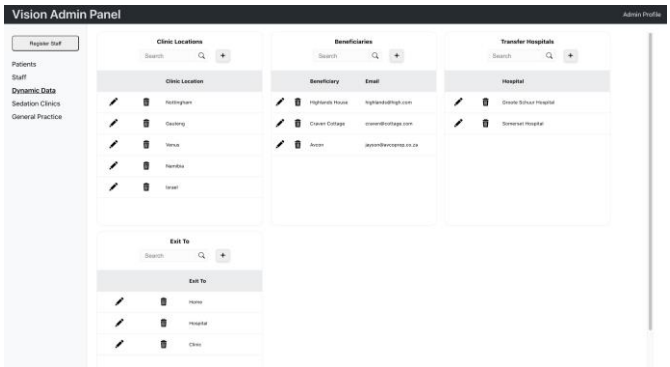


Figure 28: Dynamic data

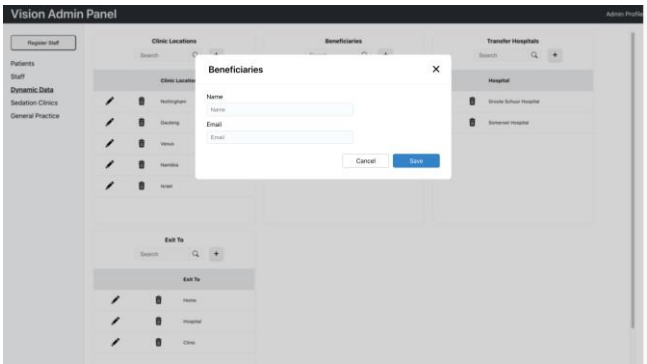


Figure 29: Adding beneficiaries dynamic data

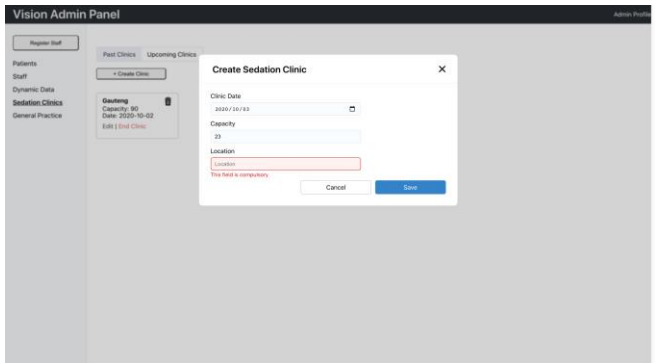


Figure 30: Create sedation clinic (with validation)

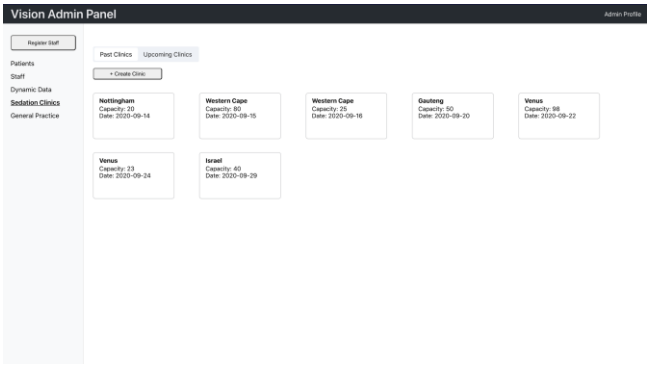


Figure 31: Past sedation clinics

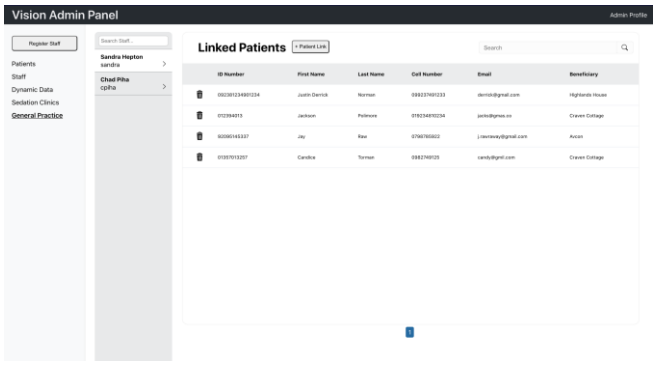


Figure 32: Patient-staff link

Mobile application sedation clinic

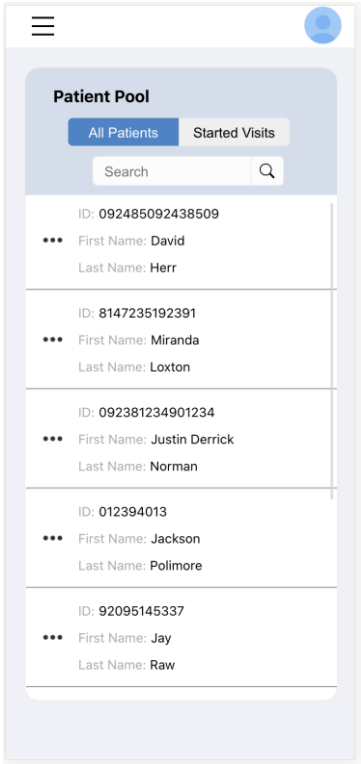


Figure 33: List of all patients table

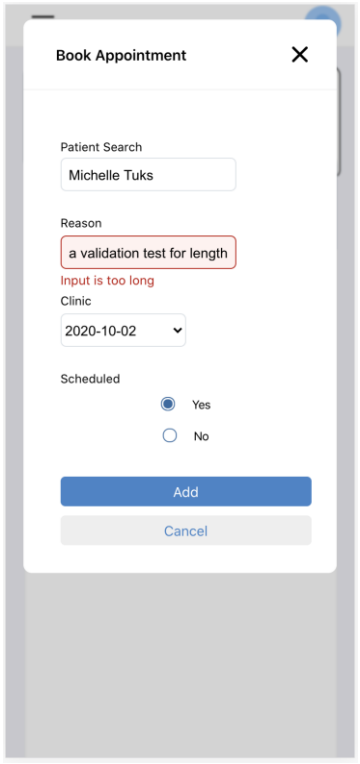


Figure 34: Book appointment (w/ validation)

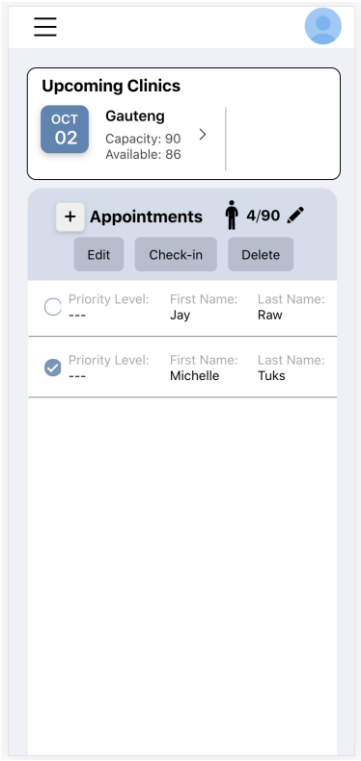


Figure 35: Appointments actions

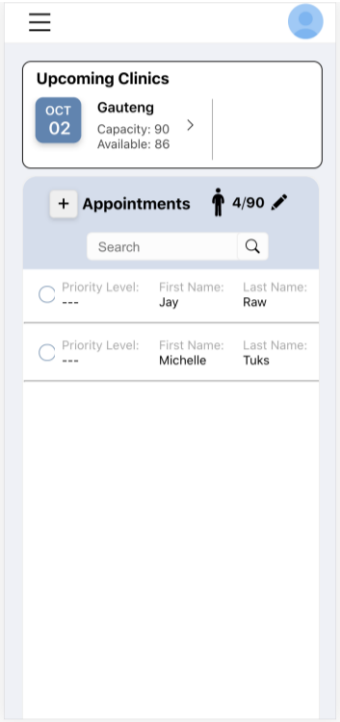


Figure 36: Appointments

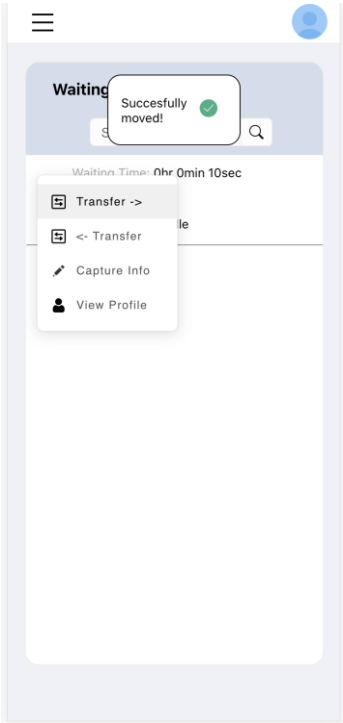


Figure 37: Feedback modal

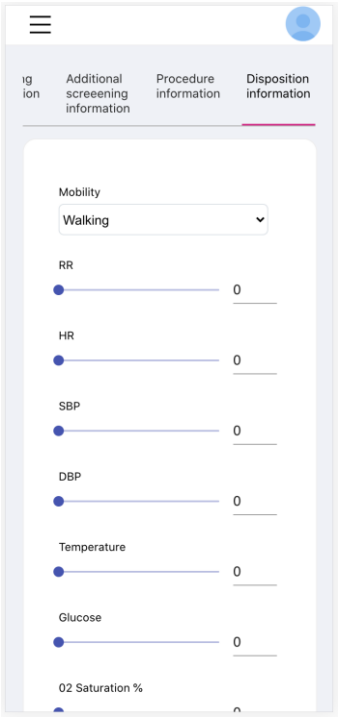


Figure 38: Mobile form

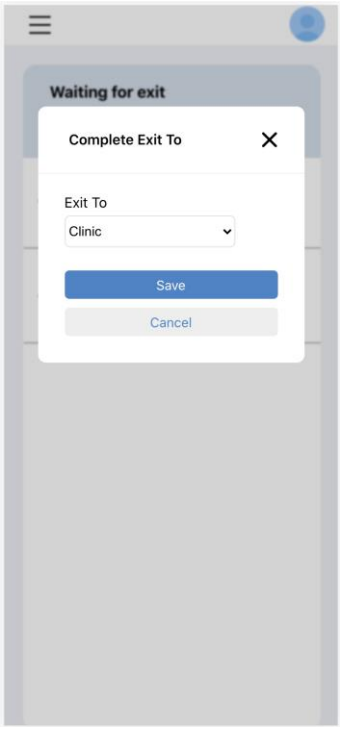


Figure 39: Exit to modal

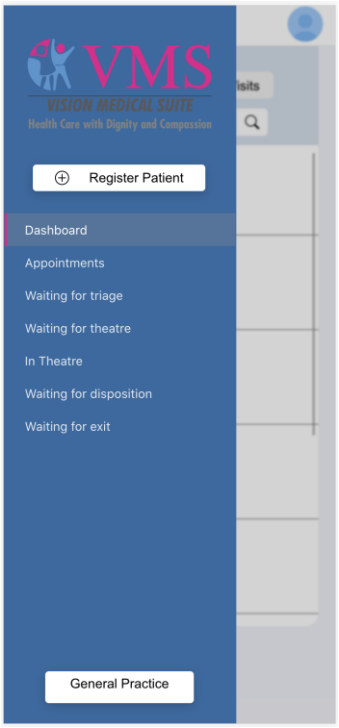


Figure 40: Expanded hamburger menu

Mobile General Practice:

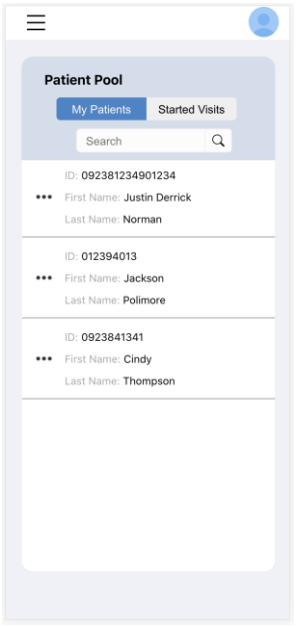


Figure 41: Linked patients table

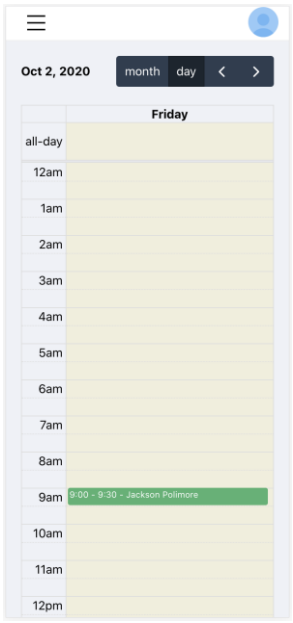


Figure 42: Appointments

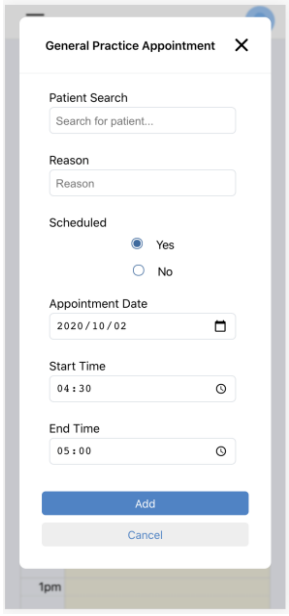


Figure 43: Book appointment.

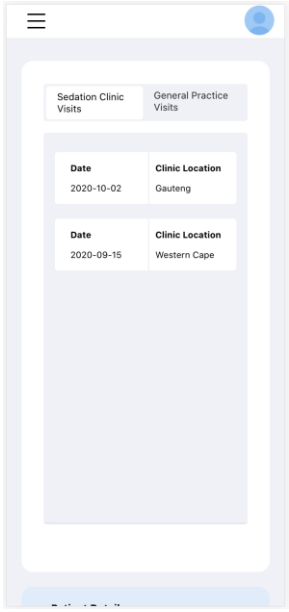


Figure 44: Patient profile

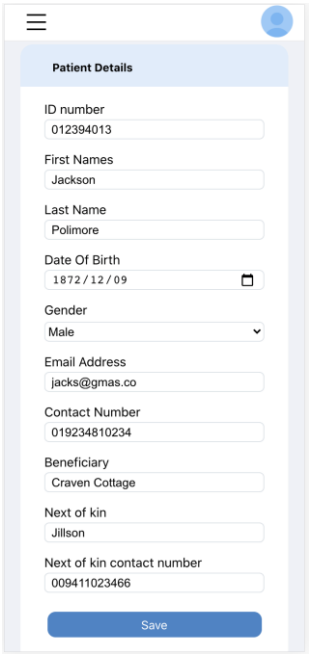


Figure 45: Patient profile (cont.)

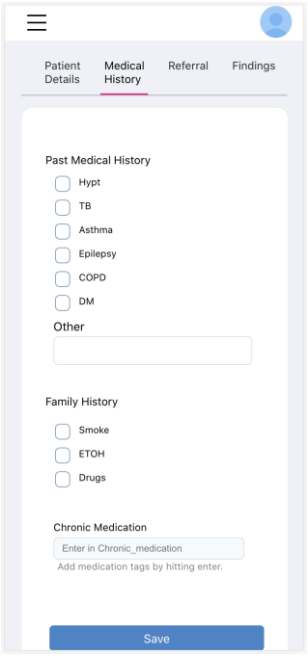


Figure 46: GP form