UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours
Final Paper 2020

Title: Software Implementation of a Healthcare Management System

Author: Zachary Bresler (BRSZAC002)

Project Abbreviation: Vision

Supervisor: 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**	

Software Implementation of a Healthcare Management System

Zachary Bresler
Computer Science Department
University of Cape Town
Cape Town, South Africa
BRSZAC002@myuct.ac.za

ABSTRACT

Vision Medical Suite is a Non-Profit Company that helps provide healthcare to vulnerable South Africans. The company currently uses manual systems to help manage their staff and patients which is inefficient and detrimental to their mission of providing the best quality healthcare possible.

This paper presents a solution in the form of a web application which is accessible via desktop and mobile devices. The system provides an advanced tool to its users which enables them to easily manage staff, patients and related clinic and general practice information. All information and data are stored and tracked for ease of access and for analysis. Healthcare professionals are able to clearly manage patients from beginning to end of their journey and are able to keep track of patients' history and future interactions. This paper describes the system in depth to provide a clear understanding of the software solution for healthcare professionals in clinics and general practices.

The web application is able to help Vision Medical Suite manage their operations as well as enable healthcare professionals to manage patients feeding into the clinic and managing patients transferred after the clinic.

CCS CONCEPTS

• Software and its engineering → Software organization and properties • Information systems → Information systems applications

KEYWORDS

React.js; Management systems; Healthcare; Web Application; Agile Development; State; Components, Actions and Reducers

1 INTRODUCTION AND BACKGROUND

1.1 Background

The project was proposed by Vision Medical Suite (VMS) which is a Non-Profit Company (NPC) that helps provide healthcare to vulnerable South Africans. The company provides free medical, dental, and other health services to its affiliated beneficiaries – there being around forty beneficiaries at the time of writing this paper. [44] Their biggest contribution is running a free once a month clinic where the company provides its services to the beneficiaries. At the once a month clinic they also allow walk-ins to vulnerable people in need of healthcare services. In order to manage the system, they follow a pipeline which manages patients through five phases (*Waiting for Triage; Waiting for Theatre; In Theatre; Waiting for Disposition; and Waiting for Exit*) each with its own medical form.

Medical clinicians volunteer their time to VMS to ensure that the clinics can in fact continue their operations. The volunteers that have private practices will sometimes refer patients back to their practice for check-ups or additional healthcare, after the clinic has ended.

1.2 Problem

VMS currently uses manual systems (pen, and paper) to help manage their staff (volunteers) and patients. They capture all relevant information using a paper-based system which is inefficient and detrimental to their mission of providing the best quality healthcare to vulnerable South Africans. Hence, it makes it difficult for VMS to operate as it is still using an antiquated system of information capturing and management. When using a paper-based system it is easy for information and documents to get lost; the overhead that comes with dealing with large amounts of paperwork reduces the ability to provide the best quality healthcare to its patients; staff are having to manually capture important information about their patients which is error-prone and inefficient; storing the files becomes a problem in terms of storage and security; and it becomes difficult to share and transfer patient information with different doctors for referrals and check-ups.

1.3 Aim formulation

Due to the problems facing VMS there is a need for a solution to help streamline its operations as well as something that can provide it with ease of use, easy access to information and having the ability to store and manage sensitive information securely. This paper presents a solution to its problems in the form of a web application. The aim was to create a fully functional, custom full stack application that would ultimately improve the operations of VMS by enabling it to easily manage patients and staff as well as manage its respective information. The interface needs to be simple, clean, and easy to use. The software needs to be fast, secure and maintainable. Furthermore, it needs to meet the stipulated requirements set out during the requirements analysis phase.

This paper is relevant to Computer Science in terms of the software development processes and practices that were followed. Additionally, the architecture and formulation of the software is relevant to Computer Science.

1.4 Application Characteristics

The main characteristics of the application encompass features that are essential to the management of a healthcare clinic. Features include, appointment management; tracking of patients through the respective phases; information capturing for respective phases; a user interface (UI) that is easy to use; admin functionalities; and enabling a user to access their own practice in addition to the sedation clinic.

1.5 Ethical, Professional and Legal Issues

1.5.1 Legal Issues. This paper and the web application developed, deals with sensitive personal information, hence it is important that the *Protection of Personal Information Act 4 of 2013* [1] is abided by to ensure protection of staff and especially patients' information.

Node.js, React, Express, MySQL, and Adobe XD are free to use in commercial use and hence there are no legal issues with using the software for development of the application. [2,3,4,5,6]

1.5.2 Ethical Issues. Due to COVID-19 is was essential that the testing conducted was not done in person as we wanted to reduce the spread of the virus and reduce the possibility of infections.

1.5.3 Professional Issues. The software does not form ownership of the group but rather will fall ownership to Vision Medical Suite (VMS) and UCT.

1.6 Structure of Report

This paper will describe related work with a focus on comparisons to similar systems and comparisons of software available for use. An analysis of the requirements gathered for the system is provided and included in this a detailed description of the design processes and design thinking. The approach and implementation make up the bulk of the paper and will clearly outline and explain the approach that was taken in the development of the system and how the core features were implemented. There will be a focus on state management with the use of Redux and how the system followed key principles of React. The system deals with sensitive information, hence the security implementation is discussed.

The different testing methods will be provided, including the explanation of why testing was conducted. Following this, the results and discussion will outline the results of the testing, evaluating the system, and determining the success of the tests. An additional reflection on the development of the system will be provided – allowing a clear understanding of what processes worked and which did not. Finally, conclusions will be made from the results and whether the system is successful in achieving the aims set out. A brief discussion of future work of this project will be provided while giving some additional features that could improve the system for deployment.

2 RELATED WORK

2.1 Similar Systems

2.1.1 Open Source Software. OpenMRS is an open source medical record system which allows for easy access of information as well as storage of patient and staff details [7]. Due to it being open source it is possible to make changed to the system. The software has to cater to many different people and thus has to encompass a large range of features. Therefore, the code base is rather bloated with unrequired features. [7]

Another system identified is HospitalRun, which is an open source software focusing on providing a tool to hospitals in developing areas which do not have the resources to develop or purchase a management system. Its biggest advantage is being able to run offline, which is essential in developing countries with intermittent internet connection and limited electricity. [8]

The reason for not using open source software is because it would mean having to learn the existing software thoroughly and having to compromise on certain sections of the software. A significant issue with these systems is that they do not provide a pipeline for a clinic – such as being able to move patients through different phases and enabling clinicians to manage the patients in each phase. [7, 8]

2.1.2 Paid Software. TeamDesk is a paid system that helps manage stored information for hospitals. It is a customizable web application that enables users to manage different tables of their database. A disadvantage to this software is that there is no mobile version, thus while clinicians are working, they cannot update information on the go. This software also does not have a pipeline to manage patients through a clinic. Additionally, the software has too many features for a small company such as VMS. [9] Furthermore, VMS would need to purchase the software when they could be spending money on resources and materials for the clinic.

As a group we decided that creating the web application from scratch would not only benefit VMS but would also benefit the group. VMS will be receiving a fully custom web application catered to their specifications, additionally, as students we are able to gain experience in developing a full stack application for an NPC that will hopefully be used to benefit the lives of vulnerable South Africans.

2.2 Technologies and Software

A web application was intended to be developed and hence on the frontend a comparison of the different frameworks is provided. The three main JavaScript frameworks are React.js, Angular, and Vue.js.

	React.js	*Angular*	*Vue.js*
Performance	High performance due to virtual DOM. [28, 30]	High performance but is bloated which can hinder performance. [31]	High performance due to it being so lightweight. It also uses a virtual DOM. [30, 32]
Learning Curve	Low. Not as easy as Vue.js but easier than Angular. [28]	High [28, 30]	Low [29]
Size	Medium, around 40KB. [31]	Large, around 140KB. [31]	Small, around 20KB. [31]
Community	Large community as it is a very popular framework. [15]	Large community as it is the most mature framework. [31]	Due to less people using it, there are less resources available. [30]

Figure 1: Table comparing the top three JavaScript front-end frameworks.

React was the best choice for this development project as it has a small learning curve, is powerful and has a large community. The team has experience in developing web applications using React.js and hence we could produce high quality results faster.

3 REQUIREMENTS ANALYSIS AND DESIGN

3.1 Requirements Analysis

To clearly determine the scope of the project it was essential to conduct meetings with the project proposer to outline the software requirements of the system. The purpose of the initial meeting with the project proposer was to discuss the overall project and to get a better understanding of what was required of the system. He first gave context to the system in terms how VMS operates and the type of work that they do. This gave the team a better understanding of the environment that the software would be used in and it would help in identifying some key non-functional requirements. The proposer continued by conducting a walkthrough of a much larger system (used for Emergency Centres around South Africa) as an example of what VMS would need. He explained how the system works from a high-level view. While going through the system, he highlighted the features that would be necessary for VMS. The meeting was recorded and enabled the team to analyse the system more closely. The team then collaborated to identify the functional and non-functional requirements that would define the VMS system.

The functional requirements included information about what the system should be able to do. In respect to the sedation clinic, it is essential to be able to manage patients through the different phases (which include *Waiting for Triage; Waiting for Theatre; In Theatre; Waiting for Disposition; and Waiting for Exit*). Hence being able to register the patient and then having the ability to move the patient through the different phases and capturing the respective information for those phases. Users should be able to add appointments for the scheduled sedation clinics for patients who are registered on the system. In addition to the sedation clinic, volunteering staff need to manage their patients coming to and from the sedation clinic. The proposer referred to this as the general practice. In this part of the system, it would be necessary for users to add appointments and fill in relevant information about the patient's visit. Furthermore, VMS requires the ability to make changes and have a high-level control of the system. This includes information that will be used in the system (beneficiaries, locations, hospitals etc.). The admin user should be able to manage the staff and patients, in terms of deleting, editing, registering and linking patients to respective doctors (for their practice and to access patient information).

The non-functional requirements included information about who the users are (referred to throughout the paper) – doctors, nurses, dentists, dental assistants, anaesthetists, secretaries, and other allied service professionals; the devices the software needed to be compatible with – which included all android and iOS devices; the type of environment where the software is required to function includes areas with internet access and that are fast-paced, hence the software must be user friendly and actions should be clear in the interface. In addition to these requirements, the security and performance of the application is of the utmost importance as the system deals with patients' sensitive information and the staff's time cannot be wasted on a slow ineffective system.

To ensure the best outcome of the project, the team needed to validate and verify the identified requirements. In order to do this, a detailed list of the software requirements was sent to both the supervisor and project proposer to analyse and ensure that we had captured the correct needs of the system. The requirements allowed for a detailed view of the system but ensuring changes could be made if necessary, for VMS.

A member of the team developed a prototype following an evolutionary methodology [10]. The software requirements were used to develop a prototype which was presented to the project proposer for feedback. Subsequent prototypes were developed to adapt to the feedback received from the proposer. These prototypes included additional features and software improvements. In addition to validating the software requirements, the feedback gathered at each iteration of the prototype further clarified that we were developing a system that fitted the requirements of VMS. [10]

To provide a clear overview of the functionality of the system a use case diagram was created (*refer to Appendix A*). The use case helped identify the requirements of the system and provide some clarity on the actions performed by users of the system. [11]

3.2 Overview of System Architecture Design

To effectively convey the importance of this paper, it is imperative to give a brief overview of the overall system design.

The system followed a layered architecture which was segmented into the presentation layer, business logic layer and the data layer [46]. The data layer contains the database which was designed using a relational model to ensure the complex relationships of the data are handled [12]. The business logic layer includes the Application Programming Interface (API) which was designed to follow a REST (REpresentational State Transfer) architecture. This ensured that the API followed a set of principles laid out in its architecture, enabling the creation of a uniform interface for the frontend to access. [13] The frontend can then make calls to the different endpoints defined in the API to retrieve, update, delete or create data in the database [14] (REST and CRUD have been explained in detail in a team member's paper). The presentation layer contains the frontend and is designed so the information that is retrieved from the database will be stored in a global storage of information. The storage is accessible from all parts of the frontend making it easy to retrieve, update, delete or create data in storage. The information from storage will then be displayed in a clean user-friendly interface. The UI (user interface) of the system will need to be separated into specific components to ensure reusable code and code maintainability. The system uses JSON format to standardise the communication between the different components, both on the frontend and the backend. [33]

3.3 Software Design

As discussed in section *3.1*, meetings were conducted with both the supervisor and the project proposer. The meetings with the supervisor occurred weekly, where progress was discussed, and the team had an opportunity to ask questions we had from the previous week. Due to the Covid-19 pandemic the project proposer was busy and hence we scheduled meetings whenever possible. The meetings were used to receive feedback on the system while we developed it as well as providing him with progress updates. Additionally, the meetings allowed us to ask the project proposer questions about feature implementation decisions.

Once the software requirements had been generated and validated, the design of the system could take place. As discussed in section *3.1*, the prototype was developed and checked by the project proposer. A team member developed the prototype, and another member designed the database and application server (*refer to Appendix C and D for an ERD of the database and a backend component tree, respectively*). The software design will be discussed in this paper.

3.3.1 Web Application Design. The design of the software is directly linked to the requirements gathered and analysed. Hence, this ensures the product matches the required behaviour for VMS. The discussion in section *3.1* about the functional requirements gives a clear outline of what was required for the system and the type of features that are essential for a successful final product.

3.3.2 Software Systems Design. Due to the complex nature of the web application, the group decided that based on the requirements, using a tech stack that integrated well together and that was consistent throughout, would be beneficial. Hence, due to the group's experience in JavaScript, a JavaScript stack was preferable. The frontend was built using React, and the backend was built using Node.js and Express. Due to the structured nature of the data that would be required in the system, MySQL was chosen as the database.

The software design of the frontend focused on dividing the system into its key components. React has a component-based architecture [15] and hence designing the frontend architecture using a component tree enables clear visualisation of the structure of the system. In React, it is essential to follow the principle of DRY (Don't Repeat Yourself) [15]. Following this principle ensures the creation of reusable components and hence reducing the size of the code base (ultimately improving performance due to loading reduced files). Additionally, creating reusable code results in a reduction in development time, and it makes code consistent throughout by enabling different parts of the system to use the same code. [34]

In Appendix B the component tree is displayed. The figure visualises the overall system architecture in regard to the structure of the different components used in the frontend. The design of the software was created to ensure that changes can be made at different stages of development as it is crucial to ensure that development was prepared for changes in requirements, features or implementation decisions (agile processes described in section *4.1 Approach*). The frontend components were broken up into Containers and Components. The Containers are used like a shell for the different Components. Although this is a single page application (SPA) [16], the Containers act like different pages of the application. The parent node of the component tree is the App component as it determines which containers are displayed. The Layout component is used to structure the Components and Containers onto the screen. The Layout component itself is a Container. The Main component takes in all the different Containers and displays them on the Layout. Therefore, the Main component changes according to the current Route (Routing explained in section *4.2.4 Routing*) of the application.

The state of the application is essential for storing information about the current activity of the user and information about components displayed on the screen. The design of the state of the application is crucial to ensure easy access to the state and ensuring that the DOM (Document Object Model) [45] does not render too often. In order to improve performance, controlling how the DOM updates is essential. The concept of state in React is tightly linked to the Virtual DOM. When state is changed in the application the render method is invoked and the virtual DOM is compared to the real DOM and React determines if the real DOM needs to be updated [15]. Reducing the number of re-renders of the DOM will improve performance by reducing the number of

calls made to the API resulting in a reduction in fetching time. Due to the complex nature of the data dealt with in the application, a single source of truth was desired to ensure that all main data of the system was in a central place [15].

4 SYSTEM DEVELOPMENT AND IMPLEMENTATION

4.1 Approach

4.1.1 Methodology. The group followed an agile methodology with a focus on scrum practices. The design of the system was broken into smaller parts which helped focus on tasks that needed to be completed. [17] With the use of Jira, the group was able to manage the different tasks effectively. The sprint board was separated into through columns, namely, *To Do*, *In Progress*, and *Done* [18]. All tasks start in the *To Do* column and would be moved into the *In Progress* column once a member had started working on that task. Once that task was completed the member would move the task into the *DONE* column. Each sprint was approximately two weeks and was focused toward completing significant features of the system. On the Sunday before the start of the sprint, the group would meet to map out all tasks to be completed in the upcoming sprint. This planning meeting aimed to ensure that all team members could work on their assigned segments of the system simultaneously, with minimal dependency on other members' sections. For example, one member would work on the appointments feature on the frontend whilst another member worked on the appointment routes on the backend. This task management structure enabled the entirety of the sprint backlog to be completed by the end of the sprint.

4.1.2 Chosen Software. In section *3.3.2*, the technology stack was discussed briefly. In more depth, the frontend of the application is implemented using a JavaScript library called React. In conjunction with React, Redux was used to help structure and centralise the state/data of the application. Redux is a third-party state management library that is separate to React. It is used primarily for its ability to create a global store of the state of an application whilst changes to the store update the DOM. [19]

On the backend of the application, Node.js and Express were used to develop the application server of the system. The Express framework was used to create the API which followed a REST architecture. The API was created as an interface between the frontend and the database. MySQL was used as the database, because the data required for the application dealt with complex relationships [12], hence it was essential to inherently incorporate this characteristic into the database which is achieved using a SQL/relational database.

4.1.3 Allocation of Features for Implementation. Developer A was in charge of the frontend software design, state management and security. Developer B was in charge of the database, the application server, and authentication and authorisation. Finally, developer C was in charge of the UI/UX, device compatibility, and information capture.

4.2 Features and Implementation

4.2.1 File Structure. The file structuring of React projects are unopinionated, meaning the developer decides how to structure the relevant JavaScript and CSS files [15]. In this project's case, separating of files into their respective roles in the code was essential to ensure separation of concerns [39]. The use of the component tree in Appendix B provides a clear understanding of the separation of different components. The system was broken up into Containers and Components. The Containers are used to house the different components. Hence, two folders were created with titles namely Components and Containers. All UI components reside in the Components folder and all the container components housing the UI components reside in the Containers folder. Structuring the files in this way allows for easier code reuse as the components can be used many times throughout the system. Additionally, it is important to reduce the amount of nesting, because as nesting increases in the file structure the more difficult is it to define the imports for files and to change those imports if files are moved or changed. Hence, the folders and files are at maximum four nests deep.

4.2.2 State Implementation.

4.2.2.1 Redux State Management. Redux was used to achieve a central store which enables React to update the DOM on changes of the central store (which it otherwise would not). To make changes to the central store, actions are dispatched from the different components. Once the actions are dispatched, they invoke a Reducer which updates the state based on the command of the action. When changes are made in the Reducers these updates are made to the central store which then triggers a subscription. The subscription passes the updated state as component properties (props) to the respective component. The components can then access the state via props. This process can be seen below in Figure 2 [20].

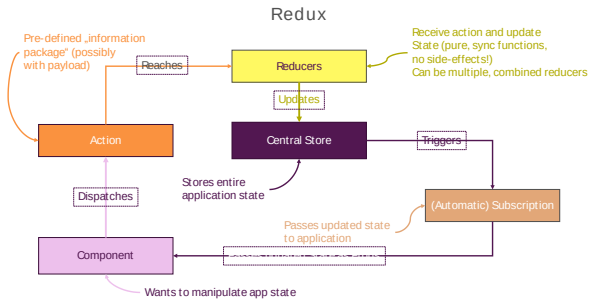


Figure 2: Process of changing central store from a component

A file with a list of action types is imported and used in the different action-creators. In these action-creators, different actions are created, and each action is referenced with an action type. These action types help determine what actions are changing the

central store. In the development browser a plugin called *Redux DevTools* was used to monitor the Redux actions and the resulting effects on the central store. The data that is relevant to the user such as *authentication* and their *user_id* as well as all main state of the application is managed in Redux.

State is temporary, and hence if the user reloads the page the state will return to its initial predefined state. To prevent this from occurring a npm library called Redux Persist is used to persist a stipulated whitelist of the reducers to local storage of the user's browser. The state gets rehydrated on reload/refresh. [36] Local storage is also used to keep track of information about the user such as *authentication* (e.g. token) and their *user_id.*

4.2.2.2 Component State Management. In the application a combination of component state and Redux was used. The state that primarily deals with the UI of the application will be managed by component-based state. Majority of the components are function components[1] and hence the useState hook was used to deal with state in these components [15].

4.2.3 Security Implementation. Due to the security concerns surrounding privacy of data, the security needed to be implemented effectively throughout. Between the frontend and the backend an HTTPS (Hypertext Transfer Protocol Secure) connection was implemented. HTTPS secures the channel of communication between the web application (frontend) and the server (backend), hence the transmission of sensitive patient/staff data between the two is encrypted. [35] HTTPS was implemented on the UCT web servers with an installed activated certificate.

To protect the data that is stored in local storage, including the persisted Redux state and other sensitive information about the user's session, two npm libraries were used. Secure-ls is an npm library that encrypts and compresses the data in local storage. [37] To ensure the best possible security of the data in local storage, AES[2] (Advanced Encryption Standard) with compression was used. The other npm library was redux-persist-transform-encrypt which was used to encrypt the persisted Redux in local storage. A secret key is given to the library to encrypt the data. [38]

To prevent users from highlighting confidential text and copying it, the css command *user-select: none; was used*. Furthermore, the application prevents autofill of login details on the sign in pages as the application might run on computers that are accessed by many users. The *autocomplete* attribute on the input elements were set to a random string which results in the prevention of autofill by the browser.

4.2.4 Routing. A package called *react-router-dom* was used to handle the routing of the application. With the use of this package, the user is able to navigate to different parts of the application while React only needs to determine which container to render [21]. The reason for this implementation is to build a Single Page Application (SPA). A SPA is a web application that has the aim to provide a user experience similar to a native application [16]. The transitions between screens are much faster and in the case of this application the side navigation remains on the screen while navigating through the application, hence removing the feeling of constant refreshing and loading.

In the application, routing is implemented by wrapping the entire application with a router known as *BrowserRouter*. It enables the application to load a component for each route based on the URL segments[3]. To explicitly define these routes, *Route* and *Switch* are used. The *Switch* component wraps all *Route* components. When *Switch* renders, it searches through the *Route* components and determines which path matches the current URL segment. Once the *Switch* finds the correct match, the respective *Route* is rendered, and all other *Route* components are ignored. The ordering of the *Routes* is important, as the first *Route* to match will be rendered. Therefore, the more specific *Route* components are defined first. [21]

To improve the performance of the application, React.lazy [15] was used to reduce the number of imported components. As a result, components that are not used as often, are only imported when they are called upon, hence reducing the number of initial imported components. React.lazy was used in the app component for the staff profile and the patient profile as these components will be used the least by users. When the user navigates to these screens an initial loading symbol will be displayed while the component is imported into application.

4.2.5 Main Actions. The main actions of the system include, creating, retrieving, updating and deleting information. As discussed in *4.2.2.1*, actions are called from components in order to make changes to the central store of the application. In most of the action-creators a request is sent to the server requesting that some action be performed with relevant data. *Axios* was used in order to make requests to the server [22]. Actions that make requests to the server are required to change the state of the application. Hence, why the *axios* calls are made in the Redux action-creators.

Axios is a "Promise based HTTP client" and hence it returns a promise to ensure that an action can be called on completion of the request [22]. The use of *then()* after the request helps identify the response after completion of the request. By using ES6 arrow functions [23], an anonymous function[4] is created to get the response sent from the server. To inform the user of requests in progress (loading), loading is set to true on dispatch of *action start* and loading is set to false on either *success* or *fail*. On *success* or

[1] Function components are defined using function notation, and do not extend React.Component. To manage state in these components, useState is required. [15]
[2] Advanced Encryption Standard (AES) is an algorithm designed to secure data. AES is a symmetric block cipher which is used to encrypt and decrypt data, from its original form to ciphertext and back. [40]
[3] URL segments are the added words at the end of the main URL e.g. https://vms/dashboard.
[4] Anonymous function is a function that has no name. [41]

fail of a request, a *success* or *fail* action is dispatched respectively, and a reducer modifies the state.
Specific information is required for different calls to the server, which may be, for example, the id of a patient or the data of a new appointment. Either the information is sent as parameters (e.g. id or limit) or is sent as data in the request (e.g. new appointment, new staff member, etc.).

4.2.6 Patient Table. The Patient Table is a component reused throughout the system. The component provides a view of the patients in the overall system as well as the current patients in the clinic and general practice. The table was developed to be reusable throughout the system. In order to achieve this, the table accepts props that are specific to the location in which the table is being used. By implementing the table in this way, the table is able to be used anywhere in the system and any data, headings or other specifics (such as buttons, actions and navigation) can be specified by sending the relevant information via props.

The *Patient Table* is a parent component that houses many *Patient Row* components. Each *Patient Row* component is a patient's data and hence the data array that is sent to the table via props is then mapped into each *Patient Row* component. With the use of JavaScript's *map* [24], each iteration of the data is passed off as props to a new *Patient Row* component.

In order to improve the user experience, pagination was used when fetching data. This incorporates sending a *limit* and *page number* to the API in order for the server to return a limited amount of data whilst returning data that is relevant to the page number sent. Pagination improves the performance of the application and reduces the time spent contacting the server. [43]

4.2.7 Dashboard. The Dashboard of the sedation clinic is the page (Container) that provides an overall state view of the application. The user is able to toggle between two views of the *Patient Pool* table, which are *All Patients* and *Current Visits.* This is implemented by dispatching an action to Redux to alter the state of view of the table. The value in state is then used to either fetch all patients in the database or to fetch all started visits in the current clinic.

The Dashboard in the general practice provides an overall view of the patients in the signed-in staff member's general practice. The user is able to toggle between two views of the *Patient Pool* table, which are *My Patients* and *Current Visits*. This is implemented in the same way as the sedation clinic dashboard, but different API endpoints are called. The *My Patients* view displays all patients that are linked to the signed-in staff member, while the *Current Visits* view displays all patients whose visits have started in the general practice.

4.2.8 Appointments.

4.2.8.1 Sedation Clinic Appointments. On the Appointments page in the sedation clinic a list of Sedation Clinics is displayed – this is achieved using the *map* function explained in *4.2.6* – where the data and actions for each clinic is mapped into a new *Sedation Clinic* component. On navigating to this page, an algorithm runs to determine which upcoming sedation clinic to select by default. On each re-render of the page an asynchronous request is made to the database to fetch the upcoming sedation clinics and a function is called on the completion of the fetch to find the closest date to the current date; once this date is obtained that date becomes the default selected date.

The user is able to book patient appointments for scheduled clinics. The available spaces in the clinic is checked in Redux to determine if the new appointment can be added to the selected clinic, if not the modal state is set to true and the user will be informed that they cannot add the patient to the clinic as it is full. When adding an appointment, the patient is searched for in the database to determine if they can be added to the clinic as they cannot be booked twice in the same clinic.

On selecting the radio button in the patient row, the patient id is stored in component state of the Sedation Clinic container which is passed as props to the Patient Table which determines if the actions should show. The actions include *edit, delete* and *check-in* (only if the clinic date is the same as the current date, the *check-in* action will be available). If the user selects *check-in*, an action is dispatched to Redux and a PUT request is made to the server to request that the relevant tables, for the start of a visit, are created. Once the action is complete the list of action buttons will disappear, and the component state is wiped. If the user selects the same patient, the *check-in* action will have changed to *start visit*, which when selected will start the visit of the patient by making a request to the API and resulting in the patient being moved into the first phase of the system (add to the *triage* Reducer*)*. The user will be able to view the patient in the *Current Visits* view on the Dashboard.

4.2.8.2 General Practice Appointments. The calendar used is a calendar API called FullCalendar [25]. The appointments are fetched on re-render of the page and the appointments are sent from Redux as props to the FullCalendar component. In order for the calendar to handle adding, removing, selecting, resizing and dropping, event functions were created and sent as props to the FullCalendar component. On selecting a day, the calendar view is altered to the day view. The user can then select a time which changes the component state to show the add appointment modal. The form in the modal captures the information and on save, dispatches an action which sends a request to the API to create an appointment. On selecting an appointment, the component state is changed to show the modal. The modal accepts the data from the selected appointment as props. The component state of the form determines if the modal is being used for editing or adding an appointment by checking whether props is undefined or not. The modal allows the user to edit, delete and start the visit. On selecting start visit, an action is dispatched, and a request is made to start the patient visit. On success the appointment colour is changed by adding a hex colour to the appointment object.

4.2.9 Phase Management. The system's main feature is to manage the patients through the five different phases of a visit which include *Waiting for Triage; Waiting for Theatre; In*

Theatre; Waiting for Disposition; and Waiting for Exit. Each one of these phases is a navigation item in the navigation bar. Each phase uses the Patient Table component to display the list of the patients in that phase. The user is able to click the three dots (options) which provides a dropdown list of actions. Two of the actions allow the user to transfer the patient through the phases, either backwards or forwards. Transferring the patient is implemented by dispatching an action in Redux which makes a request to the server to move that patient into the previous or next phase (depending on which action is selected in the dropdown). When the request is completed the "success" action is dispatched which calls the reducer and removes the patient out of the phase (deletes the patient from the array of patients in that reducer's state). *Refer to Appendix H for screenshots of the application.*

4.2.10 Capture Information. In the dropdown list of actions provided, the user can choose to Capture Info which will navigate them to the Edit screen where the different tabs represent the forms required for the visit. (A team member handled the capturing of information). When the user has clicked the Capture Info option the action to fetch the forms for the relevant Patient will be dispatched (using the patient id). The data is returned from the server in one JSON object. In the reducer the response is separated by its keys and the state is updated for each respective form state. The state of each form is sent to the respective form component in the Edit container. Local state is used in the forms as this data is temporary until the user has saved what they have changed. When the user saves, it dispatches an action to update the data in the database for that specific visit and patient. The update is also made in Redux on success of the request to the server.

In the procedure notes, the user is able to add ICD10 codes, here an API of ICD10 codes [26] was used to ensure that there was a correct mapping between the code and the description. It also increased the speed at which a user can add codes to a visit. An algorithm was created for adding and removing codes. The adding only adds the code to Redux if the code does not appear in the list already. The first two codes were given a type of *primary* and *secondary* respectively, and the rest are given a type of *additional*. Hence when adding, the type was determined based on the length of the list at the time of adding the next ICD10 code. In order to remove the code, the algorithm needs to check what the next code's type is and check what the to-be-deleted-code's type is and determine which codes need to change their type value.

4.2.11 Admin Panel. The admin panel provides a management application for all the data in the application. The admin is able to register staff members and on success of the server registering the staff member, an email with a magic link will be sent to the registered staff member's email address (discussed in detail in a team member's report). Once the user clicks on the magic link, they will be directed to a page to change their password. The link is valid for as long as the token is valid. The token placed in the URL will be used to authenticate the request to change the password of the staff member. On success of the request, the user will be navigated to the sign in page of the application.

In the admin panel, the admin is able to delete and edit all patients and staff members. They are also able to add, edit, end and delete sedation clinics. For the general practice, the staff members need to have access to prior information about the patients but to reduce exposing sensitive information to many different people, only the admin can grant this permission. Hence the selected patient id and staff id is sent to an API endpoint where the patient id is added to the selected staff member's patients. On success of the request, the patient is added to Redux. (*Refer to Appendix H for screenshots of the application and admin panel*).

5 TESTING METHODS

5.1 User Testing

In order to determine the success of the system in regard to its usability (UI/UX) and functionality, the team conducted usability tests. The tests were structured into two phases. Phase one was conducted after the completion of the foundation of the application – most were features completed. The tests were conducted with six users, each of these users had some background knowledge of computer science, UI/UX or they work in industry. By selecting these participants, it provided the team with a good technical analysis of the system and gave good insight into what did and did not work. Phase two was conducted after gaining feedback from the phase one usability tests and applying the changes to the system. Phase two tests were conducted with 3 users, each either had healthcare experience, have used a healthcare system or are involved in healthcare. Phase two provided information about what people in healthcare thought about the system and if it could be useful and effective in deployment. Finally, as part of phase two a demo to the project proposer was given to gain feedback on the completed system to ensure the requirements were met.

The usability tests all followed the same order which was first obtaining written consent from the participants prior to the test; on the day of the test, verbal consent was obtained before continuing with recording of the video chat; an explanation was given to ensure the user completely understood what their role was in the study and what they would be doing in the test; and then the test would commence with the tester asking the user to complete a set of tasks on the application. (*Refer to Appendix E for set of tasks*)

5.2 Software Testing

Software testing involved testing the inner workings of the software and ensuring the software performs how it was intended to. In order to conduct testing on the software a JavaScript Testing Framework known as Jest [27] was used. The reasons for using Jest was that it is easy setup and use; it is a favoured choice for testing React applications; it is compatible with Node.js [42]; and the team has some experience using Jest. Tests were conducted

separately for the frontend and the backend of the application. On the frontend the main features were tested; on the backend the most significant routes of the API were tested to ensure the correct implementation of the CRUD operations.

6 RESULTS AND DISCUSSION

6.1 Phase One

Task: *Link a patient to a staff member.* The user was unable to link a patient to a staff member because by default no staff member was selected in the list of staff members. Additionally, the *+Patient Link* button was confusing as it was displayed even though a staff member was not selected. The user then clicked the button with no selected staff member to a link a patient resulting in an error. Otherwise, users were able to understand the linking of patients to staff members and found it easy to complete the task.

Task: *Register a staff member.* A user entered a name without providing a first and last name. Hence, this means that there is no validation on the full name field. Additionally, when they clicked the register button an error with the feedback modal occurred resulting in a crash of the application. In addition to this, users did find that the staff register button was clear and easy to find, and that filling in the information was quick and simple.

Task: *Add a sedation appointment.* A user was unaware that they had to select a clinic before adding an appointment. As a result, when the modal showed it caused an error as the system tried to select the selected clinic's date (which was undefined) in the dropdown of clinic dates. Most users were confused as to what the difference between scheduled and unscheduled was. Otherwise, users found that adding an appointment was easy to understand and that the modal was clearly laid out and organised.

Task: *Check-in a patient and start their visit.* Users struggled to identify how to check-in the patient. It took time for them to realise they needed to select the checkbox in the patient row in order for a list of actions to show. Users were unsure about how to start a visit once they checked-in the patient due to the actions disappearing after check-in of the patient. Additionally, the checkbox made the users think they were able to select many patients when they should only be allowed to select one.

Task: *Fill in the appropriate information for the current phase.* Users were unsure of how to capture the information for the current phase. They liked that if they clicked the three dots a dropdown would show but the list item, *edit*, was unclear. Users thought that the edit was to edit the patient details. One user clicked on the edit button which resulted in an error as the application tried to navigate to a patient with id of undefined. In the sedation clinic forms all the sliders had a max value of 100 which made it impossible to fill in details where the value could be over 100. In the procedure notes the user was not sure after clicking the microphone button if the voice-to-text had started. Additionally, the user was confused about how to stop the voice-to-text once it had started as the icon did not change. Furthermore, when adding ICD10 codes the user was not sure if clicking the + ("plus") next to the code was adding the code to the notes as the modal did not close nor did it give feedback as to which ones had been added. Otherwise, users enjoyed the fact that it was so easy to navigate through the different forms in both the sedation clinic and general practice. Users found the forms easy to use, understandable, organised, and easy to access and edit.

Task: *Sign into the application.* The user entered incorrect details for their account, but they were unsure what happened because the system did not inform the user of the incorrect action. Otherwise, all users enjoyed the simplicity of the sign in page and found the interface clear and user friendly.

General issues included modals not closing on the success of an action (e.g. linking a patient to a staff member); no feedback on the completion of an action (e.g. deleting an appointment); errors with data not loading correctly into the correct components. General positives were that users felt the system was built with the user in mind; the actions were simple enough to understand; really enjoyed the voice-to-text feature; they liked the colour scheme; and finally users found it easy to complete almost all tasks except for a few issues mentioned above.

6.2 Phase Two

After implementing the fixes and changes discussed in *6.1*, the team also ensured that all gaps of the system had been completed. Once the team was happy with the system, phase two of usability testing commenced.

All users were able to complete all the tasks they were asked to complete. They commented that it was easy for them to manage the patients of the system and that the flow of the system was intuitive and easy to grasp. The most important feedback was about the capturing of information as all three participants stated that the forms easily organised the information into a clear interface. The most liked feature was the addition of the voice-to-text as it provided the users with a quick and easy way to take down notes. No problems arouse and users really enjoyed the experience of using the system and got the hang of it quickly. They also found the system to be responsive and efficient.

A demo of the system was conducted with the project proposer after the above testing was completed. In this demo, a walkthrough of the application was shown, making sure to explore every feature and action the user could accomplish. The demo was successful as the project proposer was impressed by the application the team had built and he stated that the application met all the requirements.

6.3 Software Testing

6.3.1 Frontend testing. Jest enabled the testing of the reducers in Redux. The tests conducted ensured that the functional

requirements of the software were correct. The focus of the testing was on the reducers because it allowed a reflection of the action-creators' results on the different instances of the central store (Reducer). Hence, the unit tests tested whether given some data that the action-creator had correctly changed/updated the Reducer.

All tests that were conducted were successful, meaning all functional requirements were successfully implemented using Redux. *(Refer to Appendix F for screenshots of the successful tests run on the frontend.)*

6.3.2 Backend testing. The backend of the application was tested using the Jest framework. The tests on the backend were conducted to ensure the CRUD operations of the system were implemented correctly. The testing ensured that the API was successfully changing/updating the database and thus reassuring the success of the functional requirements.

All tests that were conducted were successful meaning the CRUD operations implemented using Node.js was successful in completing the different CRUD operations. *(Refer to Appendix G for screenshots of the successful tests run on the backend.)*

7 REFLECTION AND RECOMMENDATIONS

A full acceptance test could not be conducted as the clinics were postponed during the Covid-19 lockdown in South Africa. Hence, the system could not be implemented in the area in which it would be used in deployment which prevented the team from analysing the improvements and advantages that the system would provide to VMS. Furthermore, the restrictions implemented reduced the ability to ascertain the achievement of the aims set out for the application developed. The success of the application was thus determined by the usability tests and the software tests conducted. Additionally, due to Covid-19 the proposer (who is a doctor) was very busy during the development of the system. Hence, the progress and feedback meetings with him were irregular.

Issues arouse from the separation of responsibilities of the project earlier on in the development of the application. We solved the issue by separating the project into its three main parts (Backend, Frontend and UI/UX) and hence we were able to allocate the sections effectively.

The implementation of the Scrum methodology worked very well for this project. The team found that the morning stand-ups provided a great way to align with one another.

If the project was to be replicated, the use of a simpler task management tool such as Trello, would be advised. Following an agile methodology would be beneficial for developing such a complex project and to ensure that changes to the system could be managed. Using a VCS (Version Control System) such as Git is essential for the success and managing of the development of the system. The team found that the use of Git made it easy to manage our different features and handle the combining of the different versions of the system.

8 CONCLUSIONS

This paper aimed to describe the development of a web application which would be used as a software solution to improve the operations of Vision Medical Suite and how it can effectively manage their clinics, stakeholders and the relevant information pertaining to operational requirements.

Due to the completion of the development of the application and the satisfaction of the team, the choice of software for development was effective in handling the different tasks and features of the system. Additionally, the software is fast, secure (based on best practices) and maintainable (because of the user manual, clear code, comments and documentation). Although the team was unable to determine the success of the system in the actual clinics, we could still derive success from the tests conducted.

The results of the usability tests have provided evidence that the software is easy to use. Users were able to complete all the tasks they were asked to do thus we can decisively say that in terms of the usability of the system it achieves its goal of being a simple, clean, and easy to use interface that manages to help its users easily achieve the tasks they set out to do. Additionally, the project proposer was satisfied with the application and that it had met the system requirements.

In regard to the software tests run on the system, the unit tests were implemented on the main features of the system and highlighted the core functionality of the application. The success of these tests shows that the inner workings of the software, both on the frontend and backend, work correctly and that the developed system has met the stipulated software requirements.

9 FUTURE WORK

If the system is to be maintained and development continued, some additional features to add would include integrating a messaging system into the application so that staff can talk to one another directly through the application; in addition to the calendar on the application, connecting to the Google calendar API to mirror the appointments and schedules onto staff members' Google calendar would help improve managing their patients' appointments; and adding an overview page to display the average waiting times in each phase of the clinic and having a list who is next to be checked-into the clinic.

ACKNOWLEDGEMENTS

I would like to thank my team, Justin Dorman and Chad Piha, for coming together to produce an impressive application. I would also like to thank Aslam Safla, Melissa Densmore and Dr. Moosa for their invaluable input.

REFERENCES

[1] South African Government, 2013. Protection of Personal Information Act.

[2] Facebook, 2018. GitHub: facebook/react. Retrieved September 8, 2020 from https://github.com/facebook/react/blob/master/LICENSE

[3] Node, 2020. GitHub: nodejs/node. Retrieved September 8, 2020 from https://github.com/nodejs/node/blob/master/LICENSE

[4] Express. 2020. GitHub: expressjs/express. Retrieved September 8, 2020 from https://github.com/expressjs/express/blob/master/LICENSE

[5] MySQL. 2020. MySQL: Products – Community Edition. Retrieved September 8, 2020 from https://www.mysql.com/products/community/

[6] Adobe. 2020. Adobe: Adobe XD. Retrieved September 8, 2020 from https://www.adobe.com/africa/products/xd.html

[7] OpenMRS. 2020. OpenMRS. Retrieved September 13, 2020 from https://openmrs.org/

[8] HospitalRun. 2020. HospitalRun. Retrieved September 13, 2020 from https://hospitalrun.io/

[9] TeamDesk. 2020. Medical Practice Manager database. Retrieved September 13, 2020 from https://www.teamdesk.net/

[10] Floyd, C. 1984. A Systematic Look at Prototyping, in: Budde, R., Kuhlenkamp, K., Mathiassen, L. and Zullighoven, H. (Eds.) Approaches to Prototyping, Springer-Verlag: Heidelberg, 1-17. DOI: https://doi.org/10.1007/978-3-642-69796-8_1

[11] Siau, K., & Lee, L. 2001. Role of Use Case Diagram in Requirement Analysis. AMCIS 2001 Proceedings, 251.

[12] Codd, E. F. 1990. The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc. DOI: https://dl.acm.org/doi/pdf/10.5555/77708

[13] Fielding, Roy Thomas. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD. Dissertation. University of California, Irvine, 2000.

[14] M Fikri Setiadi. 2018. 7 Steps to Create Simple CRUD application using Node.js, Express and MySQL (November 2018). Retrieved September 15, 2020 from http://mfikri.com/en/blog/nodejs-mysql-crud

[15] React. 2020. A Javascript library for building user interfaces. Retrieved September 15, 2020 from https://reactjs.org/

[16] Michael Mikowski and Josh Powell. 2013. Single Page Web Applications: JavaScript end-to-end (1st. ed.). Manning Publications Co., USA.

[17] Schwaber, K., & Beedle, M. 2002. Agile software development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

[18] Artlassian. 2020. Jira Software. Retrieved September 16, 2020 from https://www.atlassian.com/software/jira

[19] Redux. 2020. Redux: A Predictable State Container for JS Apps. Retrieved September 17, 2020 from https://redux.js.org/

[20] Maximilian Schwarzmüller. 2020. React - The Complete Guide (incl Hooks, React Router, Redux). Retrieved September 17, 2020 from https://www.udemy.com/course/react-the-complete-guide-incl-redux

[21] React Router. 2020. React Training/ React web guides. Retrieved September 20, 2020 from https://reactrouter.com/web/guides/primary-components

[22] axios. 2020. npmjs.com: axios. Retrieved September 17, 2020 from https://www.npmjs.com/package/axios

[23] w3schools. 2020. ECMAScript 6 – ECMAScript 2015. Retrieved September 17, 2020 from https://www.w3schools.com/js/js_es6.asp

[24] w3schools. 2020. JavaScript Array map() Method. Retrieved September 17, 2020 from https://www.w3schools.com/jsref/jsref_map.asp

[25] FullCalendar. 2020. FulCalendar Docs. Retrieved September 18, 2020 from https://fullcalendar.io/docs

[26] Brian Fritz. 2019. ICD10 code API. Retrieved September 18, 2020 from icdhttp://icd10api.com/

[27] Jest. 2020. Retrieved September 22, 2020 from https://jestjs.io/

[28] Anurag Kumar, and Ravi Kumar Singh. 2016. Comparative analysis of angularjs and reactjs. International Journal of Latest Trends in Engineering and Technology. 4, vol 7. 225-227. DOI: http://dx.doi.org/10.21172/1.74.030

[29] Eric Wohlgethan. 2018 Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js. Ph.D. Dissertation. Hochschule für Angewandte Wissenschaften Hamburg

[30] TechMagic. 2018. React vs Angular vs Vue.js – What to choose in 2020. Retrieved April 30, 2020 from https://medium.com/@TechMagic/reactjs-vsangular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

[31] Jens Neuhaus. 2017. Angular vs. React vs. Vue: A 2017 comparison. Retrieved April 30, 2020 from https://medium.com/unicorn-supplies/angularvs-react-vs-vue-a-2017-comparison-c5c52d620176

[32] Vue.js. 2020. Comparison with Other Frameworks. Retrieved May 6, 2020 from https://vuejs.org/v2/guide/comparison.html

[33] B. Lin, Y. Chen, X. Chen and Y. Yu. 2012. *Comparison between JSON and XML in Applications Based on AJAX*. International Conference on Computer Science and Service System, Nanjing, 2012, pp. 1174-1177, DOI: 10.1109/CSSS.2012.297.

[34] R. G. Lanergan and C. A. Grasso. 1984. *Software Engineering with Reusable Designs and Code*. IEEE Transactions on Software Engineering, vol. SE-10, no. 5, pp. 498-501, Sept. 1984, DOI: 10.1109/TSE.1984.5010273.

[35] CloudFlare. 2020. What Is HTTPS? Retrieved September 29, 2020 from https://www.cloudflare.com/learning/ssl/what-is-https/

[36] Zack Story. 2019. Npm redux-persist. Retrieved September 26, 2020 from https://www.npmjs.com/package/redux-persist

[37] Varun Malhotra. 2020. Npm secure-ls. Retrieved September 26, 2020 from https://www.npmjs.com/package/secure-ls

[38] Marshal Bowers. 2018. Npm redux-persist-transform-encrypt. Retrieved September 26, 2020 from https://www.npmjs.com/package/redux-persist-transform-encrypt

[39] Hürsch, Walter L., & Lopes, Christina Videira. 1995. *Separation of concerns*. DOI: 10.1.1.29.5223

[40] Pub, NIST FIPS 2001. 197: Advanced encryption standard (AES). *Federal information processing standards publication*, 197(441), 0311.

[41] JavaScript Anonymous Functions. 2020. Introduction to JavaScript anonymous functions. Retrieved September 29, 2020 from https://www.javascripttutorial.net/javascript-anonymous-functions/

[42] Jash Unadkat. 2019. Top 5 Javascript Testing Frameworks. Retrieved September 24, 2020 from https://www.browserstack.com/guide/top-javascript-testing-frameworks.

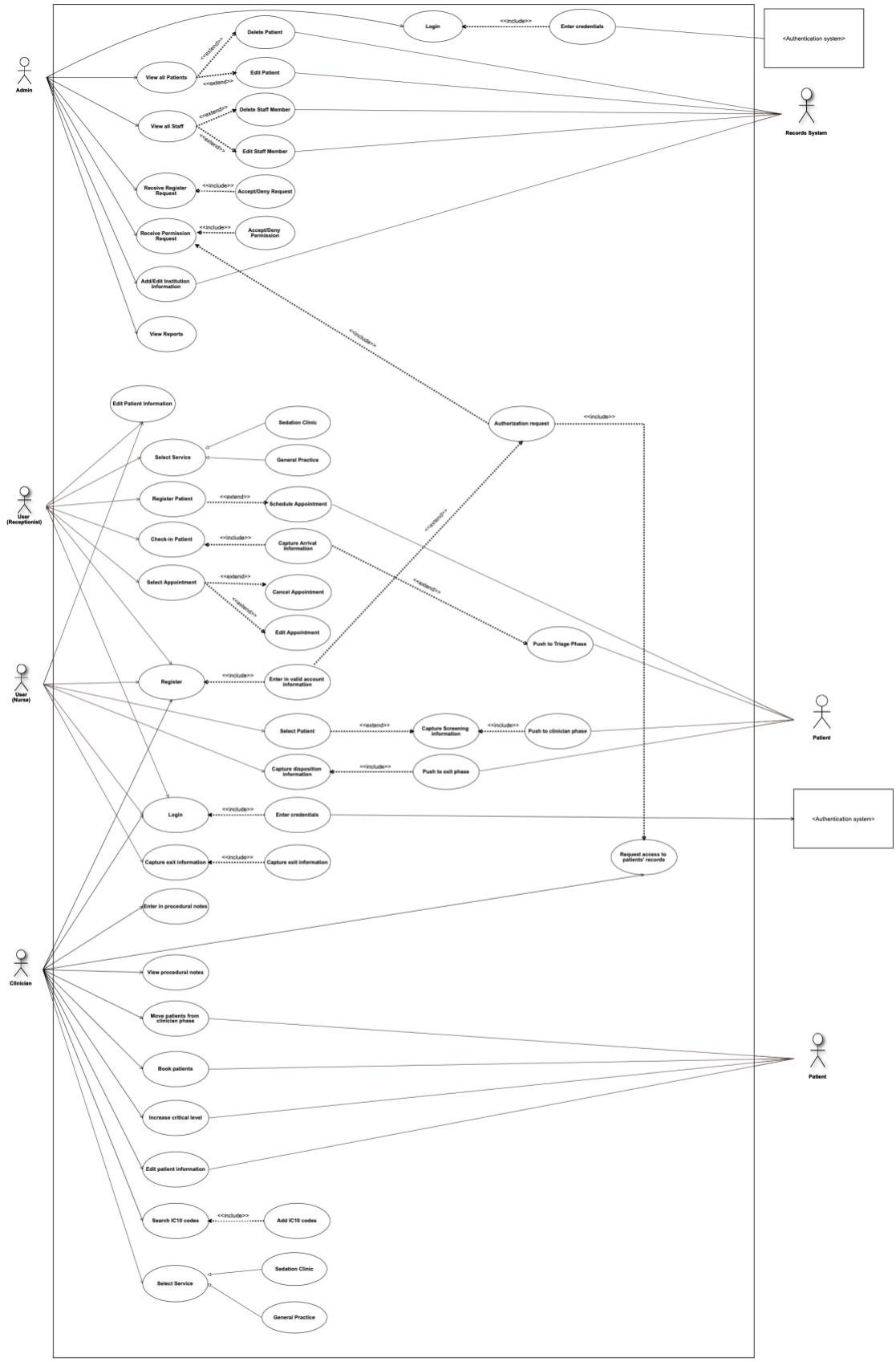
[43] Slack engineering. 2020. Evolving API Pagination at Slack. Retrieved September 30, 2020 from https://slack.engineering/evolving-api-pagination-at-slack/

[44] Vision Medical Suite. 2020. Vision Medical Suite: about us. Retrieved September 2, 2020 from http://www.visionmedicalsuite.co.za/about-us/.

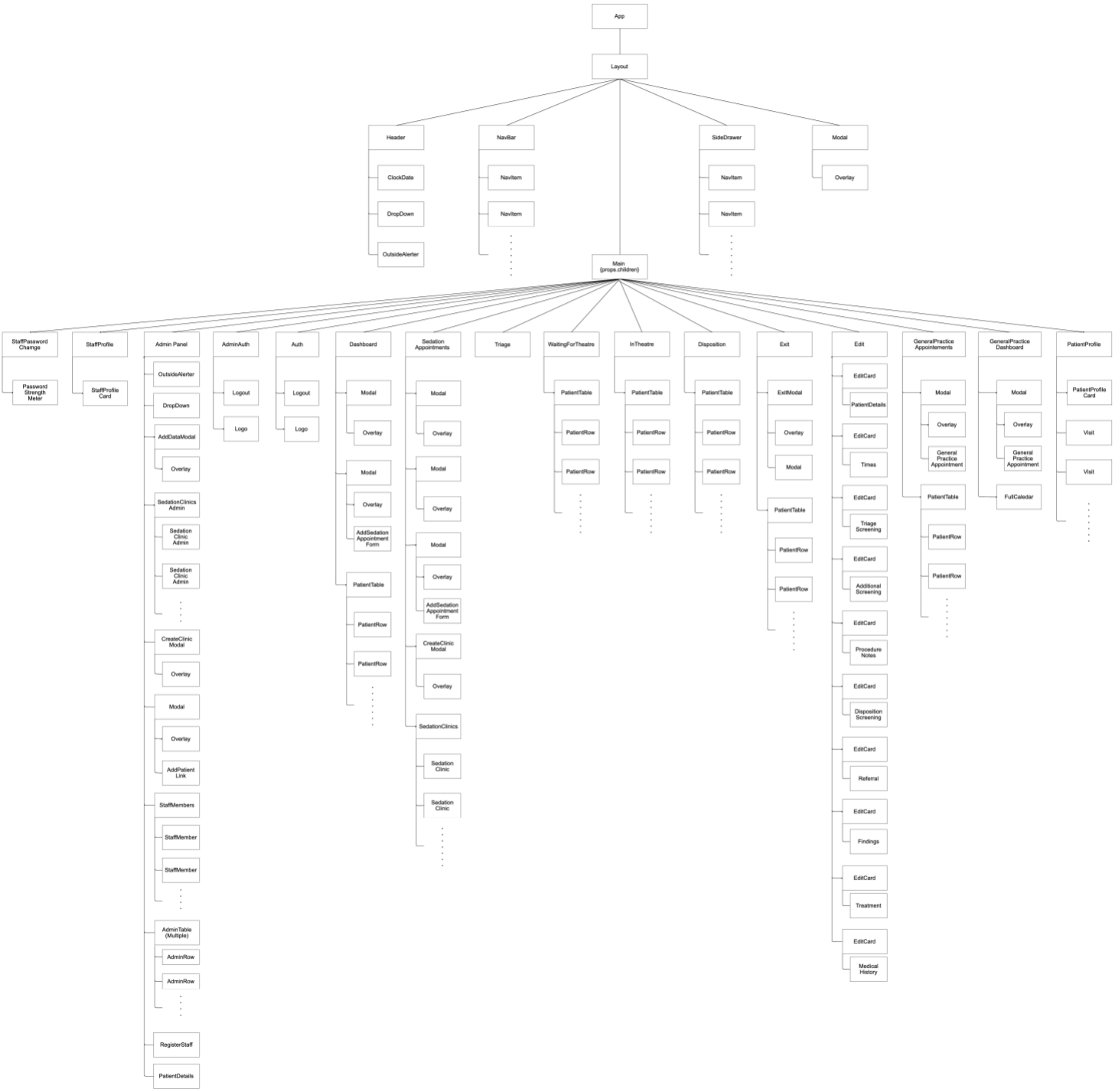
[45] w3schools. 2020. JavaScript HTML DOM. Retrieved September 17, 2020 from https://www.w3schools.com/js/js_htmldom.asp

[46] Richards, Mark. 2015. *Software architecture patterns (Vol. 4).* 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated.

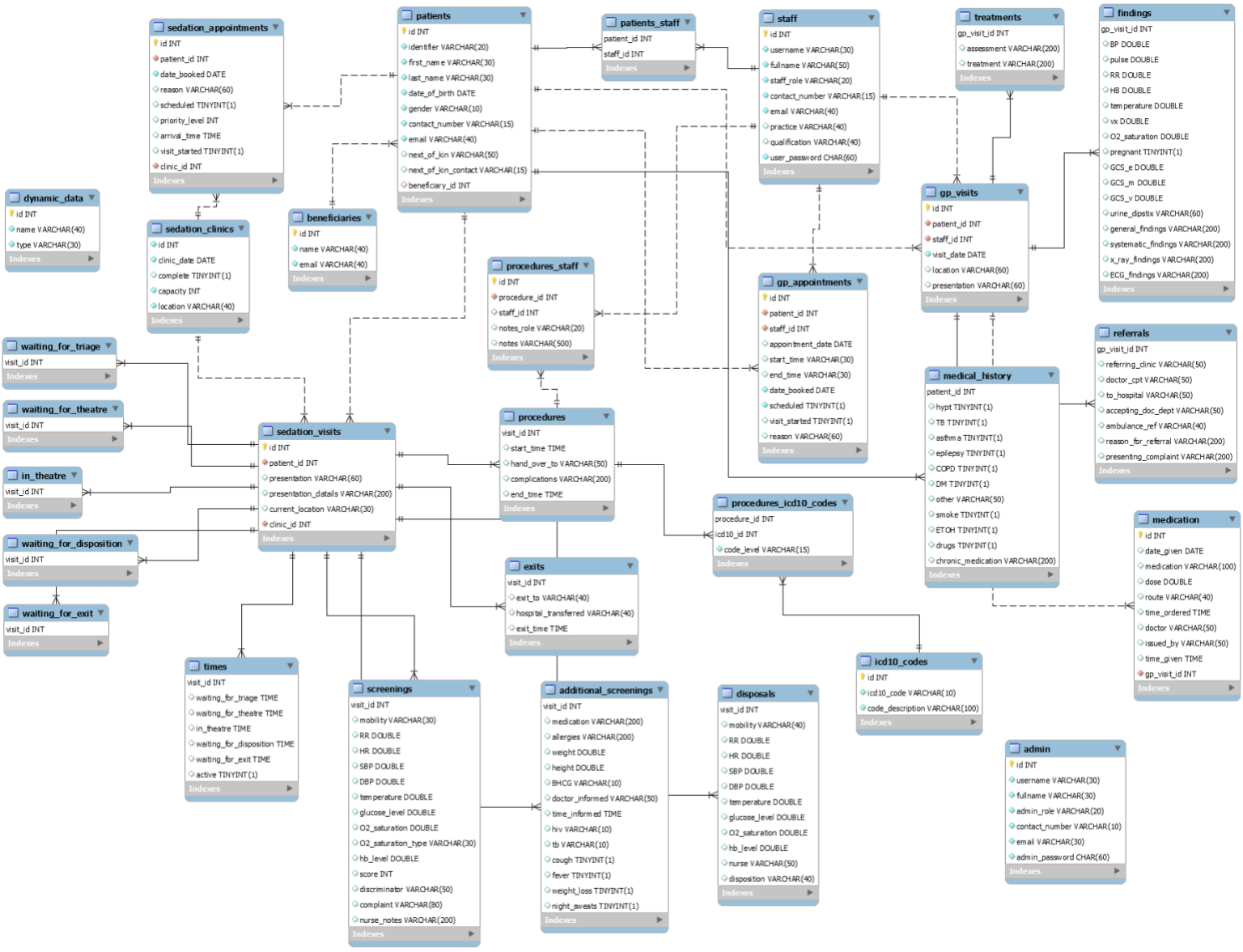
SUPPLEMENTARY INFORMATION
APPENDIX A: USE CASE DIAGRAM



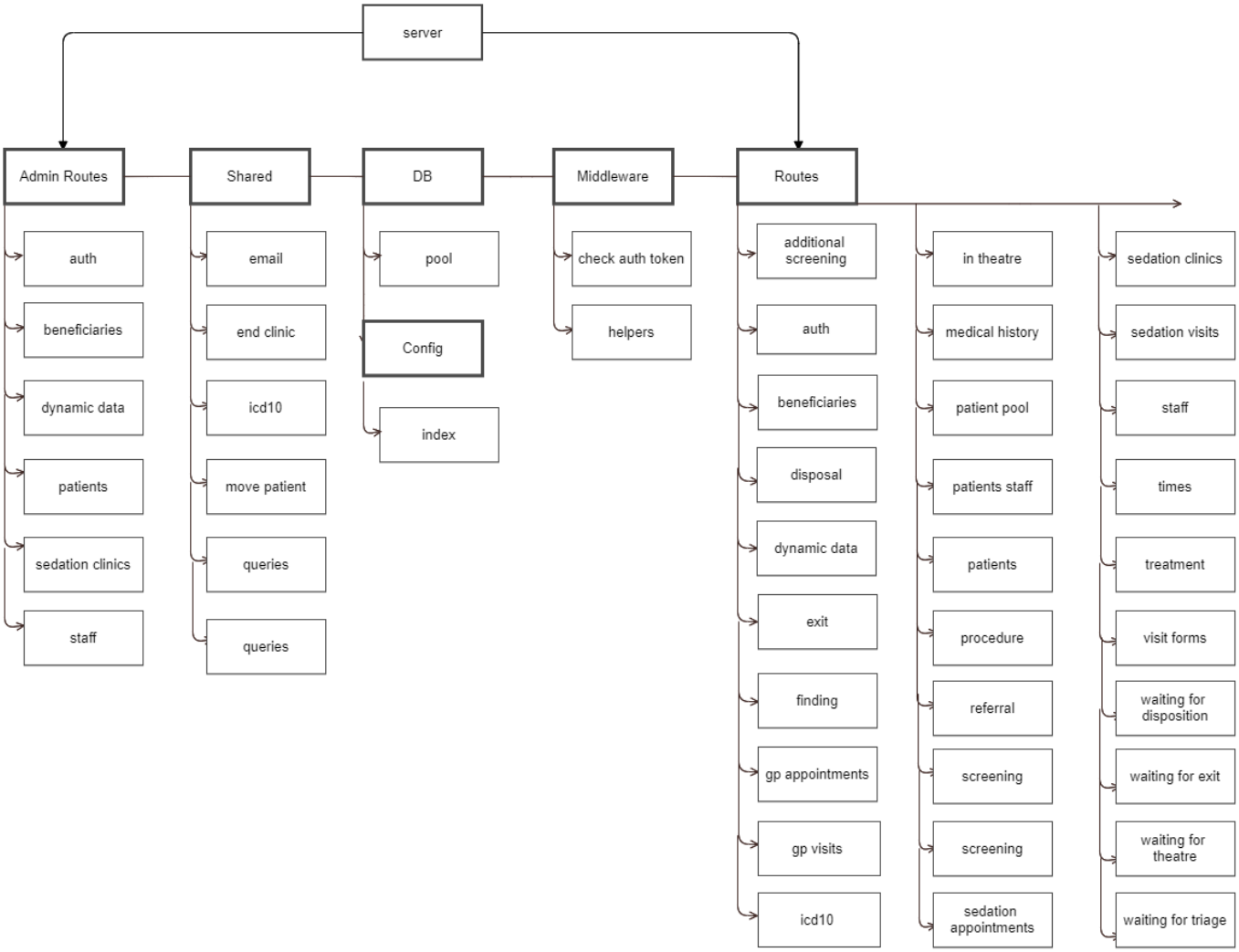
APPENDIX B: FRONTEND COMPONENT TREE



APPENDIX C: ERD OF DATABASE



APPENDIX D: BACKEND COMPONENT TREE



APPENDIX E: USER TESTS - HIGH LEVEL 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 them 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: SUCCESSFUL FRONTEND UNIT TESTS

```
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/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/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/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.
```

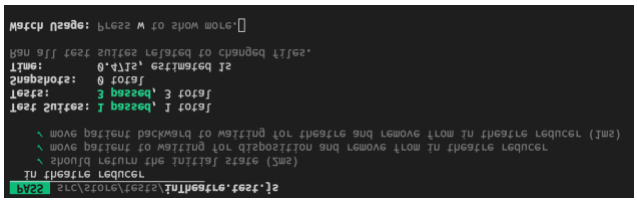
```
PASS src/store/tests/beneficiaries.test.js
  beneficiary reducer
    ✓ should return the initial state (4ms)
    ✓ fetch beneficiaries
    ✓ add beneficiary (1ms)
    ✓ edit beneficiary (1ms)
    ✓ delete beneficiary

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/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.
```

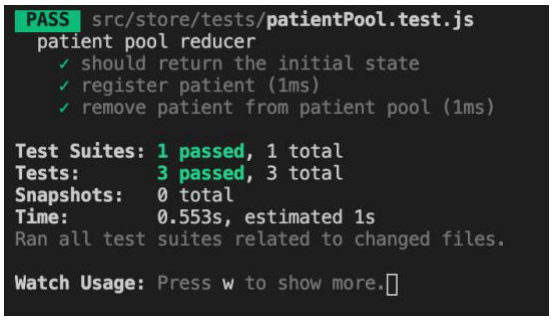
```
PASS src/store/tests/exit.test.js
  waiting for exit reducer
    ✓ should return the initial state (2ms)
    ✓ exit patient from sedation clinic and remove from waiting from exit reducer
    ✓ move patient backward to waiting for disposition and remove from waiting for exit reducer (1ms)

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.
```

```
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/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.
```

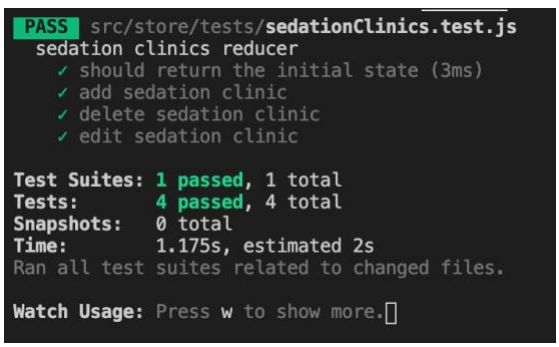


```
PASS src/store/tests/gpVisitForms.test.js
  gp visit forms reducer
    ✓ should return the initial state (3ms)
    ✓ fetching visit forms (1ms)
    ✓ edit referral information
    ✓ edit finding information
    ✓ edit medical history information (1ms)
    ✓ edit treatment information

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.
```

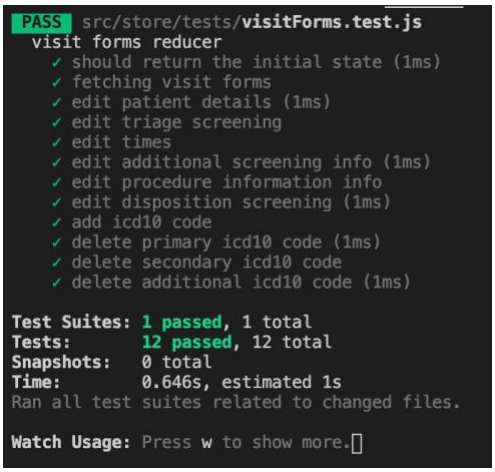


```
PASS src/store/tests/patientPool.test.js
  patient pool reducer
    ✓ should return the initial state
    ✓ register patient (1ms)
    ✓ remove patient from patient pool (1ms)

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.
```

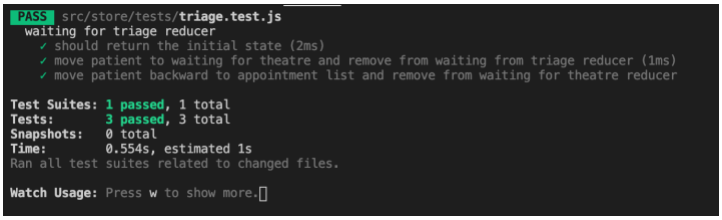
```
PASS src/store/tests/sedationAppointments.test.js
  sedation appointments reducer
    ✓ should return the initial state (1ms)
    ✓ add sedation appointment (1ms)
    ✓ delete sedation appointment (1ms)
    ✓ edit sedation appointment

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.
```



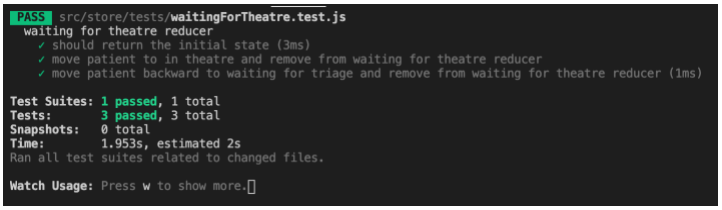
```
PASS src/store/tests/sedationClinics.test.js
  sedation clinics reducer
    ✓ should return the initial state (3ms)
    ✓ add sedation clinic
    ✓ delete sedation clinic
    ✓ edit sedation clinic

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/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/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.
```

APPENDIX G: SUCCESSFUL BACKEND UNIT TESTS

```
PASS testing/patients.test.js
PASS testing/visits.test.js
PASS testing/info_capture.test.js
PASS testing/visit_forms.test.js
PASS testing/gp_appointments.test.js
PASS testing/sedation_appointments.test.js
PASS testing/auth.test.js
A worker process has failed to exit gracefully and has been force exited. This is li

Test Suites: 7 passed, 7 total
Tests:       24 passed, 24 total
Snapshots:   0 total
Time:        7.012 s
Ran all test suites.
PS C:\Users\Justin Dorman\Desktop\vision-server>
```

```
PASS testing/visits.test.js
  Start Sedation Visit
    √ start a new sedation visit (84 ms)
  Start GP Visit
    √ start a new gp visit (22 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.196 s, estimated 3 s
Ran all test suites matching /visits/i.
```

```
PASS testing/patients.test.js
  Create beneficiary
    √ create a new beneficiary (50 ms)
  Register patient
    √ create a new patient (24 ms)
  Fetch patients
    √ fetch all patients (6 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.108 s, estimated 3 s
Ran all test suites matching /patients/i.
```

```
PASS testing/auth.test.js
  Register staff user
    √ create a new staff member (179 ms)
  Change password
    √ change the password of the new staff member (97 ms)
  Sign In
    √ sign the new user in (108 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.658 s, estimated 4 s
Ran all test suites matching /auth/i.
```

```
PASS testing/sedation_appointments.test.js
  Add new clinic
    √ create a new clinic (57 ms)
  Add new appointment
    √ create a new appointment (9 ms)
  Edit appointment
    √ edit an appointment (10 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.405 s, estimated 4 s
```

```
PASS testing/gp_appointments.test.js
  Add new appointment
    √ create a new appointment (92 ms)
  Edit appointment
    √ edit an appointment (9 ms)
  Fetch appointments
    √ fetch all appointments (7 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.652 s, estimated 4 s
Ran all test suites matching /gp_appointments/i.
```

```
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
Ran all test suites matching /info_capture/i.
```

```
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
Ran all test suites matching /visit_forms/i.
```

APPENDIX H: WEB APPLICATION SCREENSHOTS

Sedation Clinic:

Image 1: Application login screen

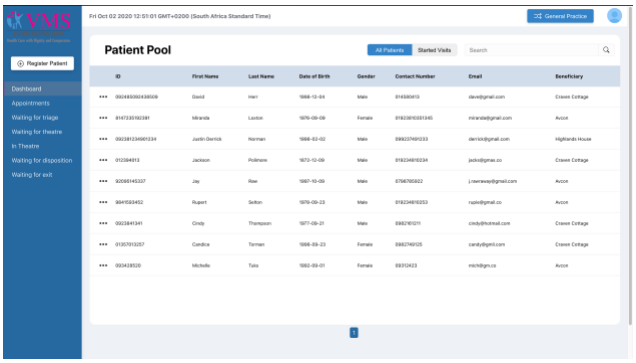


Image 2: Dashboard

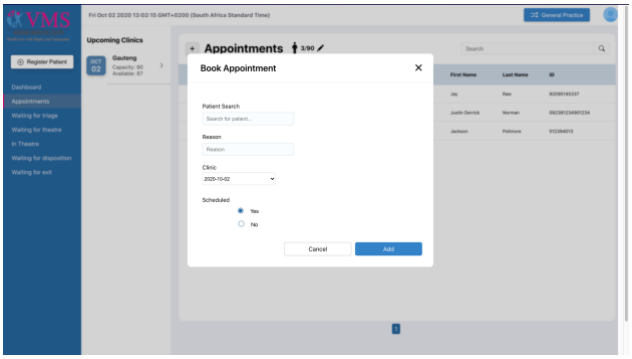


Image 3: Creating a sedation clinic appointment

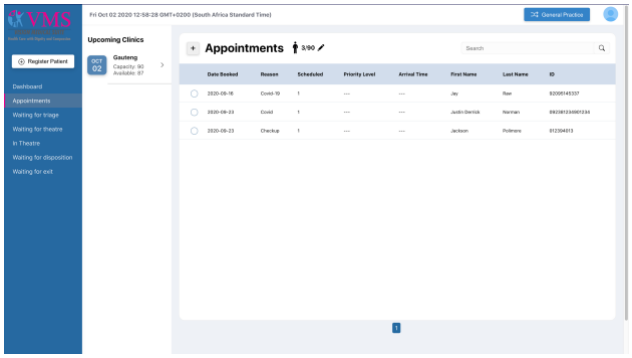


Image 4: Sedation clinic appointments

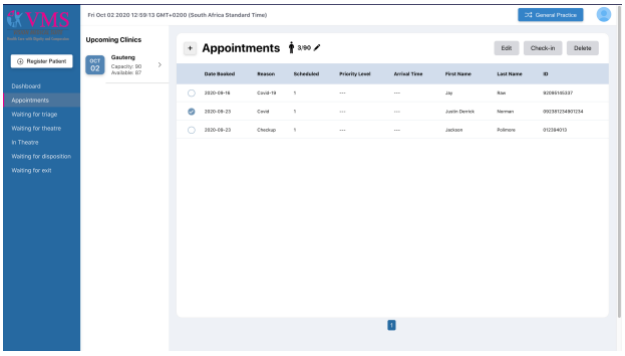


Image 5: Edit, Check-in and delete appointment function

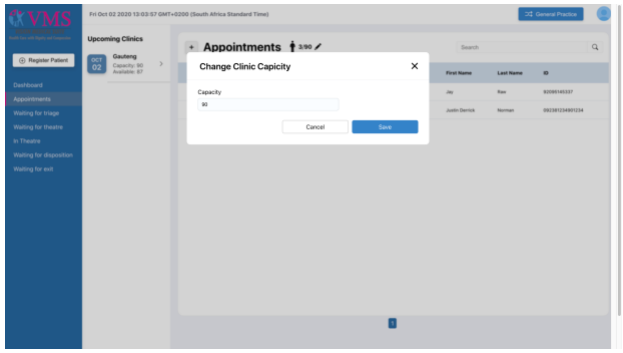


Image 6: Edit sedation clinic capacity

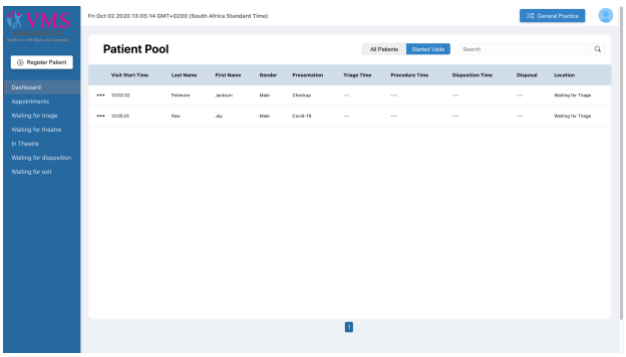


Image 7: Sedation clinic ongoing visits

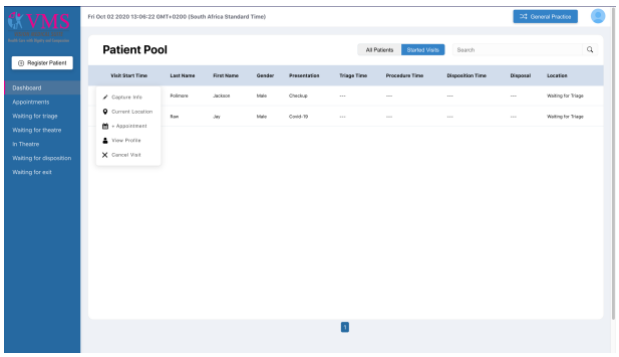


Image 8: Patient actions

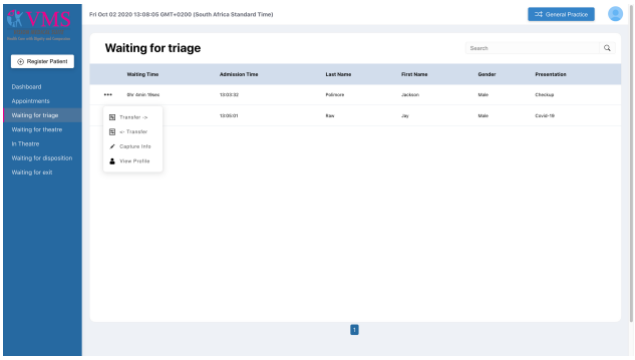


Image 9: Waiting for triage table with patient actions

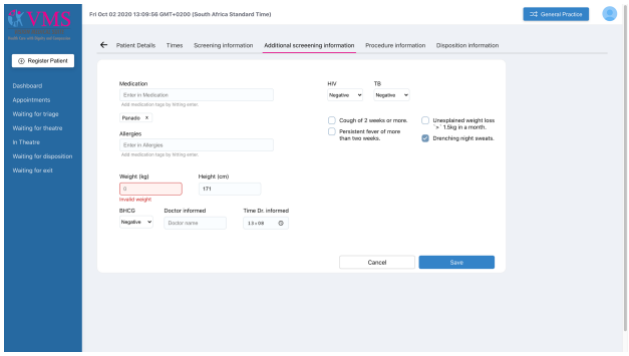


Image 10: Additional screening form (with validation)

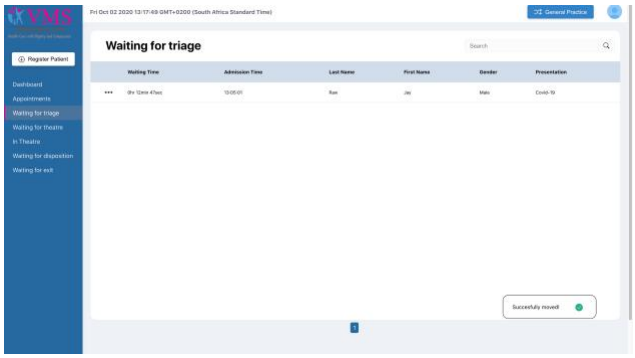


Image 11: Moving patient between phases feedback modal

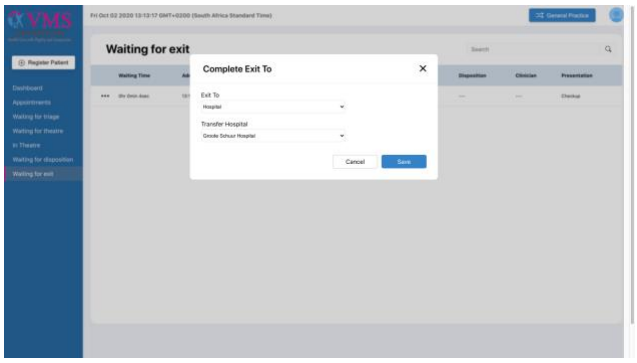


Image 12: Completing sedation clinic visit

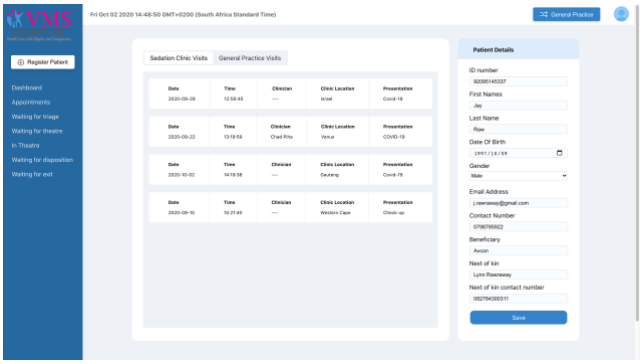


Image 13: Patient Profile

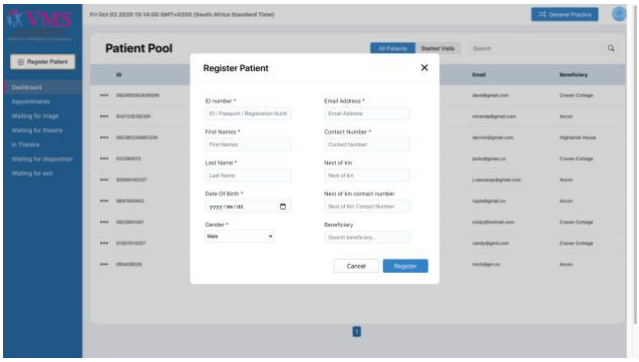


Image 14: Register Patient

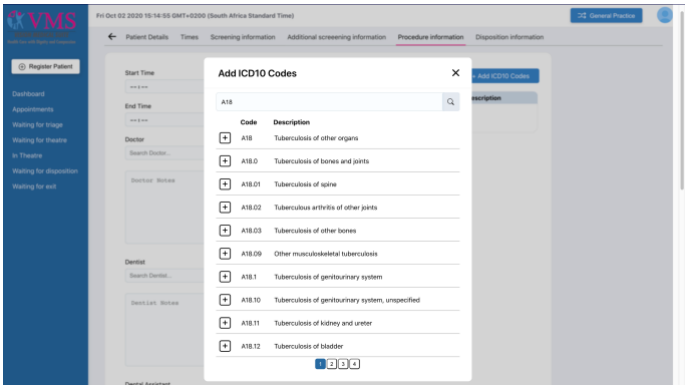


Image 15: Add ICD10 code modal

General Practice (GP):

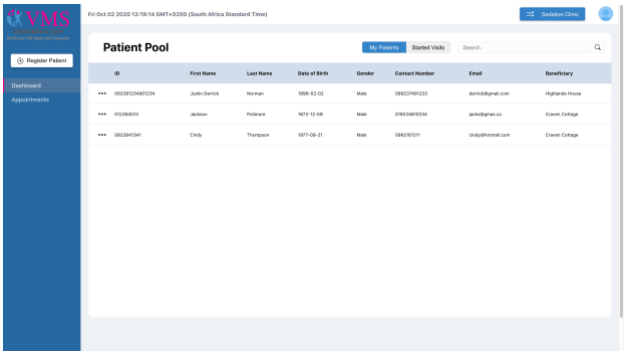


Image 16: GP Dashboard

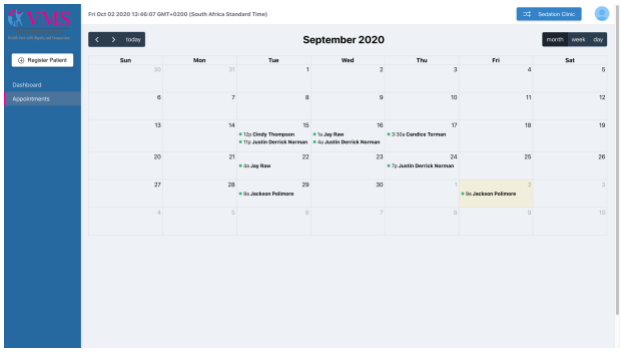


Image 17: GP appointments

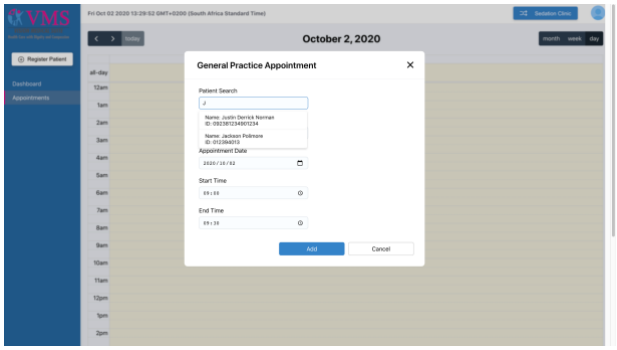


Image 18: Create GP appointment (with patient list dropdown)

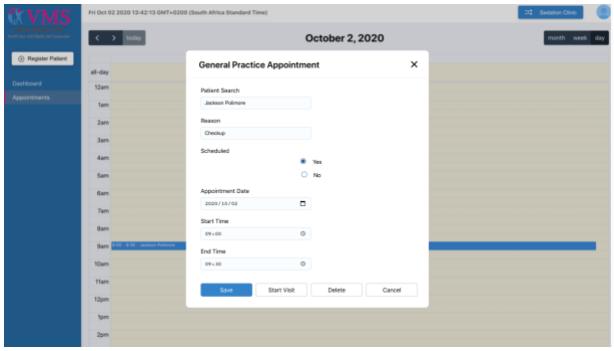


Image 19: Appointment actions

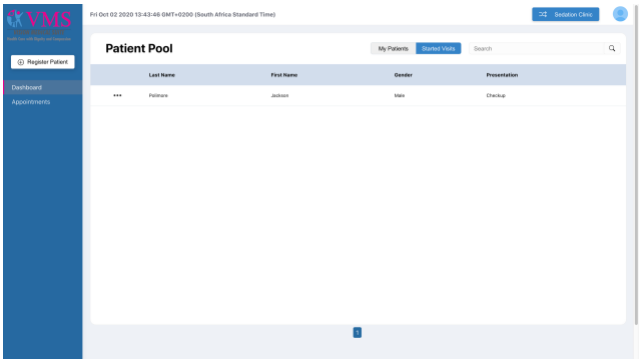


Image 20: Patients undergoing their appointment

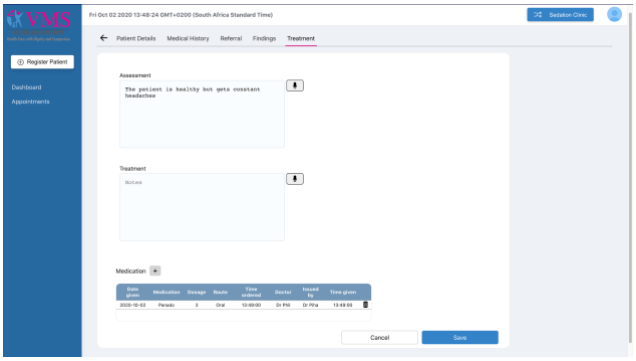


Image 21: GP Treatment Information Capture (with Voice to text)

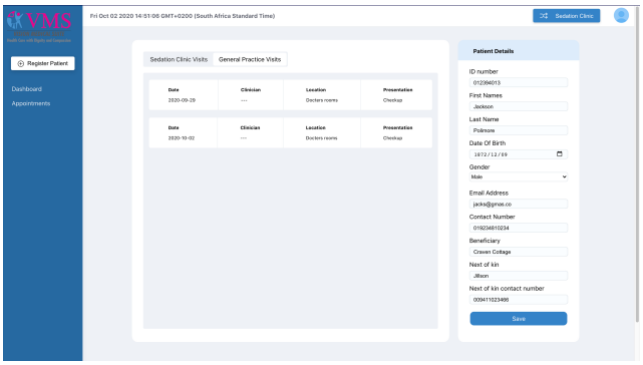


Image 22: GP patient profile

Software Implementation of a Healthcare Management System

Admin Panel

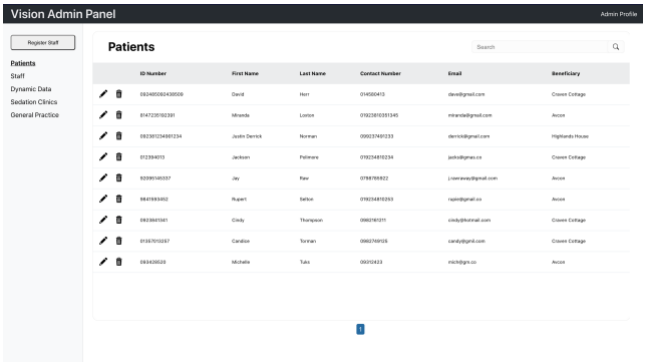


Image 23: List of all patients

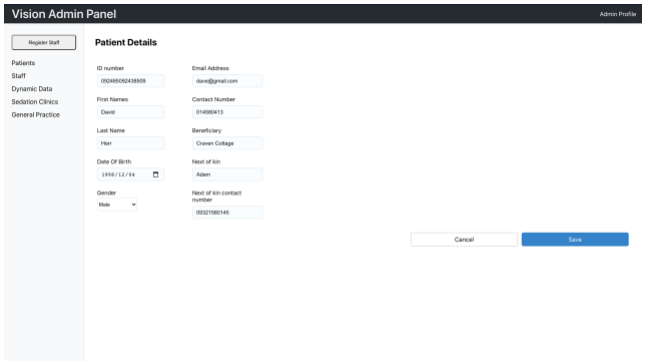


Image 24: Editing patient details

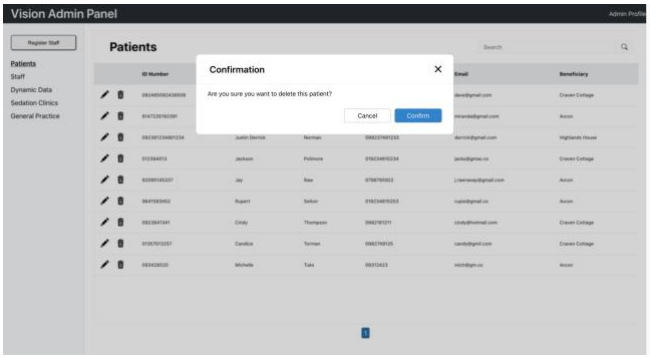


Image 25: Confirmation modal

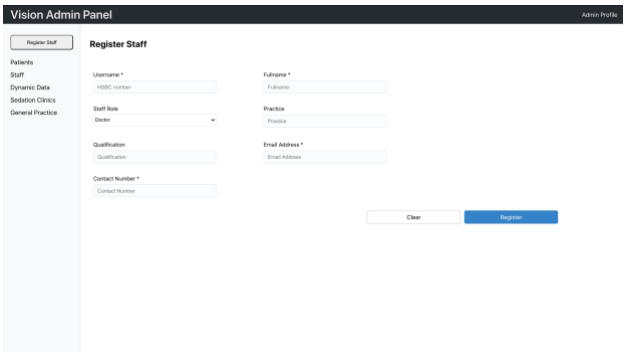


Image 26: Register staff

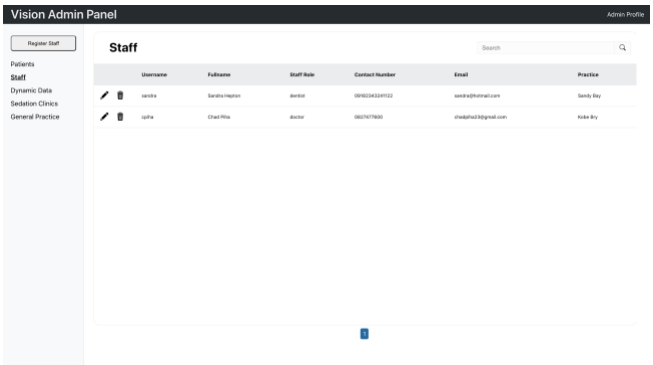


Image 27: List of all staff members

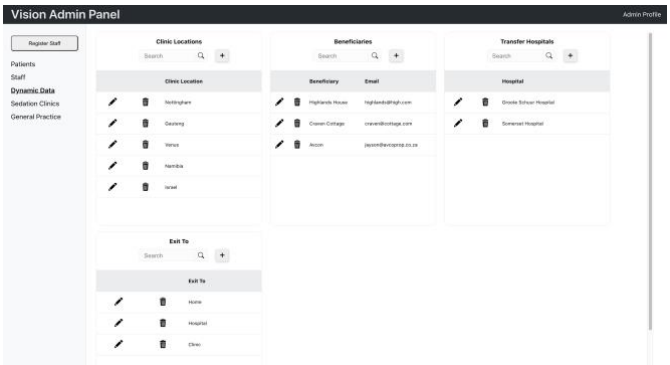


Image 28: Dynamic data

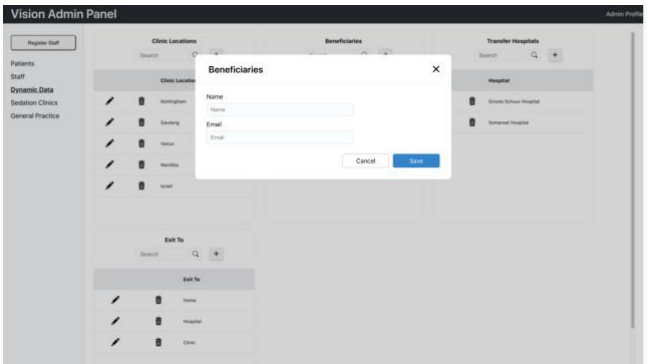


Image 29: Adding beneficiaries dynamic data

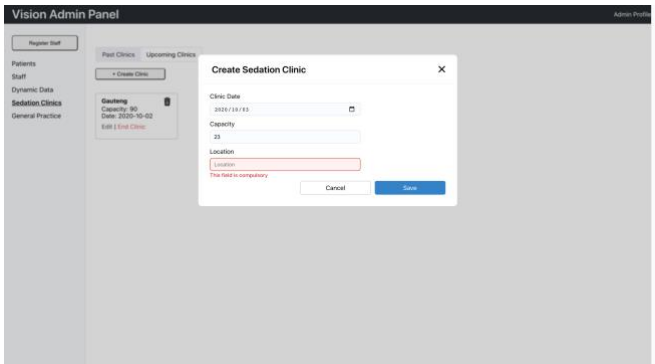


Image 30: Create sedation clinic (with validation)

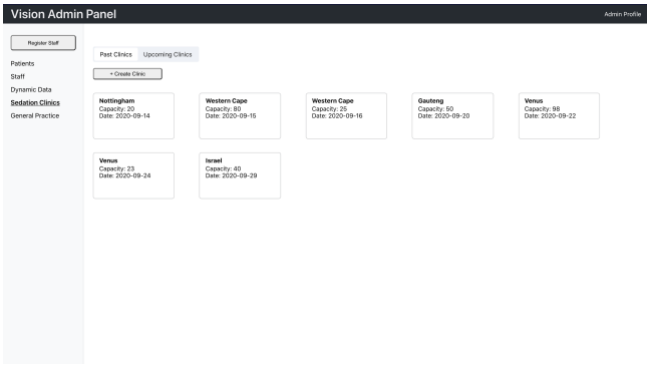


Image 31: Past sedation clinics

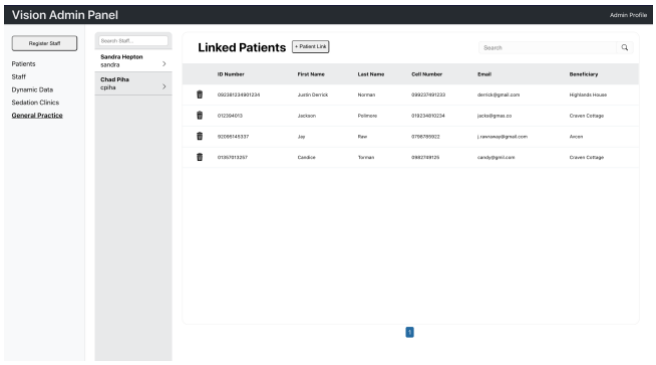


Image 32: Patient-staff link

Mobile Application - Sedation Clinic

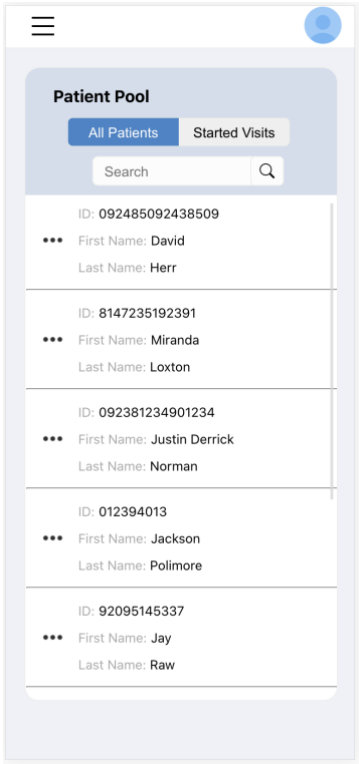


Image 33: List of all patients table

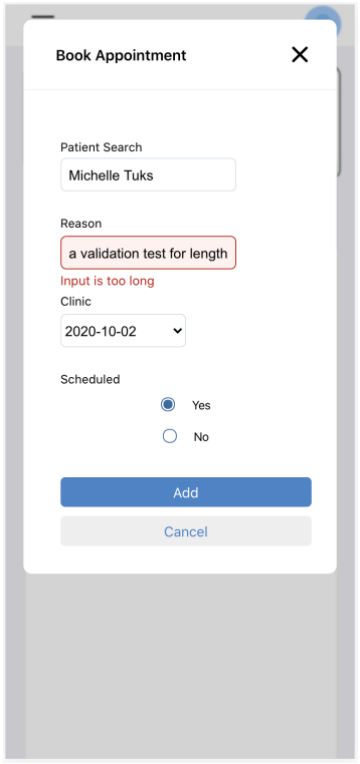


Image 34: Book appointment (with validation)

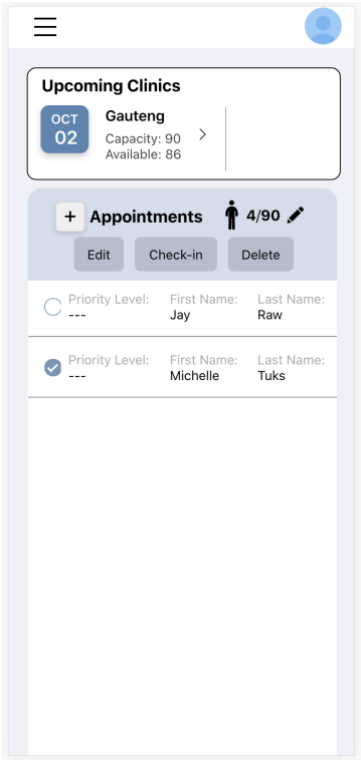


Image 35: Appointments actions

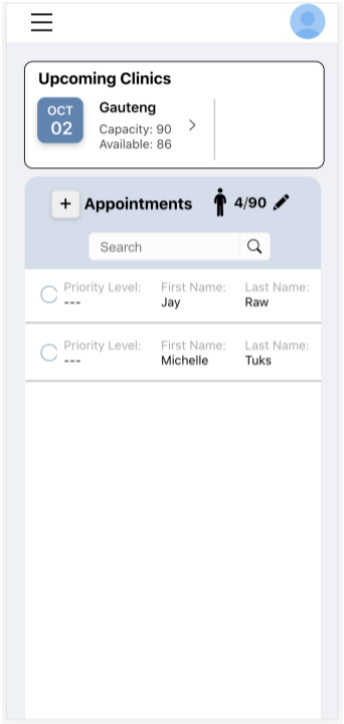


Image 36: Appointments

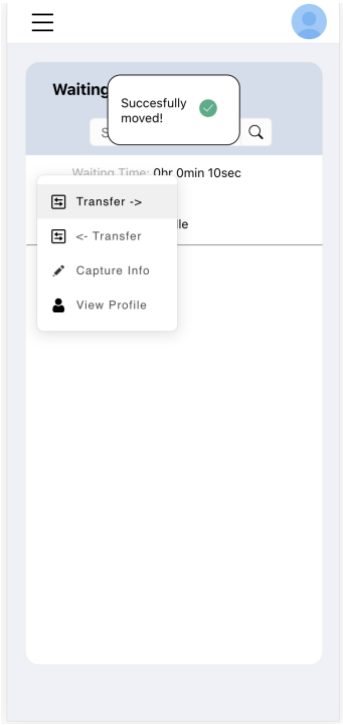


Image 37: Feedback modal

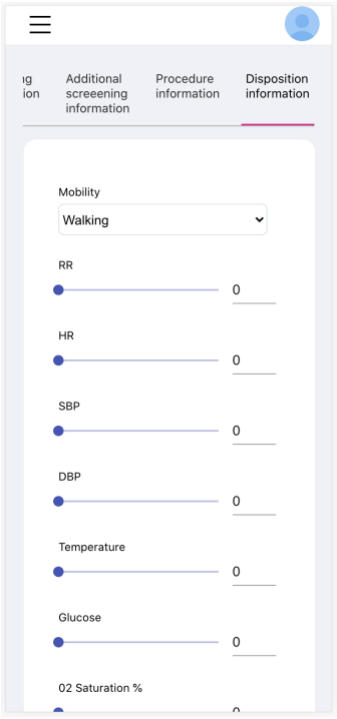


Image 38: Mobile form

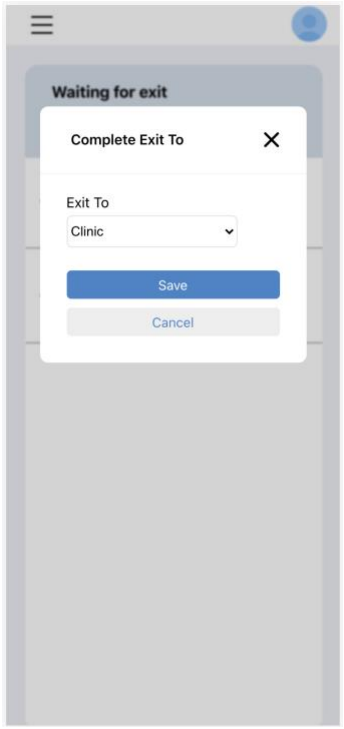


Image 39: Exit to modal

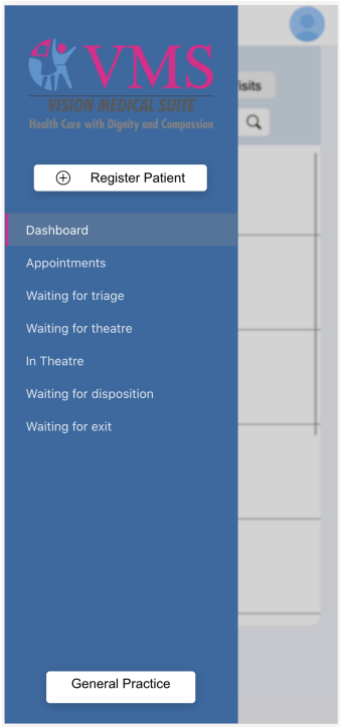


Image 40: Expanded hamburger menu

Software Implementation of a Healthcare Management System

Mobile Application- General Practice:

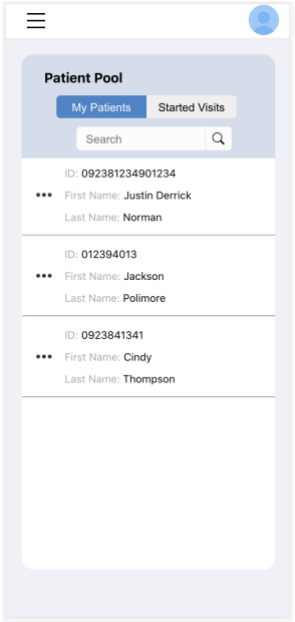


Image 41: Linked patients table

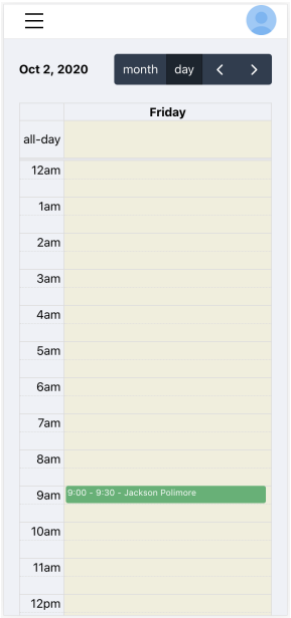


Image 42: Appointments

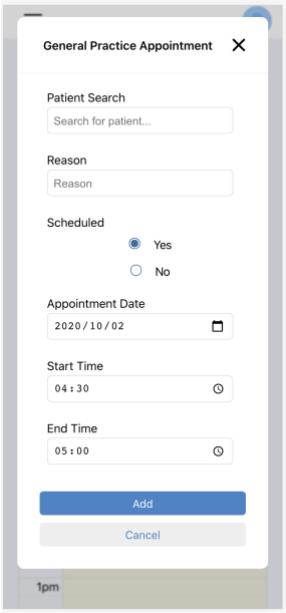


Image 43: Book appointment.

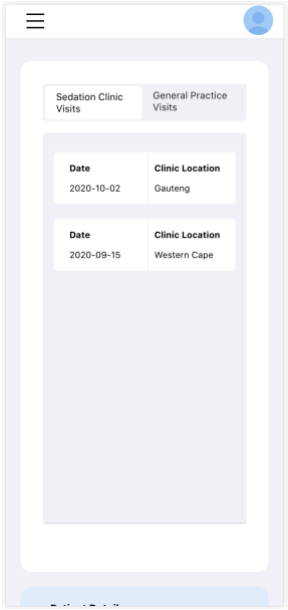


Image 44: Patient profile

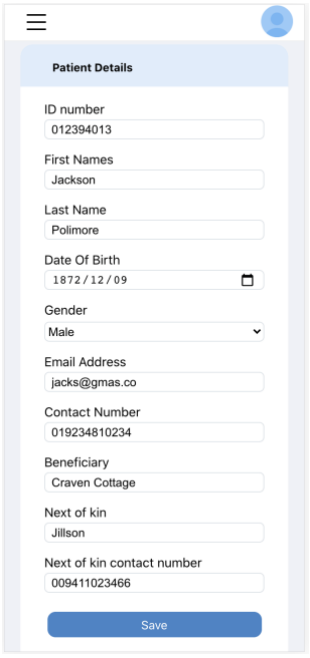


Image 45: Patient profile (cont.)

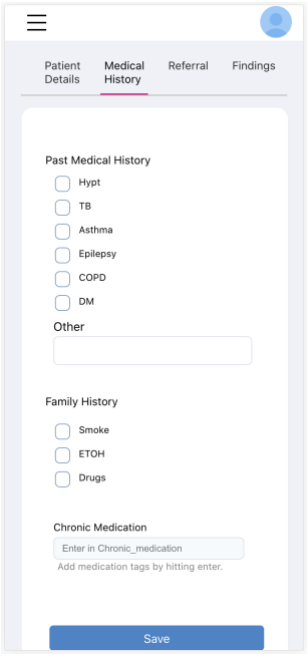


Image 46: GP form