UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: Developing a Hospital Management System

Author: Chad Piha

Project Abbreviation: Vision

Supervisor(s): Aslam Safla

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

Developing a Hospital Management System

A system tailored for Vision Medical Suite

Chad Piha
Computer Science
University of Cape Town
South Africa
chadpiha23@gmail.com

ABSTRACT

Vision Medical Suite is a Non-profit Organization that provides free medical and dental care to those in need. They currently manage all their operations manually, which is simply no longer a viable approach and therefore they require a hospital management system to manage their operations more accurately and efficiently.

Vision has a unique set of operations that no software solutions on the market currently provide for. Therefore, our goal was to develop a full-stack web application that streamlines the management of staff, patients and the various procedures performed in Vision's operations. The system needs to be fast, reliable, secure, and easily maintainable whilst simultaneously making tasks easy to complete, and enabling the management of staff and patients to become a more efficient process which will facilitate the staff's capacity to coordinate care.

A single-page web application was developed using a React.js front-end combined with a Node.js server which made use of the Express.js framework, and a MySQL database. This application includes features such as a booking system, a tailored phase-transition system that captures patient information at each transition, voice-to-text, and an ICD10 API, among other functionalities.

The results of the usability tests found that the UI/UX was intuitive, simple, easily navigable, and visually appealing. Therefore, we were able to conclude that the system was a success in terms of the UI/UX. The non-functional requirements were met, and unit tests were also conducted to verify that the functional requirements of the system were met. However, due to Covid-19, we were unable to deploy the system in the hospital to perform an acceptance test as no clinics were running at the time.

CCS CONCEPTS

• Computer systems architecture → Real-time systems • Information systems → Data management systems • Human-centred computing → Human computer interaction (HCI) → HCI design and evaluation methods

KEYWORDS

Hospital management system, User Interface / User Experience, Information capture, Validation, Error handling, Voice-to-text

1 INTRODUCTION

Over the last 40 years, information systems have progressively become an integral part of modern healthcare, with varying degrees of success in improving patient care. These hospital management systems focus on automating the many vital daily processes in order to streamline the administrational needs of hospitals. This helps them operate more efficiently, thereby creating better hospital records and allowing more patients to be treated, whilst maintaining high standards of care. [1]

Vision Medical Suite (VMS) is a Non-Profit Organization that provides free medical and dental care to vulnerable communities and beneficiaries. VMS has two aspects to their operations. The first is a monthly sedation clinic where they provide their services to beneficiaries such as orphanages, frail care centres, and care facilities for the physically and mentally challenged. The second facet is their daily general practice where they treat walk-in patients as well as members of the beneficiaries. VMS currently manage all operations manually which has created an environment where organization and efficiency are not at their maximum capacity. It is well understood that timely and accurate information is a fundamental pre-requisite for the delivery of high-quality patient care, therefore these inefficiencies need to be addressed. [2]

Vision lacks the necessary funding to outsource the development of a hospital management system from a third party and have therefore liaised with the University of Cape Town for assistance. As a result, our team has been tasked with developing a full-stack web application that streamlines the management of staff, patients and the various processes executed in Vision's operations.

The main aim of the system is to heighten the efficiency of Vision's operations. As a result, the system needs to be fast, reliable, secure, easily maintainable, intuitive and easy to use in the fast-paced, time-critical environment that characterizes Vision's operations. The core functionality of the system requires staff to have the ability to register patients, book appointments, capture and store patient visit records for future viewing, and manage patient transitions throughout their hospital visit for both the sedation clinic and general practice. The system also requires an admin, who are the higher organizational level users, and are responsible for

creating sedation clinics, registering and managing staff roles and permissions, to ensure access control. This system should also be compatible with desktop, tablet and mobile devices.

This report will begin by summarizing the background research performed before development began. It then describes the requirements analysis and all the functional and non-functional requirements of the system. Thereafter, the design of the system is explained and all decisions are justified. The system development and implementation are then critically discussed, with the focus on how the design was implemented, and the process of how the requirements were met and the final product was developed. The testing process and subsequent results are then examined and are linked to the success of the project. The paper is concluded by measuring the overall success of the project, measuring the possible impact this system has for Vision and finally giving a brief reflection on the entire project by outlining possible future extensions, and providing recommendations to those trying to build a similar system in the future.

2 BACKGROUND RESEARCH

Extensive background research was performed to ensure that the team had a thorough understanding of how to implement a system of this nature in the best way possible. Each team member based their research on the sections they were responsible for. It was also important to understand what ethical, legal, and professional limitations that could arise and thus needed to be accounted for.

2.1 Standardization formats

As I was responsible for the information capture, it was important to research the different data standardization formats as the data being transmitted from the web app to the database needs to be consistent. The two most popular alternatives are JavaScript Object Notation (JSON) and Extensible Markup Language (XML).
According to Šabanović et al. one should use JSON technology when dealing with a small amount of data. JSON is faster as it consumes far less memory space, and it is closer to the original object structure. XML should be used for larger data transfers, however, when segmentation is done for large transfers of data, one should use JSON. Šabanović also states that JSON should be adopted for web applications as it is a very thin object which can be communicated to any application. [20] JSON is also a subset of JavaScript and provides an ideal format for exporting data from a web app to a database since every programming language can parse JSON. [21] This makes it a natural fit for web applications and therefore, JSON was our chosen standardization format.

2.2 User Interface/User Experience (UI/UX)

The UX works conjointly with the UI to create an effectively designed application. A seamless design is imperative as it has a direct correlation with the rate at which users can accomplish a task, which is important in the system being developed. [26]

A study by Silvennoinen et al. showed how colour is seen as an organizer of information and how it creates an overall clearer impression of the system.[28] A white background was found to provide a high level of readability and quick perception of text-based content. [27] Moreover, it adds space to the screen which avoids the feeling of the pages feeling overloaded. [33] This should be used in combination with the colour blue as it symbolizes healing and tranquillity. This will also show contrast, allowing for greater scannability as text elements can be read at a glance, thus improving task-completion rates. [30]

For feedback, it is good practice to have a dialog box indicating the success or failure of user actions. [10] All validation errors should be highlighted in red as the colour is associated with error, and it grabs the user's attention. [4] Validation errors should also be in-line with the input. A study has shown that inline validation has a 22% decrease in errors made; a 31% increase in satisfaction rating; a 42% decrease in completion times; and a 47% decrease in the number of eye fixations. [8] These forms of feedback adhere to Shneiderman's golden rule of interface design of "Offering informative feedback." [5]

It was found that one should use a hamburger menu on mobile devices as it saves limited screen space while allowing information to be presented more clearly as pages are less busy. This improves usability and speed as users can get to the desired screen easily and quickly, which is important in a hospital system. CSS3 should also be utilized as it can reduce data transfer, particularly with images. [26,31]

2.3 Related Work

We analysed some of the market's leading open-source and paid hospital management systems, namely Sapphire, Medstar HIS, eHospital, Athenahealth, TeamDesk, OpenMRS, and OpenVista. The functionalities common across all these platforms include staff, patient, financial, inventory, bed, and claims management systems, as well as online appointment scheduling, HR management, dedicated patient portals, and native applications in conjunction with web applications. [13,14,15,16,17,18,25]

All of these services are offered on a more general management scale and are catered towards much larger organizations that have many more departments requiring many different functionalities. Vision, however, do not comprise all of these departments and therefore only require few of the functionalities provided by these other systems. A big system with a large range of non-essential features may result in a degree of impracticality and may reduce user efficiency. [34] Furthermore, Vision's monthly sedation clinic is a unique service that none of the other systems on the market currently cater for.

The main constraint with the paid systems is that they require expensive licenses, which Vision, as an NPO, cannot afford. We investigated the possibility of using an open-source system and adding this functionality to it, but although most of the code is

componentized, there will be a steep learning curve in trying to understand the system. Additionally, most of the forms did not contain all the fields Vision need to capture for each patient, therefore we would have to redesign the majority of the system, from the front-end to the database. Additionally, in-house development would allow for greater control over the system as well as greater customization. Therefore, the decision was made to develop the entire system in-house.

2.4 Ethical, professional and legal considerations

2.4.1 *Professional considerations*

Once the system is complete and deployed, it will be owned by UCT and Vision Medical Suite. The long-term maintenance of the application will become Vision's responsibility; however, we have ensured that all the code is fully documented to make it easily understandable and easily maintainable for future developers. The documentation can be found on the project website.

2.4.2 *Ethical considerations*

We received ethical clearance from the UCT Human Research Ethics Committee before we conducted usability tests with real users. To ensure ethical standards, users were required to give signed consent before they participated in the study. Furthermore, all tests were conducted over a video conference call to avoid the risk of exposure to Covid-19. These tests were recorded, and all information collected will be kept private so that the participant will not be identified by name or by affiliation to an institution. All audio and video recordings will be kept for no longer than a year. Confidentiality and anonymity were maintained as pseudonyms were used, and it was emphasized to participants that all data entered should be fictitious. All meetings with the clients, supervisor, and other team members were also all held over video conference calls to avoid the risk of exposure to Covid-19.

Since some of the participants that were tested are health workers, there was a potential ethical issue of taking their time during a pandemic. We resolved this by ensuring that the tests were conducted as efficiently as possible and to the convenience of the workers when the active number of Covid-19 cases were low.

2.4.3 *Legal considerations*

The final application will deal with real, sensitive patient data. Therefore, security was a priority in the design of the system in order to comply with the Department of Health's requirements around patient data. This was achieved by incorporating authentication, authorization, encryption and various other security measures into the application, which ensured access control, data security, and confidentiality.

The Protection of Personal Information (POPI) act was also taken into consideration. This states how institutions should act when storing and sharing an entity's personal information, which ensures accountability. We ensured that doctors are only allowed to view the information of patients who are linked to them, and only the admins are allowed to link a patient with a doctor. This ensures access control and doctor-patient confidentiality. Information is also encrypted when stored.

3 REQUIREMENT ANALYSIS

The requirements analysis process began in a meeting with the stakeholder who verbally provided us with a scope of what the functional and non-functional requirements of their desired system are. This meeting was conducted over a video conference call to prevent exposure to Covid-19. The stakeholder ran through a similar system that is provided by the state, where he pointed out what needs to be included in their desired system, as well as the flaws and inefficiencies of the system demonstrated. These flaws included unnecessary and unorganized inputs, complicated user navigation, cluttered screens, and a poor UI. He left us responsible for making improvements to the aforementioned inefficiencies.

He explained that the new system needs to consist of two parts. The first part of the system is the general practice, which is the standard practice where they provide their services on a daily basis. The second part is the sedation clinic, which is run on a monthly basis and aims to treat 50 to 100 patients in a single day.

3.1 Requirements/Features

The team compiled a list of all functional and non-functional requirements by re-watching the meeting recording. A more detailed list of the functional and non-functional requirements can be found in Appendix A, however, this section highlights the main requirements.

3.1.1 *Functional requirements*

The functional requirements will be divided up between the three main parts to the system.

3.1.1.1 *Admin Panel Functionalities*

Admins should be able to register staff members and create login details for them. The admin should also be able to add, edit, and delete beneficiaries, hospitals, and sedation clinics. Admins should also be able to control which staff members have access to which patients (in order to comply with the POPI act).

3.1.1.2 *Sedation clinic functionalities*

Once an admin has created a clinic, hospital staff should be able to schedule patient appointments on behalf of beneficiaries. On the day of the clinics, the patients can be checked in when they arrive at the hospital, and a staff member can then start their visit. Each visit will consist of 5 different stages, where different information is captured at each stage, and stored for future viewing at any given time. The patient should be exited from the hospital once all stages have been completed. Staff should also be able to view the patients previous sedation visit records at any point in time. Voice to text is also required for the procedure information. Staff members should also be able to register new patients anytime.

3.1.1.3 *General practice functionalities*

Staff members should be able to view patients linked to them by admins, and book appointments for them. They should also be able to capture all relevant information about that patient during their visit, which can be viewed in the future. All details including past visit information, sedation clinics, appointments, patient, and staff information should be able to be edited and deleted by those who have permission. There should be voice-to-text functionality for input fields which are likely to require a lot of text, and validation should be implemented throughout.

3.1.2 *Non-functional requirements:*
Throughout the entire system, there should be authorization and authentication protocols, patient details should be confidential, there must be efficient retrieval of information, an intuitive and easy to use UI/UX, and adequate security for a project of this nature.

3.1.3 *My responsibility*
We split up the entire system according to its functionalities and divided them up evenly between the three team members. In dividing up the work we played to each other's strengths and previous development experience to ensure that the development process will be as efficient as possible. The functionalities I was responsible for were: Information Capture for the general practice and sedation clinic, the UI/UX of the whole system, the editing of all patient details and their visit information, the mobile application, screen responsiveness for tablet devices, form validation, error handling, feedback, voice-to-text functionality, prototyping, and QA and testing.

3.1.4 *Team member responsibilities*
One member is responsible for the authentication and authorisation, the database design and queries, as well as all the server side functionality. The other member is responsible for the frontend state management, security, the component structure, performance enhancing techniques, the admin panel and managing sedation clinic and general practice appointments.

3.2 Analysis

Once we had a list of all the requirements, we drew up a use case diagram to specify the actions of the different actors of the system, and to ensure we had a good high-level understanding of the main requirements of the system. Access control is an important aspect that necessitates consideration as the system needs to comply with POPI, therefore we needed to be clear on which actors could perform which actions. This was then sent to our supervisor who approved of it. This can be found in Appendix B.

An ERD diagram was also developed for the database. This was to illustrate the entities of the system and the relationship between these entities. The group member responsible for the database was responsible for this task, as found in Appendix C.

Meanwhile, I was developing a horizontal, evolutionary prototype for the system. I had to ensure that the prototype captured all the requirements and information that the stakeholder requested. The first iteration focused on ensuring that it captures all the requirements outlined in the meeting so that we do not waste time in the development process trying to add in missing requirements. The first horizontal prototype iteration was meant to be reviewed by the stakeholder, however due to Covid-19, it was hard for him to find time and thus only viewed the second iteration. The second iteration of the prototype was focused on making the changes suggested by a UI/UX designer, and demonstrating it to the stakeholder.

4 SYSTEM DESIGN & IMPLEMENTATION

4.1 Design

4.1.1 *Prototype and general UI/UX design*
The design process began by breaking up the system design between the database and server (found in Appendix D), the structuring of system components (found in Appendix E), and the prototype. As I was responsible for the UI/UX, it was my responsibility to develop the prototype. This prototype was developed using Adobe XD which is a specialized prototyping tool for web and mobile applications. [35] I was responsible for engineering ways in which we can overcome and improve upon the aforementioned inefficiencies of the previously demonstrated system. I used an agile development process in developing the prototype. It began with the planning phase where I performed extensive research into UI/UX best practices, as summarized in section 2.2. I first created a low-fidelity wireframe sketch of the system.

I then began the design of the next prototype iteration based on the wireframe. The first thing I took into consideration was the colour scheme for the system. As a result of the research performed in 2.2, the colour scheme consisted of three different shades of blue, a grey and white background, and pink for small accent details (since pink is the colour of the VMS logo). The different shades of blue aimed to create a visual hierarchy that organizes information and enhances clarity. The white background adds space to the screen which allows for the quick perception of text-based content. This colour scheme has a high contrast which improves readability and scannability. The next aspect which was redesigned was the navigation bar. This needed to be redesigned from the demonstrated system as its layout was unorganized and cluttered, since the navigation bar was presented as tabs across the top of the page, with inadequate feedback as to which tab was selected. This meant that users on a smaller tablet screen had to scroll horizontally to see all the tabs, which is inefficient. As a result, I moved the navigation bar to the left of the screen with a blue background to stick with the colour scheme. The navigation item selected was also highlighted in white. With regards to the form design, all related inputs are grouped, with each of their labels to their left. This conforms to Gestalt's law of proximity. [9] The size of the input also relates to the maximum length of the input as per the database limits to give the user a hint as to what the maximum length of the input is (e.g. an input field with a maximum length of 20 characters will have a

smaller height/width than an input field with a maximum of 50 characters). The stakeholder also requested for sliders to be included in the forms. This could be considered an inefficient form of input, thus all sliders will have an input field next to them to give users an option to type in a number instead. Selected screens from the first prototype can be found in Appendix F.

To ensure I was using the best UI/UX practices, I had a meeting with a UI/UX designer in the industry who reviewed the prototype. He informed me to move the labels above the inputs, and to move the buttons to the bottom right of the form to enhance simplicity. He also recommended that the navigation item styling should change, and the tables should be redesigned and made a lot simpler. He praised the colour scheme, the design of the validation feedback and the patient profile.

I then began the second iteration of the prototype. I made the selected navigation item in a lighter colour with a pink accent to create a sleeker look, and to provide adequate feedback on which item is selected. All the other changes were made as recommended by the designer before the prototype was demonstrated to the client in another meeting. He reviewed the prototype and informed me of a few inputs that were missing. Those inputs were added and we received approval to begin development as the stakeholder was happy with the UI/UX. Selected screens of the final prototype can be found in Appendix G.

The design of the mobile application began once the desktop web application was approved. The mobile version went through a similar agile development process. For the first iteration of the prototype, the same colour scheme as the desktop was used. The header was adapted to include a hamburger menu that toggles the navigation bar. I had to redesign the tables as there was not enough space on the screen for them. They were redesigned to show only the three most important pieces of information at that phase in their hospital visit due to the limited space in the tables. The layout of the forms was also redesigned as the input fields all had to be in a single column to fit on the screen. The validation and error handling and feedback all remained the same. The buttons fit the entire width of the form container to make them easily pressable.

I had another meeting with the same UI/UX designer I spoke to for the desktop prototype. He was happy with the overall UI/UX design especially with the effective ease of navigation that the hamburger menu allows. He did however say I should redesign the tables, and gave me a design that he recommended I replicate. I replicated that design, and completed the second iteration of the prototype for the mobile version, as found in Appendix H.

4.1.2 *Information capture*

On each stage in a patient visit, several inputs need to be filled in and saved for future viewing to create patient medical records. With the press of a button, all the data needs to be collected from the form and sent to the server which will post the data to the relevant tables in the database. The team member responsible for the server and database specified the layout in which he wanted the data to be sent to the server, which I had to reproduce. This conformed to best practices and made his job easier as it required less formatting on the server-side. I handled capturing the data on the front-end and sending it to the server where it was then dealt with by another team member.

POST and PUT requests were used as they are methods supported by HTTP to make the server accept the data enclosed in these requests. When sending these requests from the web app, it will make use of Axios, an external ReactJS library, which is used to make promise-based HTTP calls. Axios was used as it simplifies HTTP requests and performs CRUD operations in a simple manner. It also provides convenience as it performs automatic transformations for JSON data. Axios also works very well with Node.js and Express.js, and using these libraries and frameworks is considered best practice for web applications of this nature. [11]

4.1.3 *Form validation*

It was imperative to correctly design and implement client-side validation to protect against bad form data being posted to the database. Every user input throughout the application should have validation to protect against this. In designing the forms, every input field is an object that has validation rules that need to be satisfied. Many of these inputs have the same rules, therefore it was important to create a reusable function to check the validation properties. The validation checks that are made include, but are not limited to, compulsory field checks, whether a field should only contain alphabetical or numerical characters, maximum and minimum value and length checks, if a date or time is valid and if it is a valid email address. I used inline validation for the reasons discussed in 2.2. If an input field is invalid, the input field will turn red, and a custom error message will appear underneath stating what is wrong with the entered input. This gives appropriate feedback to the user as it shows exactly where the error occurs and what the error is. If a user tries to submit the form while there are still validation errors, the invalid fields will be highlighted in red and a message will appear at the top of the form indicating to the user that the indicated fields need to be fixed. The form will not attempt to post until all validation errors are fixed.

4.1.4 *Error Handling*

A piece of software on this scale is likely not going to be bug-free due to time limitations of not being able test every single way the user interacts with the system. Therefore, sufficient error handling needs to be put in place for the application to gracefully respond to errors and exceptions. This is important for a good UX. The team member who designed the server sends a status response code in response to every request made from the web app to the server. The error handler makes use of this to show a custom error message for each type of status response code, which ensures that the right message is shown to the user. The server provides 400, 401, 403, 404, and 500 client and server error responses, each of which the error handler should respond to with a separate message to give the user adequate feedback. The error message will be displayed clearly as a modal which forces the user to complete a specific action to ensure that they respond to the error correctly. This

conforms to Nielsen Norman group's guidelines on error messages being human-readable, polite, visible and it advises the user on what to do. [10]

4.1.5 *Responsiveness*
The entire web application needs to be compatible with tablet devices. I used CSS3 Media queries as it provides a dynamic way to make the application responsive to different screen sizes. These media queries targeted specific screen sizes, and uniquely styled each component in the application based on the screen size. The media queries created are compatible with tablets between the screen-width of 768px and 1400px. This includes, but is not limited to, the iPad, iPad Pro and Samsung Galaxy tablets. The main purpose of this was to ensure that everything was displayed clearly on the screen, and that information was not cluttered.

4.1.6 *Voice to text*
In designing the voice to text, the decision was made to use a free Web Speech API that uses your built-in microphone to capture your voice. Unlike other paid APIs, the Web API is limited to use on Google Chrome only, but the saved costs massively outweigh its inability for browser compatibility (as the paid versions charge an amount for every second recorded, and every device can use Chrome). Due to the zero budget of the project, this was the right decision. The voice to text was only implemented for text areas where excessive typing is expected, such as findings and clinician notes. The voice to text prints out directly to the text area as the user speaks into the microphone, therefore it does not need to store the voice note in the database, thus saving space. All mobile and tablet devices have voice to text functionality built into their keyboards; therefore, this was only necessary for the desktop version of the web application.

4.2 System development and implementation

4.2.1 *Approach*
We followed an agile development process in developing the system. The agile development process was beneficial to our team as we did not have experience in developing a system of this size, therefore this approach allowed for change in terms of features and algorithms used. Furthermore, since we were all working on different parts of the same system, we needed to be regularly merging our sections to ensure that we could test that they worked in conjunction with the other team members' parts. We used sprints which lasted 2 weeks each, and the tasks set out for each sprint were managed on Jira, an agile software development tool. This helped us stay organized and motivated by setting goals for each sprint. In order to ensure that all team members were held accountable for their actions and that we communicated effectively, the team had daily stand-up meetings. In these meetings, we discussed what we did the previous day, any problems that arose and what we planned to do that day. We used Slack to communicate with each other about anything to do with the project.

4.2.2 *System architecture*

The front-end was developed using React.JS which is an open-source Javascript library for building UI components and developing single-page applications. React.js was used as it provides a Virtual DOM where all changes are stored. Therefore, once a change is committed, the Virtual DOM is compared against the real DOM, and only the related component is updated. This results in fewer page re-renders, thus improving application performance. [3] This is important in a Hospital management system as staff often find themselves in time-critical situations. Furthermore, all team members had experience using React, and due to time constraints, it would not have been beneficial to attempt to learn a new language in the short space of time we had. The state of the application was managed using a library called Redux, which was set up by another team member. This provided us with a central store for the state of the entire application, and allowed the state of all data to be accessible throughout the application.

We used a Node.js application server. This was used in conjunction with the Express.js framework which allowed us to build APIs, handle data updates from the front-end, and build a scalable network application that can process multiple simultaneous user requests. We used Node because React code can be executed directly in the Node.js environment, and the React DOM has components that are specifically designed to work with Node.js to reduce the lines of code and make server-side rendering easier when compared to other backend alternatives. [7] The system will be hosted on UCT servers.

We used a relational database run on a MySQL server. This was appropriate for our system as it relies heavily on complex relationships, which relational databases are built for. For example, for each patient visit, a visit instance will be created, which has a many to many relationships with the patient table. Any other information to be captured during the visit will be contained in its appropriate table, which has a one to one relationship with the visit table.

The frontend and backend work together as follows: The front end sends HTTP requests using Axios, which the webserver processes. It then sends these requests to the application server. The logic developed in the server then determines what information it should request from the database [22, 23]. This logic is the API (Application Programming Interface) which defines different routes that will be called from the frontend. These routes each represent a different action that makes requests to the database to create, remove, update, or delete information. This data will then be returned to the frontend where it will be managed and dealt with to present to the user. JSON will be used as the standardized format to interchange data between the client and the server. This API will follow a REST architecture which means that any interaction with the API requires that all information needed to perform a specific task must be provided. This increases the performance and scalability of the server as it allows the server to remain stateless. [24]

4.2.3 *Functionality* Implementation

4.2.3.1 *UI/UX*

The prototype was replicated in the structuring of the web application pages, before moving on to structuring the forms and starting the information capturing. The forms, along with the rest of the UI was developed using JavaScript, HTML, and CSS3 Modules. We chose to use CSS3 Modules as the styles are scoped locally so that they do not leak into other components. The CSS was loaded into Webpack, which builds the module and allows for its use throughout the application. A module was created for every component and container, and the relevant module file was imported inside the target components.

Feedback is also important for a good UX as it informs the user of the success of their actions. The feedback was implemented using a reusable modal class that had its state saved in the central redux store. When a user performs an action, the server responds with a status code indicating if it is successful. If it is successful, it sets the state of the modal in redux to true, which allows it to pop up on the screen. The feedback is toggled back to false after 3 seconds, which is adequate time for the user to view the message. All the screens of the final system can be found in Appendix I.

There are also confirmation dialogues for whenever data is added, edited, or deleted to avoid user errors. The state of these dialogues are stored in the local state and are triggered on button click, once all validation checks have been completed.

4.2.3.2 *Information Capture*

Each input field is stored in an object, with its current value as part of that object. As users type in a text field, this value gets instantly updated using a custom handler function which I created. When the button to submit the form is pressed, and all the validation checks have passed, the information capture process begins. When the form is submitted, using Axios, it fires an action that sends a POST or PUT request to the Node.js server, and Express.js saves the data in JSON format in the database. When the promise resolves, we get the JSON data, and another action calls the reducer function and saves the data in the Redux store so it can change the UI state which allows the user to see the data that was posted. Another team member was responsible for the server correctly handling the data and posting it to the database.

4.2.3.3 *Form Validation*

Validation checks are all contained in a utility class that is imported by every form. As mentioned in the design, each input field is an object stored in the local state which has a list of validation rules it needs to satisfy for the input to be valid. This object also contains ‘valid’ and ‘errorMessage’ properties. For every keystroke which changes the value of the input field, the value and the list of validation properties of that input get sent as parameters to a function in a utility class. This function iterates through each validation rule and tests the value of the input against a regex pattern to check if the value satisfies that property. If it does not satisfy the property, it sets the ‘errorMessage’ property of that object to a custom error message based on the failed property; and sets the ‘valid’ to false. This valid property and the error message then gets passed as props to the Input class where the input field is updated based on the value of ‘valid’. If valid is false, then the CSS class will dynamically change which will highlight the input field in red, and an error message will appear below this field stating exactly what is wrong with the entered text. When the user fixes the error, the checkValidity function will set valid to true, and this value will be passed to the Input class where it will update the CSS to change the input field back to its normal display. If the user presses the button to save the form details, it will check the local state and see if there are any invalid fields. If there is, an error message will appear indicating what fields need to be filled in or changed, otherwise, the information capture process will begin.

4.2.3.4 *Error Handling*

To ensure error handling, I created a higher-order component that wrapped the app class, which is the main component. This higher-order component uses a response interceptor which is a method that is triggered before the main method and is called before the promise is completed and before data is received by the callback. The interceptor also makes use of Axios. In this higher-order component, this interceptor receives a status response code which is sent from the server to the web application with every single request. Based on the status request code, an error message specific to that request will be saved in the local state and is passed to a reusable Modal component which appears when there is an error message present in the state. For example, a 401-status response gives an error message stating that the user has been logged out, and it redirects them to the login page. This allows the web application to respond to errors gracefully. By wrapping the app container class with this higher-order component, it allowed the component to react to errors regardless of where you are in the application. This error handler only catches runtime errors.

4.2.3.5 *Mobile application and Screen Responsiveness*

The mobile and tablet applications were both created using Javascript, HTML, CSS3, and Media queries. For the mobile, the entire application had to be redesigned and resized to make it compatible with mobile devices. This was achieved using media queries that restyled all components when it picked up a screen size which had a width below 480px. In developing the mobile application, the mobile prototype was replicated as it had received approval from a UI/UX developer in the industry. Therefore, I was happy with all of the design decisions made, and they were all justified in 4.1.2.

For the tablet version of the application, no prototype was developed due to time constraints. Therefore, I used my previous HCI knowledge to redesign the various aspects which did not fit on the screen. When the screen size gets smaller, the width of the navigation bar gets smaller which gives more space for the all tables and forms. The tables also had to be fully responsive based on the screen size to ensure that it was still presented clearly. This was done by setting a priority level to each column of every table, which allowed certain columns to automatically be hidden based on the screen size and their priority level. The layout of the forms also becomes a single column when the screen gets too small. It was

also important that the whole app was touch-friendly. To achieve this, the buttons were made slightly bigger as using your fingers to touch items are far less precise than using a curser. I also took advantage of the built-in speech to text of tablet keyboards which allowed me to remove the voice to text buttons. The application's ability to be used on any device, as long as it has Chrome downloaded, allows it to be portable.

4.2.3.6 *Voice to text*
The voice to text functionality uses a Web API. This API involves receiving speech through the device's microphone, which is then checked against grammars in the English (United States) language. Once a word or phrase is successfully recognized by the grammar, it is returned as a text string. This is all managed from a SpeechRecognition main controller interface which sends the audio to Google's servers to perform the transcription (This is why it is online supported in Google Chrome). When the microphone button is pressed, I send a method to toggle the speech recognition controller which turns on the mic and begins to listen to the user. Then using a custom handler function, it adds the transcription to the value of the input field, then updates the value in the input's object state, and saves it in the local store ready for it to be posted to the database.

5 Testing

To test the system, we conducted usability tests at two different phases of our agile development process. As a result of Covid-19, all the tests were conducted over a video conference call to avoid the risk of exposure to the virus. The users all received a set of tasks that they were told to complete, while the team member conducting the test took field notes on the interactions and behaviour demonstrated by the user while they used the system. This was to measure the effectiveness in the design of the UI/UX. The list of tasks can be found in Appendix J. The user was told to use fictitious data throughout the test, and they were required to read through and sign a consent form (approved by the UCT Ethics Committee) in order to participate in the test and allow for it to be recorded. Usability tests were important as the design needs to allow medical practitioners to complete tasks quickly and with ease, as they often find themselves in time-critical situations. Unit tests were also performed to test if the functional requirements were working correctly.

Initially, we also planned on conducting an acceptance test that involved deploying the system in a VMS clinic to test and monitor it. However, due to Covid-19, there have not been any clinics, hence we were unable to perform the acceptance test. As a result, we have not been able to test the system in a busy environment with many users interacting with it simultaneously.

5.2 Phase 1 Usability tests

The first phase of tests started once we had developed all the functionality of the system. We conducted six cognitive walkthroughs with people who have a computer science background. We decided not to test any medical practitioners as we found that it was unethical to take their time during this period where the number of COVID cases is rising rapidly. We wanted to test the UI/UX of the system and ensure it was learnable and tasks were easy to complete.

5.3 Phase 2 Usability tests

The second phase of tests started once we were close to finalizing the UI/UX and all the problems identified in phase 1 were fixed. We conducted four cognitive walkthroughs with a mix of people in the medical industry and people with a computer science background. We used medical practitioners in this phase of testing because South Africa had recently entered phase 1 of the lockdown, therefore it was an acceptable period to begin testing them as the number of cases of COVID had slowed down. These tests aimed to test the UI/UX of the system and ensure it was learnable and tasks were easy to complete.

As part of the phase 2 tests, we performed a final demo for the stakeholder.

5.4 Unit tests

We performed unit tests using Jest, a JavaScript testing framework. We used Jest because it is easy to setup in React, and its parallelization allows for very fast tests. [32] On the frontend we performed 76 unit tests which covered all of the functional requirements of this system. Our aim of these tests was to test the effect of the various action types on different instances of the central store. On the backend, Jest was also used to test various endpoints and their effect on the database.

6 Results and Findings

6.1 Phase 1 Usability tests

The feedback will be split up between the three main sections:

6.1.1 *Admin panel*
Registering staff, adding/editing/deleting staff details, sedation clinics, and dynamic data were all easily accomplished and were found to have a very simple, intuitive, and easily navigable UI/UX. Users also appreciated the simplicity of the navigation bar. However, an issue in the admin panel is that there was a lack of feedback on actions that left users confused as to whether or not their task was successful. We found that it was also unclear about how to link patients to corresponding doctors.

6.1.2 *Sedation clinic*
In the sedation clinic, we found that the input field labels were sometimes ambiguous which confused the user. Users also found the size of the buttons were too small on tablet and mobile devices. For some of the inputs, we gave users multiple ways of inputting the information which turned out to be confusing as they did not know which way to use it. It was also not immediately apparent to users where to edit the capacity of the clinics. This is interesting to

note as the UI/UX designer approved of the design of this feature. However, on the mobile version, users found it intuitive to change the capacity. Furthermore, the users found it easy to book and edit appointments, check-in patients, and start their visit to the hospital. Three users also noted that they liked that they were able to do all of this on a single page. Two users did however mention that there should be confirmation dialogues before deleting and editing appointments. Users also found it very intuitive to move patients between the different phases of their hospital visit. Four users mentioned that the validation was implemented effectively and it helped them clearly see where their mistakes were. Six users added that the feedback modals were also very helpful, and they appreciated the simplicity of the navigation bar.

6.1.3 *General Practice*

The general practice had largely positive feedback. Users liked the calendar and the ease of booking appointments. In particular, they appreciated the ability to drag-and-drop to move appointments. Users also liked the feedback on successful actions. Two users did however mention that the feedback disappeared too quickly.

6.1.4 *Resulting changes from phase 1 testing*

We simplified the process of linking the patients to the staff by making a list of staff on the side of the screen, where you just click on a staff member and press only two buttons to add a patient. Feedback was also added to all actions performed in the admin panel. This feedback will be presented in the same way as the sedation clinic and general practice as we received largely positive responses on the feedback. The input fields that were unclearly labelled were renamed and the size of the buttons on mobile and tablets were all increased by 4px. For the inputs which allowed users multiple options to add data, we have removed the second option and given a hint on how to input the data. This excludes the sliders as they were specifically requested by the client. The format in which the clinic capacity can be changed was also redesigned and moved to the top of the appointments table to make it more intuitive. Confirmation dialogues were added for all the editing, adding, or deleting of information. The feedback was increased to last an extra second, and was made consistent throughout.

6.2 Phase 2 Usability tests

After the changes described in 6.1.4 were implemented, there was only positive feedback in phase 2 of the usability tests. We found the users to be very happy with the UI/UX of the system, with one user quoted saying it was a ‘very clever, modern design’. All actions were found to be very intuitive and navigation was found to be effortless. Another user was quoted saying that the application was ‘visually appealing, engaging and I cannot fault it.’

We also performed a final demo for the stakeholder. The stakeholder was extremely happy with the system and said it exceeded his expectations. He was happy with the UI/UX of the system, and it met the original functional requirements he set for us. He has since contacted the Vision board and arranged for us to do a demo for them before deploying it into the hospital for the acceptance test.

6.3 Unit tests

All of the tests for the frontend and backend passed. This provided adequate validation that our functionalities were working as they should be.

On the frontend, we tested pure functions that take in the previous state and an action, and return the next state. The pure functions were easy to test as they produce no side effects and does not rely on external state. It was important to test these functions as this is where the business logic happens and where new application state is formed based on the API or internal responses. [36] The results of these tests ensured that there were no issues related to global application state. The results of the testing can be found in Appendix K.

On the backend, each unit test tests for a particular API endpoint, which will make the call to the database and the response status code will be compared with what is expected and if they match, the test is successful. This test both the endpoint as well as database to see if it's working as expected, and if the correct data is being sent. All core endpoints were tested, and their results can be found in Appendix L.

7 Conclusions

Due to Covid-19, we were unable to deploy the system in a real clinic to conduct an acceptance test. However, we received very positive feedback in phase 2 of the usability tests and the stakeholder stated that the UI/UX exceeded his expectations. From these responses, we were able to conclude that the aim of creating an intuitive and easy to use system that encapsulated all of the requirements, was met. Moreover, as the unit testing shows, all functional requirements are working correctly, with many additional edge cases taken into account. This verifies that the system is reliable and works correctly. The non-functional requirements were also all met. The other team members ensured adequate measures were put in place to ensure that the application is secure and various performance enhancing techniques were used to ensure that the application performance is fast. That being said, all the original aims set out were met.

It is expected that this system will help Vision manage their clinics and associated general practices more effectively and efficiently. Their processes will become seamless, removing the effort of manually recording and managing operations. It will enhance the staff’s ability to coordinate care, streamline the search of patient files, and increase data security. This system could potentially enable Vision to accommodate more patients in their once a month clinic, as well as allow general practices to see more patients from Vision’s affiliated beneficiaries. People in the beneficiaries who are unable to, or find it difficult, to receive healthcare will now have access to well run, organised clinics and general practices.

Ultimately, this will allow them to make a bigger impact on local communities.

In terms of future extensions to this project, a new dashboard could be integrated to give daily, weekly or monthly reports on the average waiting times between different phases of the hospital. The dashboard could also show hospital visit trends to allow the directors to allocate more staff members to clinics at different times of the day/year depending on the identified trends.

When reflecting upon the entire project, I believe that our agile development process, our effective communication through daily stand-up meetings and willingness to help one another is the reason why we were able to exceed in producing a fully functional, visually appealing system. Based on my responsibilities in this project, as a recommendation to others trying to replicate this system, I would say that talking to a UI/UX developer during the prototyping phase is invaluable and saves a lot of time in the development as few changes to the UI/UX will have to be made. If they use React to develop the web application, Axios should be to send HTTP requests as it provides automatic conversion to JSON, making information capturing much easier. I would also recommend using CSS Modules for styling and for making the application responsive for both mobile and tablet devices. CSS is easy to work with and allows for efficiency in design and updates due to its media query functionalities. [12]

8 Acknowledgements

I want to thank my team members for their unbelievable commitment to the project and their consistent willingness to assist each other. I would also like to thank our supervisor, Aslam Safla, for his guidance and help throughout the project. I would like to thank Melissa Densmore for all her feedback and time taken to help us. Finally I would like to thank Dr Moosa Parak for providing us with the opportunity to develop this system and do our part in helping them care for the vulnerable communities.

9 References

[1] David Avison and Terry Young. 2007. Time to rethink healthcare and ICT?. Communications of the ACM 50, 6, 69-74. DOI: https://www.researchgate.net/publication/220421651_Time_to_Rethink_Healthcare_and_ICT

[2] Alexander Mathioudaki, Ilona Rousalova, Ane Gagnat, Neil Saad and Georgia Hardavella. 2016. How to keep good clinical records. Breathe (Sheff) 12, 4, 369-373. DOI: https://www.ncbi.nlm.nih.gov/pubmed/28210323

[3] React. 2020. Virtual DOM and Internals. Retrieved September 21, 2020 from https://reactjs.org/docs/faq-internals.html

[4] N. Kolenda. 2016. The Psychology of User Experience. Retrieved April 29, 2020 from https://www.nickkolenda.com/pdf/usability-tactics.pdf

[5] B. Shneiderman. 1987. Designing the User Interface: Strategies for Effective Human-Computer Interaction

[6] B. Armour. 2018. 5 Key Benefits of Native Mobile App Development. (August 2018). Retrieved April 25, 2020 from https://clearbridgemobile.com/benefits- of-native-mobile-app-development/

[7] BezKoder. 2020. React+Node.js+Express + MySQL example: Build a CRUD app. (Septmeber 2020) Retrieved September 7, 2020.

[8] Luke Wroblewski. 2009. Inline Validation in Web Forms. (September 2009). Retrieved Spetember 8, 2020 from https://alistapart.com/article/inline-validation-in-web-forms/

[9] Interaction Design Foundation. 2019. Gestalt Principles. Retrieved September 9, 2020 from https://www.interaction-design.org/literature/topics/gestalt-principles

[10] UXPin. 2020. Design Consistency Guide: Best Practices for UI and UX Designers. (April 2020). Retrieved April 29, 2020 from https://www.uxpin.com/studio/blog/guide-design-consistency-best-practices-ui- ux-designers/

[11] Janelle Wong. 2017. Why Use Axios in Your Next App. Retrieved April 30, 2020 from https://medium.com/@janelle.wg/why-use-axios-in-your-next-app-c44ad3508e93#:~:text=By%20using%20axios%2C%20it%20simplifies,automatic%20transformations%20for%20JSON%20data.

[12] Purely Branded. 2017. Why Use CSS in Website Design. Retrieved September 18, 2020 from https://www.purelybranded.com/notes/why-use-css-in-website-design/

[13] Adroit Infosystems. eHospital Systems. Retrieved from https://www.adroitinfosystems.com/products/ehospital-systems

[14] Sapphire. Sapphire Hospital Management System. Retrieved from https://www.sapphirehms.com/

[15] Pinaacle, "MedStar Hospital Management and Information System," Available: http://medstarhis.com/docs/Medstar-Brochure.pdf. [Accessed 6 May 2020].

[16] AthenaHealth, "Home Page," 2020. Available: https://www.athenahealth.com/. [Accessed 6 May 2020].

[17] TeamDesk. 2020. Medical Practice Manager database. Retrieved May 7, 2020 from https://www.teamdesk.net/

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

[19] Ramya Shankar. 2020. JSON vs XML. (April 2020). Retrieved May 2, 2020 from https://hackr.io/blog/json-vs-xml

[20] Munir Šabanović, Muzafer H. Saracevic and Emrus Azizovic. 2016. Comparative analysis of AMF, JSON and XML technologies for data transfer between the server and the client. Periodicals of engineering and natural sciences, 1-7. Retrieved from https://www.semanticscholar.org/paper/Comparative- analysis-of-AMF%2C-JSON-and-XML-for-data-Saracevic-%C5%A0abanovi%C4%87/ab9ff25deafecbd78132136047259373a2f4a3a1

[21] RestfulAPI. 2019. JSON vs XML. Retrieved May 4, 2020 from https://restfulapi.net/json-vs-xml/

[22] IBM. 2020. Introduction: Application servers. Retrieved April 24, 2020 from https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.base.doc/ae/welc_servers.html

[23] NGINX. n.d. NGINX: What Is an Application Server vs. a Web Server? Retrieved April 25, 2020 from https://www.nginx.com/resources/glossary/application-server-vs-web-server/

[24] Roy Fielding. 2000. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation. University of California, Irvine.

[25] Andy Pardue, Derek Veit, Pete Johanson, Rob Kilian. 2020. OpenVista®. Retrieved July 11 from https://sourceforge.net/projects/openvista/

[26] Web Designer Hub. 2017. Benefits and disadvantages of using the hamburger menu in responsive websites. (January 2017). Retrieved April 29, 2020 from https://www.webdesignerhub.com/hamburger-menu/

[27] Tubik Studio. 2017. Case Study: Health Care App. UI for Doctors. (March 2017). Retrieved 30 April 2020 from https://uxplanet.org/case-study-health- care-app-ui-for-doctors-826741027950

[28] Johanna Silvennoinen, Marlene Vogel, Sari Kujala. 2014. Experiencing Visual Usability and Aesthetics in Two Mobile Application Contexts. Journal of usability studies 10, 1, 57-58. DOI: https://www.researchgate.net/publication/268978666_Experiencing_Visual_Usability_and_Aesthetics_in_Two_Mobile_Application_Contexts

[29] SteelKiwi inc. 2017. How to design a great medical app. (December 2017). Retrieved 30 April 2020 from https://medium.muz.li/how-to-design-a-great- medical-app-c3079f1390e7

[30] Nemanja Banjanin. 2019. UI Design Best Practices for Better Scannability. (February 2019). Retrieved April 29, 2020 from https://www.toptal.com/designers/web/ui-design-best-practices

[31] Ho Tran Tony. 2019. Inside Design. (March 2019). Retrieved April 29, 2020 from https://www.invisionapp.com/inside-design/pros-and-cons-of-hamburger-menus/

[32] Gigi Sayfan. 2018. 8 Things that make Jest the best React Testing Framework. Retrieved September 22, 2020 from https://code.tutsplus.com/tutorials/8-things-that-make-jest-the-best-react-testing-framework--cms-30534

[33] Tubik Studio. 2017. Case Study: Health Care App. UI for Doctors. (March 2017). Retrieved 30 April 2020 from https://uxplanet.org/case-study-health- care-app-ui-for-doctors-826741027950

[34] Federico Botella, Juan P. Moreno, Antonio Peñalver. 2015. Comparing the efficiency of performing complex tasks with a tablet and a smartphone. DOI: http://dx.doi.org/10.15446/dyna.v82n193.53492

[35] Adobe. 2020. Adobe XD. Retrieved Septmeber 30, 2020 from https://www.adobe.com/africa/products/xd.html

[36] Alex Bachuk. 2017. Testing Redux Reducers with Jest. Retrieved September 30, 2020 from https://medium.com/@netxm/testing-redux-reducers-with-jest-6653abbfe3e1

Appendix A

Functional Requirements:

Based on the requirements analysis meeting, the following functional requirements were outlined:

A system is required for Vision Medical Suite, an NPO that provides free medical and dental care. The system will be used to manage the staff, patients and overall processes and operations involved. The system will be made up of a web application which incorporates the following requirements:

- Users should have the option to login or register (if permission received)
- A pool of all patients that are registered in the system will be displayed
- The user should have the option to select the sedation clinic or a general practice

  [Sedation clinic is selected]:

- Users should be able to register patients
- Users should be able to book patients for the clinics
- Users should be able to cancel bookings and push bookings forward
- A scheduling tool to be implemented within the booking system
- Upon arrival of the patient, the user (receptionist) should have the option to select patient from booking list, and thereafter remove the patient from such list
- Mandatory requirement for a user to timestamp event, capture arrival information, vital signs and screening information
- The user should then move the patient to the triage waiting list
- A list of patients who are waiting for triage should be displayed, along with their information, reason for visit and waiting time
- The user should then be able to capture all necessary screening information for each patient on the list
- After all required screening information is captured, the user should then have the option to move the patient to the next stage. The timestamp will be automatically captured.
- The patient should then be moved to the theatre waiting list
- A list of patients who are waiting for theatre should be displayed, along with their information and waiting time
- The user should be able to remove the patient from the waiting list, when ready for theatre. Timestamp will be captured.
- The user should be able to edit the patient and fill in the necessary information regarding the procedure
- This section should contain four capture points to tag the clinicians
- Clinicians should be able to capture notes about the procedure. The user should have the ability to either convert voice notes into text using an API or type out the notes onto a device. Either way, all pieces of information will be stored in the database and be available to the user on request.
- User should be able to capture ICD10 codes associated with the procedure
- Option to search through the database containing all ICD10 codes, using an existing API
- User will be able to select ICD10 codes which get added to the list in the patient record
- The user should then be able to move the patient to the disposition waiting list. Timestamp captured
- A list of patients who are waiting for disposition should be displayed, along with their information and waiting time
- The user should be able to fill in disposal information, tag his/her name, input the time the patient will spend in the recovery area and complete full vital signs of the patient
- The user should then be able to move the patient to the exit waiting list
- A list of patients who are waiting for exit should be displayed, along with their information and waiting time
- The user should be able to fill in where the patient will go after

- Allow for adding and changing facilities
- The user should be able to remove the patient from the exit waiting list. Timestamp captured

  [End of sedation clinic process]

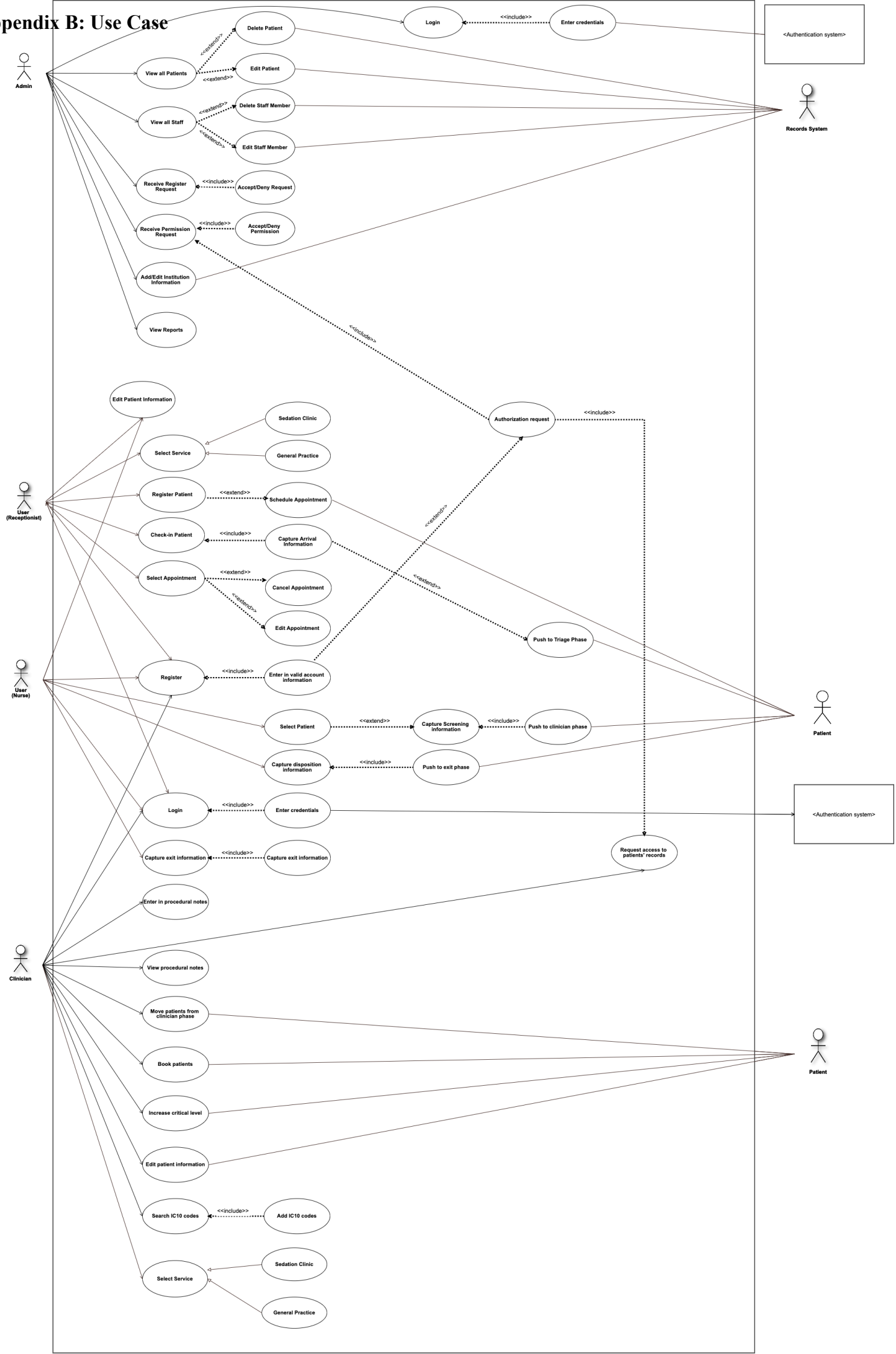
- General practice system should include entry and vital signs
- A user management system should be implemented to control access and permissions
- Staff should be given permission from a higher level user before getting access to the system and viewing information in another sector
- Prompts should be displayed to update previous information of patients
- Admin should have access to everything and have the ability to grant permissions
- Screenshots should be blocked
- Saved credentials on browser should be blocked
- No two users should be able to open a specific patient's record simultaneously. A warning should be given if the record is already opened
- Reports should be generated on various activities
- These reports should be visualised

Non-functional requirements:

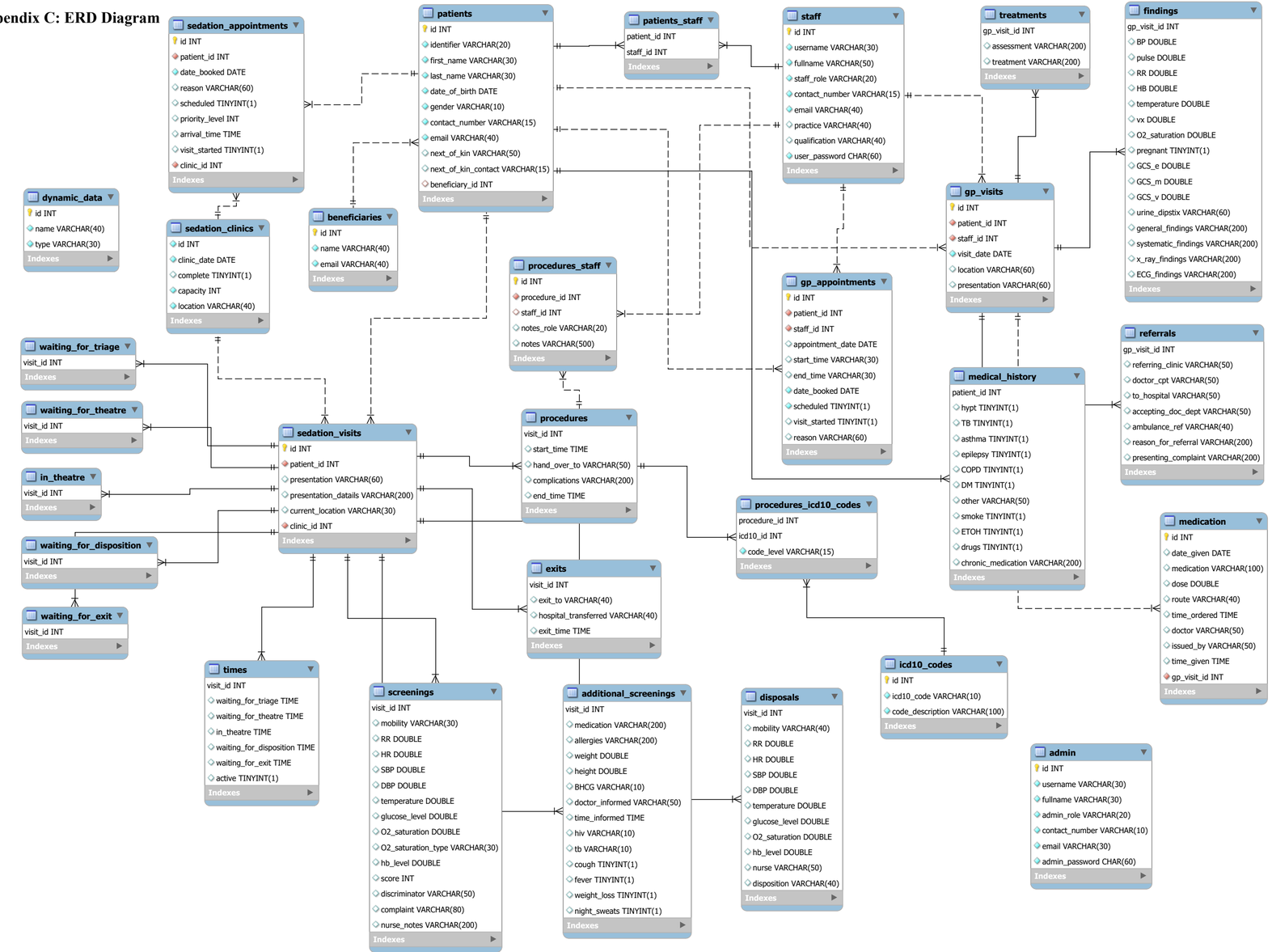
- Ensuring privacy and confidentiality with respect to clinical notes written by staff members.
- Privacy and confidentiality of patient's records is essential. The records can only be accessed by authorised staff members.
- Security of the system will be enforced through the use of login details.
- When a user signs into the system, a unique token id will be created for that session. This token will be used to access certain information within the database. This token id will expire after a set expiration time. If the user is still using the application, the token id will be refreshed. [token authentication]
- The user's user id will be used to help access information only authorised for that user. This user id will be checked with every request that is made by the user.
- The user will be logged out of the system after a certain amount of time has elapsed - this falls under the expiration time that is set for the token. Unless they are still using the application.
- Fast retrieval of information from the database at all times is required.
- Ability to either convert voice notes into text using an API or typing out the notes onto a device or taking a picture of the device of the notes written out by the user. Either way, all pieces of information will be stored in the database and be available to the user on request.
- UI/UX of the application needs to be of a high standard to ensure that the app is user friendly and intuitive. The application needs to be easy to use in high-stress situations. Information and actions need to be displayed clearly and accurately.
- HTTPS connection is essential for an encrypted secure connection. This will help prevent malicious attacks on information that may be travelling to and from the website and the user's browser.
- Users are error-prone and thus modal/pop-ups are required to confirm the intent of the actions that the user might make. For example, deleting a patient or moving the patient from one area to another. This will help mitigate unwanted side effects.
- Providing the user with the tools to edit actions.

- The system needs to be both accurate and reliable - in terms of both the information that is provided as well as the functionality. The user should be able to retrieve and post information to the database.
- The application needs to be compatible with all devices. These include both desktop and mobile devices.

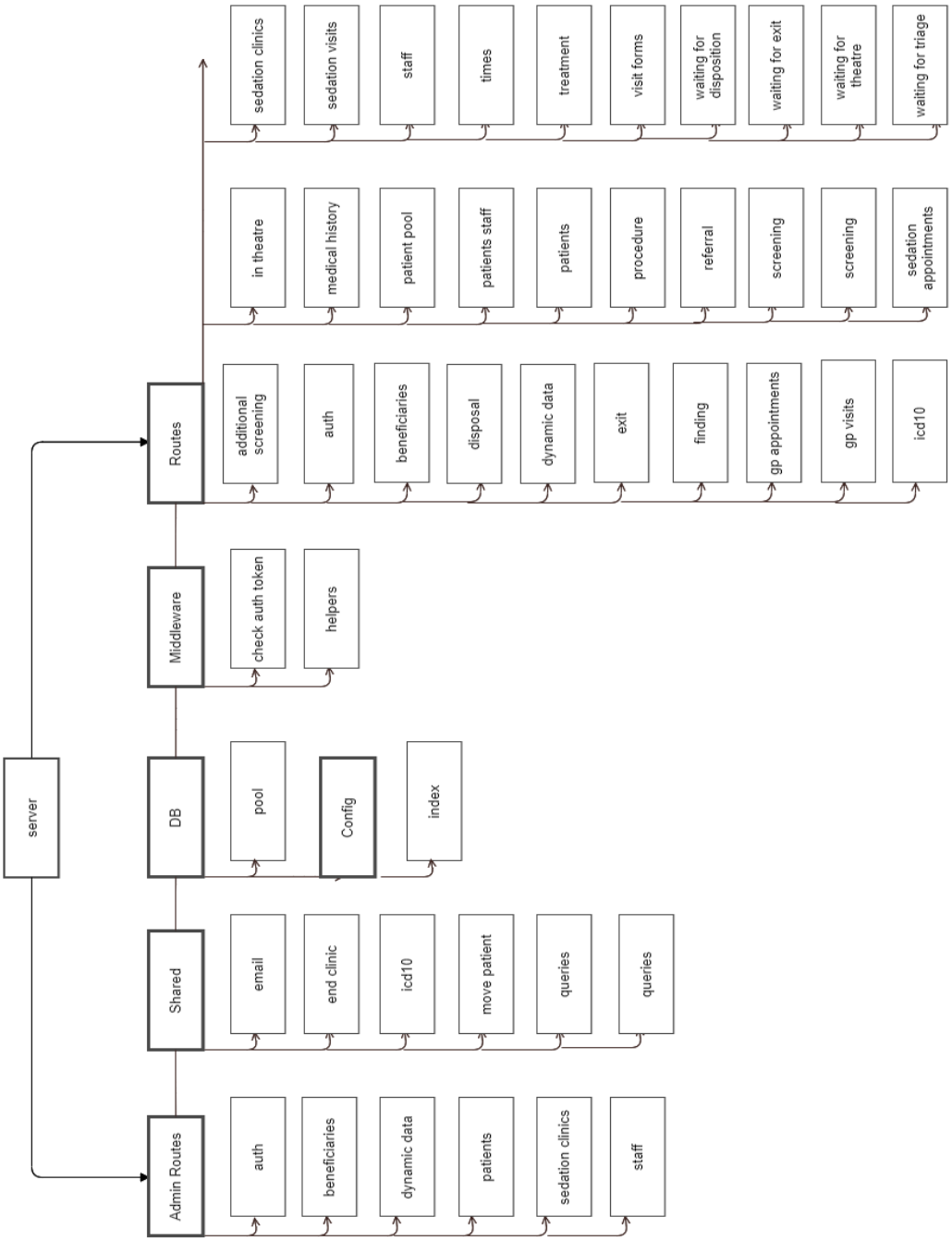
Appendix B: Use Case



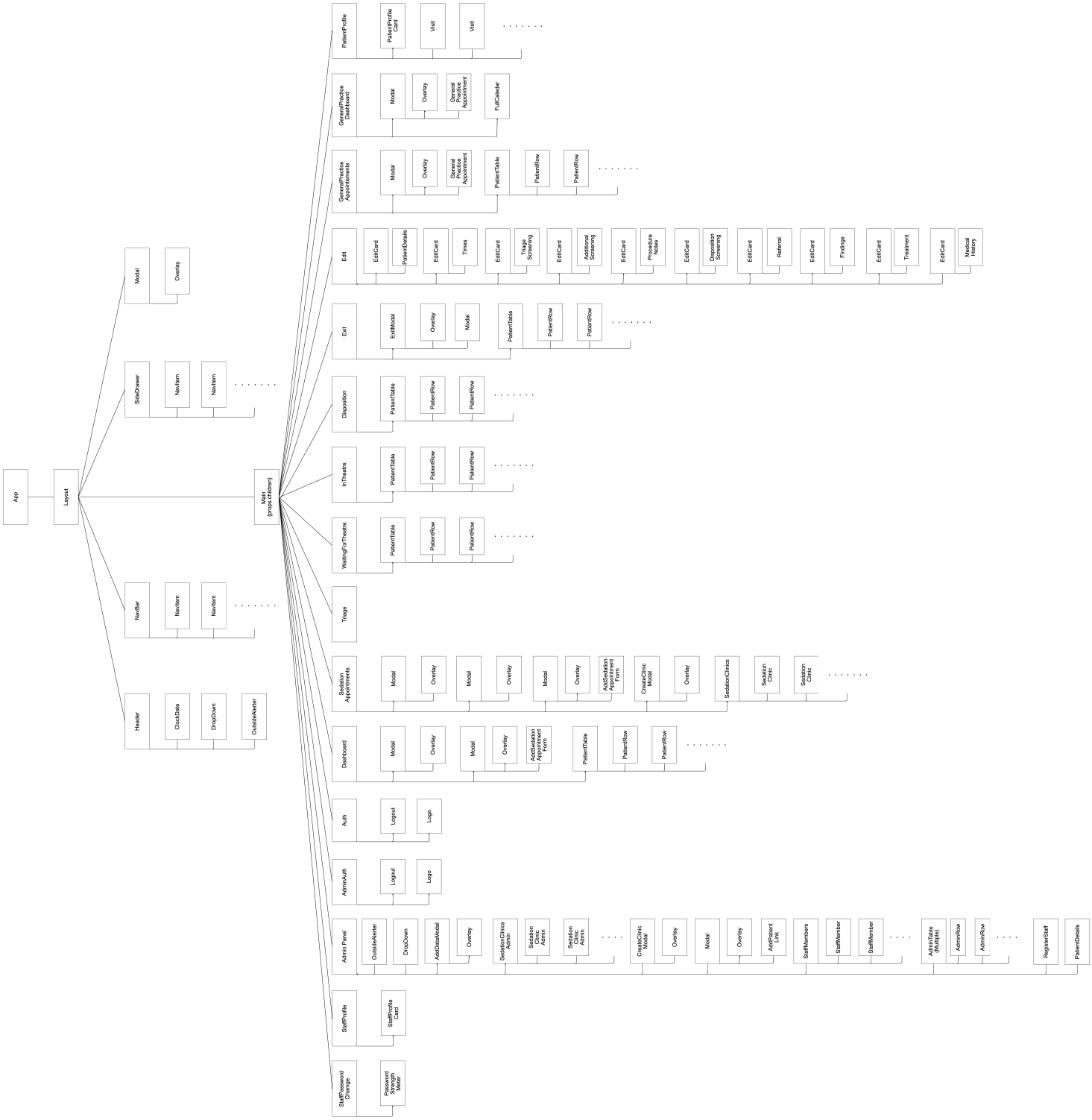
Appendix C: ERD Diagram



Appendix D: Backend component structure



Appendix E: Frontend Component Structure



Appendix F: First Prototype

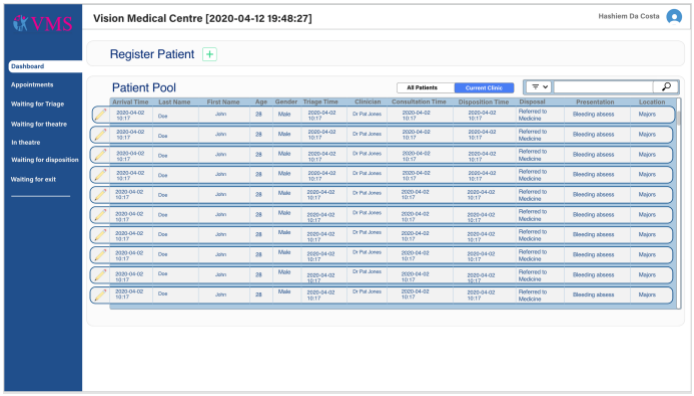


Figure 1: Dashboard

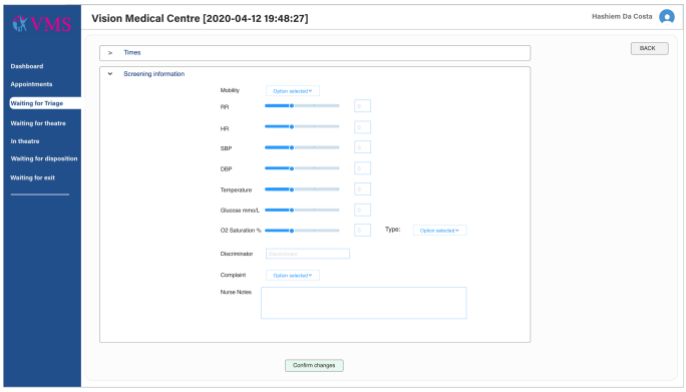


Figure 2: Screening Form

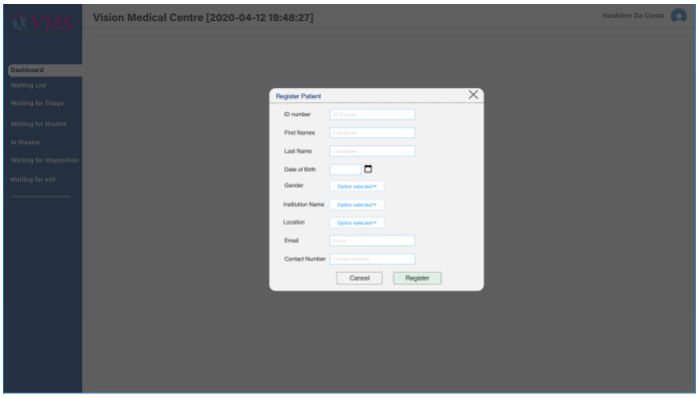


Figure 3: Register patient modal

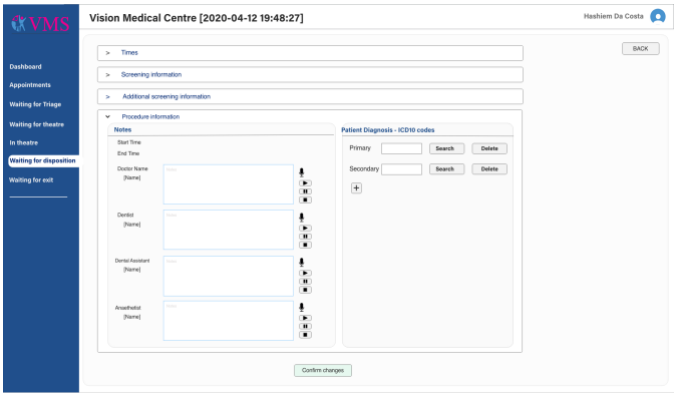


Figure 4: Procedure form (with voice to text)

Appendix G: Final Prototype

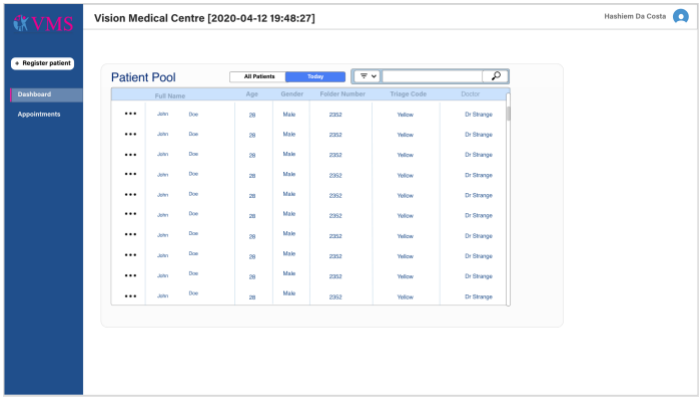


Figure 1: Patients table

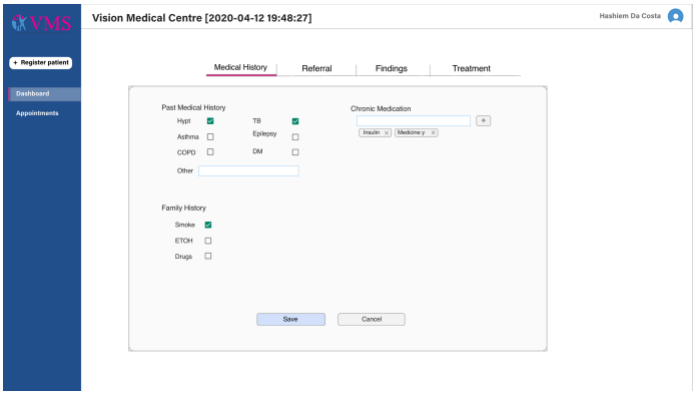


Figure 2: Form layout

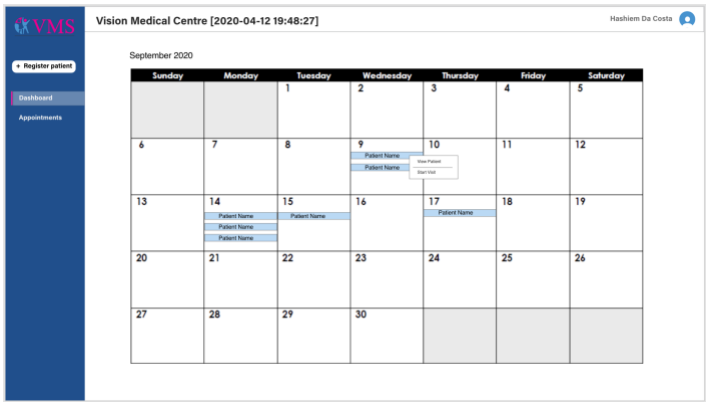


Figure 3: GP appointments

Appendix H: Mobile prototype

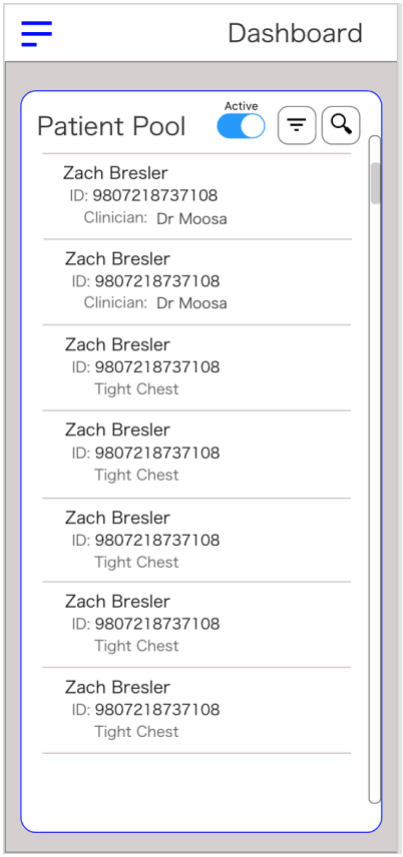


Figure 1: List of patients all patients

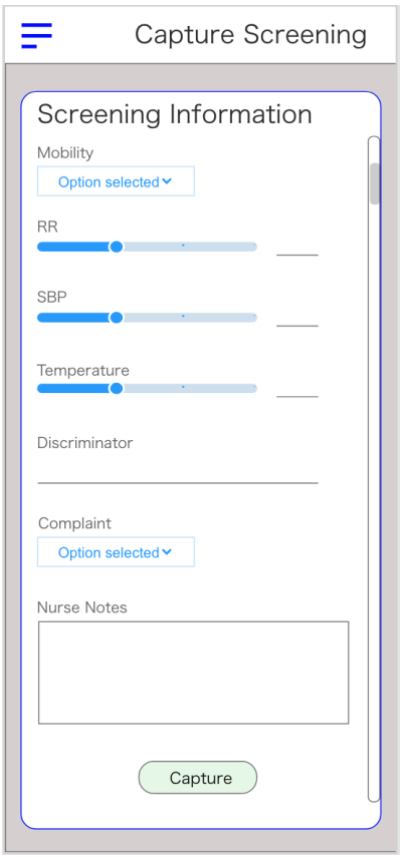


Figure 2: Screening information form

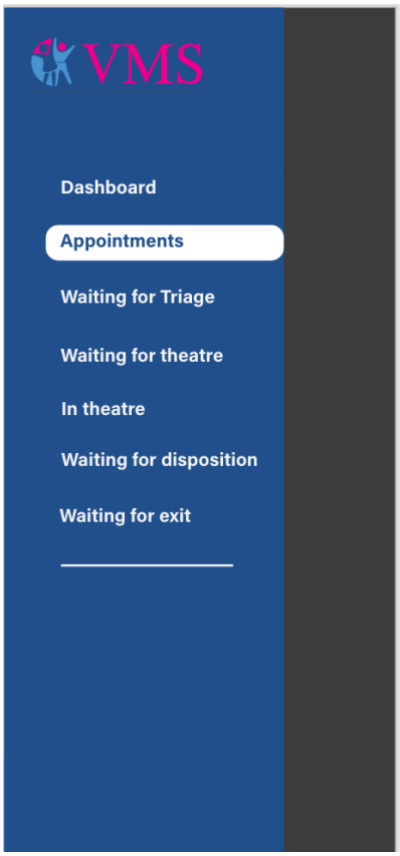


Figure 3: Side drawer

Appendix I:

Sedation Clinic:

Figure 1: Application login screen

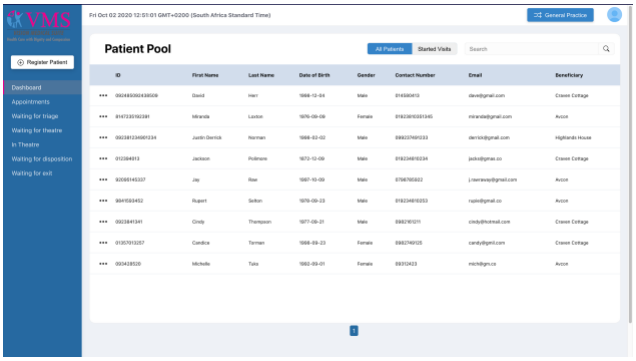


Figure 2: Dashboard

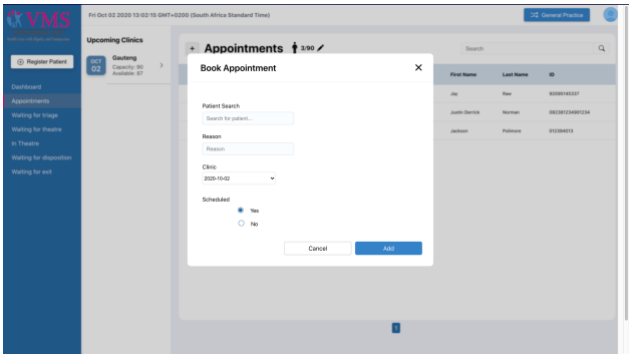


Figure 3: Creating clinic appointment

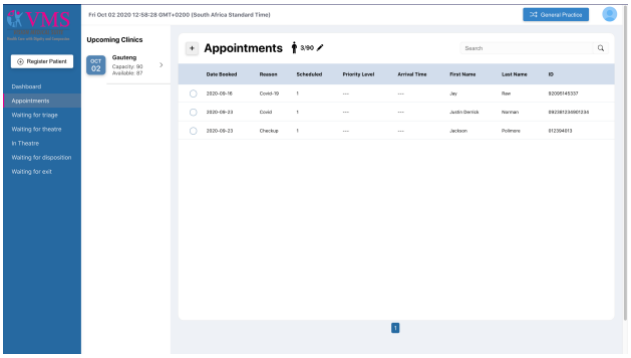


Figure 4: Sedation clinic appointments

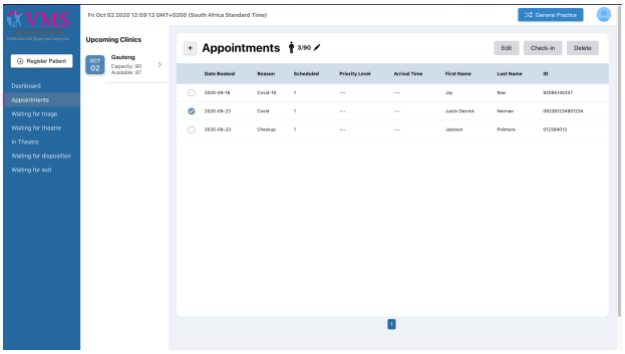


Figure 5: Editing, Checking-in and delete appointment function

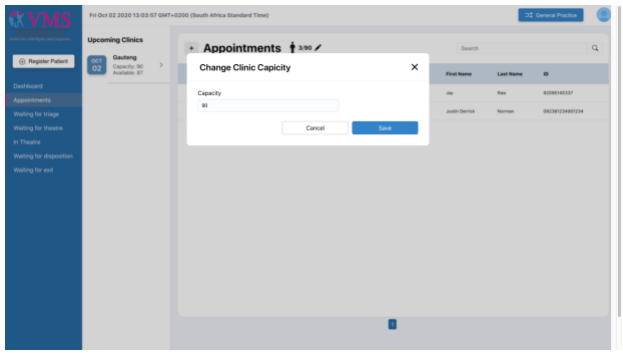


Figure 6: Edit sedation clinic capacity

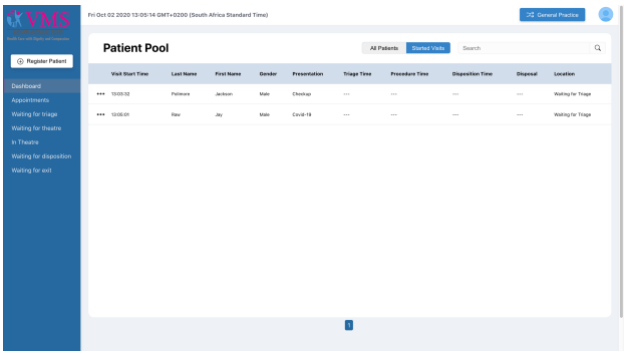


Figure 7: Sedation clinic ongoing visits

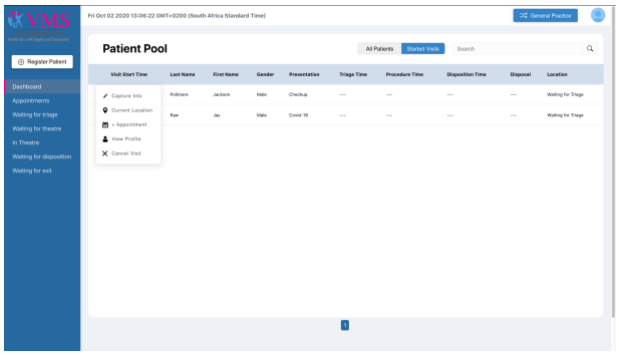


Figure 8: Patient actions

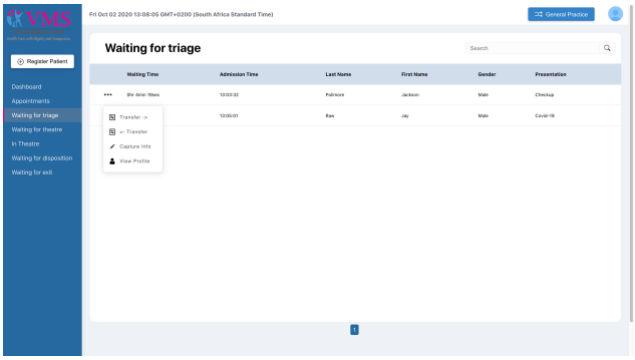


Figure 9: Waiting for triage table with patient actions

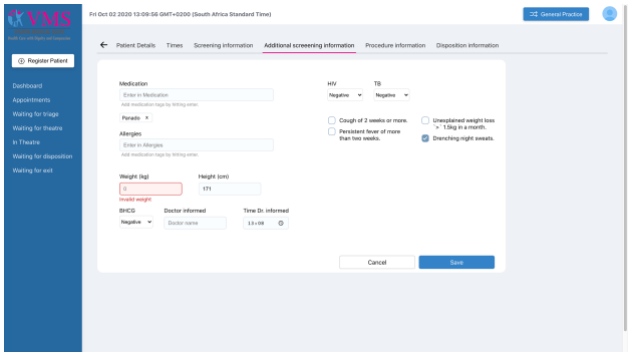


Figure 10: Additional screening form (with validation)

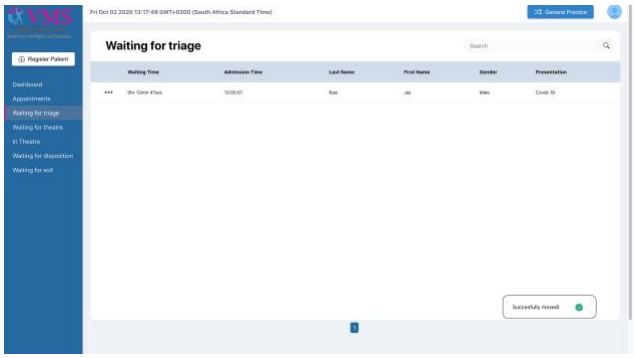


Figure 11: Moving patient between phases feedback modal

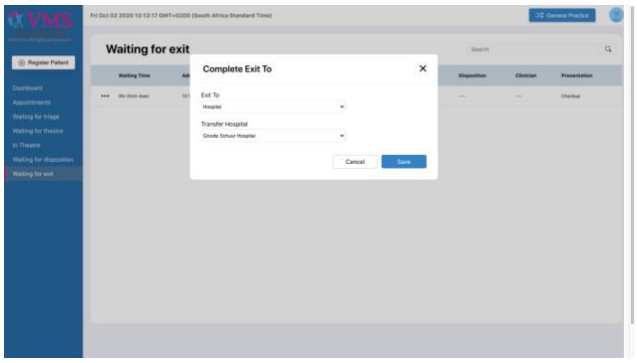


Figure 12: Completing sedation clinic visit

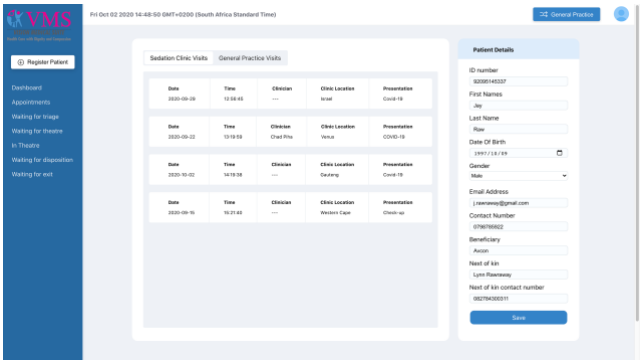


Figure 13: Patient Profile

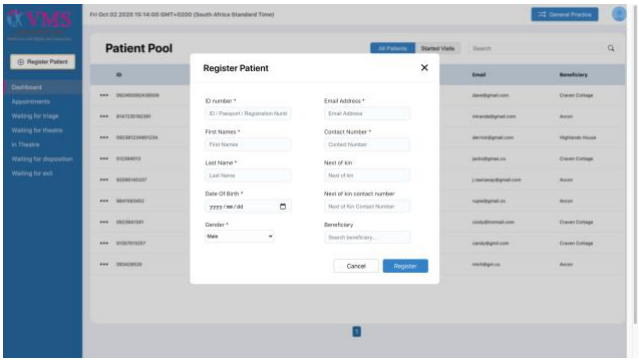


Figure 14: Register Patient

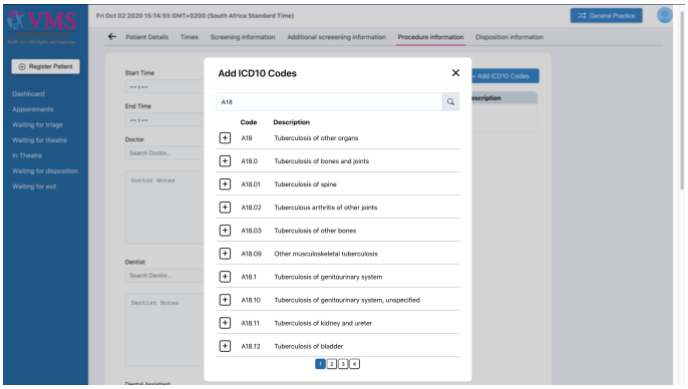


Figure 15: Add ICD10 code modal

General Practice (GP):

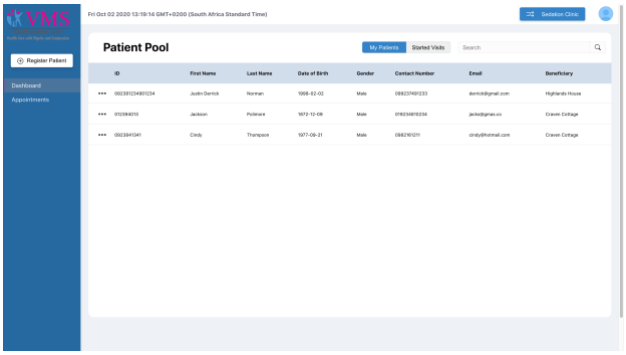


Figure 16: GP Dashboard

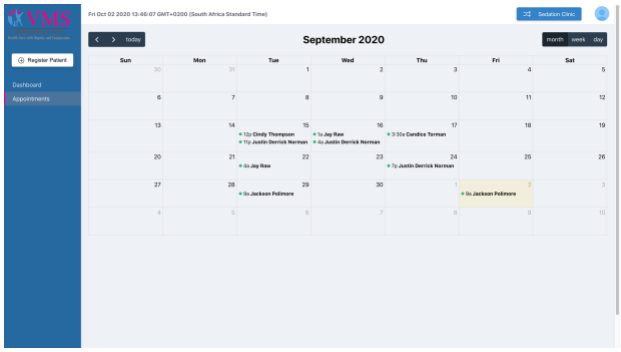


Figure 17: GP appointments

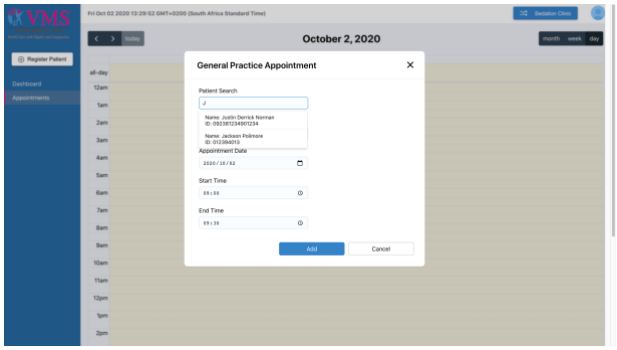


Figure 18: Create GP appointment (with patient list dropdown)

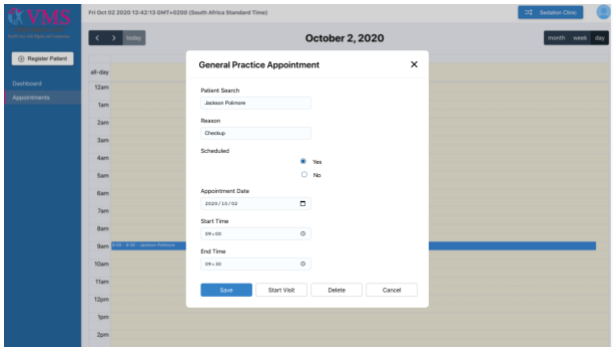


Figure 19: Appointment actions

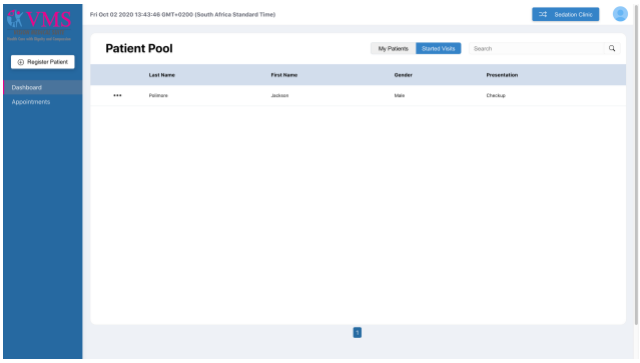


Figure 20: Patients undergoing their appointment

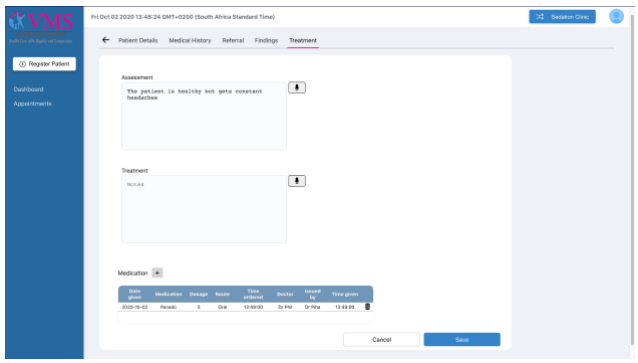


Figure 21: GP Treatment Information Capture (w/ Voice to text)

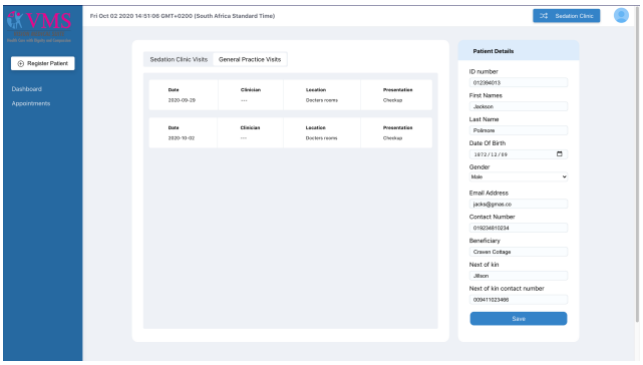


Figure 22: GP patient profile

Admin Panel

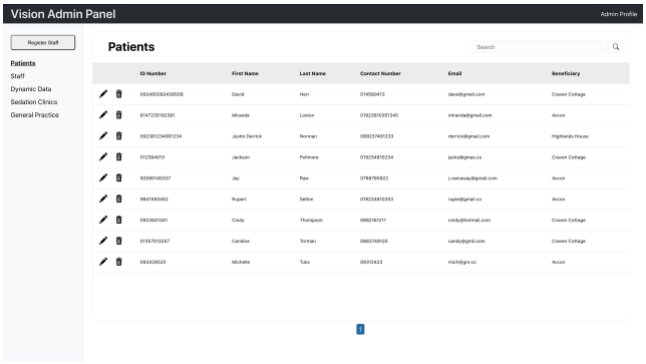


Figure 23: List of all patients

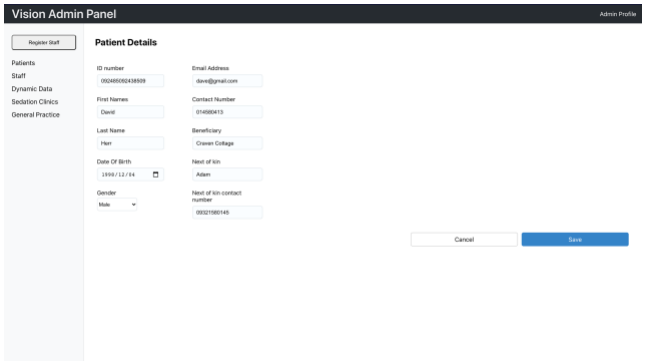


Figure 24: Editing patient details

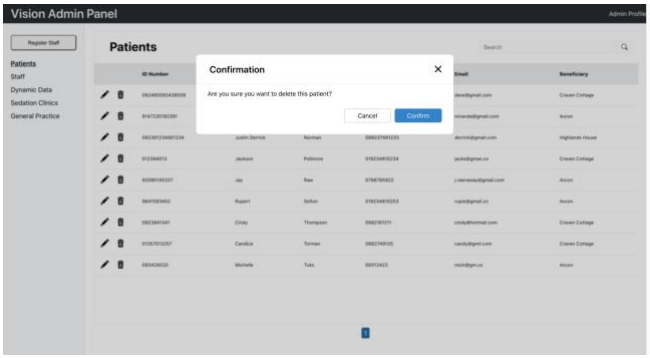


Figure 25: Confirmation modal

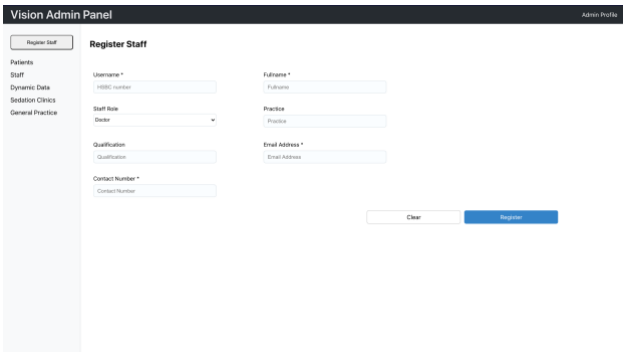


Figure 26: Register staff

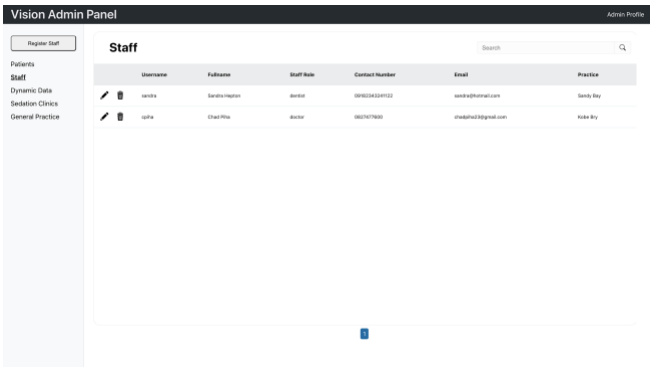


Figure 27: List of all staff members

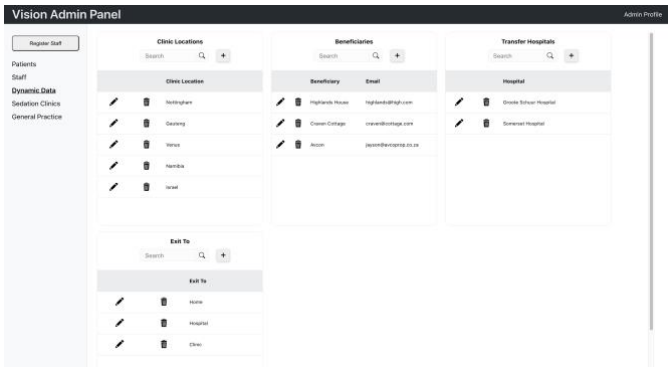


Figure 28: Dynamic data

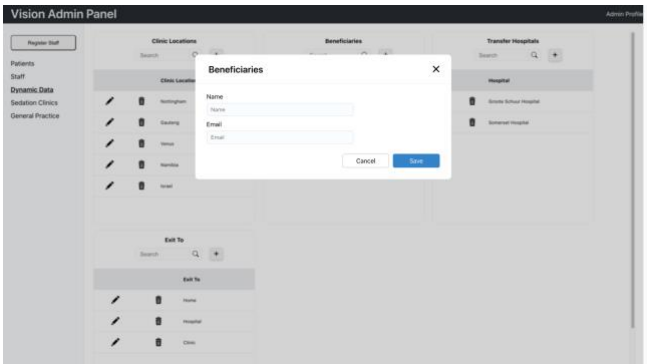


Figure 29: Adding beneficiaries dynamic data

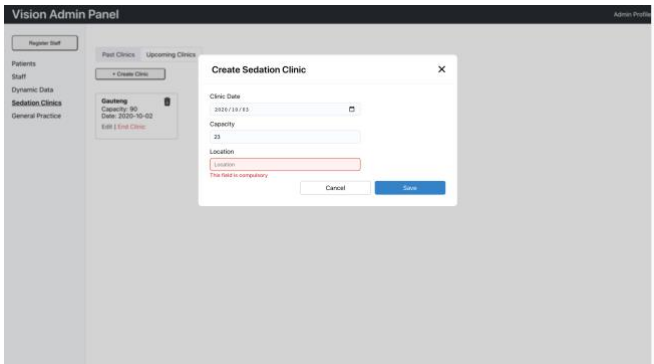


Figure 30: Create sedation clinic (with validation)

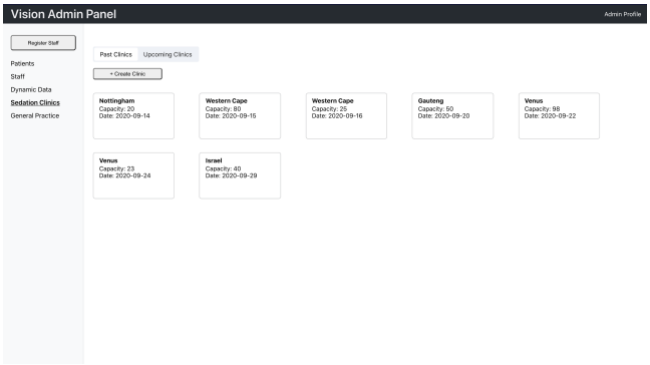


Figure 31: Past sedation clinics

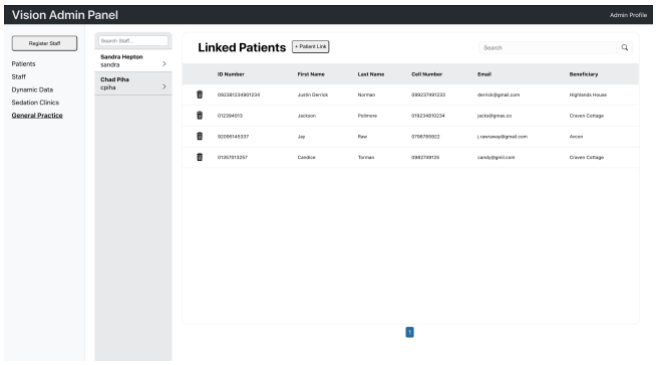


Figure 32: Patient-staff link

Mobile application sedation clinic

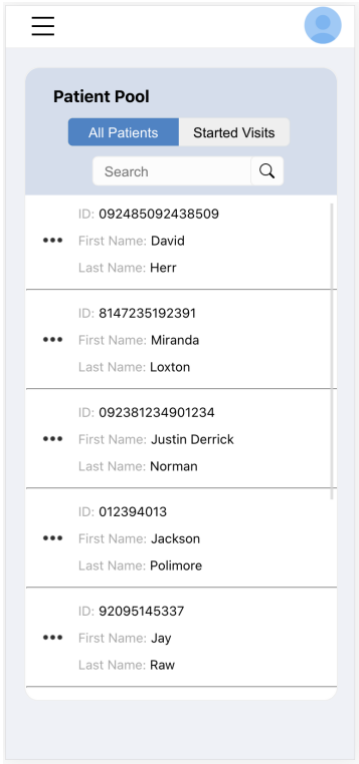


Figure 33: List of all patients table

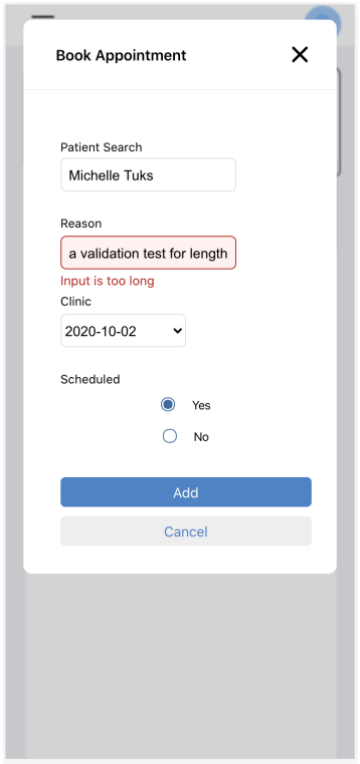


Figure 34: Book appointment (w/ validation)

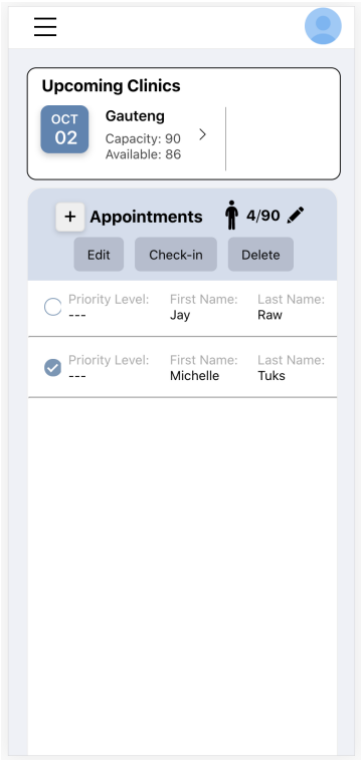


Figure 35: Appointments actions

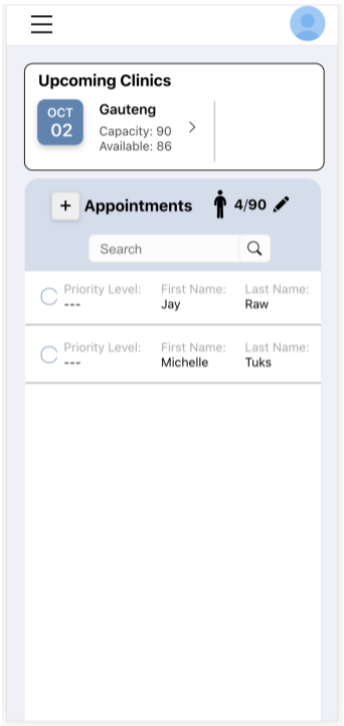


Figure 36: Appointments

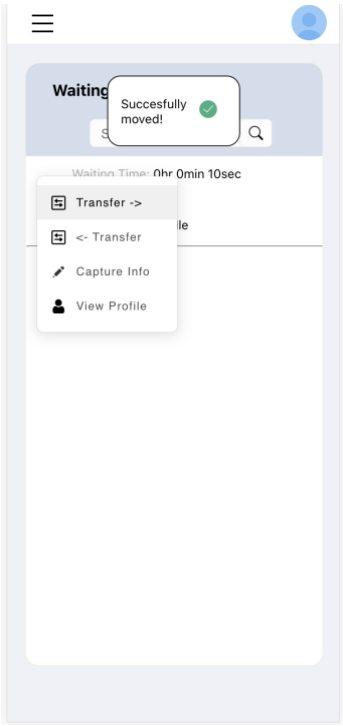


Figure 37: Feedback modal

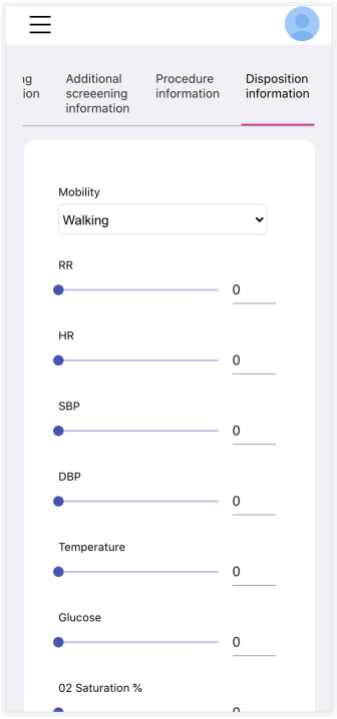


Figure 38: Mobile form

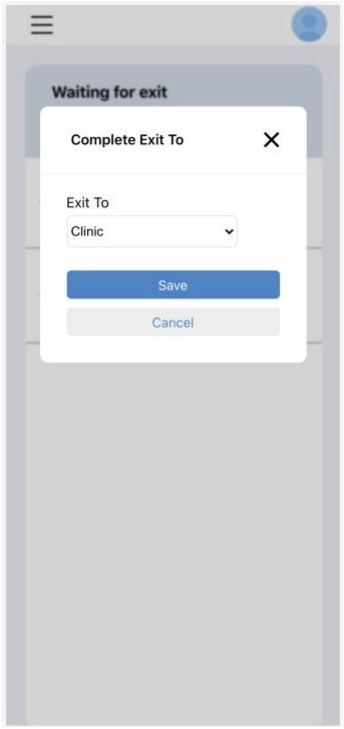


Figure 39: Exit to modal

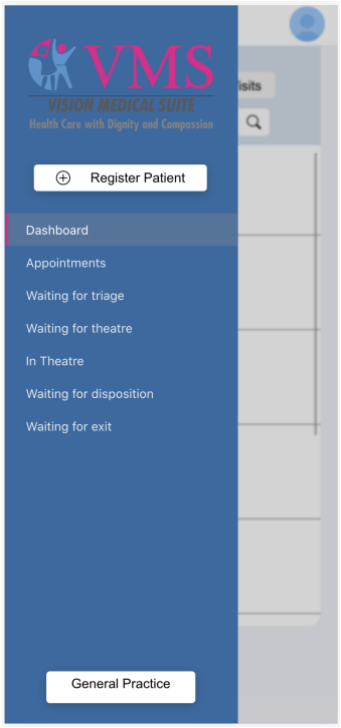


Figure 40: Expanded hamburger menu

Mobile General Practice:

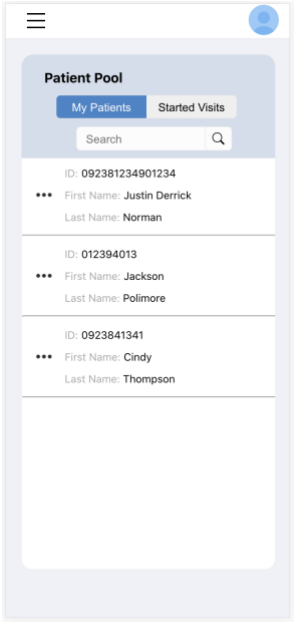


Figure 41: Linked patients table

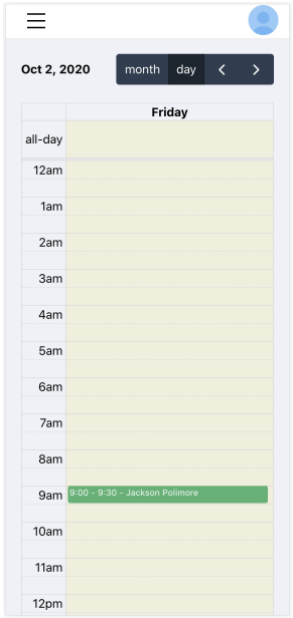


Figure 42: Appointments

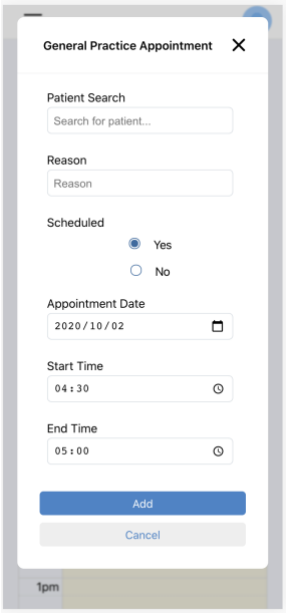


Figure 43: Book appointment.

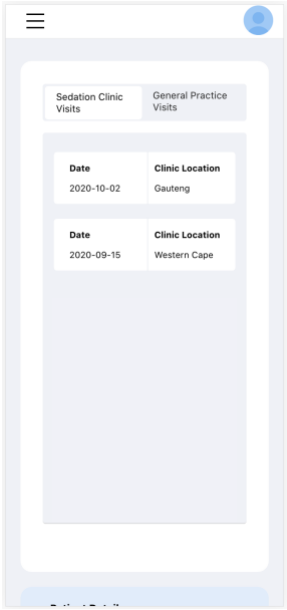


Figure 44: Patient profile

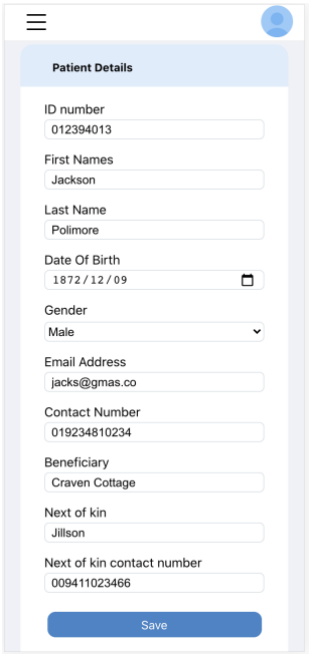


Figure 45: Patient profile (cont.)

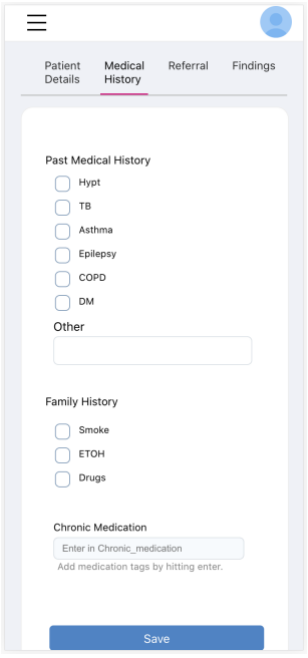


Figure 46: GP form

Appendix J:

Usability tasks

1. Log in to admin panel. Username: admin, Password: admin
2. Register staff member and change their password
3. Add dynamic data for locations, beneficiaries and hospitals
4. Create a sedation clinic for today
5. Link patients to yourself
6. Navigate to your email, and change your password
7. Register a patient
8. Create an appointment with recently registered patient
9. If you wanted to increase the capacity of the clinic, what would you do?
10. This new patient walks in for his appointment. Please check him in.
11. 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 steps 13 and 14 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 information for that visit
23. Logout

Appendix K: 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.
```

Figure 1: Summary of all tests

```
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.
```

Figure 2: Staff unit tests

```
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.
```

Figure 3: Token authentication tests

```
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.
```

Figure 4: Beneficiary unit tests

```
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.
```

Figure 5: Dynamic data unit tests

```
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.
```

Figure 6: General practice unit tests

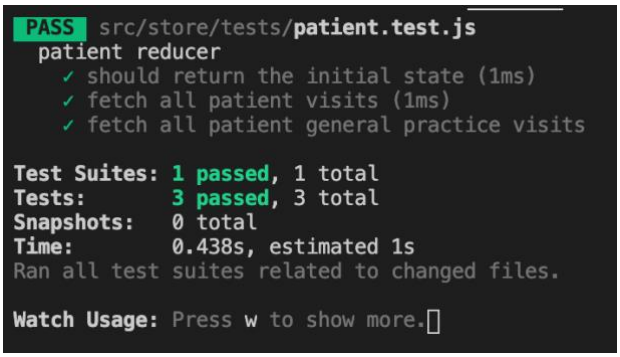


Figure 7: Patient unit tests

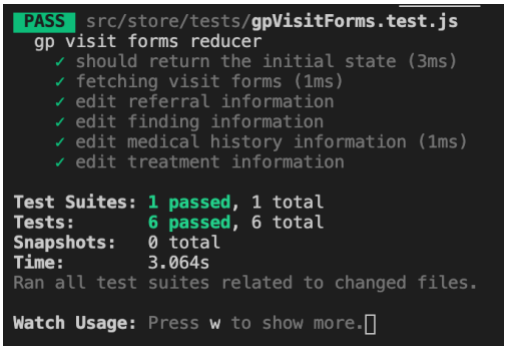


Figure 8: General practice visit forms unit tests

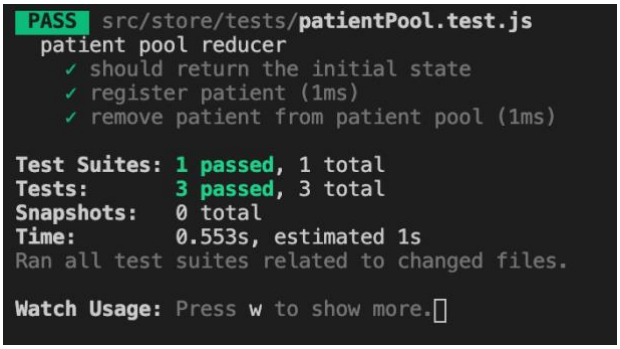


Figure 9: Patient pool unit tests

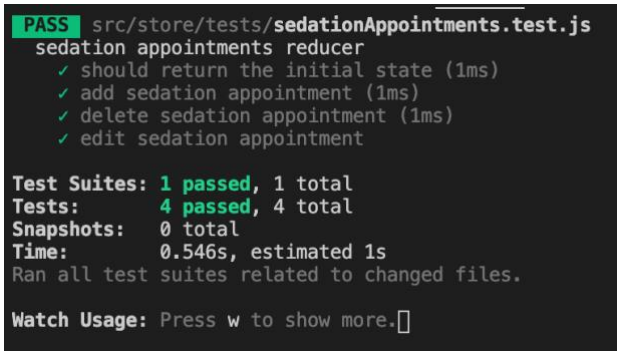


Figure 10: Sedation clinic appointments unit tests

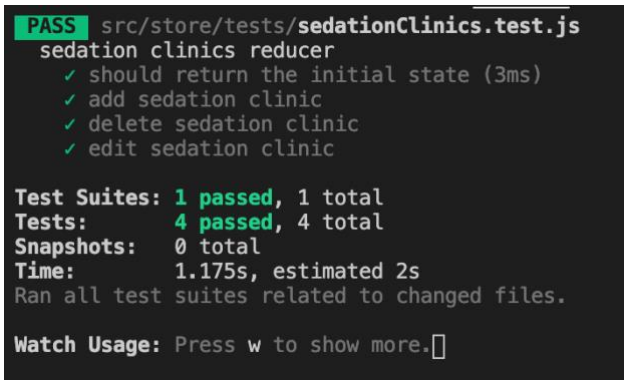


Figure 11: Sedation clinic unit tests

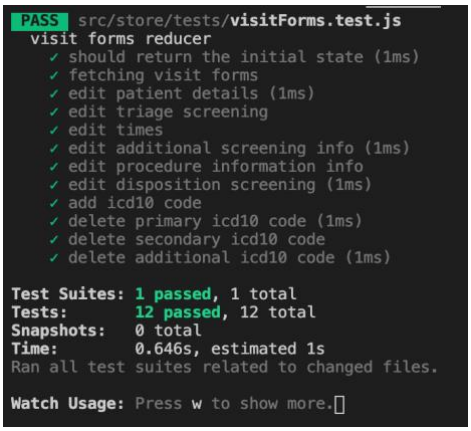


Figure 12: Sedation clinic visit forms unit tests

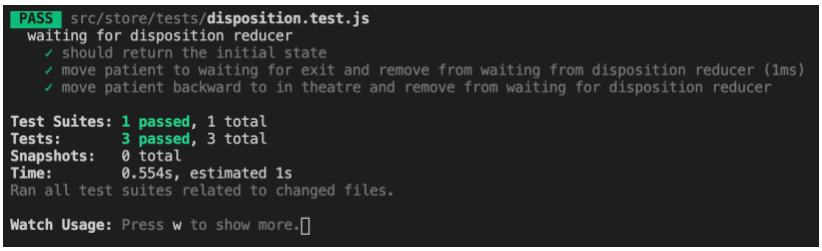


Figure 13: Disposition unit tests

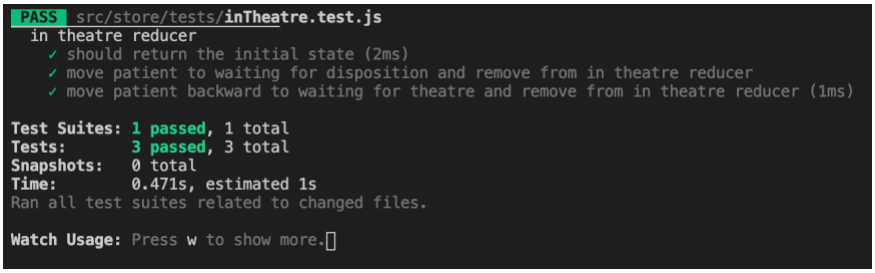


Figure 14: inTheatre unit tests

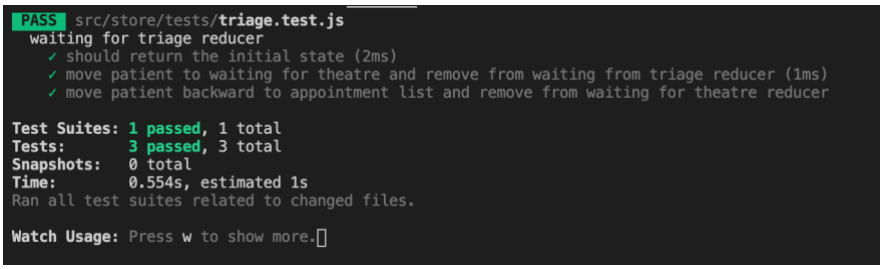


Figure 15: Triage unit tests

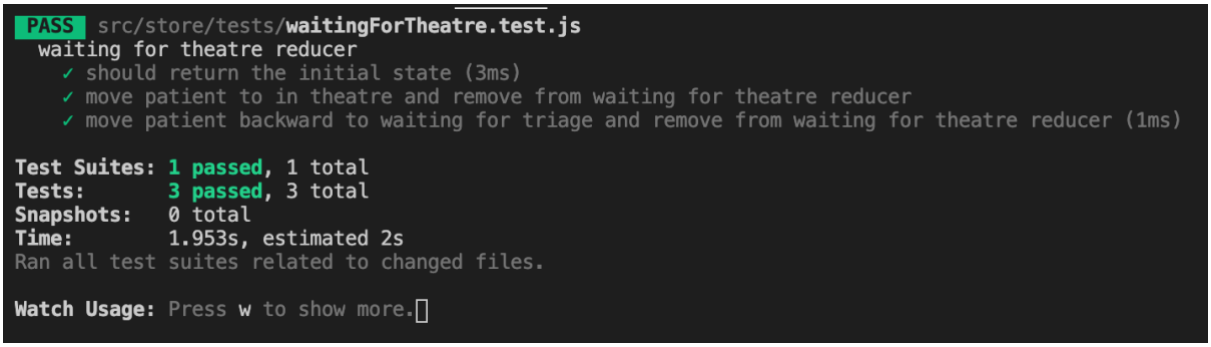


Figure 16: Waiting for theatre unit tests

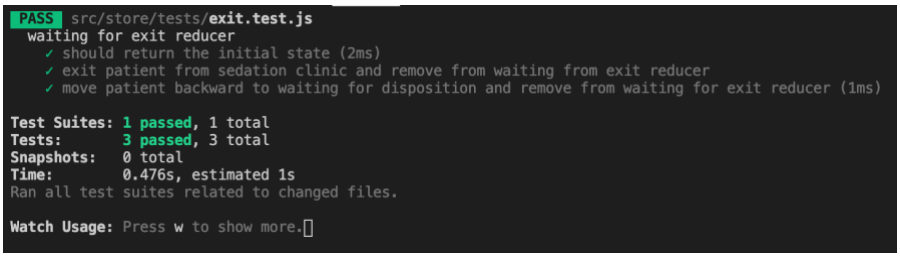


Figure 17: Exit unit tests

Appendix L: Backend tests

```
> vision-server@1.0.0 test C:\Users\Justin Dorman\Desktop\vision-server
> jest

 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/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/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/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.
```