FACULTY OF SCIENCE & TECHNOLOGY

CST3990

Undergraduate Individual Project

SCHOOL OF DIGITAL TECHNOLOGIES

FINAL SUBMISSION

PROJECT TITLE: Hospital Management System for public health sector in Mauritius

Name: Pavan Hanjary

Student ID: M00912754

Supervisor: Dr. Aditya Santokhee

1 Table of Contents

2 Table of Figures

3 Table of Tables

4 Abstract

The Public Health Care Sector (PHC) of Mauritius uses paper-based records to keep records of patients. Paper-based administration faces various issues such as loss of patient records, medication errors, storage limitation and the inability to share patient records.

Applications exist which contains these features, but they lack all the functionalities necessary for the PHC of Mauritius. Some lack complete records of patients while some lack a proper appointment scheduling system. This project aims to cater for these limitations with the implementation of SmartCare.

SmartCare has been designed to view patient records, update patient records and an appointment scheduling system which makes use of machine learning. The viewing and updating of records include e-prescriptions sent via email to patients, viewing and uploading of doctor's notes and the viewing and uploading of radiology and laboratory reports. The appointment scheduling system comprises the use of the greedy algorithm to for priority scheduling of patient appointments. Priority of patients is determined by using NLP to extract symptoms from the notes uploaded by doctors and a random forest classification model.

The evaluation of the system was carried out by assessing the project functionalities and by assessing the appointment scheduling system. The evaluation process led to conclusion that the project needs a better classification model with higher accuracy compared to the 0.93 obtained from the random forest model to determine severity of patients and a slight improvement in the greedy algorithm used to cater for unexpected circumstances. Besides, the project provided a user-friendly interface and the ability to view and update patient records which can prove beneficial for the PHC of Mauritius. The use of emails to send prescriptions and book appointments is another strong feature of the project. Therefore, SmartCare can be considered as a step towards the digitalising health records for the PHC of Mauritius with future improvements for the enhancement of the application.

Acknowledgement

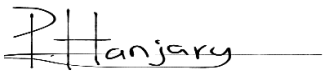
I would like to express my deepest gratitude to Dr. Aditya Santokhee, my supervisor, who has guided me through the different steps in realising this project with his continuous support and constructive feedback during our meeting sessions. His insights played a vital role in the development of SmartCare.

I also express my heartfelt thanks towards my friends and family members who have supported me throughout this journey. Their patience, inspiration and belief in this project made it an enriching experience which is to be bestowed upon as it gave me confidence and strength to persevere.

Declaration

I hereby confirm that the work presented here in this report and in all other associated material is wholly my own work.

Signature:



Date: 01/07/2025

5 Chapter 1 – Introduction

5.1 Background of study

Healthcare is the most important aspect of our society without which the life expectancy of humans would most likely drastically decrease. Healthcare providers face many challenges to offer their services to numerous patients on a daily basis. Handling the records of such a large number of patients can be tedious and difficult. This is where health management systems help accomplish this task effectively and efficiently (Mocdoc.com, 2024).

The healthcare system in Mauritius comprises of both the public sector and private sector. The healthcare facilities consist of 5 major public hospitals, 6 specialised public hospitals, 18 private clinics and 28 medical laboratories. The public sector is accountable for the treatment of 73% of the population (MAURITIUS' HEALTH SECTOR, n.d.). The healthcare system in Mauritius has many issues which include inefficient distribution of human resources and the use of paper-based administration (Thelwell, 2022).

In its Health Sector Strategic Plan 2020-2024, the Ministry of Health and Wellness (MOHW) claimed that Integrated Primary Health Care Services as its priority. The MOHW then requested the World Health Organisation (WHO) to review the current Primary Health Care (PHC) system in Mauritius and provide insights on how to improve the PHC system. From November 2023 to May 2024, a team from WHO worked with the MOHW to conduct a thorough assessment on the weaknesses and strengths of the PHC in Mauritius. Among the recommendations made by WHO, digitalising healthcare and enhancing the availability and utilisation of data to inform decision making were mentioned (WHO | Regional Office for Africa, 2024).

5.2 Problem Statement

As mentioned above, Mauritius uses a paper-based method to store patient records. The latter causes a plethora of challenges for the healthcare sector. The following list outlines the issues faced which leads to need for an improved hospital management system:

1. **The limitation of storage**
   Medical records use up a lot of space, specially in hospitals with high inflow of patients. With increased number of patients, it becomes costly to upscale storage space to accommodate the records of new patients (John, 2022).

2. **Limited backups**
   Physical files once lost is irreversible and may have life threatening consequences to some patients. With physical medical records, the risk of damaged files due to

unforeseen circumstances like a fire is high. With no backup available of the records, it becomes impossible to recover lost information.

3. **Higher risk of errors**
   Paper records are time consuming and more prone to errors. Different handwriting of doctors can lead to others misreading crucial information such as giving the wrong medications to a patient (John, 2022).

4. **Difficult to share information**
   With paper-based medical health records, it is difficult and time consuming to locate and share critical patient information across different healthcare providers and across different departments. This lack of accessibility can slow down decision making and affecting patient outcomes (Ehrinpractice.com, 2024).

5.3 Aims and Objectives

5.3.1 Aims

The aim of this application is to increase the effectiveness of the healthcare system in Mauritius in terms of better organisation of patient medical records, better appointment scheduling system and enabling access to patient records in all the public hospitals.

5.3.2 Objectives

The main objectives of the application are outlined below:

- Carry out literature review.
- Requirement analysis and planning.
- Design a user-friendly frontend for users of every age and technological background.
- Using AI to create a model for automatic appointment scheduling.
- Use appropriate technologies to enable interoperability.
- Implement proper security to prevent loss of data or illegal access to patient's personal data.
- Testing and quality assurance of the application to ensure maximum user satisfaction.
- Proper evaluation of the system to suggest improvements for the future.
- Proper documentation of all the steps involved in the project.
- Evaluation of the system.

5.4 Key activities of the project

5.4.1 Development methodology

For the development of this application the agile methodology will be used. The app will consist of short sprints and increments. The feedback from supervisors will then be used to make the necessary modifications to ensure better end product. The agile method is more suitable as changes can be easily made compared to the waterfall method where the development lifecycle must be restarted to implement new features to the application.

Figure 1 - Agile Method

(Michaud, 2024)

5.4.2 Literature review

A literature review will be conducted before starting the design of the application. This will involve reading and summarising scholarly articles, journals and articles related to the topic to identify related projects. A critical analysis of the related projects will be conducted to find gaps and problems to help improve the currently available systems. Some of the sources where information will be gathered are Google Scholar, Springer, IEEE Xplore among many other relevant sources.

5.4.3 System analysis and design

Thorough research will be carried out on existing technologies that can be used to implement the features of the project. The different technologies will be compared, and the most appropriate ones will be chosen. For the frontend of the application, wireframes will be created and will be used as a skeleton. The system architecture will be designed through diagrams to illustrate the relationship between the different components of the project. The database structure will be designed prior to implementation to ensure optimal data management.

5.4.4 Implementation

As mentioned earlier, the agile methodology will be used for this project. The implementation with kickstart with the frontend implementation following the wireframes designs. Then the different features such as patient record management, use

of AI for appointment scheduling system and the login system will be implemented. At the same time, the database will also be linked to the backend to enable data storage, retrieval and any altering that needs to be made.

5.4.5 Testing

Testing will be carried out after each feature implementation with respect to the agile methodology being used. Input validation will also be used to ensure data accuracy and prevent errors. At the last stage of the project, integration testing, user testing and feature testing will be carried out to evaluate the system for errors and further improvements.

5.5 Project Plan

5.5.1 Milestones

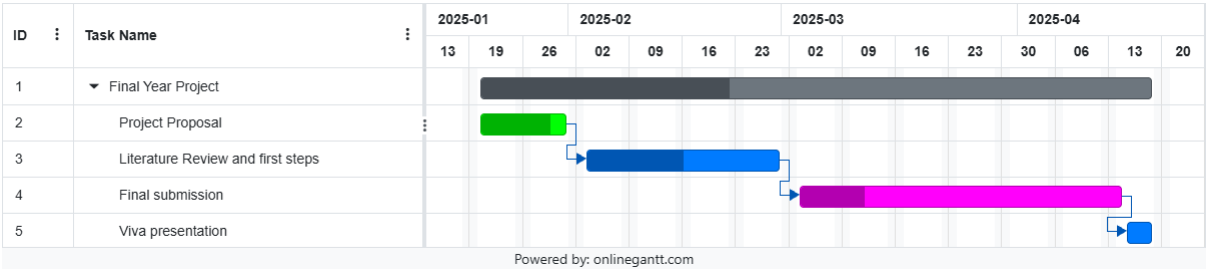


Figure 2 - Milestones

5.5.2 Main deliverables

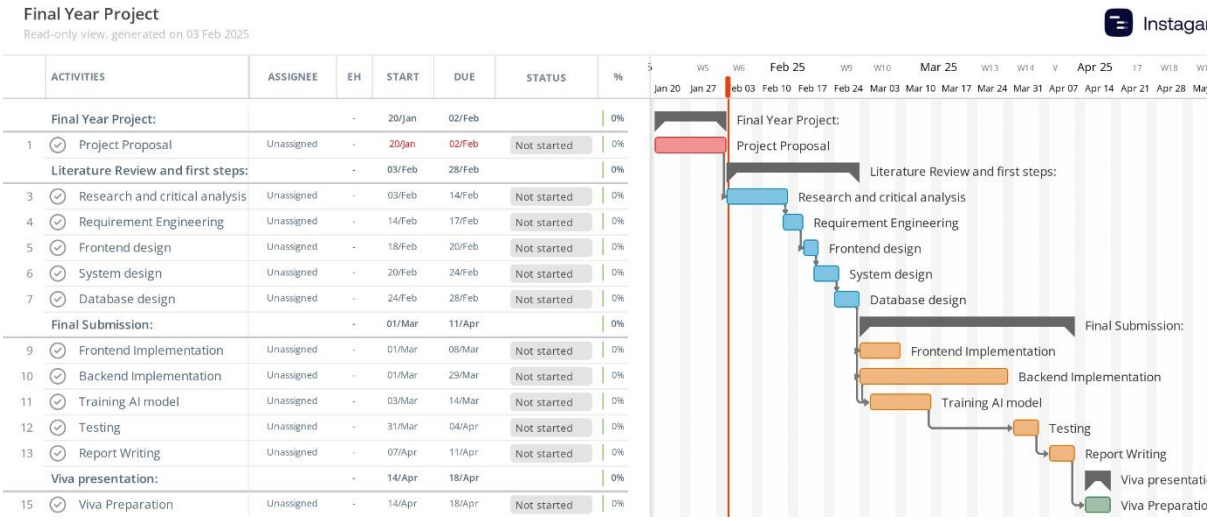


Figure 3 - Main Deliverables

5.6 Project Resources

Personal laptop: Used for research, planning and report writing.

Project management tools

- GanttProject: Used to create Gantt chart for the project planning.

System design tools

- UMLet: Used for drawing UML diagrams for the system design.
- Figma: Used to design wireframes for the frontend.

Development tools

- Visual Studio Code: Used the Integrated Development Environment (IDE) for the project.
- Github: Used for version control.
- Postman: Used to test APIs.
- MongoDB compass: Used to view MongoDB database collesctions.

6 Chapter 2 - Literature Review

The literature review for this project has been caried out using resources like Google Scholar, IEEE Xplore and springer. Research was carried out based on the keywords related to the topic which includes “Hospital Management System”, “Hospital Information System”, “Electronic Health Records” and “interoperability”. The papers obtained from the research were filtered and the papers which best suit the problem were selected for the literature review. Only papers for the past 5 years were considered to review only the most recent solutions implemented.

6.1 Evaluation of existing solutions

6.1.1 Existing Solution 1

This existing solution is a web-based hospital management system (HMS) aimed to be used in healthcare facilities who want to move to an online management system from the deprecated paper-based system. The system allows doctors to manage appointments and write prescriptions. It also consists of a patient module which provides an online appointment booking system, allows patients to view doctor’s prescriptions and enables online payment of doctor’s fees (Babu et al., 2023).

6.1.1.1 Strengths of existing solution 1

1. Online appointment booking system
   The patients can log into the system and book appointments. They can choose the doctor with whom they want an appointment, view consultation fees of the different doctors, view their appointment status and details of their upcoming appointments. These features facilitate the booking and managing of appointments while sparing the need for patients to travel to the hospital to make a booking (Babu et al., 2023).

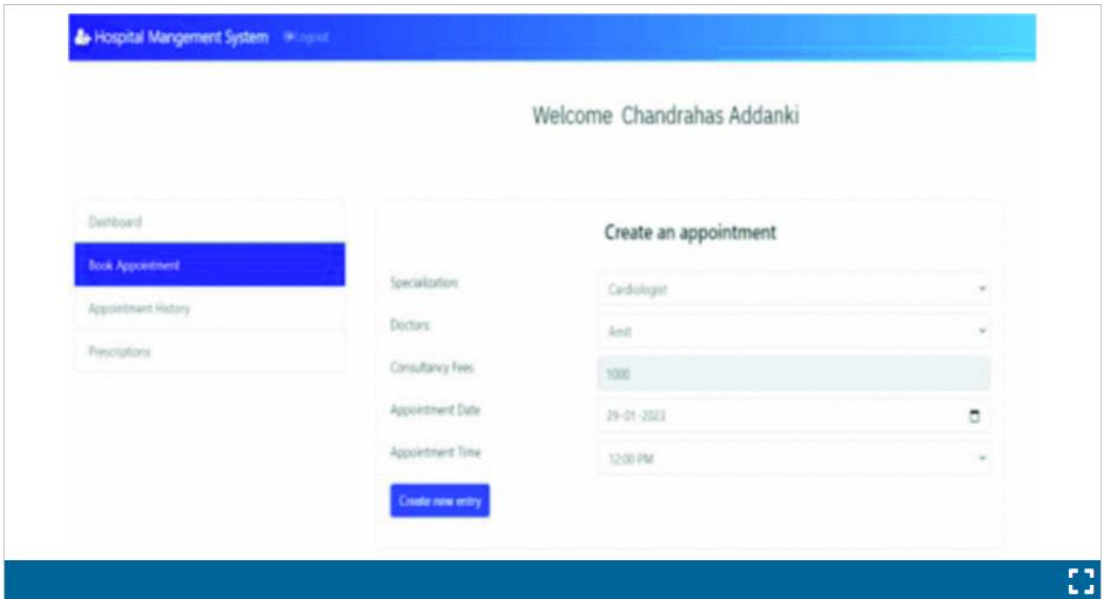


Figure 4 - Book appointment (Babu et al., 2023)

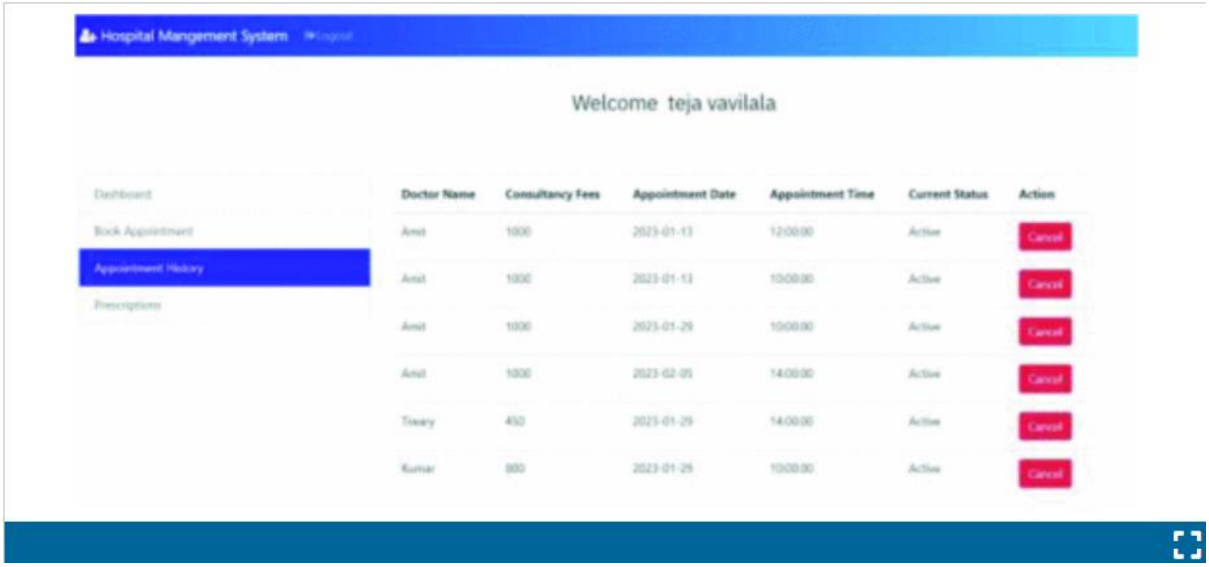


Figure 5 - View appointment history (Babu et al., 2023)

2. Online prescriptions

   One key feature of this system is the ability to have online prescriptions. When a doctor consults a patient, there is no need to prescribe medicines on paper. Instead, the doctor can write the prescriptions online in the system where later the patient can log into the system and view the prescribed medicines. The feature contributes to the aim of the system to make the old paper-based method obsolete by using more efficient and cost saving methods in healthcare systems (Babu et al., 2023).

3. Easy payment of doctor's fees
   Another key feature of the system is the online payment of doctor's fees. Nowadays, digital payment is slowly taking over the traditional cash payment for various transactions. It is considered more secure as we do not need to carry cash around with the risk of it being stolen or lost. Therefore, this feature allows patient to pay their bills with higher security and without the need to carry cash around. It also implies that the hospital no longer has to employ a cashier to collect fees and saves money (Babu et al., 2023).

6.1.1.2 Weaknesses of existing solution 1

1. Uses a queue algorithm for appointments
   The appointment scheduling feature, despite being a brilliant feature, has one major drawback. It uses a queue algorithm for patient appointments. In other words, patients are placed at the end of a queue of patients if they are late for their appointment. This may increase the waiting times of patients significantly and reduce the efficiency of the healthcare system.

2. Doctors cannot view patient's health record
   One of the most important features of a hospital management system is probably the ability to view the record of patients. This system was designed to enable healthcare providers to shift from the paper-based method of keeping records to an online method. However, the system lacks this feature which is a major drawback. The healthcare centre will still have to have staff members to handle records and dedicate large storage areas to store the patient records.

3. Patients must manually book appointments
   The appointment booking system does not cater for automatic reappointments in a situation where a doctor needs to review a patient after a certain amount of time. This could be achieved through an automatic booking system where a patient is automatically assigned an appointment date based on the recommendation of the doctor.

6.1.1.3 Summary

This web-based solution solves issues in current paper-based hospital management system partially. The system has interesting features like the online payment of fees, online prescriptions and online appointment booking system, but it lacks one of the most important features of hospital management systems. The latter being the ability to view a patient's medical history. In addition to that, this system lacks some automation whereby all the tasks must be carried out by either the patient or the doctor. Thus, this system doesn't suit the current Mauritius public healthcare system due to its limitations in terms of managing patient records and the type of appointment scheduling used.

6.1.2 Existing solution 2

This solution is a web application designed for hospitals to increase the efficiency and facilitate hospital management. The latter enables patient to book appointments, view their online prescriptions and their medical profile. On the same wavelength, doctors can write online prescriptions, view patients' medical history and manage their schedule. All the information related to patients and doctors is stored using a database management system (Devi et al., 2021).

6.1.2.1 Strengths of existing solution 2

1. Online appointment booking and reminder
   This web application provides an online appointment booking system for patients. This makes appointments bookings much easier as patient no need to travel to the hospital for this task. If a patient has booked a time slot with a doctor, a reminder will be via the patient's email address near the date of the appointment. If cancellation of the appointment is done, an email will also be sent to notify the patient. This feature equips the system with some automation and may reduce the need to employ staff, lowering the expenses of the hospital. The figure below shows the feature described (Devi et al., 2021).

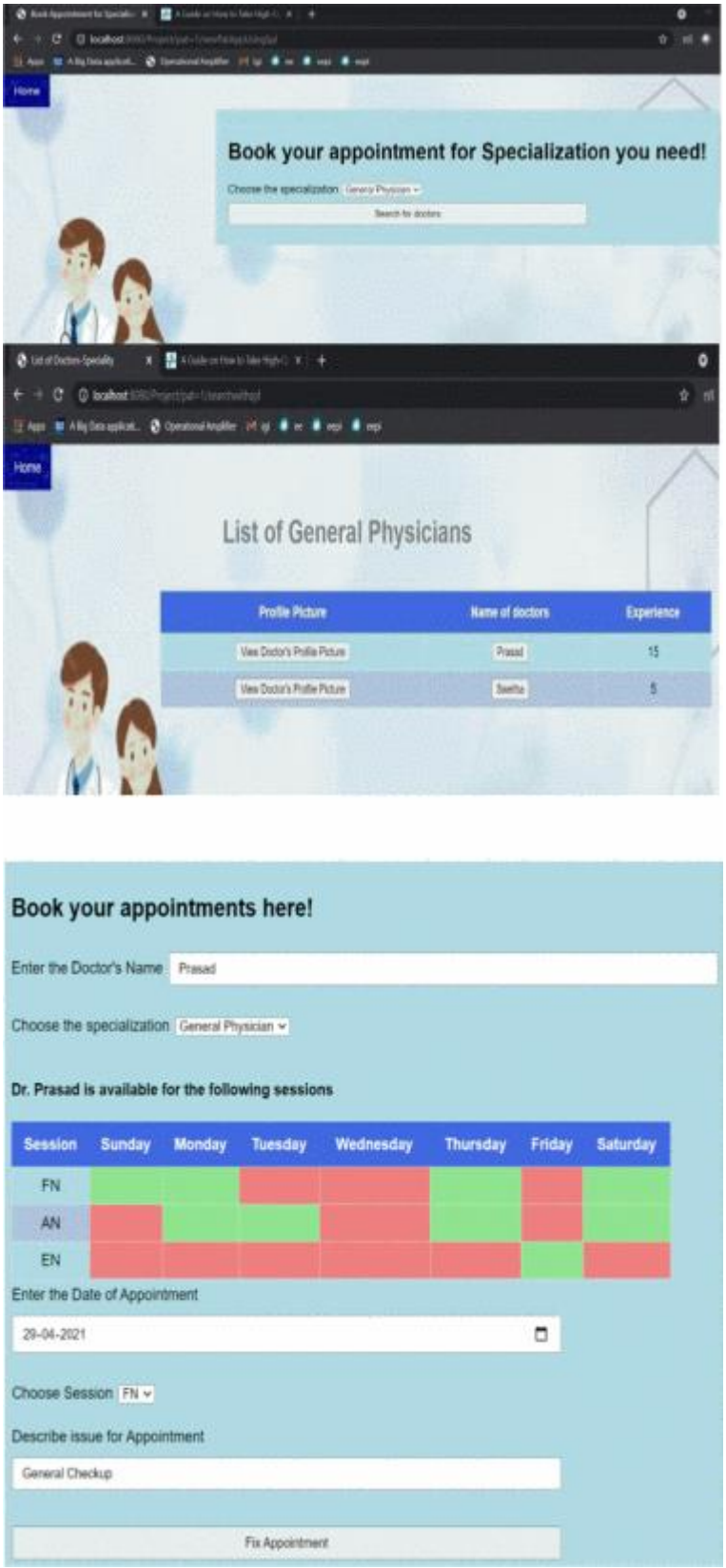


Figure 6 - appointment booking (Devi et al., 2021)

2. Online prescriptions
   Another beneficial feature of this system is the ability to write online prescriptions. Doctors list the medications and the time at which the patient needs to take these

medications. This can be accessed by the patient by logging into the system. This feature may lower medication errors, specially for patients with dementia or Alzheimer, if ever the patient forgets the time of medication. The patient can simply log into the system and view the prescription. The figure below illustrates this feature (Devi et al., 2021).

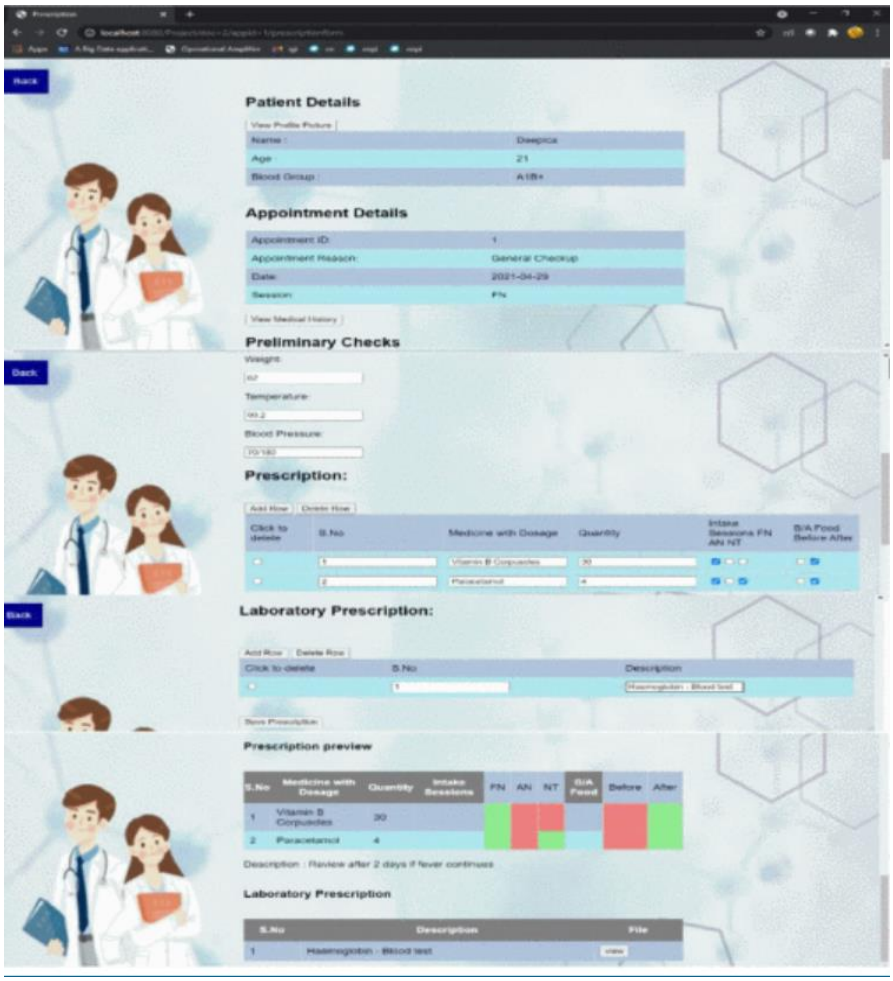


Figure 7 - Online prescription (Devi et al., 2021)

3. Doctor can view medical history of patients
   The ability to view patient medical history is crucial in a hospital management system. This feature allows the doctor to view the previously diagnosed diseases, the laboratory prescriptions and the medicine prescriptions of a patient. This eradicates the necessity for paper-based records and makes consultation more efficient. The figure below illustrates this feature (Devi et al., 2021).

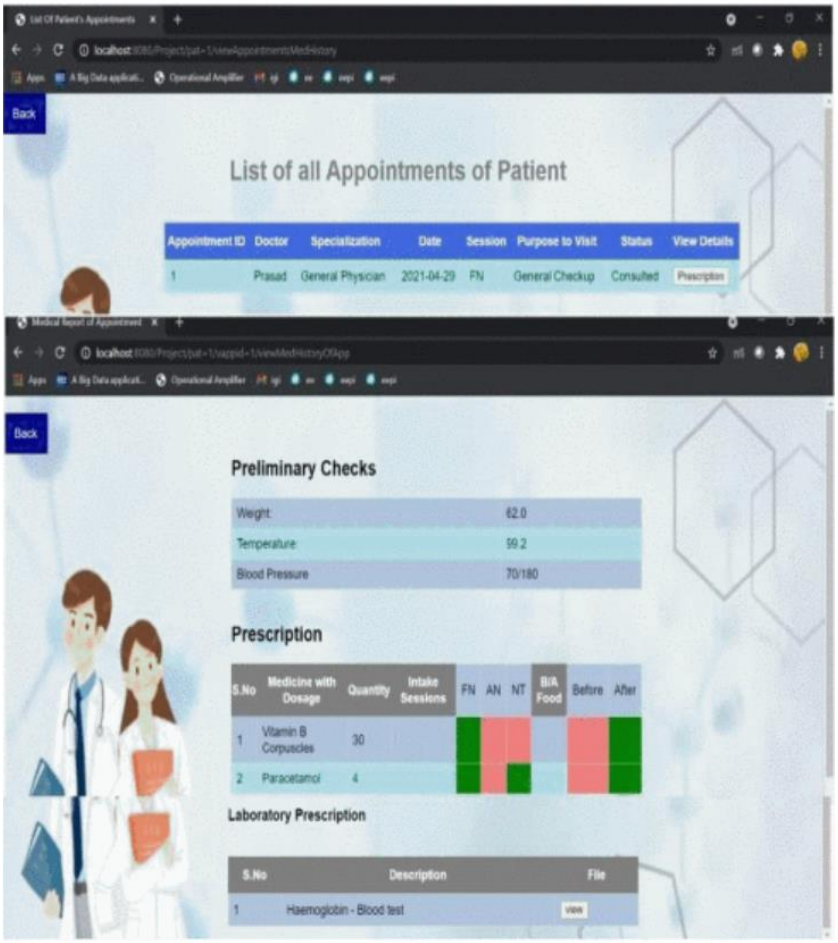


Figure 8 - Medical history of patient (Devi et al., 2021)

6.1.2.2 Weaknesses of existing solution 2

1. Not user friendly
   The website designed is not user friendly to use. The medical history for patients is not well organised in this website which might be detrimental. Separating the different categories into different sections might have been a better way to display patient data.

2. Patient's medical history is incomplete
   The medical record of a patient compromises of crucial information and might be advantageous in numerous ways. A good medical record would help save time, would help doctors to provide effective diagnosis and might help reduce medical errors. In this system, the medical records lack a lot of important information such as doctor's notes, the medical condition of the patient and the chronological order in which the patient has received diagnosis or treatment.

3. Manual booking of appointments
   The system only supports the manual booking of appointments. This may be detrimental for use in the context of the public health sector in Mauritius as it

allows the patients themselves to book the appointments. This may be detrimental for patients requiring urgent treatment if early bookings are not available.

6.1.2.3 Summary

This system contains useful features necessary for a good hospital management system such as online appointment booking system with reminders, the ability to view partial of the medical history of patients and online prescriptions. However, for a hospital management system to be used in the public hospitals of Mauritius, the drawback of this system outweighs its advantages. A better solution suitable for our needs would be a more user-friendly system with detailed information of patient's medical history and a better scheduling system would be more suitable.

6.1.3 Existing Solution 3

This system is a web application designed for Rural Health Units (RHUs) in the Philippines. The goal of this system was to switch to digital record-keeping systems from the paper-based method to increase the effectiveness and efficiency of healthcare services. The application enables recording of doctor's notes, uploading of prescriptions, recording follow-up appointments and shows the diagnoses for a patient (Diaz et al., 2024).

6.1.3.1 Strengths of existing solution 3

1. Sharing of records

   This system allows sharing of patient records based on a request from another public hospital. This sharing of data can be beneficial as a patient can visit another RHU and still have a medical record there. This might improve the effectiveness and efficiency of health services in RHUs. The picture below shows a request for a patient's medical record from another RHU (Diaz et al., 2024).

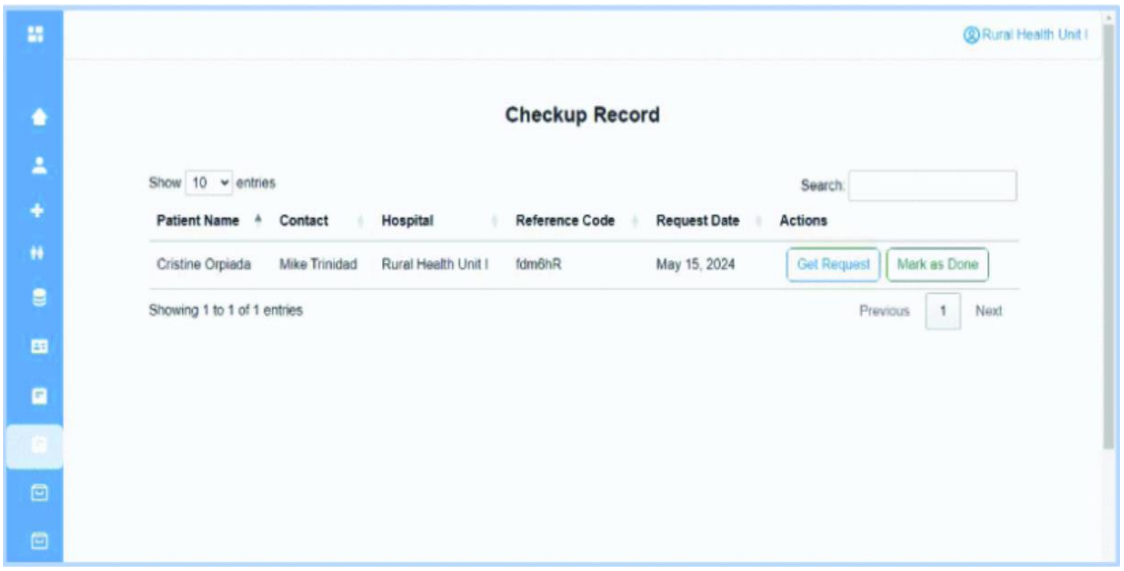


Figure 9 - Sharing of records (Diaz et al., 2024)

2. Uploading of prescriptions
   The system allows doctors to upload prescriptions which can be viewed by the patient. The doctor simply must take a picture of the written prescription and upload it. The picture below illustrates this feature (Diaz et al., 2024).

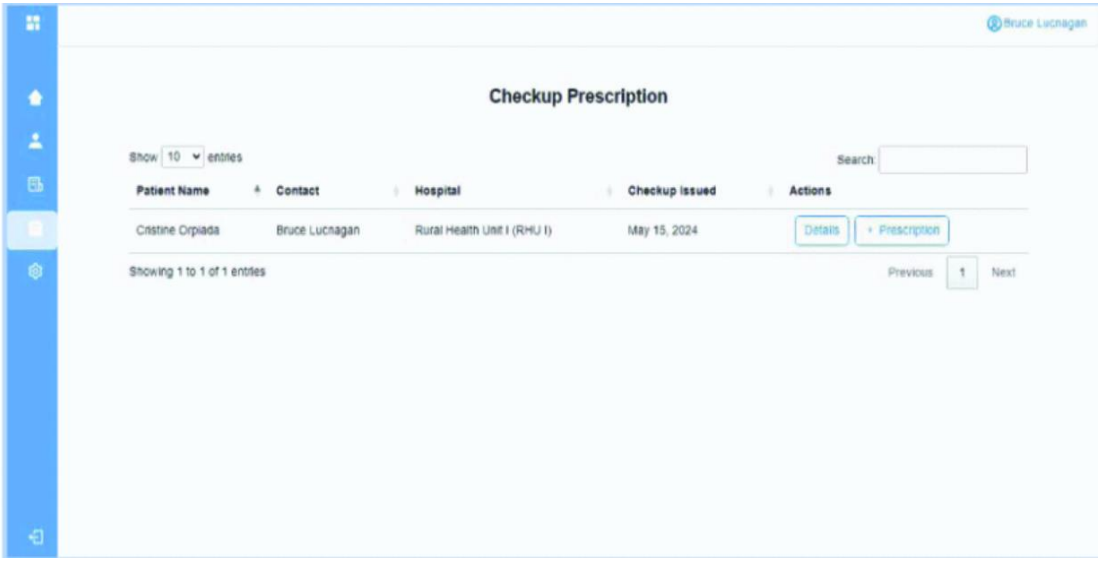


Figure 10 - Online prescription (Diaz et al., 2024)

3. Patient can request a copy of their medical record
   One major advantage with this system is that patients can request a copy of their medical report online. This may be beneficial if a patient wants to consult a private doctor. This may save costs for the patient who does not need to pay for medical tests that have already been done in the public hospital. The picture below shows a request made by a patient for a medical report (Diaz et al., 2024).

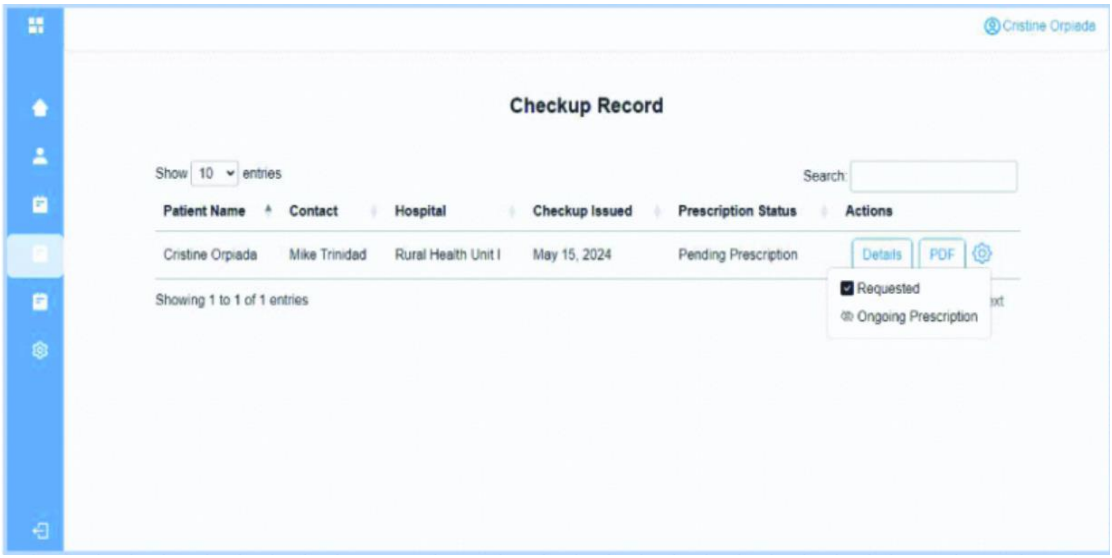


Figure 11 - Medical report request feature (Diaz et al., 2024)

4. Two factor authentication for patient login
   Two factor authentication(2FA) is a method used to double check that it is really you logging into your account. This can be in the form of a one-time password (OTP) sent to your phone (Bu.edu, 2025). 2FA reduces the risk on unauthorised access. In this system, 2FA has been implemented to prevent unauthorised access to personal patient information (Diaz et al., 2024).

6.1.3.2 Weaknesses of existing solution 3

1. Cannot edit record of referred patient
   As mentioned earlier, one RHU can share patient record to another RHU upon receiving a request. However, since the record is only a shared copy, it cannot be edited on the system. This creates inconsistency in a patient's record at different RHUs which might affect the quality of healthcare services.

2. Prescriptions are uploaded instead of written online
   Online prescriptions can only be uploaded by taking a picture of the written prescription. Instead of saving time, this will require the doctor to spend additional time to upload the prescription which might not be very effective.

3. Not user friendly
   The system implemented is not user friendly. The navigation to different sections of the website is crowded without clear indication where each section takes us to. For such a system where doctors of all age with different learning pace with technology will be users, a more user-friendly interface might be more suitable.

6.1.3.3 Summary

This system may be effective and beneficial in terms of the ability to share records between different RHUs, in terms of patient being able to request a copy of their medical record and the security implemented to prevent unauthorised access. However, the features are not cut out for the healthcare system in Mauritius. A more interoperable, user friendly, flexible and automated solution might be better suited for this project.

6.2 Comparative analysis of existing solutions

The following table highlights the comparative analysis of the existing solutions described in the section above. The comparative analysis has been carried out based on the feature considered necessary for a hospital management system for public hospitals in Mauritius.

Criteria	Existing Solution 1	Existing Solution 2	Existing Solution 3
User-friendly	Clear navigation and easy to use User Interface (UI).	Data not organised into sections.	Attractive UI but navigations are not indicative.
Sharing of records	Cannot be shared.	Cannot be shared.	Records can be shared but are not editable.
Online prescriptions	Can prescribe medicines online.	Can view medicine prescribed online.	Present but must be uploaded as a picture of the written prescription.
Appointment scheduling	Present	Present	None
Data Security	Password authentication	Password authentication	2 Factor authentication when logging in.
Electronic Health Records	None	Partial records	Present
Target Audience	General purpose	General purpose	RHUs in Philippines

Table 1 - Comparative analysis of existing solutions

6.3 Appointment Scheduling Problem and algorithm

Appointment scheduling plays an important role in the quality of healthcare service provided. There are multiple factors to be considered when implementing appointment scheduling for outpatients for the public health sector in Mauritius such as length of slots, the scheduling algorithms used and how to deal with no-shows. A study performed

by Kuiper, de Mast and Mandjes, to assess the clinical practices when scheduling outpatient appointments made the following observations:

- No-shows are usually not considered when scheduling appointments. Clinics concentrate more on reminding the patients about the appointment date. This has proven to reduce the no-shows to around 2% - 8%.
- Late arrival of patients is seen as minor issue as the schedule is usually overrun.
- Clinics allocate slots with fixed time length for every patient.
- The study suggests allocating looser schedules to reduce long patient waiting times and reduce session overruns (Kuiper, Mast and Mandjes, 2019).

To address the scheduling of appointment in the public healthcare in Mauritius, some of the practices outlined above may not be the best fit. Since Mauritius has a free healthcare service, the outpatient load is high. Therefore, applying loose schedules may reduce the long patient waiting time for a specific session but it will not be sufficient to cater for the large number of patients on the long run. This method may increase the waiting time for other patients to obtain an appointment due to the fewer number of slots per schedule and due large amount of patient, which may decrease healthcare quality. A study conducted for the National Health Service (NHS) in Portugal concluded that a prioritization-scheduling approach decreases long waiting time and promotes equity for large number of outpatient load (Moura and Pinho, 2025). Therefore, a priority-based scheduling system may be best suited for appointment scheduling in Mauritius. There exists a number of priority scheduling algorithms but the one we will consider is the greedy algorithm. This algorithm will be further analysed below highlighting its features.

6.3.1 Greedy algorithm

The greedy algorithm is known for its simplicity, ease of implementation and efficiency. The greedy algorithm gives preference to the solution that is currently optimal even though the solution might not be globally optimal. This means that it chooses the current best option without taking into consideration if it is the best option for the overall task. It is used in scheduling tasks which includes aspects like skipping idle time, shortest job priority, earliest deadline first and for least time-consuming tasks. Additionally, the algorithm has a low time complexity as it does not need to traverse through all possible solutions. This also makes it highly efficient to find a solution. These features will help change the appointment scheduling system in Mauritius and makes the algorithm a suitable choice for this project (Wang, 2023).

6.4 Critical analysis

6.4.1 Scheduling system

The scheduling system used in the existing solutions all rely on manual booking of slots by the patients themselves. This might not be the best suited approach for the public

health system in Mauritius. In Mauritius, doctors recommend the time delay in the patient record to book the next appointment which is then booked by another staff in charge of this task. The patient is then given a card containing the details of the appointment. This system is not very efficient as it involves redundant work and with the use of physical cards, loss of the card may lead to no shows. Instead, a more practical solution can be implemented in the form of an automatic priority-based scheduling system. Natural Language Processing (NLP) can be used to extract information from doctor's notes and determine the priority of appointment by using a machine learning model. The appointment can then be booked based on the priority of the patient.

6.4.2 Online prescriptions

Written prescriptions may lead to medication errors. A study found that prescription errors are responsible for 70% of medication errors that may potentially result in adverse effects (Velo and Pietro Minuz, 2009). In Mauritius public hospitals, written prescriptions are still being used. In the existing solutions described above, the feature of online prescriptions has been implemented. The doctors can either write e-prescriptions or upload a picture of the written prescription. However, due to the limitations of written prescriptions, e-prescriptions would be a more suitable fit for the healthcare system in Mauritius. This may help keep an organised record of medicines prescribed to patients.

6.4.3 User friendly system

A user-friendly website should be professional and the information on the website should be well organised for good usability (Singh, 2013). A user-friendly website should be self explanatory, that is much effort should not be put to navigate and use the website (Merge.rocks, 2022). For the existing solutions, the information on the website have not been properly organised and for some, the navigation is not clear and easy to use. A hospital management system is used by doctors and hospital staff of all ages. Some of them may be more technologically challenged than the other. Thus, a user-friendly HMS is of paramount importance.

6.4.4 Electronic health records

An electronic health record (EHR) is a digital copy of a patient's medical record. The EHR includes past medical history, laboratory data, radiology reports, medications and progress notes (Cms.gov, 2024). EHR also allows electronic sharing of data and can be beneficial in numerous ways. This may be in the form of higher quality medical care being delivered, better management of patient records and providing accurate, up-to-date, and complete information about patients (Healthit.gov, 2025). The existing solutions have features such as storing partial information about patient's medical history and sharing records which cannot be edited. The unorganised nature and limitations of these features make them unsuitable for the HMS for public hospitals in Mauritius.

6.5 Proposed solution

SmartCare will be a web-based hospital management system (HMS) for Mauritius public hospitals. The system will be accessible to all hospitals sharing the same data to enable access to patient records in all hospitals. The patient records will be editable at any hospital unlike the existing solutions which allowed only sharing of immutable records. The patient records will be stored as electronic health records (EHRs) containing information about their medical history, medication history, laboratory results and radiology results. The system will also consist of an e-prescribing system with the aim to reduce medication errors. The medicines prescribed will then be emailed to the patient and stored in the patient's record to keep up to date medical history.

An appointment booking system will also be implemented to give appointments based on patient health condition severity. The system will extract symptoms from doctor's notes and use machine learning classification techniques to classify the patient into a patient severity category. This will then be used to book appointments on a priority-based technique using the greedy algorithm. This will be aimed to increase the efficiency and quality of healthcare provided. The system will also take into consideration doctor's schedule when booking appointments. The appointment details will then be emailed to the patient.

The system will also be designed to be user-friendly for staff of any age and technological background. Clear navigations will be used to increase the usability of the system.

7 Chapter 3 – Requirement Analysis and System Design

7.1 System requirements

7.1.1 Functional requirements

ID	Function	Description
F1	Users must be able to log into the system.	The user will be able to log into the system using a username and a password.
F2	Users must be able to search for a patient.	On the main page, users must be able to search for a patient using their name. The user will then select the patient to view their medical record.
F3	Users must be able to view patient's medical record.	Users will be able to view a patient's medical history in chronological order. The user will be able to access past laboratory results, radiology

		results, doctor's notes and medication history.
F4	User must be able to add new data to a patient's record.	Users will be able to create new medical notes, upload new laboratory results and upload new radiology results to a patient's record.
F5	User will be able to prescribe medicines to patient.	Users will be able to write prescriptions for a patient which will be automatically added to the record of the patient and sent via email to the patient.
F6	The system will automatically book appointment for patients and send an email to the patient for the appointment.	The system will be able to extract information from doctor's notes and determine the gravity of the patient. Based on the gravity of the patient, an appointment will be booked for the patient. A confirmation email will be sent to the patient containing the appointment details.

Table 2 - Functional requirements

7.1.2 Non-Functional requirements

The following non-functional requirements will be present in the system:

- The UI must be user-friendly and easy to use.
- Doctor's profile must be secured.
- The system should support concurrent users.
- Data should be accessible in different hospitals.
- In case of power cut or server downtime, no data should be lost.

7.2 Tools and technologies

The following section consists of the selection of the appropriate tools and technologies that will be used for the development of the system. The tools and technologies are going to be selected for the frontend, backend, hosting services and data storage. The selection process will aim to choose the most appropriate tool which aligns with the aim of the project, and which will facilitate development of our system.

7.2.1 Frontend

React.js: React.js is a framework which is used in the development of interactive website frontend. It uses components to help build user interfaces (React.dev, 2015). It is better suited for this project due to its high performance and extensive toolset available (Oleksandr Hutsulyak, 2024).

7.2.2 Backend

Node.js: Node.js is an open-source JavaScript runtime environment. Node.js is more suitable for this project as it allows multiple concurrent connections with a single server (Nodejs.org, 2015).

7.2.3 Database

MongoDB atlas: The cloud version of MongoDB will be used to store the data for the application as the data will be accessible in all the public hospitals aligning with the aim of the project. MongoDB allows flexible document schemas which allows data to be modelled and manipulated easily (MongoDB, 2024).

7.3 Design Diagrams

7.3.1 Wireframes

The wireframes below show the user interface design for SmartCare.

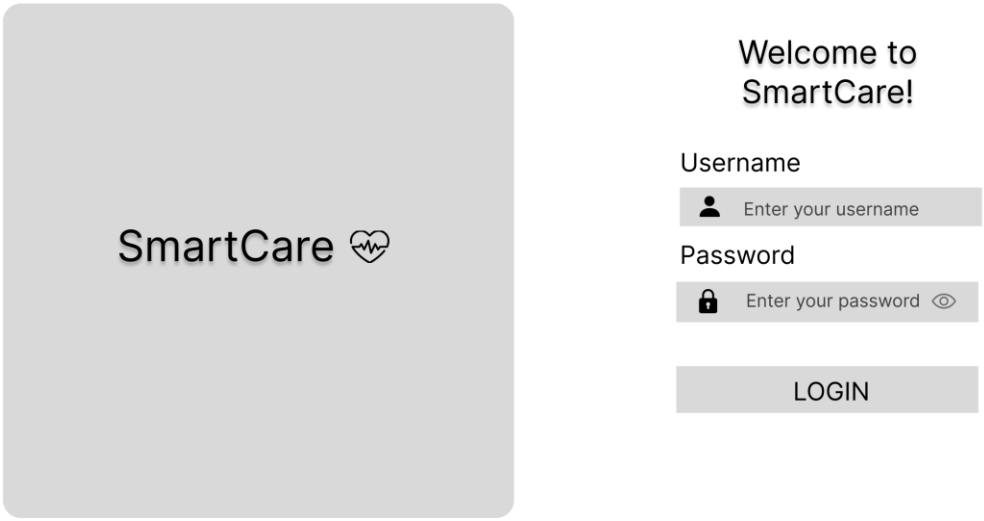


Figure 12 - wireframe for login page

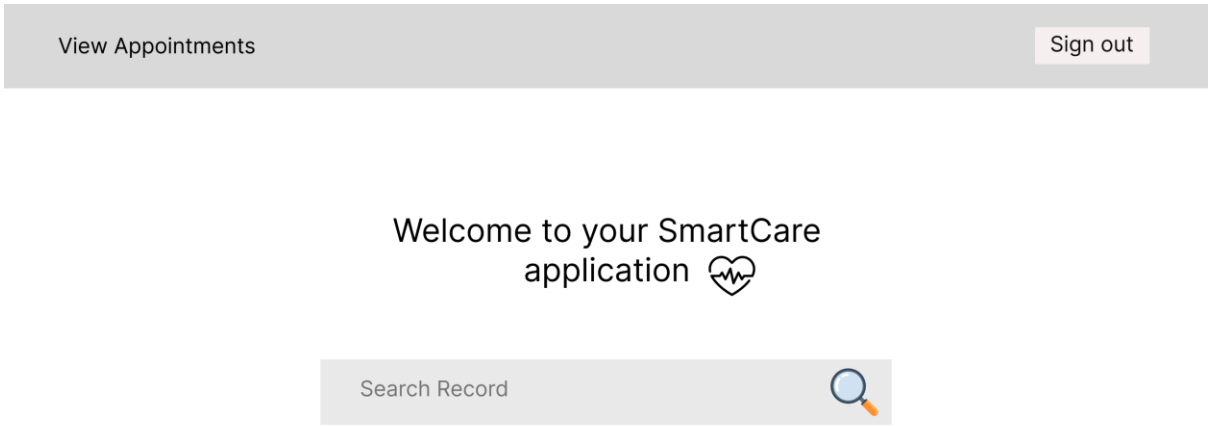


Figure 13 - wireframe for home page

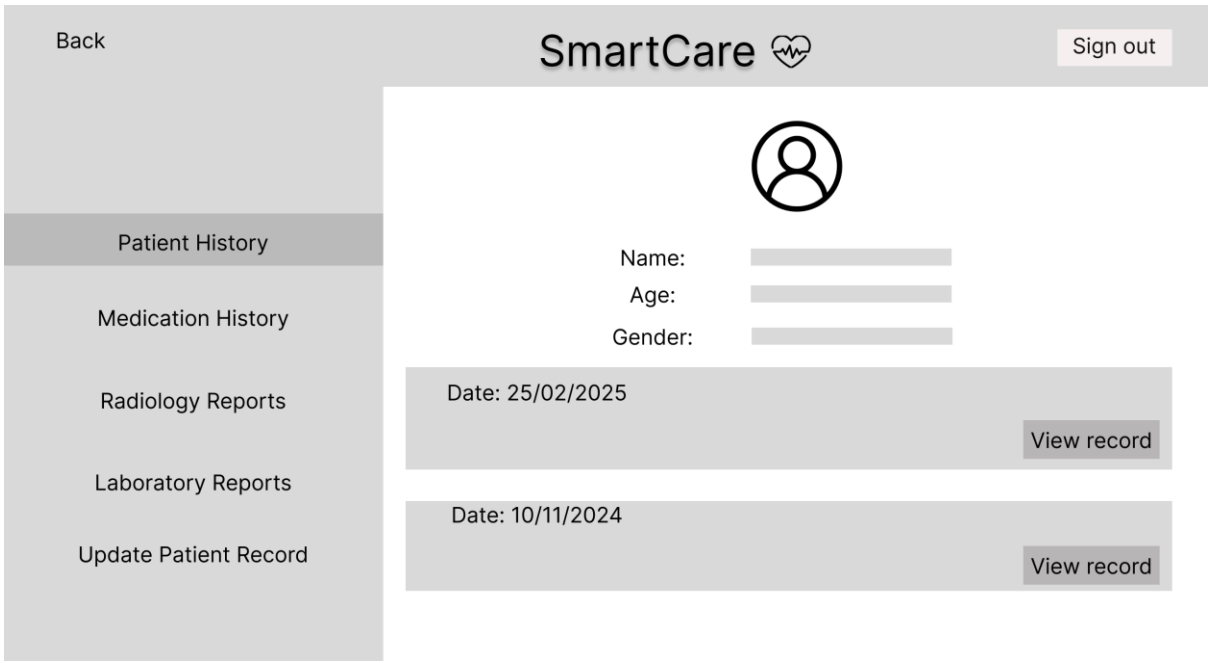


Figure 14 - wireframe for view patient history

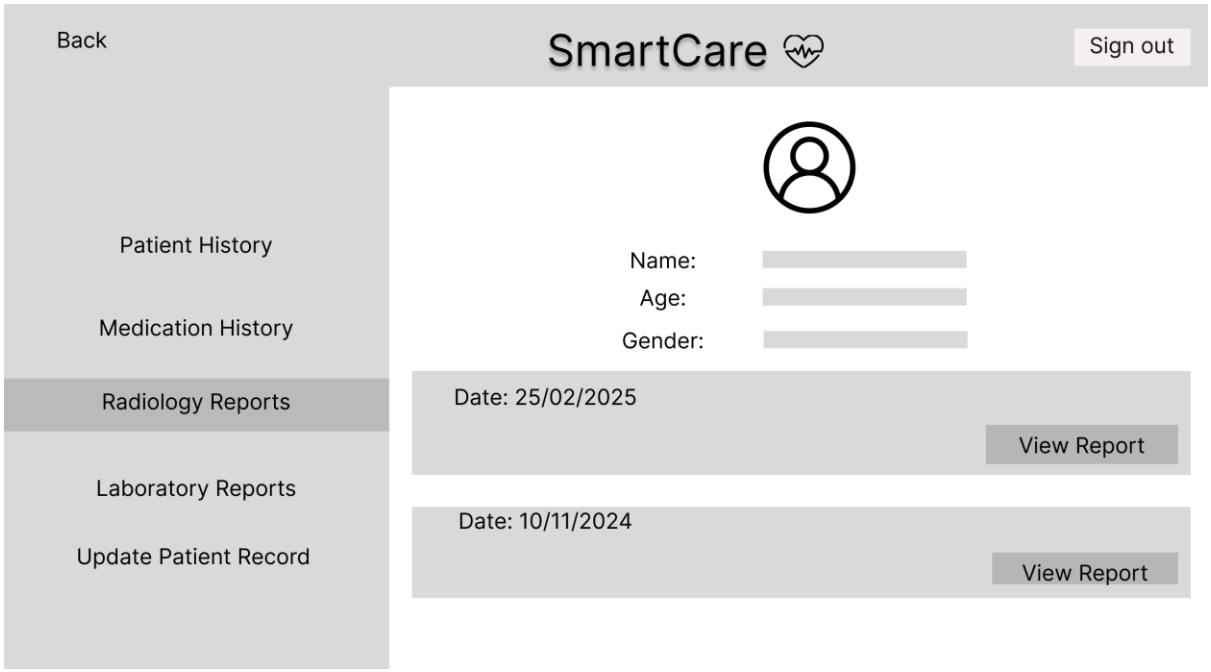


Figure 15 - Wireframe for viewing radiology reports

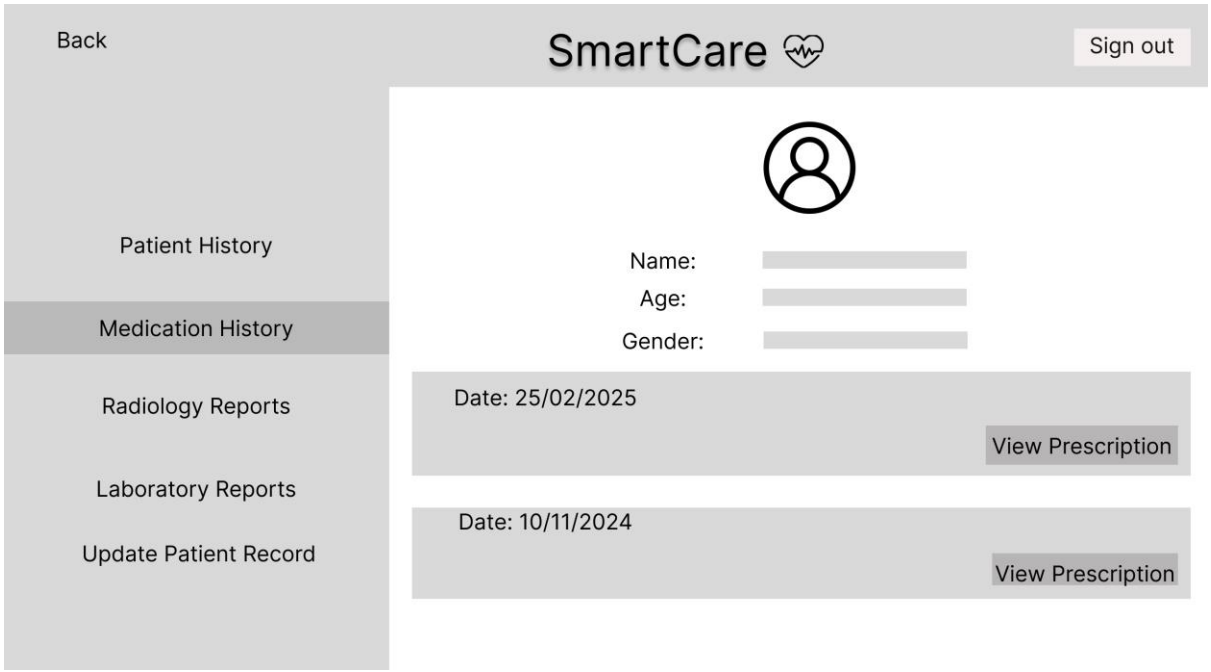


Figure 16 - wireframe to view prescription history

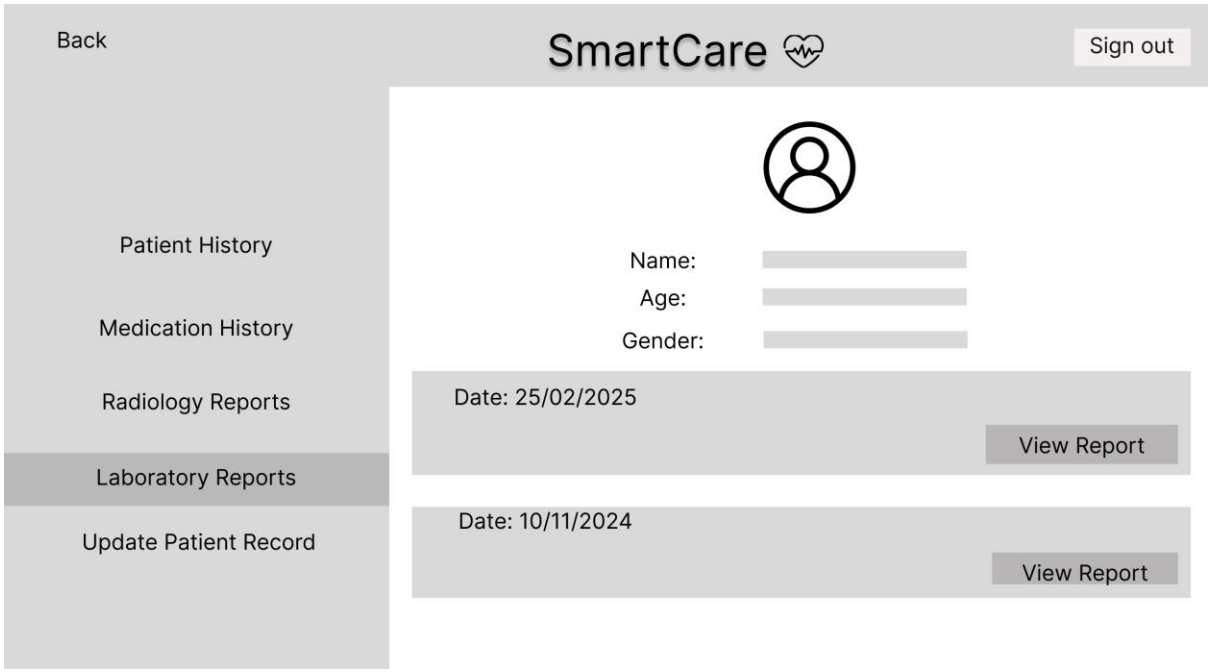


Figure 17 - Wireframe to view laboratory reports

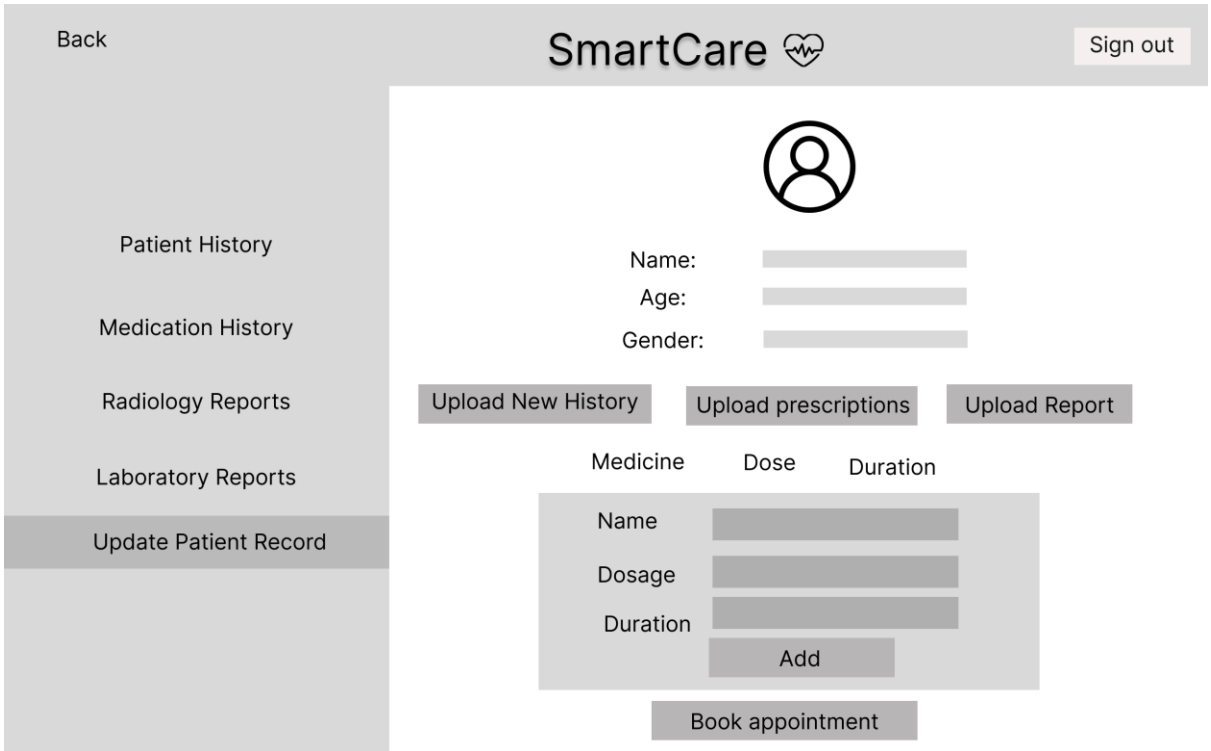


Figure 18 - Wireframe to add new patient notes

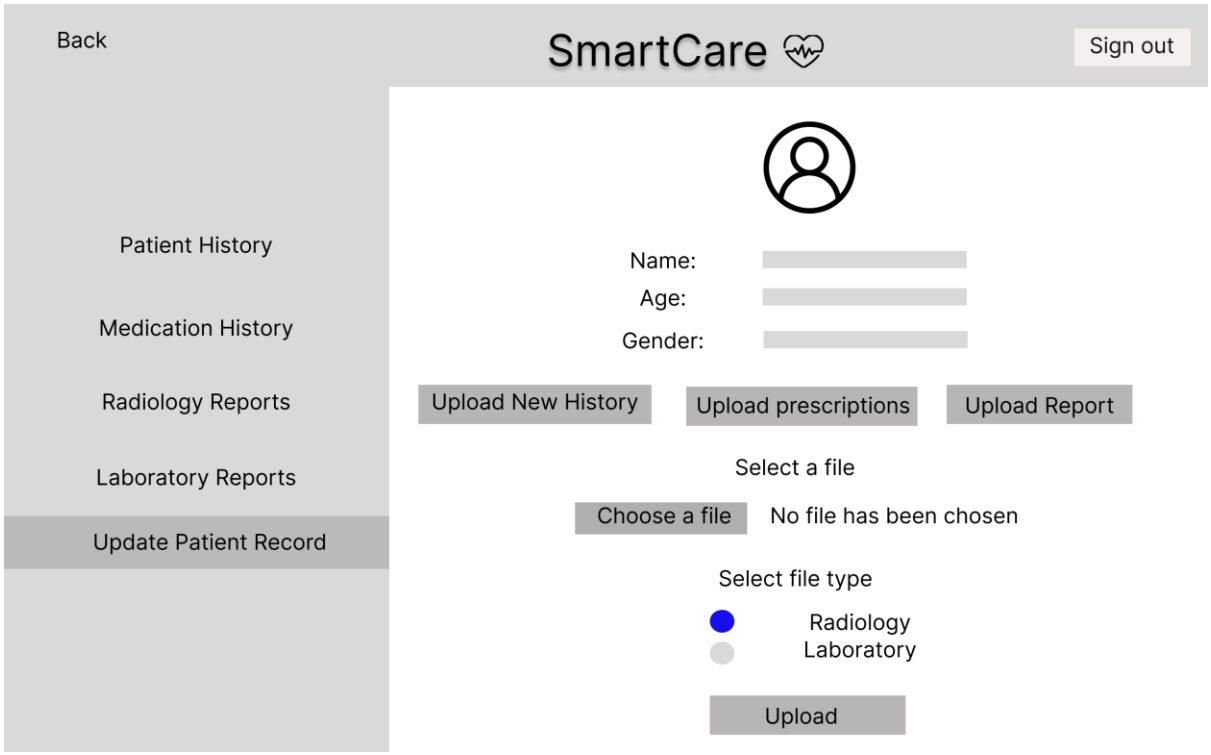


Figure 19 - Wireframe to upload new file

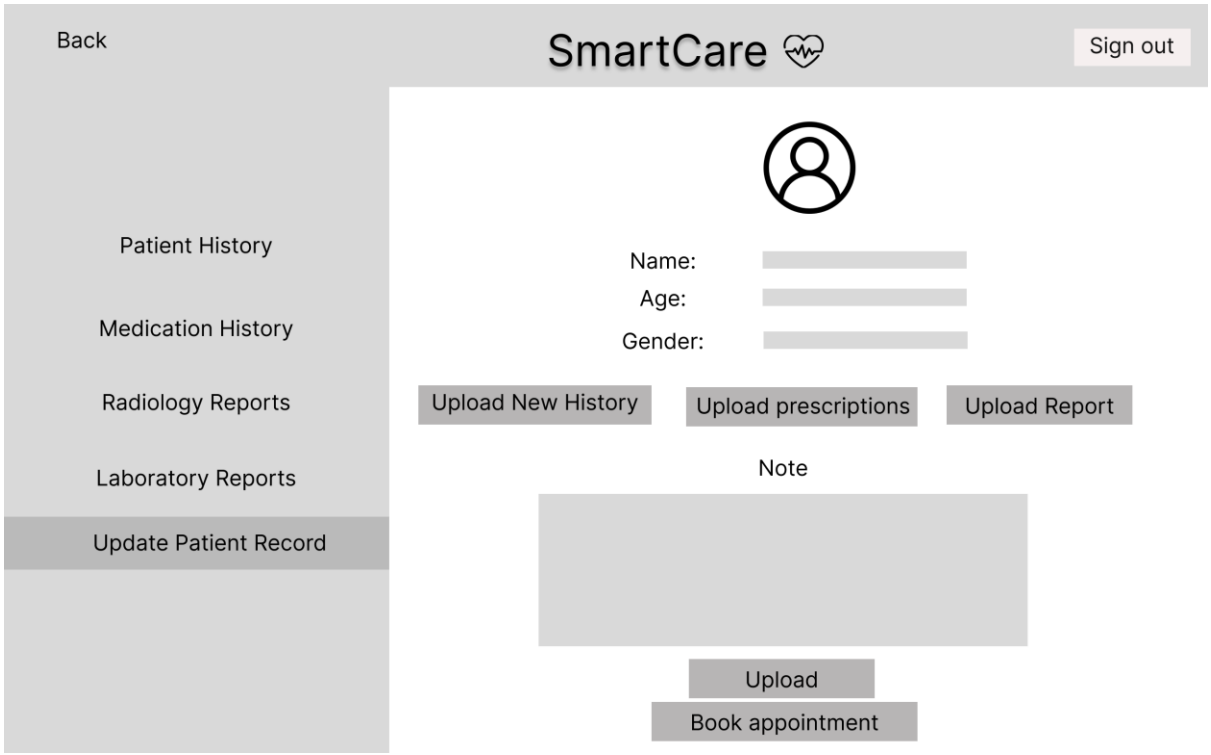


Figure 20 - wireframe to write e-prescription

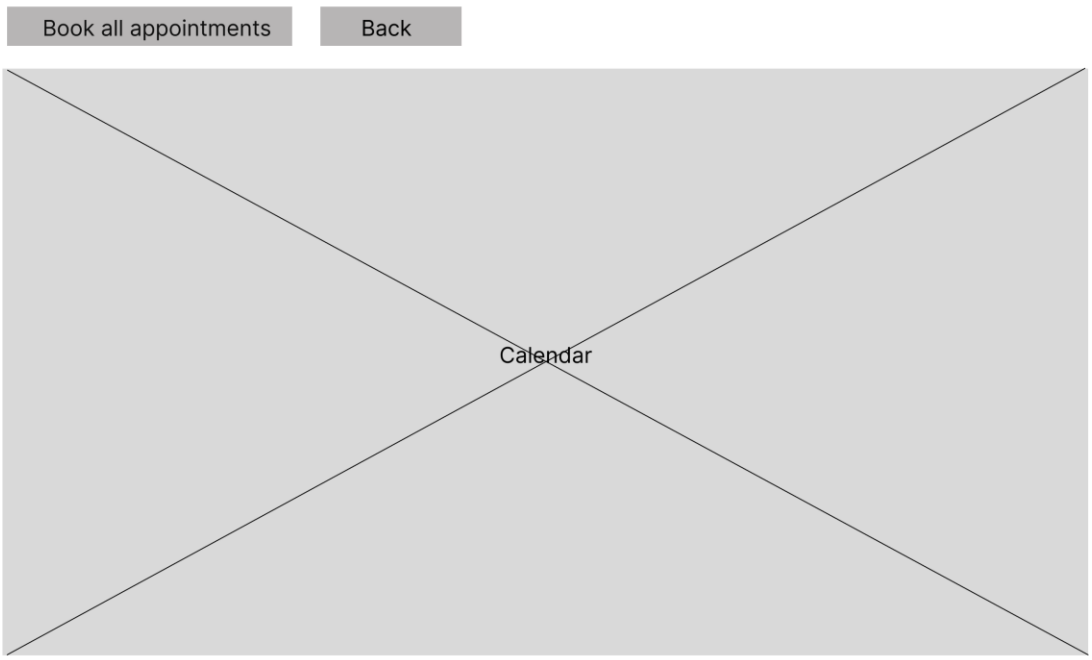


Figure 21 - Wireframe to view doctor's calendar

7.3.2 Database Design

1. **Patient collection (patients)**
    - patient_ID
    - name
    - email
    - age
    - gender
    - dateOfBirth
2. **Medical history collection (patientHistory)**
    - patient_ID
    - date
    - note
3. **Prescription collection (medicationHistory)**
    - patient_ID
    - date
    - medications (name, dosage, duration)
4. **Radiology report collection (radiologyReports)**
    - patient_ID
    - date
    - file
5. **Laboratory report collection (laboratoryReports)**

- patient_ID
- date
- file

6. **Doctor collection (doctors)**
    - doctor_ID
    - name
    - password
    - appointments (title, start, end)

8 Chapter 4 – Implementation

This section follows the successful implementation of the SmartCare application. In this section, we will discuss the key features of the project, a description of the development environment, the project folder description for the whole application, including the server side, the client side and the API folder structure, and a detailed explanation of the functionalities of the project.

For better understanding and illustration of the concepts and functionalities implemented, code snippets will be added followed by description of the implemented code. This will help in understanding the backend, server and API logic used in making the SmartCare application.

8.1 Key Features

SmartCare comprises of many key features which aids to fulfilling its goal in changing how the healthcare system in Mauritius operates. The key features are outlined in this section along with how the key features have been implemented and the impact of these features in revolutionising the healthcare system in Mauritius.

Login

- **Functionality:** The login is used to allow only authorised users into the system to view patient records.
- **Technical Implementation:** The user inputs the username and password in the frontend and clicks on the login button. A post request is sent to the Node.js server to validate the user credentials and return an appropriate response. If user credentials are valid, the user is redirected to the home page of the application.
- **Impact:** Prevents unauthorised access to the system.

Patient Search

- **Functionality:** The patient search functionality helps the doctor to look for patients by typing their full name in an input box. On clicking the search button,

the patient's medical record is displayed along with an option to view the patient's full medical record.

- **Technical Implementation:** The frontend for the functionality has been implemented using React.js which uses a state to store the input from the search bar. The input is then used to send a request to the backend Node.js server to query the patient from MongoDB. The response from the server is an array of the patient records having the same name as the value of the search bar.
- **Impact:** This provides faster searching of records compared to the old paper-based methods used and improves the quality and efficiency of healthcare service.

View patient's medical records

- **Functionality:** The view patient records functionality allows the doctor to view a chronological order of the different types of patient history. For instance, the doctor can access all the doctor's notes from previous visits, all the prescriptions from previous visits and all laboratory and radiology reports.
- **Technical Implementation:** The frontend has been implemented using React.js. The frontend queries the Node.js backend to fetch all the records from MongoDB. The response is then displayed in the frontend. To display patient laboratory or radiology results, **React-pdf** package has been used to enable to view the pdf in the application itself.
- **Impact:** This functionality provides easy access to all the patient's medical history and mitigates the issue of lost records when using traditional paper-based methods. This helps preserve complete history of patient and increase healthcare service quality.

Update patient's medical records

- **Functionality:** This functionality allows doctors to add new information to the patient's medical history. New radiology and laboratory reports can be uploaded. It also enables the uploading of new doctor's notes and new prescriptions.
- **Technical Implementation:** The React.js frontend allows both file input and text input for the required part of medical history to be uploaded. The input is then sent to the Node.js backend to be saved to MongoDB. For the file uploading, **multer** (NPM library) is used to save the files to the server.
- **Impact:** This functionality enables easy uploading of new patient history, keeping records up to date. This helps in improving the efficiency of healthcare service provided.

Book patient appointments

- **Functionality:** This functionality allows the automatic booking of patient appointment based on the patient severity. The concept is based on priority scheduling of patient's appointment.
- **Technical Implementation:** The patient symptoms are extracted from the doctor notes using **wink-nlp**. A random forest classification model has then been used to classify the patients in three different severity categories namely, 'H', 'M' and 'L', using the symptoms extracted. The patients whose appointments are to be scheduled are placed in a queue. At the end of the day, the doctor can book all appointments by clicking on a button in the appointment page. The appointments will be booked using the greedy algorithm based on priority of the patients and an email will be sent to the patients to notify the date of the appointment. The appointments are then displayed in a calendar implemented using **fullcalendar**.
- **Impact:** The automatic appointment reduces waiting time for high priority patients and makes the booking of appointments more efficient along with email confirmation of the appointment.

Email for appointment and prescription

- **Functionality:** Emails are sent to patients to notify an appointment booked and to provide e-prescription of medicines to the patient.
- **Technical Implementation:** Email.js has been used to send email directly from the frontend. A service and templates are used to send preformatted emails with parameters passed when sending the email. Parameters include the recipient of the email and either the prescribed medicines or the appointment booking.
- **Impact:** E-prescriptions helps reduce medication errors and email confirmation of appointments mitigates the loss of physical appointment cards.

8.2 Development Environment

This section gives an insight about the development environment used when developing SmartCare. This includes the integrated development environment, the tools used for testing, the different frameworks used for the backend and frontend and the version control used.

8.2.1 Integrated Development Environment (IDE)

- **Visual Studio Code (VS Code):** This code editor has been used for the development of SmartCare as it contains features and support which makes development easier and faster. It consists of great tooling for web technologies like React.js, HTML, CSS, JavaScript and Node.js, making it suitable for this project (Microsoft, 2021).

8.2.2 Testing

- **Postman:** Postman has been used for the building and testing of all the APIs implemented. Its comprehensive set of tools help build and test APIs faster to make sure they are working as intended.

8.2.3 Tools and Frameworks

- **React.js:** React.js has been used to implement the frontend of the website. It provides fast and efficient framework for building web applications. It also enables code reusability in terms of components, making it suitable for implementing SmartCare.
- **Node.js and Express:** Node.js and express have been used to implement the backend. Combined, they help to handle server requests effectively and provide appropriate responses from MongoDB or the API implemented.
- **MongoDB Atlas:** This cloud-based storage has been used to enable easy access to patient data from anywhere, making it a crucial part of the project.
- **Flask**: Flask has been used to implement python API to use the model to determine patient severity. Its lightweight, flexible and ease of use makes it suitable for this project.
- **Google collab**: Google collab is a cloud-based Jupyter notebook environment which has been used to train the patient severity classification models.

8.2.4 Version control

- **Git:** Git version control was used to keep track of all the changes during the development phase ensuring smooth implementation of different features along with debugging.

8.3 Project Folder Description

The project is divided into three parts, mainly the server side (server folder), the frontend (client folder) and the python API (model folder). This section describes how the folder of each section has been arranged to facilitate the development of the application.

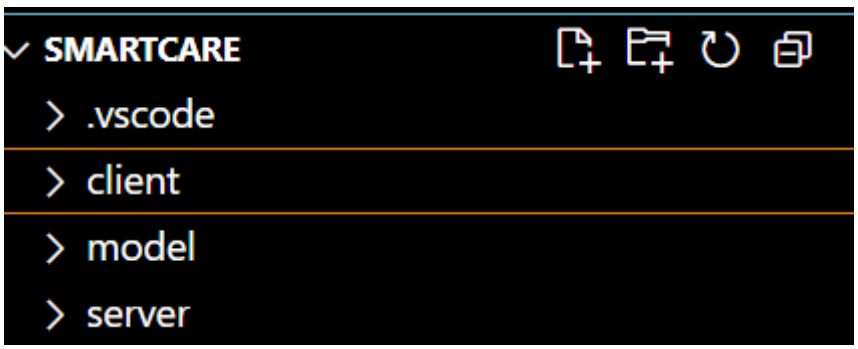


Figure 22 - Project Folder Description

8.3.1 Frontend

The client folder is the React.js frontend application. It consists of the **node modules** folder containing all the node packages being used, the **public folder** where the html page is served from and lastly the **src** folder containing the pages folder, components folder and css folder. All the components used are stored in the components folder and their respective css file is stored in the css folder. The pages folder contains the four main pages of the application mainly the home page, the medical history page, the login page and the calendar. The main app component (app.js) found in the src folder is rendered in the index file (index.js). The snapshot below shows the file arrangement.

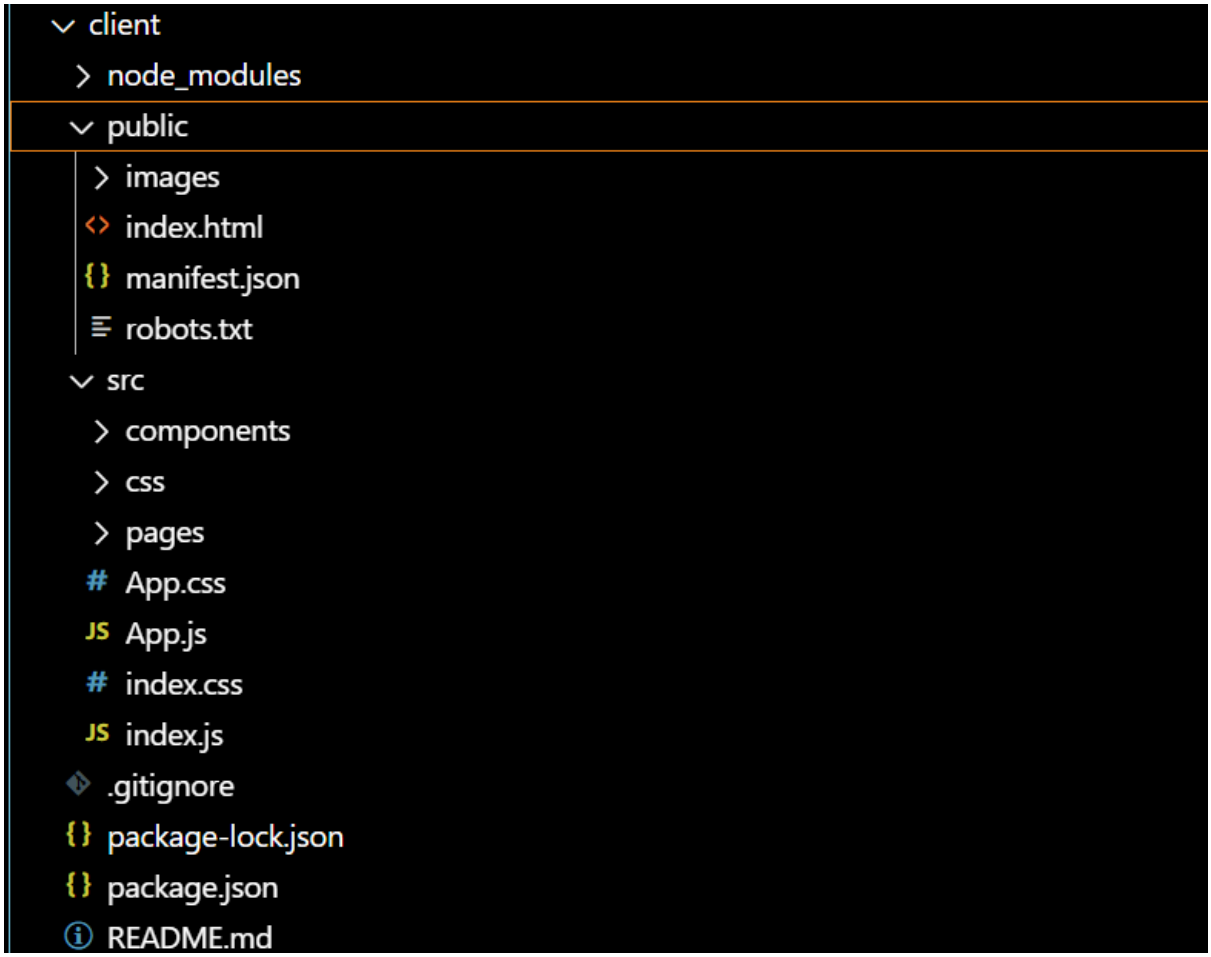


Figure 23 - frontend folder arrangement

8.3.2 Server Side

The server side is the Node.js backend where all the frontend requests are handled. The server folder comprises of three parts. The first part is the **conf** folder where all the properties required to connect to the database are stored. The second part is the **node modules** folder where all the node packages are stored. Finally, the third folder is the static **patientFiles** folder where all the uploaded patient's laboratory or radiology reports are stored. The server folder also comprises of the **server.js file** where all the express

routes are defined to handle the different request to the frontend and to establish the connection to the database.

Figure 24 – Server side folder arrangement

8.3.3 Python API

The model folder stores the **api.py file** where the flask application is defined with a route to use the model. The random forest classification model is also stored in the same folder (**rf_model.pkl**).

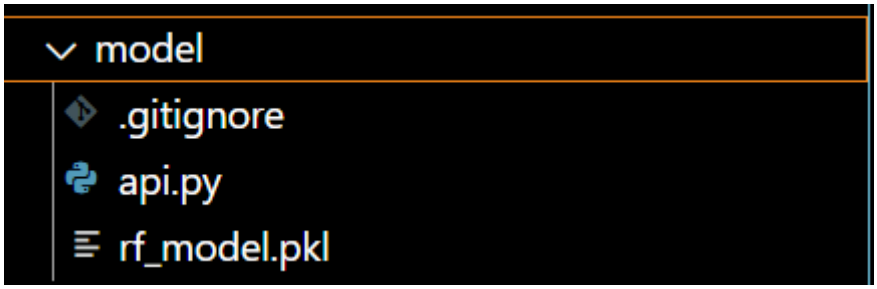


Figure 25 - Python API folder arrangement

8.4 Detailed explanation of functionalities

In this section, the implementation of the functionalities mentioned earlier in the chapter will be discussed in detail. This includes the logical implementation of the functionalities, and the technical approach used to achieve the required feature. Code snippets have been included for better understanding of the feature implementation.

8.4.1 Login

The login consists of two input boxes for the username and password input as shown in the figure below. When the user has filled in the login details and the login button is clicked, the **handleSubmit** function is executed. This function sends a post request to the server to validate the user details. The server queries the database using the function

dbSearch to check if the user exists. An appropriate message is sent back to the frontend to inform whether login has been successful, whether the user does not exist or whether the password input is incorrect. The code snippets below illustrate the implementation.

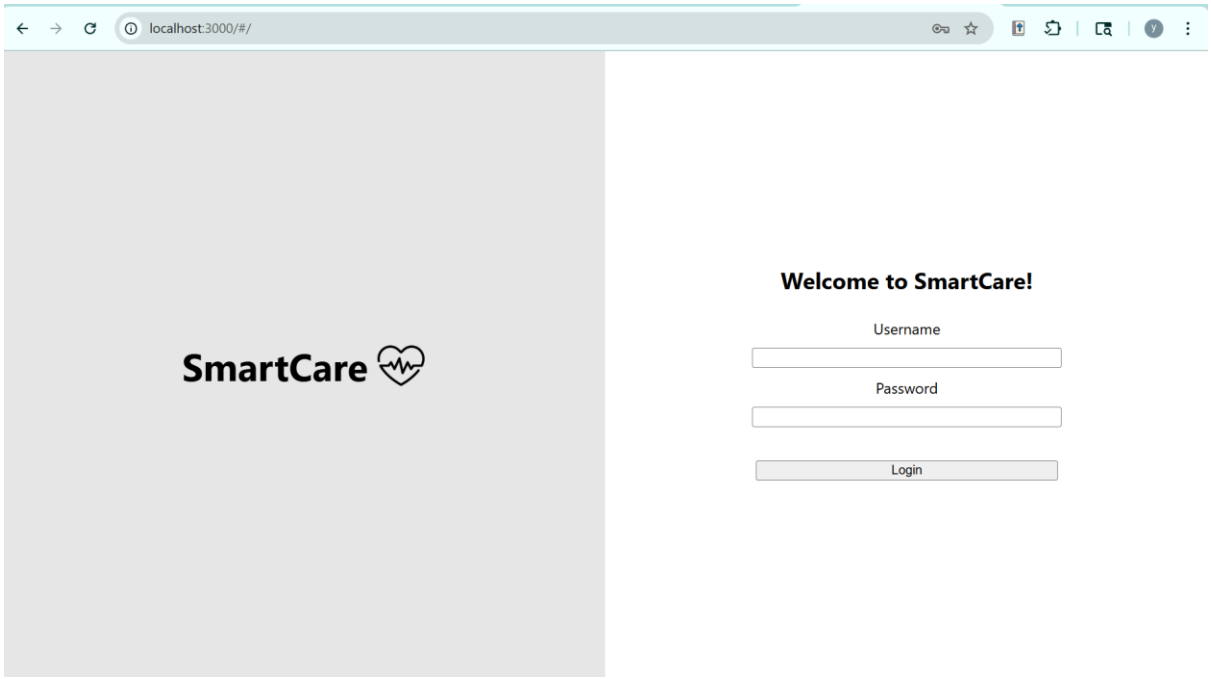


Figure 26 - login page

```
async function handleSubmit(event)  {
    event.preventDefault();
    const response = await fetch("http://localhost:5000/signin/doctors", {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({"username": username, "password": password})
    });
    const result = await response.json();
    if(result.login === true) {
        alert(result.message);
        setUser(result.user);
        navigate('/homePage');
    } else {
        alert(result.message);
    }
}
```

Figure 27 – login frontend handleSubmit function

```
app.post('/signin/:collectionName', async function(req, res) {
    try{
        const password = req.body.password;
        const username = req.body.username;
        const query = {"name" : username};
        let result = await dbSearch(query, req.collection);
        if(result.length == 0) {
            res.json({"login" : false, "message" : "user not found"});
        } else {
            if(password == result[0].password) {
                res.json({"login" : true, "message" : "Successful login", "user" : result[0]});
            } else {
                res.json({"login" : false, "message" : "Wrong password"})
            }
        }
    } catch(err) {
        console.error(err);
        res.json(err);
    }
});
```

Figure 28 - server route to handle signin request

8.4.2 Patient Search

The search functionality consists of an input box and a search button. The search button is only enabled when the input box is not empty. React state, **allPatientSearched**, is used to store all the patient searched in an array and to update the frontend display. The array is mapped into an array of **PatientSearched components** which are then displayed in the frontend. When the search button is clicked, the **search function** is executed. The search function sends a get request to the server to get all the patient with name matching that entered by the user. The server queries the MongoDB database and sends and array of the patients which match the user input. The **setAllPatientSearched function** is then used to update the state which stores all the patients. If there is no match, the message “No records” is displayed. The response is then displayed on the frontend as shown in the figure below. Code snippets which illustrate the state initialisation, the mapping of the state to form PatientSearched components, the PatientSearch component, the HomePage component, the search function and the server route handling the request are also shown below.

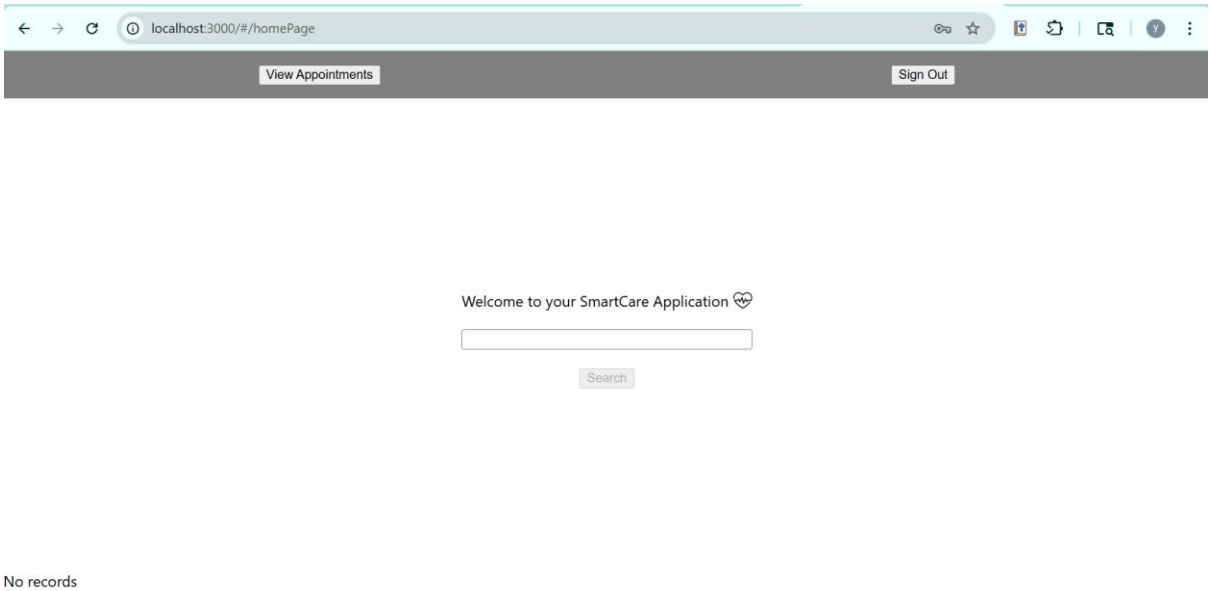


Figure 29 -Homepage with no search results

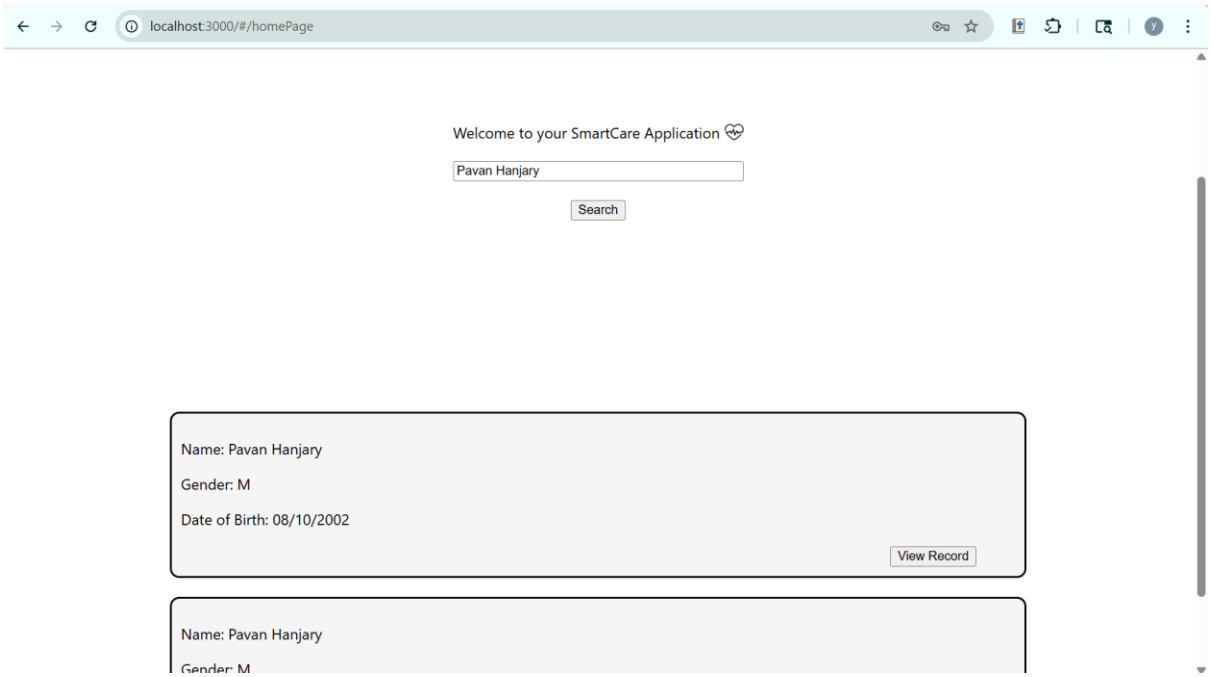


Figure 30 - homepage with search results

```
const [ allPatientSearched, setAllPatientSearched ] = useState([])
```

Figure 31 - State initialisation for patient searched

```
let patientElements = allPatientSearched.map((patient) => {
    return <PatientSearched
                patient={patient}
                key={patient.patientID}
                setPatient={props.setPatient}
            />
})
```

Figure 32 - state mapping to form PatientSearched component

```
import { useNavigate } from "react-router-dom"
import '../css/patientSearched.css'
export default function PatientSearched(props) {
    const navigate  = useNavigate();
    function viewRecord() {
        props.setPatient(props.patient)
        navigate('/medicalHistory')
    }

    return(
        <div className='recordsSearched'>
            <p>Name: {props.patient.name}</p>
            <p>Gender: {props.patient.gender}</p>
            <p>Date of Birth: {props.patient.dateOfBirth}</p>
            <button onClick={viewRecord}>View Record</button>
        </div>
    )
}
```

Figure 33 - PatientSearched component

```
    return(
        <>
            <header className="homePageHeader">
                <button onClick={handleViewAppointmentClick}>View Appointments</button>
                <button>Sign Out</button>
            </header>
            <div className='patientSearch'>
                <div>
                    <span>Welcome to your SmartCare Application</span>
                    <img src='images/logo.png' alt='SmartCare logo'/>
                </div>
                <input type='text' onChange={handleChange}></input>
                {searchBoxValue === "" ? <button disabled>Search</button> :
                <button onClick={search}>Search</button>}
            </div>

            {patientElements.length > 0 ?
            <div className="searchResults">
                {patientElements}
            </div> : <div>No records</div>}
        </>
    )
}
```

Figure 34 - HomePage component

```
  async function search() {
      let query = {"name" : searchBoxValue}
      query = encodeURIComponent(JSON.stringify(query))
      const response = await fetch(`http://localhost:5000/getPatientData/patients/${query}`);
      let patientSearched = await response.json();
      setAllPatientSearched(patientSearched);
  }
```

Figure 35 - search function

```
app.get('/getPatientData/:collectionName/:query', async function(req, res) {
    try {
        const query = JSON.parse(decodeURIComponent(req.params.query));
        var patients = await dbSearch(query, req.collection);
        res.json(patients);
    } catch(err) {
        console.error(error);
        res.json({"message" : "Error when getting data"})
    }
});
```

Figure 36 - Server route to get patient searched

8.4.3 View Patient's medical record

The user can view patient's previous medical history which includes the previous notes of earlier visits, prescription history, radiology reports and laboratory reports. When the view record button is clicked after searching a patient, the **MedicalHistoryPage** is

displayed. This page comprises of the above mentioned four sections. Buttons have been used to change the view of the div with **className records** to view the sections. A state named **navbarOption** has been used to switch between the different sections. The patient whose records are to be displayed is passed as a prop to the component MedicalHistoryPage. The prop is further passed to the components of the four sections namely **PatientHistory**, **MedicationHistory**, **RadiologyHistory** and **LaboratoryHistory**. The four sections are further discussed below. The code snippets below show the MedicalHistoryPage implementation.

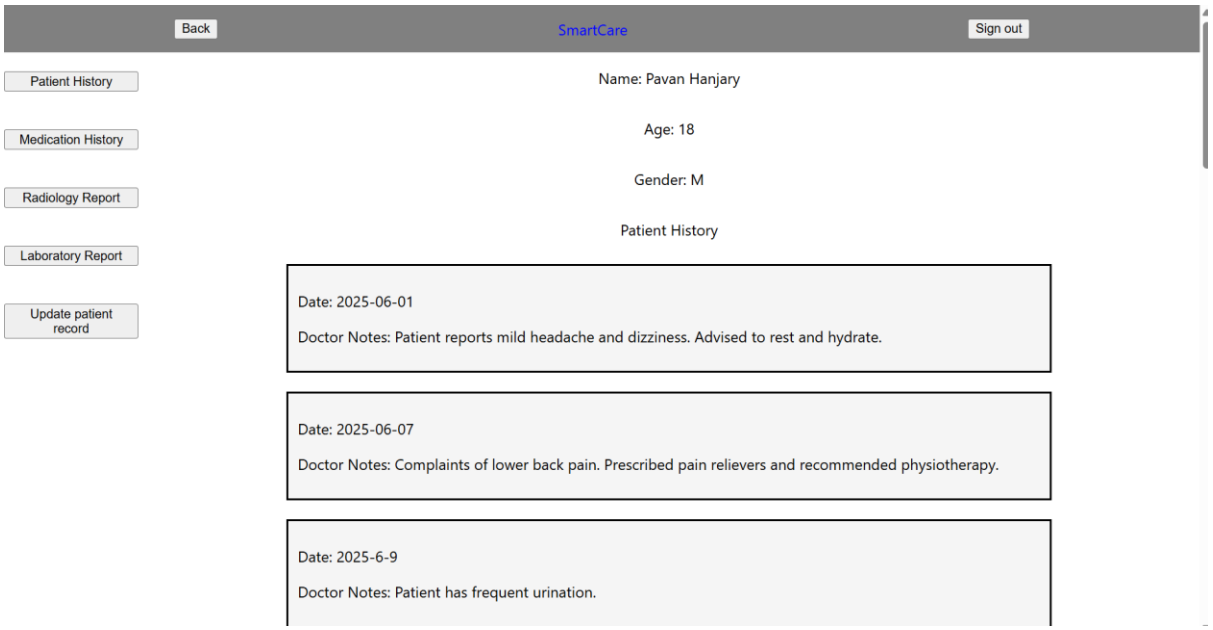


Figure 37 - Medical history page

```
export default function MedicalHistoryPage(props) {
    const navigate = useNavigate();
    const [ navbarOption, setNavbarOption ] = useState('patientHistory')
```

Figure 38 - navbarOption state to change view

```
export default function MedicalHistoryPage(props) {
    return(
        <>
            <header>
                <button onClick={backButtonClick}>Back</button>
                <p>SmartCare</p>
                <button>Sign out</button>
            </header>
            <div className="medicalHistory">
                <div className="medicalHistoryNav">
                    <nav>
                        <button onClick={() => setNavbarOption('patientHistory')}>Patient History</button>
                        <button onClick={() => setNavbarOption('medicationHistory')}>Medication History</button>
                        <button onClick={() => setNavbarOption('radiologyHistory')}>Radiology Report</button>
                        <button onClick={() => setNavbarOption('laboratoryHistory')}>Laboratory Report</button>
                        <button onClick={() => setNavbarOption('updatePatientHistory')}>Update patient record</button>
                    </nav>
                </div>
                <div className="patientDetails">
                    <div className="patientPersonalDetails">
                        <p>Name: {props.patient.name}</p>
                        <p>Age: {props.patient.age}</p>
                        <p>Gender: {props.patient.gender}</p>
                    </div>
                    <div className="records">
                        {navbarOption === 'patientHistory' && <PatientHistory patient={props.patient}/>}
                        {navbarOption === 'medicationHistory' && <MedicationHistory patient={props.patient}/>}
                        {navbarOption === 'radiologyHistory' && <RadiologyHistory patient={props.patient}/>}
                        {navbarOption === 'laboratoryHistory' && <LaboratoryHistory patient={props.patient}/>}
                        {navbarOption === 'updatePatientHistory' && <UpdatePatientHistory patient={props.patient} updateAppointmentList={pr
                    </div>
                </div>
            </div>
```

Figure 39 - MedicalHistoryPage component

8.4.3.1 Patient History

The patient history section displays all previous notes of patient visits in chronological order. The patient prop is used to send a get request to the **/getPatientData** route of the server to obtain the patient's records when the component is first rendered using useEffect. The records are then displayed using the component **PatientMedicalData** comprising of the date of visit and the notes passed as props. The code snippets below show the react useEffect to fetch the data and the use of PatientMedicalData component to display the records.

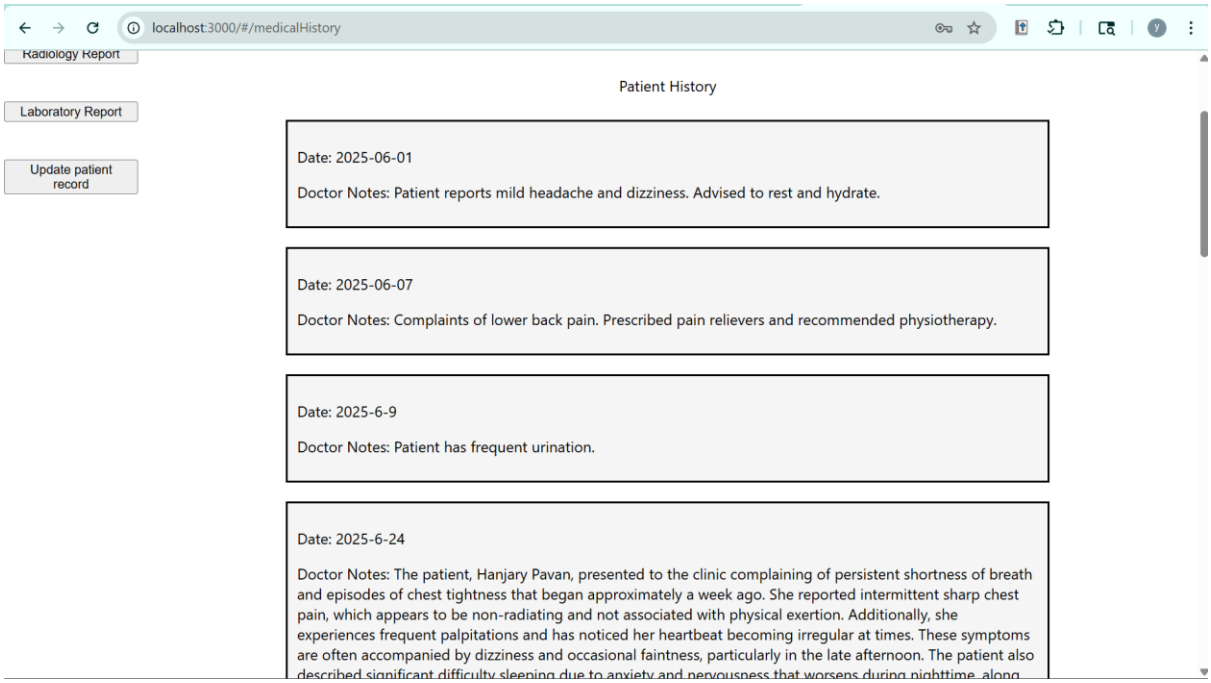


Figure 40 - patient history

```
export default function PatientHistory({patient}) {
    const [ patientData, setPatientData ] = useState([]);
    useEffect(function() {
        // code to fetch patient medical history from mongoDB
        async function fetchData() {
            let query = {"patientID" : patient.patientID};
            query = encodeURIComponent(JSON.stringify(query));
            const response = await fetch(`http://localhost:5000/getPatientData/patientHistory/${query}`);
            let result = await response.json();
            setPatientData(result);
        }
        fetchData();
    }, [patient])

    let patientHistoryElements = patientData.map((data) => {
        return <PatientMedicalData date={data.date} notes={data.note}/>
    });

    return(
        <div className="patientHistoryRecords">
            <p>Patient History</p>
            <div>{patientHistoryElements}</div>
        </div>
    )
}
```

Figure 41 - PatientHistory component with useEffect implementation

```
import '../css/patientMedicalData.css'
export default function PatientMedicalData(props) {
    return(
        <div className='patientHistoryRecordElement'>
            <p>Date: {props.date}</p>
            <p>Doctor Notes: {props.notes}</p>
        </div>
    )
}
```

Figure 42 - PatientMedicalData component

8.4.3.2 Medication History

The medication history section displays all previous prescriptions in chronological order. The patient prop is used to send a get request to the **/getPatientData** route of the server to obtain the patient's prescriptions when the component is first rendered using useEffect. The records are then displayed using the component **MedicationData** comprising of the date of visit, the medicines and their respective duration and dosage. The code snippets below show the react useEffect to fetch the data and the use of MedicationData component to display the records.

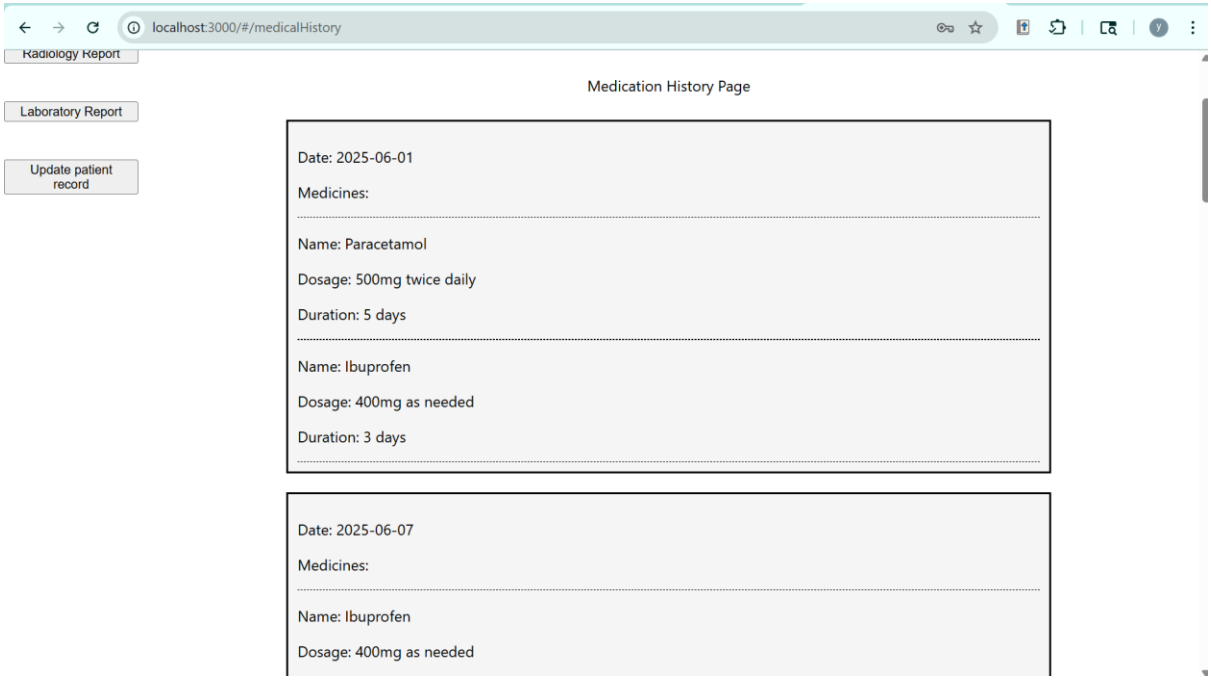


Figure 43 - medication history

```jsx
import { useState, useEffect } from 'react';
import MedicationData from './MedicationData';
import '../css/medicationHistory.css'

export default function MedicationHistory({patient}) {
    const [ patientData, setPatientData ] = useState([]);
    useEffect(function() {
        // code to fetch patient medical history from mongoDB
        async function fetchData() {
            let query = {"patientID" : patient.patientID};
            query = encodeURIComponent(JSON.stringify(query));
            const response = await fetch(`http://localhost:5000/getPatientData/medicationHistory/${query}`);
            let result = await response.json();
            setPatientData(result);
        }
        fetchData();
    }, [patient])

    // mapping patientData to form MedicationData components
    let medicalHistoryElements = patientData.map((data) => {
        return <MedicationData date={data.date} medication={data.medications}/>
    });

    return(
        // display of prescriptions using MedicationComponent
        <div className="medicationHistory">
            <p>Medication History Page</p>
            <div>{medicalHistoryElements}</div>
        </div>
    )
}
```

Figure 44 - MedicationHistory component

```jsx
import '../css/medicationData.css'
export default function MedicationData(props) {
    let medicationElements = props.medication.map((medication) => {
        return(
            <div className="medicineDetails">
                <p>Name: {medication.name}</p>
                <p>Dosage: {medication.dosage}</p>
                <p>Duration: {medication.duration}</p>
            </div>
        )
    })

    return(
        <div className="medicationHistoryElement">
            <p>Date: {props.date}</p>
            <p>Medicines:</p>
            <div>{medicationElements}</div>
        </div>
    )
}
```

Figure 45 - MedicationData component

8.4.3.3 Radiology and Laboratory History

The two functionalities have been implemented in the same way. React's useEffect is used to fetch all the file paths of the patient database by making a get request to the server on the route **/getPatientData**. The file paths received are displayed using the Reports component which displays the date and a view file button for each file record obtained. When the view file button is clicked, the file is displayed on the screen with navigation buttons to go to the next page and previous page. A state name **viewReport** is used to toggle view from viewing all the reports and viewing a single file. The file display functionality has been implemented using **react-pdf**. React-pdf is an NPM package used to display files. The snippets below show the **RadiologyHistory** component, the **Reports** component and the implementation of react-pdf with appropriate description.

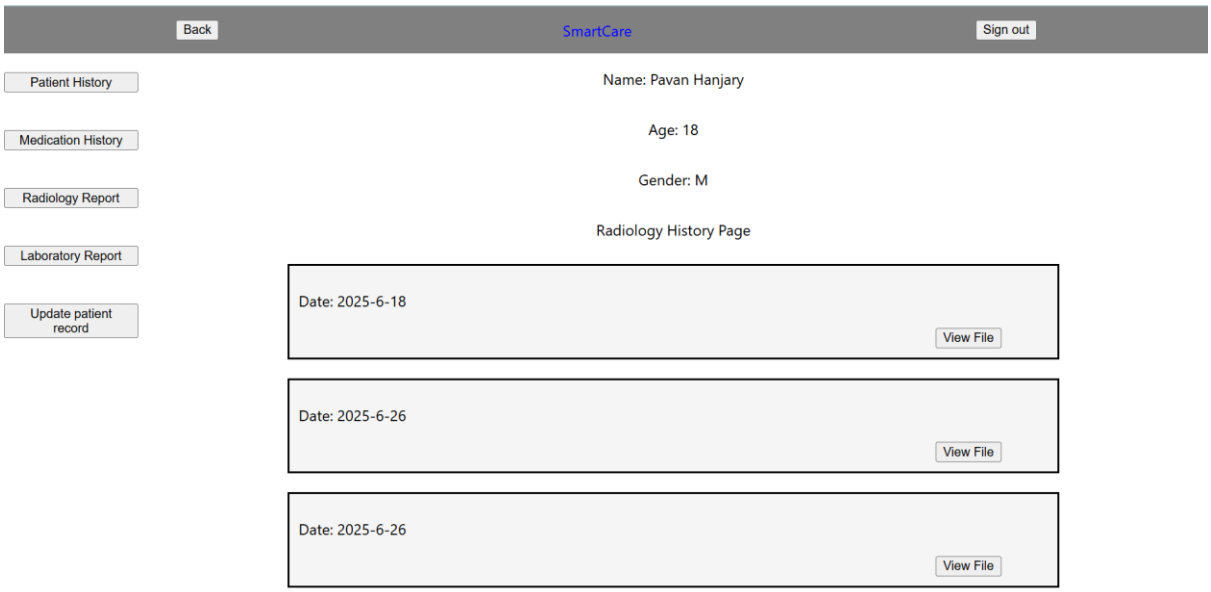


Figure 46 - View radiology reports

```
import { useEffect, useState } from 'react'
import Reports from './Reports';
import '../css/radiologyHistory.css'
import PDFViewer from './PDFViewer'
export default function RadiologyHistory({patient}) {
    const [ patientData, setPatientData ] = useState([]);
    const [ viewReport, setViewReport ] = useState(false);
    const [ currentFilePath, setCurrentFilePath ] = useState("");

    useEffect(function() {
        // useEffect to fetch reports from server
        async function fetchData() {
            let query = {"patientID" : patient.patientID};
            query = encodeURIComponent(JSON.stringify(query));
            const response = await fetch(`http://localhost:5000/getPatientData/radiologyReports/${query}`);
            let result = await response.json();
            setPatientData(result);
        }
        fetchData();
    }, [patient])
```

Figure 47 - RadiologyHistory useEffect to fetch reports

```
let radiologyResultElements = patientData.map((data) => {
    return <Reports date={data.date} filePath={data.file} setFilePath={setFilePath}/>
});

// display either all reports or display one report when view report button is clicked
return(viewReport ?
    <div>
        <button onClick={handleBackClick}>Back</button>
        <PDFViewer filePath={currentFilePath}/>
    </div> :
    <div className="radiologyHistory">
        <p>Radiology History Page</p>
        <div>
            {radiologyResultElements}
        </div>
    </div>
)
```

Figure 48 - View reports display in RadiologyHistory component

```
import '../css/reports.css'
export default function Reports(props) {
    function viewReport() {
        props.setFilePath(props.filePath);
    }

    return(
        <div className="reportElement">
            <p>Date: {props.date}</p>
            <p>FilePath: {props.filePath}</p>
            <button onClick={viewReport}>View File</button>
        </div>
    )
}
```

Figure 49 - Reports component

The code snippets below show the use of react-pdf to view files.

This line of code is used to set the set the location of the PDF.js web worker which parses and render pdf files.

```
export default function PDFViewer(props) {
    pdfjs.GlobalWorkerOptions.workerSrc = new URL(
    'pdfjs-dist/build/pdf.worker.min.mjs',
    import.meta.url,
    ).toString();
```

Figure 50 - setting up web worker for react-pdf

The image below shows the functions prevButtonClick and nextButtonClick which are used to change page number. The document component of react-pdf is used to display the pdf on the website. It takes as parameter the file path of the file to be rendered and a function onLoadSuccess to set the number of pages of the pdf once it is loaded.

```
    function onDocumentLoadSuccess({ numPages }) {
        setNumPages(numPages);
    }

    function prevButtonClick() {
        setPageNumber(function(prevPageNumber) {
            return prevPageNumber - 1;
        })
    }

    function nextButtonClick() {
        setPageNumber(function(prevPageNumber) {
            return prevPageNumber + 1;
        })
    }
    return(
        <div>
            <button onClick={prevButtonClick} disabled={pageNumber <= 1}>Prev</button>
            <button onClick={nextButtonClick} disabled={pageNumber >= numPages}>Next</button>
            <Document file={props.filePath} onLoadSuccess={onDocumentLoadSuccess}>
              <Page pageNumber={pageNumber} renderTextLayer={false} renderAnnotationLayer={false}/>
            </Document>
            <p>
              Page {pageNumber} of {numPages}
            </p>
        </div>
    )
}
```

Figure 51 - Implementation of react-pdf components to view files

8.4.4 Update Patient's medical records

The update patient record functionality is divided into three parts. This constitutes of the upload patient history, upload prescription and upload report sections. A state **navbarOption** is used to switch to a specific section when the respective button from the navbar is clicked. The code snippet below shows the **updatePatientHistory** component.

```jsx
import { useState } from 'react'
import UploadPatientHistory from './UploadPatientHistory';
import UploadPrescription from './UploadPrescription';
import UploadReport from './UploadReport';
import '../css/updatePatientHistory.css'

export default function UpdatePatientHistory(props) {
    const [ navbarOption, setNavbarOption ] = useState('uploadPatientHistory');
    return(
        <div className="updateHistory">
            <nav className="updateHistoryNav">
                <button onClick={() => setNavbarOption('uploadPatientHistory')}>Upload New History</button>
                <button onClick={() => setNavbarOption('uploadPrescriptions')}>Upload Prescriptions</button>
                <button onClick={() => setNavbarOption('uploadReport')}>Upload Report</button>
            </nav>
            <div className="updateHistoryDisplay">
                {navbarOption === 'uploadPatientHistory' && <UploadPatientHistory patient={props.patient}
                updateAppointmentList={props.updateAppointmentList}/>}
                {navbarOption === 'uploadPrescriptions' && <UploadPrescription patient={props.patient}/>}
                {navbarOption === 'uploadReport' && <UploadReport patient={props.patient}/>}
            </div>
        </div>
    )
}
```

Figure 52 – UpdatePatientHistoryComponent

Upload Patient History

This feature contains an input box for user to input notes about patient consultation. The input is stored in a state called **notes** which updates on every input change in the textbox. Upon completing the notes, the upload button can be clicked to save the new patient notes to the database. A post request is sent to the **/uploadNotes** route of the server with the notes, the date and the patient ID as body. On the server side, the new record is then inserted into the MongoDB database. On successful upload, a message is displayed in the frontend. The code snippets below illustrate the description.

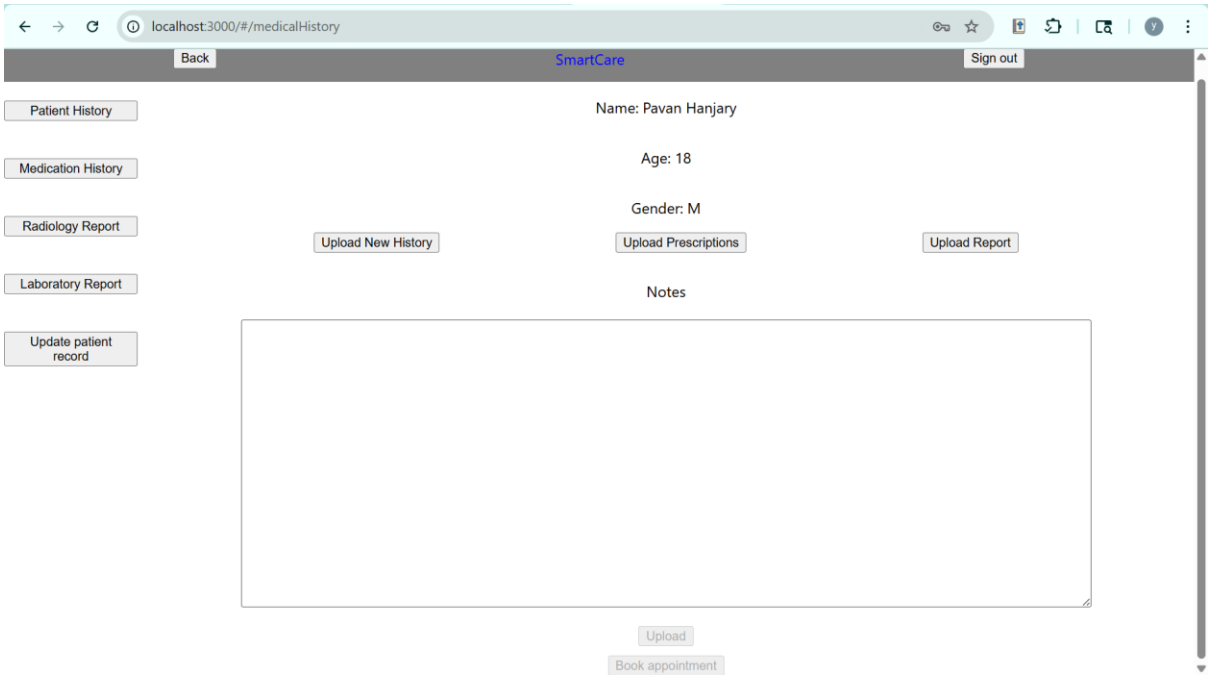


Figure 53 - Update patient history

```
return(
    <div className="uploadHistory">
        <form onSubmit={handleSubmit}>
            <label htmlFor="notes">Notes</label>
            <textarea onChange={handleNoteChange} type="text" id="notes" name="notes" required></textarea>
            {notes === "" ? <button disabled>Upload</button> : <button>Upload</button>}
        </form>
        {notes === "" ? <button disabled>Book appointment</button>: <button onClick={bookAppointment}>Book appointment</button>}
    </div>
)
}
```

Figure 54 - upload patient notes

```
async function handleSubmit(event) {
    event.preventDefault();
    let date = formatDate(new Date());
    let patientData = {"patientID" : patient.patientID, "date" : date, "note": notes};
    const response = await fetch('http://localhost:5000/uploadNotes/patientHistory', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(patientData)
    });
    const result = await response.json();
    alert(result.message);
}
```

Figure 55 - function for upload button when uploading notes

```
app.post('/uploadNotes/:collectionName', async function(req, res) {
    try{
        var patientData = req.body;
        const result = await req.collection.insertOne(patientData);
        console.log(result);
        res.json({"message" : "Document successfully uploaded!"})
    } catch(err) {
        console.error(err);
        res.json({"message" : "Error when uploading document"})
    }
});
```

Figure 56 - Server route to upload notes

Upload Prescription

This feature contains a form for user to input medicine name, dosage and duration for a new prescription. When the add button is clicked the new medicine details are added to a state called prescription and is displayed in table format on the screen. When all the medicines have been added to the prescription, the upload prescription must be clicked. This sends a post request to the /uploadPrescription route of the server to save the prescription to the database. Upon successful upload, an appropriate message is displayed. The snapshots below illustrate the description.

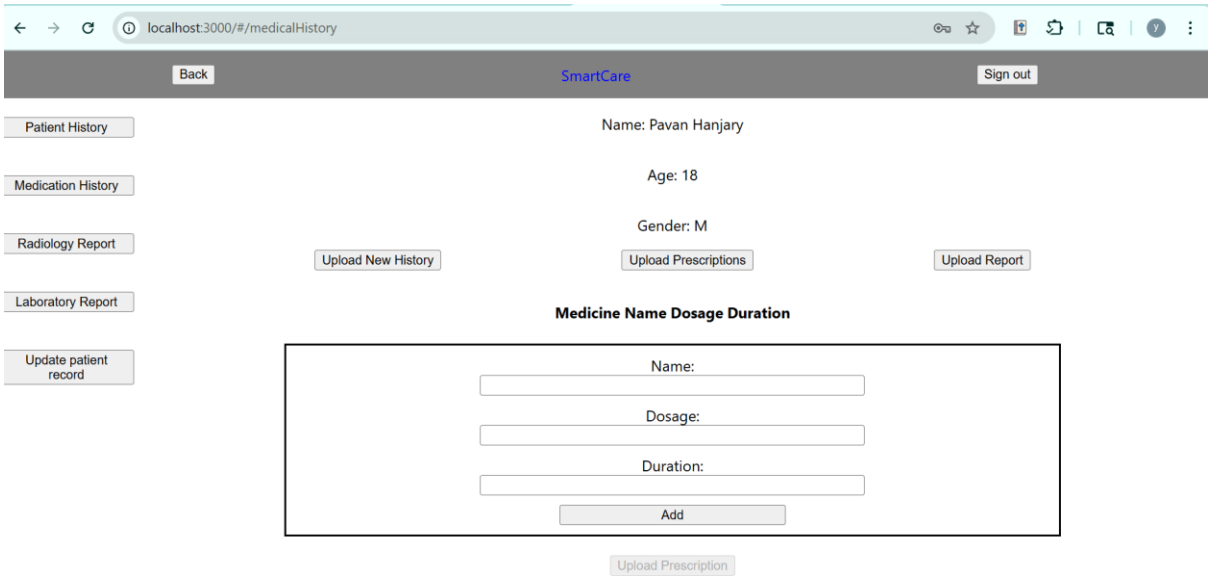


Figure 57 - Upload prescription page

```
return(
    <div className="uploadPrescription">
        <div className="prescription">
            <table>
                <thead>
                    <tr>
                        <th>Medicine Name</th>
                        <th>Dosage</th>
                        <th>Duration</th>
                    </tr>
                </thead>
                <tbody>
                    {prescriptionElements}
                </tbody>
            </table>
        </div>
        <form onSubmit={handleSubmit}>
            <label>Name:</label>
            <input type="text" id="name" name="name" onChange={handleNameChange} required/>
            <label>Dosage:</label>
            <input type="text" id="dosage" name="dosage" onChange={handleDosageChange} required/>
            <label>Duration:</label>
            <input type="text" id="duration" name="duration" onChange={handleDurationChange} required/>
            <button>Add</button>
        </form>
        {prescription.length === 0 ? <button disabled>Upload Prescription</button>:
        <button onClick={uploadPrescription}>Upload Prescription</button>}
    </div>
)
```

Figure 58 - upload prescription frontend display

```
async function uploadPrescription() {
    let date = formatDate(new Date());
    let patientData = {"patientID" : patient.patientID, "date" : date, "medications": prescription};
    const response = await fetch('http://localhost:5000/uploadPrescription/medicationHistory', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(patientData)
    });
    const result = await response.json()
    sendPrescriptionEmail(patient.name, patient.email, prescription, date);
    alert(result.message);
}
```

Figure 59 - function to upload prescription

```
app.post('/uploadPrescription/:collectionName', async function(req, res) {
    try{
        var prescription = req.body;
        const result = await req.collection.insertOne(prescription);
        console.log(result);
        res.json({"message" : "Prescription uploaded successfully!"});
    } catch(err) {
        console.error(err);
        res.json({"message" : "Failed to upload prescription"});
    }
});
```

Figure 60 - server route for upload prescription

Upload report

The upload report functionality consists of two inputs. The first one is the file input, and the second one is a radio button to choose whether the file being input is a radiology or a laboratory report. The file and the file type input are stored in react state namely **file** and **option**. When the upload button is clicked, a post request is sent to the **/uploadFile** route of the server along with a form data as body containing the file, the date and the patient data. On the server side, the npm package multer is used to store the files in the **/patientFiles** folder initialised by the multer disk storage. The folder is made static so that files can be accessed using their path. The path to the file is the stored in MongoDB. The code snippets below illustrate this description.

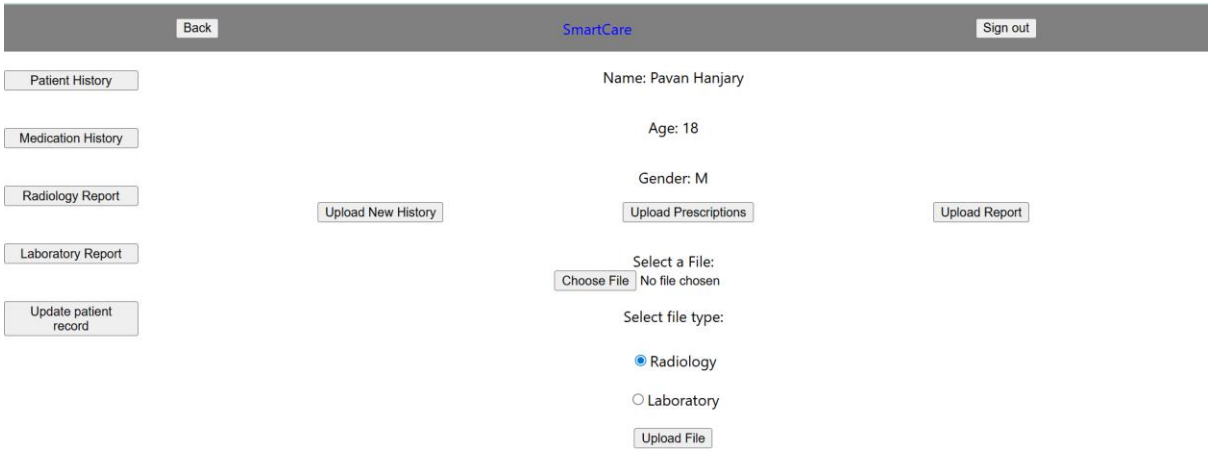


Figure 61 - upload new report

```
return(
    <div className="uploadReport">
        <form onSubmit={handleSubmit}>
            <label className="fileInput" htmlFor="testFile">Select a File:</label>
            <input onChange={handleChange} type="file" accept="application/pdf"
            id="testFile" name="testFile" required></input>
            <p>Select file type:</p>
            <label className="reportType">
                <input type="radio" name="reportType" value="radiology"
                checked={option === 'radiology'} onChange={handleOptionChange}/>
                Radiology
            </label>
            <label className="reportType">
                <input type="radio" name="reportType" value="laboratory"
                checked={option === 'laboratory'} onChange={handleOptionChange} />
                Laboratory
            </label>
            <button>Upload File</button>
        </form>
    </div>
)
}
```

Figure 62 - Upload file frontend

```
export default function UploadReport({patient}) {
    const [ file, setFile ] = useState("");
    const [ option, setOption ] = useState('radiology')
    async function handleSubmit(event) {
        event.preventDefault();
        const formData = new FormData();
        let date = formatDate(new Date());
        formData.append('file', file);
        formData.append('date', date);
        formData.append('patient', JSON.stringify(patient));
        let fetchUrl = '';
        if(option === "radiology") {
            fetchUrl = 'http://localhost:5000/uploadFile/radiologyReports';
        } else {
            fetchUrl = 'http://localhost:5000/uploadFile/laboratoryReports';
        }
        const response = await fetch(fetchUrl, {
            method: 'POST',
            body: formData
        });

        const uploadResponse = await response.json();
        alert(uploadResponse.message);
        event.target.reset();
    }
```

Figure 63 - function to upload file

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'patientFiles/')
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
    cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname))
  }
})
```

Figure 64 - multer disk storage definition

```
app.post('/uploadFile/:collectionName', upload.single('file'), async function(req, res) {
    try {
        let date = req.body.date;
        let patientID = JSON.parse(req.body.patient).patientID;
        let filePath = `http://localhost:5000/patientFiles/${req.file.filename}`
        const result = await req.collection.insertOne(
            {"patientID" : patientID, "date": date, "file" : filePath}
        );
        console.log(result);
        res.json({"message" : "file uploaded successfully!"})
    } catch(err) {
        console.error(err);
        res.json({"message" : "Error when uploading file!"})
    }
});
```

Figure 65 - file upload server route

8.4.5 Patient severity classification model training

Two models have been trained, namely the random forest model and the k-nearest neighbour model, based on the dataset accessible on the link given below. The dataset is a free dataset which contains 773 unique diseases and 377 one hot encoded symptoms for each disease. The dataset contains 246,000 samples. However, for each disease, a severity level has been manually assigned before being able to train the classification models for prediction of patient severity levels. After assigning a severity level to each disease, the diseases column has been removed from the dataset and only the one hot encodings of the symptoms and the severity levels were used for model training. Missing values were then dropped from the dataset to obtain a clean dataset ready for training. The cleaned dataset was then split into train and test data with the training data consisting of 80% of the data and the test data consisting of 20% of the data. The training set was used to train the model using sklearn. The test set was then used to evaluate the model. The snapshots show the google collab notebook of the training of the models.

Link to dataset: https://www.kaggle.com/datasets/dhivyeshrk/diseases-and-symptoms-dataset

```
# matching severity from severity data to symptoms data
priority_map = dict(zip(priorityData['diseases'], priorityData['Priority']))
symptomsData['severity'] = symptomsData['diseases'].map(priority_map)
```

Figure 66 - mapping severity to diseases

```
[ ]  # removing missing values and dropping unecessary column
     symptomsData['severity'] = symptomsData['severity'].str.strip()
     symptomsData['severity'].replace('', np.nan, inplace=True)
     symptomsData['severity'].replace('?', np.nan, inplace=True)
     symptomsData = symptomsData.dropna(subset=['severity'])
     symptomsData = symptomsData.drop(columns=['diseases'])
```

Figure 67 - dropping null values in dataset

```
# RandomForestClassifier Training
X = symptomsData.iloc[:, :-1]
y = symptomsData.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
rf = RandomForestClassifier(n_estimators=300, criterion='entropy', min_samples_split=7)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

Figure 68 - splitting train and test data + training of random forest model

```python
# K-nearest neighbour classifier
neigh = KNeighborsClassifier(n_neighbors=7, weights='uniform')
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
print(classification_report(y_test, y_pred))
```

Figure 69 - training KNN classifier

8.4.6 Booking of patient's appointment

The booking of patient appointment functionality is divided into two parts. The first part involves the adding of the patient in an array of patients who have to book an appointment. This is carried out when the doctor has written a new patient note on the **UploadPatientHistory** part of the website as shown in the figure below. The user then clicks on the book appointment button to execute the **bookAppointmentFunction**. A post request is then sent to the **/getPatientSeverity** route of the server with the notes as the body of the request. On the server side, the symptoms are extracted from the notes using **wink-nlp** and an array of symptoms. **Wink-nlp** is used to get the lemmatised version of the notes before performing string matching with the array of symptoms. The extracted symptoms are then used to send a request to the **python flask API** to get the severity of the patient using the **random forest classification** model. The patient severity is sent back as a response to the frontend where it is added to the array of patients whose appointments need to be booked. The code snippets below illustrate the first part of appointment booking.

```javascript
async function bookAppointment() {
    // POST request to get patient severity
    const response = await fetch("http://localhost:5000/getPatientSeverity", {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({"notes": notes})
    });
    let patientSeverity = await response.json();
    console.log(patientSeverity);
    let appointment = {patient: patient, severity: patientSeverity.severity}
    updateAppointmentList(appointment);
    alert("Patient added to appointment list");
}
```

Figure 70 - bookAppointment function in UpdatePatientHistory component

```
app.post('/getPatientSeverity', async function(req, res) {
    try {
        // Use of wink-nlp to lemmatise notes
        const { notes } = req.body;
        console.log(notes);
        const doc = nlp.readDoc(notes);
        var notesLemma = doc.tokens().filter(t => t.out(its.type) === 'word').out(its.lemma);
        notesLemma = notesLemma.join(' ');
        // extraction of symptoms from lemmatised notes
        var result = symptoms.map((symptom) => {
            var symptomDoc = nlp.readDoc(symptom);
            var symptomsLemma = symptomDoc.tokens().out(its.lemma).join(' ');
            return notesLemma.includes(symptomsLemma) ? 1 : 0;
        });
        // request to flask API to get severity of patient
        const response = await fetch('http://localhost:5001/getPatientSeverity', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({'symptoms' : result})
        });
        const patientSeverity = await response.json();
        console.log(patientSeverity);
        res.json(patientSeverity);
    } catch(err) {
        console.error(err);
        res.json(err);
    }
});
```

Figure 71 - getPatientSeverity route on server

```
// Use of wink-nlp to lemmatise notes
const { notes } = req.body;
console.log(notes);
const doc = nlp.readDoc(notes);
var notesLemma = doc.tokens().filter(t => t.out(its.type) === 'word').out(its.lemma);
notesLemma = notesLemma.join(' ');
```

Figure 72 - Wink-nlp to lemmatise notes

```
// extraction of symptoms from lemmatised notes
var result = symptoms.map((symptom) => {
    var symptomDoc = nlp.readDoc(symptom);
    var symptomsLemma = symptomDoc.tokens().out(its.lemma).join(' ');
    return notesLemma.includes(symptomsLemma) ? 1 : 0;
});
```

Figure 73 - symptoms extraction from lemmatised notes

```
    // request to flask API to get severity of patient
    const response = await fetch('http://localhost:5001/getPatientSeverity', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({'symptoms' : result})
    });
    const patientSeverity = await response.json();
    console.log(patientSeverity);
    res.json(patientSeverity);
} catch(err) {
    console.error(err);
    res.json(err);
}
```

Figure 74 - request to flask api to get patient severity

```
import json
from flask import Flask, jsonify, request
import joblib
import numpy as np
model = joblib.load('./rf_model.pkl')

app = Flask(__name__)
@app.route('/getPatientSeverity', methods=['POST'])
def getPatientSeverity():
    try:
        data = request.get_json()
        patientSymptoms = data.get('symptoms', [])
        prediction = model.predict(np.array([patientSymptoms]))
        return jsonify({"severity": prediction[0]})
    except Exception as e:
        print(e)
        return jsonify({"error" : "error"})

if __name__ == '__main__':
    app.run(port=5001)
```

Figure 75 - Flask API implementation

The second part of the appointment booking is the booking of all patient appointment in the array of patients with their severity. This can be carried out in the appointment page where the calendar is displayed. When the book all appointments button is clicked, the greedy algorithm is used to book appointments. In other words, the array of patients is sorted in order form 'H' to 'L'. The calendar of the user is then looped to book empty slots in order of priority. The new schedule is then updated to the MongoDB collection by sending a PUT request to the server.

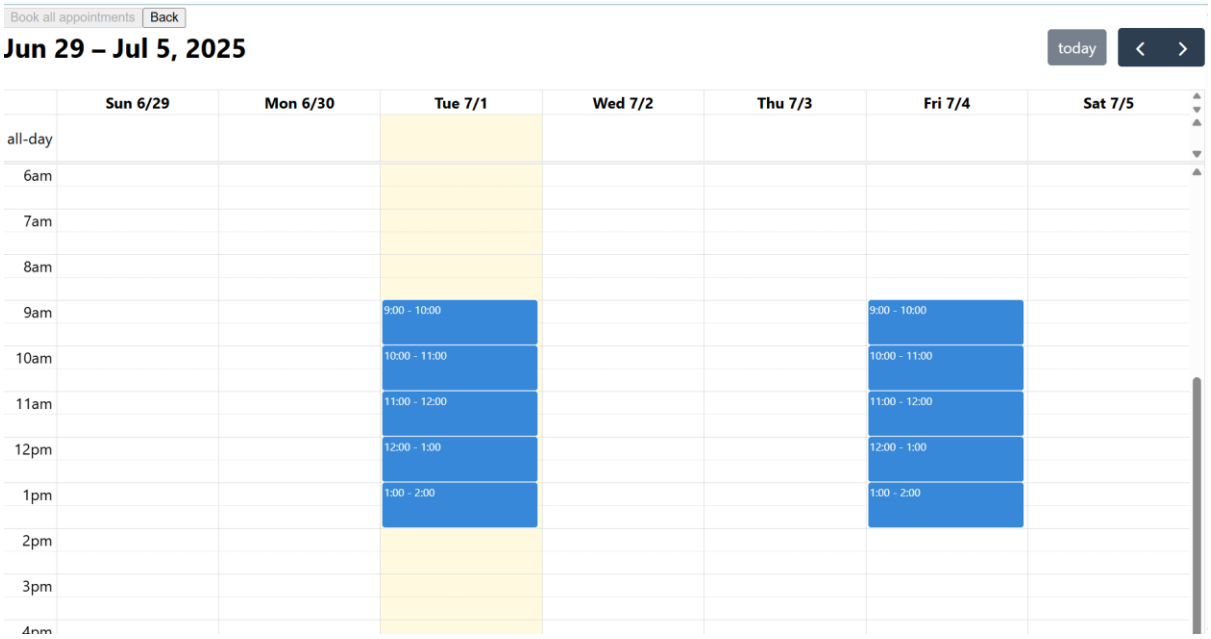


Figure 76 - calendar and book all appointments

```
async function bookAllAppointments(){
    const priorityOrder = { H: 1, M: 2, L: 3 };
    appointmentList.sort((a, b) => priorityOrder[a.severity] - priorityOrder[b.severity]);
    let appointmentCopy = appointments;
    for(let i = 0; i < appointmentList.length;i++) {
        for(let j = 0; j < appointmentCopy.length; j++) {
            if(appointmentCopy[j].title === '') {
                appointmentCopy[j].title = appointmentList[i].patient.name;
                sendAppointmentEmail(appointmentList[i].patient.name, appointmentCopy[j].start,
                    appointmentList[i].patient.email);
                break;
            }
        }
    }
```

Figure 77 - book appointment logic

8.4.7 Email confirmation

Email confirmation feature has been implemented using email.js. Email.js is a JavaScript library which enables to send email directly from the frontend. An email service has been created along with two email templates. One template is for sending prescriptions to patients and the other template is used to send appointment bookings to the patient. The email parameters to be replaced in the email template is passed as template parameters in the frontend. The snapshots below show the code to send emails in the frontend and the email templates created.

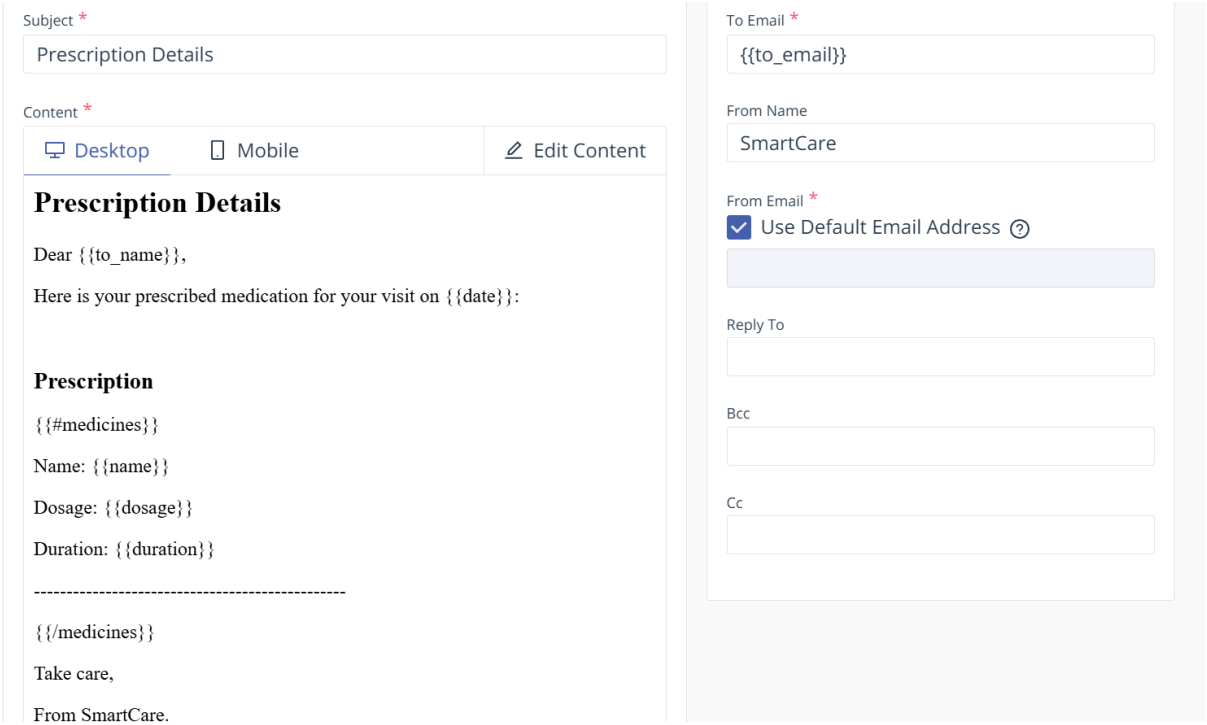


Figure 78 - Prescription email template

```
function sendPrescriptionEmail(name, patientEmail, prescription, date) {
    const templateParams = {
        to_email: patientEmail,
        to_name: name,
        date: date,
        medicines: prescription
    }

    emailjs
    .send('service_p6cyfms', 'template_qurjmyq', templateParams, '_Ulwx2cxo7Nbw1ce8')
    .then(
      () => {
        alert('Prescription email sent successfully');
      },
      (error) => {
        alert('Failed to send prescription email' + error.text);
      }
    );
}
```

Figure 79 - function to send prescription email

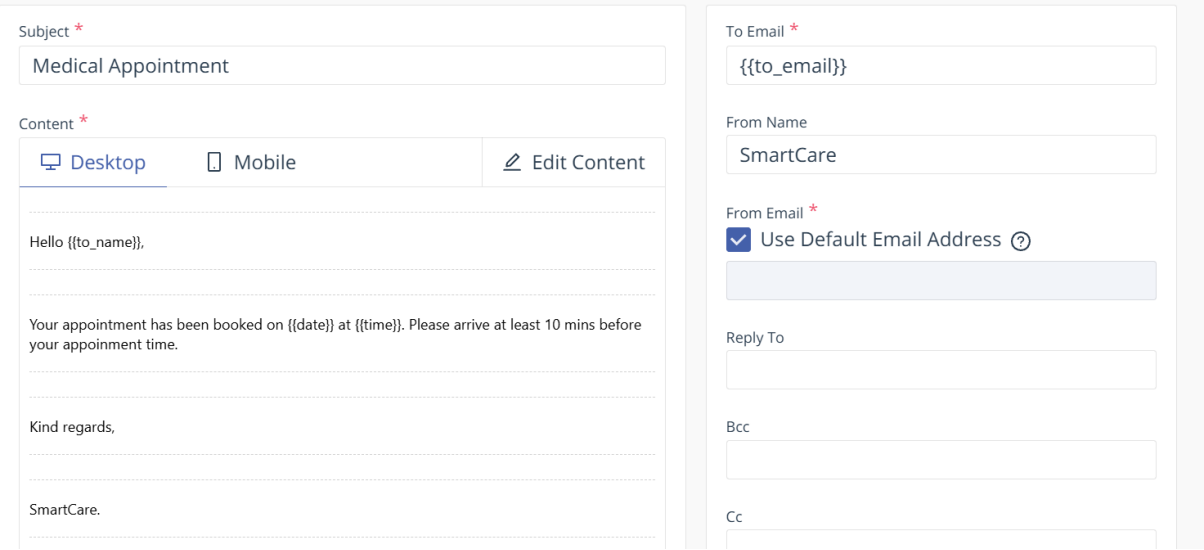


Figure 80 - Template for appointment email

```
function sendAppointmentEmail(name, dateTime, patientEmail) {
    const [ date, time ] = dateTime.split("T");
    const templateParams = {
        to_email: patientEmail,
        to_name: name,
        date: date,
        time: time
    }

    emailjs
    .send('service_p6cyfms', 'template_1ze6zpb', templateParams, '_Ulwx2cxo7Nbw1ce8')
    .then(
      () => {
        alert('Appointment email sent sucessfully');
      },
      (error) => {
        console.log('Failed to sent appointment email' + error.text);
      }
    );
}
```

Figure 81 - function to send appointment emails

9 Chapter 5 – Testing

Testing is an important aspect in software development life cycle. It involves validation and verification with the aim to test if all the technical requirements described in design and development phase are working efficiently and effectively (GeeksforGeeks, 2017). In this section the application will be tested in three different ways, mainly unit testing, integration testing and feature testing.

9.1 Unit testing

Unit testing is the testing of the smallest piece of code which can be logically isolated in the application (smartbear.com, 2025). This section will involve testing of the server

routes using postman and testing of the model API using postman. These tests will make sure that each component is functioning individually as expected while returning the appropriate responses.

9.1.1 Server-side testing

Postman has been used to test the different express routes to make sure the expected output is received. The snapshots below show the different postman snapshots and description.

Get patient data

The **/getPatientData** route is used to retrieve patient history from any collection of the database passed as a parameter using the query which is also passed as a parameter to the route. The snapshots below show a request to this path along with the response of all patient data with patientID of 1.

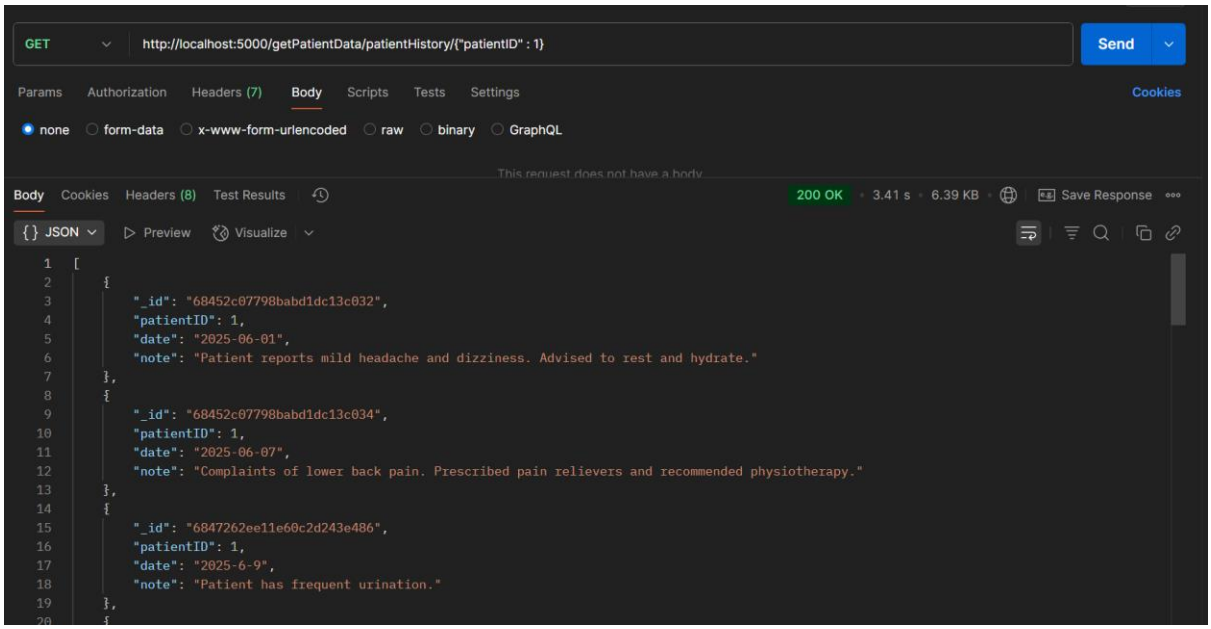


Figure 82 - getPatientData postman test

Upload file

The /uploadFile path is used to upload files to the server. The file is passed as form data to the body of the request. The snapshots below show the postman request for saving a file to the server with its response. The file was saved to the folder patientFiles and added successfully to the MongoDB collection.

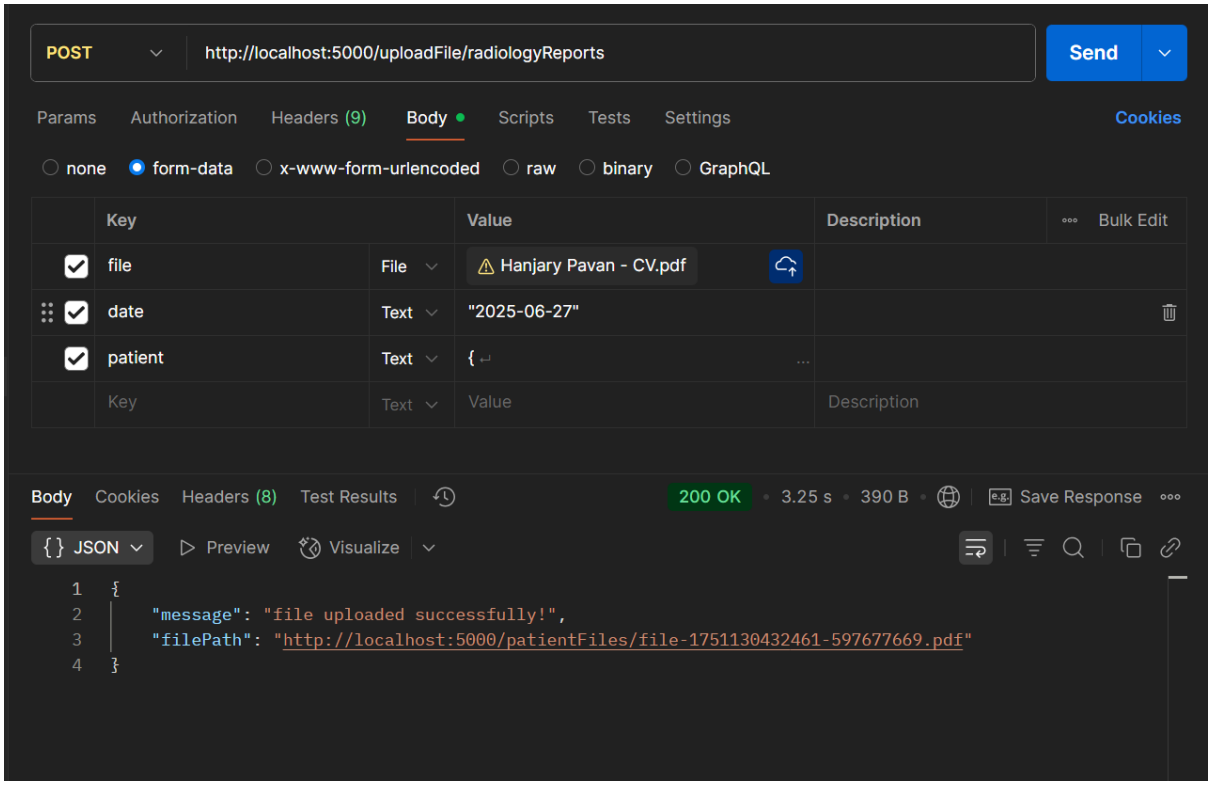


Figure 83 - uploadFile postman test

Figure 84 - file saved in database from postman request

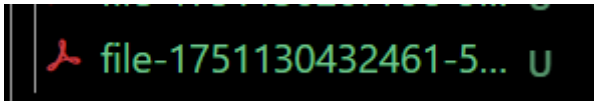


Figure 85 - file saved on server from postman request

Upload notes

The **/uploadNotes** route uploads new patient notes to the MongoDB collection. The note is sent as a body to the request. The snapshots below shows the test with the updated MongoDB collection.

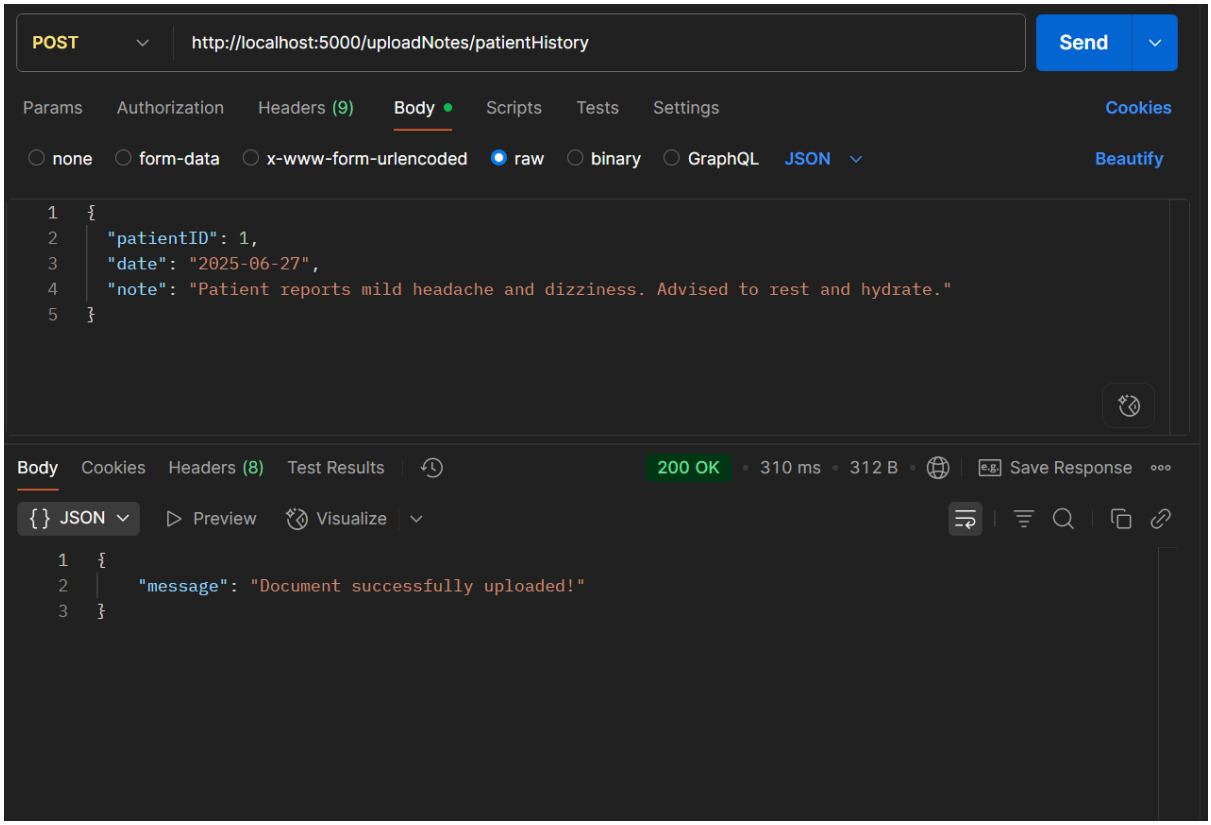


Figure 86 - uploadNotes postman request

Figure 87 - notes uploaded to MongoDB collection

Upload prescription

The **/uploadPrescription** route uploads new prescription to the MongoDB collection. The prescription is sent as a body to the request. The snapshots below shows the test with the updated MongoDB collection.

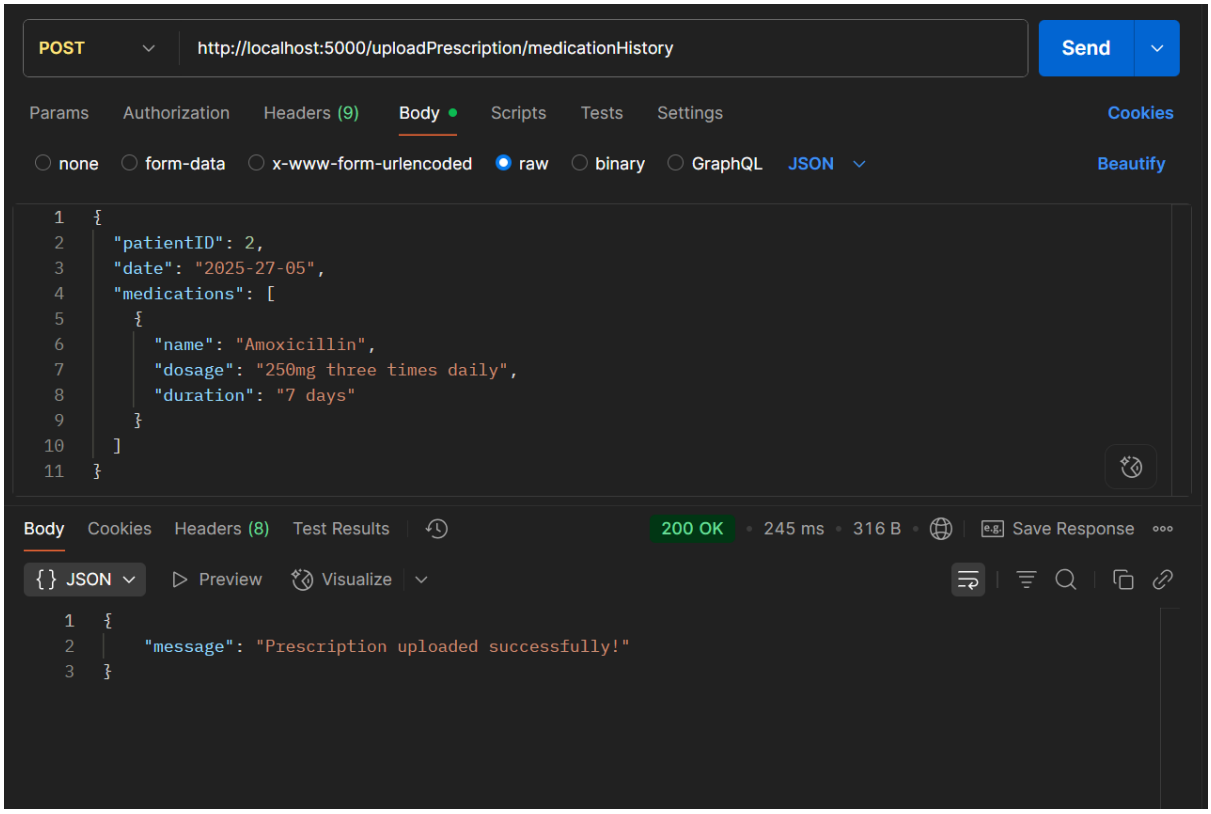


Figure 88 - prescription upload postman test

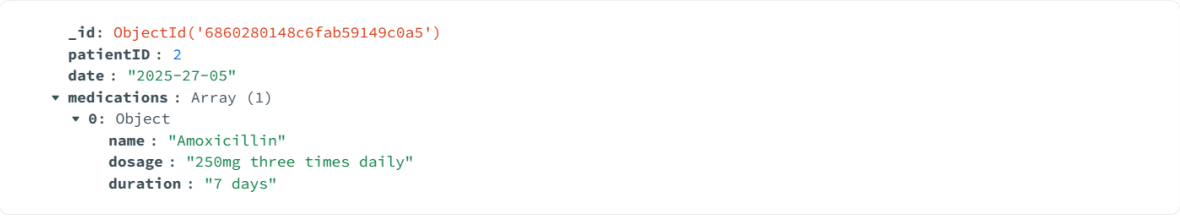


Figure 89 - MongoDB collection update for prescription upload

Sign in

The /signin route is used to validate user's login credentials. The username and password are sent as a request. The snapshots below shows the responses for different username and password verification.

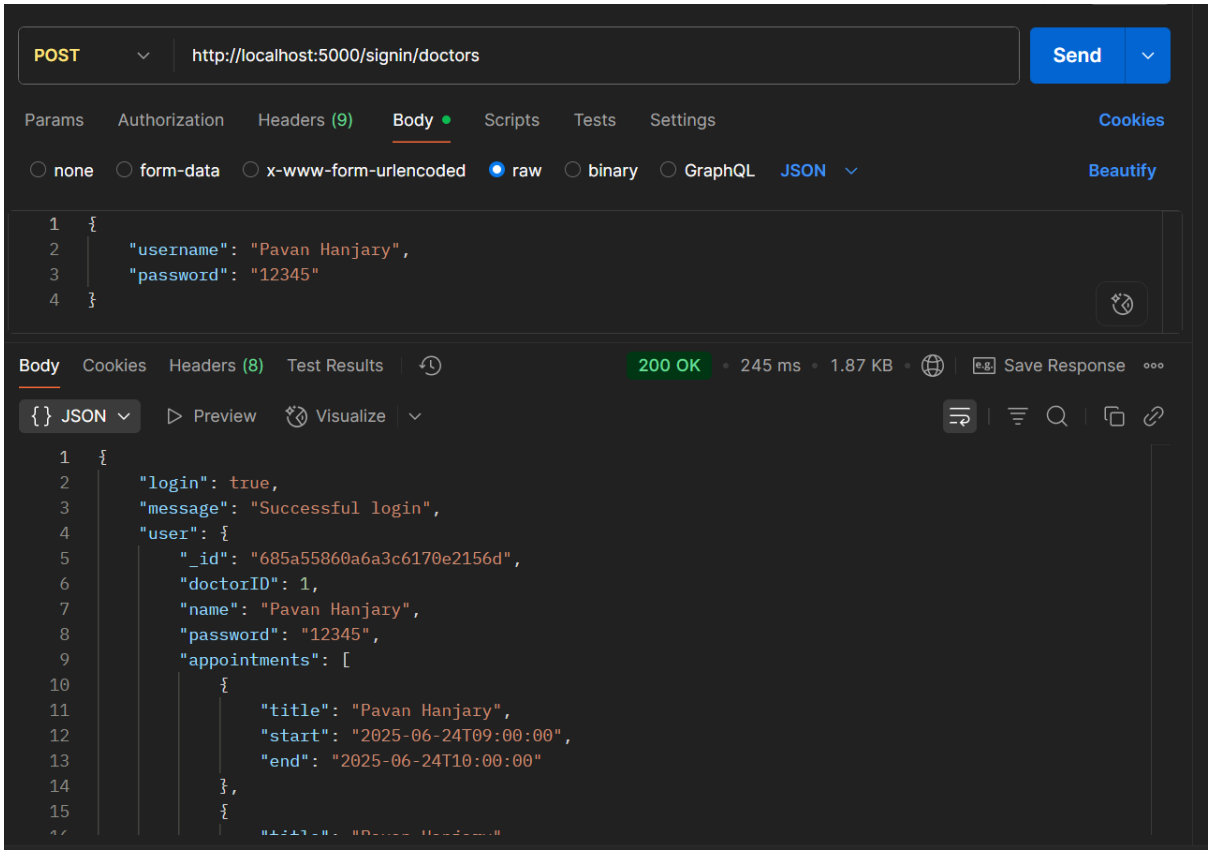


Figure 90 - signin Postman request 1

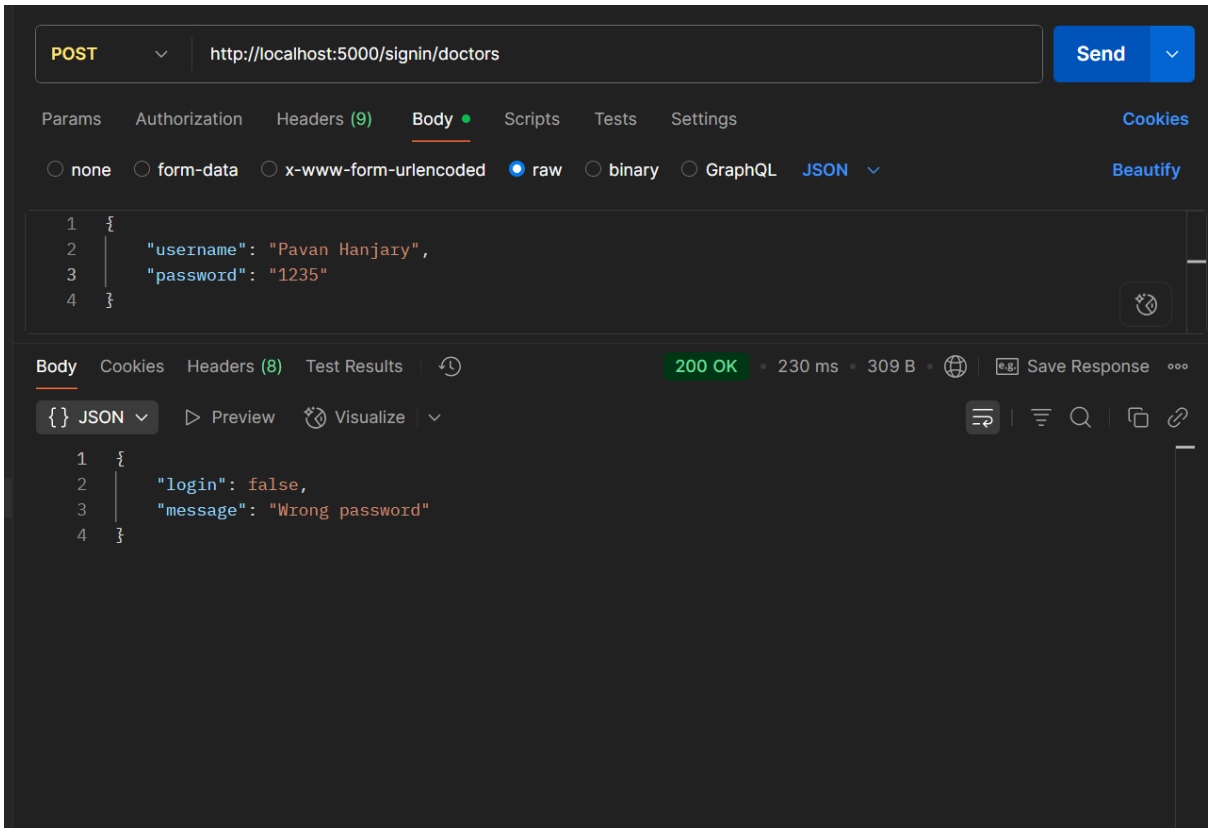


Figure 91 - signin Postman request 2

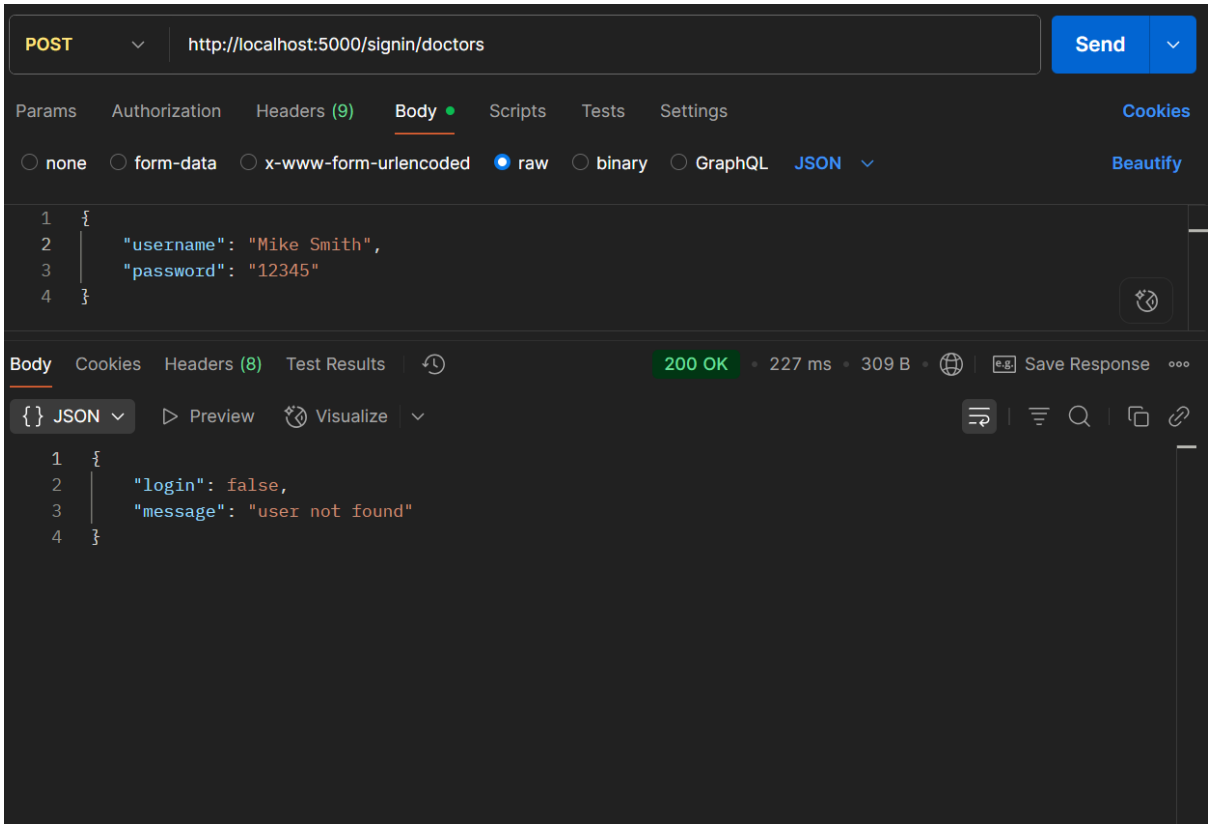


Figure 92 - signin Postman request 3

Get patient severity

The **/getPatientSeverity** route of the server is used to get patient severity using the patient's notes. The note is sent as body to the request. The snapshot below shows the postman test with the patient severity as response.

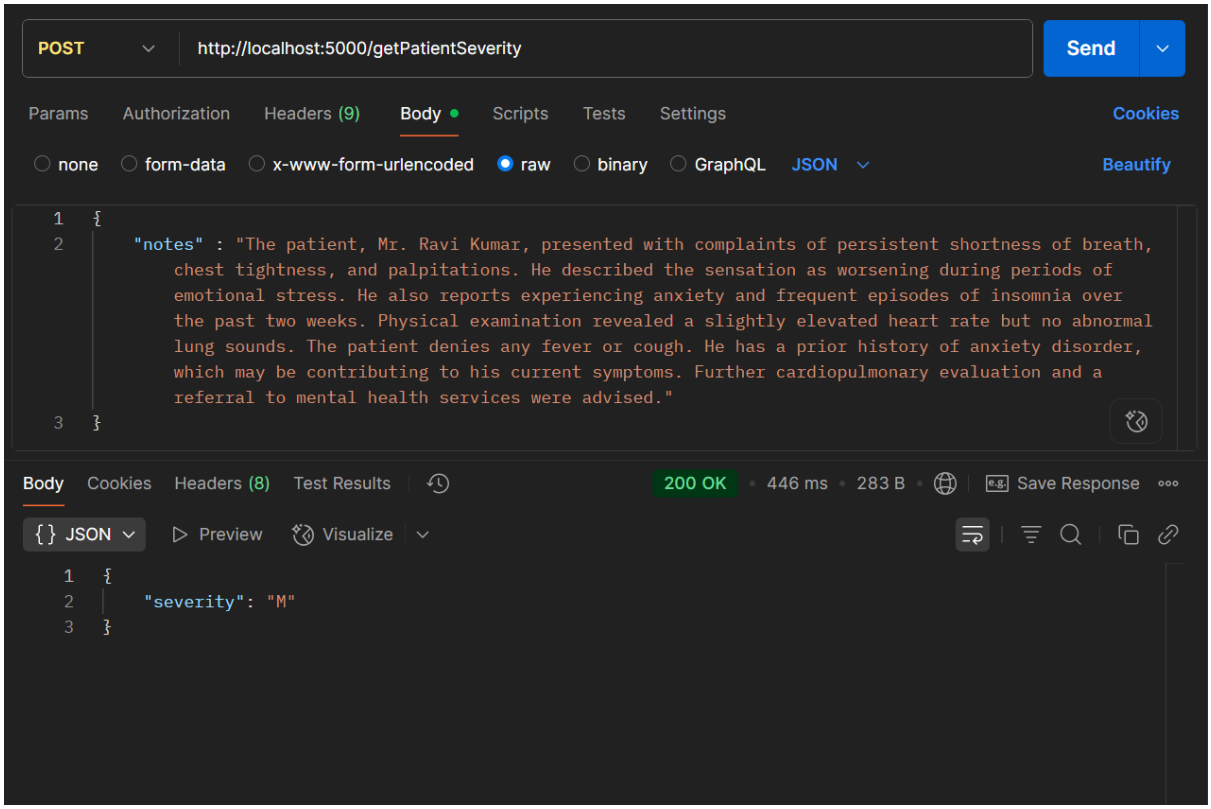


Figure 93 - get patient severity postman test

Update doctor calendar

The **/updateDoctorCalendar** route is used to update doctor calendar when new appointments are booked. The snapshots below show the postman request and the updated MongoDB collection.

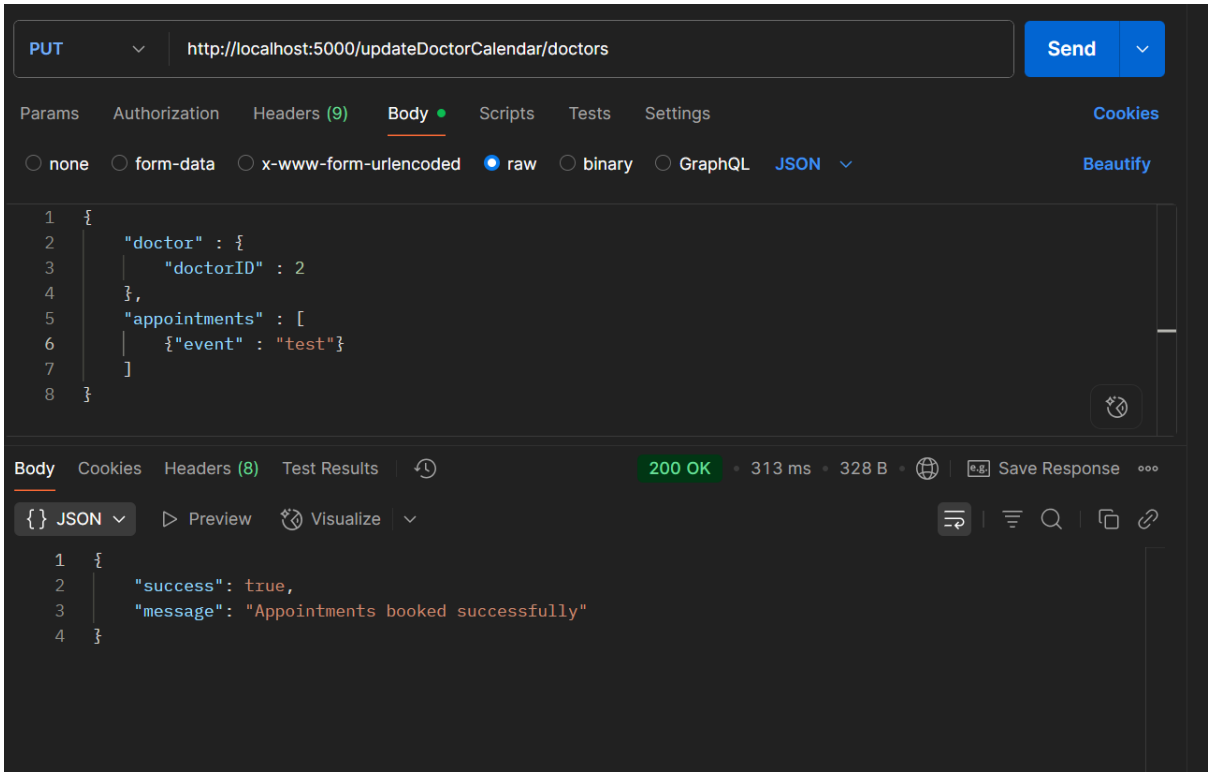


Figure 94 - postman request for updating doctor calendar

```
_id: ObjectId('685ef5d53297bf0bf06ffd14')
doctorID : 2
name : "Shana Purwanand"
password : "12345"
appointments : Array (1)
  0: Object
    event : "test"
```

Figure 95 - Updated mongodb collection for update calendar

9.1.2 Flask API testing

The postman request below shows the test for the flask API to test if the model returns the correct severity of a patient based on the symptoms sent as the body to the request. The response shows successful classification of the severity of the patient.

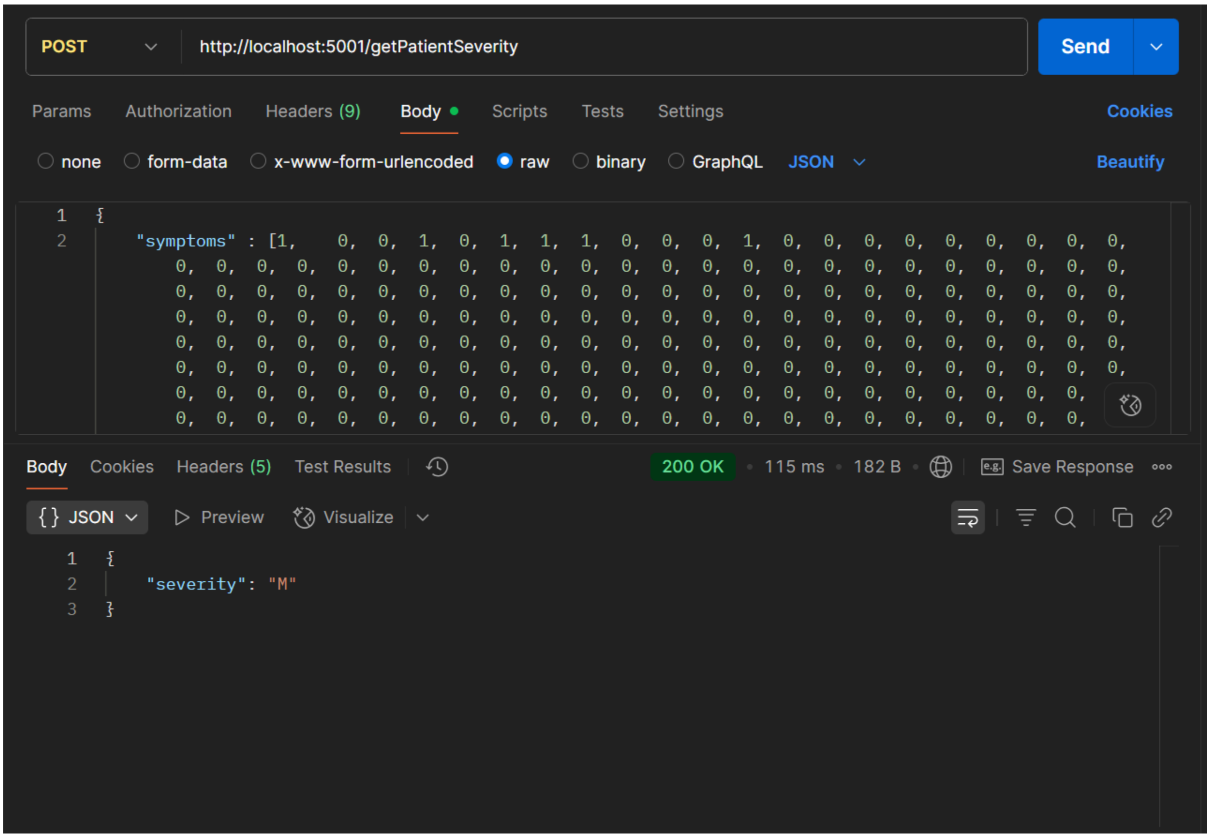


Figure 96 - Postman request for flask API

9.2 Integration testing

Integration testing is performed to check the interactions between integrated components to ensure the different parts of a system work together as expected (Katalon.com, 2023). Following the unit tests, integration testing will be carried out to ensure the different components work properly to achieve the functionalities of the software. For this project, integration testing will focus mainly on the different workflow and on the interaction between the backend and the frontend.

Frontend and fetch API

The different fetch API endpoints have been tested from the frontend to ensure that frontend requests are correctly handled, and the correct outputs are received. Different inputs have been used to test if the endpoints return the appropriate error messages. All the endpoints function as intended and send the appropriate responses to the frontend. The frontend also successfully displayed the response. For example, error messages are displayed in the frontend when wrong login information are entered.

Database and server

The interaction between the server and the database has been tested for every server endpoint where interactions with the database is present. The tests show successful update, read and create operations for the different collections present in the MongoDB

database. For example, when a new patient record is uploaded, the new record is visible in the MongoDB collection patientHistory viewed using MongoDB compass.

Server and flask API

The interaction between the server and the flask API has been tested to ensure the severity of the patient is correctly predicted. The tests have been designed to test for different severity levels of the patients. The results obtained are accurate in determining the patient severity and same notes return the same response on multiple requests. This concludes the proper interaction between these two crucial components of the project.

End to end workflow testing

Complex workflows have been tested to ensure the proper functioning of the application without the application crashing and with proper functioning of different features. For example, several appointments have been added to appointment queue followed by viewing of patient files and updating new records of several patients. Later, when the appointments were booked, the bookings were successfully completed. This confirmed the proper functioning of the application without major issues.

9.3 Feature testing

Feature testing involves thorough testing of a feature which might help identify any bugs, errors or problems when using the said feature (Testsigma, 2024). In this section, the different features will be tested for different use scenarios enduring they work with the best user experience and as intended.

Sign in

The sign in feature tests involved simulation for different user credentials to log onto the system. The different responses were tested to ensure unauthorised access to the system was denied when wrong user credentials were entered. The tests ensured only authorised users get access to the system.

Search records

The search record feature testing involved simulations for different patient names for the appropriate records to be returned. The different tests ensured patient search was accurate and no records were returned for inexistant patients. Multiple searches was also tested to ensure search functionality works for multiple searches.

View patient records

The view patient records testing involved simulations for the different patient records that could be viewed and involved the viewing of records of different patients. This was carried out to make sure the correct records of the selected patient were viewed and to ensure that the records of different patients were not mixed.

Update patient records

The update patient records testing involved simulations for the uploading of different files, notes and prescription of different patients. This was to ensure that the uploaded documents were correctly updated for the right patient in the database and to ensure correct messages were displayed for different use case scenarios.

Book appointment

The book appointment feature test involved simulations to ensure appointments were correctly booked for different severity levels of patients. The tests involved different severity levels of patients placed at random places in the appointment queue before booking the appointments. This was to ensure that the greedy algorithm was working properly in the booking of appointments in order of priority.

9.4 Conclusion

This section involved unit testing, integration testing and feature testing of the SmartCare application. The tests ensured that the application is free of bugs and is working as per the functional requirements of the application. The tests were successful is ensuring that the application is ready for use for its designed purpose.

10 Chapter 6 - Evaluation

The evaluation of SmartCare will involve a systemic approach to see if the system achieves its objectives and to identify the potential fallbacks of the system. In this section, the classification models used to determine severity of patients will be assessed, the algorithm used for appointment scheduling will be assessed and the system will be compared to the state-of-the-art applications mentioned in the literature review. At the end of the evaluation process, the results will be used to identify the contribution of the application to the public healthcare in Mauritius and identify future improvements for the system.

10.1 Evaluation of classification models

The two models that have been trained on the dataset for the classification of patient on different severity levels are a random forest model and a k-nearest neighbour model. In this section, the models will be evaluated based on the following four criteria:

- precision: Precision is the ratio of the actual positives that the model correctly identified and the total number of positives including false positives that the model identified.
- recall: Recall is the ratio of the actual positives that the model correctly identified and the total number of actual positives.
- f1-score: f1-score is the mean of precision and recall.

- accuracy: Accuracy is the ratio of all correct predictions whether positive or negative.

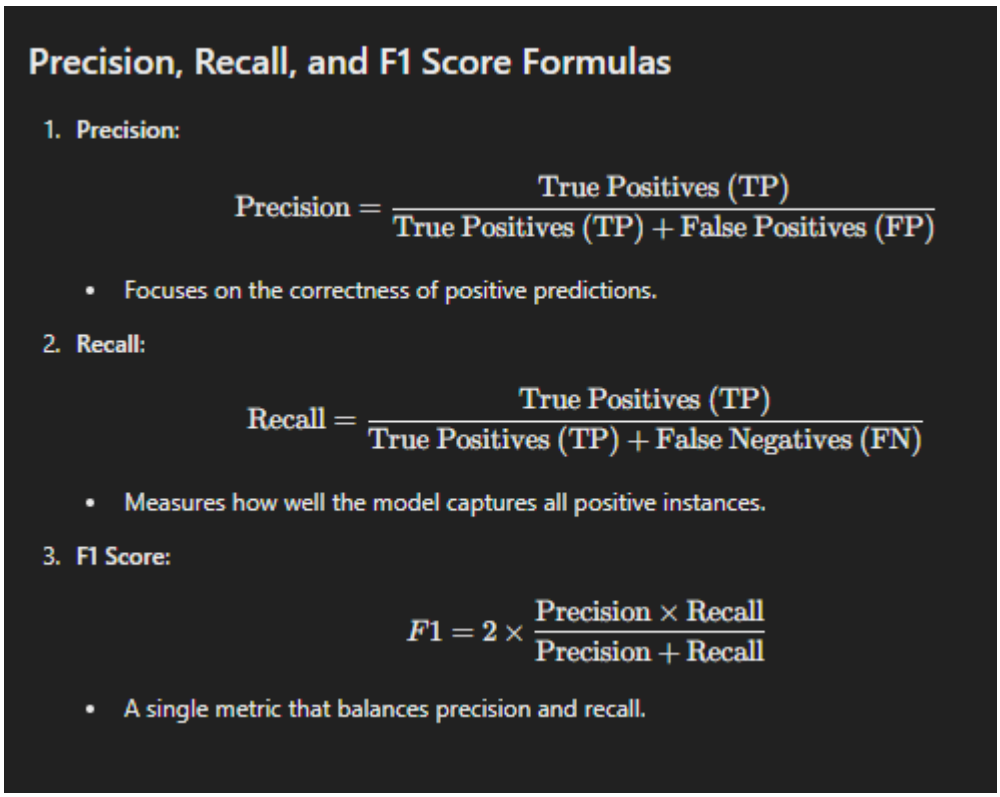


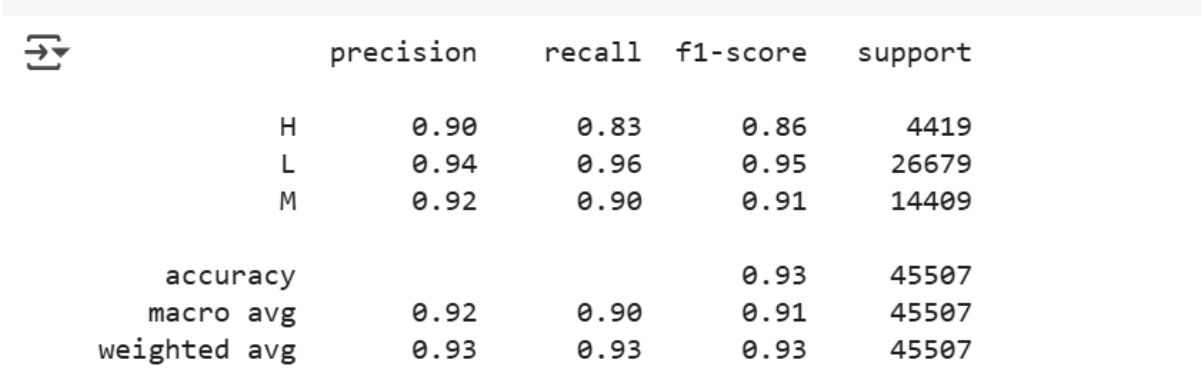
Figure 97 - Model evaluation metrics

The pictures below show the metrics of both models and confusion matrix of the random forest model. The random forest classifier (RF classifier) outperforms the k-nearest neighbour classifier on all performance metrics. This shows that the random forest model is the better suited among the two models. However, the RF classifier has precision scores of 0.90-0.94 for the different severity levels, indicating that the model correctly classifies only 90-94% into the correct category for true positives. It also has recall scores ranging from 0.83-0.90, indicating that the model correctly classifies 83-90% into the correct category for true positives and false negatives.

Even though the model has an overall accuracy of 0.93, it is not the most appropriate model to be used in determining severity of patients. The 7% chance of the classifier not making the correct predictions may have consequences on patient health. The metrics scores for the high category (precision: 0.90, recall: 0.83, f1-score: 0.86), 'H', is the lowest compared to the scores of the 'M' and 'L'. Categories. This further indicates that the margin of error is too high for patients with higher severity diseases when the model is making predictions. The confusion matrix also shows that out of 4419 cases of high severity only 3659 have been correctly classified. For the other two categories of 'M' and 'L'. The model has fairly good scores of 0.91 and 0.95 respectively (f1-scores). The confusion matrix also indicates that for the 'M' category, 13000 cases have been correctly classified out of 14409 cases and for the 'L' category, 25663 cases have been correctly

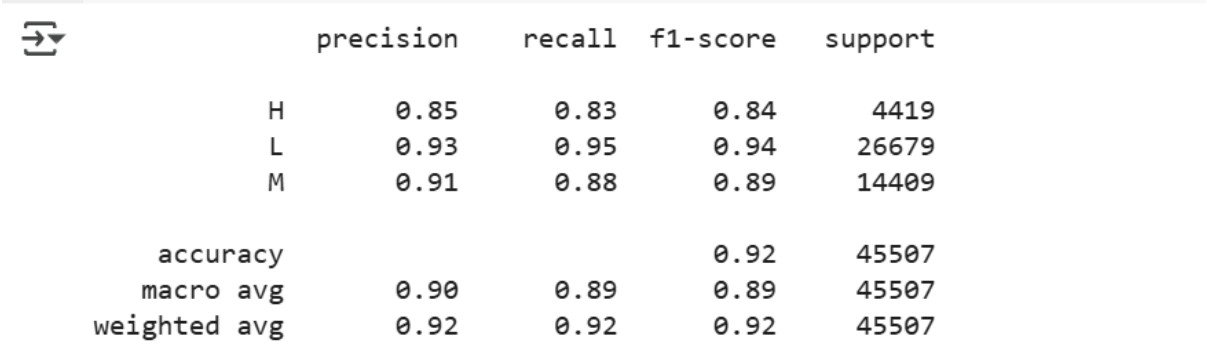
classified out of 26679 cases. These results indicates that for patients with priority other than ‘H’, the model gives good results. This imbalance in metrics scores for the different categories of patients may be the result of an imbalance in the dataset for the different categories. The data available for high category illnesses of 4419 is much lower compared to the 26679 for low priority patients and 14409 for medium priority patients.

To conclude, the model is not suitable for high priority patients as it can be life threatening and defeats the purpose of priority scheduling if a high priority patient is wrongly classified. However, for medium and low priority patients, the model gives good results and can be used for classification of patients.



	precision	recall	f1-score	support
H	0.90	0.83	0.86	4419
L	0.94	0.96	0.95	26679
M	0.92	0.90	0.91	14409
accuracy			0.93	45507
macro avg	0.92	0.90	0.91	45507
weighted avg	0.93	0.93	0.93	45507

Figure 98 - Random Forest scores



	precision	recall	f1-score	support
H	0.85	0.83	0.84	4419
L	0.93	0.95	0.94	26679
M	0.91	0.88	0.89	14409
accuracy			0.92	45507
macro avg	0.90	0.89	0.89	45507
weighted avg	0.92	0.92	0.92	45507

Figure 99 - k-nearest neighbour scores

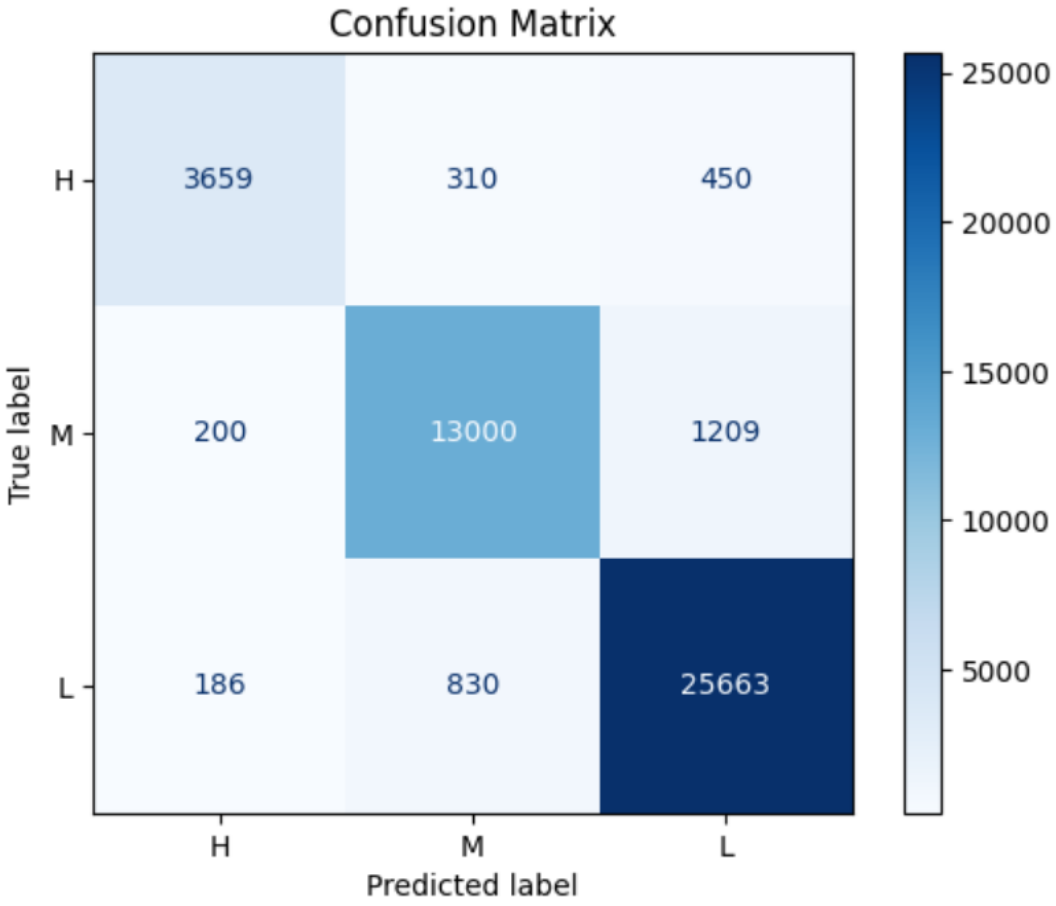


Figure 100 - confusion matrix for Random Forest classifier

10.2 Evaluation of greedy algorithm in appointment scheduling

In this section, the greedy algorithm used will be evaluated to assess if it meets the requirements of the scheduling functionality and whether it is the most suitable algorithm to be used in the public healthcare of Mauritius. The algorithm will be evaluated based on the following criteria:

- Speed of scheduling
- Priority satisfaction
- Adaptability

Speed of scheduling

In appointment, scheduling where there is a high patient load, scheduling must be fast and effective. The greedy algorithm used has a time complexity of O(n) in the best-case scenario. This time complexity works well for the healthcare system in Mauritius as one doctor will likely have no more than 100 appointments to book per day. Thus, the appointments will be quickly booked without much waiting time.

Priority satisfaction

The system has been tested for different batch of patients with different severity levels placed randomly in the waiting list for booking appointments. Each time, the patients have been successfully booked in terms on priority. Therefore, the greedy algorithm fulfils its objective of successfully booking schedules for a batch of patients in order of priority.

Adaptability

The evaluation of the adaptability of the greedy algorithm has been determined by trying different patterns for booking appointments patients. All the patterns involved the following steps but in different orders.

1. Booking of appointment of group of patients with varied severity levels.
2. Booking of appointments of group of patients with low severity levels.
3. Booking of appointments of group patients with high severity levels.

The different tests carried out help draw the following observations:

- When the steps were executed in the order 3, 1 and 2, the patients with high severity had the least amount of waiting time until their next appointment.
- When the steps were executed in the order 2, 1, and 3, only some patients with high severity were allocated earlier appointment dates whereas some patients with high severity levels had long waiting times for before their next appointment.

These observations help to conclude that the greedy algorithm used provide only the best solution for the current batch of patients being scheduled for appointments and does not cater for unexpected arrival of patients with higher severity. Therefore, the scheduling process is only optimised for a current batch but does not provide the best overall global solution in terms of adaptability.

10.3 Comparative analysis of SmartCare with state-of-the-art systems

In this section, an evaluation of the application and its features implemented will be carried out using a comparative analysis of the state-of-the-art systems mentioned in the literature review. The evaluation process will be aimed at finding the strengths and limitations of the project compared to the other systems. The strengths and limitations of the systems will mainly focus on the security of the system, e-prescriptions, viewing of digital patient records, user friendliness and the scheduling of patient appointments.

10.3.1 Existing solution 1

This system is a system mainly designed for managing appointments and for prescriptions. It enables booking and viewing of appointments for both doctors and

patients along with the ability to write prescriptions online. The strengths and weaknesses of the system are outlined below.

Strengths

- It is user-friendly and easy to use.
- Provides easy booking of appointments.
- Enables e-prescriptions.

Weaknesses

- Patient's records cannot be viewed. Thus, the paper-based method of patient records would still have to be used for managing patient history.
- Booking of appointments is manually done which increases labour cost.
- Uses a queue system for booking appointments which not very efficient in handling patients with different severity levels.

10.3.2 Existing solution 2

This system has been designed to manage patient records and for appointment scheduling for patients. It mainly excels in the scheduling appointment feature rather than the viewing of patient records. The strengths and weaknesses of the system are outlined below.

Strengths

- It consists of reminders for appointment scheduling near the appointment dates of a patient. This helps decrease no shows as mentioned earlier in the report.
- E-prescriptions can be written and viewed online.

Weaknesses

- Only supports manual booking of appointments which may not be efficient.
- Records of patients are only partially available with the inability to view patient's laboratory or radiology reports.
- The application is not well organised and is not user-friendly.

10.3.3 Existing solution 3

This application was designed to help switch from paper-based record keeping to digital record keeping. It included features where patient could request for their medical record, storing of patient's visit notes and viewing prescriptions. The strengths and weaknesses are listed below.

Strengths

- Prescriptions can be uploaded and viewed by the patient.
- Records can be shared upon request.

- Secure login to prevent unauthorised access.

Weaknesses

- No appointment scheduling feature available.
- Not user-friendly as the navigations are not clearly labelled.
- Records of patients only include notes of past visit and no files of patient's reports.

10.3.4 SmartCare

SmartCare is a system designed for the public healthcare system in Mauritius. Its main objectives are to reduce medication errors, make appointment scheduling more efficient, provide access to patient's medical records and to digitalise the record keeping of patients. The strengths and weaknesses are outlined below.

Strengths

- Provides full access to history of patients including the viewing of radiology and laboratory reports.
- Enables the uploading of new reports and visit notes.
- Provides a scheduling system on a priority basis with email confirmation of the appointment details and using machine learning to classify patients into different severity levels.
- Enables e-prescriptions with a copy of the prescription sent to the patient via email.
- User-friendly interface with clear navigations.

Weaknesses

- Lack of security for doctor login.

10.3.5 Conclusion

The SmartCare application provides a better automated scheduling system which makes use of machine learning for priority scheduling compared to the other systems. SmartCare also provides complete viewing and updating of patient records compared to the other systems. The ability to access records from anywhere is also a major advantage. The only lacuna of the system compared to the other systems, based on the criteria used to assess the systems, is the lack of proper security for SmartCare.

11 Chapter 7 – Conclusion

The project's main objective was to implement a hospital management system which makes the public healthcare system of Mauritius more effective. This has been planned to be achieved using several features such as the use of electronic records, e-

prescriptions and a better appointment scheduling system. Throughout the project, countless frameworks and technologies such as React.js, Node.js, MongoDB, machine learning among others, have been used in realising this application.

After the implementation phase, the application has been thoroughly tested to ensure that the objectives are met. The tests showed that the functionalities implemented were correctly functioning and that the application was ready to be used.

After the testing phase, several aspects of the application have been evaluated to determine the overall success of the application. The evaluation process showed that the appointment scheduling system did a good job scheduling appointments for a group of patients of different severity levels but does not cater for future cases of high severity patients. The model has also been found to be limited in terms of the accuracy of predicting the correct severity of high severity patients compared to low and medium severity patients due to an imbalance in the dataset. However, the evaluation process also helped establish that the project was successful in some areas. For example, the e-prescriptions and email system were a success compared to the existing solutions. The viewing and uploading of different patient records ranging from files to doctor's notes also proved to be a success in digitalising the records of patients and providing a shared access to patient records.

To conclude, the SmartCare application features which may help the healthcare system in Mauritius in terms of effectiveness and better handling of patient records. However, there is also room for improvement for the scheduling system to address priority scheduling.

11.1 Limitations

Despite the successful implementation of the SmartCare application, there are some limitations to the project. The limitations are discussed below:

- **Appointment scheduling algorithm:** The appointment scheduling algorithms does not cater for the best global solution but only the optimal local solution when booking appointment slots.
- **Security**: The doctor login is not secure enough to prevent unauthorized access to the system. The use of password only to access the system can be easily breached.
- **Doctor schedule**: The doctor's schedule has been manually added to the MongoDB. This is then used for booking of appointments.
- **Classification model**: The model to classify the patients into severity categories is not accurate enough to be used for high severity patients.
- **System testing:** The system has been tested without any user testing. Thus, from a user's point of view, the limitation of the project is unknown.

- **Add new users to the system:** New users cannot be added to the system unless they are added directly to the database.
- **Wink-nlp for text extraction:** The wink-nlp library used for text extracted is limited in terms of its dictionary of words to lemmatise the words.
- **Search functionality:** The search functionality only caters for exact matches and can only search using the name of the patient.

11.2 Future works

Based on the limitations of the project, the future works to improve the current system are discussed below:

- Use a different algorithm to cater for unforeseen cases of high severity patients to book appointments. For example, double booking of slots or leaving some empty slots for early booking can be used to mitigate this issue.
- Two factor authentication can be implemented to provide a more secure application.
- A feature can be implemented where the doctor can set the days of consultation in a week, the consultation time for the days and the setting of slot intervals for each session can be implemented to facilitate the dynamic change of doctor schedules.
- A more balanced dataset and better machine model line neural network can be used to create a model with better accuracy.
- The system must be tested by actual users of the system to have a better idea of the limitations and strengths of the system.
- An admin profile must be created. The admin will have the ability to create new login accounts for doctors.
- A better NLP model can be trained and used for text extraction.
- Fuzzy search can be implemented to handle wrongly type names. Options can also be added to enable patient search using different patient attributes.

12 References

Africa Health Business Limited (2021) MAURITIUS' HEALTH SECTOR. (n.d.). Available at: https://ahb.co.ke/wp-content/uploads/2021/07/Mauritius-Country-Overview.pdf.

Amazon Web Services, Inc. (2024). *Benefits*. [online] Available at: https://aws.amazon.com/application-hosting/benefits/ [Accessed 2 Mar. 2025].

Babu, A.C., Sai, C., Reddy, A.D., Kumar, E.N. and Srinivas, V. (2023). Web Based Hospital Management System. *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*. [online] doi:https://doi.org/10.1109/icaccs57279.2023.10112962.

Bu.edu. (2025). *Why Use 2FA? : TechWeb : Boston University*. [online] Available at: https://www.bu.edu/tech/support/information-security/why-use-2fa/ [Accessed 22 Feb. 2025].

Cms.gov. (2024). *Electronic Health Records | CMS*. [online] Available at: https://www.cms.gov/priorities/key-initiatives/e-health/records [Accessed 26 Feb. 2025].

Devi, S.Sharmila., J.S. Deepica, K. Dharshini and G. Dhivyashree (2021). User Interactive Hospital Management System by using Web application. *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, [online] pp.1578–1585. doi:https://doi.org/10.1109/icesc51422.2021.9532887.

Diaz, A.G., Gumtang, A.D., Joy, C., Balagot, A.S., Villanueva, E.A. and Manalang, M.A. (2024). PHIrecord: A Medical Record Management System for Rural Health Facilities in the Philippines. [online] pp.188–193. doi:https://doi.org/10.1109/isci62787.2024.10668022.

Ehrinpractice.com. (2024). *The problem with paper charting - and how EHR can help*. [online] Available at: https://www.ehrinpractice.com/problem-with-paper-charting.html [Accessed 2 Feb. 2025].

GeeksforGeeks (2017). *What is Software Testing?* [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/software-testing-basics/ [Accessed 27 Jun. 2025].

Healthit.gov. (2025). *What are the advantages of electronic health records? | HealthIT.gov*. [online] Available at: https://www.healthit.gov/faq/what-are-advantages-electronic-health-records [Accessed 26 Feb. 2025].

John (2022). *8 Pros and Cons of Paper-Based Medical Records*. [online] Dallas/Ft. Worth | ISOwire LLC. Available at: https://www.isowire.com/pros-cons-paper-medical-records [Accessed 2 Feb. 2025].

Katalon.com. (2023). *What is Integration Testing? Definition, How-to, Examples*. [online] Available at: https://katalon.com/resources-center/blog/integration-testing [Accessed 27 Jun. 2025].

Kuiper, A., Mast, J. de and Mandjes, M. (2019). The problem of appointment scheduling in outpatient clinics: A multiple case study of clinical practice. *Omega*, [online] 98, pp.102122–102122. doi:https://doi.org/10.1016/j.omega.2019.102122.

Merge.rocks. (2022). *User-friendly interface design - why have it and what are its principles | Merge Development*. [online] Available at: https://merge.rocks/blog/what-is-a-user-friendly-interface [Accessed 26 Feb. 2025].

Michaud, P.-J.L. (2024). *Agile software development: everything you need to know*. [online] Nexapp.ca. Available at: https://www.nexapp.ca/en/blog/agile-software-development [Accessed 3 Feb. 2025].

Microsoft (2021). *Visual Studio Code*. [online] Visualstudio.com. Available at: https://code.visualstudio.com/docs/editor/whyvscode [Accessed 19 Jun. 2025].

Mocdoc.com. (2024). *6 Reasons Why You Need a Hospital Management System*. [online] Available at: https://mocdoc.com/blog/6-reasons-why-you-need-a-hospital-management-system [Accessed 2 Feb. 2025].

MongoDB. (2024). *Advantages Of MongoDB*. [online] Available at: https://www.mongodb.com/resources/compare/advantages-of-mongodb [Accessed 2 Mar. 2025].

Moura, A. and Pinho, M. (2025). A Scheduling Optimization Approach to Reduce Outpatient Waiting Times for Specialists. *Healthcare*, [online] 13(7), pp.749–749. doi:https://doi.org/10.3390/healthcare13070749.

Nodejs.org. (2015). *Node.js — Introduction to Node.js*. [online] Available at: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs [Accessed 2 Mar. 2025].

Oleksandr Hutsulyak (2024). *Why Use React for Web Development: 10 Reasons To Apply*. [online] Blog | TechMagic . Available at: https://www.techmagic.co/blog/why-we-use-react-js-in-the-development [Accessed 1 Jul. 2025].

React.dev. (2015). *React*. [online] Available at: https://react.dev/ [Accessed 1 Jul. 2025].

Singh, K. (2013). *Web Design: 11 Characteristics of a User-Friendly Website*. [online] Social Media Today. Available at: https://www.socialmediatoday.com/content/web-design-11-characteristics-user-friendly-website [Accessed 19 Feb. 2025].

smartbear.com. (2025). *What Is Unit Testing?* [online] Available at: https://smartbear.com/learn/automated-testing/what-is-unit-testing/ [Accessed 27 Jun. 2025].

Testsigma (2024). *Feature Testing- What It Is and How It Works?* [online] Testsigma.com. Available at: https://testsigma.com/blog/feature-testing/ [Accessed 29 Jun. 2025].

Thelwell, K. (2022). *Health Care in Mauritius - The Borgen Project*. [online] The Borgen Project. Available at: https://borgenproject.org/health-care-in-mauritius/ [Accessed 1 Mar. 2025].

Velo, G.P. and Pietro Minuz (2009). Medication errors: prescribing faults and prescription errors. *British Journal of Clinical Pharmacology*, [online] 67(6), pp.624–628. doi:https://doi.org/10.1111/j.1365-2125.2009.03425.x.

Wang, Y. (2023). Review on greedy algorithm. *Theoretical and Natural Science*, [online] 14(1), pp.233–239. doi:https://doi.org/10.54254/2753-8818/14/20241041.

WHO | Regional Office for Africa. (2024). *Strengthening Mauritius' Primary Health Care approach to improve the population's health and well-being*. [online] Available at: https://www.afro.who.int/countries/mauritius/news/strengthening-mauritius-primary-health-care-approach-improve-populations-health-and-well-being [Accessed 28 Feb. 2025].

12.1 Annex

12.1.1 Ethics form

Research Ethics Screening Form for Students

Only for students on taught programmes – e.g., BSc, MSc, MA, LLM etc
NOT for PostGraduate Researchers – e.g., MRes/MPhil/PhD degrees

Middlesex University is concerned with protecting the rights, health, safety, dignity, and privacy of its research participants. It is also concerned with protecting the health, safety, rights, and academic freedom of its students and with safeguarding its own reputation for conducting high quality, ethical research.

This Research Ethics Screening Form will enable students to self-assess and determine whether the research requires ethical review and approval via the Middlesex Online Research Ethics (MORE) form before commencing the study. Supervisors must approve this form after consultation with students.

Student Name:	Hanjary Pavan	Email: PH526@live.mdx.ac.uk
Research project title:	Hospital Management System	
Programme of study/module:	CST3990 – Undergraduate Individual Project	
Supervisor Name:	Dr. Aditya Santokhee	Email: A.Santokhee@mdx.ac.mu

Please answer whether your research/study involves any of the following given below:			
1. [H]ANIMALS or animal parts.	☐ Yes	☒ No	☐ Maybe
2. [M]CELL LINES (established and commercially available cells - biological research).	☐ Yes	☒ No	☐ Maybe
3. [H]CELL CULTURE (Primary: from animal/human cells- biological research).	☐ Yes	☒ No	☐ Maybe
4. [H]CLINICAL Audits or Assessments (e.g. in medical settings).	☐ Yes	☒ No	☐ Maybe
5. [X]CONFLICT of INTEREST or lack of IMPARTIALITY. If unsure see “Code of Practice for Research” (Sec 3.5) at: https://unihub.mdx.ac.uk/study/spotlights/types/research-at-middlesex/research-ethics	☐ Yes	☒ No	☐ Maybe
6. [X]DATA to be used that is not freely available (e.g. secondary data needing permission for access or use).	☐ Yes	☒ No	☐ Maybe
7. [X]DAMAGE (e.g., to precious artefacts or to the environment) or present a significant risk to society).	☐ Yes	☒ No	☐ Maybe

8. [X]EXTERNAL ORGANISATION – research carried out within an external organisation or your reseach is commissioned by a government (or government body).	☐ Yes	☒ No	☐ Maybe
9. [M]FIELDWORK (e.g biological research, ethnography studies).	☐ Yes	☒ No	☐ Maybe
10. [H]GENETICALLTY MODIFIED ORGANISMS (GMOs) (biological research).	☐ Yes	☒ No	☐ Maybe
11. [H]GENE THERAPY including DNA sequenced data (biological research).	☐ Yes	☒ No	☐ Maybe
12. [M]HUMAN PARTICIPANTS – ANONYMOUS Questionnaires (participants not identified or identifiable).	☐ Yes	☒ No	☐ Maybe
13. [X]HUMAN PARTICIPANTS – IDENTIFIABLE (participants are identified or can be identified even with pusedo names or subject coding used): survey questionnaire/ INTERVIEWS / focus groups / experiments / observation studies/ evaluation studies.	☐ Yes	☒ No	☐ Maybe
14. [H]HUMAN TISSUE (e.g., human relevant material, e.g., blood, saliva, urine, breast milk, faecal material).	☐ Yes	☒ No	☐ Maybe
15. [H]ILLEGAL/HARMFUL activities research (e.g., development of technology intended to be used in an illegal/harmful context or to breach security systems, searching the internet for information on highly sensitive topics such as child and extreme pornography, terrorism, use of the DARK WEB, research harmful to national security).	☐ Yes	☒ No	☐ Maybe
16. [X]PERMISSION is required to access premises or research participants.	☐ Yes	☒ No	☐ Maybe
17. [X]PERSONAL DATA PROCESSING (Any activity with data that can directly or indirectly identify a living person). For example data gathered from interviews, databases, digital devices such as mobile phones, social media or internet platforms or apps with or without individuals'/owners' knowledge or consent, and/or could lead to individuals/owners being IDENTIFIED or SPECIAL CATEGORY DATA (GDPR[1]) or CRIMINAL OFFENCE DATA. [1]Special category data (GDPR- Art.9): "personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation".	☐ Yes	☒ No	☐ Maybe
18. [X]PUBLIC WORKS DOCTORATES: Evidence of permission is required for use of works/artifacts (that are protected by Intellectual Property (IP) Rights, e.g. copyright, design right) in a doctoral critical commentary when the IP in the work/artifactis jointly prepared/produced or is owned by another body	☐ Yes	☒ No	☐ Maybe

19. [H]RISK OF PHYSICAL OR PSYCHOLOGICAL HARM (e.g., TRAVEL to dangerous places in your own country or in a foreign country (see https://www.gov.uk/foreign-travel-advice), research with NGOs/humanitarian groups in conflict/dangerous zones, development of technology/agent/chemical that may be harmful to others, any other foreseeable dangerous risks).	☐ Yes	☒ No	☐ Maybe
20. [X]SECURITY CLEARANCE – required for research.	☐ Yes	☒ No	☐ Maybe
21. [X]SENSITIVE TOPICS (e.g., anything deeply personal and distressing, taboo, intrusive, stigmatising, sexual in nature, potentially dangerous, etc).	☐ Yes	☒ No	☐ Maybe

M – Minimal Risk; X – More than Minimal Risk. H – High Risk

If you have answered 'Yes' to ANY of the items in the table, your application REQUIRES ethical review and approval using the MOREform **BEFORE commencing your research**.

Please apply for ethical approval using the MOREform (https://moreform.mdx.ac.uk/). Consult your supervisor for guidance. Also see *Middlesex Online Research Ethics (*MyLearning area*) and www.tiny.cc/mdx-ethics*

If you have answered 'Maybe' to items L, M, X, H then you need to have your application reviewed by your supervisor **BEFORE commencing your research**.

If you have answered 'No' to ALL of the items in the table, your application is Low Risk and you may NOT require ethical review and approval using the MOREform before commencing your research. Your research supervisor will confirm need to confirm this below **BEFORE commencing your research**.

Student Signature:…… Date: 02/03/2025

To be completed by the supervisor:

Based on the details provided in the self-assesment form, I confirm that:	Insert Y or N
The study is Low Risk and *does not require* ethical review & approval using the MOREform	Y
The study *requires* ethical review and approval using the MOREform.	N

Superivsor Signature:…….. ……..... Date:……27/02/2025……

12.1.2 Meeting Logs

Student Name: Pavan Hanjary

Student Number: M00912754

Date	Comments	Supervisor's signature
21 Jan 2025	Discussed project proposal and project ideas.	ADS
27 Jan 2025	Discussed about possible features to implement in the project	ADS
05 Feb 2025	Discussed literature review based on the proposal.	ADS
27 Feb 2025	Review of literature review draft and possible improvements.	ADS
05 Mar 2025	Discussed on appointment scheduling feature.	ADS
17 June 2025	Discussion on final report and amendments for final submission.	ADS