**A picture containing text

Description automatically generated**

**School of Computing and Mathematical Sciences**

**CO7201 Individual Project**

**Final Report Template**

**AN AUTOMATED SYSTEM FOR LOCAL GPS**

**[SHREEYA DINESH DESAI]**

**[sdd13@student.le.ac.uk]**

**[239038636]**

**Project Supervisor: [DR. YAKUN JU]**

**Principal Marker: [DR. STANLEY FUNG]**

**Word Count: []**

**[27/02/2025]**

**DECLARATION**

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people’s work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people’s work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: [Shreeya Dinesh Desai]

Date: [27/04/2025]

Table of Contents

[1. Introduction 5](#_Toc197896412)

[1.1 Aim. 6](#_Toc197896413)

[1.2 Objective. 6](#_Toc197896414)

[1.3 Requirements. 6](#_Toc197896415)

[1.3.1 Essential Requirements. 6](#_Toc197896416)

[1.3.2 Recommended Requirements. 7](#_Toc197896417)

[1.3.3 Optional Requirements. 7](#_Toc197896418)

[1.4 Tools and Technology used. 7](#_Toc197896419)

[1.5 Structure Overview. 8](#_Toc197896420)

[2. Background Research 9](#_Toc197896421)

[2.1 Literature Review 9](#_Toc197896422)

[2.2Existing Applications. 10](#_Toc197896423)

[2.2.1 Website 1 10](#_Toc197896424)

[2.2.2 Website 2 10](#_Toc197896425)

[2.2.3 Website 11](#_Toc197896426)

[3. System Architecture & Design 11](#_Toc197896427)

[3.1 Backend Architecture 11](#_Toc197896428)

[3.1.1 Microservice Architecture Overview. 11](#_Toc197896429)

[3.1.2 Service Description. 11](#_Toc197896430)

[3.1.3 ER-Diagrams. 13](#_Toc197896431)

[3.1.4 Class Diagram. 13](#_Toc197896432)

[3.1.5 Use- Case Diagram. 13](#_Toc197896433)

[3.2 Frontend Architecture. 14](#_Toc197896434)

[3.2.1 UI design (Wireframes and High-Fidelity Prototype). 14](#_Toc197896435)

[3.2.2 Core Functionalities. 14](#_Toc197896436)

[3.3 Design challenges. 16](#_Toc197896437)

[4. Development Process. 16](#_Toc197896438)

[4.1 Backend Development 16](#_Toc197896439)

[4.1.1 Patient Routes/ API endpoints. 17](#_Toc197896440)

[4.1.1.1 Authentication (auth.py) 17](#_Toc197896441)

[4.1.1.2 Book Appointments 17](#_Toc197896442)

[4.1.1.3 Prescriptions 18](#_Toc197896443)

[4.1.1.4 Buy Prescriptions 18](#_Toc197896444)

[4.1.2 Staff Routes/ API endpoints. 18](#_Toc197896445)

[4.1.2.1 Admin Routes 18](#_Toc197896446)

[4.1.2.2 Doctor Manage Appointments 19](#_Toc197896447)

[4.1.2.3 Prescription 19](#_Toc197896448)

[4.1.2.4 Staff Authentication 19](#_Toc197896449)

[4.1.2.5 Staff Availability 20](#_Toc197896450)

[4.2 Medical records. 20](#_Toc197896451)

[4.3 Frontend Development. 21](#_Toc197896452)

[4.3.1 Development Process 21](#_Toc197896453)

[4.3.2 Login Page 22](#_Toc197896454)

[4.3.3 Register Page 22](#_Toc197896455)

[4.3.4 Staff Login Page 23](#_Toc197896456)

[4.3.5 Patient Dashboard Page 23](#_Toc197896457)

[4.3.5.1 Appointment Booking Component 23](#_Toc197896458)

[4.3.5.2 Prescriptions Component 24](#_Toc197896459)

[4.3.5.3 Medical Records Component 24](#_Toc197896460)

[4.3.5.4 Profile Component 24](#_Toc197896461)

[4.3.6 Staff Dashboard Page 24](#_Toc197896462)

[4.3.6.1 Set Availability Component 24](#_Toc197896463)

[4.3.6.2 Get Patient Booking 25](#_Toc197896464)

[4.3.6.3 Prescription Component 25](#_Toc197896465)

[4.3.6.4 Medical Records 26](#_Toc197896466)

[4.3.6.5 Profile 26](#_Toc197896467)

[4.3.7 Admin Dashboard Page 26](#_Toc197896468)

[4.3.7.1 Manage Bookings Component 26](#_Toc197896469)

[4.3.7.2 Add Staff Component 26](#_Toc197896470)

[4.3.7.3 Add Patient Component 26](#_Toc197896471)

[4.3.7.4 Patient List Component 27](#_Toc197896472)

[4.3.7.5 Staff List Component 27](#_Toc197896473)

[5. Testing. 27](#_Toc197896474)

[5.1 API Testing (Postman). 27](#_Toc197896475)

[5.2 System Testing. 27](#_Toc197896476)

[5.3 Usability testing. 27](#_Toc197896477)

[6. Deployment. 27](#_Toc197896478)

[6.1 Architecture. 27](#_Toc197896479)

[6.2 Azure Services Used. 28](#_Toc197896480)

[6.3 Deployment Process. 28](#_Toc197896481)

[7. Results. 29](#_Toc197896482)

[7.1 Achievements. 29](#_Toc197896483)

[7.2 Lesson Learned. 29](#_Toc197896484)

[7.3 Limitations 29](#_Toc197896485)

[8. Conclusion. 29](#_Toc197896486)

[8.1 Future Scope. 29](#_Toc197896487)

[9. References 30](#_Toc197896488)

[10. Appendix. 31](#_Toc197896489)

[10.1 Background Research 31](#_Toc197896490)

[10.2 Existing Websites 35](#_Toc197896491)

# Introduction

Currently in the United Kingdom, the National Health Service (NHS) healthcare staff are facing countless problems out of which two primary problems are lack of staff and insufficient funds to improve the existing digital infrastructure of the healthcare sector. According to the (Medical staffing in the NHS, 2025) report, a large percentage of General Practitioners / Doctors are resigning from their job due to lack of work life balance and degrading personal health. Therefore, most of workload of the senior staffs is now handed over to beginner level or early age Practitioners which lack experiences.

In the year 2024, approximately 42.19% of the staff experienced workload-stress. While 30.24% of the staff felt burnt out as an outcome of their work. Additionally, the healthcare workers felt unappreciated and unhappy with their salaries (Medical staffing in the NHS, 2025).

Due to these challenges, there was an impact on the services provided to the patients. For instance, “Unfortunately, we don’t have any appointments available for today” is the most common phrase used by the receptionists due to the shortage of availability of GP (MacConnachie, 2024).The General Practitioner and NHS staff are facing numerous challenges such as shortage of staff, limited availability of appointments, adolescents not opting to study in the healthcare industry, lack of skilled Doctors. Due to these issues, patients have been suffering from adequate treatment and care. (Khan, 2023).

## 1.1 Aim.

This paper mainly aims to develop An Automated System for Local GP’s, which would be less stressful and user friendly for both healthcare staff and Patients. The Web App would help the staff to set their availability smoothly and cancel their scheduled appointment with an ease. Also allowing Patients to manage their appointments, pay for their Prescriptions, all these with a user-friendly interface of the applications for the Admin, Staff and Patients allowing all the users of the web-app to feel supported and less stressed while performing any task on the system. It is essential to understand that this Web App has been developed considering a specific GP.

## 1.2 Objective.

The objective for this project would be to design a secure and user-friendly web application considering different demography of users, short appointment booking forms to patients with read through articles for minor injuries. Additionally, patients should be able to pay for prescriptions from anywhere in the UK.

On the other hand, the General Practitioner will have a simplified dashboard which would help the doctors to set their availability, view the booked appointment, cancel their availability, view the patient previous medical history and prescribe the medicine.

Lastly, the entire application would be deployed on the Microsoft Azure cloud considering the CAP (Consistency, Availability and Partition Tolerance) theorem.

## 1.3 Requirements.

### 1.3.1 Essential Requirements.

**Registration**

The patient will be able to register themselves to the webapp using secure login credentials.

Registration of Doctors and Nurses will be performed by Admin.

**Availability**

The Doctor and Nurses will be allowed to set their availability two months prior.

**Book appointment**

According to the patient’s requirement, they can book the appointment with the available Doctor.

**Provide prescription**

The Doctor will be able to access medical history of patients and provide a digital prescription on the web app.

**Admin Dashboard**

The Admin Dashboard will allow administrators to add, remove healthcare staff and will also allow administrators to book appointments for aged patients, view the list of patients and the staff.

**Staff Dashboard**

The Dashboard will help staff to set their availability, view booked appointments, provide prescriptions, view patients medical history and send prescription to pharmacy.

**Patient Dashboard**

The Dashboard will help patients to book appointment, view prescriptions, upload the prior medical history, previous booked appointments records.

### 1.3.2 Recommended Requirements.

**Deployment on the cloud**

**Articles for minor injuries & awareness**

**View prescription**

**Buy and Pay prescriptions**

For the prescribed medicine the patient can buy and pay for the prescription either online or offline depending upon the mode of delivery.

### 1.3.3 Optional Requirements.

**Responsive Web Application.**

**One to one chat**

Due shortage of Doctor/Nurse, if in case there's a follow-up required for a specific patient, or a patient requires immediate attention the chat feature can be leveraged.

**Video Consultation**

## 1.4 Tools and Technology used.

The below mentioned details are the tool and technologies used in this project.

|  |  |  |
| --- | --- | --- |
| **Component** | **Name** | **Summary** |
| Database | Azure SQL and Cosmos DB, Blob Storage | Storing and managing the data either in structured or unstructured format. |
| Backend | Python, Flask, Azure Container Apps | Develop business logic. |
| Frontend | React JS, HTML, CSS, JS, Azure App service | Design a dynamic and responsive web application. |
| API | REST | Used to communicate with the database. |
| Authentication | JWT | Used for Secure authentication. |
| Cloud Deployment | AWS / Azure | Hosting the application, considering availability and consistency from CAP principles. |
| Version control | GitLab | Version controls the project. |
| IDE | Visual Studio Code | Tool for editing code. |
| Testing | Manual testing, User Feedback(frontend), Test Cases, Postman (API) | Tests the robustness and security of the application. |
| Designing | Figma, Sketch (paper & pen), Draw.io | Designed a low-fidelity design, wireframes, ER Diagram, Use case diagram and Class diagram. |
| Documentation | MS Word |  |
| Operating System | Windows |  |

## 1.5 Structure Overview.

Introduction: Provides a brief idea about the project including the aim, objective, requirements and tool and technologies used.

Background Research: This section tells us about the research that has been conducted during these three months of the dissertation. Moreover, this section is divided into two parts Literature review and existing applications.

System architecture and Design: This section is divided into two parts: Backend architecture which would have the designs of ER- diagrams, Class diagram and Use-case diagram. Frontend architecture: Display the High-Fidelity designs prototypes and the wireframes plus the core functionalities of the project.

Development process: This section will give us an idea about the entire development process from the scratch to the integration.

Testing: The entire system has been tested manually considering different test case scenarios.

Deployment: This section will speak about the detailed deployment process.

Results: Learning outcomes, conclusion and Future scope of this project would be discussed in this project.

# Background Research

This paper involves ample number of background research with reference to problems faced by the General Practitioner and Patients. In this section, questions like how the existing Web-Apps or Mobile applications helped the patients to monitor their health and access GP services, how impactful will the cloud technologies be in the healthcare sector, how the security of user data should be handled with regards to the confidentiality. Research has been conducted with the help of google scholar, IEEE, articles, PubMed, BBC news (for the most updated news about General Practitioner), Oxford academic and Springer.

## Literature Review

Security has been considered as a major part of any Web-App development, primarily for the healthcare sector, where the confidentiality and integrity of the data needs to be maintained. Inorder to maintain the confidentiality and integrity of the data there are various approaches of developing a secure application or software. According to (Shuaibu & Ruqayyat Ahmad Ibrahim, 2017) the three known approaches used even today are Microsoft Security Development Life Cycle (SDLC), Software Security Touchpoint (SST) and Comprehensive Lightweight Application Security Process (CLASP). Since their approaches are different, but the crucial details would always be around developing a secure application. (Refer Appendix 10.1-a).

Additional considerations should be given to various types of cyber-attacks taking place every second, the goal of these attacks is not just to damage the application servers, but they also intend to steal Users data and sell it on Dark Web. Data stealing is a new norm today, where one can detect or conduct cross-site scripting (XSS) across your web apps and can steal your user data in some hours. The authors (Shuaibu & Ruqayyat Ahmad Ibrahim, 2017) have proposed their own methodology that tries and helps to avoid any sorts of attacks: first was the selection of the framework and designing of the UI for the development, and the second phase was the evaluation. This research paper in specific, exclaims the methodology which the authors seem to rely on. Additionally giving an idea of how important it is to choose appropriate development methodology. (Refer Appendix 10.1-c ).

In the previous decades, the manual health records had various cons such as inconsistent data, missing required files, limited storage and misdiagnosed cases. Thanks to the Electronic Healthcare Record system (EHRC) where one could get the patients record fingertip. This technology has helped the doctors to improve the results of any disease accuracy as well as the communication with the patient. But the storage of patient history has always been a concern. Not only due to the size & scale of patient data, but also due to concern around data privacy. Yes, there are various norms and standards around data compliance and protection. But cybercrime still exists and is carried out on dark web even today.

Back in 2015, there was an increase in patients demand for the medical services, where healthcare system might not have been advanced in countries such as Iraq, India, Yemen, Myanmar. A handful of hospitals and private clinics may have computer-based systems that would enable them to maintain data, but there was no mechanism for sharing the information. Inorder to implement an advanced e-healthcare system there are significant components that need to be considered: Personal Health Record (PDR), Medical Health Record (MDR) and Electronic Health Record (EHR).

This research paper written by (Cloud\_02) Rasha Talal Hameed informs us about the healthcare system of Iraq and the model that has been designed by them, which is based on the service-orientated architecture and cloud computing. One of the most efficient solutions to tackle the digital problems is with the help of cloud computing. In other words, provision of services over the internet (Mohammad Mehrtak, 2021) Currently, the technologies of the cloud are being used in the healthcare industry in order to maintain the electronic records, which would help the patients to easily access their respective information. However, the cloud's storage, real-time information exchange, infrastructure and operating costs, and security are its most worrisome aspects. On the contrary, perks include scalability, flexibility, speed enhancement, cost reduction, and user cooperation ease (Mohammad Mehrtak, 2021) The proposed system management system makes use of computing with REST (Representational State Transfer), Amazon Relational Database Service (RDS), the cloud, SOA and services for web patients (Refer to Appendix, 10.1-10 f). The use-case diagram for this project has users’ doctors, pharmacists, laboratory admins, employees, administration and radiologists, where each of them has various functions (Refer to Appendix, 10.1-10 g).

The rapid evolvement of technology has supported and will support in solving problems which the modern world faces, however none of this would even be possible without regular feedback and thorough checks of the existing services provided in the healthcare domain. Authors (Nasaruddin & Izzatdin Abdul Aziz, 2018) thought of developing a web-based application where one could provide a feedback based on the service provided by the healthcare provider. The main objective behind this was to improve the facilities and services provided by the Staff. For example, an e-commerce website has various type of products where there’s a section of feedback that has been provided by the customers which therefore help the other customer to decide on that product. Similarly, if a patient provides feedback for that healthcare staff and facilities provided, this will help the other patients decision making.

The authors (Nasaruddin & Izzatdin Abdul Aziz, 2018) have conducted research on websites and review systems, including Doctor2U E-commerce, Australian digital My Health Record, Rating, review, chat box, and feedback techniques, as well as gaps discovered in related work on the trust factor, which have given them a better understanding of the methodology used to obtain the feedback. Their research paper explains the development of their system considering different Use cases for Patient (Refer Appendix, 10.1- e), Doctor (Refer Appendix, 10.1- d), Admin (Refer Appendix, 10.1- b). Furthermore, the testing of their application was done using methodologies like Unit testing and integration testing. Initially, this research paper considers use cases which could be used as reference for this paper of Automated GP System.

## 2.2 Existing Applications.

General Practitioners (GP) in the United Kingdom have their website which helps the patient to know about the services they provide. Most of the common services include registering with a general practitioner, scheduling an appointment, viewing prescriptions, and many more. Some GP websites even offer additional information pages about minor-injuries. During the period of research, I had a chance to view a number of GP websites, the main intentions behind doing so was to note the lack of Features and improvement areas.

### 2.2.1 Website 1

The Queens Road Surgery is a General Practitioners who have their clinic located at 282 Queens Rd, Knighton, Leicester. the website displayed an error message “This site can’t be reached.”. Therefore, this type of websites leaves a negative impression to the patients of the services that they might provide.

### 2.2.2 Website 2

The Leicester Holistic GP is another clinic of general practitioners located at 158 Upper New Walk, Leicester, United Kingdom. The official website has advanced services that they provide, such as remote consultation, home visits, face-to-face consultation, and various types of injections for vitamins. Though the website offers multiple services, the website lacks a user-friendly interface.

As reflected in the image, the colour combinations are poor, with no consistency in the alignment of contents, unequal font style, and inconsistent design language for the website, which makes the readability of the website poor. Additionally, the information provided is lengthy, which makes it difficult for the reader to find or locate the required information. A skilfully designed website is as equally important as a broad range of services since both offer the overall impression of the clinic. Working on the cons would give the website a user-friendly interface and help the patients with the easy navigation of the website.

### 2.2.3 Website

The Highfield Surgery is a clinic of general practitioners situated at 25 Severn St, Leicester, United Kingdom. The website (https://highfieldsurgerysevernstreet.co.uk/ ) has essential features that include updating the profile, registration of new patients, access to online prescriptions, booking appointments, and much more. However, there’s a gap in advanced features not provided by the clinic.

As reflected in the image, the website has a handful of areas to improve. The website has plenty of white space, making it appear unevenly balanced. Additionally, the layout of the tile is improper, making it difficult for the poor user experience and the colour combination used for each tile. Furthermore, the colour and the typography used for the website don’t align with the modern website, which is simple and easy to use. The overall website lacks professionalism, which can have a negative impact on the patients.

# System Architecture & Design

## Backend Architecture

### Service Description.

**Users Account**

This service manages essential functionalities related to users, including registration and login. It enables the patient, nurse, doctor and admin to login with their credentials, and according to the user role which is assigned after login, the user would be navigated to their respective dashboard. Additionally, the admin has the privileges to manage the staff, either to add or delete staff members. This ensures that the users are appropriately managed across the system.

**API endpoints**

POST/patient/register: Register a patient.

POST/patient/login: Authenticate patient and generates a token.

POST/staff/login: Authenticate staff and generates a token.

POST/staff/registration: Admin Registers staff.

POST/gp-patient/registration: Admin registers the patient.

The associated entity relationship diagram includes tables such as patient, doctor and nurse. The password for each of the users has been protected with the help of SHA hashing. The login sessions are being maintained, with the help of JWT tokens. The JWT Tokens are generated in a way that they should expire within an hour, each user is assigned a unique JWT and CSRF token upon login. These tokens are required to access protected routes, if these tokens are missing, the user cannot access any protected route. JWT token has been implemented inorder to reduce the risk of unauthorised access. To maintain integrity and safety, the tokens are generated based on the email of the users, and these tokens are sent to the browser of the user via cookies.

**Appointment Service**

This service handles the key functionality of the website, which is appointment booking. The patient books the appointment by selecting the type of disease. According to the selection of disease, the specialized doctor would be displayed for the selected date and the slots available will also be displayed. The patient then selects the convenient slot and books the appointment. Additionally, the patient can cancel the appointment if required.

On the doctor’s side, the scheduled appointment could be viewed and cancel the same if incase of uncertain circumstances. If the appointment has been cancelled by doctor, an email will be sent to specific patient who has booked that specific appointment **[Attach image]**.This ensures clear communication and flexibility between the patient and doctor.

API endpoint

GET/ get\_diseases: Retrieves the list of disease.

POST/get\_doctors\_list: Displays the list of doctors.

GET/ get\_doctor\_availability/<int:doctor\_id>/<string:date>: Retrieves the availability of the doctor on a specific date.

POST/book\_appointment: Books an appointment with the doctor.

GET/my\_appointment: Displays the appointment booked for patient.

DELETE /cancel\_appointment/<int:appointment\_id>: Delete the booked appointment.

GET/ get\_patient\_bookings/<string:date>: Retrieves the patient booking on a specific date.

DELETE/cancel\_doctor\_appointment/<int:appointment\_id>: Doctor cancels the appointment.

**POST/set\_doctor\_availability : Sets the availability for the doctor.**

**DELETE/ cancel\_doctor\_availability/<int:doctor\_id>/<string:date>/<int:slot\_id>: Deletes the availability for the doctor for a specific date..**

**POST/ set\_nurse\_availability: Sets the availability for the nurse.**

**DELETE/ cancel\_nurse\_availability/<int:nurse\_id>/<string:date>/<int:slot\_id>: Deletes the availability for the nurse for a specific** date.

This service has the most complicated entity relationship diagram that includes tables such as appointment, availability, and slots. These tables are interlinked to each other as they share relationship between them. For instance, the doctor must set their availability with the help of slots. Upon adding the availability, if incase the doctor or nurse wants to cancel the availability, they can cancel the same. The available slots are then displayed to the patient inorder to book the appointment. This process needs to be handled with utmost care, since the details displayed on the patient and doctor dashboard need to be accurate and up to date, without any misinformation.

**Prescription Service**

This service helps the doctor to provide digital prescription to the patient. The doctor verifies the patient, then prescribes the medicine to the patient which includes fields such as medicine name, dosage and instruction. The doctor then verifies if the patient is student or general inorder to apply any applicable discount on the medicine. Further, the doctor sends the prescription to the pharmacy and the type of delivery according is selected based on their personal choices.

On the patient dashboard, the system helps the patient to view the prescription provided by the doctor including all the details. If incase, the patients has been provided a prescription, he/she gets an option to buy the prescription, where the payment is done online and the medicine is delivered to the home.

API endpoint.

GET/prescription: Receives the prescription provided by the doctor.

GET/medicines: Retrieves the medicine.

GET/pharmacies: Retrieves the pharmacies.

GET/patients/verify: Verifies the patient.

POST/providePrescription: Doctor provides the prescription to the patient.

The associated entities involved in this service are the pharmacy, patient and payments. The backend uses the structed SQL to store the data of the pharmacy, patients and payments. However, the prescription data is being stored in NOSQL, specifically the Azure Cosmos DB. For instance, the doctor prescribes the medicine to the patient and sends it to the pharmacy. The relationship involved in these are the doctor, patient, payment, medicine, pharmacy making the structing of the database much more complicated.

Additionally, this service assures secure payment using STRIPE and seamless transfer of prescription between the patient and pharmacy, making this as the crucial functionalities of the system.

### ER-Diagrams.

The entity relationship diagram is a frequently used diagram for structured analysis and conceptual modelling (Song, Mary Evans , & E.K. Park, 1995). Numerous entities, attributes, and relationships between them are used in the ER model (Begum & C. P. Indumathi, 2016). As a foundational step towards the designing, the ER model is the initial step in the project that aids in modelling the database (Begum & C. P. Indumathi, 2016). The ER model is easy to understand, helps to solve the issue related to the real world, and can be easily translated to a relational database structure (Song, Mary Evans , & E.K. Park, 1995).

The essential entities identified for this project are admin, patients, appointments, doctors, nurses, specialisations, nurse specialisations, doctor availability, nurse availability, medicine, disease, pharmacy, slots and payments. The relationship between these entities and the attributes to which each corresponds is depicted in the diagram which is designed with the help of draw.io. With the help of the ER diagram, designing the database becomes much easier. For instance, the nurse and doctor availability is linked to the slots, while the book appointment is associated with the patient, doctor and the slots.

### Class Diagram.

The class diagram describes the structure of the system that’s static where it displays the classes and relationship between them. The figure () highlights the name, attributes and operations performed by each of the class.

For instance, the doctor class performs the operations such as cancel appointment, login, profile and delete doctor. Other instances such as the patient, nurse, doctor availability, nurse availability, appointment, admin performs operations such as the book appointment (), set availability () and get list of nurse and doctor.

### Use- Case Diagram.

Upon completion of the ER and the Class diagram, the use case diagram was introduced. The use case diagram in this context helps to represent the functional requirements of the webapp, in another words, it provides information about the behaviour of the application rather than how’s its going to be implemented.

The actors in this are admin, doctor, nurse and patient. However, the figure (), highlights the actor as admin, patient, doctor and nurse, where the admin has the right to adds / remove doctors, nurse and patient. Additionally, the admin can view the list of patients and staff. If incase, any of the patient contact to book the appointment, the admin can book one for them.

On the contrary the second figure (), the patients register themselves and can also books an appointment. The doctor and nurse set the availability two months prior, where the doctor provides the prescription to the patient and also can access the medical history of the patient. Upon prescribing the medicine, the patient could view the prescription and buy the same.

## Frontend Architecture.

### UI design (Wireframes and High-Fidelity Prototype).

Upon successful completion of the technical diagrams (ER, Class and Use case diagram), the **initiation for the designing the UI of webapp has taken place**. This section focuses on user interface and user interaction starting from drawing the wireframes, where the layout would be done with the help sketch, followed by **the high fidelity, which means the final design of the website would look similar, considering the layout, image, font size and typography.**

The main objective of designing the wireframe was to identify the placement of Navigation bar, left side panel, booking appointment form, upload medical history and view the prescription**. The wireframes were completed to get feedback on the same by explaining the ideology and motive behind the same**.

Following the successful completion of the wireframes, the high-fidelity prototype were designed with the help of Figma, where the colour contrast, layout, look and feel were considered for different age group. As a result, this section has aided in achieving successful frontend design while taking the application's features into account.

### 3.2.2 Core Functionalities.

Before heading towards the challenges, it is necessary to outline the core functionalities. As mentioned prior, this website mainly focuses to improve the user experience plus provide a user-friendly interface for the staff, admin and patient. The following are the key dashboards along with their respective function in this project:

Patient dashboard: Designing the dashboard specially for the patient which includes features as book appointment, medical records, prescriptions and profile. Inorder to access these features there was the left side panel and the top navbar where one could logout from the website.

Admin Dashboard: This dashboard was the crucial one as the admin handled adding the patient and staff plus could book the appointment for the patients and get the list of staff and patients.

Staff Dashboard: The dashboard would be accessed by the doctor as well as the nurse where the nurse only the option to set availability and view their profile. On the contrary, the doctor had the function as set availability, cancel availability, view the previous medical history of the patients and provide prescription.

Below mentioned are the functionalities of the dashboards:

Login Screen:

The login page is the main screen of the website, as it has two input fields where one should enter the email address and the password. Upon entering the details, one should click on the login button and then accordingly the patient would be navigated to the dashboard.

On the contrary, the staff also have to login screen. The staff would be the admin, doctor and nurse where the input fields of the form are the same as patient but a twist in this is that for each of the staff there’s a condition as .doctor@, .nurse@, .admin@ where according to this the staff would be navigated to their respective dashboard.

Book appointment:

Displays a form which has an input field as disease description. Also, the patient must select the type of disease from the drop down. Upon selecting the disease, the patient must opt for the appointment date and following to that the specialized doctor in that disease would be displayed. Additionally, after selecting the doctor, the slots available would be seen and the patient could select the most convenient one.

Once the appointment is booked, the details of the appointment would be seen on the same page in the form of a card. If incase, due to any inconvenience the patient wants to cancel the appointment, there a cancel button where one could just click and receive a pop and confirm the cancellation.

Medical Records:

This page would display a drop-down box where the patient has to select the type of record they prefer to upload. Upon selecting, one should select the files from the system and click on upload, this will successfully upload the file. The uploaded files would be seen on the same page.

On the contrary, the doctor would be able to access the same, but firstly they need to verify the patients details by entering their first name, last name and date of birth. Upon confirming the details, the doctor would be able to view the uploaded files.

Set availability:

Displays a calendar upon selecting the select date, the staff is able to set their availability for two months prior and select the slots that they’ll be available. If incase, available for the whole day one could opt for the select all button and then add the availability. But incase, if they aren’t available for a specific time, then one could select the time they are available for and add availability.

The availability set by the staff would be seen on the same page. Additionally, if incase the staff wants to cancel their availability, they could cancel without any hassle by clicking the button cancel.

Provide Prescription:

Display the verify patients page first where the doctor has to enter the patients first name , last name and dob. Upon verifying the details, the doctor is able to provide the prescription by selecting the medicine, dosage and instructions. The doctor also selects the type of patient and the delivery option according to the patient such as if the patient is student they get some discount on the medicine unlike the normal patient. Upon providing the prescription, according to the patients needs the doctor select the pharmacy available and send it to the pharmacy.

View Prescription:

This screen displays the prescription provided by the doctor with details such as medicine name, dosage, instructions along with the type of delivery mode. If incase the patient opted for home delivery, the patient could do the payment online and the medicine would be delivered at their doorstep.

Profile:

This page is for both the staff and patient. For the patient, the screen displays the first name, last name, date of birth, email address, gender, contact number, street address, city and postcode. On the contrary, for the staff the screen displays first name, last name, date of birth, contact number, specialization, email address and registration number.

## 3.3 Design challenges.

During the design phase of this project, there were several challenges encountered that required thoughtful consideration and evaluation. The challenges faced were both technical and non-technical.

Starting with the technical design challenges, Entity relationship diagram was the first step towards the database. Designing the complex ER diagram and integrating the same in the SQL database was a challenging part, since the entities were the admin, book appointment, patient, doctor, setting the availability. One of the key challenges in this was the linking the entities along with the attributes considering the primary key and foreign key. For instance, a patient wants to book an appointment with the doctor, where the appointment table would have the primary key as appointment id and foreign key as the doctor id and the patient it. The entity relationship diagram required maximum attention, as many iterations were done inorder to get the required structure of the database and the endpoint, plus the relationship between them.

Secondly, the non – technical challenge was designing the layout for the dashboard of admin, staff (doctor and nurse) and the patient. One of the key points while designing, was the look and feel of the website and keeping the user interface as simple as possible since there would be different age group patients accessing the website. Also choosing the colour combination and typography was also a tricky part. The colours do have their own psychology as calm, emergency, growth, nature, trust and many more. Since the website was being designed for medical purpose, the colour psychology plays an important role, and the chosen colour were red, green, blue, white and off white as the color itself informed the patients about each of the features.

Additionally, easy navigation through the website with the required information was a key point to provide the users a straightforward experience without much of re-routing or confusion through the website.

# Development Process.

## Backend Development

The backend was developed in Flask API, Python, ensuring maintainability and scalability. The services were designed as a self-standing endpoint, having its associated serviceability and business logic.

The database-first approach was employed where designing the database for each of the entity was done, followed by the endpoints. Each of the endpoints was built according to the features and later were tested in Postman to confirm the desired output.

The folder structure of the backend was organized considering the best practices, ensuring scalability, maintainability, understandability, and readability of the application. Under the application folder, sub-folders were created for models, routes, and utils. The model’s folder defined the admin, the routes folder was subdivided into patient routes and staff routes, each of them having their respective endpoints and functionalities, and lastly, the utils had the helper functions.

Additionally, a Docker file, environment file, and main file were also created, which would handle the configuration, development and deployment of the application. In order ensure security and cross-origin request, CORS and JWT tokens have been implemented which ensures that data is allowed only from the authorized domain.

### Patient Routes/ API endpoints.

This section provides thorough information about the endpoints that are used for the functionalities of the patient, which are authentication, booking appointments, prescriptions, and buying prescriptions.

### Authentication (auth.py)

For the Authentication module to be implemented, which must only allow access to authorised user. The patient register endpoint is developed which is a POST method that includes the required parameters. In case any of the fields missing, a 400 Bad request error message would be sent in response to this API call. The API also checks in the database if the patient is already registered based on the email or phone number and accordingly displays another error message stating that user already exist. In this module itself, when a patient is registered successfully a blob storage is assigned to the respective patient. This blob storage has been used to upload the medical records of patients.

The patient login endpoint is a POST method that validates the entered credentials against SQL database. In case any of the fields are missing or invalid credentials are entered, error messages 400 and 401 are sent in response. Upon successful login, the cookies and refresh cookies are produced and are sent to the user’s browser.

The patient profile API is implemented to get the current patients details which is a protected route with a JWT token. This endpoint is a GET method where the patient details are retrieved from the patient email ID and maps the details of the patient from the database to the dictionary. If, in case the patient is not found with the email ID, it responds with an error message.

Additionally, for all the endpoints, if there’s any database connection failure or any exception error 500 error message is displayed.

The utility functions used in this section were the password hashing done in the SHA256 algorithm, get database connection from db helpers and the blob storage access rights are provided, which were made public when the patient was registered.

### Book Appointments

This module will basically allow patients to book appointments with an ease. To implement this, feature a set of APIS are used in sequence to allow patient to have a user-friendly experience. All the APIS which will be used are protected routes hence they will be secured with a JWT token required flag.

Starting with the get disease endpoint, which is a GET method used to retrieve the list of disease from which patients can select their disease.

The get doctor list endpoint is the next API call, which is a POST method, where a list of specialized doctors for the selected disease is provided when the post methods is called. In this endpoint the slots table, appointment table, availability table and disease table are accessed using the SQL query, which in response returns with list of doctors.

Followed by the get doctor's availability endpoint which is a GET method. This endpoint executes the availability of the selected doctor for a particular date. In other words, the endpoint displays the slots available for the doctor for the specific date.

The book appointment is a post method mainly used to schedule the appointment for the patient. The endpoint fetches the details sent by the patient and checks if the slot selected by the patient is already booked. If booked, it responds with an error message Slot is already booked. If the slot is available, the appointment is booked for that specific patient for the decided datetime and slot with the respective doctor.

Finally, My appointment endpoint is a GET method that will be used by patient to view the details of the appointment like the appointment ID, doctor's name, specialization ID, date, time, and status.

The delete appointment endpoint manages to delete the appointment based on the appointment ID. This endpoint can only be used by patients to cancel their appointment.

### Prescriptions

The get prescription is a GET method that fetches the prescription of a specific patient. For this endpoint, the patient ID has been used as a query parameter, and the endpoint is secured with JWT authentication, which ensures that only the authorized users can access the prescribed medicine.

The endpoint retrieves the prescription data from the Azure Cosmos DB, which is a NoSQL database. In order to retrieve the data from Cosmos DB, the setup of the database includes the Cosmos endpoint, Cosmos key, database name, and container name. In the database, the prescriptions are segregated based on the patient id.

### Buy Prescriptions

The buying of prescriptions was one of the crucial endpoints, as it included the Stripe integration. Firstly, the payment sessions (create-payment-session) endpoint checked whether the patient had done the payment or not with the help payment table. If the payment was not done, a Stripe checkout session was created, including the total amount. By default, session ID (stripe\_payment\_intent\_id) and payment status pending were stored in the payment table.

Upon successful payment, Stripe provides secure webhooks that confirm the payment status. If the payment is successful, the webhook sends a message that includes the session ID, prescription ID, stripe\_payment\_intent\_id and updates the payment status. The update payment status is a helper function that updates the payment table when the webhook sends the response; until that time, the payment status is displayed as pending. This ensures that the transactions made are secure using the JWT authentication and the Stripe signatures.

### Staff Routes/ API endpoints.

This section outlines the endpoints developed for the staff functionalities, including registration of staff and patients, prescribing medicine prescriptions, availability setting, and managing the booking appointment.

### Admin Routes

The admin registers the nurse and doctor with the help of the staff registration endpoint. In this process, the admin ID is verified. Upon successful verification of the admin, the helper functions are invoked: register\_doctor and register\_nurse. The admin then sends the details of the staff to the database and stores it. On the contrary, if the admin ID is missing or invalid, the backend API for staff and patient registrations will not be allowed or executed.

Moreover, the admin has the right to register the patients. During this process, the register patient API is being called. Like the staff registration endpoint, the admin is verified based on the admin ID. Upon successful validation, the admin enters the data and sends it to the database. However, if the admin ID is not verified or validated, the backend API will not be executed.

The API endpoints getPatient, getDoctor, and getNurse are developed to retrieve a list of patients, doctors, and staff. These endpoints could only be accessed by the admin when the admin email had been verified. Upon successful verification, the details are being fetched from the respective database: Get patients from the patient table, get doctors from the doctors table, and get nurses from the nurse table.

Furthermore, the admin has the authority to delete the patient and staff. During this stage, the admin is verified based on the email address, and the endpoints involved in this are deletePatient, deleteNurse, and deleteDoctor. The deletion logic is carried out based on their respective email addresses. Additionally, if the email address doesn't exist, the status code 404 which stands for Not found is responded.

### Doctor Manage Appointments

The endpoint get patient booking was developed for the doctor to check the appointment scheduled for a particular date. In this endpoint, the doctor is verified based on the email address (included in the JWT token). Upon successful verification, the appointments were retrieved from the database based on the doctor's ID with the date.

In addition, if the doctor wants to cancel a specific appointment due to uncertain circumstances, the cancel doctor appointment endpoint would be triggered. During this, the endpoint fetches the appointment details from the database, displays them to the doctor, and then the doctor selects the appointment to be canceled. Upon cancellation of the appointment, send\_cancellation\_email will be initiated. The send\_cancellation\_email helper function sends a structured email to the patient's email address. This ensures clear communication between the doctor and patient, where, if required, the patient could reschedule the appointment.

### Prescription

The doctor provides the prescription to the patient, which includes the medicine details, collection method, prescription date and doctor's email. As an initial step of this process, the doctor must verify the patient by entering the first name, last name and date of birth. To ascertain the patient's identity, the verify patient endpoint would be triggered, which validates the provided data against the patient's database. Upon successful verification, the doctor proceeds to issue the prescription.

During the process of prescribing the medicine, the medicine details along with their prices are extracted from the medicine table. The price of the medicine is mainly based on the type of patient: student/normal. If the patient is a student, each medicine price is calculated with a 10% discount.

The finalised prescription is then stored in Azure CosmosDB which is a NoSQL database. To retrieve the pharmacy and medicine details, the get\_pharmacy and get\_medicine endpoints are created, which helps us dynamically retrieve all additional information about medicines and pharmacies.

### Staff Authentication

The staff login endpoint is activated when the staff type is doctor, nurse, or admin. The credentials entered would be the password and email address. During the verification process, the verify\_staff helper function was invoked to check the content after and before a Upon verification of the credentials, the cookies are generated which hold the JWT and CSRF token. Conversely, if any email address or password is invalid, the backend will signal a status code 401.

The get staff profile endpoint is a GET method where the staff is verified based on the email address. After the verification has been completed, the endpoint executes the details of the staff irrespective of their type (nurse/doctor), including the name, email address, registration number, specialization, and phone number.

The get specialization endpoint extracted the specialization of doctor/nurse. This was retrieved with the help of specialization and the nurse specialization table.

### Staff Availability

In order for the doctor to set their availability, the set\_doctor\_availability endpoint is called. This endpoint sets the doctors' slots selected for a particular date and inserts them into the doctor availability database table. In case any of the selected slots is already set in the table, a status code 400 will be returned.

Likewise, the nurse can set the availability with the help of the set\_nurse\_availability endpoint. This endpoint initially examines if the selected slot already exists in the nurse availability table, and if so, displays a status code 400. Conversely, the selected slots are sent to the database along with the nurse\_id and the date. As a result, the message is returned as "Nurse Availability set successfully."

The staff (either nurse or doctor) has the option to cancel the availability that is previously set in the set\_availability endpoint.

To retrieve the availability set by nurse/doctor, the get availability endpoint is handled. This endpoint utilises the staff type and staff ID in order to fetch the availability from the doctor availability or nurse availability database table.

### Medical records.

The medical record endpoint allows the authorised patient to upload the medical files (prescriptions, X-rays and reports) and allows the doctor to view the uploaded files. The files are securely uploaded in the Azure Blob Storage, where each of the containers is linked with patient\_id.

Additionally, the connection string for the Azure Blob Storage has been configured, as it's necessary for secure communication.

The patients' records are protected with the JWT token, where each of the patients must be authenticated via JWT. Each patient is assigned a unique container along with an ID in blob storage during registration itself. The files can only be accessed once the patient is verified.

The upload medical record (/upload/<patient\_id>) endpoint is secured with JWT. In this process, the API queries the patient table with patient\_id to retrieve the Azure container name assigned to that patient. The patient then uploads the files based on the selected category and returns a success message.

The patient can view the uploaded medical documents with the get/patient/files endpoint. Like the /upload/<patient\_id> endpoint, the API queries the patient\_id to fetch the container\_name. The response received presents the file name along with the URL published by the blob to access the files.

The doctors are only restricted to accessing the medical records of the patient with the post/patient/medical records endpoint. During this process, the patient must be validated first with first name, last name and date of birth and If the patient is a valid user, only then the container name is fetched, which retrieves a list of files along with the URL. This helps the doctor to view the records without any complications.

## Frontend Development.

The main aim behind the frontend development is to design a user-friendly interface. This section provides a details frontend process, including the development process and the features of frontend application. This application is built using the React bootstrap, which helps to reuse the components and minimise the code duplication.

### Development Process

The frontend development is a web application that primarily focuses on developing a responsive and user-friendly interface. The high priority during this process was to ensure an interactive and simplified design, as there would be different age group patients accessing the website.

Starting the development with the navigation system, a key component that’s been responsible for managing the redirection between the pages/ screens. This component was reused within all the dashboards, but the naming tags were different, as each of the dashboards had different features. This system was the base component for the app’s functionality.

The Top Navigation bar is designed to be responsive for all types of screens and had the logout button. The logout button was highlighted in a white box background with black text. The layout for the nav bar was horizontal with a dark blue background, which made it highlight the button.

The code snippet has the logic of the Navbar. The integration of the hamburger icon was initiated, since this would be displayed for the small screens. This was done with the help of the react bootstrap class d-md-none, a predefined utility, that hides icon for larger and medium screen but displays for the small screen. The logout button was imported from the logout component, which had its own functionalities to handle the logout sessions smoothly.

A screen shot of a computer code

AI-generated content may be incorrect.

The left side panel is been designed as a unique reusable component, as it would be used across the dashboard. The background colour of the panel is dark blue, with white text and an icon .

The code snippet utilizes the React props which means the props could be reused through the dashboards. A React hook (useEffect) has been defined inorder to handle the smaller screen. When clicked on the hamburger icon as defined in the top navbar, the side panel would open and display a list of options for navigation. In order to improve the usability, implementation of the event listener are been done. This would help to close the left panel whenever the user clicks outside, as this would ensure easy interaction for mobile devices.

Going ahead towards the icons, they are defined in the code, and the names HAVE been mapped as Appointment booking, Prescriptions, Medical Records and Profile. The icons are being imported from react icons fa and styles are been applied.

The Nav.Link is been defined, which has several props defined, including id, href, onClick, originalName and displayName, plus the style. The key difference between the original name and the display name is that the names defined for icons are mapped to the original name. For instance, in the patient dashboard, the left panel has a button called Appointment booking. Inorder to display an appropriate icon, the original name defined should be referred to.

Additionally, the role-based restrictions are been implemented inorder to control the access of functionalities. The nurses are restricted from using certain features, due to which the implementation of the link.disabled property takes place. When the link is disabled, the React bootstrap classes are used to style the opacity, transparency, and aria disabled.

### Login Page

The login screen is the start the application, where the users can authenticate themselves. It has two input fields named email and password. The user enters the details and upon clicking submit, the API endpoint verifies sends the data to the database to check if the details are correct. If the details are matched, the tokens are generated which are stored in the cookies and the patient is navigated to the Dashboard. In case of incorrect details or network problem, the error is been displayed which helps the user to understand the issue.

Additionally, there has been two buttons namely Register here and Staff login.

Appendix , displays the entire implementation of the login screen along with the layout and the navigations. The styling has been done with the help of React bootstrap classes and is made responsive for all screen types.

Fig. displays the image of the login page.

### Register Page

The register page helps the new users to create an account. It includes fields as first name, last name, date of birth, gender, contact number, email address, street address, postcode, city and password. All of these fields have the tag required.   
The register form is been divided into two halves named step1 and step2. Unless the step 1 is completed one cannot click the NEXT button. Once entered all the details in step1, the Next button activates and the step 2 form is been seen. Upon successfully entering all the details, the Register button is been clicked. If incase, one wants to check the details before clicking Register button, a Back button is been implemented which helps the user to go back to step 1. This setup enables a user-friendly interface with clear information been conveyed to the user along with buttons.

After successfully registering, the user is navigated to the login screen and the data is been sent to the database with the help of API call and been stored successfully.

Appendix, shows the implementation of register page, where the register form is been imported as a component along with the login.css file plus the image being imported from the assets.

Fig, shows the register page.

### Staff Login Page

The staff login page authenticates the staff and the admin to login with their credentials. It includes input fields as email address and password.

The staff submits their credentials, where the system sends data to the database to verify the credentials and upon successful validation, the tokens are generated and navigated to the dashboard. The validation of the staff type is been done based on the email address. Additionally, the page has a highlighted text with a link saying “Are you a patient? Login here”. This has been implemented, if incase any of the patient clicks on the staff login.

Appendix, shows the implementation of the register page, including the login form components and the responsiveness achieved with the React bootstrap.

Fig, shows the register page.

### Patient Dashboard Page

The Patient Dashboard page is displayed upon successful login. This is the central point of contact for the patients to access the functionalities.

The dashboard includes the top navbar and left side navbar/panel. The top navbar displays the logout button and the left side navbar contains pages such as appointment booking, medical records, prescriptions and profile.

Each of the pages are implemented with the React component and work independently according to their functionalities.

In this dashboard, the components are being placed inside the React Bootstrap Containers, as it helps to organize the alignment of the content. To achieve the responsiveness, the rows and columns are being defined, to ensure smooth transition between the screen size.

By default, the my\_appointment API is triggered upon successful login and the patient views the scheduled appointment. Th appointment is viewed inside the React Card, which has a cancel button. If incase due to any circumstances the patient is not able to make it for the appointment one can cancel it. Therefore, this makes the patient access the functionalities without much of hassle and redirection through the website.

Appendix. Shows the implementation of the Patient Dashboard page.

Fig. displays the page

### Appointment Booking Component

The booking appointment components has a short form imported from the BookignAppointmentForm component. This form includes short description, selection of the type of disease. If incase of minor injuries, articles are being displayed.

The components is divided into four steps

Step 1: The patient enters the details in the form and selects the type of disease. During the selection of disease, the get disease API is executed.

Step 2: A calendar is displayed where the patient has to opt for the convenient date to schedule the appointment.

Step 3: The specialized doctor for the selected disease is been displayed for the date opted by the patient, if available. During this the get\_doctors\_list API is triggered.

Step 4: Lastly, the time slots available for doctor are displayed. After selecting the time slot, the appointment is scheduled.

The scheduled appointment is then viewed on the patient’s dashboard with my\_appointment API call, if required the patient can also cancel the appointment with the help of cancel button.

### Prescriptions Component

The prescription component has been designed to view the prescribed medications by the doctor. Upon clicking Prescription button, API call is triggered, where the patient id is retrieved from the local storage which helps to view the prescriptions details on the system.

The prescription includes doctor's email, medicine prescribed, collection method, and a payment button displaying the total cost.

To initiate the payment, a Stripe API has been implemented where the API endpoints get triggered upon clicking the pay now button and is directed to the payment page. The Stripe key is provided in the component, which is retrieved from stripe dashboard, which generates secure payment.

The prescriptions are displayed inside the React Card. If, in case, the patient has multiple prescriptions, a vertical scrolling is applied.

### Medical Records Component

The medical records component's aim is to upload the medical-related documents. The patient firstly selects the type of the folder ( X-ray, Prescription, Reports and other) and uploads the file where the upload API call is triggered. If incase the patient selects the wrong file, the remove button is triggered. The uploaded files are viewed in the table format where the API call is executed(patient/files?patient\_id=${patient\_id}).

### Profile Component

The profile component fetches the GET API call (patientProfile), which displays the data in the form of a card. The details displayed on the system are first name, last name, date of birth, gender, email address, and phone number, with a read only constraint.

### Staff Dashboard Page

The staff (doctor and nurse) are able to view the Staff dashboard page, upon successful login. The dashboard has the top navbar and the left navbar which are imported as a component. For the left navbar there’s a condition that has been implemented, if the staff type is nurse the features prescriptions, medical records and patient bookings won’t be accessible. On the contrary, the doctor would have the permit to access all the functionalities including the appointment, medical records, prescription and profile.

In this dashboard, components such as SetDoctorAvailability, StaffProfile, GetPatientBooking, providePrescription and ViewMedicalHistroy has been imported. Each of the pages have their own functionalities and work independently. All these components are placed inside the React Container, with a padding style.

Additionally, the rows and columns are been defined for better visuals and alignment of the website.

### Set Availability Component

The component was developed to set the availability for the doctor/nurse. Firstly, the staff type along with their ID is retrieved from the local storage. In case due to any error the staff type or ID isn’t being fetched, the error message “Please Login again” is displayed.

To set the availability, the slots were defined. The official working hours configured were from 9:00 AM to 5:00 PM, where each of the slots were divided into 30-minute time intervals. A constraint applied to availability is that, it can only be set two months prior.

The staff then selects the slots and receive a modal that would display “Are you sure you want to add the availability?” with two buttons, Cancel and Confirm. The modal has been imported named SetAvailabilityModal, where the modal has a header, body, and footer with the buttons. This would help the staff to set the availability without any hassle. The added availability would then be sent to the database with the help of the API call (set\_doctor\_availability / set\_nurse\_availability) and stored successfully.

The availability set by the staff would be displayed under the set availability section. Additionally, in case due to any unforeseen circumstances the availability needs to be cancelled, a cancel button has been implemented, which would help the staff to cancel the availability without any hassle. This has been handled by the API calls, and the database has been updated with the current changes.

### Get Patient Booking

The component main aim is to fetch the bookings done for the particular date and doctor. The doctor selects a date, which then retrieves the appointment scheduled and the system displays the appointments.

The data is being displayed in the form of a table including the header and columns name. Each of the column’s name is mapped to the table body having stripped layout. The table body displays the patient’s name(first and last name), appointment id, disease name, appointment start time and end time.

If incase, the doctor wanted to cancel the appointment, a delete button was added where upon cancellation an email would be sent to the patient.

The API calls handled in this component are get patient bookings and delete the appointment. Further, if any appointments were cancelled, the appointment table instantly gets updated. Additionally, for a particular date if there aren’t any appointments scheduled there a message displayed “No appointments found for this date.”.

### Prescription Component

The prescription component was designed to prescribe the medicine to the patient. Along with the prescription component, the pharmacy form plus prescription form component was introduced and were integrated into the provide prescription component.

The provide prescription component starts by verifying patient where the doctor enters first name, last name and date of birth. Upon clicking verify, the patients details including the address are retrieved and displayed on the system. Once verified, the doctor clicks on provide prescription button which displays the prescription form component (Step ==2) and lastly the (step ==3) pharmacy form component .

The prescription form component displays the patient information as the patient’s name , date of birth and the address. The form is been displayed on the system which has fields such as select medicines, quantity, instructions, patient type and collection method. When the doctor selects the medicines from the drop down, the API call is been performed the get medicines and then further the quantity and instructions are provided on when to consume them. Additionally, the doctor can prescribe multiple medicines to a patient. A key point in is that the type of patient that’s either student or normal plus the collection method (In store or home delivery)of the medicine has been introduced here. This helps the patient to get the medicine at their home without any time invested to travel inorder to buy the medicine.

The pharmacy form component, display a form where the doctor selects the pharmacy from the dropdown which is populated with the help of API call. Upon selecting the pharmacy according to the Patients choice, the doctor then sends the prescription to the pharmacy using the API endpoint(providePrescription). Additionally, an alert with a green variant is displayed which includes the prescription Id, upon successful submission of the prescription.

### Medical Records

The medical record component was introduced inorder to view the medical documents uploaded by the patients. To view the medical records, the doctor firstly need to enter the details of the patients that’s first and last name and date of birth inorder to check if any medical records are uploaded. If uploaded, a list of folders with the respective files are being displayed. This has been achieved with the help of the post API call where the medical records are fetched based on the patients id. Inorder to access the files, a URL link is provided where the doctor can click and view the documents. Furthermore, if no files are uploaded a text has been displayed “no files in this folder.”

### Profile

The profile component displays the details of the staff such as first name, last name, registration number, specialization, email address and phone number. This has been achieved with get API call for the StaffProfile. The details seen on the system has a readOnly constraint where the doctor cant incorporate any changes.

### Admin Dashboard Page

The admin dashboard page has restricted access only to the admin. The dashboard also has the left navbar and top navbar. The left navbar includes the functionalities Add Staff, Add Patient, Book Appointments, and Get the List of Patients and Staff.

The components used in this dashboard are being imported as separate React components, which are placed inside the React container. Moreover, the rows and columns are being defined not only for better user experience but also for the dashboard to be responsive across various devices. However, each of the components utilized in this page has been explained below.

### Manage Bookings Component

The appointment booking component is the same component implemented in the patient dashboard with a minor addition. The form has introduced an input field labelled: Enter the patient ID. The would require the admin look for patient's ID in the patient list page and schedule the appointment for patient accordingly.

### Add Staff Component

The admin has the key right to register the staff that are nurses or doctors. The system displays the form where the admin firstly selects the staff type, including the first name, last name, specialization, email, phone, registration number, and password. After entering the details, the admin clicks on the add staff button, which triggers the register staff API, and the data is sent and stored in the database.

### Add Patient Component

The admin has the right to add the patients. The admin needs to fill the form with patients’ details, including first name, last name, email, phone, password, date of birth, gender, street address, city, and postcode. Upon entering all the details, the admin clicks the Add Patient button, which triggers an API call, where the data is sent to and stored in the database.

### Patient List Component

The patient list component helps the admin to view the list of registered patients. The patients are filtered based on their email address and first name. The list of patients is being displayed in a table format with the columns named ID, name, email, gender, DOB, and phone, which have been retrieved with the help of an API call.

Further, there’s an additional column named Action where each of the rows has the remove button. Upon clicking "remove, the patient will be deregistered from the general practitioner clinic, and the delete API call will be initiated.

### Staff List Component

The staff list components help the admin to view the staff that are currently registered. The admin has two options to search the staff: firstly, by entering the name in the search box, and secondly, by selecting the role that’s either nurse or doctor, where the details are retrieved by the GET API call.

In case any of the staff is currently not working at the healthcare facility, the admin can click on the remove button, which would trigger the API call to delete the staff, and a delete confirmation modal would be displayed with the header and the body message, ensuring that the admin confirms the deletion of the staff.

# Testing.

## API Testing (Postman).

The testing of the API with the help of postman would be done and screenshot would be shown in the Appendix.

## System Testing.

In this section, the testing of the entire system would be done where the test cases would be written.

## Usability testing.

# Deployment.

This section provides an in-depth description of the deployment done on Microsoft Azure leveraging Azure resources.

## Architecture.

A simple cloud-based architecture is designed to ensure security, maintainability and scalability across the services. The big three cloud providers currently in the market are Microsoft Azure, Amazon Web Services, and Google Cloud Platform, along with IBM Cloud, Oracle Cloud, Salesforce, DigitalOcean and many more. Thus, Microsoft Azure was the selected cloud provider for this project.

The figure illustrates the system components, such as frontend , backend, and database (SQL and NoSQL), are loosely coupled but interconnected with of the services. In this deployment process, the frontend is deployed within the Azure App Service, as it helps to run the application without the burden of managing the underlying architecture. Similarly, the backend is deployed in Azure Container Apps since it enables reduced infrastructure maintenance and cost savings for containerised applications. Additionally, the backend communicates with Azure SQL, Azure Cosmos DB, Blob Storage and Azure Communication Services for email, which is achieved with the help of the connection string provided by Azure services that is then set in the environment file of the backend.

In terms of security, cross-origin resource sharing is implemented to ensure that the server accepts and responds to the authorised frontend application. This methodology ensures that only authenticated and authorised requests are accepted, which would prevent malicious cyber attacks. Additionally, along with CORS, the JWT and CSRF tokens were generated for a secure communication between the backend server and frontend application. The CORS would only accept the request sent by *thehealme.com*; this domain is purchased from GoDaddy.

## Azure Services Used.

The Azure services used throughout this project are as follows:

Azure Container App

The Azure Container App is a serverless platform, mainly used for the deployment of backend (Flask AP) applications. The container app eases the management with auto-scaling, security management, monitoring and flexibility in deployment.

Azure App Service

The Azure App Service is mainly used for the deployment of React frontend applications. The Azure App Service is a platform that runs the application, abstracting away the infrastructure. Additionally, this service supports various web stacks such as .NET, Java (in Java SE, Tomcat, and JBoss flavours), Node.js, Python, and PHP and runs either on Linux or Windows.

Azure SQL

Azure SQL is a fully managed relational database-as-a-service (DBaaS). Moreover, Azure SQL is a managed SQL server database engine which falls under the platform as a service. Moreover, SQL Server is a built-in functionality and feature that requires extensive configuration. Therefore, this service is mainly used for the SQL database queries in this application.

Azure Cosmos DB

The Azure Cosmos DB is a NoSQL database and is primarily used to store the prescription details. The Cosmos DB simplifies the development process of the application much faster in a single database.

Azure Communication Service.

Azure Communication Service is used for the notification purpose, which will send an email to the patients about the cancellation of the appointment. Additionally, the service supports diverse communication formats.

Azure Storage Accounts

The Azure Storage service is Microsoft's cloud storage solution for modern data storage scenarios. The Azure storage offers durability, high availability, scalability, accessibility, and security for storing the data in the cloud. This service is primarily used to store the medical documents of patients, ensuring easy accessibility from anywhere.

## 6.3 Deployment Process.

The below-mentioned steps are the deployment process of the application.

Step 1:

The backend and frontend were partitioned into separate components.

The frontend should only send the request to the origin: https://api.thehealme.com.

The environment (.env) file was implemented in the backend in order to ensure that backend should connect and communicate to the mentioned origins only. This reinforced the backend's defences against the threats. The .env file included linkage for the storage account, Cosmos DB, SQL, Communication Service for Azure, Container Apps and App Service.

Step 2

The Docker image is built locally for both the frontend and backend, and the images were generated .

Initially, the frontend Docker image is built, and the command is:

*docker build -t gpsystemfrontend:v6 .*

The command includes the image name along with the tag.

Subsequently, the backend Docker image is built, and the command is :

*docker build -t gpsystemapi:v8 .*

The command includes the image name along with the tag.

Step 3

Once the images are successfully generated, the following step is to authenticate the Azure. The command to log in is:

az login.

Upon the execution of the command, a pop-up is displayed where the user has to select the appropriate Azure account. The chosen account is then utilised for the deployment process, which ensures that the further commands executed align with the proper subscription and resources.

Step 4:

This process entails logging in to the Azure portal and setting up the Azure Container App and Azure App Service.

The container app is configured mainly to require the subscription name along with the Azure resource group name. Further, the instance details and pricing plans were set up. During the instance configuration, the publish offered options: code and container, and the container was chosen to deploy the application.

Moreover, the container configurations included the selection of the image source, which in our case was Azure Container Registry. Upon selecting Azure Container Registry, the registry name was selected, admin rights were provided, and the image name along with the tag was selected.

After successfully creating the Azure Web Apps, a custom domain was added. In this process the domain provider was selected, and TLS/SSL certificates were managed along with the TLS/SSL type. Additionally, the domain name thehealme.com was added, which further requested the domain validation that was done at the GoDaddy website under the DNS section. After successful validation, the certificate was added.

Step 5:

The Docker image was tagged to the Azure Container Registry domain, where the Azure registry name, along with the image name and tag. The command executed was:

docker tag gpsystemfrontend:v6 acrgpsystem.azurecr.io/gpsystemfrontend:v6

docker tag gpsystemapi:v9 acrgpsystem.azurecr.io/gpsystemapi:v9

Step 6:

The Docker image further was pushed to the Azure Container Registry. The command employed was:

docker push acrgpsystem.azurecr.io/gpsystemfrontend:v5

docker push acrgpsystem.azurecr.io/gpsystemapi:v5

The Azure dashboard would be needed inorder to select and apply the up-to-date image for the application.

In terms of frontend Firstly, choose the Azure App service called FrontendGpSystem, assuming we are in the Azure Resource Group (rg\_gp\_uk). After choosing the app service, the Deployment section appears in a left panel. To view the application deployment details, choose the Deployment Centre inside the deployment. Here, we select and save the revised picture tag. This procedure assures that the right code is being retrieved and displayed.

Conversely, for the backend application, select the Azure App Container named gpsystemukcont, assuming we are in the Azure Resource Group (rg\_gp\_uk). After selecting the container app, the Application section appears in the left panel. To view the application deployment details, choose the Containers inside the application section. Here, we should select the revised image tag and save. Therefore, this confirms that the correct code is being retrieved and displayed.

In order to check if the image is successfully pushed to the Azure container registry, the Azure container registry was accessed through the Azure dashboard. The left panel of the Azure Container Registry had a section named "Services". Upon selecting services, the repository was selected from the drop-down, which displayed the Docker image's name. As the frontend and backend were pushed independently, both of the images' names were displayed. Additionally, to check if the correct tag along with the version was pushed, you can select the image name, and this will display the details of the tags pushed until now.

The Azure Container Apps is configured mainly for Flask API backend applications, which include details of the subscription and resource group. In this process the container app environment was newly created, as the environment would secure the container apps.

The deployment source chosen for this application is the container image. The container setups primarily involved selecting the image source and container name. Azure Container Registry was chosen as the image source. This option incorporates the image, tag, registry, and subscription.

The new custom domain api.thehealme.com has been added. During this process, the domain validation was essential. Azure performed the validation since it gave information to fill in the DNS area at GoDaddy. Upon successful verification, the domain was added. Additionally, the certificate is secured with https.

# Results.

## Achievements.

The outcomes that the project has led to are outlined below:

Successful integration of the backend and frontend using React and Flask API.

Cross-site request forgery (CSRF) and JWT tokens (cookies) have been used to ensure the secure communication within the frontend and backend application. Additionally, CORS has been implemented as an additional layer of security.

The dashboards for admin, doctors/nurses and patients were designed. Each of them was accessed based on their role, ensuring smooth navigation.

The application is deployed on the Azure cloud. Azure Cloud services like Azure Web Apps, Azure Container Apps, Storage Accounts, Azure Cosmos DB, and Azure SQL are used during the deployment process.

The medical documents uploaded by the patients are stored securely in the Azure Blob Storage, linking them to patient\_id.

The application is responsive for all types of screens. This has been achieved using React Bootstrap's classes and has been tested across mobile devices.

Stripe payment integration is implemented to securely purchase the medicines prescribed by the doctor, and webhooks were used to track the status of payment.

The application's overall usability and navigation are quite straightforward, ensuring simple, user-friendly website navigation.

The Azure SQL was used to store the structured database, whereas the Azure Cosmos DB was used to store the NoSQL database, mainly for the prescriptions.

The frontend follows a scalable, modular, and clear and understandable design with reusable React components and integration of an API. This guarantees that additional features could be implemented in the future easily without affecting the previous features.

## Lesson Learned.

The lessons learnt from this project are mentioned below:

Importance of a literature survey of the existing applications and technology used for developing the application.

Gained hands-on experience in developing the full-stack application using the React and Flask API (Python) as tech stacks.

Learnt about the significance and implementation of JWT tokens, CSRF and handling the application with the cookies.

Grasped the knowledge regarding CORS and implemented it by purchasing a domain from GoDaddy.

Designing the database for both SQL and NoSQL with the help of ER diagrams.

As a fresher, I learnt how to debug the error faced during the development and deployment process.

Became familiar with the deployment process on Microsoft Azure and also explored the services provided by Azure. Some of the services were used during the application deployment for better understanding and gaining knowledge.

The importance of testing after each endpoint was developed.

Learnt how to design a user-friendly interface using React Bootstrap classes.

Understood the colour psychology before designing the website.

## Limitations

In this section, the limitation would be of the project would be explained.

# Conclusion.

## Future Scope.

In this section, the future scope that would be explained.

## References

Begum, S., & C. P. Indumathi. (2016). *ER diagram based web application testing.* Madurai, India: IEEE. doi:10.1109/ICCIC.2015.7435786

Khan, Z. (2023). The Emerging Challenges and Strengths of the National Health Services: A Physician Perspective. *Cureus*. doi:10.7759/cureus.38617

MacConnachie, V. (2024, May 28). *Is it impossible to see a GP?* Retrieved from NHS Confederation: https://www.nhsconfed.org/articles/it-impossible-see-gp

*Medical staffing in the NHS.* (2025, March 28). Retrieved from BMA: https://www.bma.org.uk/advice-and-support/nhs-delivery-and-workforce/workforce/medical-staffing-in-the-nhs#:~:text=The%20high%20rate%20of%20doctors,their%20organisation%20values%20their%20work.

*Medical staffing in the NHS.* (2025, April 24). Retrieved from BMA (British Medical Association): https://www.bma.org.uk/advice-and-support/nhs-delivery-and-workforce/workforce/medical-staffing-in-the-nhs

Mohammad Mehrtak, S. S. (2021, Jul-Aug). *Security challenges and solutions using healthcare cloud computing.* doi: 10.25122/jml-2021-0100

Nasaruddin, N. S., & Izzatdin Abdul Aziz. (2018). *Web-Based Electronic Healthcare Record System (Ehrs) Based on Feedback.* Langkawi, Malaysia: IEEE. doi: 10.1109/AINS.2018.8631427

Shuaibu, M. B., & Ruqayyat Ahmad Ibrahim. (2017). *Web application development model with security concern in the entire life-cycle.* Salmabad, Bahrain: IEEE. doi:10.1109/ICETAS.2017.8277849

Song, I.-Y., Mary Evans , & E.K. Park. (1995). A Comparative Analysis of Entity-Relationship Diagrams. *Journal of Computer and Software Engineering*, 427 - 459. Retrieved from chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cin.ufpe.br/~in1008/aulas/A%20Comparative%20Analysis%20of%20Entity-Relationship%20Diagrams.pdf

# Appendix.

## 10.1 Background Research

A screenshot of a computer

AI-generated content may be incorrect.

Literature Review 10.1‑10‑a: Major points that remain the same for the approaches.

A screenshot of a computer

AI-generated content may be incorrect.

Literature Review 10.1‑10‑b: Use Case Diagram for Staff Admin.

A diagram of a company structure

AI-generated content may be incorrect.

Literature Review 10.1‑10‑c: Web application Development Model with security Concerns

A diagram and diagram of a diagram

AI-generated content may be incorrect.

Literature Review 10.1‑10‑d: Use Case Diagram for Doctor Admin.

A screenshot of a document

AI-generated content may be incorrect.

Literature Review 10.1‑10‑e: Use Case diagram for patient.

A diagram of a diagram of a cloud computing system

AI-generated content may be incorrect.

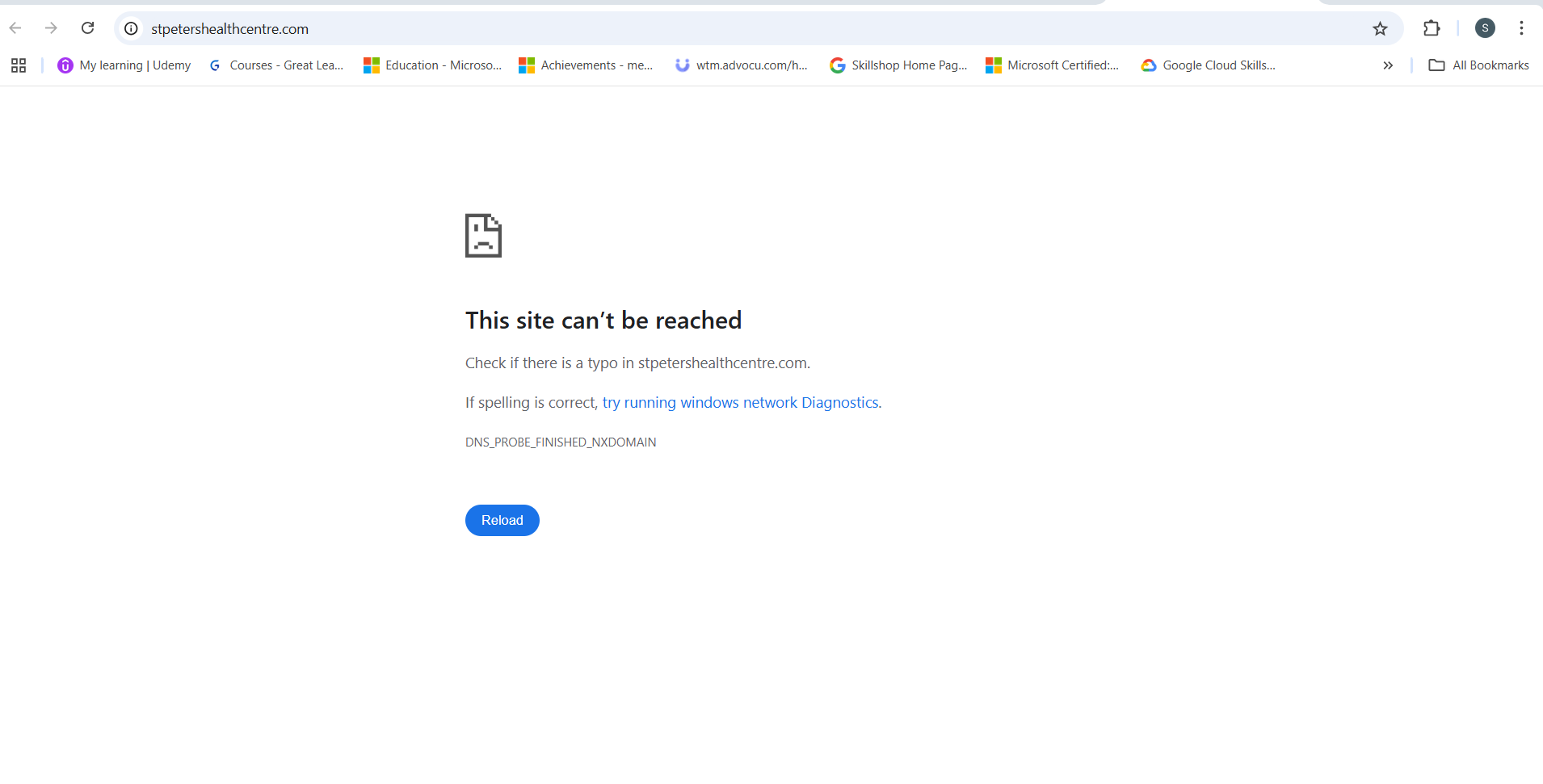
Literature Review 10.1-10‑f: Proposed System Architecture.

A diagram of a patient

AI-generated content may be incorrect.

Literature Review 10.1-10‑g: Use Case Diagram

## 10.2 Existing Websites



Existing Websites 10.2 - a: Website 1



Existing Websites 10.2 - b: Website 2

