

# **PROJECT REPORT**

## **DEPARTMENT OF COMPUTER SCIENCE**

### **SREE NARAYANA COLLEGE CHERTHALA**

**Affiliated to University of Kerala  
NAAC Re-Accredited 'A+' Grade**



### **MEDICAID**

**NITHIN M N**

**Reg. No: 32022131033**

**SREERAJ S**

**Reg. No: 32022131038**

**VISHNU RAJ K R**

**Reg. No: 32022131043**

**SUBMITTED IN PARTIAL FULFILMENT OF REQUIREMENT FOR THE AWARD  
OF**

**BSC COMPUTER SCIENCE DEGREE OF UNIVERSITY OF KERALA**

**2025**



# **SREE NARAYANA COLLEGE, CHERTHALA**

Affiliated to University of Kerala

NAAC Re-Accredited with 'A+' Grade

## **BSC COMPUTER SCIENCE 2022 - 2025**

### **CERTIFICATE**

This is to certify that the project work entitled “**MEDICAID**” is Bonafide report of the work done by **NITHIN M N, SREERAJ S, VISHNU RAJ K R** in partial fulfilment of the requirements for the award of the Bachelor’s degree in BSc. Computer Science of the institution during the year 2022-2025.

Staff in charge

Head of Department

Principal

Presented for the viva voce examination conducted by the University of Kerala held on..... /.../2025 at Sree Narayana College, Cherthala & verified by:

Date:

Internal Examiner

External Examiner

## DECLARATION

We hereby declare that the project report entitled “**MEDICAID**” is a record of project work done under the guidance of Aswathy K, Assistant Professor on contract & Chinnu K Deepu, Assistant Professor on contract, Department of Computer Science, Sree Narayana College, Cherthala.

We also declare that this report has not been submitted to any other University or Institute for the award of any fellowship or Degree or Diploma.

Place: Cherthala

Names: NITHIN M N

Date:

SREERAJ S

VISHNU RAJ K R

## ACKNOWLEDGMENT

This project marks our inaugural endeavour in developing a web application, and we are pleased to report that the resultant software aligns closely with our initial objectives. Throughout the project's duration, we received invaluable support and guidance from our peers, family members, and faculty advisors, to whom we extend our sincere gratitude. Their contributions have been instrumental in facilitating the successful completion of this project.

We are esteemed to thank, The Principal **Dr. T.P. BINDU**, Sree Narayana College, Cherthala for granting us permission to do the project.

We convey our sincere thanks to **Mr. ANSU A T (Head of the Department of Computer Science)**, **Mrs. Aswathy K (Assistant Professor on contract, Department of Computer Science)**, **Mrs. Chinnu K Deepu K (Assistant Professor on contract, Department of Computer Science)** for their valuable instructions and guidance to the project and their apt encouragement towards the successful completion of our project.

NITHIN M N

SREERAJ S

VISHNU RAJ K R

## ABSTRACT

In today's digital era, healthcare management demands seamless, efficient, and user-friendly solutions to enhance accessibility and patient care. Traditional methods of handling medical appointments, prescriptions, and diet recommendations are often inefficient and time-consuming. To address these challenges, this AI-powered healthcare platform is designed to integrate patients, doctors, and administrators into a unified system. It streamlines medical consultations, report analysis, and personalized health recommendations, ensuring a smooth and effective healthcare experience for all users.

The admin module plays a crucial role in managing doctors and patients, overseeing appointment bookings, and monitoring diet plan recommendations. Doctors can efficiently handle patient appointments, provide prescriptions, and offer medical advice tailored to individual health conditions. Patients benefit from a comprehensive suite of features, including registration, appointment booking, and the ability to upload and analyze medical reports. Additionally, a personalized diet planning system ensures that patients receive nutritional guidance based on their health conditions, promoting better wellness outcomes.

Beyond its core functionalities, the platform introduces an AI-powered voice chatbot that offers real-time medical guidance, making healthcare more interactive and accessible. Patients can engage with the chatbot to receive instant support on health-related queries, enhancing their overall healthcare experience. By leveraging advanced digital solutions, this platform bridges the gap between patients and healthcare providers, offering a more efficient, transparent, and patient-centric approach to medical management.

# CONTENTS

INDEX NO	CONTENTS	PAGE NO
1	INTRODUCTION	1
1.1	EXISTING SYSTEM AND DISADVANTAGES	1
1.2	PROPOSED SYSTEM AND ADVANTAGES	2
1.3	PROJECT PROFILE	3
1.4	PROJECT OVERVIEW	4
2	REQUIREMENT ENGINEERING	5
2.1	REQUIREMENT ELICITATION	6
2.1.1	FEASIBILITY STUDY	6
2.1.1.1	OPERATIONAL FEASIBILITY	6
2.1.1.2	TECHNICAL FEASIBILITY	6
2.1.1.3	ECONOMICAL FEASIBILITY	7
2.1.2	REQUIREMENT DEFINITION	7
2.1.2.1	FUNCTIONAL REQUIREMENTS	7
2.1.2.2	NON FUNCTIONAL REQUIREMENTS	9
3	DEVELOPMENT ENVIRONMENT	11
3.1	HARDWARE REQUIREMENTS	12
3.2	SOFTWARE REQUIREMENTS	12

<b>3.2.1</b>	<b>HTML</b>	<b>13</b>
<b>3.2.2</b>	<b>CSS</b>	<b>13</b>
<b>3.2.3</b>	<b>JAVASCRIPT</b>	<b>13</b>
<b>3.2.4</b>	<b>PYTHON DJANGO</b>	<b>14</b>
<b>3.2.5</b>	<b>SQLite3</b>	<b>14</b>
<b>4</b>	<b>SOFTWARE DESIGN</b>	<b>14</b>
<b>4.1</b>	<b>SYSTEM DESIGN</b>	<b>14</b>
<b>4.2</b>	<b>DATA FLOW DIAGRAM</b>	<b>15</b>
<b>4.3</b>	<b>ER DIAGRAM</b>	<b>20</b>
<b>4.4</b>	<b>USE CASE DIAGRAM</b>	<b>21</b>
<b>4.5</b>	<b>ACTIVITY DIAGRAM</b>	<b>22</b>
<b>4.6</b>	<b>SEQUENCE DIAGRAM</b>	<b>23</b>
<b>4.7</b>	<b>PROGRAM LIST</b>	<b>24</b>
<b>4.8</b>	<b>DATABASE DESIGN</b>	<b>25</b>
<b>4.9</b>	<b>INPUT DESIGN</b>	<b>31</b>
<b>4.10</b>	<b>OUTPUT DESIGN</b>	<b>32</b>
<b>5</b>	<b>CODING</b>	<b>33</b>
<b>6</b>	<b>SOFTWARE TESTING</b>	<b>47</b>
<b>6.1</b>	<b>TESTING STRATEGY</b>	<b>47</b>

<b>6.1.1</b>	<b>UNIT TESTING</b>	<b>47</b>
<b>6.1.2</b>	<b>INTEGRATION TESTING</b>	<b>48</b>
<b>6.1.3</b>	<b>SYSTEM TESTING</b>	<b>48</b>
<b>6.2</b>	<b>TEST CASES</b>	<b>48</b>
<b>7</b>	<b>IMPLEMENTATION</b>	<b>52</b>
<b>8</b>	<b>SCREENSHOTS</b>	<b>53</b>
<b>9</b>	<b>CONCLUSION</b>	<b>57</b>
<b>10</b>	<b>FUTURE ENHANCEMENT</b>	<b>58</b>
<b>11</b>	<b>BIBLIOGRAPHY</b>	<b>59</b>
<b>11.1</b>	<b>TEXT REFERENCES</b>	<b>59</b>
<b>11.2</b>	<b>ONLINE REFERENCES</b>	<b>59</b>
<b>12</b>	<b>APPENDIX</b>	<b>60</b>
<b>12.1</b>	<b>GANTT CHART</b>	<b>60</b>



## LIST OF FIGURES

Figure 1-level 0 DFD.....	16
Figure 2-level 1 DFD DOCTOR.....	17
Figure 3-level 1 DFD ADMIN.....	18
Figure 4-level 1 DFD PATIENT.....	19
Figure 5-ER DIAGRAM.....	20
Figure 6-USE CASE DIAGRAM.....	21
Figure 7-ACTIVITY DIAGRAM.....	22
Figure 8-SEQUENCE DIAGRAM.....	23
Figure 9-LOGIN PAGE .....	53
Figure 10-HOME PAGE.....	53
Figure 11-DOCTOR REGISTRATION PAGE.....	54
Figure 12-PATIENT REGISTRATION PAGE.....	54
Figure 13-BOOKING PAGE.....	55
Figure 14-VIEW DOCTOR PAGE.....	55
Figure 15-FEEDBACK LIST PAGE.....	56

## GANTT CHART

Figure 16-Jan 03,2025.....	61
Figure 17-Jan 08,2025.....	63
Figure 18-Jan 10, 2025.....	65
Figure 19-Jan 29, 2025.....	67
Figure 20-Feb 07, 2025.....	69
Figure 21-Feb 14, 2025.....	71
Figure 22-Feb 21, 2025.....	73
Figure 23-Feb 27, 2025.....	75
Figure 24-Mar 05, 2025.....	77

## LIST OF TABLES

Table 1–Program list.....	24
Table 2-Admin.....	25
Table 3-Doctor.....	26
Table 4-Patient.....	27
Table 5-Booking.....	28
Table 6-Feedback.....	29
Table 7-Medicine.....	29
Table 8-Prescription.....	30
Table 9-Result Analysis.....	31

# 1. INTRODUCTION

## 1.1 EXISTING SYSTEM AND DISADVANTAGE

The current health-care management system primarily relies on manual or semi-digital processes, which are often inefficient, time-consuming, and prone to errors. Patients typically book appointments through phone calls or in-person visits, leading to scheduling conflicts and delays. Medical records and reports are usually maintained in physical files or basic digital databases, making retrieval and analysis cumbersome. Personalized diet plans are rarely automated, requiring multiple consultations. In addition, patients lack instant medical guidance, as traditional systems do not incorporate AI-driven support. These limitations create inefficiencies in patient care, increased administrative workload, and delays in medical decision-making, highlighting the need for a more integrated, AI-powered solution to enhance accessibility and streamline health-care processes.

1. **Manual Appointment Scheduling** – Patients often book appointments via phone calls or in-person visits, leading to scheduling conflicts and delays.
2. **Lack of AI-driven Medical Assistance** – No instant AI chat-bot support for medical guidance, requiring patients to rely on consultations for basic queries.
3. **Inefficient Medical Report Management** – Medical records are stored physically or in basic databases, making retrieval and analysis time-consuming.
4. **Limited Personalization in Diet Plans** – Diet recommendations are often generic and require multiple consultations, lacking automation based on patient health conditions.
5. **Increased Administrative Workload** – Managing doctors, patients, and appointments manually leads to inefficiencies and errors in record-keeping.
6. **Delayed Medical Decision-Making** – Without AI-based analysis, diagnosing health conditions and recommending treatments take longer.
7. **Poor Patient Engagement** – Lack of digital interaction options limits accessibility and convenience for patients seeking health-care services.

## 1.2 PROPOSED SYSTEM AND ADVANTAGES

The proposed system is an AI-powered health-care management platform designed to overcome the inefficiencies of traditional healthcare systems by integrating patients, doctors, and administrators into a single digital solution. The system enables automated appointment scheduling, digital medical report analysis, and AI-driven personalized diet planning based on patient health conditions. Administrators can efficiently manage doctors, patients, and appointments, while doctors can provide prescriptions and medical recommendations seamlessly. Patients benefit from a user-friendly interface that allows them to register, book appointments, upload and analyze medical reports, and receive personalized healthcare guidance. A key feature of the system is the AI-powered voice chatbot, which offers real-time medical support, enhancing accessibility and patient engagement. By leveraging automation, AI, and digital health solutions, this platform aims to streamline healthcare management, improve patient care, and reduce administrative workload, ensuring a more efficient and interactive healthcare experience.

1. **Automated Appointment Scheduling** – Reduces waiting times and eliminates scheduling conflicts, ensuring a smooth booking process.
2. **AI-Powered Medical Assistance** – Provides real-time medical guidance through a voice-enabled chatbot, improving accessibility for patients.
3. **Efficient Medical Report Management** – Allows patients to scan, upload, and analyze medical reports digitally, enhancing diagnosis and record-keeping.
4. **Personalized Diet Planning** – Generates customized diet plans based on patient health conditions, improving overall wellness.
5. **Seamless Doctor-Patient Interaction** – Enables doctors to manage appointments, prescribe medications, and provide medical recommendations efficiently.
6. **Reduced Administrative Workload** – Automates patient and doctor management, minimizing manual effort and human errors.
7. **Faster and Informed Decision-Making** – AI-driven analysis helps doctors make quicker and more accurate medical decisions.
8. **Enhanced Patient Engagement** – Digital tools and AI chatbots ensure better interaction and continuous support for patients.
9. **Improved Accessibility** – Patients can access healthcare services remotely, reducing the need for frequent hospital visits.

**10. Data Security and Transparency** – Ensures safe storage and easy retrieval of medical records while maintaining patient confidentiality.

### **1.3 PROJECT PROFILE**

Title : MEDICAID

Type : Python Django, C++

Objective : AI powered health care management system

Duration : 3 Months

Internal Guide : Mrs. Aswathy K

: Mrs. Chinnu K Deepu

Project Team : NITHIN M N

SREERAJ S

VISHNU RAJ K R

## 1.4 PROJECT OVERVIEW

The AI-powered healthcare management system is designed to address inefficiencies in traditional healthcare processes by integrating patients, doctors, and administrators into a single, digital platform. The system enables automated appointment scheduling, digital medical record management, and AI-driven personalized diet planning based on individual health conditions. Administrators can efficiently oversee doctor and patient management, while doctors can handle appointments, provide prescriptions, and offer medical recommendations. Patients benefit from features such as medical report scanning and analysis, personalized healthcare suggestions, and interactive AI-driven medical assistance. This seamless integration enhances efficiency, reduces administrative workload, and ensures a more structured and patient-centric healthcare experience.

One of the most innovative aspects of the system is the AI-powered voice chatbot, which offers real-time medical guidance, making healthcare services more accessible, interactive, and efficient. By leveraging automation, AI, and digital health solutions, the platform streamlines healthcare management, improves patient engagement, and accelerates medical decision-making. Additionally, the system enhances data security, transparency, and accessibility, ensuring that medical records are securely stored and easily retrievable.

## 2. REQUIREMENT ENGINEERING

Requirement engineering is a critical phase in the development of traditional healthcare management systems face several challenges that impact efficiency, patient care, and accessibility. Manual appointment booking, inefficient medical report management, and the lack of AI-driven support often lead to delays in medical services, miss communication, and increased administrative workload. Patients struggle with long waiting times, lack of personalized healthcare recommendations, and difficulty in accessing instant medical guidance. Additionally, medical records are typically stored in physical or outdated digital formats, making retrieval, analysis, and diagnosis slower and more complex. These limitations hinder effective healthcare delivery, creating a need for a more integrated, automated, and intelligent healthcare solution.

The primary objective of this project is to develop an AI-powered healthcare management system that integrates patients, doctors, and administrators into a single digital platform. The system aims to automate appointment scheduling, provide secure medical record management, and generate personalized diet plans based on health conditions. It seeks to enhance doctor-patient interactions by enabling digital prescriptions, real-time health monitoring, and AI-assisted diagnosis. A key feature of the platform is the AI-powered chatbot, which offers instant medical guidance through voice interaction, improving accessibility and engagement for patients.

Furthermore, the project focuses on reducing administrative workload, improving healthcare decision-making, and ensuring data security. By implementing AI-driven automation and intelligent healthcare solutions, the system aims to streamline medical processes, enhance patient care, and improve overall healthcare accessibility. The ultimate goal is to bridge the gap between patients and healthcare providers, creating a faster, more efficient, and patient-centric healthcare experience.

## **2.1 REQUIREMENT ELICITATION**

Requirements elicitation or Requirements gathering focuses on the objectives of the system.

### **2.1.1 Feasibility Study**

Feasibility Study is conducted to determine whether the proposed system is feasible. Now it is running in manual system, in which all processing is done manually. Lots of stress and manpower is needed here. Implementing the proposed system can eliminate failures of the current system which saves money and time. Thus, the proposed system is economically feasible. The new system can support in the later versions of windows operating system. The proposed system is technically feasible. The new system is also behaviorally feasible as the person who is currently doing the student detail process can reduce his works by implementing the proposed system and the state of his/her job will not change.

#### **2.1.1.1 Operational Feasibility**

Proposed project is beneficial only if it can be turned into information systems that will meet to the organizations operating requirements. Simply stated, this test of feasibility ask if the system will work when it is developed and installed Here are questions that will help test the operational feasibility of a project.

Is there sufficient support for the project from management from users? If the current system is well liked and used to the extent that persons will not be able to see reasons for change, there may be resistance are the current business methods acceptable to the user. If they are not, Users may welcome a change that will bring about a more operational and useful systems Have the user been involved in the planning and development of the project.

#### **2.1.1.2 Technical Feasibility**

Evaluating the technical feasibility is the trickiest part of a feasibility. This is because, at this point of time, not too many detailed designs of the system, making it difficult to access issues like performance, cost on (account of the kind of technology to be deployed) etc. A number of issues have to be child's details that provides efficient and faster methods that



simplifies the task of managing the information about the child. This software provides easy and efficient access of the details including their considered while doing a technical analysis.

Understanding the different technologies involved in the proposed system before commencing the project we have to be very clear about what are the technologies that are to be required for the development of the new system. Find out whether the organization currently possesses the required technologies.

### **2.1.1.3 Economical Feasibility**

Economic Feasibility attempts to measure the cost of developing and implementing a new system, against the benefits the accrue from having the new system in place, this feasibility study gives the top management the economic justification for the new system.

A simple economic analysis which gives the actual comparison of costs and benefits are much more meaningful in this case. In addition, this proves to be a useful point of reference to compare actual costs as the project progresses. There could be various types of intangible benefits on account of automation. These could include increased customer satisfaction, improvement in services quality better decision-making timeliness of information, expediting activities, improved accuracy of operations, better documentation and record keeping faster retrieval of information, better employee morale.

### **2.1.2 Requirement Definition**

In software engineering, a requirement definition is a clear statement of what a software system must do, have, or achieve to meet a specific need or goal. These requirements can be categorized into two types:

- Functional requirements: Describe what the system should do
- Non-functional requirements: Define how the system should perform or behave

#### **2.1.2.1 Functional Requirements**

Functional requirements describe what the software system should do, i.e., the features and functionality that it should provide.

1. **User Management:** Users can register, log in, and manage their profiles.

- Pet owners can add multiple pets to their profile with details like breed, age, and medical history.
2. **Pet Health Monitoring:** Users can track their pet's health, including vaccinations, medical history, and upcoming appointments.
    - The system provides automated reminders for vaccinations, deworming, and vet check-ups.
  3. **Appointment Scheduling:** Users can book, reschedule, or cancel veterinary appointments.
    - Veterinary professionals can manage appointment slots and availability.
  4. **Diet and Exercise Recommendations:** The system suggests personalized diet plans based on pet breed, age, and health conditions.
    - Exercise recommendations help pet owners maintain their pet's physical fitness.
  5. **AI Chatbot for Pet Care Queries:** The chatbot provides instant responses to common pet care questions.
    - It assists in basic health assessments and offers guidance on nutrition, behavior, and emergency care.
  6. **Pet Medical Records Management:** The system stores medical records, prescriptions, and diagnostic reports.
    - Veterinarians can update health records after each consultation.
  7. **Notifications and Alerts:** Users receive reminders for upcoming vaccinations, medication schedules, and appointments.
    - Emergency alerts notify pet owners of critical health conditions.
  8. **Feedback and Review System:** Users can provide feedback on veterinary services.

- Ratings and reviews help improve the quality of service.
9. **Secure Data Storage:** The system securely stores user and pet data using an SQLite database.
    - User authentication ensures only authorized access to sensitive pet health records.
  10. **Offline Functionality:** Pet care information and medical history can be accessed offline.
    - AI chatbot can provide limited responses without an internet connection.

### 2.1.2.2 Non-Functional Requirements

Non-functional requirements describe how the software system should behave, i.e., performance, security, reliability, and usability requirements.

1. **Performance Requirements:** The system should provide a fast response time for user interactions, with AI chatbot replies within 2 seconds.
  - It should handle multiple users simultaneously without significant delays.
2. **Scalability:** The system should support an increasing number of users, pets, and stored medical records without performance degradation.
  - Future updates should allow for the integration of additional features like IoT-based pet tracking.
3. **Security Requirements:** User authentication is required for accessing personal and pet-related data.
  - Sensitive data, including medical records, should be encrypted to ensure privacy and protection against unauthorized access.
4. **Usability:** The interface should be user-friendly, intuitive, and accessible to both tech-savvy and non-tech-savvy users.

- The system should follow a simple navigation structure to make pet management easy for users.
5. **Reliability & Availability:** The system should ensure high availability, functioning at least 99% of the time without downtime.
- Backup mechanisms should be in place to prevent data loss.
6. **Maintainability & Adaptability:** The system should be designed for easy updates and bug fixes.
- The SQLite database should be structured to allow smooth migration to more advanced databases if needed in the future.
7. **Data Integrity:** The system must ensure accurate and consistent data storage for pet health records, appointments, and notifications.
- No unauthorized modification of critical pet health data should be allowed.
8. **Portability:** The system should be accessible on multiple devices, including desktops, tablets, and mobile phones, without losing functionality.
9. **Compliance Requirements:** The system should comply with data protection laws to ensure user privacy and security.
- It must follow veterinary regulatory guidelines for storing and managing pet health records.
10. **Efficiency:** The system should be optimized to run on low-resource devices, ensuring smooth functionality without excessive CPU or memory consumption.
- Database queries should be optimized for quick retrieval of information.

## 3. DEVELOPMENT ENVIRONMENT

### 3.1 HARDWARE REQUIREMENTS

- PROCESSOR : i3 or above
- STORAGE : 256 GB or above
- MEMORY : 8 GB RAM or above

### 3.2 SOFTWARE REQUIREMENTS

- OPERATING SYSTEM : Windows, Linux
- FRONT END : HTML, CSS, JavaScript
- BACK END : Python Django, SQLite3
- DATABASE : SQLite3
- BROWSER : Google Chrome, Mozilla Firefox, Microsoft Edge
- WEB SERVER : Django
- CODE EDITOR : Visual Studio Code

### 3.2.1 HTML

HTML (Hypertext Markup Language) is the backbone of the World Wide Web, serving as the standard markup language for creating web pages and web applications. It provides the structure and layout of content on the internet, using tags to define different elements such as headings, paragraphs, images, links, and more. HTML is not a programming language but rather a markup language that works alongside CSS (Cascading Style Sheets) and JavaScript to create dynamic and visually appealing web experiences. With its simple syntax and widespread adoption, HTML is accessible to beginners while offering powerful capabilities for professional developers. Its continuous evolution, marked by updates and new standards, ensures that it remains relevant in the ever-changing landscape of web development. Mastering HTML is essential for anyone looking to build websites or contribute to the digital realm.

### 3.2.2 CSS

CSS (Cascading Style Sheets) is a fundamental technology for web design, allowing developers to control the visual presentation of HTML documents. With CSS, designers can specify styles for elements like fonts, colours, layout, and spacing, creating visually appealing and consistent web pages across different devices and browsers. It separates the structure (HTML) from the presentation (CSS), enhancing accessibility and maintainability. CSS operates on a cascade principle, where styles are applied in a hierarchical manner, allowing for both general rules and specific overrides. Its flexibility and modularity enable efficient styling of complex web layouts, facilitating responsive design for mobile devices and dynamic user interfaces. Moreover, CSS frameworks like Bootstrap extend its capabilities, providing pre-designed components and utility classes to streamline development workflows and enable rapid prototyping. In essence, CSS, along with frameworks like Bootstrap, is indispensable for crafting engaging and user-friendly web experiences.

### 3.2.3 JAVASCRIPT

JavaScript is a versatile programming language primarily used for web development. Initially designed to make web pages interactive, it has evolved into a full-fledged language capable of building complex web applications, server-side applications, and

even mobile apps. JavaScript is supported by all modern web browsers, making it a ubiquitous tool for front-end development. Its dynamic nature allows developers to manipulate webpage content, respond to user actions, and create dynamic effects seamlessly. With the advent of frameworks like React, Angular, and Vue.js, JavaScript has become even more powerful, enabling developers to build robust and responsive user interfaces. JavaScript's versatility, coupled with its large ecosystem of libraries and frameworks, makes it an indispensable tool for modern web development.

### **3.2.4 PYTHON DJANGO**

Python Django is a popular open-source web framework that simplifies the development of web applications using the Python programming language. It provides a high-level, reusable, and scalable architecture for building robust and secure web applications. Django follows the Model-View-Template (MVT) pattern, allowing developers to easily separate the data models, application logic, and user interface components of their web applications. Django also includes a comprehensive set of tools and features for tasks such as routing, authentication, data manipulation, form handling, and more. Django's built-in admin interface offers a convenient way to manage the backend of web applications. With its extensive documentation, active community, and vast ecosystem of third-party libraries, Django enables rapid development and deployment of web applications with enhanced security and scalability. Overall, Python Django is a powerful and versatile web framework that simplifies the development of web applications in Python, making it a popular choice among developers for building robust and scalable web applications.

### **3.2.5 SQLite3**

SQLite3 is a lightweight, self-contained, serverless, open-source relational database management system. Designed for simplicity, efficiency, and reliability, it's widely used in embedded systems, mobile devices, and small-scale applications. With its small memory footprint and zero-configuration requirement, SQLite3 is easy to deploy and manage. It operates using a single disk file, making it convenient for applications that need local data storage without the complexity of a client-server model. Despite its simplicity, SQLite3 supports most of the SQL92 standard and provides features like ACID (Atomicity, Consistency, Isolation, Durability) transactions, triggers, and views.

## 4.SOFTWARE DESIGN

### 4.1 SYSTEM DESIGN

System design is the process of defining the architecture, components, interfaces, and data for a system to satisfy specified requirements. For the web application, we employed object-oriented analysis and design methods to translate the system requirements into operational specifications.

#### Logical Design

During the logical design stage, we focused on defining the system's architecture, components, and interfaces. We identified the following key components:

- User interface
- Application logic
- Database management

We also defined the interfaces between these components, ensuring seamless communication and data exchange.

#### Physical Design

In the physical design stage, we focused on the detailed implementation of the system. We selected the following technologies and tools:

- Frontend: HTML, CSS, JavaScript
- Backend: Python Django, SQLite3
- Security: HTTPS protocol, encrypt for password hashing, and data encryption, google authentication

#### Characteristics of the System

The designed system exhibits the following characteristics:

1. Security: Ensures data confidentiality, integrity, and authenticity
2. Practicality: Meets the functional requirements and user needs
3. Efficiency: Optimizes system resources and performance



4. Acceptability: Provides a user-friendly interface and experience
5. Flexibility: Allows for future enhancements and scalability
6. Economy: Minimizes development and maintenance costs
7. Reliability: Ensures consistent and dependable system operation

## 4.2 DATA FLOW DIAGRAM

A DFD is a graphical representation of the flow of data through a system, showing the inputs, processing, storage, and outputs.

### BASIC DFD SYMBOLS

#### 1. ARROW



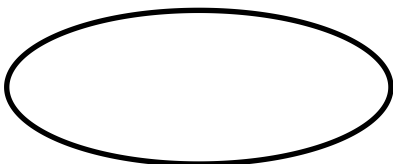
A data flow is a route, which enables packets travel from one point to another. Data may flow from a source to a processor and from data store or process. An arrow line depicts the flow, with arrow head pointing in the direction of the flow.

#### 2. RECTANGLE



A process represents transformation where incoming data flows are changed into outgoing data flows.

#### 3. ELLIPSE



A data store is a repository of data that is to be stored for use by a one or more process may be as simple as buffer or sophisticated as relational database. They should have clear names.

#### 4. SIDE OPEN RECTANGLE



A source or sink is a person or part of an organization which enter or receives information from the system but is considered to be the context of data-flow model.

### LEVEL 0 CONTEXT DIAGRAM

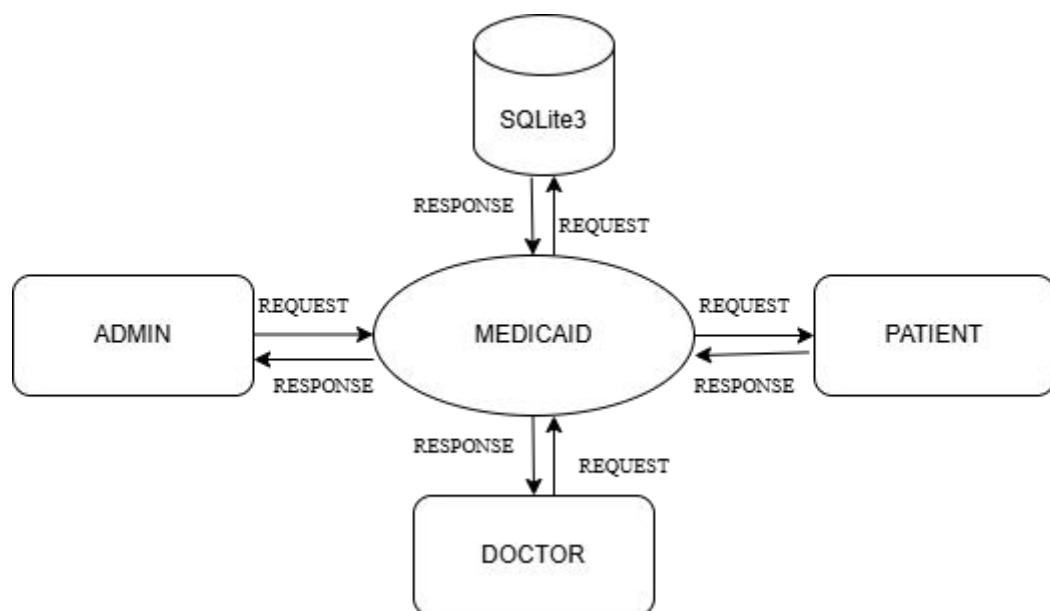


Figure 1-level 0 DFD

## LEVEL 1 DFD DOCTOR

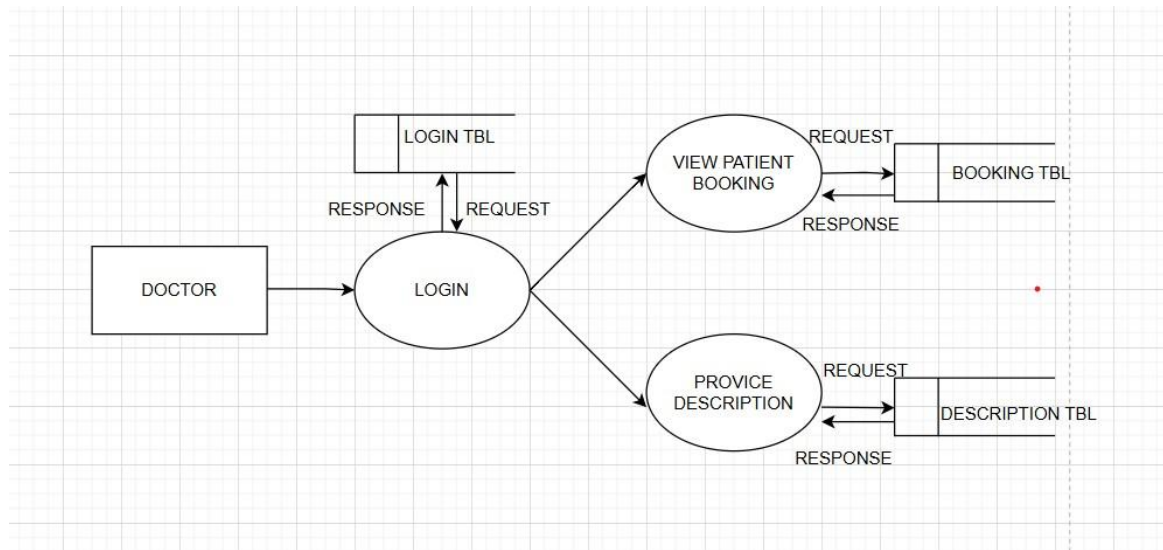


Figure 2–level 1 DFD DOCTOR

## LEVEL 1 DFD ADMIN

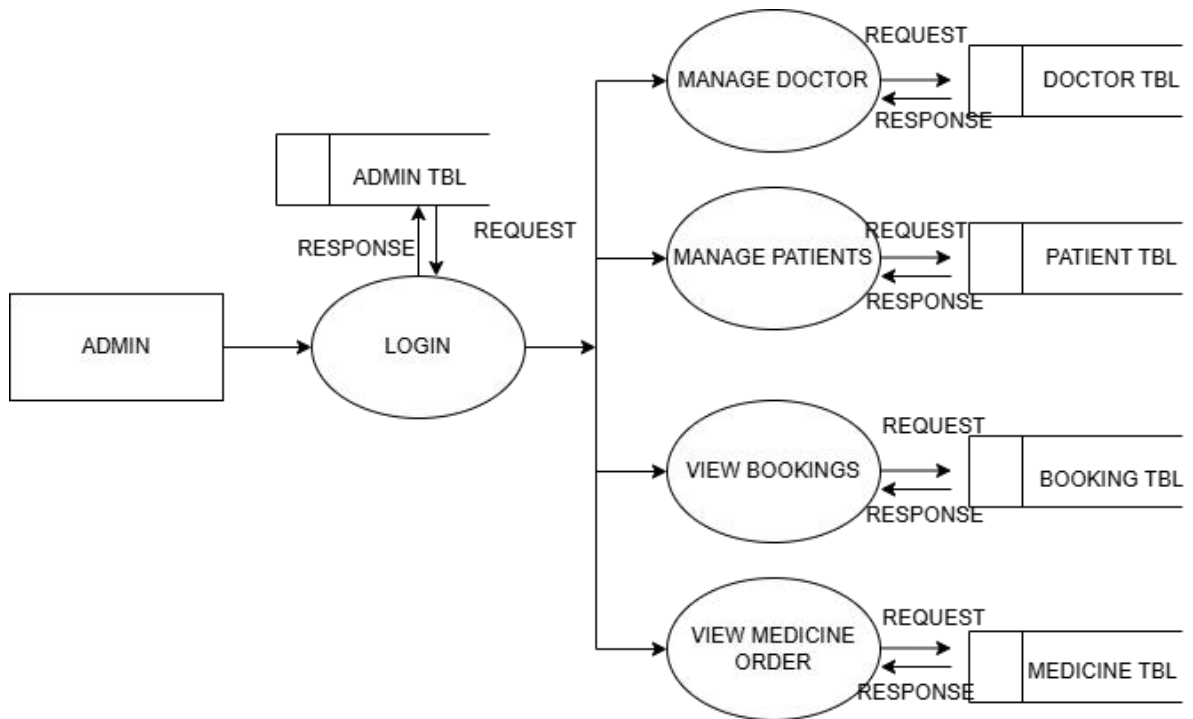


Figure 3–level 1 DFD ADMIN

## LEVEL 1 DFD PATIENT

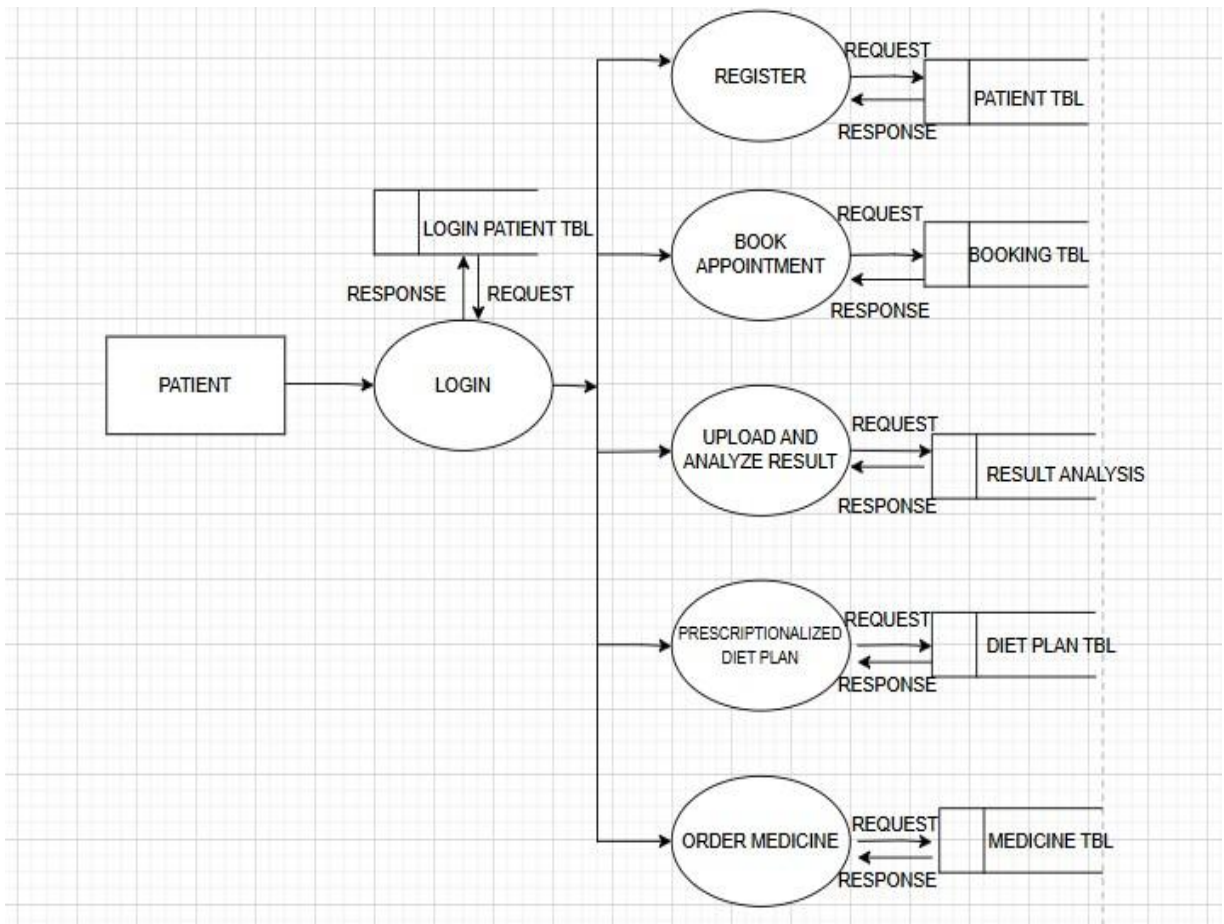


Figure 4–level 1 DFD PATIENT

## 4.3 ENTITY RELATIONSHIP DIAGRAM (ER DIAGRAM)

An Entity-Relationship (ER) diagram is a visual representation of the structure and relationships in a database. It's a tool used in database design to graphically illustrate the relationships between entities (tables) and their attributes (columns).

ER diagrams typically consist of:

1. Entities: Represented by rectangles, entities are the tables in the database.
2. Attributes: Represented by columns, attributes are the individual pieces of data stored in each entity.
3. Relationships: Represented by lines, relationships show how entities interact with each other.

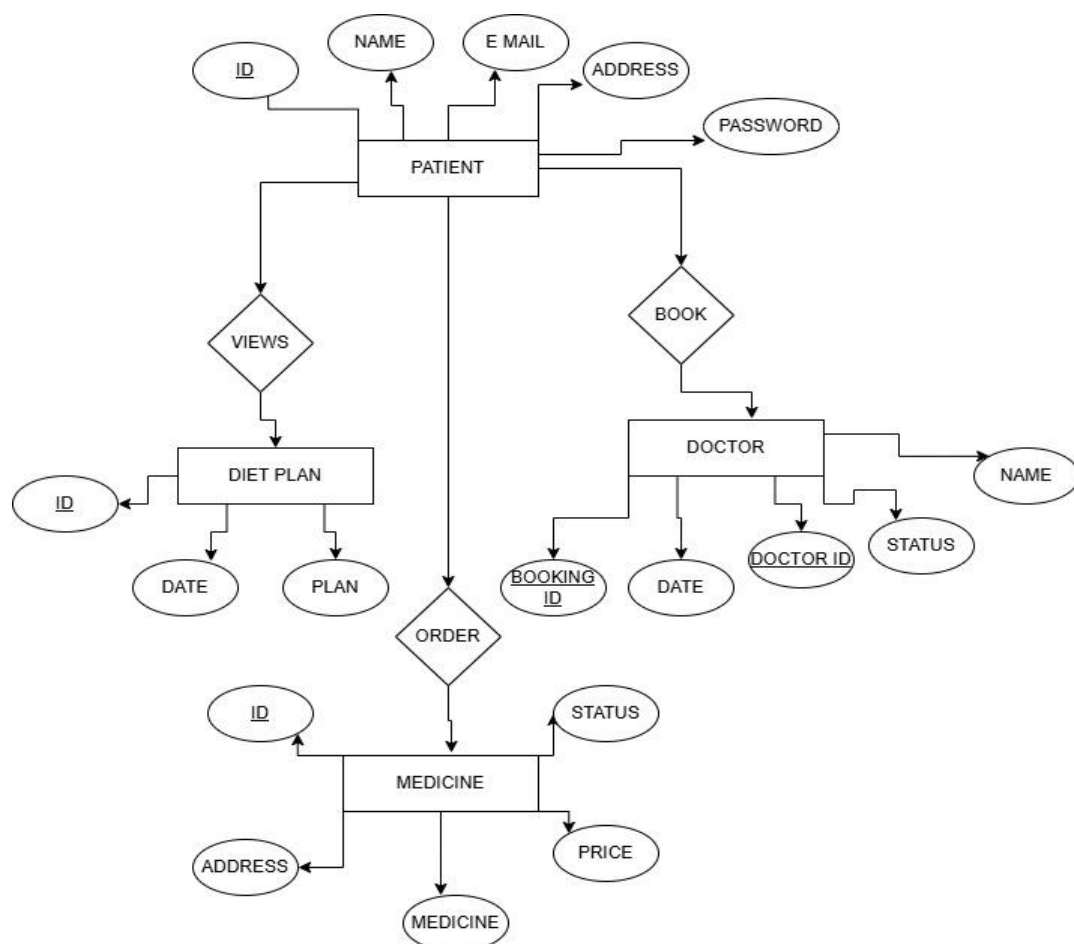


Figure 5-ER DIAGRAM

## 4.4 USE CASE DIAGRAM

A use case diagram is a way of visualizing the interactions and relationships between the users and the system. It shows the different types of users, the use cases they can perform, and how they are connected. A use case diagram can help to understand the functional requirements of a system, and to design a system from the user's perspective.

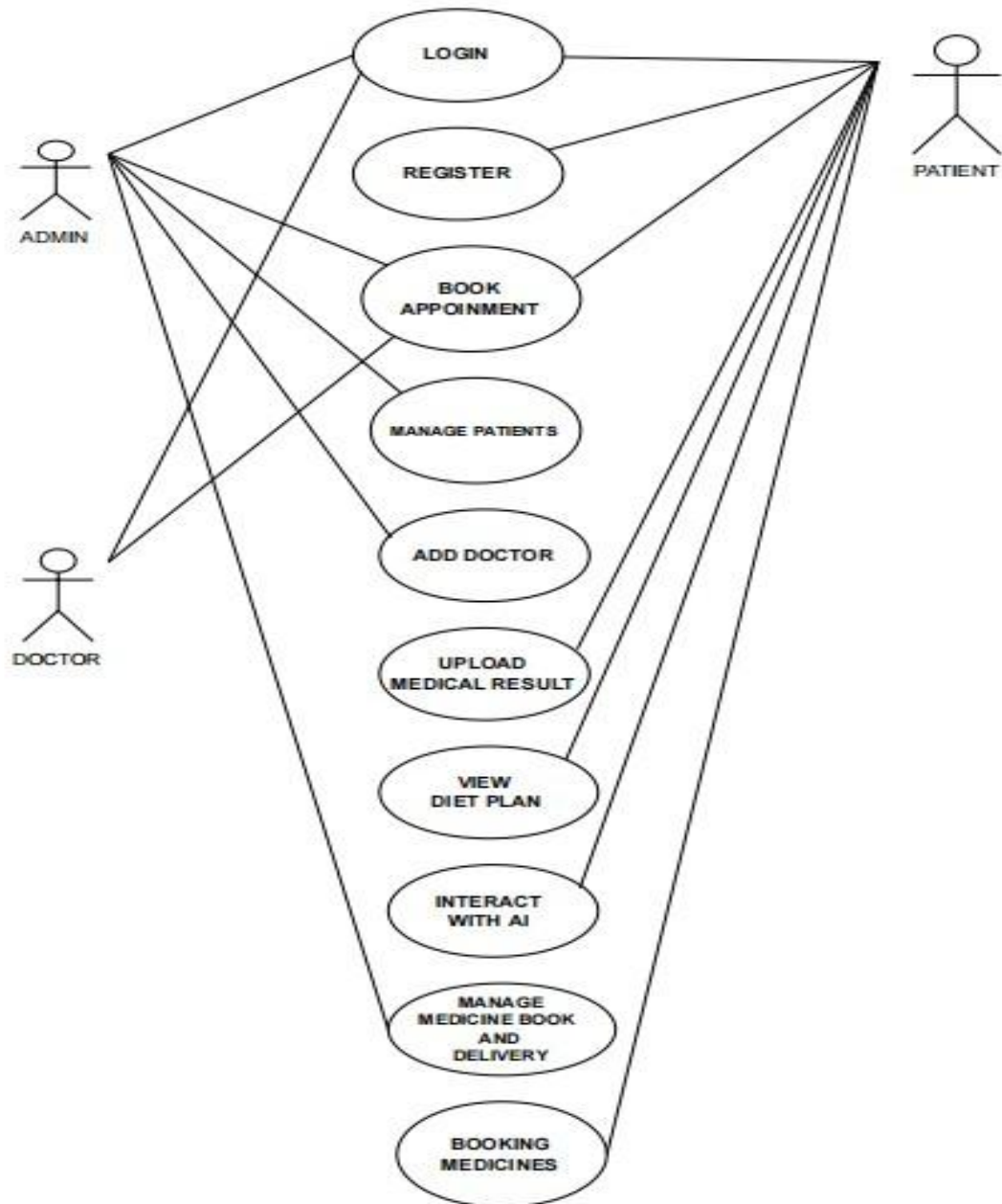


Figure 6-USE CASE DIAGRAM

## 4.5 ACTIVITY DIAGRAM

An activity diagram is a way of visualizing the dynamic behaviour of a system or a process. It shows the flow of control and data from one activity to another, as well as the choices, iterations, and concurrency that may occur. An activity diagram can help you understand how a system works, what are the inputs and outputs, and what are the possible outcomes.

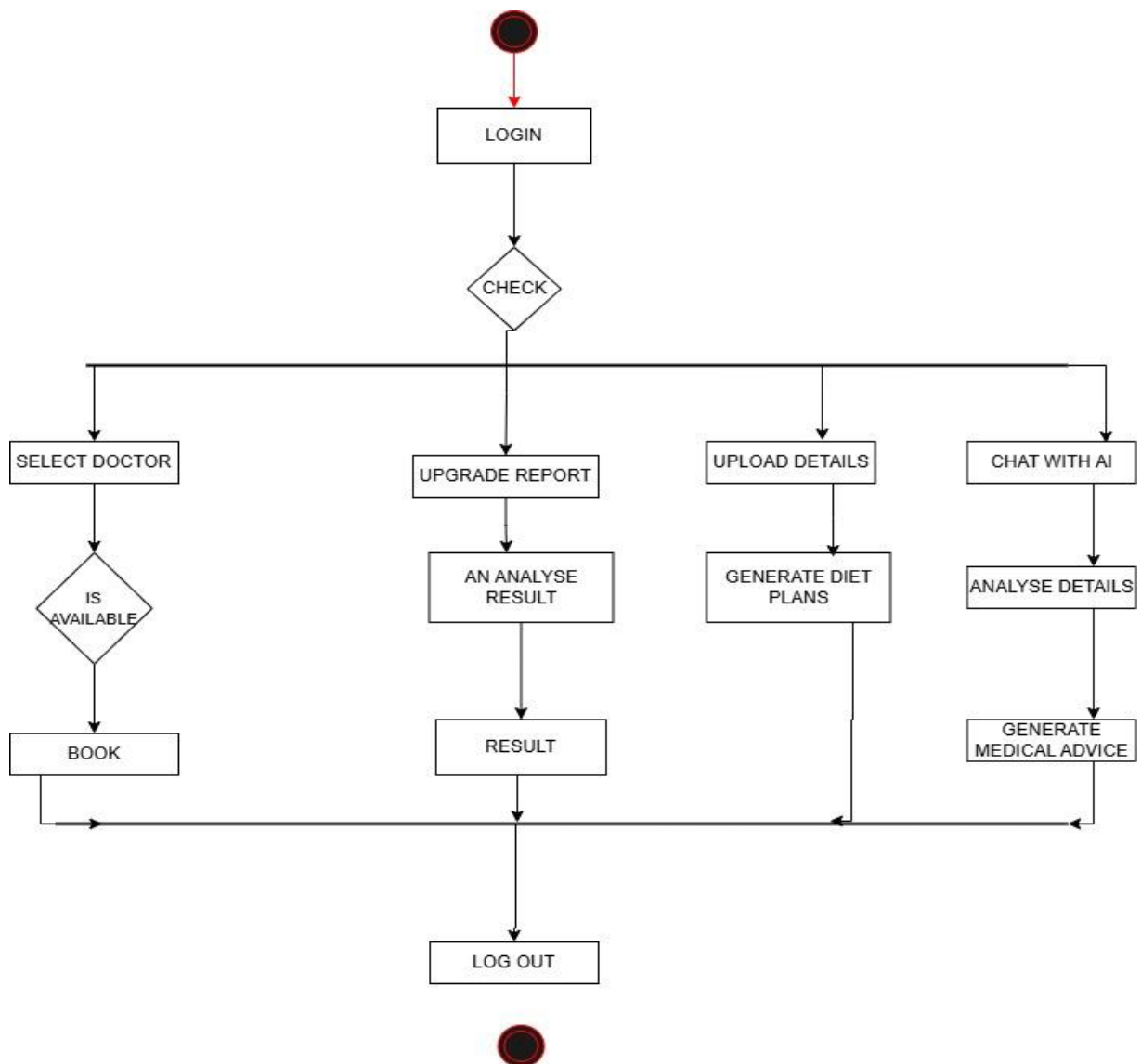


Figure 7-ACTIVITY DIAGRAM



## 4.6 SEQUENCE DIAGRAM

A sequence diagram is a UML diagram that visualizes interactions between system components over time, emphasizing the order of messages exchanged. It uses lifelines (vertical lines representing objects or actors), activation bars (showing active processing periods), and messages (arrows for communication like method calls or data transfers). Messages can be synchronous (solid arrows, waiting for a response) or asynchronous (dashed, non-blocking). These diagrams simplify workflows, such as user authentication or API requests, by mapping step-by-step interactions. They bridge technical specifications and implementation, helping validate system design and clarify dependencies for developers and stakeholders.

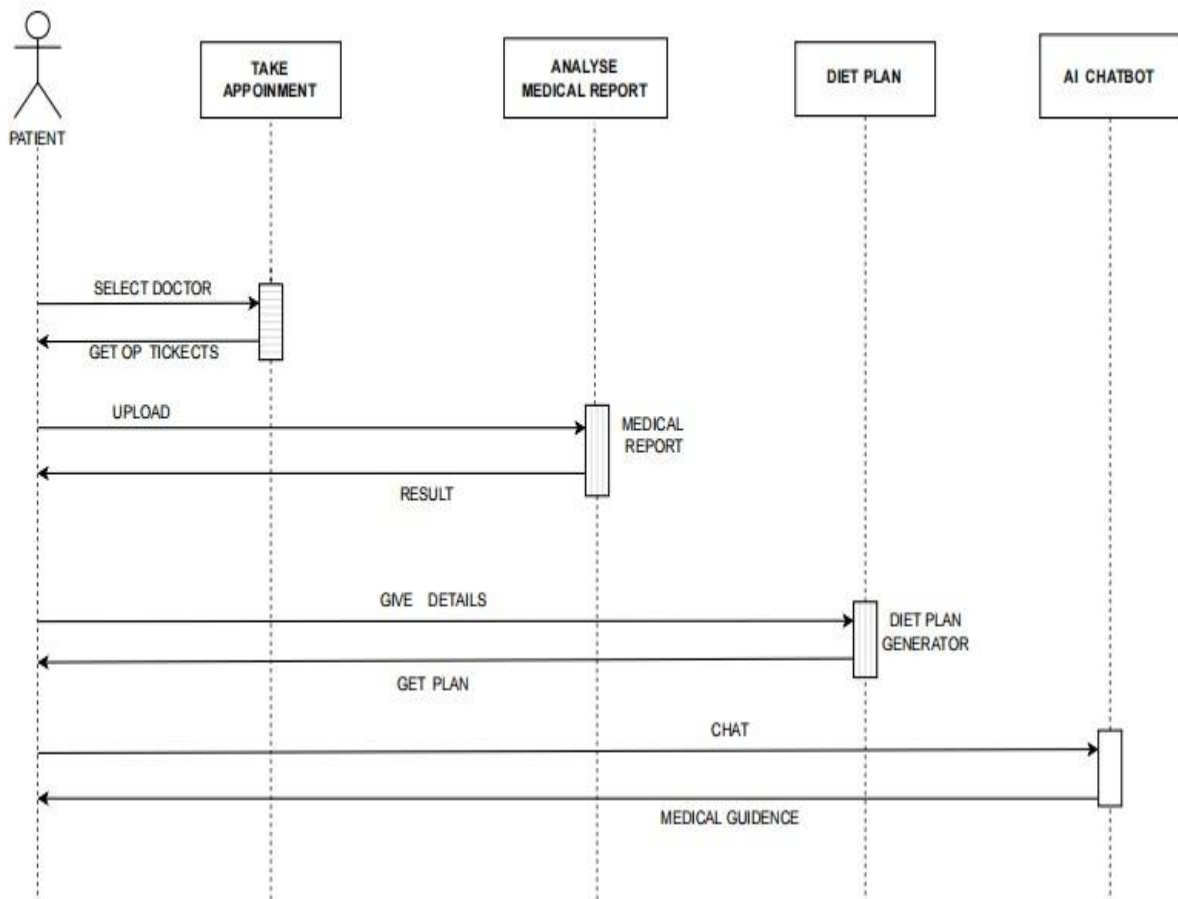


Figure 8–SEQUENCE DIAGRAM

## 4.7 PROGRAM LIST

SL. NO.	FIELD	DESCRIPTION
1	<i>Admin/views.py</i>	<i>Admin backend code</i>
2	<i>Admin/models.py</i>	<i>Database details</i>
2.1	<i>Doctor/views.py</i>	<i>Doctor backend code</i>
2.2	<i>User/views.py</i>	<i>User backend code</i>
2.3	<i>Base.html</i>	<i>Base page of home</i>
2.4	<i>Home.html</i>	<i>Home page of guest</i>
2.5	<i>Login.html</i>	<i>Login page</i>
3	<i>Register.html</i>	<i>Register page</i>
3.1	<i>Admin/base.html</i>	<i>Admin base page</i>
3.2	<i>Admin/AddDoctor.html</i>	<i>Main navigation with user dropdown</i>
3.3	<i>Admin/DoctorBooking.html</i>	<i>Auth check for private routes</i>
3.4	<i>Admin/Feedback.html</i>	<i>Interactive maze game visualization component</i>
3.5	<i>Doctor/doc_viewmore.html</i>	<i>Level display card with status</i>
3.6	<i>Doctor/Booking.html</i>	<i>Admin CRUD table for challenges</i>
3.7	<i>Doctor/base.html</i>	<i>Admin navigation bar with logout</i>
3.8	<i>Patient/base.html</i>	<i>Main navigation with authentication state</i>
3.9	<i>Patient/viewmore.html</i>	<i>Auth check for private routes</i>
4	<i>Patient/TakeAppointment.html</i>	<i>Interactive maze game visualization component</i>
4.1	<i>Patient/MyAppointment.html</i>	<i>Challenge level display card UI</i>
4.2	<i>Patient/feedback.html</i>	<i>CRUD interface for managing challenges</i>

## 4.8 DATABASE DESIGN

Database design involves creating a comprehensive data model that outlines the logical and physical structure of a database. This process encompasses making key design decisions and defining storage parameters, which are then used to generate a data definition language. This language is used to bring the design to life and create a functional database.

TABLE NAME: ADMIN

PRIMARY KEY: Email

NAME	TYPE	CONSTRAINTS	DESCRIPTION
EMAIL	SERIAL	PRIMARY KEY	UNIQUE IDENTIFIER FOR ADMIN
PASSWORD	SERIAL	NOT NULL	PASSWORD

*Table 1-admin*

TABLE NAME: DOCTOR  
PRIMARY KEY: Email

NAME	TYPE	CONSTRAINTS	DESCRIPTION
EMAIL	SERIAL	PRIMARY KEY	UNIQUE IDENTIFIER FOR DOCTOR
PASSWORD	SERIAL	UNIQUE, NOT NULL	PASSWORD FOR DOCTOR
NAME	TEXT	NOT NULL	NAME OF THE DOCTOR
SPECIALITY	TEXT	NOT NULL	SPECIALITY OF DOCTOR
PHONE	INTEGER	NOT NULL	PHONE NUMBER

*Table2: doctor*

TABLE NAME: PATIENT  
PRIMARY KEY: Email

NAME	TYPE	CONSTRAINTS	DESCRIPTION
EMAIL	SERIAL	PRIMARY KEY	EMAIL OF PATIENT
PASSWORD	SERIAL	NOT NULL	PASSWORD OF PATIENT
FULL NAME	TEXT	NOT NULL	FULL NAME OF PATIENT
DATE OF BIRTH	INTEGER	NOT NULL	DATE OF BIRTH OF PATIENT
ADDRESS	TEXT	NOT NULL	ADDRESS OF PATIENT
PHONE	INTEGER	NOT NULL	PHONE NUMBER OF PATIENT
MEDICAL HISTORY	TEXT	NOT NULL	MEDICAL HISTORY OF PATIENT

*Table 3-patient*

TABLE NAME: BOOKING  
PRIMARY KEY: NONE

NAME	TYPE	CONSTRAINTS	DESCRIPTION
PATIENT EMAIL	SERIAL	FOREIGN KEY	REFER PATIENT TABLE
DOCTOR EMAIL	SERIAL	FOREIGN KEY	REFER DOCTOR TABLE
BOOKING DATE	INTEGER	NOT NULL	DATE OF APPOINTMENT
BOOKING SLOT	INTEGER	NOT NULL	SLOT TIME OF APPOINTMENT
SYMPTOMS	TEXT	NOT NULL	SYMPTOMS DESCRIBED BY PATIENT
STATUS	INTEGER	STATUS OF PATIENT	STATUS OF APPOINTMENT

*Table 4-booking*

TABLE NAME: FEEDBACK

PRIMARY KEY: NONE

<i>NAME</i>	<i>TYPE</i>	<i>CONSTRAINTS</i>	<i>DESCRIPTION</i>
<i>PATIENT EMAIL</i>	<i>SERIAL</i>	<i>FOREIGN KEY</i>	<i>REFER PATIENT TABLE</i>
<i>DOCTOR EMAIL</i>	<i>SERIAL</i>	<i>FOREIGN KEY</i>	<i>REFER DOCTOR TABLE</i>
<i>DATE</i>	<i>DATE</i>	<i>NOT NULL</i>	<i>DATE OF FEEDBACK SUBMISSION</i>
<i>COMMENT</i>	<i>TEXT</i>	<i>NOT NULL</i>	<i>COMMENT LEFT BY PATIENT</i>

*Table 5- feedback*

TABLE NAME: MEDICINE

PRIMARY KEY: NONE

NAME	TYPE	CONSTRAINTS	DESCRIPTION
BOOK ID	SERIAL	FOREIGN KEY	REFER BOOKING TABLE
SYMPTOMS	TEXT	NOT NULL	SYMPTOMS NOTED BY DOCTOR
DISEASE	TEXT	NOT NULL	DISEASE DIAGNOSIS BY DOCTOR
MEDICINES	TEXT	NOT NULL	MEDICINES

*Table 6-medicines*

TABLE NAME: PRESCRIPTION

PRIMARY KEY: NONE

NAME	TYPE	CONSTRAINTS	DESCRIPTION
BOOKING ID	SERIAL	FOREIGN KEY	REFER BOOKING TABLE
PRESCRIPTION	TEXT	NOT NULL	PRESCRIPTION TO PATIENT
REMARKS	TEXT	NOT NULL	REMARKS BY DOCTOR

*Table 7-prescription*



TABLE NAME: RESULT ANALYSIS  
PRIMARY KEY: ID

NAME	TYPE	CONSTRAINTS	DESCRIPTION
PATIENT EMAIL	SERIAL	FOREIGN KEY	REFER PATIENT TABLE
RESULT ID	INTEGER	PRIMARY KEY	ID
MEDICAL REPORT	TEXT	NOT NULL	MEDICAL REPORT
ANALYSIS RESULT	SERIAL	NOT NULL	ANALYSIS RESULT
CREATED_AT	SERAIL	NOT NULL	RESULT ANALYSIS CREATED

## 4.9 INPUT DESIGN

The input system revolves around user authentication and learning progression management. Users enter the platform through either email/password credentials or Google OAuth integration, where input validation ensures secure access. The core interaction happens through an integrated code editor where users input code solutions in their chosen programming language (Python, Java, or C). Each challenge accepts user code submissions and provides immediate feedback through test case execution. For maze challenges, users input directional commands that control character movement, while multiplayer debug sessions accept both code modifications and room creation parameters (player limit, time constraints).

The real-time elements of the platform process various forms of input simultaneously. For standard challenges, this includes code compilation requests and submission evaluations. The maze system processes continuous directional inputs that affect character position, while the multiplayer component handles concurrent debugging attempts, and room management actions. All user progress, including completed challenges and debug ratings, is tracked and stored in the database

## 4.9 OUTPUT DESIGN

When users register, they gain access to a personalized learning dashboard that displays their progress metrics, achievement badges, and current ranking. Upon logging in, users see their debug rating, completed challenges, and available learning paths. The platform provides immediate feedback on code submissions through test case results and execution outputs. For maze challenges, users receive visual feedback of their code's execution through animated character movement and collision detection.

The multiplayer debugging interface shows real-time rankings, opponent progress, and time remaining. Users can view their historical performance analytics, including solve rates, and language-specific proficiency scores.

## 4.10 INTERFACE DESIGN

In this system, we prioritize user interface design to create an immersive and engaging learning experience for coding enthusiasts. Our design aims to facilitate seamless interactions, making it easy for users to navigate through coding challenges, track their progress, and participate in multiplayer debugging sessions. We strike a balance between visual elements, mental models, and technical functionality to create a usable and adaptable system.

Our interface design ensures that users can easily access their personalized dashboard, view their achievements, and engage with various learning paths without unnecessary complexity or visual clutter. By applying graphic design principles judiciously, we create a visually appealing and consistent theme throughout the platform, enhancing the overall user experience without compromising usability. The dark-themed IDE-like interface, responsive layout, and real-time feedback mechanisms contribute to a cohesive and intuitive user journey, making this system an effective and enjoyable platform for learning to code.

## 5. CODING

### LOGIN.HTML

```
{% extends "base.html" %}

{% block content %}

<head>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<style>

.container {

    max-width: 450px;

    padding: 40px;

    background-color: #fff;

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

    border-radius: 8px;

    text-align: center;

}

.logo h1 {

    font-size: 2.5rem;

    font-weight: bold;

    color: #ff6363;

    margin-bottom: 20px;

}

.login-form h2 {

    margin-bottom: 20px;

    font-weight: 500;

    color: #333;

}

.input-group {

    margin-bottom: 15px;
```

```

    text-align: left;
}

.btn-custom {
    width: 100%;
    background-color: #ff6363;
    color: #fff;
    font-weight: 600;
}

.register-link {
    margin-top: 20px;
    font-size: 0.9rem;
}

.register-link a {
    color: #ff6363;
}
</style>
</head>
<body>
<div class="container">
    <div class="logo">
        <h1>H-Care</h1>
    </div>
    <form class="login-form" id="loginForm" method="POST">
    <h2>Login</h2>
    <div class="input-group">
    <label for="type">User Type </label>
    <select id="type" name="type" class="form-control" required>
    <option value="">Select</option>
    <option value="ADMIN">Admin</option>
    <option value="USER">Patient</option>
    <option value="DOCTOR">Doctor</option>

```

```

</select>
</div>

<div class="input-group">
<label for="username">Username </label>
<input type="text" id="username" name="username" class="form-control" required>
</div>

<div class="input-group">
<label for="password">Password </label>
<input type="password" id="password" name="password" class="form-control" required>
</div>

<button type="button" id="submit" class="btn btn-custom mt-3">Login</button>

<p class="register-link">Don't have an account? <a href="{% url 'Register' %}">Register here</a></p>
</form>
</div>
<script>

$( '#submit' ).on( 'click', function () {
    var userid = $( '#username' ).val();
    var password = $( '#password' ).val();
    var userType = $( '#type' ).val();
    if (userid && password && userType) {
        $.ajax({
            method: 'POST',
            data: {
                userid: userid,
                password: password,
                type: userType,
                csrfmiddlewaretoken: '{{ csrf_token }}'
            },
            success: function (response) {
                if (response.status === 1) {
                    window.location.href = "{% url 'adminhome' %}";

```

```

    } else if (response.status === 2) {
        window.location.href = "{% url 'userhome' %}";
    } else if (response.status === 3) {
        window.location.href = "{% url 'doctorhome' %}";
    } else {
        alert("Invalid login details");
    }
},
error: function (xhr, textStatus, errorThrown) {
    console.error("AJAX request failed:", textStatus, errorThrown);
    alert("There was an error with the login request.");
}
});
} else {
    alert("Please fill in all fields.");
}
});
</script>
</body>{% endblock content %}

```

## ADMIN VIEWS.PY

```

from django.shortcuts import render

import datetime

from sqlite3 import IntegrityError

from django.db.models import Q

from django.shortcuts import get_object_or_404, render

from django.shortcuts import render, redirect

from django.http import HttpResponseRedirect, JsonResponse, response

from .models import *

from django.views.decorators.csrf import csrf_exempt

```

```

import os

from django.conf import settings

from django.http import FileResponse

from Admin.models import *

# Create your views here.

from django.contrib.auth import authenticate, login

# Create your views here.

@csrf_exempt

def adminhome(request):

    return render(request, "Admin/base.html")

@csrf_exempt

def AddDoctor(request):

    if request.method == 'POST':

        full_name = request.POST.get('full_name')

        specialty = request.POST.get('specialty')

        phone = request.POST.get('phone')

        address = request.POST.get('address')

        available_days = request.POST.get('available_days')

        email = request.POST.get('email')

        password = request.POST.get('password')

    if not all([full_name, specialty, phone, address, available_days, email, password]):

        return JsonResponse({"msg": "All fields are required."}, status=400)

    doctor = Doctor(

        full_name=full_name,

        specialty=specialty,

        phone=phone,

        address=address,

```

```

        available_days=available_days,

        email=email,

        password=password
    )

    doctor.save()

    return JsonResponse({"msg": "Doctor added successfully!", "doctor_id": doctor.id}, status=201)

    return render(request, "Admin/Adddoctor.html")

@csrf_exempt
def ViewDoctor(request):

    D=Doctor.objects.all()

    return render(request, "Admin/viewdoctors.html", {"ob":D})

@csrf_exempt
def ViewAppointments(request):

    D=DoctorBooking.objects.all()

    return render(request, "Admin/DoctorBooking.html", {"ob":D})

@csrf_exempt
def Patients(request):

    D=Patient.objects.all()

    return render(request, "Admin/viewpatients.html", {"ob":D})

@csrf_exempt
def ViewFeedback(request):

    feed=Feedback.objects.all().order_by('-id')

    return render(request, "Admin/Feedback.html", {'ob':feed})

```

## ADMIN MODELS.PY

```

from django.db import models

# Create your models here.

```



```
class Patient(models.Model):
```

```
    full_name = models.CharField(max_length=100)
```

```
    date_of_birth = models.DateField()
```

```
    address = models.TextField()
```

```
    phone = models.CharField(max_length=15)
```

```
    medical_history = models.TextField(blank=True, null=True)
```

```
    email=models.CharField(max_length=80,primary_key=True)
```

```
    password=models.CharField(max_length=18)
```

```
    def _str_(self):
```

```
        return self.full_name
```

```
class Doctor(models.Model):
```

```
    full_name = models.CharField(max_length=100)
```

```
    specialty = models.CharField(max_length=100)
```

```
    phone = models.CharField(max_length=15)
```

```
    address = models.TextField()
```

```
    available_days = models.CharField(max_length=100) # E.g., "Mon, Wed, Fri"
```

```
    email=models.CharField(max_length=80,primary_key=True)
```

```
    password=models.CharField(max_length=18)
```

```
    def _str_(self):
```

```
        return f"Dr. {self.full_name} - {self.specialty}"
```

```
class DoctorBooking(models.Model):
```

```
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE, related_name="bookings")
```

```
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE, related_name="appointments")
```

```
    booking_date = models.DateField()
```

```
    booking_sloat = models.IntegerField()
```

```
    symptoms = models.TextField()
```

```
    status = models.CharField(max_length=20, choices=[
```

```

        ('Pending', 'Pending'),

        ('Confirmed', 'Confirmed'),

        ('Completed', 'Completed'),

        ('Cancelled', 'Cancelled')

    ], default='Pending')

    def _str_(self):

        return f"Booking: {self.patient} with {self.doctor} on {self.booking_date} at {self.booking_sloat}"

class Feedback(models.Model):

    patient = models.ForeignKey(Patient, on_delete=models.CASCADE, related_name="feedbacks")

    date = models.DateTimeField(auto_now_add=True)

    comment = models.TextField(blank=True, null=True)

    def _str_(self):

        return f"Feedback from {self.patient} to {self.doctor} - Rating: {self.rating}"

class Medicines(models.Model):

    bookid=models.ForeignKey(DoctorBooking,on_delete=models.CASCADE)

    symptoms=models.TextField(blank=True, null=True)

    disease=models.TextField(blank=True, null=True)

    prescription=models.TextField(blank=True, null=True)

    remarks=models.TextField(blank=True, null=True)

```

## DOCTOR VIEWS.PY

```

from django.shortcuts import render
import datetime
from sqlite3 import IntegrityError
from django.db.models import Q
from django.shortcuts import get_object_or_404, render
from django.shortcuts import render,redirect
from django.http import HttpResponseRedirect,JsonResponse, response
from .models import *
from django.views.decorators.csrf import csrf_exempt

```

```

import os
from django.conf import settings
from django.http import FileResponse
from Admin.models import *
# Create your views here.
from django.contrib.auth import authenticate, login
# Create your views here.
@csrf_exempt
def doctorhome(request):
    return render(request, "Doctor/base.html")
@csrf_exempt
def Bookings(request):
    if request.method == 'POST':
        user=request.session['gmail']
        user=Doctor.objects.get(email=user)
        selected_date = request.POST.get('date')
        bookings = DoctorBooking.objects.filter(booking_date=selected_date,doctor=user)
        booking_list = []
        for booking in bookings:
            booking_list.append({
                'id': booking.id,
                'patient_name': booking.patient.full_name, # Assuming the Patient model has a full_name field
                'symptoms': booking.symptoms,
                'Sloat': booking.booking_sloat,
                'status':booking.status
            })
        print(booking_list)
        return JsonResponse(booking_list, safe=False)
    return render(request, "Doctor/Bookings.html")
@csrf_exempt
def update_booking_status(request):
    if request.method == 'POST':
        booking_id = request.POST.get('booking_id')
        new_status = request.POST.get('status')
        try:
            booking = DoctorBooking.objects.get(id=booking_id)
            booking.status = new_status
            booking.save()
            return JsonResponse({"status": "success"}, status=200)

```

```

except DoctorBooking.DoesNotExist:
    return JsonResponse({"status": "error", "message": "Booking not found"}, status=404)
return JsonResponse({"status": "error", "message": "Invalid request"}, status=400)
def doc_viewmore(request,id):
    booking = DoctorBooking.objects.get(id=id)
    if request.method == 'POST':
        symptoms = request.POST.get('symptoms')
        disease = request.POST.get('disease')
        prescription = request.POST.get('prescription')
        remarks = request.POST.get('remarks')
        # Save prescription details
        Medicines.objects.create(
            bookid=booking,
            symptoms=symptoms,
            disease=disease,
            prescription=prescription,
            remarks=remarks
        )
        booking.status = "Completed"
        booking.save()
        return redirect('Bookings') # Redirect to the same page after saving
    return render(request,"Doctor/doc_viewmore.html",{ "booking":booking})

```

## PATIENT VIEWS.PY

```

from django.shortcuts import render
import datetime
from sqlite3 import IntegrityError
from django.db.models import Q
from django.shortcuts import get_object_or_404, render
from django.shortcuts import render,redirect
from django.http import HttpResponse,JsonResponse, response
from .models import *
from django.views.decorators.csrf import csrf_exempt
import os
from django.conf import settings
from django.http import FileResponse
from Admin.models import *
# Create your views here.

```

```

from django.contrib.auth import authenticate, login
# Create your views here.
@csrf_exempt
def home(request):
    return render(request, "base.html")
@csrf_exempt
def login(request):
    if request.method == 'POST':
        print("----- Login Request Processing -----")
        userid=request.POST.get('userid')
        password=request.POST.get('password')
        type=request.POST.get('type')
        print(request.POST)
        try:
            if userid=="admin" and password=="admin" and type=="ADMIN":
                return JsonResponse({'status': 1})
            elif type=="USER":
                ob=Patient.objects.get(email=userid,password=password)
                request.session['gmail']=ob.email
                return JsonResponse({'status': 2})
            elif type=="DOCTOR":
                ob=Doctor.objects.get(email=userid,password=password)

request.session['gmail']=ob.email
                return JsonResponse({'status': 3})
            else:
                ob=Users.objects.get(email=userid,password=password,status=1)
                request.session['gmail']=ob.email
                return JsonResponse({'status': 4})
        except Exception as e:
            print(e)
            return JsonResponse({'status': "failed"})
    return render(request, "LOGIN.html")
@csrf_exempt
def Register(request):
    if request.method == 'POST':
        try:
            print("..... USER REGISTRATION .....")
            name = request.POST.get('name')

```

```

    email = request.POST.get('email')
    password = request.POST.get('password')
    phone = request.POST.get('phone')
    address = request.POST.get('address')
    date_of_birth = request.POST.get('date_of_birth')
    medical_history = request.POST.get('medical_history', '') # Optional field

    print(password)

    patient = Patient.objects.create(
        full_name=name,
        email=email,
        password=password,
        phone=phone,
        address=address,
        date_of_birth=date_of_birth,
        medical_history=medical_history,
    )

    print("success")
    data={"msg": "Successfully Registered"}
    return JsonResponse(data,safe=False)
except IntegrityError:
    data={"msg": "Already exists"}
    return JsonResponse(data,safe=False)

except Exception as e:
    print(e)
    data={"msg": "Failed"}
    return JsonResponse(data,safe=False)

return render(request, "Register.html")

# user home
@csrf_exempt
def userhome(request):
    return render(request, "Patient/base.html")

@csrf_exempt
def viewmore(request,id):
    id=DoctorBooking.objects.get(id=id)
    med=Medicines.objects.filter(bookid=id).first()
    return render(request, "Patient/viewmore.html", {"ob":med})

@csrf_exempt
def GiveFeedback(request):

```

```

if request.method == 'POST':
    user=request.session['gmail']
    user=Patient.objects.get(email=user)
    feedback_text = request.POST.get('feedback_text')
    Feedback.objects.create(
        patient=user,
        comment=feedback_text
    )
    return redirect('userhome')
return render(request,"Patient/feedback.html")
@csrf_exempt
def MyAppointment(request):
    user=request.session['gmail']
    user=Patient.objects.get(email=user)
    doc=DoctorBooking.objects.filter(patient=user).order_by('-id')
    return render(request,"Patient/MyAppointment.html",{ "ob":doc})
@csrf_exempt
def TakeAppointment(request):
    doc=Doctor.objects.all()
    if request.method == 'POST':
        user=request.session['gmail']
        user=Patient.objects.get(email=user)

        doctor_id = request.POST.get('doctor_id')
        booking_date = request.POST.get('booking_date')
        booking_sloat = request.POST.get('booking_sloat')
        symptoms = request.POST.get('symptoms')
        doctor_id=Doctor.objects.get(email=doctor_id)
        existing_booking = DoctorBooking.objects.filter(
            doctor_id=doctor_id,
            booking_date=booking_date,
            booking_sloat=booking_sloat
        ).exists()
        if existing_booking:
            return render(request, "Patient/TakeAppointment.html",{ "message": "This time slot is already
booked.", "ob":doc})
# Create a new booking
        booking = DoctorBooking.objects.create(
            patient=user,

```

```
        doctor=doctor_id,  
        booking_date=booking_date,  
        booking_sloat=booking_sloat,  
        symptoms=symptoms  
    )  
    return render(request, "Patient/TakeAppointment.html", {"message": "Appointment booked  
successfully!", "ob": doc})  
    doc=Doctor.objects.all()  
    return render(request, "Patient/TakeAppointment.html", {"ob": doc})
```



## 6.SOFTWARE TESTING

Testing is the process of evaluating a software application or system to determine whether it meets the required specifications, works as expected, and is free from defects. It involves executing a series of test cases and scenarios to validate the software's functionality, performance, security, and usability.

The primary purpose of testing is to:

1. Ensure the software meets the required specifications and user expectations.
2. Identify and report defects, errors, or inconsistencies.
3. Validate the software's functionality, performance, security, and usability.
4. Provide feedback to the development team to improve the software quality.
5. Reduce the risk of software failures or errors in production.

### 6.1 TESTING STRATEGY

A test strategy is a comprehensive, high-level document that defines the overall testing approach and outlines the specific types and levels of testing to be conducted for a product. It establishes the testing framework for the software development life cycle, ensuring that the appropriate testing types and levels are executed to validate the product's quality. The test strategy document is a crucial component of testing documentation, encompassing essential elements such as test strategy components, types of test strategies, and various testing activities. In software engineering, common testing strategies include black box testing, white box testing, and unit testing, among others. This document serves as a foundational guide for testing, ensuring that all stakeholders are aligned on the testing approach and objectives.

#### 6.1.1 Unit Testing

Each module is subjected to individual testing, which takes place during the programming stage. This iterative process involves identifying and correcting any logical errors that arise. The ultimate goal is to verify that every module functions as intended. In our system, we have successfully conducted separate testing on all modules, ensuring their independent operation meets expectations.

## 6.1.2 Integration Testing

In our project, we successfully conducted integration testing to ensure that individual modules worked together seamlessly. We employed an incremental approach, integrating and testing each module separately to identify and debug defects efficiently. Through this process, we validated data integrity and module interactions, guaranteeing cohesive and functional systems. By doing so, we ensured that our project's components interacted correctly, exchanging data accurately and consistently.

## 6.1.3 System Testing

We successfully completed system testing, thoroughly evaluating our software's functionality, performance, security, usability, and compatibility. Through rigorous end-to-end testing, we ensured our system works seamlessly, identifying and fixing critical defects. Our team's dedication and meticulous testing approach paid off, resulting in a high-quality software system that meets the required standards and exceeds user experience.

## 6.2 TEST CASES

### Registration

Test Case 1:

Title: Successful Registration

Input:

- Email: "newuser@example.com"

Password: "password123" Action:

- Enter registration details

Click register button Expected Response:

- Registration successful message

- User redirected to login page

Test Case 2:

Title: Duplicate Email Input:

- Email: "existinguser@example.com"

- Password: "password123" Action:
- Enter registration details
- Click register button Expected Response:
- Error message "Email already exists"
- Registration page remains displayed

## Login

Test Case 3:

Title: Successful Login Input:

- Email: "testuser@example.com"
- Password: "password123" Action:
- Enter login credentials
- Click login button Expected Response:
- Login successful message
- User redirected to dashboard

Test Case 4:

Title: Invalid

Password Input:

- Email: "testuser@example.com"
- Password: "wrongpassword" Action:
- Enter login credentials
- Click  
login  
button  
Expected  
Response:
- Error message "Invalid password"
- Login page remains displayed

## Booking

Test Case 5:

Title: Book Appointment

Input:

- Email: "testuser@example.com"

Password:

"password123" Action:

- Book an appointment

- Click

submit

button

Expected

Response:

- Booking successful message

Test Case 6:

Title: View Booking Input:

Input:

- Email: "testuser@example.com"

- Password:

"password123"

Action:

- View booking details

- Expected

Response:

- Error message "Sucessfully completed booking"

- Code editor remains displayed

## Patient Details

Test Case 7:

Title: Patient Details

Input:

- Login

Action:

- Show patient details

- Expected Response:

## Feedback

Test Case 8:

Title: Give feedback

Input:

- Login to give feedback

Action:

- Enter Email

## 7. IMPLEMENTATION

Implementation is the process of converting the designed system into a fully functional and efficient product that meets both technical specifications and user needs. It involves transforming theoretical designs into practical solutions by developing robust software components, establishing secure databases, and integrating every aspect of the system seamlessly. This process ultimately delivers a tool that empowers users through interactive and dynamic experiences, enhancing learning and productivity.

### Implementation Details

- Python Django, and SQLite3, following coding best practices to ensure a clean, modular, and maintainable codebase.
- Database Creation: We established the SQLite3 database using the defined schema, ensuring robust data integrity, efficient query performance, and scalability Programming: We can developed the core application using other programming languages for future growth.
- Testing: We thoroughly tested the system with operational data to ensure its success.

### Implementation Plan

The implementation plan for the platform involves a series of well-structured steps that began with developing the core functionalities and proceeded through extensive testing, deployment, and user training. Initially, we focused on building the system's fundamental features and setting up the database, followed by rigorous unit and integration testing to ensure reliability and performance. Once the system was validated, we deployed it to a production environment while continuously monitoring its performance and addressing any emerging issues. Finally, we rolled out detailed user documentation and conducted training sessions to ensure a smooth transition and successful adoption, resulting in a product that meets user expectations and stands as a robust and scalable web application.

## 8.SCREENSHOTS

### LOGIN PAGE

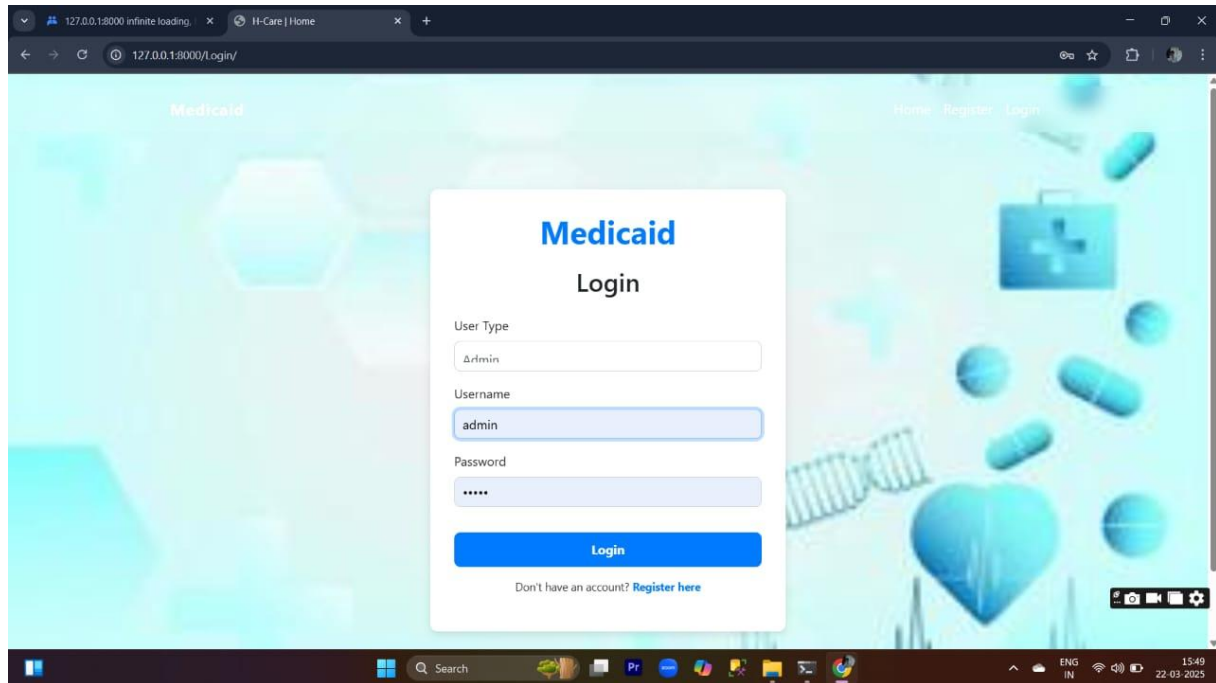


Figure 9–LOGIN PAGE

### HOME PAGE

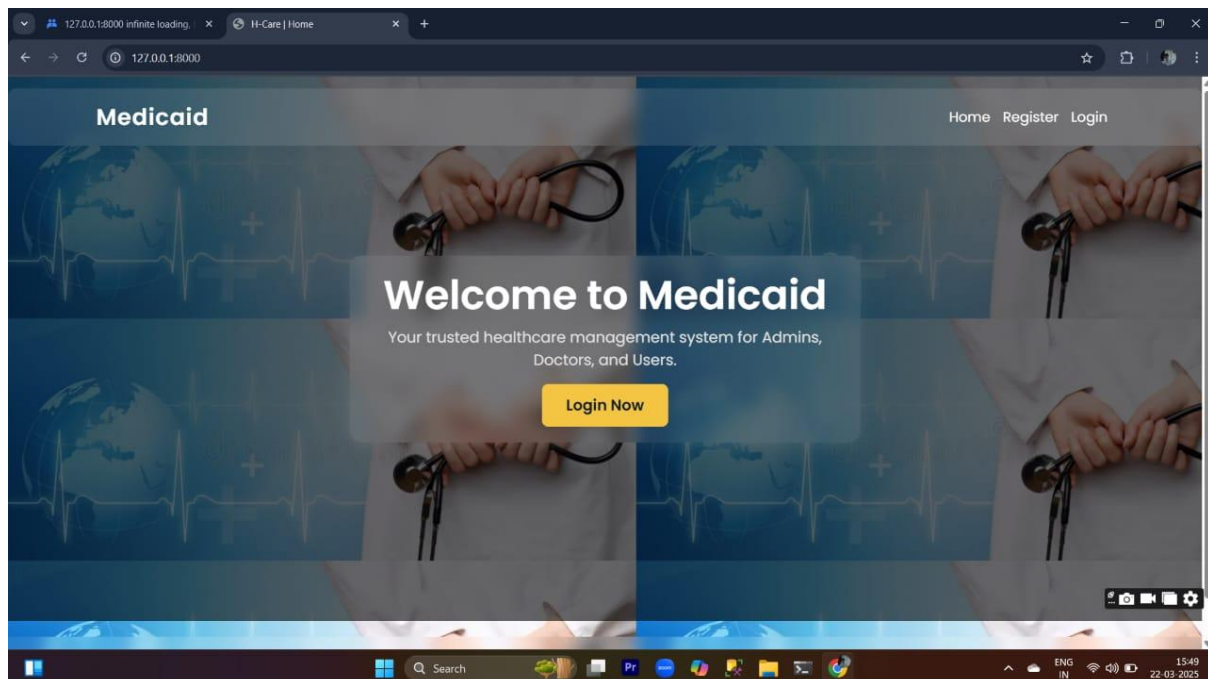
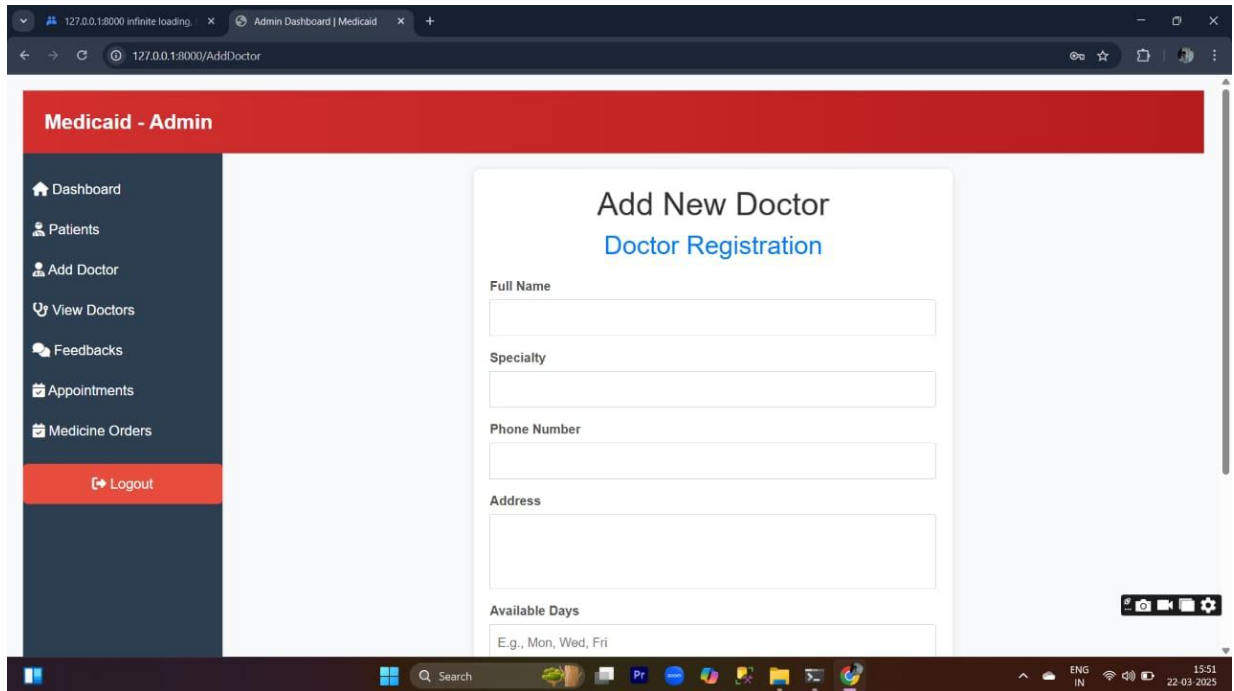


Figure 10–HOME PAGE

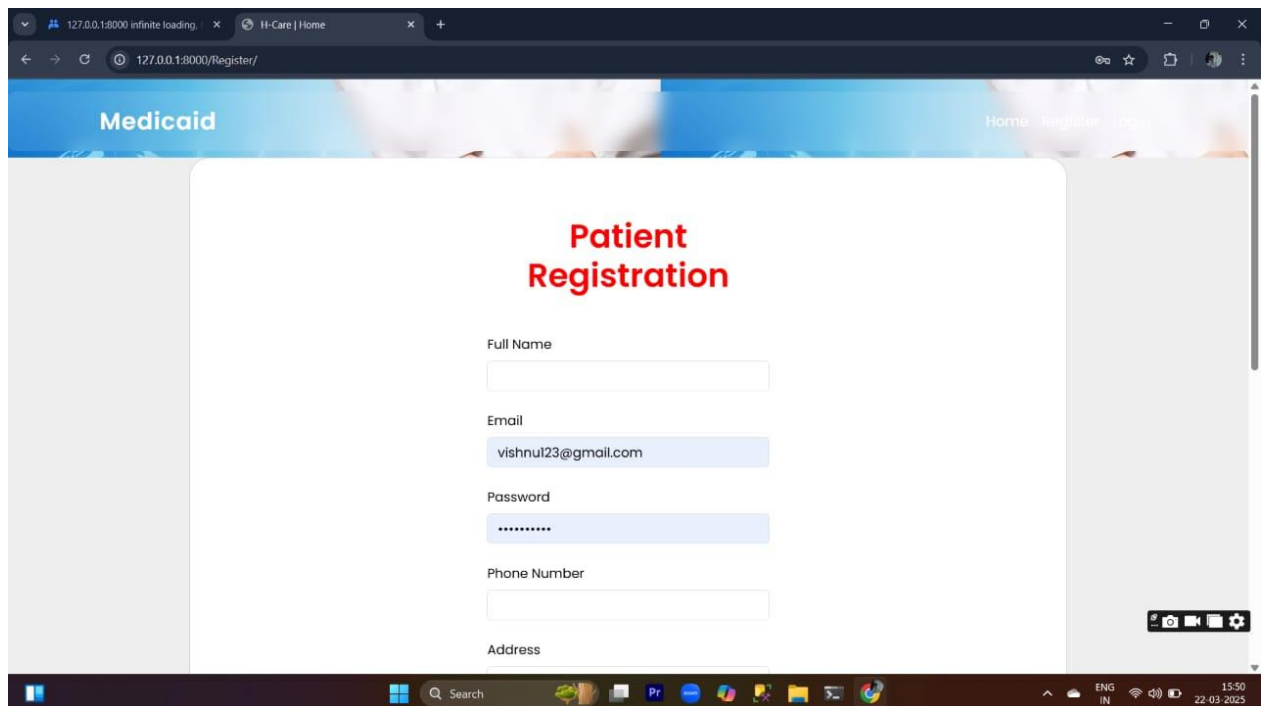
## DOCTOR REGISTRATION PAGE



The screenshot shows a web browser window with the URL `127.0.0.1:8000/AddDoctor`. The page has a red header bar labeled "Medicaid - Admin". On the left is a dark sidebar with navigation links: Dashboard, Patients, Add Doctor, View Doctors, Feedbacks, Appointments, Medicine Orders, and a red "Logout" button. The main content area is titled "Add New Doctor" with a subtitle "Doctor Registration". It contains a form with the following fields: Full Name, Specialty, Phone Number, Address, and Available Days (with a hint "E.g., Mon, Wed, Fri"). A Windows taskbar is visible at the bottom.

Figure 11- DOCTOR REGISTRATION PAGE

## PATIENT REGISTRATION PAGE



The screenshot shows a web browser window with the URL `127.0.0.1:8000/Register/`. The page has a blue header bar labeled "Medicaid" with navigation links: Home, Register, and Login. The main content area is titled "Patient Registration" in red. It contains a form with the following fields: Full Name, Email (pre-filled with "vishnul23@gmail.com"), Password (masked with "\*\*\*\*\*"), Phone Number, and Address. A Windows taskbar is visible at the bottom.

Figure 12-PATIENT REGISTRATION



## BOOKING PAGE

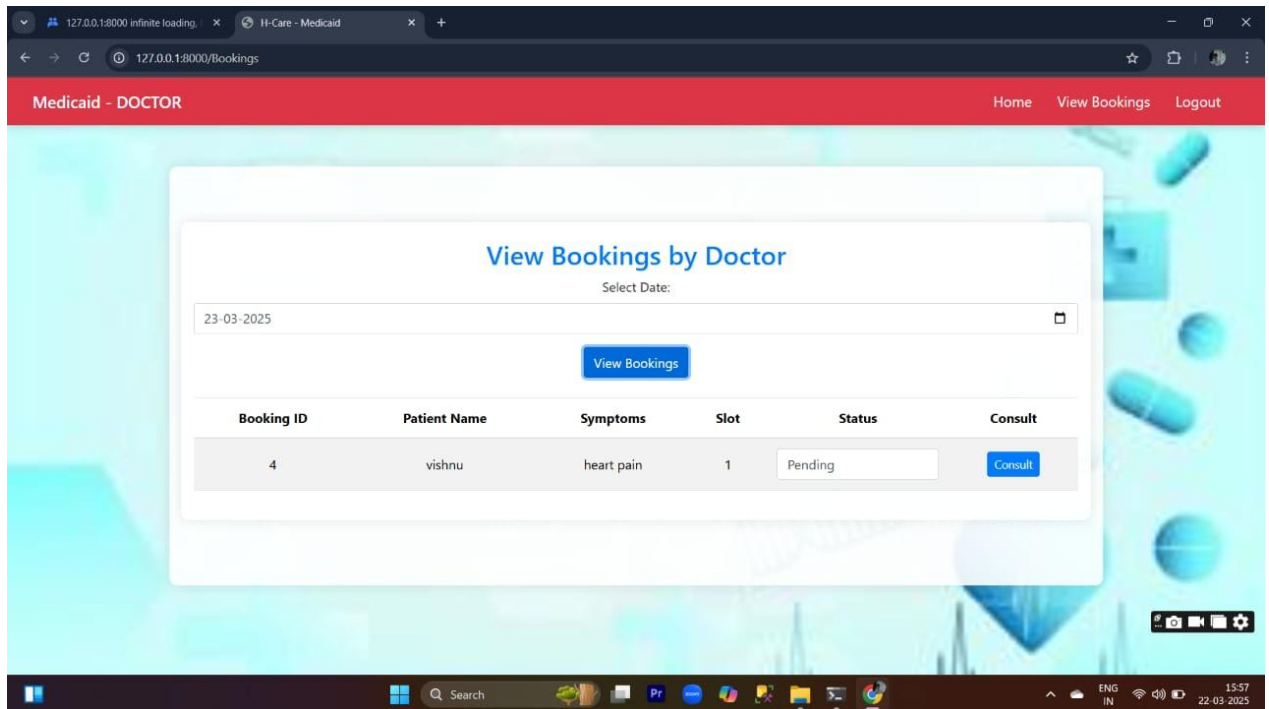


Figure 13-BOOKING

## VIEW DOCTORPAGE

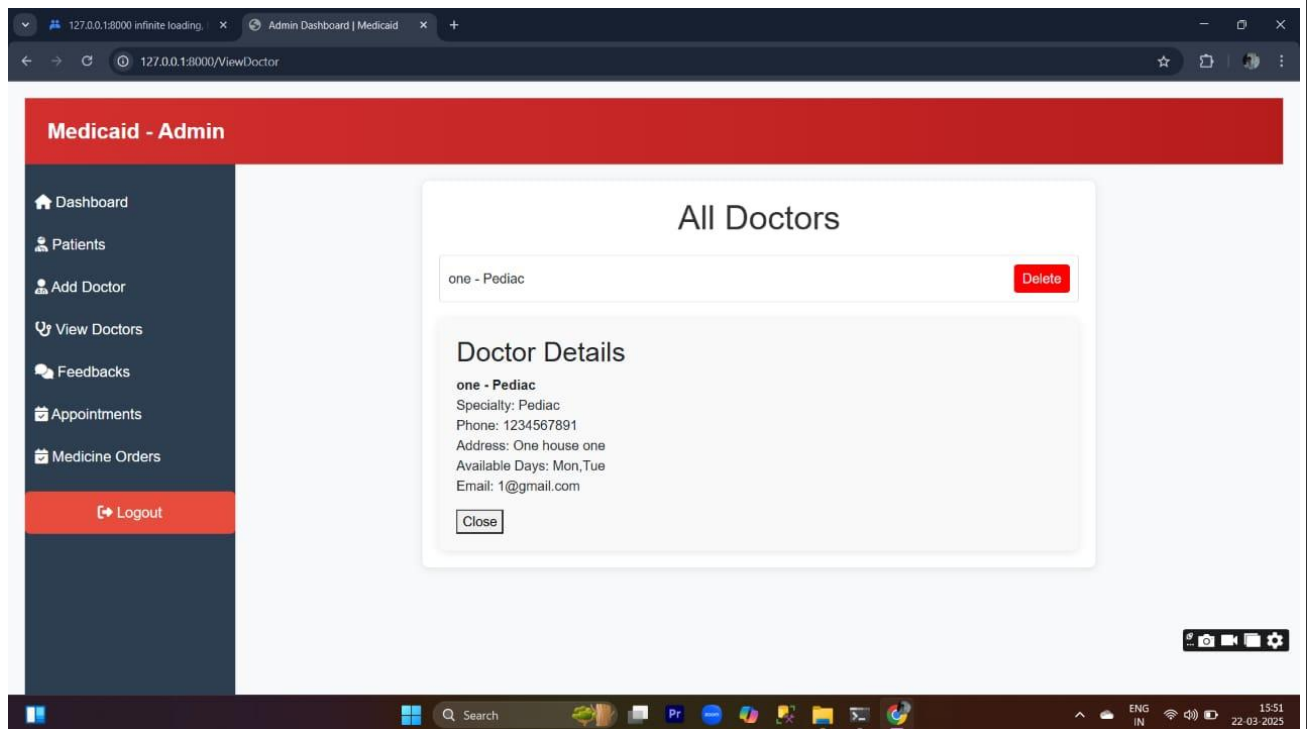
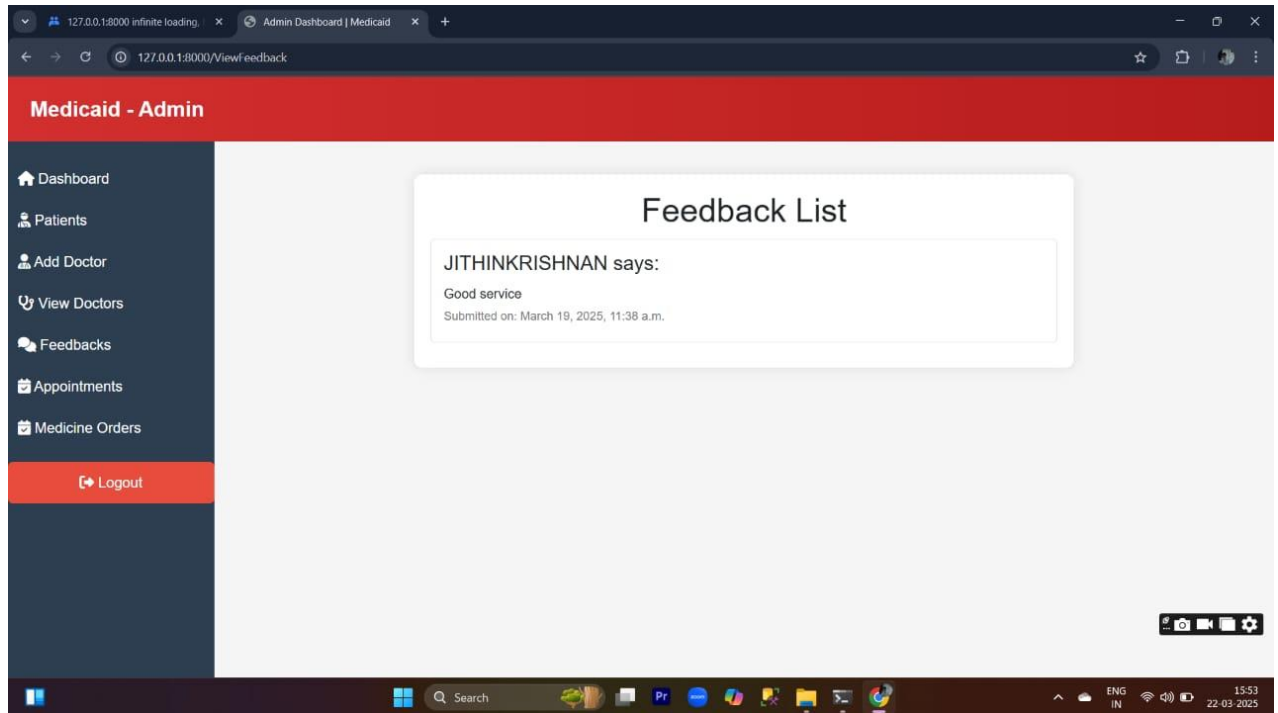


Figure 14-VIEW DOCTORPAGE

## FEEDBACK LIST PAGE

*Figure 15–FEEDBACK LIST PAGE*

## 9.CONCLUSION

The project titled "**MEDICAID**" is an AI-powered healthcare management system designed to enhance efficiency, accessibility, and patient care by integrating patients, doctors, and administrators into a unified digital platform. By addressing the limitations of traditional healthcare systems, this system gives streamlines appointment scheduling, medical report management, and personalized health recommendations, ensuring a more seamless and patient-centric healthcare experience. The system's AI-powered voice chatbot provides real-time medical guidance, improving accessibility and engagement for users.

With its automation-driven approach, secure data management, and AI-assisted decision-making, this system significantly reduces administrative workload, enhances healthcare delivery, and accelerates medical diagnoses. The integration of machine learning and Natural Language Processing (NLP) enables intelligent patient support, making healthcare services more efficient and accessible. Additionally, real-time medical data analysis and personalized diet planning contribute to better health outcomes for patients.

In conclusion, this system revolutionizes modern healthcare management by offering an intelligent, scalable, and user-friendly solution that bridges the gap between patients and healthcare providers. With its innovative AI-powered features, the system ensures faster medical assistance, improved decision-making, and a more efficient healthcare ecosystem. By leveraging technology, this system sets a new standard for digital healthcare solutions, making medical services smarter, safer, and more accessible for everyone.

## 10.FUTURE ENHANCEMENTS

- **AI-Driven Disease Prediction** – Integrate machine learning models to predict potential health risks based on patient history and symptoms.
- **Telemedicine Integration** – Enable video consultations between doctors and patients for remote medical assistance.
- **Wearable Device Connectivity** – Connect with smartwatches and fitness trackers to monitor real-time health metrics like heart rate and oxygen levels.
- **Blockchain for Medical Records** – Implement blockchain technology to enhance data security, transparency, and interoperability of medical records.
- **Multi-Language Support** – Expand the AI chatbot to support multiple languages for better accessibility across diverse user groups.
- **Automated Medicine Reminders** – Develop a feature that reminds patients to take prescribed medications on time.
- **Health Insurance Integration** – Incorporate an insurance management module for seamless claims processing and policy tracking.
- **Advanced AI Chatbot Features** – Improve the chatbot with **emotion recognition and personalized health advice** based on patient interactions.
- **Mental Health Support Module** – Add features for mental health tracking, stress management tips, and virtual therapy sessions.
- **Emergency Alert System** – Implement an emergency feature that allows patients to contact nearby hospitals or emergency services instantly.

These enhancements will significantly enhance the user experience, making it even easier for users to learn and improve their coding skills. We are committed to continuously improving and expanding our platform to meet the evolving needs of our users.

# 11.BIBLIOGRAPHY

## 11.1 TEXT REFERENCES

1. Karan A Wager, Frances W Lee, John P Glaser, Healthcare Information System: A Practical Approach for Health Care Management
2. Parag Mahajan, Artificial Intelligence in Healthcare
3. B. Cherny, Programming JavaScript

## 11.2 ONLINE REFERENCE

1. Python Django, Official Documentation. <https://nodejs.org/en/docs/>
2. SQLite, Official Documentation. [https://www. Sqlite3.org/docs/](https://www.Sqlite3.org/docs/)
3. HTML, Official Documentation. <https://html.org/docs/>
4. JavaScript, Official Documentation. <https://www.javascript.org/docs/>

## 12.APPENDIX

### 12.1 GANTT CHART

A Gantt chart, invented by Henry Gantt, is a specialized bar chart that visually represents a project's schedule. It showcases the commencement and completion dates of both terminal and summary elements, which collectively form the project's work breakdown structure. This chart provides a clear and concise overview of the project's timeline, enabling efficient tracking and management of tasks.

#### Meeting Minute

Date : Jan 03,2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

#### Participants

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

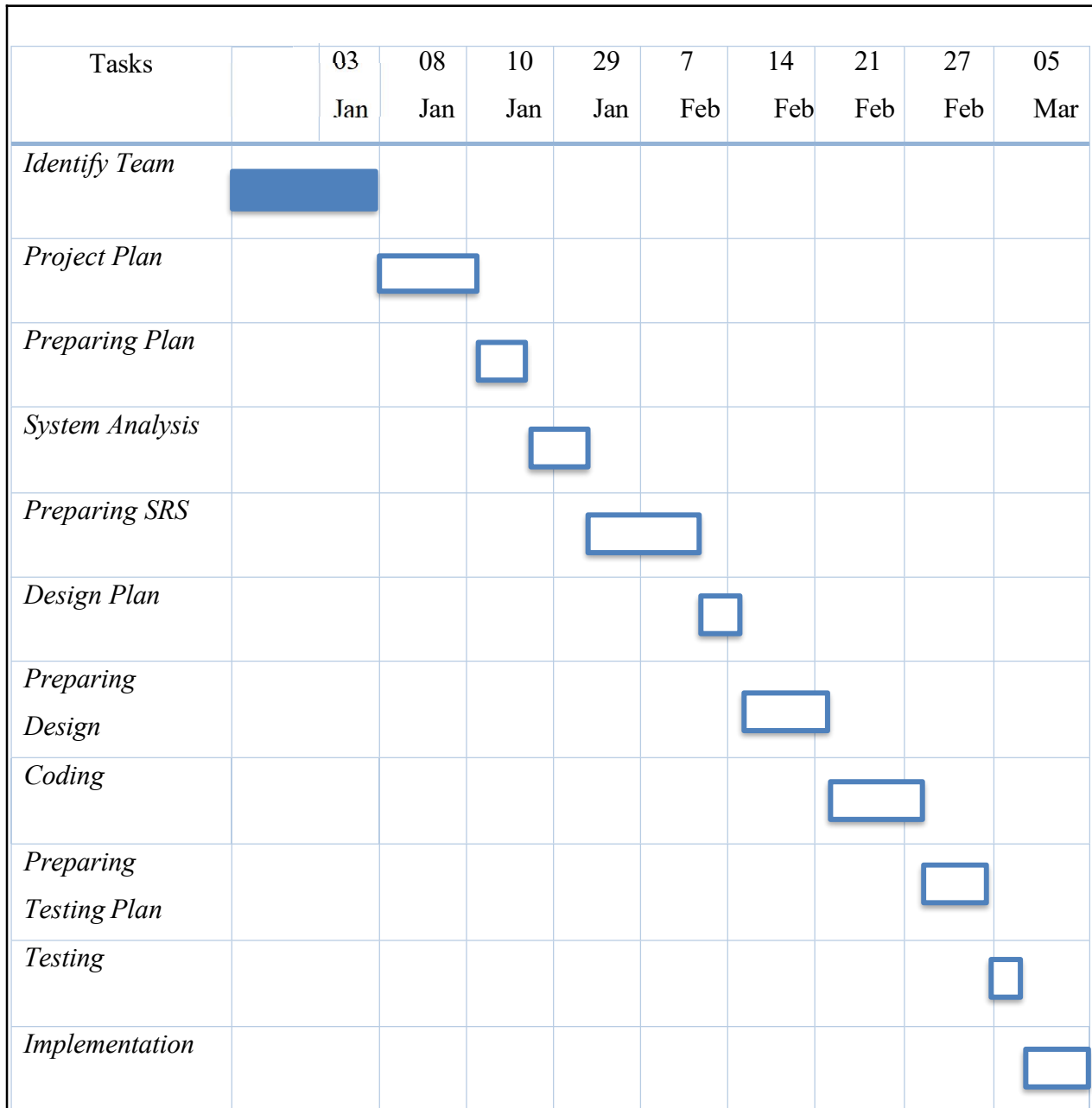


Figure 30–Jan 03,2025

**Meeting Minute**

Date : Jan 08,2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science



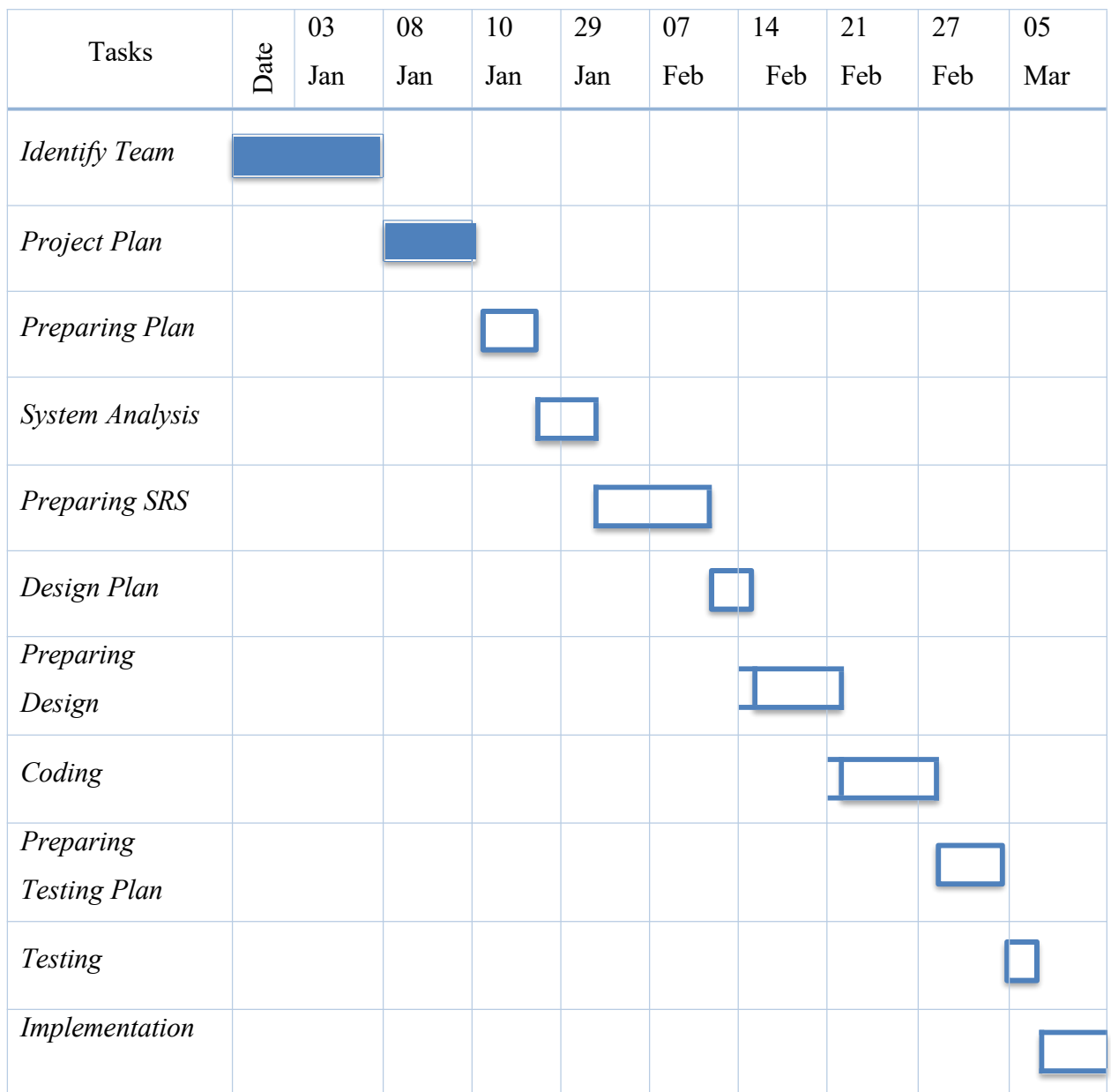


Figure 31-Jan 08,2025

**Meeting Minute**

Date : Jan 10, 2025  
Time : 10:00 AM – 11.00 AM  
Location : Sree Narayana College, Cherthala  
Topic : Medicaid

**Participants**

Guide Name : Aswathy K  
: Chinnu K Deepu  
  
Students Name : Nithin M N  
: Sreeraj S  
: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

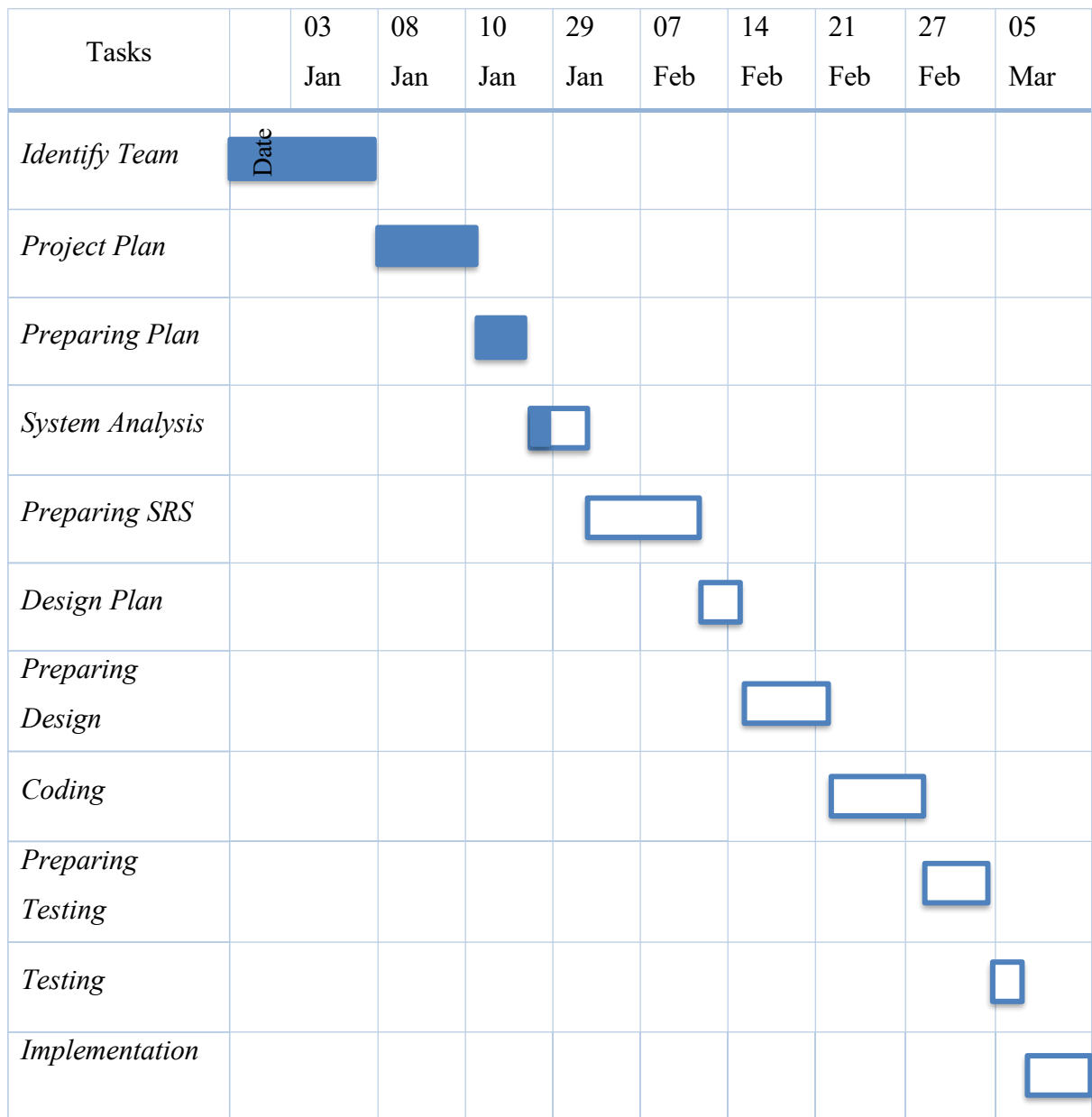


Figure 32-Jan 10, 2025

**Meeting Minute**

Date : Jan 29, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

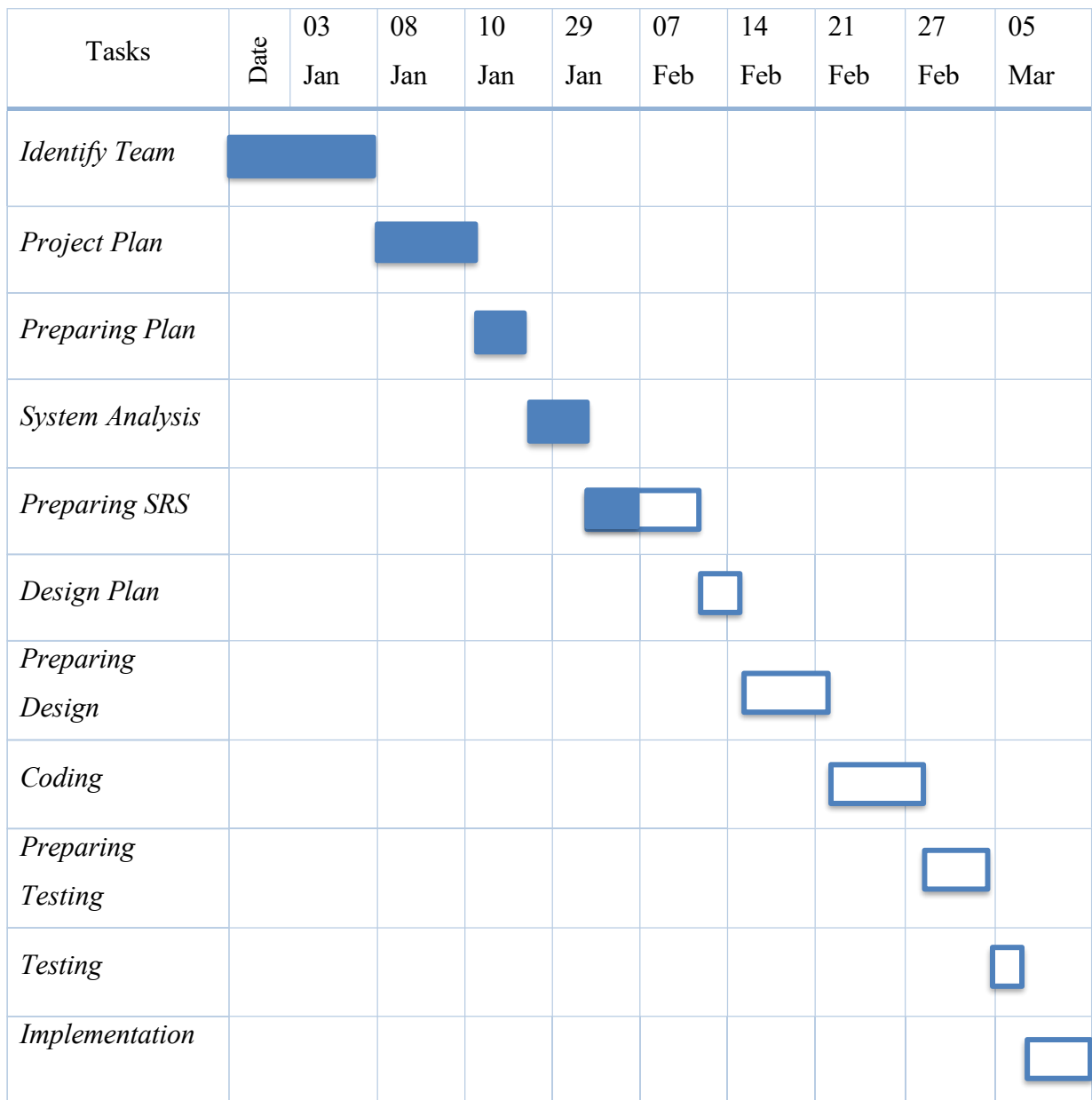


Figure 33-Jan 29, 2025

**Meeting Minute**

Date : Feb 07, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

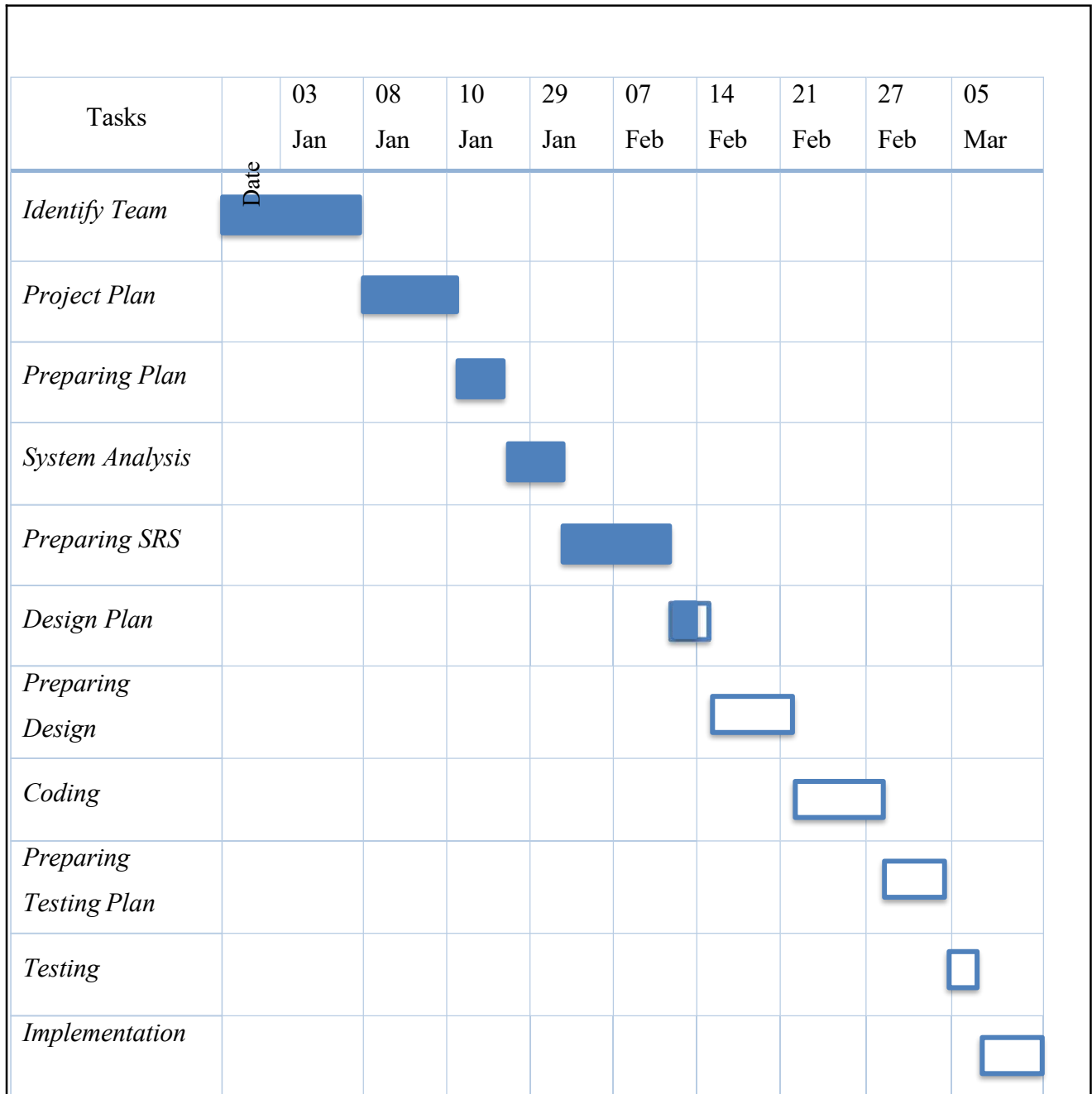


Figure 34-Feb 07, 2025

**Meeting Minute**

Date : Feb 14, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science



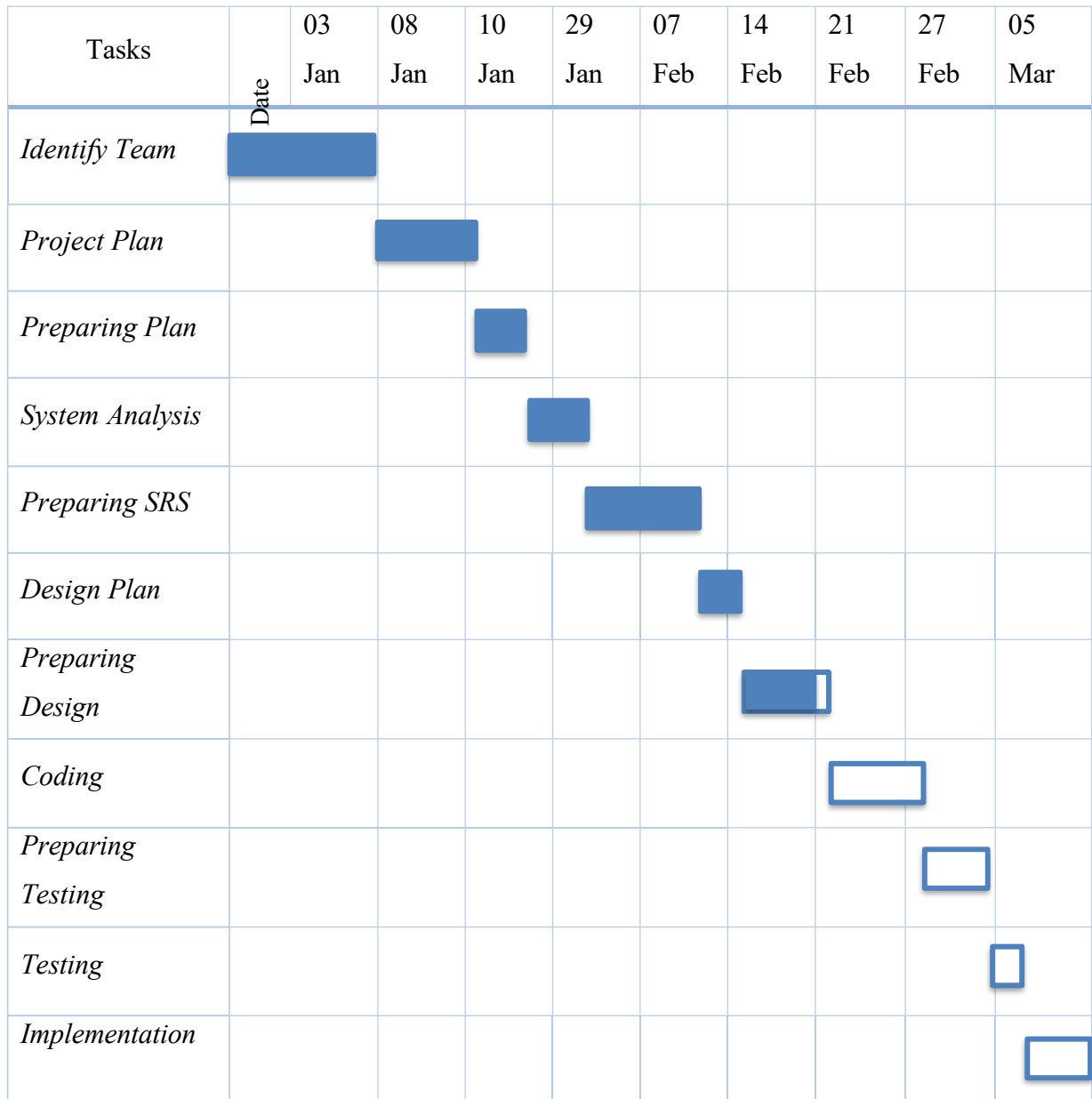


Figure 35-Feb 14, 2025

**Meeting Minute**

Date : Feb 21, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

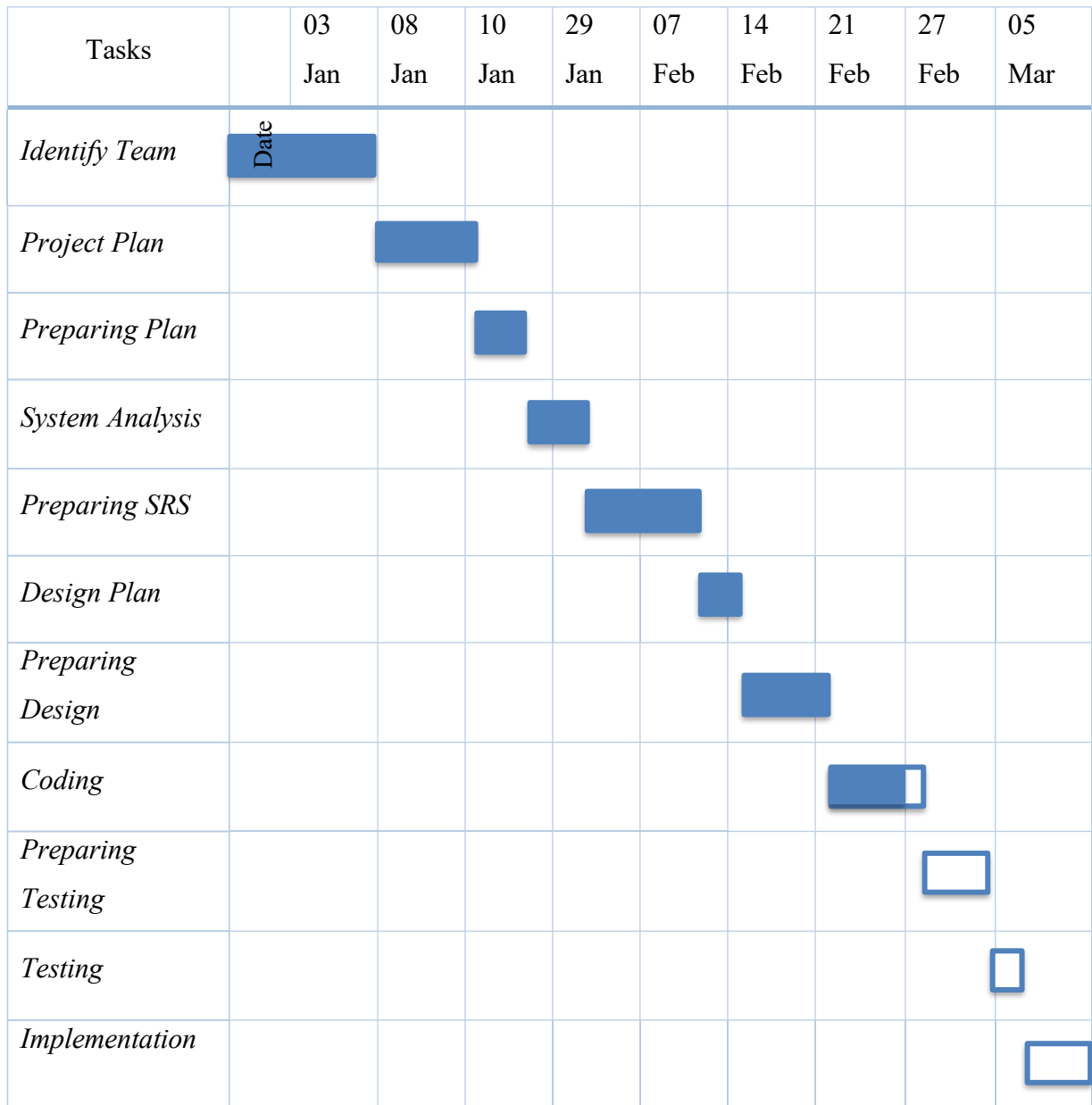


Figure 36-Feb 21, 2025

**Meeting Minute**

Date : Feb 27, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

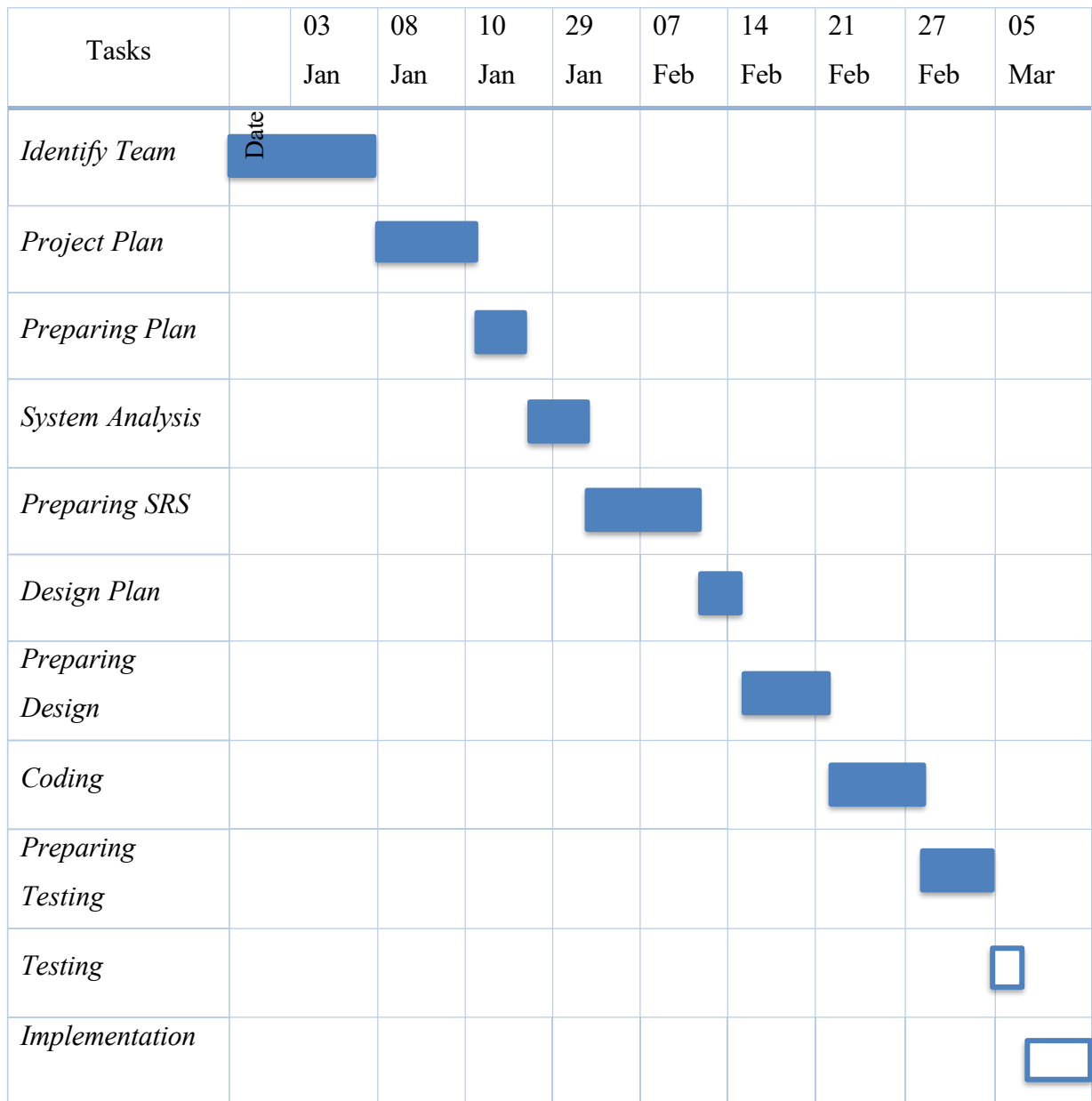


Figure 37-Feb 27, 2025

**Meeting Minute**

Date : Mar 05, 2025

Time : 10:00 AM – 11.00 AM

Location : Sree Narayana College, Cherthala

Topic : Medicaid

**Participants**

Guide Name : Aswathy K

: Chinnu K Deepu

Students Name : Nithin M N

: Sreeraj S

: Vishnu Raj K R

Completion of Assigned Task

Submitted to Department of Computer Science

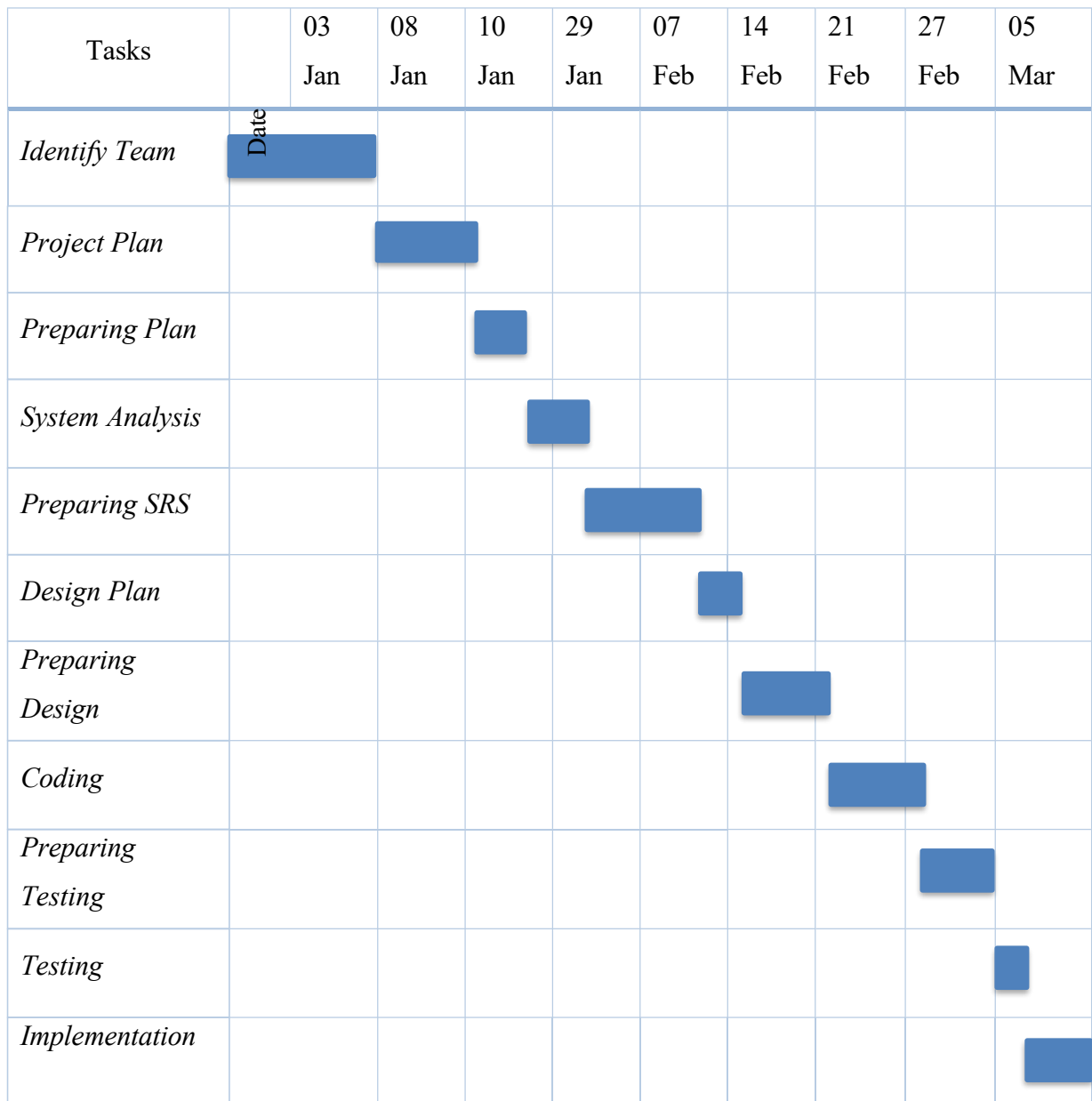


Figure 38-Mar 05, 2025