

SWASTHYA SETU

BITE304L – WEB TECHNOLOGIES PROJECT

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Information Technology

In

Department of Information Technology

by

Arya Sachin Mulay (23BIT0131)

Yash Kumar Singh (23BIT0288)



School of Computer Science Engineering and Information Systems [SCORE]

VIT, Vellore

April, 2025

DECLARATION

I hereby declare that the **BITE304L - Web Technologies** entitled “**Swasthya Setu**” submitted by me, for the award of the degree of *Bachelor of Information Technology in Department of Information Technology, School of Computer Science Engineering and Information Systems* to VIT is a record of bonafide work carried out by me under the supervision of **Dr. R. VIJAYAN, PROFESSOR GRADE 1**, SCORE, VIT, Vellore.

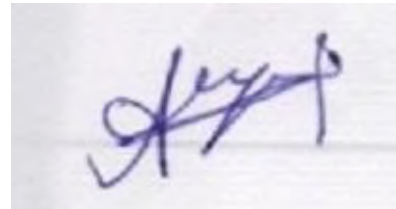
I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 15-04-2025

Signature of the Candidate

Arya Mulay:



CERTIFICATE

This is to certify that the **BITE304L – Web Technologies Project** entitled “**SWASTYA SETU**” submitted by **ARYA SACHIN MULAY (23BIT0131)** and **YASHKUMAR SINGH (23BIT0288)**, SCORE, VIT, for the award of the degree of *Bachelor of Information Technology in the Department of Information Technology* *provision* during the period, 13. 12. 2024 to 17.04.2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Capstone project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 15-04-2025

**Signature of the VIT-SCORE -
Guide**

Acknowledgement

I would like to express my heartfelt gratitude to all those who have supported and guided me throughout the development of this project titled **“Patient Record Management System”**.

First and foremost, I extend my sincere thanks to my project guide, VIJAYAN R, for their invaluable support, constant encouragement, and expert guidance. Their insights and suggestions have played a crucial role in the successful completion of this work.

I would also like to thank the faculty members of the SCORE DEPARTMENT, VIT VELLORE, for providing me with the technical knowledge and practical exposure that laid the foundation for this project. A special thanks to my peers and friends for their moral support, timely feedback, and assistance during the various stages of development.

Lastly, I am thankful to my family for their unwavering support and motivation throughout the duration of this project. This project would not have been possible without the collective efforts and contributions of everyone mentioned above.

Executive Summary

The Patient Record Management System is a web-based application developed to streamline the process of storing, retrieving, and managing patient information in a digital format. Designed using Node.js, Express.js, EJS, and MongoDB, the system aims to replace traditional paper-based records with a more efficient, reliable, and scalable solution. It allows users to perform key operations such as adding new patients, updating existing details, deleting records, and searching for patient data in real-time using a user-friendly interface.

The backend is powered by Express and MongoDB, with Mongoose handling data modeling and validation. The frontend uses EJS templates for dynamic rendering and jQuery for enhanced interactivity. The system architecture follows an MVC (Model-View- Controller) pattern, ensuring a clean separation of concerns and easy maintainability.

This project was developed with a focus on simplicity, functionality, and expandability, making it suitable for use in small clinics, educational institutions, or prototype development. It not only improves operational efficiency but also enhances the accuracy and accessibility of medical records. The successful implementation of this system demonstrates core concepts of full-stack development, database integration, and responsive design, providing a strong foundation for future enhancements like user authentication and cloud deployment.

CONTENTS	Page No.
Acknowledgement	iii
Executive Summary	iv
Table of Contents	v
List of Figures	vi
List of Tables	vii
Abbreviations	viii
Symbols and Notations	ix
1 INTRODUCTION	1-2
1.1 Objective	1
1.2 Motivation	1
1.3 Background	1-2
2 PROJECT DESCRIPTION AND GOALS	3-4
3 TECHNICAL SPECIFICATION	5-7
4 DESIGN APPROACH AND DETAILS (as applicable)	8-11
4.1 Design Approach / Materials & Methods	8
4.2 Codes and Standards	9-10
4.3 Constraints, Alternatives and Tradeoffs	10-11

5	SCHEDULE, TASKS AND MILESTONES	12
6	PROJECT DEMONSTRATION	13-60
6.1	Sample Codes	13-50
6.2	Sample Screen Shots	51-55
7	COST ANALYSIS / RESULT & DISCUSSION (as applicable)	56-57
8	SUMMARY	58
9	REFERENCES	59

APPENDIX A

List of Figures		
Figure No.	Title	Page No.
1.1	arya.jpg	19
1.2	yash.jpg	20
1.3	Background.jpg	20
2.1	home.html	51
2.2	Sign up page	52
2.3	Login page	53
2.4	Add patient page	54
2.5	CRUD working page	54
2.6	About page	55
2.7	Contact page	55

List of Table

	List of Tables	
Table No.	Title	Page No.
1	Abbreviation Table	vii
2	Symbols and notations	viii

List of Abbreviations

API Interface	Application Programming
DB	Database
UI	User Interface
UX	User Experience
CRUD	Create, Read, Update, Delete
HTTP	HyperText Transfer Protocol
HTTPS Secure	HyperText Transfer Protocol
JSON	JavaScript Object Notation
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JS	JavaScript
MVC	Model-View-Controller
MFA	Multi-Factor Authentication
REST Transfer	Representational State
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Symbols and Notations

Symbol / Notation	Meaning
<code>req</code>	Request object in Express
<code>res</code>	Response object in Express
<code>=></code>	Arrow function notation in JavaScript
<code>{}</code>	Object or block in JavaScript syntax
<code>[]</code>	Array or optional parameters
<code>/api/...</code>	REST API endpoint route path
<code>req.body</code>	Access posted data from forms or APIs
<code>req.params</code>	Access route parameters
<code>req.query</code>	Access query string parameters

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this project is to develop a web-based Patient Record Management System using Node.js, Express, and EJS that allows medical institutions to efficiently manage and store patient information. This system simplifies the handling of patient data such as personal details, medical history, and visit records. The application provides CRUD (Create, Read, Update, Delete) functionalities, streamlining administrative operations and improving the accuracy of medical records.

1.2 Motivation

In many healthcare environments, especially smaller clinics, patient data is still managed through manual or fragmented digital methods, which can lead to inefficiencies, errors, and data loss. The motivation behind this project stems from the need to automate and centralize patient data management. By creating a lightweight, easy-to-use system, healthcare providers can spend less time on paperwork and more time focusing on patient care.

1.3 Background

With the growing adoption of digital solutions in healthcare, Electronic Health Records (EHR) have become increasingly important. This project uses Node.js and Express to create a server-side application that stores patient information in a structured format, rendered via EJS templates on the frontend. The system integrates

with a simple JSON-based storage model, simulating a lightweight database. The project structure includes RESTful routes, modular models, and dynamic web pages to provide a fully functional patient record system suitable for educational, prototype, small-scale clinic.

PROJECT DESCRIPTION

The Patient Record Management System is a web-based application designed to store, retrieve, update, and delete patient data in an efficient and organized manner. Built using Node.js, Express.js, and EJS, the system provides a user-friendly interface for managing patient records, including their name, contact details, gender, age, and medical conditions. The backend is responsible for handling routing and data logic, while the frontend dynamically displays the information using server-rendered templates.

The application serves as a simplified alternative to complex electronic medical record systems, making it ideal for educational purposes, prototype development, or use in small clinics where full-scale hospital management systems are not feasible.

Key features include:

- Add new patient records
- View a list of all patients
- Edit existing patient details
- Delete patient records
- Search functionality using jQuery for real-time filtering

The system maintains patient data in a JSON format (patients.json) and dynamically renders the user interface with EJS, allowing smooth integration of logic and design.

Project Goals

The goals of this project are as follows:

- **Simplicity and Accessibility**

Build an easy-to-use system that minimizes the learning curve for users, ensuring healthcare professionals can adopt it quickly without technical expertise.

- **Data Accuracy and Management**

Ensure reliable and structured storage of patient information to minimize manual errors and improve data consistency.

- **Performance and Efficiency**

Provide fast CRUD operations with minimal latency for improved workflow efficiency.

- **Customizability and Scalability**

Lay the groundwork for future expansion, such as integrating authentication, appointment scheduling, or migrating from a JSON-based system to a real database (e.g., MongoDB).

- **Educational Value**

Serve as a practical learning tool for understanding full- stack development concepts using JavaScript technologies.

TECHNICAL SPECIFICATIONS

- **Frontend**

- **Technology Used:**

- **EJS (Embedded JavaScript Templates):** Used to dynamically render HTML pages with server-side data.
 - **HTML5 & CSS3:** For structure and basic styling.
 - **Bootstrap (optional):** Can be integrated for responsive design and UI components.
 - **jQuery:** Used for DOM manipulation and implementing the real-time search/filter functionality on the patient list page.

- **Backend**

- **Runtime Environment:**

- **Node.js:** JavaScript runtime used to build scalable network applications.

- **Framework:**

- **Express.js:** Lightweight web framework for handling routes and middleware.

- **Routing & Logic:**
 - RESTful routing structure to manage CRUD operations.
 - Middleware used for request parsing (express.urlencoded) and static file handling.
- **Data Management**
 - **Data Format:**
 - **JSON File (patient.json):** Acts as the data store simulating a database. All patient records are stored in this file.
 - **Operations:**
 - Data is read and written using Node.js fs module.
 - NoSQL-like structure using array of objects, each representing a patient.
- **Database Integration**
 - **Database Used:**
 - **MongoDB:** NoSQL database used to store patient records in a flexible, document-oriented format.

- **ODM (Object Data Modeling):**
 - **Mongoose:** A Node.js library used to define schemas, models, and interact with MongoDB easily.
- **Dependencies**
 - **express** – Web framework for routing and middleware.
 - **ejs** – Template engine for rendering HTML.
 - **body-parser** (built-in in Express v4.16+) – For parsing form data.
 - **fs (File System)** – Node.js module for reading/writing the JSON file

DESIGN APPROACH AND DETAILS

4.1 Design Approach / Materials & Methods

The system is designed using a **Model-View-Controller (MVC)**

architecture, which separates concerns to enhance maintainability and scalability:

- **Model (Mongoose Schema):**

Defines the structure of patient data using a schema that maps to a MongoDB collection. All database operations such as insert, update, delete, and query are handled via Mongoose.

- **View (EJS Templates):**

Uses Embedded JavaScript Templates (EJS) for rendering HTML pages dynamically with backend data. Views are responsible for displaying forms and patient lists with styling and search functionalities handled using HTML, CSS, and jQuery.

- **Controller (Express Routes):**

Manages business logic, handles HTTP requests, connects to the database, and determines which views to render. Controllers also validate input and manage response flow.

Materials & Methods:

- **Tools:** Node.js, Express, MongoDB, Mongoose, EJS, jQuery

- **Methodology:** Incremental development and testing; route-based request handling; server-side rendering
- **Data Handling:** MongoDB used to store patient records in BSON format, retrieved via Mongoose and rendered using EJS.

4.2 Codes and Standards

While not governed by formal engineering standards, the project adheres to best practices in web application development:

- **JavaScript ES6+ conventions** for code clarity and modularity.
- **ExpressJS** routing follows RESTful API standards for CRUD operations.
- **MongoDB schema modeling via Mongoose** ensures validation, consistency, and data integrity.
- **HTML5/CSS3** and optional **Bootstrap 4/5** for responsive UI development.
- **Project structure standards:**
 - /models: Mongoose schema definitions

- /views: EJS templates
 - /routes: Express route handlers
 - /config: Database configuration
- Code is formatted for readability, with modularization (require and module.exports) for reusability.

4.3 Constraints, Alternatives and Tradeoffs

Constraints

- **No authentication or user roles:** The system does not implement login functionality, which limits its suitability for multi-user environments.
- **No frontend frameworks:** The UI relies on jQuery and EJS rather than React, Vue, or Angular, which limits interactivity and modern UX features.
- **Data hosted locally:** In the current form, MongoDB is expected to run locally unless configured with a cloud instance (e.g., MongoDB Atlas).

Alternatives Considered:

- **Data Storage:**
 - Initially implemented using patient.json for simplicity and offline use.

- Migrated to **MongoDB** for better scalability, reliability, and query capabilities.
- **Frontend Rendering:**
 - EJS was chosen over more complex frontend frameworks like React or Angular to keep the project lightweight and server-rendered.
- **Database Drivers:**
 - Chose **Mongoose** for ease of schema creation and data validation instead of using MongoDB's native driver.

Tradeoffs

- **Simplicity vs. Functionality:**

Opting for EJS and basic HTML makes the system simple to learn and use but limits UI/UX sophistication.
- **Local JSON vs. MongoDB:**

JSON files are beginner-friendly but not suitable for concurrent access or scalability—hence replaced with MongoDB for reliability.
- **No REST API/SPA:**

The project doesn't expose a public API or use frontend frameworks; while this reduces complexity, it limits integration and client-side control.

SCHEDULE, TASKS AND MILESTONES

Phase	Key Tasks	Milestones
1. Requirement Analysis & Planning	<ul style="list-style-type: none">- Identify project scope and features- Select tech stack (Node.js, Express, EJS, MongoDB)	Project proposal and feature list finalized
2. Environment Setup	<ul style="list-style-type: none">- Initialize Node.js project- Install dependencies (Express, EJS, Mongoose)- Setup MongoDB connection	Development environment ready
3. Backend Development	<ul style="list-style-type: none">- Design patient schema (Mongoose)- Setup Express server and RESTful routes- Implement MongoDB CRUD operations	Functional backend API completed
4. Frontend Integration	<ul style="list-style-type: none">- Create EJS templates for listing, adding, editing patients- Integrate with backend logic- Add jQuery-based search	Basic frontend UI functional
5. Testing & Debugging	<ul style="list-style-type: none">- Unit test data operations- Fix bugs in data rendering and form submission- Validate frontend functionality	Bug-free CRUD flow verified
6. Finalization & Documentation	<ul style="list-style-type: none">- Prepare project report- Finalize UI/UX	System finalized and documented

PROJECT DEMONSTRATION

Code:

1. about.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8" />
```

```
<title>About - Patient Record System</title>
```

```
<link rel="stylesheet" href="style.css" />
```

```
<link
```

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
```

```
<style>
```

```
.developer
```

```
-card {
```

```
display:
```

```
flex;
```

```
flex-direction:
```

```
column; align-
```

```
items: center;
```

```
}
```

```
.developer-card img {
```

```
width: 150px;
```

```
height: 150px;
```



```

    object-fit: cover;

    border-radius: 20%;

    margin-bottom: 1rem;

}

```

```

.developer-card a {
    margin: 0 0.1rem;
}

```

```
</style>
```

```
</head>
```

```
<body>
```

```
<!-- Navigation Bar -->
```

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark px-3">
```

```
    <a class="navbar-brand fw-bold"
```

```
href="/home.html">PatientPortal</a>
```

```

    <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target="#navbarNav">

```

```
    <span class="navbar-toggler-icon"></span>
```

```
</button>
```

```
<div class="collapse navbar-collapse justify-content-end" id="navbarNav">
```

```

<ul class="navbar-nav">

    <li class="nav-item"><a class="nav-
link" href="/home.html">Home</a></li>

    <li class="nav-item"><a class="nav-
link" href="/signup.html">Sign Up</a></li>

    <li class="nav-item"><a class="nav-link" href="/login.html">Log In</a></li>

    <li class="nav-item"><a class="nav-link active"
href="/about.html">About</a></li>

    <li class="nav-item"><a class="nav-
link" href="/contact.html">Contact</a></li>

</ul>

</div>

</nav>

```

```

<!-- About Section -->

```

```

<section class="container mt-5">

```

```

<h1 class="display-5 fw-semibold mb-4">About the Patient Record System</h1>

```

```

<p class="lead">

```

This Patient Record System is built to simplify and secure the way doctors manage patient records. Features include:

```

</p>

```

```

<ul class="list-group list-group-flush mb-4">

```

```
<li class="list-group-item">+ Add patient records with ease</li>
```

```
<li class="list-group-item">🔍 View and search patients quickly</li>
```

```
<li class="list-group-item">✏️ Edit or update medical  
information</li>
```

```
<li class="list-group-item">🗑️ Delete records securely</li>
```

```
</ul>
```

```
<p>
```

Built using Node.js, Express, MongoDB, EJS, and Bootstrap.
Secure, scalable, and ready to use.

```
</p>
```

```
</section>
```

```
<!-- Developer Section -->
```

```
<section class="container mb-5">
```

```
<hr />
```

```
<h3 class="fw-bold mb-4">👤 Developed By</h3>
```

```
<div class="row text-center">
```

```
<!-- Arya Card -->
```

```
<div class="col-md-6 mb-4">
```

```
<div class="card p-4 developer-card shadow-sm">
```

```

```

```
<h5 class="fw-bold mt-2">Arya Mulay</h5>
```

<div>

LinkedIn

GitHub

</div>

</div>

</div>

<!-- Yash Card -->

<div class="col-md-6 mb-4">

<div class="card p-4 developer-card shadow-sm">

<h5 class="fw-bold mt-2">Yash Singh</h5>

<div>

LinkedIn


```
<a href="https://github.com/yashsingh" target="_blank" class="btn btn-  
outline-dark btn-sm">
```

```
  GitHub
```

```
</a>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
</section>
```

```
<script  
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap  
.bundle.min.js"></script>
```

```
</body>
```

```
</html>
```

2. app.js:

```
document.addEventListener('DOMContentLoaded', () => {  
  
  const searchBtn = document.getElementById('searchBtn');  
  
  const searchInput = document.getElementById('searchInput');  
  
  const patientCards = document.querySelectorAll('.patient-card');  
  
  
  searchBtn.addEventListener('click', () => {  
  
    const searchTerm = searchInput.value.trim().toLowerCase();
```

```
patientCards.forEach(card => {  
  
    const nameElement = card.querySelector('.patient-name');  
  
    const name = nameElement ? nameElement.textContent.toLowerCase() : "";  
  
    if (name.includes(searchTerm)) {  
  
        card.style.display = 'block';  
  
    } else {  
  
        card.style.display = 'none';  
  
    }  
  
});  
  
});  
  
});
```

3. **arya.jpg:**



Fig 1.1: arya.jpg

4. yash.jpg:



Fig 1.2: yash.jpg

5. background.jpg:



Fig 1.3: background.jpg

6. contact.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8" />
```

```
<title>Contact Us - Patient Record System</title>
```

```
<link rel="stylesheet" href="style.css" />
```

```
<link
```

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
```

```
</head>
```

```
<body>
```

```
<!-- Navigation Bar -->
```

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark px-3">
```

```
<a class="navbar-brand fw-bold" href="/home.html">PatientPortal</a>
```

```
<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarNav">
```

```
<span class="navbar-toggler-icon"></span>
```

```
</button>
```

```
<div class="collapse navbar-collapse justify-content-end" id="navbarNav">
```

```
<ul class="navbar-nav">
```

```
<li class="nav-item"><a class="nav-link"
href="/home.html">Home</a></li>
```

```
<li class="nav-item"><a class="nav-link" href="/signup.html">Sign
```


Up

<li class="nav-item">Log
In

<li class="nav-item"><a class="nav-link"
href="/about.html">About

<li class="nav-item"><a class="nav-link active"
href="/contact.html">Contact

</div>

</nav>

<!-- Contact Section -->

<section class="container mt-5 mb-5">

<h1 class="display-5 fw-semibold mb-4">Get in Touch</h1>

<p class="lead">We'd love to hear your feedback or help with any
queries.</p>

<form>

<div class="mb-3">

<label for="name" class="form-label">Your Name</label>

<input type="text" class="form-control" id="name" placeholder="Enter

```
your name" />
```

```
</div>
```

```
<div class="mb-3">
```

```
<label for="email" class="form-label">Your Email</label>
```

```
<input type="email" class="form-control" id="email"  
placeholder="name@example.com" />
```

```
</div>
```

```
<div class="mb-3">
```

```
<label for="message" class="form-label">Your Message</label>
```

```
<textarea class="form-control" id="message" rows="5"  
placeholder="Write your message here..."></textarea>
```

```
</div>
```

```
<button type="submit" class="btn btn-primary">Send Message</button>
```

```
</form>
```

```
</section>
```

```
<script
```

```
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
```

```
</body>
```

```
</html>
```

7. **logout.html:**

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8" />
```

```
<title>Logged Out</title>
```

```
<link rel="stylesheet" href="style.css" />
```

```
<link
```

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
```

```
</head>
```

```
<body>
```

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark px-3">
```

```
<a class="navbar-brand fw-bold" href="/home.html">PatientPortal</a>
```

```
<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarNav">
```

```
<span class="navbar-toggler-icon"></span>
```

```

</button>

<div class="collapse navbar-collapse justify-content-end" id="navbarNav">

  <ul class="navbar-nav">

    <li class="nav-item"><a class="nav-link active"
href="/home.html">Home</a></li>

    <li class="nav-item"><a class="nav-link" href="/signup.html">Sign
Up</a></li>

    <li class="nav-item"><a class="nav-link" href="/login.html">Log
In</a></li>

    <li class="nav-item"><a class="nav-link"
href="/about.html">About</a></li>

    <li class="nav-item"><a class="nav-link"
href="/contact.html">Contact</a></li>

  </ul>

</div>

</nav>

<div class="container text-center mt-5">

  <h2 class="text-success mb-4">You have successfully logged out.</h2>

  <a href="/login.html" class="btn btn-primary">Log In Again</a>

  <a href="/home.html" class="btn btn-secondary ms-2">Back to Home</a>

</div>

```

</body>

</html>

8. Patient.js:

<!DOCTYPE html>

<html>

<head>

<title>Patients</title>

<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="/style.css">

</head>

<body data-doctor="<%= doctor.username %>">

<div class="container mt-4">

<h3>Welcome, Dr. <%= doctor.username %> 🏥 </h3>

<div class="input-group mb-3">

<input type="text" id="searchInput" placeholder="Search patients by
name..." class="form-control">

<button class="btn btn-primary" id="searchBtn">Search</button>

</div>

<div id="patientList" class="row">

```

<% patients.forEach(patient => { %>

  <div class="col-md-4 mb-3 patient-card">

    <div class="card p-3">

      <h5 class="patient-name"><%= patient.name %></h5>

      <p>Age: <%= patient.age %></p>

      <p>Disease: <%= patient.disease %></p>

      <!-- Edit -->

      <form action="/edit/<%= patient._id %>" method="POST"
class="mb-1">

        <input type="text" name="name" placeholder="New name"
class="form-control mb-1">

        <input type="number" name="age" placeholder="New age"
class="form-control mb-1">

        <input type="text" name="disease" placeholder="New disease"
class="form-control mb-1">

        <input type="hidden" name="gender" value="<%= patient.gender %>">

        <button class="btn btn-primary btn-sm">Edit</button>

      </form>

      <!-- Delete -->

      <form action="/delete/<%= patient._id %>" method="POST"
onsubmit="return confirm('Delete patient?')">

        <button class="btn btn-danger btn-sm">Delete</button>

      </form>

    </div>

```

```

    </div>

    <% }) %>

</div>
<a href="/logout.html" class="btn btn-outline- danger">Logout</a><br/>
<a href="/add.html" class="btn btn-success mt-3">+ Add New Patient</a>
</div>

<!-- Load app.js correctly -->

<script src="/app.js"></script>

</body>

</html>

```

9. Patient.json:

```

[
  {
    "name": "yash",
    "age": "19",
    "disease": "knee
    pain",
    "doctor":
    "asm1323"
  },
  {

```

```
"name": "chaitanya",

"age": "22",
"disease": "coronary heart disease",
"doctor": "asm1323"
},

{

"name": "joseph",

"age": "19",

"disease":
"pneumonia",
"doctor": "asm1323"
},

{

"name": "siddhart",

"age": "19",

"disease":
"migrane",
"doctor":
"asm1323"
},

{

"name": "Siddharth Goutam Kumar", "age":
"21",
"disease": "Coronary Heart Disease",
"doctor": "yash123"
},

{
```



```

    "name": "yash singh",
    "age": "25",

    "disease":
    "stomacache",
    "doctor": "yash123"
  },

  {

    "name": "Vedansh Gupta",
    "age": "21",
    "disease":
    "fever",
    "doctor":
    "yash123"
  }
]

```

10. server.js:

```

const express = require('express');

const mongoose = require('mongoose'); const
session = require('express-session'); const
bodyParser = require('body-parser'); const path
= require('path');
const fs = require('fs');

const app = express();
app.use(express.json());

```

```
app.use(bodyParser.urlencoded({ extended: true })); app.use(express.static('public'));
app.use(express.static(path.join(_____dirname, 'public')));
```

```
app.set('view engine',
'ejs'); app.set('views',
_____
dirname);
```

```
app.use(session({
  secret: 'secret-key',
  resave: false,
  saveUninitialized: false
}));
```

```
const Doctor = require('./models').Doctor;
const Patient = require('./models').Patient;
```

```
mongoose.connect('mongodb://127.0.0.1:27017/patientDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log('MongoDB Connected'));
```

```
// Middleware to check login
```

```
function isAuthenticated(req, res, next) {
```

```

    if (req.session.doctor) return next(); res.redirect('/login.html');
  }

  // Serve home

  app.get('/', (req, res) => {

    res.sendFile(_____dirname + '/public/home.html');

  });

  // Signup route

  app.post('/signup', async (req, res) => { try
  {
    const { username, password } = req.body;

    if (!username || username.length < 4) {

      return res.json({ success: false, message: 'Username must be at least 4
characters.' });
    }

    const passwordRegex = /^(?=.*[a-zA-Z])(?=.*\d)(?=.*[W_]).{6,}$/; if
    (!password || !passwordRegex.test(password)) {
      return
      res.json
      ({
        success
        : false,

```

message: 'Password must be at least 6 characters, contain a special character and be alphanumeric.'

```
});
```

```
}
```

```
const existing = await Doctor.findOne({ username }); if  
(existing) {  
  return res.json({ success: false, message: 'Username already exists' });
```

```
}
```

```
const newDoctor = new Doctor({ username, password }); await  
newDoctor.save();
```

```
req.session.doctor = newDoctor; // Auto-login after signup res.json({ success:  
true });
```

```
} catch (err) {
```

```
  console.error("Signup Error:", err);
```

```
  res.status(500).json({ success: false, message: "Internal Server Error" });
```

```
}
```

```
});
```

```

// Login route

app.post('/login', async (req, res) => { try
{
  const { username, password } = req.body;

  if (!username || !password) {

    return res.json({ success: false, message: "Missing credentials" });

  }

  const doctor = await Doctor.findOne({ username, password }); if
  (!doctor) {
    return res.json({ success: false, message: "Invalid username or password" });
  }

  req.session.doctor = doctor;

  res.json({ success: true, username: doctor.username });

} catch (err) {

  console.error("Login Error:", err);

  res.status(500).json({ success: false, message: "Internal Server Error" });
}
}

```

```

    }

  });

// Doctor-specific patient view

app.get('/doctor/:username/patients', isAuthenticated, async (req, res) => {

  const doctor = await Doctor.findOne({ username:
req.params.username });

  if (!doctor) return res.send("Doctor not found");

  const patients = await Patient.find({ doctor: doctor.username });
  res.render('patients', { doctor, patients });
});

// Add patient

app.post('/add', isAuthenticated, async (req, res) => { try {
  const { name, age, disease } = req.body; const
  doctor = req.session.doctor.username;

  const newPatient = new Patient({ name, age, disease, doctor }); await
  newPatient.save();

```

```

const filePath = path.join(_____dirname, 'patient.json');

let patientList = [];

if (fs.existsSync(filePath)) {

    const data = fs.readFileSync(filePath, 'utf-8'); patientList =
    JSON.parse(data || '[]');
}

patientList.push({ name, age, disease, doctor });

fs.writeFileSync(filePath, JSON.stringify(patientList, null, 2), 'utf-8');

res.status(200).json({ success: true });

} catch (err) {

    console.error("Add Patient Error:", err);

    res.status(500).json({ success: false, error: "Internal Server Error"
});

}

});

// Edit patient

app.post('/edit/:id', isAuthenticated, async (req, res) => { const {
    name, age, disease } = req.body;

```

```

const updated = {};

if (name) updated.name = name;
if (age) updated.age = age;
if (disease) updated.disease = disease;

await Patient.findByIdAndUpdate(req.params.id, updated);
res.redirect(`/doctor/${req.session.doctor.username}/patients`);
});

// Delete patient

app.post('/delete/:id', isAuthenticated, async (req, res) => { await
  Patient.findByIdAndDelete(req.params.id);
  res.redirect(`/doctor/${req.session.doctor.username}/patients`);
});

// Logout

app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.send("Error logging out");
    res.redirect('/logout.html');
  });
});

```



```
app.listen(3001, () => {

  console.log(`Server running at http://localhost:3001/home.html`);

});
```

11. add.html:

```
<!-- public/add.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Add Patient</title>
  <link                                rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"/
>
  <style>
    body {
      background-color: #f7f7f7;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    }

    .container {
      max-width: 600px;
      margin-top: 60px;
      background-color: white;
      padding: 30px;
      box-shadow: 0 4px 12px rgba(0,0,0,0.1);
      border-radius: 10px;
    }

    .form-group label {
```

```
font-weight: 600;
}
```

```
h2 {
  text-align: center;
  margin-bottom: 30px;
  font-weight: 700;
  color: #333;
}
```

```
.btn-primary, .btn-secondary {
  width: 100%;
  padding: 10px;
  font-weight: bold;
  border-radius: 5px;
}
```

```
.btn-primary:hover {
  background-color: #0056b3;
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="container">
```

```
<h2>Add New Patient</h2>
```

```
<form id="addForm">
```

```
<div class="form-group">
```

```
<label for="name">Patient Name</label>
```

```
<input type="text" class="form-control" id="name" name="name"
required/>
```

```
</div>
```

```
<div class="form-group">
```

```
<label for="age">Age</label>
```

```
<input type="number" class="form-control" id="age" name="age"
```

```

required/>
</div>
<div class="form-group">
  <label for="disease">Disease</label>
  <input type="text" class="form-control" id="disease" name="disease"
required/>
</div>
<button type="submit" class="btn btn-primary">Add Patient</button>
</form>

<a id="backBtn" class="btn btn-secondary mt-3">← Back</a>
</div>

<script>
  document.getElementById('addForm').addEventListener('submit',      async
function(e) {
  e.preventDefault();
  const name = document.getElementById('name').value;
  const age = document.getElementById('age').value;
  const disease = document.getElementById('disease').value;
  //implemnted using ajax - add function (add.html)
  const response = await fetch('/add', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ name, age, disease }) // doctor will be picked from
session
  });

  const result = await response.json();
  if (result.success) {
    const username = sessionStorage.getItem("doctorUsername");

```

```

        window.location.href = `/doctor/${username}/patients`;
    } else {
        alert("Something went wrong!");
    }
});
const username = sessionStorage.getItem("doctorUsername");
document.getElementById("backBtn").href = username ?
`/doctor/${username}/patients` : "/";
</script>
</body>
</html>

```

12. login.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Doctor Login</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link rel="stylesheet" href="/style.css" />
</head>
<style>
    .btn-secondary {
        padding: 10px;
        font-weight: bold;
        border-radius: 5px;
    }
    .back-btn-pos {
        display: flex;
        flex-direction: column;
    }
</style>

```

```

<body class="d-flex justify-content-center align-items-center vh-100 bg-light">
  <div class="back-btn-pos">
    <div class="card p-4 shadow" style="width: 22rem;">
      <h3 class="mb-3 text-center">Login</h3>
      <form id="loginForm" novalidate>
        <div class="mb-3">
          <label for="username" class="form-label">Username</label>
          <input type="text" class="form-control" id="username"
name="username" required />
          <div class="invalid-feedback">Username must be at least 4
characters.</div>
        </div>
        <div class="mb-3">
          <label for="password" class="form-label">Password</label>
          <input type="password" class="form-control" id="password"
name="password" required />
          <div class="invalid-feedback">Password must be alphanumeric and
contain a special character.</div>
        </div>
        <button type="submit" class="btn btn-primary w-100">Login</button>
      </form>
      <p class="mt-3 text-center">Don't have an account? <a
href="/signup.html">Signup</a></p>
    </div>

    <a id="backBtn" class="btn btn-secondary mt-3">⬅ Back</a>
  </div>

  <script>
    document.getElementById('loginForm').addEventListener('submit', async
function(e) {
      e.preventDefault();

```

```

const usernameField = document.getElementById('username');
const passwordField = document.getElementById('password');

const username = usernameField.value.trim();
const password = passwordField.value.trim();

const passwordRegex = /^(?=.*[a-zA-Z])(?=.*\d)(?=.*[\W_]).{6,}$/;

let valid = true;

if (username.length < 4) {
  usernameField.classList.add('is-invalid');
  valid = false;
} else {
  usernameField.classList.remove('is-invalid');
}

if (!passwordRegex.test(password)) {
  passwordField.classList.add('is-invalid');
  valid = false;
} else {
  passwordField.classList.remove('is-invalid');
}

if (!valid) return;

const response = await fetch('/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ username, password })
});

const result = await response.json();

```

```

    if (result.success) {
        sessionStorage.setItem('doctorUsername', username);
        window.location.href = `/doctor/${username}/patients`;
    } else {
        alert("Login failed");
    }
});

const username = sessionStorage.getItem("doctorUsername");
document.getElementById("backBtn").href = "/home.html";
</script>
</body>
</html>

```

13. signup.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Doctor Signup</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link rel="stylesheet" href="/style.css" />
</head>
<style>
    .btn-secondary {
        padding: 10px;
        font-weight: bold;
        border-radius: 5px;
    }
    .back-btn-pos {

```

```

display: flex;
flex-direction: column;
}

```

```

</style>

```

```

<body class="d-flex justify-content-center align-items-center vh-100 bg-light">

```

```

  <div class="back-btn-pos">

```

```

    <div class="card p-4 shadow" style="width: 22rem;">

```

```

      <h3 class="mb-3 text-center">Signup</h3>

```

```

      <form id="signupForm" novalidate>

```

```

        <div class="mb-3">

```

```

          <label for="username" class="form-label">Choose Username</label>

```

```

          <input      type="text"      class="form-control"      id="username"

```

```

name="username" required />

```

```

          <div class="invalid-feedback">Username must be at least 4
characters.</div>

```

```

        </div>

```

```

        <div class="mb-3">

```

```

          <label for="password" class="form-label">Create Password</label>

```

```

          <input      type="password"      class="form-control"      id="password"

```

```

name="password" required />

```

```

          <div class="invalid-feedback">Password must be alphanumeric and
contain a special character.</div>

```

```

        </div>

```

```

        <button type="submit" class="btn btn-primary w-100">Signup</button>

```

```

      </form>

```

```

      <p class="mt-3 text-center">Already have an account? <a
href="/login.html">Login</a></p>

```

```

    </div>

```

```

    <a id="backBtn" class="btn btn-secondary mt-3">⬅ Back</a>

```

```

  </div>

```



```

<script>
  document.getElementById('signupForm').addEventListener('submit',    async
function(e) {
  e.preventDefault();

  const usernameField = document.getElementById('username');
  const passwordField = document.getElementById('password');

  const username = usernameField.value.trim();
  const password = passwordField.value.trim();

  const passwordRegex = /^(?=.*[a-zA-Z])(?=.*\d)(?=.*[\W_]).{6,}$/;

  let valid = true;

  if (username.length < 4) {
    usernameField.classList.add('is-invalid');
    valid = false;
  } else {
    usernameField.classList.remove('is-invalid');
  }

  if (!passwordRegex.test(password)) {
    passwordField.classList.add('is-invalid');
    valid = false;
  } else {
    passwordField.classList.remove('is-invalid');
  }

  if (!valid) return;

  const response = await fetch('/signup', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },

```

```

        body: JSON.stringify({ username, password })
    });

    const result = await response.json();

    if (result.success) {
        sessionStorage.setItem('doctorUsername', username);
        window.location.href = `/doctor/${username}/patients`;
    } else {
        alert(result.message || "Signup failed");
    }
    });

    const username = sessionStorage.getItem("doctorUsername");
    document.getElementById("backBtn").href = "/home.html";
</script>
</body>
</html>

```

14. patient.ejs:

```

<!DOCTYPE html>
<html>
<head>
    <title>Patients</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
    <link rel="stylesheet" href="/style.css">
</head>
<body data-doctor="<%= doctor.username %>">
    <div class="container mt-4">
        <h3>Welcome, Dr. <%= doctor.username %> 🧑🏻 </h3>

```

```

<div class="input-group mb-3">
  <input type="text" id="searchInput" placeholder="Search patients by
name..." class="form-control">
  <button class="btn btn-primary" id="searchBtn">Search</button>
</div>

<div id="patientList" class="row">
  <% patients.forEach(patient => { %>
    <div class="col-md-4 mb-3 patient-card">
      <div class="card p-3">
        <h5 class="patient-name"><%= patient.name %></h5>
        <p>Age: <%= patient.age %></p>
        <p>Disease: <%= patient.disease %></p>

        <!-- Edit -->
        <form action="/edit/<%= patient._id %>" method="POST" class="mb-
1">
          <input type="text" name="name" placeholder="New name"
class="form-control mb-1">
          <input type="number" name="age" placeholder="New age"
class="form-control mb-1">
          <input type="text" name="disease" placeholder="New disease"
class="form-control mb-1">
          <input type="hidden" name="gender" value="<%= patient.gender
%>">
          <button class="btn btn-primary btn-sm">Edit</button>
        </form>

        <!-- Delete -->
        <form action="/delete/<%= patient._id %>" method="POST"
onsubmit="return confirm('Delete patient?')">
          <button class="btn btn-danger btn-sm">Delete</button>
        </form>
      </div>
    } %>
  </div>

```

```

    </div>
    <% }) %>
</div>

<a href="/logout.html" class="btn btn-outline-danger">Logout</a><br/>
<a href="/add.html" class="btn btn-success mt-3">+ Add New Patient</a>
</div>

<!-- Load app.js correctly -->
<script src="/app.js"></script>
</body>
</html>

```

15. db.js:

```

const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/patientSystem', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>{
  console.log("Database Connected");
})

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

module.exports = db;

```

16. models.js:

```

const mongoose = require('mongoose');

```

```
const doctorSchema = new mongoose.Schema({
  username: String,
  password: String
});

const patientSchema = new mongoose.Schema({
  name: String,
  age: Number,
  gender: String,
  disease: String,
  doctor: String
});

const Doctor = mongoose.model('Doctor', doctorSchema);
const Patient = mongoose.model('Patient', patientSchema);

module.exports = { Doctor, Patient };
```

Features (Sample Output):

1. Home page:

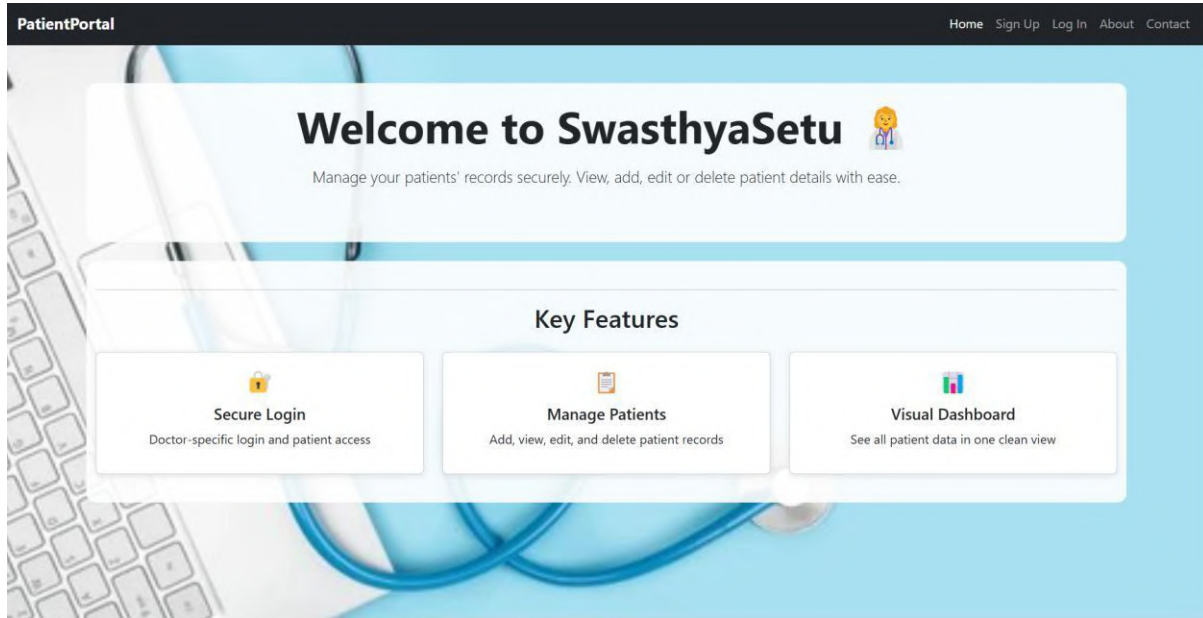


Fig 2.1: home page

2. Sign up page:

Signup

Choose Username

yash123

Create Password

.....

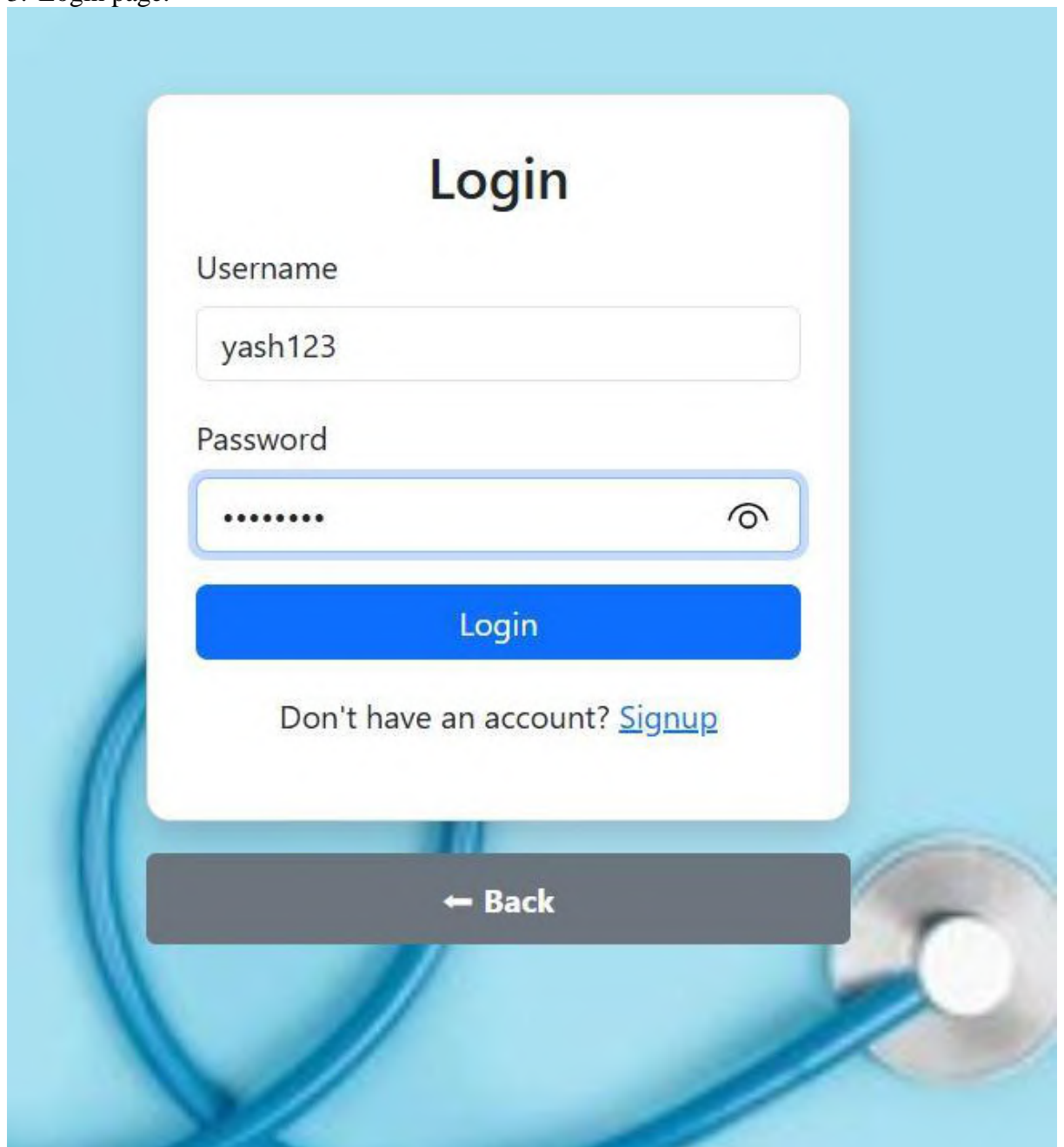
Signup

Already have an account? [Login](#)

← Back

Fig 2.2: signup page

3. Login page:

The image shows a login page with a light blue background featuring a blurred stethoscope. A white rounded rectangle contains the login form. At the top of this rectangle is the word "Login" in bold black text. Below it are two input fields: "Username" with the text "yash123" and "Password" with masked characters ".....". The password field has a blue border and a toggle icon on the right. Below the fields is a blue "Login" button. Underneath the button is the text "Don't have an account?" followed by a blue "Signup" link. At the bottom of the page is a dark grey button with a left arrow and the text "Back".

Login

Username

yash123

Password

.....

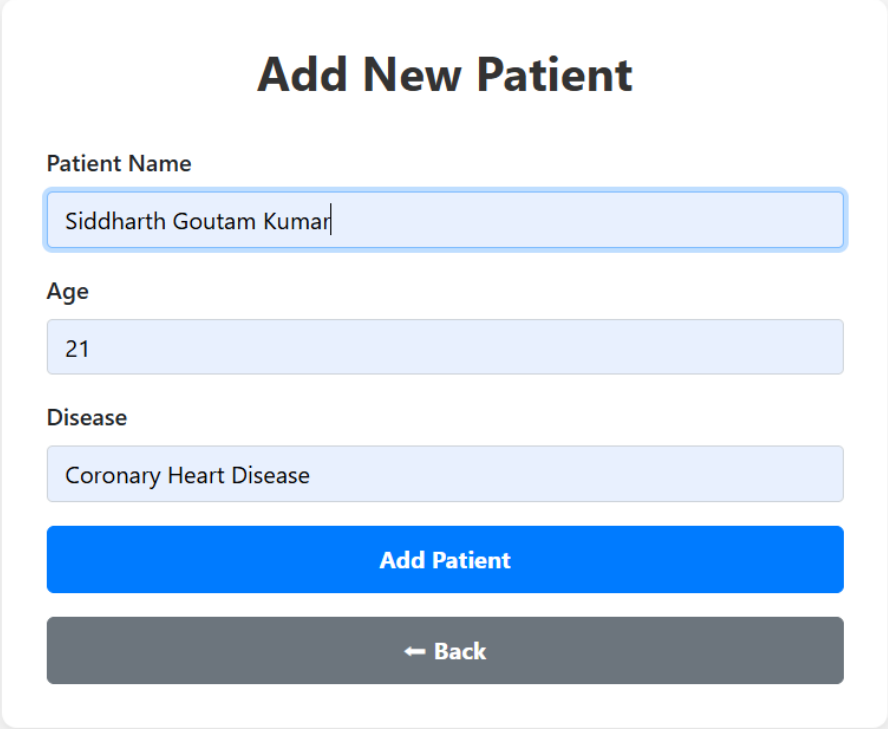
Login

Don't have an account? [Signup](#)

← Back

Fig 2.3: login page

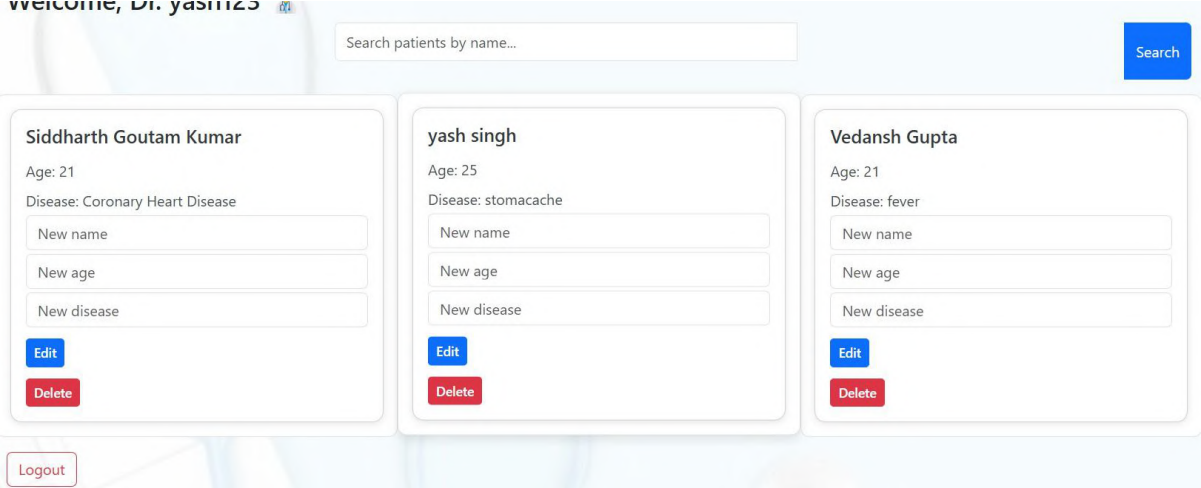
4. Adding new patient:



The image shows a web form titled "Add New Patient". It contains three input fields: "Patient Name" with the text "Siddharth Goutam Kumar", "Age" with the text "21", and "Disease" with the text "Coronary Heart Disease". Below the input fields are two buttons: a blue "Add Patient" button and a grey "← Back" button.

Fig: 2.4: add patient route(working session)

5. Edit delete and search options:-



The image shows a web interface for a medical application. At the top, there is a "welcome, Dr. yash123" message and a search bar with the placeholder text "Search patients by name...". Below the search bar, there are three patient cards. Each card displays the patient's name, age, and disease, along with input fields for "New name", "New age", and "New disease". Each card also has "Edit" and "Delete" buttons. The "Logout" button is located at the bottom left.

Fig 2.5: Working CRUD functionality

6. About:



Fig 2.6: about page

7.Contact page:

The screenshot shows the 'Get in Touch' contact page. The title is 'Get in Touch'. Below the title, a paragraph states: 'We'd love to hear your feedback or help with any queries.' The form consists of three input fields: 'Your Name' with a placeholder 'Enter your name', 'Your Email' with a placeholder 'name@example.com', and 'Your Message' with a placeholder 'Write your message here...'. A blue 'Send Message' button is located at the bottom left of the form.

Fig 2.7: contact page

Summary

The Patient Record Management System is a web-based application developed using Node.js, Express.js, EJS, and jQuery to manage patient data efficiently in clinical settings. It offers a secure and organized solution for handling medical records, appointment scheduling, and user access. The backend is structured with Express routing and middleware to handle authentication, CRUD operations, and form submissions, while the frontend uses EJS templates and jQuery for dynamic, responsive interfaces.

Users can log in to access a dashboard where they can view, add, update, or delete patient records. Each patient entry can store essential information like name, contact details, visit history, diagnoses, and prescriptions. The system also includes appointment management features, allowing staff to schedule and track consultations.

The application follows a RESTful design pattern, with clean separation between routes, views, and public assets. MongoDB (or another NoSQL database) is implied for storing records securely. This system is modular and scalable, making it suitable for small to mid-sized healthcare practices.

Overall, the system reduces paperwork, minimizes errors, and improves data accessibility, offering a foundation for future enhancements such as role-based access, medical file uploads, notifications, and analytics.

RESULT & DISCUSSION

RESULT

The Patient Record System was successfully implemented using a Node.js and Express backend, with EJS for frontend rendering. The following functionalities were achieved:

- Backend Integration:

The server.js file sets up an Express server that handles HTTP requests and routes for managing patient data.

The server connects to a database using a custom db.js module and leverages routing logic for fetching and rendering patient records.

- Data Modeling:

The models.js file defines the structure of patient data using Mongoose schemas (or a similar ORM/ODM). Each patient record contains fields such as name, age, contact information, and medical history. This ensures consistency in how data is stored and retrieved.

- Frontend Interface:

The patients.ejs template dynamically displays a list of patients, utilizing EJS templating to populate an HTML table with records fetched from the backend. It likely includes functionality to add, edit, or delete records using standard CRUD operations.

- Data Handling:

The presence of patient.json suggests mock or initial

seed data was used to populate the system, which is helpful for testing and demonstration.

DISCUSSION

This project successfully demonstrates how web technologies can be used to build a basic but functional healthcare data management system. Here are a few discussion points:

- Efficiency and Modularity:

The separation of concerns is well maintained. Server logic, data models, views, and database connection logic are modularized for maintainability and scalability.

- Usability:

The use of EJS templates allows for dynamic rendering of data, which improves user interaction. However, the UI could benefit from additional enhancements like real-time validation, filters, and responsive design.

- Data Integrity and Validation:

Although data models are in place, adding more robust input validation (both client-side and server-side) would increase system reliability.

- Future Improvements:

- Implementing user authentication and role-based access.
- Adding pagination for large datasets.

- Using a modern frontend framework (e.g., React) for a more dynamic user experience.
- Introducing API endpoints to support mobile or third-

REFERENCES

- **Bootstrap:** <https://getbootstrap.com/>
- **W3School:** <https://www.w3schools.com/>
- **jQuery:** <https://jquery.com/>
- **AngularJS:** <https://angularjs.org/>
- **MDN Web Docs:** <https://developer.mozilla.org/en-US/>
- Course Material provided by faculties (VTOP):
<https://vtop.vit.ac.in/vtop/>