EXERCISES ON JAVA FULL STACK

1. **Classes & Objects**
    i) Book class
    ii) Person class
    iii) Patient class
    iv) Dept and Emp class
    v) Customer, Orders, OrderItem

2. **Collections**
    i) ArrayList (To-DO List)
    ii) Vector (Student Management System)
    iii) Set (Unique Words)
    iv) Map (Unique Words)

3. **Inheritance**
    i) Single Level Inheritance (Vehicle-Car)
    ii) Multi-Level Inheritance (Vehicle – Car – Convertible)
    iii) Multiple Inheritance (Multiple Inheritance MediaResource, PrintResource)

4. **Exception Handling**
    i) Online Shopping Cart with Exception Handling
    ii) Exception Handling in Simple Banking System
    iii) Exception Handling in Student Grading System

5. **Threads**
    i) Implementing Thread using Thread class
    ii) Implementing Thread using Runnable interface
    iii) Implementing Thread using Anonymous class
    iv) Implementing Method Synchronization

6. **AWT/JFC&Swings**
    i) Example on all JFC basic components
    ii) Example on all JFC advanced components
    iii) Laying JFC components on JFrame
    iv) Laying JFC components on JPanel
    v) BorderLayout
        a. TopPanel
        b. Navigation Bar Panel
        c. Content Panel
        d. Footer Panel
    vi) FlowLayout
    vii) BorderLayout
    viii) GridLayout
    ix) VBoxLayout
    x) HBoxLayout
    xi) Custom Layout
    xii) Example on Custom Layout in project
    xiii) Example on All Event Handlers
        a. ActionListener

        b. AdjustmentListener
        c. ItemListener
        d. FocusListener
        e. KeyListener
        f. MouseListener
        g. MouseMotionListener
        h. WindowListener
    xiv) Displaying different Dialog Boxes

7. **JDBC**
    i) TypeIV JDBC Driver connectivity
        a. Oracle
        b. MySQL
        c. SQLite
        d. PostgreSQL
        e. HSQLDB
    ii) Perform INSERT operation using Statement object
    iii) Perform INSERT operation using PreparedStatement object
    iv) Perform UPDATE operation using Statement object
    v) Perform UPDATE operation using PreparedStatement object
    vi) Perform DELETE operation using Statement object
    vii) Perform SELECT operation using Statement & ResultSet objects
        a. PK based search
        b. No-conditional search
        c. conditional based search
        d. WHERE with single condition
        e. WHERE with multiple conditions
        f. using all relational operators
        g. JOINS based search
        h. GROUP BY
        i. GROUP BY with HAVING clause
        j. SUB-QUERY in WHERE clause
        k. SUB-QUERY in SELECT clause
    viii) Perform Atomicity Tx management in JDBC
    ix) Perform Consistency Tx management in JDBC

8. **Servlets**
    i) Servlets Introduction
    ii) Example on GenericServlet
    iii) Example on HttpServlet
    iv) Servlet Lifecycle
    v) Session management and client state persistence using HttpServlet
    vi) RequestDispatcher.include() / forward(), response.sendRedirect()
        a. Servlet-HTML communication
        b. Servlet-Servlet communication
        c. Servlet-JSP communication
    vii) ECommerce application in servlets
    viii) Example on Filters
    ix) Example on Servlets

9. **JSP**
    i) JSP Introduction

ii) JSP Lifecyle
iii) Example on Scripting elements
iv) Comment element
v) Declaration element
vi) Scriptlet
vii) Expression element
viii) Example on Directive elements
  a. <%@page%>
  b. <%@include%>
  c. <%@taglib%>
ix) Example on Standard Action elements
  a. <jsp:include>
  b. <jsp:forward>
  c. <jsp:useBean>
  d. <jsp:include>
  e. <jsp:forward>
x) Example on Custom elements
xi) Empty element
  a. Element with Attribute
  b. Element with Text
  c. Element with sub element/nested element
xii) Application on custom tag

10. **Hibernate**

ORM Introduction
First CRUD operations application in Hibernate
  Writing Java Bean
  Mapping Java Bean to hibernate mapping file
  Hibernate configuration file
  Connection Pooling
  save(), update(), delete(), get(), load(), list(), iterate()
  Criteria, Criterion, Projection
  Hibernate Query Language
  First level cache, Second level cache
  Memory level cache, Disk level cache
  Converting business component into WSDL using
Hierarchical Relationship
  Table-Per-Class
  Table-Per-SubClass
  Table-Per-Concrete Class
Association Relationship
  One-To-Many Relationship
  Many-To-One Relationship
  Many-To-Many Relationship
  One-To-One Relationship

11. **Spring**

Spring IoC Introduction
  Spring Eager & Lazy loading
  Singleton and Prototype objects
  Spring Bean Lifecycle
  Construction Injection

- Setter Injection
- Injecting Primitive Objects, Secondary objects (Arrays and Classes) and Collection objects
- Spring Bean Auto-wiring
- Annotation based configuration
- Spring Containers
  - Lazy loading container
    - BeanFactory
  - Eager loading container
    - ApplicationContext

Spring JDBC DAO Module
- Spring JdbcTemplate
- Spring Programmatic/Declarative TX management

Spring ORM DAO Module
- Spring LocalSessionFactory, HibernateTemplate

Spring Web Module
- DispatcherServlet
- MethodNameResolver
- ViewResolvers
- Sub class of Controller
- Sub class of MultiActionController
- ModelAndView object

12. **Spring Boot & MicroServices**

- Creating Spring Boot application using Spring Boot CLI, spring.io, STS
- Spring Boot Auto Configuration
- Spring Data Access
  - SQL Databases, NoSQL Databases
  - Spring JPA, CRUD applications
  - Spring Boot JPA
- Spring Boot JPA with Web
  - Building RESTful WebServices using Spring Boot
- Security in Spring Boot
- Messaging in Spring Boot
- Spring Boot Interceptors
- Spring Boot REST Template
- Thymeleaf
- Consuming RESTful service from AngularJS, Angular, ReactJS
- Spring Boot CORS Support
- Spring Boot I18N (InternationalizatioN)
- Spring Boot Eureka Server
- Service registration with Eureka
- ZUUL Proxy Server and Routing
- Spring Boot Actuator
- Spring Boot Swagger
- Creating Docker Image
- Tracing MicroServices Sleuth Logs using Zipkin Server
- Flyway Database
- Sending Email
- Spring Hystrix Circuit Breaker
- Spring Boot Unit Testing
- REST Controller Unit Testing

13. **HTML & CSS**

Text Formatting Elements

<b>, <strong>, <i>, <em>, <mark>, <small>, <del>, <ins>, <sub>, <sup>, <u>, <marquee>

HTML Menu Elements

<UL>, <OL>, <LI>

HTML Paragraph Elements

<P>, <HR>, <BR>, <PRE>

HTML Heading Elements

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

HTML Image Elements

<IMG>

HTML Anchor Elements

<A>, <MAP>, <AREA>

HTML Table Elements

<table>, <caption>, <thead>, <tbody>, <tfoot>, <tr>, <th>, <td>, <colgroup>, <col>

HTML Layout Elements

<header>, <nav>, <section>, <article>, <aside>, <footer>, <details>, <summary>

HTML Container Elements

<div>, <span>, <p>

HTML Form Elements

- <INPUT type="text|radio|checkbox|file|hidden|password|button|submit|reset|color|email|month|number|range|search|tel|date|datetime-local|time|url|week">
- <input> attributes – checked, disabled, max, maxlength, min, minlength, pattern, readonly, required, size, step, value
- <textarea>, <select>, <option>

14. **369 CSS Properties**

CSS Selectors

Simple Selectors - id, class, tag

Combinator selectors – descendant selectors (space), child selector (>), adjacent sibling selectors (+), general sibling selectors (~)

Pseudo class selectors – mouse over it, visited/unvisited links, got/lost focus

Pseudo elements selectors – style the first letter, or line, of an element, insert content before or after the content of an element

Attribute selectors – attr=value, attr~=value, attr|=value, attr^=value, attr$=value, attr*=value

CSS Property Groups

Color, Backgrounds & Borders, Basic Box, Flexible Box, Text, Text Decoration, Fonts, Reading Mode, Tables, Lists and Counters, Animation, 2D/3D Transformations, Transition, Basic User Interface, Multi-column, Paged Media, Generated Content, Filter Effects, Image/Replaced Content, Masking, Speech, Marquee

15. **HTML, CSS & Bootstrap**
    - Bootstrap 12 grid layout
    - Bootstrap forms
    - Bootstrap Typography
    - Bootstrap Images
    - Bootstrap Tables
    - Bootstrap Figures
    - accordion, alerts, badges, borders, breadcrumbs, button-groups, button-modifiers, buttons, cards, carousel, collapse, colors, custom forms, display, dropdowns, figure, flex-box

16. **JavaScript**

JavaScript Datatypes, Variables, Type conversion, Operators, Conditions, Loops, Writing and calling functions
JavaScript objects
JavaScript Events
Form Field Validations
AJAX
XML DOM
JSON Parsing
HTML DOM

17. **jQuery**

jQuery Fundamentals
Selecting elements for manipulation
Manipulating element properties and attributes
Changing element styling
Setting element content
Showing and Hiding elements
Animating the display state of elements
Talking to the server with AJAX

Introduction to JAVA Full Stack

Java Full Stack development refers to the practice of utilizing the Java programming language for both front-end and back-end development in web application development. A Full Stack developer is proficient in both client-side and server-side technologies, allowing them to work on all aspects of a web application, from the user interface to the server and database.

Front-end Technologies

HTML/CSS: HTML (Hypertext Markup Language) is used for structuring web content, while CSS (Cascading Style Sheets) is used for styling and layout.

JavaScript: A crucial scripting language for creating dynamic and interactive web pages. It enables the development of client-side logic and interactivity.

Front-end Frameworks/Libraries

Angular, React, or Vue.js: These are popular front-end frameworks/libraries that help in building scalable and maintainable user interfaces. Full Stack developers often choose one of these frameworks to enhance the efficiency of front-end development.

Back-end Technologies

Java: A versatile and widely used programming language for building robust and scalable server-side applications. Java provides a rich set of libraries and frameworks that simplify the development process.

Spring Framework: A comprehensive framework for Java development that addresses various aspects of building enterprise-level applications. Spring Boot, a part of the Spring ecosystem, is especially popular for building microservices and web applications.

Database

Relational Database Management System (RDBMS): Common choices include MySQL, PostgreSQL, or Oracle. Java Full Stack developers should be familiar with database design, SQL queries, and database management.

Server-Side Scripting

Node.js: While Java is the primary choice for server-side development in Full Stack Java, some developers may use Node.js for specific applications, enabling JavaScript to be used on the server side as well.

Version Control/Git

Git: Essential for tracking changes in code, collaborating with other developers, and managing different versions of the codebase.

APIs (Application Programming Interfaces)

RESTful APIs: Full Stack developers often work with RESTful APIs to facilitate communication between the front-end and back-end components of an application.

Build Tools and Package Managers

Maven or Gradle: These tools automate the build process, manage dependencies, and streamline project configuration.

Containerization and Deployment

Docker: Helps in creating and deploying applications in lightweight, portable containers.

Container Orchestration (e.g., Kubernetes): Manages the deployment, scaling, and operation of containerized applications.

Testing

JUnit and TestNG: Common testing frameworks for Java applications.

Front-end testing frameworks (e.g., Jest for React).

Development Tools

Integrated Development Environments (IDEs): Eclipse, IntelliJ IDEA, or Visual Studio Code are popular choices for Java Full Stack development.

1. E-Commerce Platform

1.1 Background

You are hired to develop a full-stack e-commerce platform for a client. The platform should allow users to browse products, add items to their cart, and complete the purchase. Additionally, there should be an admin panel for managing product listings and tracking orders.

1.2 Requirements

Frontend

- ✓ Implement a user-friendly interface using a modern frontend framework (e.g., React, Angular, or Vue.js).
- ✓ Create pages for product listings, product details, shopping cart, and order confirmation.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Develop a RESTful API using a Java framework such as Spring Boot.
- ✓ Implement CRUD operations for managing products and handling user authentication.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store product information, user data, and order details.
- ✓ Implement a secure checkout process with payment gateway integration.

Database

- ✓ Design a database schema to store product information, user profiles, and order details.
- ✓ Optimize the database for efficient queries, considering the potential growth of the platform.

Integration

- ✓ Connect the frontend and backend to enable seamless data flow.
- ✓ Implement features like adding products to the cart, updating inventory, and processing orders.
- ✓ Integrate third-party APIs for payment processing.

Testing

- ✓ Develop unit tests for backend components to ensure code reliability.
- ✓ Implement end-to-end tests to simulate user interactions and catch integration issues.
- ✓ Test the application's responsiveness and cross-browser compatibility.

Deployment

- ✓ Prepare the application for deployment in a production environment.
- ✓ Choose a suitable hosting solution and deploy the application.

1.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

1.4 Evaluation Criteria

- ✓ Code quality and organization.
- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Integration with third-party services.
- ✓ Testing strategy and coverage.

2. Dashboard for Social Media Analytics

2.1 Background

You are tasked with building a social media analytics dashboard that aggregates and displays data from various social media platforms. The goal is to provide users with insights into their social media performance.

2.2 Requirements

Frontend

- ✓ Develop a dashboard using a frontend framework like React or Angular.
- ✓ Implement visualizations for metrics such as engagement, reach, and follower growth.
- ✓ Allow users to connect their social media accounts securely

Backend

- ✓ Create a backend API using Spring Boot to fetch and aggregate data from social media APIs (e.g., Twitter, Facebook).
- ✓ Implement user authentication and authorization.
- ✓ Store aggregated data in a database for historical analysis.

Database

- ✓ Design a database schema to store aggregated social media data.
- ✓ Optimize the database for efficient querying and reporting.

Integration

- ✓ Connect the backend to social media APIs to fetch real-time data.
- ✓ Implement scheduled tasks to periodically update aggregated metrics.
- ✓ Handle authentication and token management for third-party APIs.

Testing

- ✓ Develop unit tests for backend components.
- ✓ Test the application's ability to fetch and process data from various social media platforms.
- ✓ Ensure the dashboard's responsiveness and usability.

Deployment

- ✓ Deploy the application in a production environment.
- ✓ Set up monitoring for data fetching tasks and potential issues.

2.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report detailing challenges faced and solutions implemented.

2.4 Evaluation Criteria

- ✓ Code quality and organization.

- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data protection).
- ✓ Database design and performance considerations.
- ✓ Effectiveness of data fetching and aggregation.
- ✓ User interface design and visualization of metrics.
- ✓ Testing strategy and coverage.

3. Platform for Online Learning

3.1 Background

You are tasked with developing an online learning platform that allows users to browse courses, enroll in classes, and track their progress. The platform should also include an administrative interface for managing courses and user data.

3.2 Requirements

Frontend

- ✓ Implement a responsive and intuitive user interface using a modern frontend framework (e.g., React or Angular).
- ✓ Create pages for course listings, course details, user enrollment, and a user dashboard.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Develop a RESTful API using a Java framework such as Spring Boot.
- ✓ Implement CRUD operations for managing courses and user data.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store course information, user profiles, and progress tracking.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ Design a database schema to store course information, user profiles, and progress tracking data.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Integration

- ✓ Connect the frontend and backend to enable seamless data flow.
- ✓ Implement features like course enrollment, progress tracking, and user notifications.
- ✓ Integrate third-party APIs for features like payment processing or content delivery.

Testing

- ✓ Develop unit tests for backend components to ensure code reliability.
- ✓ Implement end-to-end tests to simulate user interactions and catch integration issues.
- ✓ Test the application's responsiveness and cross-browser compatibility.

Deployment

- ✓ Prepare the application for deployment in a production environment.
- ✓ Choose a suitable hosting solution and deploy the application.

3.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

3.4 Evaluation Criteria

- ✓ Code quality and organization.
- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Integration with third-party services.
- ✓ Testing strategy and coverage.

4. Management Structure for Employees

4.1 Background

You are hired to develop an Employee Management System for a company. The system should include features for managing employee information, leave requests, and performance evaluations. The company also requests an analytics dashboard to monitor employee performance and attendance.

4.2 Requirements

Frontend

- ✓ Implement a user-friendly interface using a frontend framework (e.g., React or Angular).
- ✓ Create pages for employee listings, detailed employee profiles, leave requests, and performance evaluations.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Develop a RESTful API using a Java framework such as Spring Boot.
- ✓ Implement CRUD operations for managing employee data, leave requests, and performance evaluations.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store employee information, leave records, and performance data.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ Design a database schema to store employee information, leave records, and performance data.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Integration

- ✓ Connect the frontend and backend to enable seamless data flow.
- ✓ Implement features like leave request submissions, performance evaluations, and notifications.
- ✓ Create an analytics dashboard to display employee performance metrics.

Testing

- ✓ Develop unit tests for backend components to ensure code reliability.
- ✓ Implement end-to-end tests to simulate user interactions and catch integration issues.
- ✓ Test the application's responsiveness and cross-browser compatibility.

Deployment

- ✓ Prepare the application for deployment in a production environment.
- ✓ Choose a suitable hosting solution and deploy the application.

4.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

4.4 Evaluation Criteria

- ✓ Code quality and organization.
- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Analytics dashboard functionality.
- ✓ Testing strategy and coverage.

5. Workflow Administration Platform

5.1 Background

You are assigned to develop a task management system for a team of software developers. The system should allow team members to create, assign, and track tasks. Additionally, there should be a reporting feature that shows task completion statistics.

5.2 Requirements

Frontend

- ✓ Develop a responsive and user-friendly frontend using a modern framework (e.g., React, Angular, or Vue.js).
- ✓ Create pages for task creation, task assignment, task tracking, and a reporting dashboard.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Implement a RESTful API using a Java framework such as Spring Boot.
- ✓ Develop CRUD operations for managing tasks and user data.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store task information and user profiles.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ Design a database schema to store task information, user profiles, and task assignment data.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Integration

- ✓ Connect the frontend and backend to enable seamless data flow.
- ✓ Implement features like task assignment, task status updates, and notifications.
- ✓ Create a reporting dashboard that displays task completion statistics.

Testing

- ✓ Develop unit tests for backend components to ensure code reliability.
- ✓ Implement end-to-end tests to simulate user interactions and catch integration issues.
- ✓ Test the application's responsiveness and cross-browser compatibility.

Deployment

- ✓ Prepare the application for deployment in a production environment.
- ✓ Choose a suitable hosting solution and deploy the application.

5.3 Deliverables

- ✓ Source code for both frontend and backend components.

- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

5.4 Evaluation Criteria

- ✓ Code quality and organization.
- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Reporting dashboard functionality.
- ✓ Testing strategy and coverage.

6. Internet Retailer

6.1 Background

You are hired to build an online marketplace where users can buy and sell products. The platform should include features for product listings, shopping cart management, and order processing.

6.2 Requirements

Frontend

- ✓ Develop a visually appealing and responsive frontend using a modern framework (e.g., React, Angular, or Vue.js).
- ✓ Create pages for product listings, product details, shopping cart, and order checkout.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Implement a RESTful API using a Java framework such as Spring Boot.
- ✓ Develop CRUD operations for managing products, user profiles, and order data.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store product information, user profiles, and order details.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ Design a database schema to store product information, user profiles, and order details.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Integration

- ✓ Connect the frontend and backend to enable seamless data flow.
- ✓ Implement features like product search, shopping cart management, and order processing.
- ✓ Integrate third-party APIs for features like payment processing.

Testing

- ✓ Develop unit tests for backend components to ensure code reliability.
- ✓ Implement end-to-end tests to simulate user interactions and catch integration issues.
- ✓ Test the application's responsiveness and cross-browser compatibility.

Deployment

- ✓ Prepare the application for deployment in a production environment.
- ✓ Choose a suitable hosting solution and deploy the application.

6.3 Deliverables

- ✓ Source code for both frontend and backend components.

- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

6.4 Evaluation Criteria

- ✓ Code quality and organization.
- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Integration with third-party services.
- ✓ Testing strategy and coverage.

7. Health Monitoring Application

7.1 Background

A healthcare startup wants to develop a health monitoring application that collects and analyzes user data to provide personalized health insights.

7.2 Requirements

Frontend

- ✓ React for a modern and interactive user interface
- ✓ Develop a visually appealing and responsive frontend using a modern framework (e.g., React, Angular, or Vue.js).
- ✓ Create pages for product listings, product details, shopping cart, and order checkout.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Spring MVC for handling HTTP requests and managing user data
- ✓ Implement a RESTful API using a Java framework such as Spring Boot.
- ✓ Develop CRUD operations for managing products, user profiles, and order data.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store product information, user profiles, and order details.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ MongoDB for storing user profiles, health metrics, and historical data.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Authentication

- ✓ JSON Web Tokens (JWT) for secure user authentication.

Real-time Updates

- ✓ WebSocket for real-time communication with wearables and sensors.

Challenges

- ✓ Ensuring data privacy and compliance with healthcare regulations.
- ✓ Implementing algorithms for health data analysis and insights.

7.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

7.4 Evaluation Criteria

- ✓ Code quality and organization.

- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Integration with third-party services.
- ✓ Testing strategy and coverage.

8. Project Management Tool

8.1 Background

A multinational corporation needs a customized project management tool to streamline communication and collaboration among its teams.

8.2 Requirements

Frontend

- ✓ Vue.js for a reactive and intuitive user interface.
- ✓ Develop a visually appealing and responsive frontend using a modern framework (e.g., React, Angular, or Vue.js).
- ✓ Create pages for product listings, product details, shopping cart, and order checkout.
- ✓ Implement user authentication and authorization.

Backend

- ✓ Spring Boot for handling project data, user roles, and notifications.
- ✓ Implement a RESTful API using a Java framework such as Spring Boot.
- ✓ Develop CRUD operations for managing products, user profiles, and order data.
- ✓ Integrate with a database (e.g., MySQL or PostgreSQL) to store product information, user profiles, and order details.
- ✓ Implement secure user authentication and authorization.

Database

- ✓ PostgreSQL for storing project-related information.
- ✓ Optimize the database for efficient queries, considering potential scalability.

Authentication

- ✓ OAuth 2.0 for secure user authentication and authorization.

Integration

- ✓ RESTful APIs for integrating with other corporate systems.

Challenges

- ✓ Implementing role-based access control to ensure data security.
- ✓ Optimizing for performance and responsiveness in a large-scale enterprise environment.

8.3 Deliverables

- ✓ Source code for both frontend and backend components.
- ✓ Documentation on how to set up and run the application.
- ✓ A brief report outlining challenges faced and solutions implemented.

8.4 Evaluation Criteria

- ✓ Code quality and organization.

- ✓ Proper use of technologies and frameworks.
- ✓ Security measures implemented (e.g., authentication, data validation).
- ✓ Database design and performance considerations.
- ✓ User interface design and responsiveness.
- ✓ Integration with third-party services.
- ✓ Testing strategy and coverage.

9. JAVA Full Stack on Doctor-Patient-Portal

Creating a Doctor-Patient Portal involves a comprehensive full-stack application that handles user authentication, data storage, and interaction between doctors and patients.

Online Doctor Appointment System Java Project: A project that includes logins for both doctors and patients.

Hospital Management System Project In Java: A project that includes functionalities like admin login, patient details, doctor details, and room details.

Doctor-patient management software can help healthcare providers handle appointments, patient records, billing, patient portals, and administrative tasks.

Frontend

- ✓ Technology Stack: HTML, CSS, JavaScript, React (or Angular, Vue.js)
- ✓ Project Description: Develop a user-friendly interface for both doctors and patients. Include features for appointment scheduling, viewing medical records, and communication between doctors and patients.

Backend

- ✓ Technology Stack: Java, Spring Boot, Spring MVC, Spring Data JPA
- ✓ Project Description: Build a RESTful API to handle various functionalities such as user registration, appointment scheduling, and accessing medical records. Implement role-based access control for doctors and patients.

Database

- ✓ Technology Stack: MySQL or PostgreSQL
- ✓ Project Description: Design a database schema to store user data, appointments, and medical records. Create tables for users, roles, appointments, and other relevant entities.

ORM (Object-Relational Mapping)

- ✓ Technology Stack: Hibernate (or any other JPA provider)
- ✓ Project Description: Use Hibernate for mapping Java objects to database tables. Implement CRUD operations for user data, appointments, and other entities.

Authentication and Authorization

- ✓ Technology Stack: Spring Security
- ✓ Project Description: Implement user authentication with Spring Security. Set up role-based access control to ensure that only authorized users (doctors or patients) can access specific features.

Frontend-Backend Integration

- ✓ Technology Stack: RESTful API (JSON for data exchange)
- ✓ Project Description: Establish communication between the frontend and backend using HTTP requests. Implement asynchronous updates for real-time data retrieval and display.

Testing

- ✓ Technology Stack: JUnit, TestNG (for unit testing), Postman (for API testing)
- ✓ Project Description: Write unit tests for backend services and controllers. Test API endpoints using tools like Postman to ensure they respond correctly.

Build Tool

- ✓ Technology Stack: Maven or Gradle
- ✓ Project Description: Use Maven or Gradle for building, testing, and managing dependencies.

Version Control

- ✓ Technology Stack: Git, GitHub (or GitLab, Bitbucket)
- ✓ Project Description: Set up a Git repository for version control. Commit regularly and use branches for feature development.

Deployment

- ✓ Technology Stack: Docker, Heroku, AWS, or any cloud platform
- ✓ Project Description: Containerize the application using Docker. Deploy the application to a cloud platform or a server.

Continuous Integration/Continuous Deployment (CI/CD)

- ✓ Technology Stack: Jenkins, GitLab CI, Travis CI, or any CI/CD tool
- ✓ Project Description: Set up a CI/CD pipeline to automate testing and deployment processes.

Documentation

- ✓ Technology Stack: Swagger or similar tools for API documentation
- ✓ Project Description: Generate documentation for the API to help other developers understand how to interact with the backend.

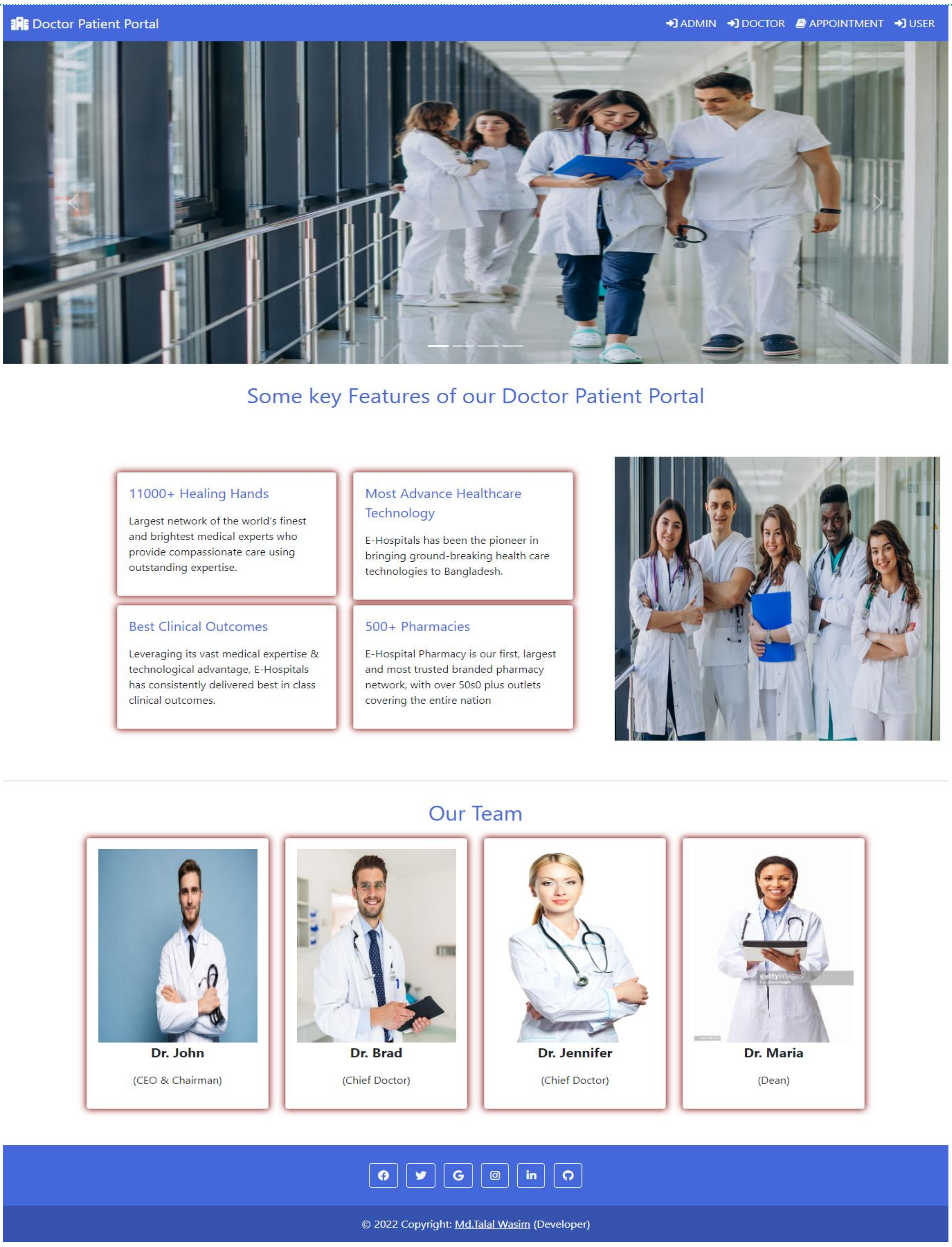
Real-Time Communication (Optional)

- ✓ Technology Stack: WebSocket (for real-time updates)
- ✓ Project Description: Implement real-time communication for features like instant messaging between doctors and patients.

9.1 Technology used

Advance JAVA concepts like JSP, JSTL, Servlet, HTML, CSS, Boostrap 5, Fontawesome and MySQL.

9.2 Project View - Home Page



Doctor Patient Portal

ADMIN DOCTOR APPOINTMENT USER

Some key Features of our Doctor Patient Portal

11000+ Healing Hands

Largest network of the world's finest and brightest medical experts who provide compassionate care using outstanding expertise.

Most Advance Healthcare Technology

E-Hospitals has been the pioneer in bringing ground-breaking health care technologies to Bangladesh.

Best Clinical Outcomes

Leveraging its vast medical expertise & technological advantage, E-Hospitals has consistently delivered best in class clinical outcomes.

500+ Pharmacies

E-Hospital Pharmacy is our first, largest and most trusted branded pharmacy network, with over 50s0 plus outlets covering the entire nation

Our Team

Dr. John

(CEO & Chairman)

Dr. Brad

(Chief Doctor)

Dr. Jennifer

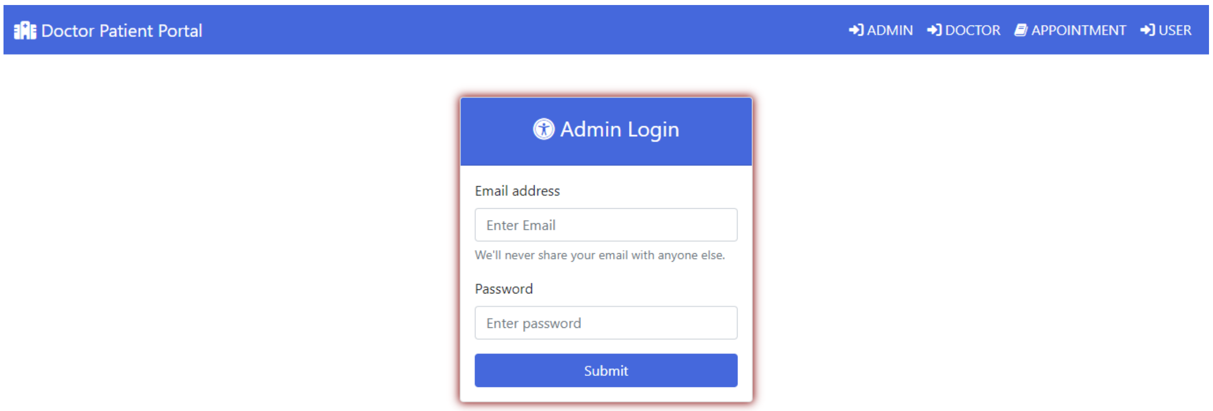
(Chief Doctor)

Dr. Maria

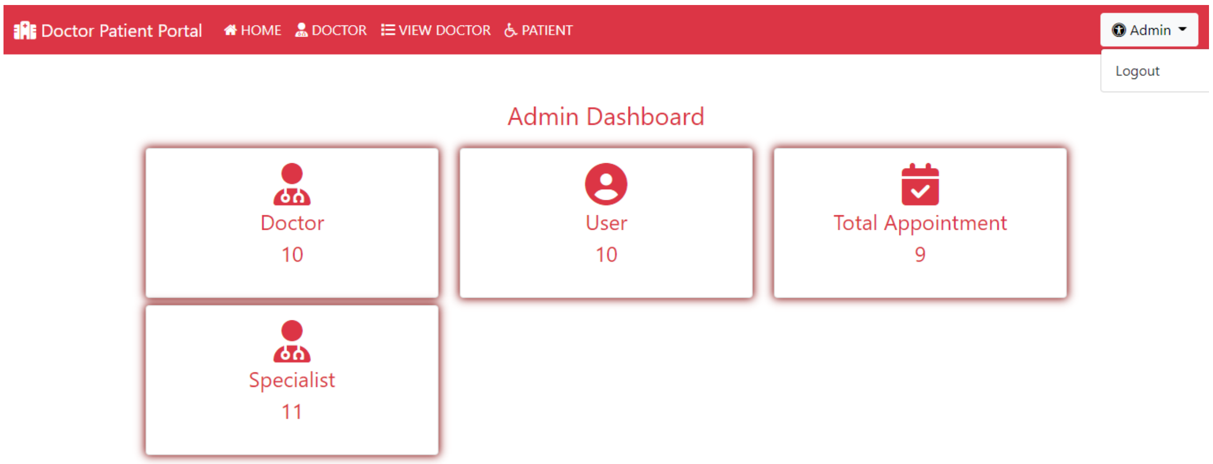
(Dean)

© 2022 Copyright: Md.Talal Wasim (Developer)

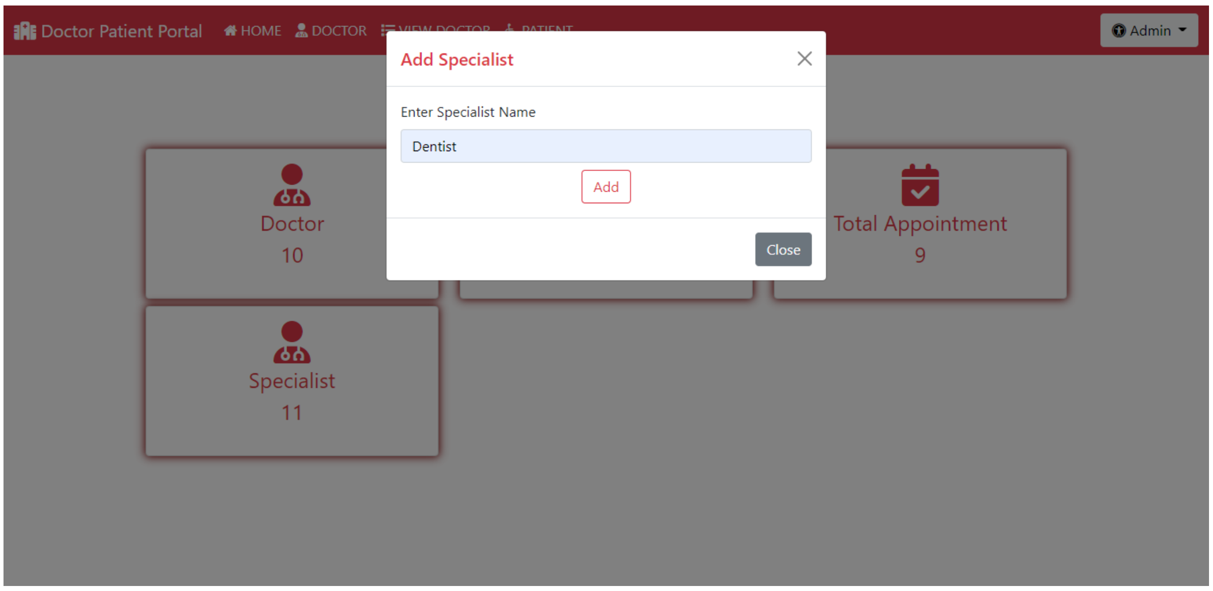
9.3 Admin Login



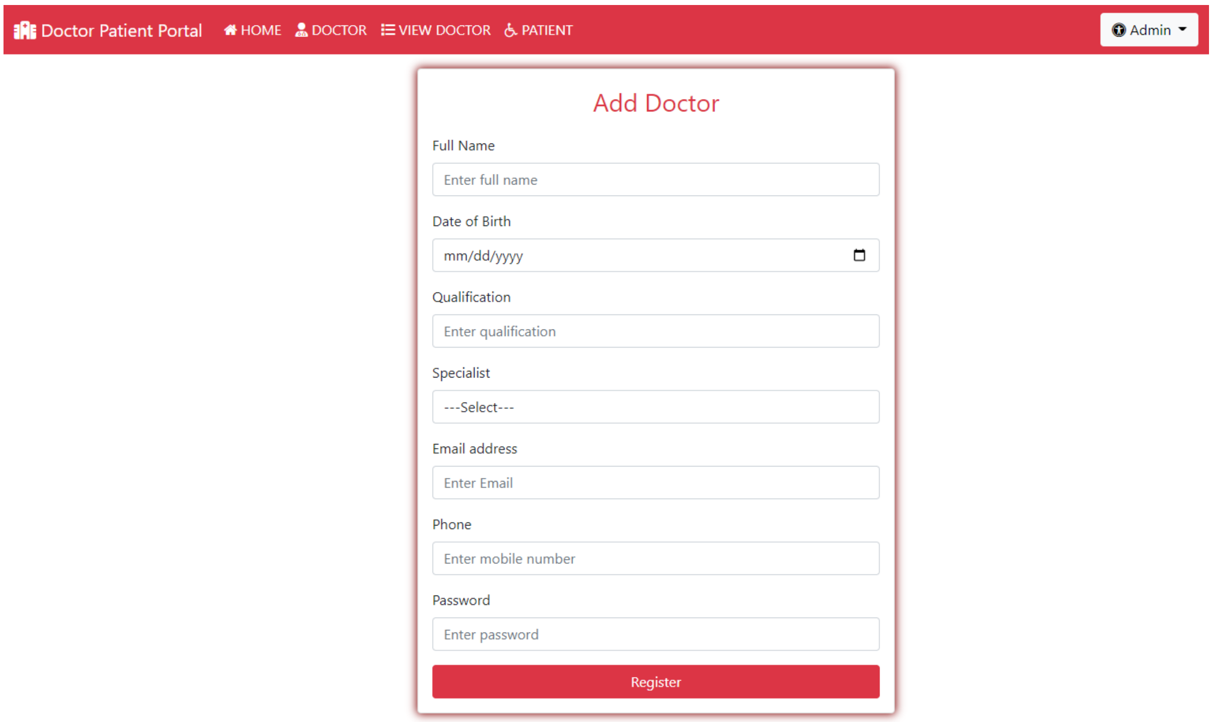
9.4 Admin Dashboard



9.5 Add Specialist



9.6 Add Doctor



Add Doctor

Full Name

Enter full name

Date of Birth

mm/dd/yyyy

Qualification

Enter qualification

Specialist

---Select---

Email address

Enter Email

Phone

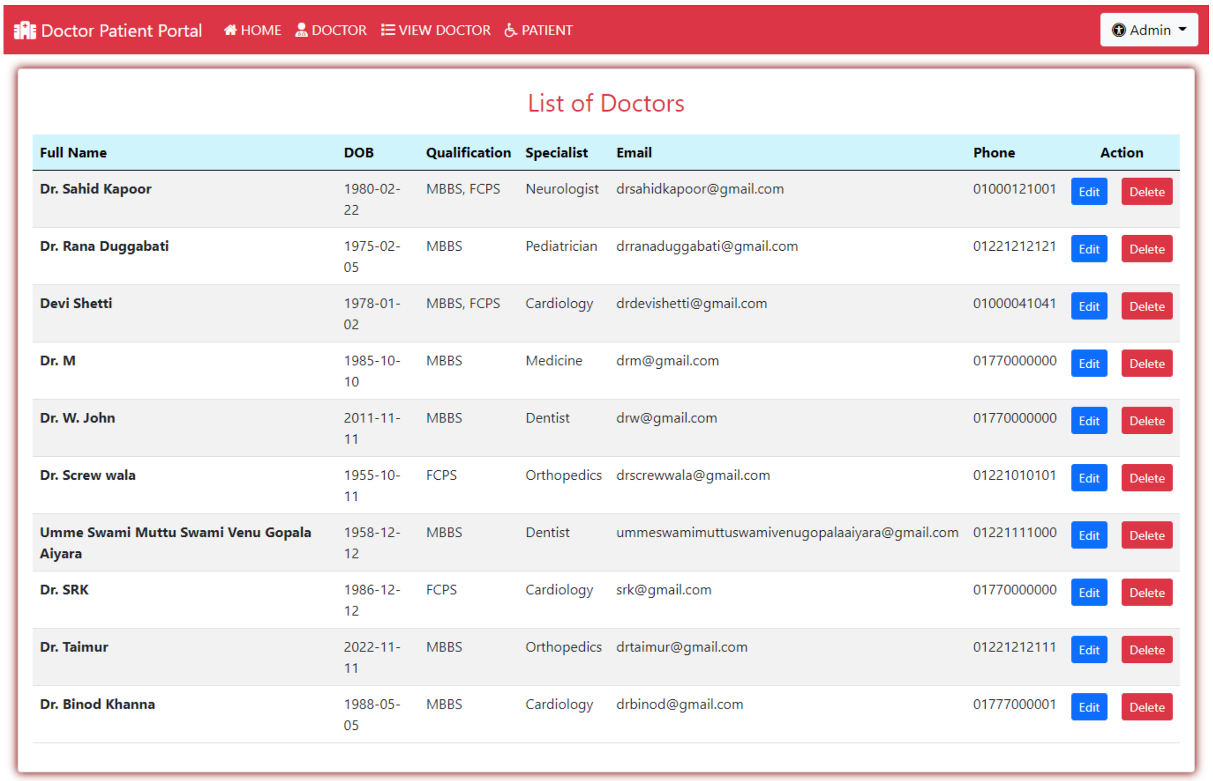
Enter mobile number

Password

Enter password

Register

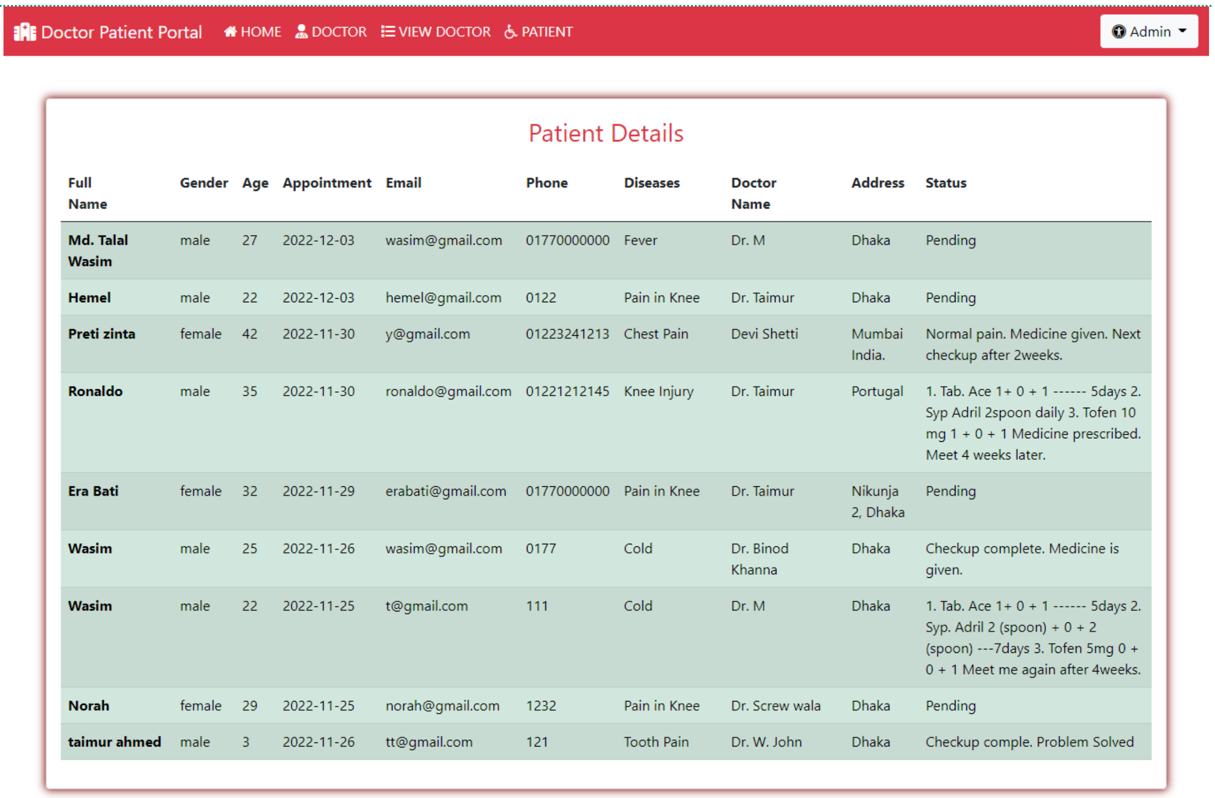
9.7 Doctor List



List of Doctors

Full Name	DOB	Qualification	Specialist	Email	Phone	Action
Dr. Sahid Kapoor	1980-02-22	MBBS, FCPS	Neurologist	drsahidkapoor@gmail.com	01000121001	Edit Delete
Dr. Rana Duggabati	1975-02-05	MBBS	Pediatrician	drranaduggabati@gmail.com	01221212121	Edit Delete
Devi Shetti	1978-01-02	MBBS, FCPS	Cardiology	drdevishetti@gmail.com	01000041041	Edit Delete
Dr. M	1985-10-10	MBBS	Medicine	drm@gmail.com	01770000000	Edit Delete
Dr. W. John	2011-11-11	MBBS	Dentist	drw@gmail.com	01770000000	Edit Delete
Dr. Screw wala	1955-10-11	FCPS	Orthopedics	drscrewwala@gmail.com	01221010101	Edit Delete
Umme Swami Muttu Swami Venu Gopala Aiyara	1958-12-12	MBBS	Dentist	ummeswamimuttuswamivenugopalaaiyara@gmail.com	01221111000	Edit Delete
Dr. SRK	1986-12-12	FCPS	Cardiology	srk@gmail.com	01770000000	Edit Delete
Dr. Taimur	2022-11-11	MBBS	Orthopedics	drtaimur@gmail.com	01221212111	Edit Delete
Dr. Binod Khanna	1988-05-05	MBBS	Cardiology	drbinod@gmail.com	01777000001	Edit Delete

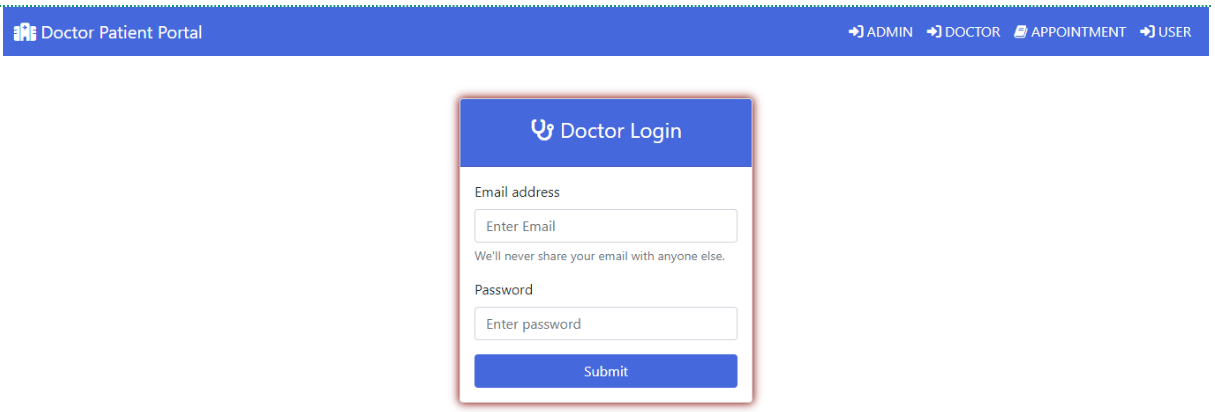
9.8 No. of Patients Appointment



Patient Details

Full Name	Gender	Age	Appointment	Email	Phone	Diseases	Doctor Name	Address	Status
Md. Talal Wasim	male	27	2022-12-03	wasim@gmail.com	01770000000	Fever	Dr. M	Dhaka	Pending
Hemel	male	22	2022-12-03	hemel@gmail.com	0122	Pain in Knee	Dr. Taimur	Dhaka	Pending
Preti zinta	female	42	2022-11-30	y@gmail.com	01223241213	Chest Pain	Devi Shetti	Mumbai India.	Normal pain. Medicine given. Next checkup after 2weeks.
Ronaldo	male	35	2022-11-30	ronaldo@gmail.com	01221212145	Knee Injury	Dr. Taimur	Portugal	1. Tab. Ace 1+ 0 + 1 ------ 5days 2. Syp Adril 2spoon daily 3. Tofen 10 mg 1 + 0 + 1 Medicine prescribed. Meet 4 weeks later.
Era Bati	female	32	2022-11-29	erabati@gmail.com	01770000000	Pain in Knee	Dr. Taimur	Nikunja 2, Dhaka	Pending
Wasim	male	25	2022-11-26	wasim@gmail.com	0177	Cold	Dr. Binod Khanna	Dhaka	Checkup complete. Medicine is given.
Wasim	male	22	2022-11-25	t@gmail.com	111	Cold	Dr. M	Dhaka	1. Tab. Ace 1+ 0 + 1 ------ 5days 2. Syp. Adril 2 (spoon) + 0 + 2 (spoon) ---7days 3. Tofen 5mg 0 + 0 + 1 Meet me again after 4weeks.
Norah	female	29	2022-11-25	norah@gmail.com	1232	Pain in Knee	Dr. Screw wala	Dhaka	Pending
taimur ahmed	male	3	2022-11-26	tt@gmail.com	121	Tooth Pain	Dr. W. John	Dhaka	Checkup comple. Problem Solved

9.9 Doctor Login



Doctor Login

Email address

Enter Email

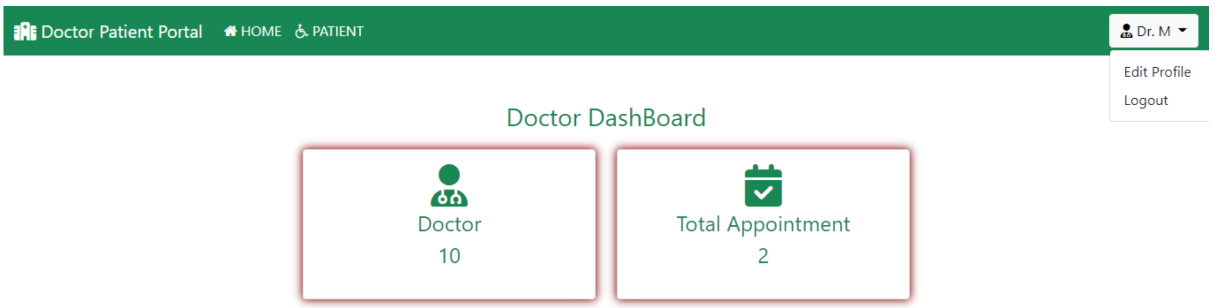
We'll never share your email with anyone else.

Password

Enter password

Submit

9.10 Doctor Dashboard

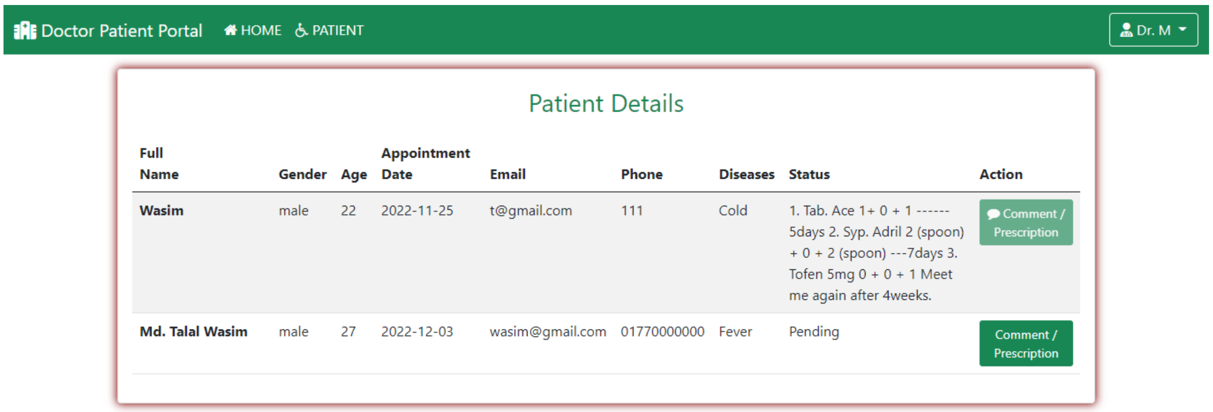


Doctor DashBoard

Doctor
10

Total Appointment
2

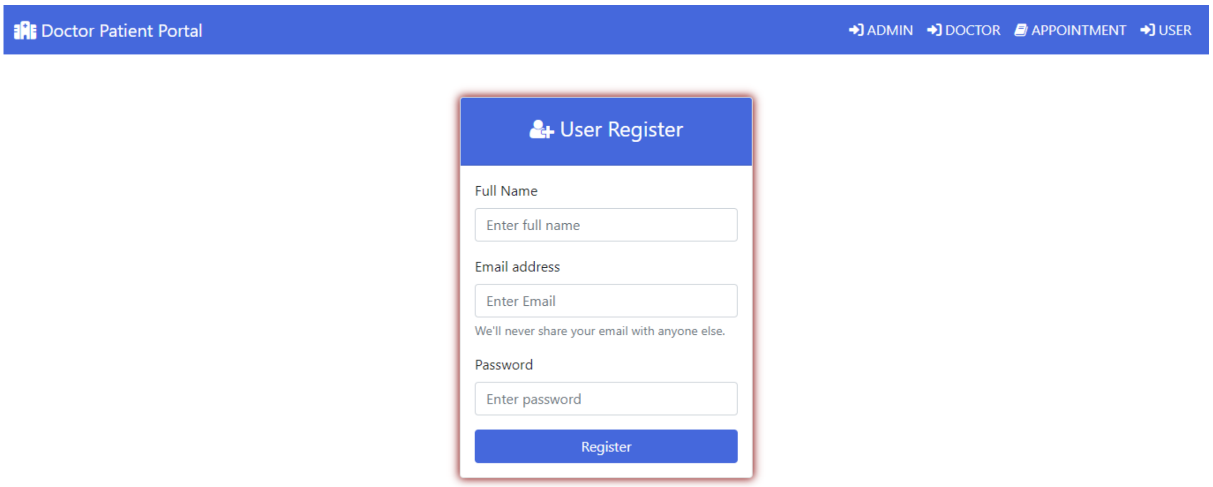
9.11 View list of Patient Appointment



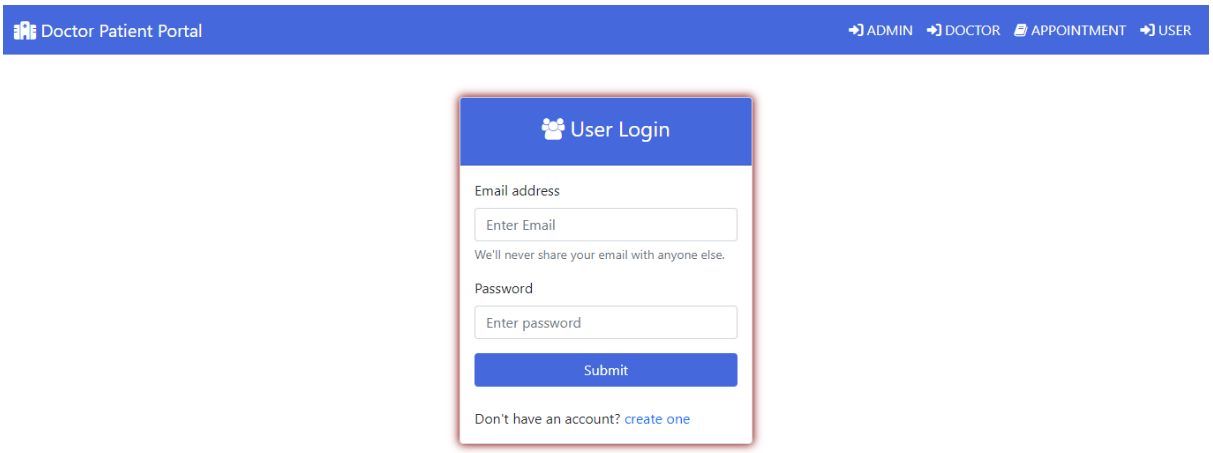
Patient Details

Full Name	Gender	Age	Appointment Date	Email	Phone	Diseases	Status	Action
Wasim	male	22	2022-11-25	t@gmail.com	111	Cold	1. Tab. Ace 1+ 0 + 1 ------ 5days 2. Syp. Adril 2 (spoon) + 0 + 2 (spoon) ---7days 3. Tofen 5mg 0 + 0 + 1 Meet me again after 4weeks.	Comment / Prescription
Md. Talal Wasim	male	27	2022-12-03	wasim@gmail.com	01770000000	Fever	Pending	Comment / Prescription

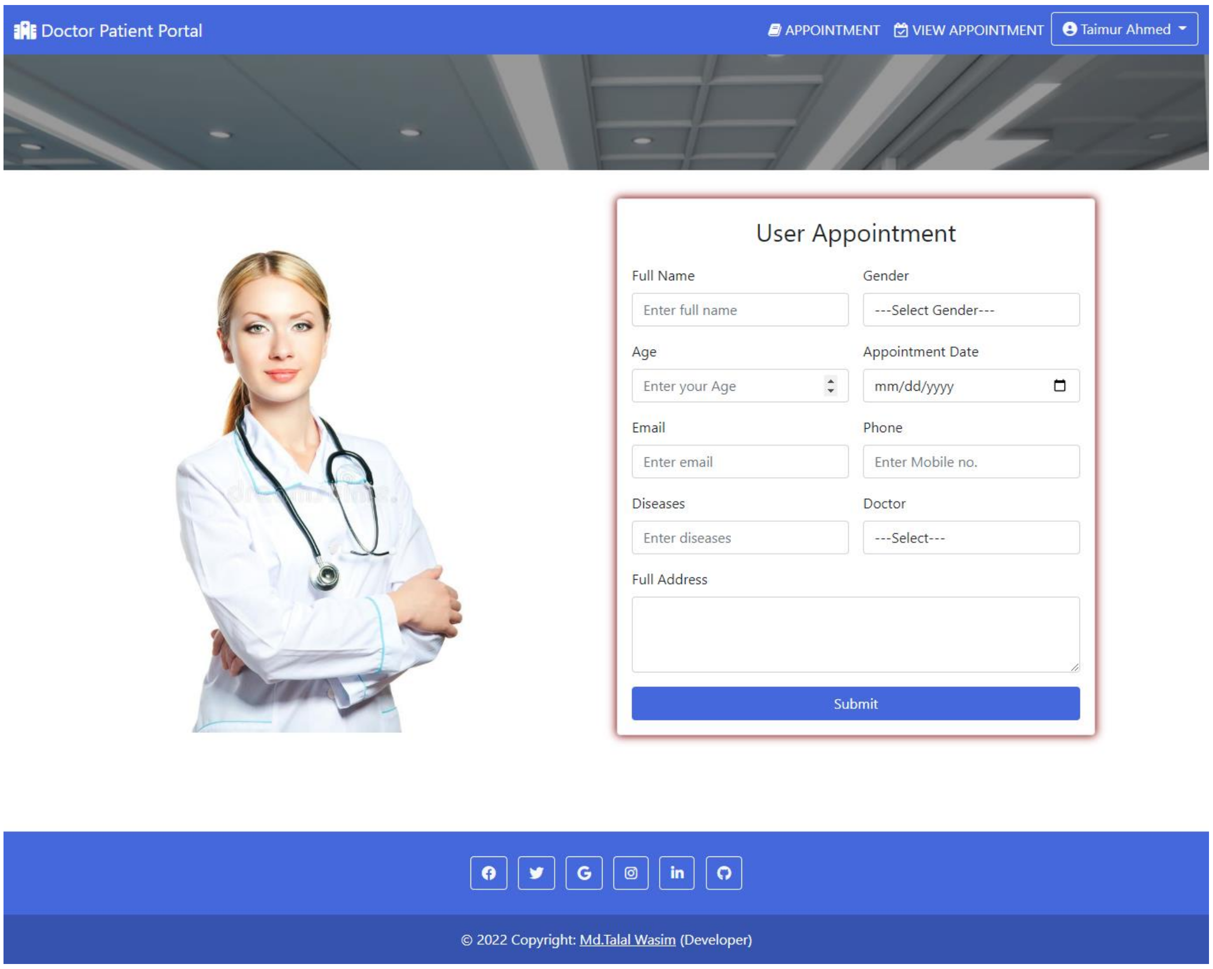
9.12 User Register first for Appointment Request



9.13 User Login



9.14 Make Appointment Request



User Appointment

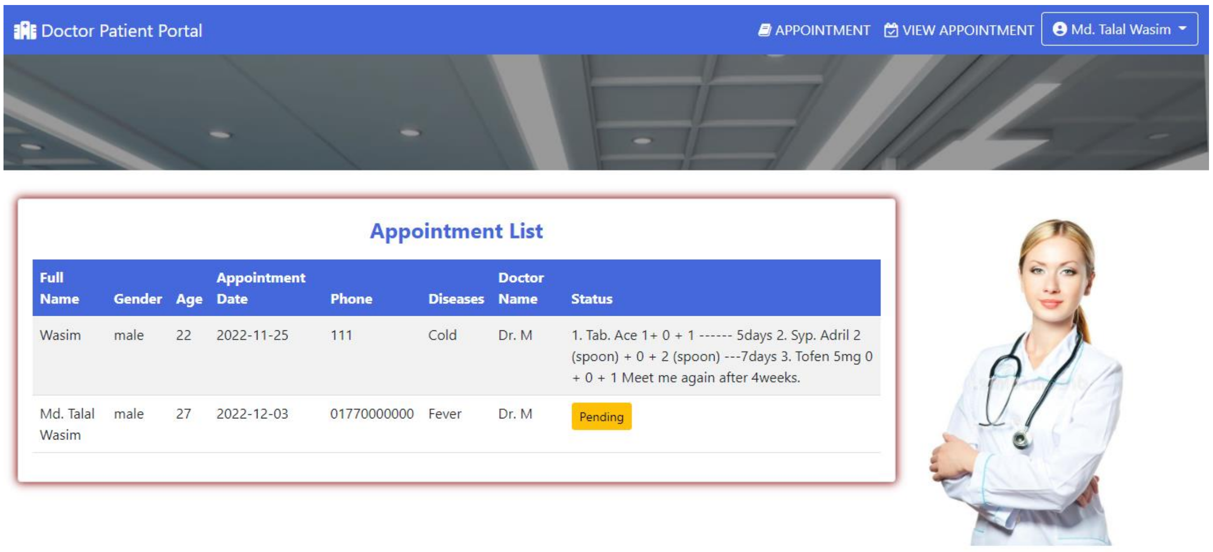
Full Name	Gender
Enter full name	---Select Gender---
Age	Appointment Date
Enter your Age	mm/dd/yyyy
Email	Phone
Enter email	Enter Mobile no.
Diseases	Doctor
Enter diseases	---Select---

Full Address

Submit

© 2022 Copyright: Md.Talal Wasim (Developer)

9.15 View list of appointment



Appointment List

Full Name	Gender	Age	Appointment Date	Phone	Diseases	Doctor Name	Status
Wasim	male	22	2022-11-25	111	Cold	Dr. M	1. Tab. Ace 1+ 0 + 1 ------ 5days 2. Syp. Adril 2 (spoon) + 0 + 2 (spoon) ---7days 3. Tofen 5mg 0 + 0 + 1 Meet me again after 4weeks.
Md. Talal Wasim	male	27	2022-12-03	01770000000	Fever	Dr. M	Pending

10. Online Book Store

A user-friendly Online Bookstore project in which users can log in or register, view the available books, select books along with their quantity, and buy them. Users can also get payment receipts after successful payment. The project can also be used by the administrator, who can add new books, remove books, increase and decrease the quantity of books, change the price of the books as well as maintain the selling history of books.

10.1 Website is built for following purpose

- For Selling books online.
- Maintaining books selling history.
- Adding and managing books.
- User Friendly.
- For Implementation of Http Servlets in Java.
- This is a Mini-project developed using Java, Jdbc, And Servlets.

10.2 Admin Have Following Access for this online store site

- Add New Books.
- View Books Available.
- Remove Books.
- Increase Books Amount.

10.3 Users Have Following Access for this online store site

- Create New Account or Register.
- Login.
- View Available Books.
- Select Books to Buy.
- Select Books Quantity.
- Buy Books.
- Get Payment Receipt.

10.4 Technologies used

1. **Front-End Development**

- HTML
- CSS
- Javascript
- BootStrap

2. **Back-End Development**

- Java [JDK 8+]
- JDBC

- Servlet

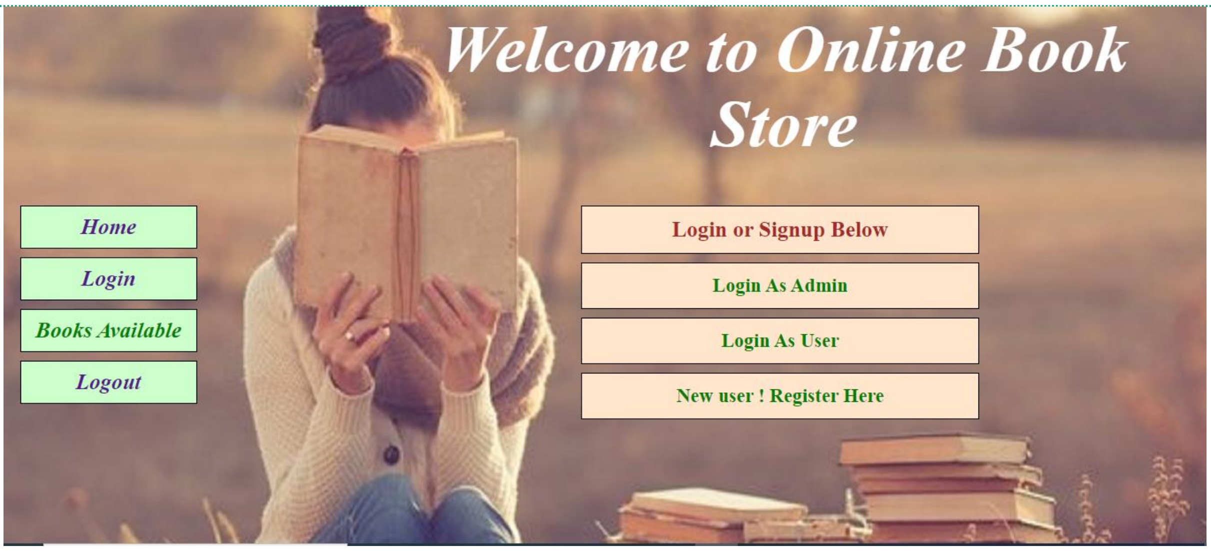
3. Database

- MySql

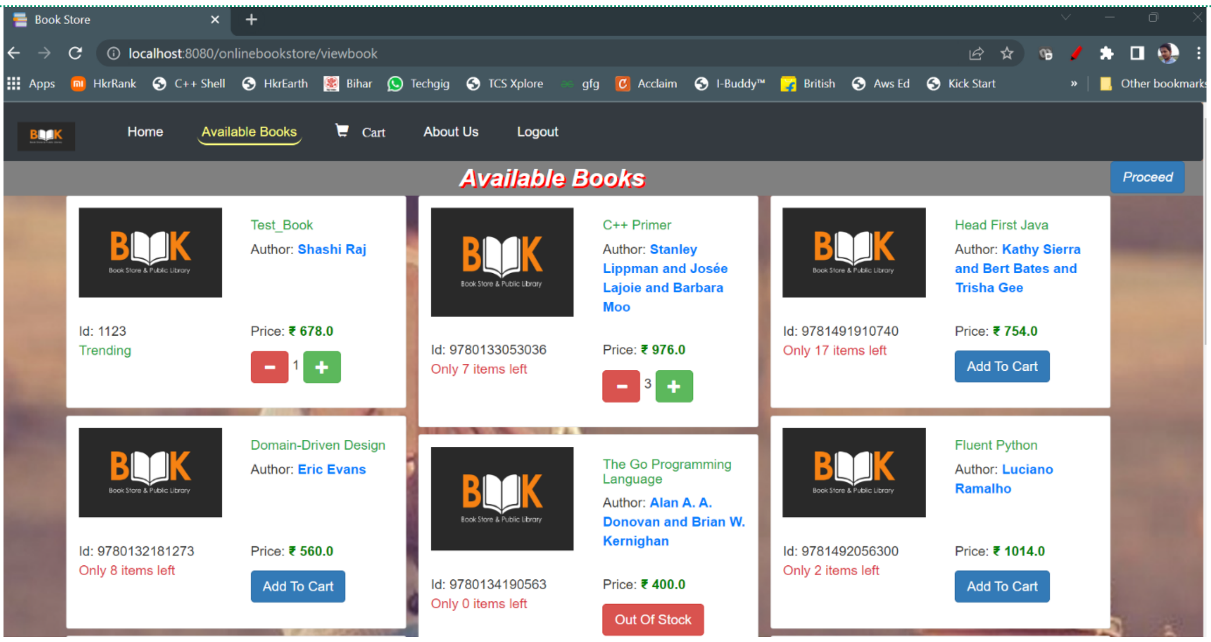
10.5 Technology used

Advance JAVA concepts like JSP, JSTL, Servlet, HTML, CSS, Boostrap 5, Fontawesome and MySQL.

10.6 Project View - Home Page



10.7 Available Books



Test_Book
Author: Shashi Raj
Id: 1123
Trending
Price: ₹ 678.0
1

C++ Primer
Author: Stanley Lippman and Josée Lajoie and Barbara Moo
Id: 9780133053036
Only 7 items left
Price: ₹ 976.0
3

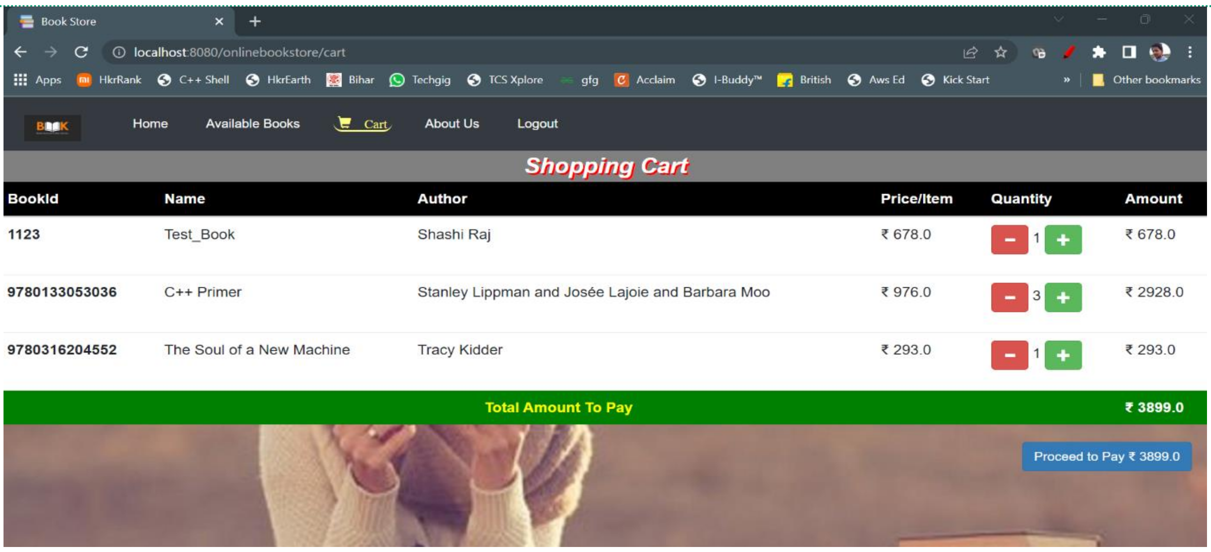
Head First Java
Author: Kathy Sierra and Bert Bates and Trisha Gee
Id: 9781491910740
Only 17 items left
Price: ₹ 754.0

Domain-Driven Design
Author: Eric Evans
Id: 9780132181273
Only 8 items left
Price: ₹ 560.0

The Go Programming Language
Author: Alan A. A. Donovan and Brian W. Kernighan
Id: 9780134190563
Only 0 items left
Price: ₹ 400.0

Fluent Python
Author: Luciano Ramalho
Id: 9781492056300
Only 2 items left
Price: ₹ 1014.0

10.8 Shopping Cart



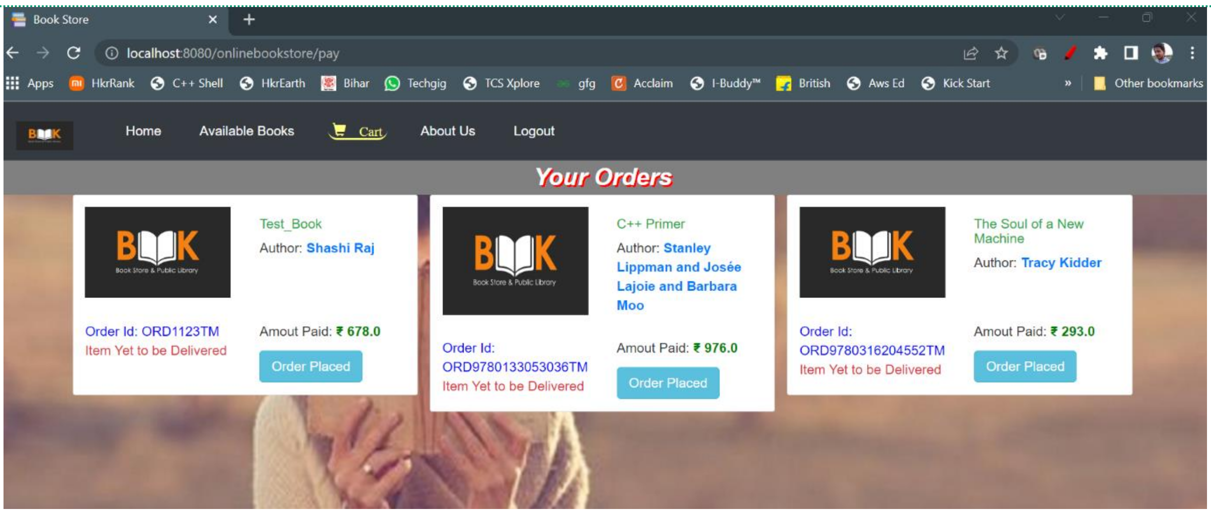
Shopping Cart

BookId	Name	Author	Price/Item	Quantity	Amount
1123	Test_Book	Shashi Raj	₹ 678.0	1	₹ 678.0
9780133053036	C++ Primer	Stanley Lippman and Josée Lajoie and Barbara Moo	₹ 976.0	3	₹ 2928.0
9780316204552	The Soul of a New Machine	Tracy Kidder	₹ 293.0	1	₹ 293.0

Total Amount To Pay ₹ 3899.0

Proceed to Pay ₹ 3899.0

10.9 Your Orders



Your Orders

Test_Book
Author: Shashi Raj
Order Id: ORD1123TM
Item Yet to be Delivered
Amout Paid: ₹ 678.0
Order Placed

C++ Primer
Author: Stanley Lippman and Josée Lajoie and Barbara Moo
Order Id: ORD9780133053036TM
Item Yet to be Delivered
Amout Paid: ₹ 976.0
Order Placed

The Soul of a New Machine
Author: Tracy Kidder
Order Id: ORD9780316204552TM
Item Yet to be Delivered
Amout Paid: ₹ 293.0
Order Placed

11. JAVA Full Stack on To-Do List Application

Here's a simple Java Full Stack exercise that an engineering student can work on. This exercise involves creating a basic web application with a front-end using HTML/CSS/JavaScript and a back-end using Java with Spring Boot.

Frontend

- ✓ Front-end (HTML/CSS/JavaScript)
- ✓ Create HTML file (index.html)

11.1 Simple Webpage

Design a simple webpage with an input field to add tasks, a list to display tasks, and buttons for adding and deleting tasks.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>To-Do List</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <div id="app">
      <h1>To-Do List</h1>
      <input type="text" id="taskInput" placeholder="Add a new task">
      <button onclick="addTask()">Add Task</button>
      <ul id="taskList"></ul>
   </div>
   <script src="script.js"></script>
</body>
</html>
```

11.2 Create CSS file (styles.css)

Style your webpage to make it visually appealing.

```
body {
   font-family: Arial, sans-serif;
   background-color: #f4f4f4;
   text-align: center;
   margin: 0;
   padding: 0;
}

#app {
   max-width: 600px;
   margin: 50px auto;
   background-color: #fff;
   padding: 20px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```

h1 {
   color: #333;
}

input {
   width: 70%;
   padding: 10px;
   margin-bottom: 10px;
}

button {
   padding: 10px;
   background-color: #4caf50;
   color: #fff;
   border: none;
   cursor: pointer;
}

ul {
   list-style-type: none;
   padding: 0;
}

li {
   background-color: #f9f9f9;
   margin: 5px 0;
   padding: 10px;
   border-radius: 5px;
}
```

11.3 Create JavaScript file (script.js)

Implement the logic for adding and deleting tasks.

```
function addTask() {
   var taskInput = document.getElementById('taskInput');
   var taskList = document.getElementById('taskList');

   if (taskInput.value.trim() !== '') {
      var newTask = document.createElement('li');
      newTask.innerText = taskInput.value;
      newTask.onclick = function () {
         this.remove();
      };

      taskList.appendChild(newTask);
      taskInput.value = '';
   }
}
```

Back-end

- ✓ Create Spring Boot Application:

Set up a new Spring Boot project using your preferred IDE or Spring Initializer (https://start.spring.io/). Include dependencies: Spring Web.

11.4 Create Controller (TaskController.java)

Implement a RESTful API to handle adding and deleting tasks.

```
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/api/tasks")
public class TaskController {

    private List<String> tasks = new ArrayList<>();

    @GetMapping
    public List<String> getTasks() {
        return tasks;
    }

    @PostMapping
    public void addTask(@RequestBody String task) {
        tasks.add(task);
    }

    @DeleteMapping("/{index}")
    public void deleteTask(@PathVariable int index) {
        if (index >= 0 && index < tasks.size()) {
            tasks.remove(index);
        }
    }
}
```

11.5 Run Your Application

Open your web browser and go to http://localhost:8080.

Now, you have a basic Java Full Stack application where users can add and delete tasks on a to-do list. This exercise covers front-end development using HTML/CSS/JavaScript and back-end development using Java with Spring Boot. You can further enhance this application by adding features like persistence, user authentication, or task prioritization.

12. Smart Shopping Cart

Creating a shopping cart using JSP (JavaServer Pages) and JDBC (Java Database Connectivity) involves building a simple web application with functionality to display products, add them to a cart, and view the items in the cart. Below is a simplified example to get you started. Note that this example uses basic JDBC for database access.

Database Table: (MySQL)

Assuming you have a table named products in your database with columns product_id, product_name, and price

12.1 Java Classes

```
Product.java (Model class).

public class Product {

    private int productId;

    private String productName;

    private double price;


    // Constructors, getters, and setters

}
```

12.2 ProductDAO.java (Data Access Object)

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;


public class ProductDAO {


    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/your_database";

    private static final String JDBC_USER = "your_username";

    private static final String JDBC_PASSWORD = "your_password";


    public List<Product> getAllProducts() {

        List<Product> products = new ArrayList<>();
        try (Connection connection = DriverManager.getConnection(JDBC_URL, JDBC_USER,
JDBC_PASSWORD);
             PreparedStatement statement = connection.prepareStatement("SELECT * FROM products");
             ResultSet resultSet = statement.executeQuery()) {

            while (resultSet.next()) {
                Product product = new Product();
                product.setProductId(resultSet.getInt("product_id"));
                product.setProductName(resultSet.getString("product_name"));
                product.setPrice(resultSet.getDouble("price"));
                products.add(product);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }

        return products;
    }
}
```

12.3 JSP Pages

index.jsp (Displaying Products)

```
<%@ page import="java.util.List" %>
<%@ page import="your.package.name.Product" %>
<%@ page import="your.package.name.ProductDAO" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Shopping Cart</title>
</head>
<body>

<h2>Product List</h2>
```

```
<%
    ProductDAO productDAO = new ProductDAO();
    List<Product> products = productDAO.getAllProducts();
%>

<table border="1">
    <tr>
        <th>Product ID</th>
        <th>Product Name</th>
        <th>Price</th>
        <th>Action</th>
    </tr>
    <% for (Product product : products) { %>
        <tr>
            <td><%= product.getProductId() %></td>
            <td><%= product.getProductName() %></td>
            <td><%= product.getPrice() %></td>
            <td><a href="addToCart.jsp?productId=<%= product.getProductId() %>">Add to 
Cart</a></td>
        </tr>
    <% } %>
</table>

</body>
</html>
```

12.4 addToCart.jsp (Adding Products to Cart)

```
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.List" %>
<%@ page import="your.package.name.Product" %>
<%@ page import="your.package.name.CartItem" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>

<%@ page import="your.package.name.Product" %>
<%@ page import="your.package.name.ProductDAO" %>
<%@ page import="your.package.name.CartItem" %>


<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="your.package.name.CartItem" %>


<%
    int productId = Integer.parseInt(request.getParameter("productId"));
    ProductDAO productDAO = new ProductDAO();
    Product product = productDAO.getProductById(productId);

    List<CartItem> cart = (List<CartItem>) session.getAttribute("cart");
    if (cart == null) {
        cart = new ArrayList<>();
        session.setAttribute("cart", cart);
    }

    CartItem cartItem = new CartItem(product, 1); // Quantity is set to 1
    cart.add(cartItem);

    response.sendRedirect("viewCart.jsp");
%>
```

12.5 viewCart.jsp (Viewing Cart)

```
<%@ page import="java.util.List" %>
<%@ page import="your.package.name.CartItem" %>


<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>


<%@ page import="your.package.name.CartItem" %>


<%@ page import="java.util.List" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>

<%@ page import="java.util.List" %>


<%

   List<CartItem> cart = (List<CartItem>) session.getAttribute("cart");

   if (cart == null || cart.isEmpty()) {

       out.println("<h2>Your cart is empty</h2>");

   } else {

%>


<h2>Shopping Cart</h2>


<table border="1">

   <tr>

       <th>Product Name</th>

       <th>Price</th>

       <th>Quantity</th>

       <th>Total</th>

   </tr>

   <%

       double grandTotal = 0;

       for (CartItem cartItem : cart) {

           out.println("<tr>");

           out.println("<td>" + cartItem.getProduct().getProductName() + "</td>");

           out.println("<td>" + cartItem.getProduct().getPrice() + "</td>");

           out.println("<td>" + cartItem.getQuantity() + "</td>");

           double total = cartItem.getProduct().getPrice() * cartItem.getQuantity();

           grandTotal += total;

           out.println("<td>" + total + "</td>");
```

```
            out.println("</tr>");

        }

    %>

</table>


<h3>Grand Total: <%= grandTotal %></h3>


<%

    }

%>


<a href="index.jsp">Continue Shopping</a>
```

This is a very basic example, and in a real-world scenario, you would need to handle more features such as updating quantities, removing items, handling user authentication, and integrating with a payment system for checkout. Additionally, using JSP for view logic is an older approach, and modern applications often use frameworks like Spring MVC or Thymeleaf for a more maintainable and modular design.