**Lab Manual**

**For**

**SKILL DEVELOPMENT COURSE**

**(NODE JS/ REACT JS/ DJANGO)**

**MANUAL**

**(CS409PC)**

**Academic Year: 2025-2026**

**II B. TECH II SEMESTER (R22 AUTONOMOUS)**



**DEPARTMENT OF CSE (Computer Science Engineering)  
 ACE**

**Engineering College**An Autonomous Institution

Ghatkesar, Hyderabad - 501 301, Telangana. Approved by AICTE & Affiliated to JNTUH NBA Accredited B.Tech Courses, Accorded NAAC A-Grade.

### 1.Department Vision, Mission, POs and PSOs

**Institute Vision:** To be a leading Technical Institute to prepare high quality Engineers to cater the needs of the stakeholders in the field of Engineering and Technology with global competence, fundamental comprehensive analytical skills, critical reasoning, research aptitude, entrepreneur skills, ethical values and social concern.

**Institute Mission:**Imparting Quality Technical Education to young Engineers by providing the state-of-the- art laboratories, quality instructions by qualified and experienced faculty and research facilities to meet the requirements of stakeholders in real time usage and in training them to excel in competitive examinations for higher education and employment to interface globally emerging techno-informative challenges in the growth corridor of techno- excellence.

**Department Vision:**To produce excellent standard, quality education of professionals by imparting cognitive learning environment, research and industrial orientation to become innovative Data Science Professional.

**Department Mission:**

* **M1:** To develop professionals in the areas of math’s (probability and statistics, linear algebra and Calculus), natural language processing, text mining, and problem solving.
* **M2:** To educate the students with the latest technologies to update their knowledge in the field of Computer Science Engineering.
* **M3:** To impart quality and value-based education and contribute towards the innovation of the computer system, Computer science to raise the satisfaction level of all stakeholders.
* **M4:** Enabling students to get expertise in critical skills with data science education and facilitate socially responsive research and innovation.
* **M5:** Our effort is to apply new advancements in high performance computing hardware and software.

**Program Educational Objectives (PEOs)**

### **Program Educational Objectives (PEOs)**

**PEO 1:** Prepare students for successful careers in Computer Science and Engineering by equipping them with the necessary skills to excel in competitive examinations for higher education and employment.

**PEO 2:** Provide a strong foundation in Computer Science and Engineering, Applied Mathematics, and Sciences, while imparting high-quality technical skills for designing, modeling, analyzing, and solving critical problems with global competence.

**PEO 3:** Foster professional, social, and ethical values, along with effective communication and entrepreneurial skills, to support students' holistic growth.

**PEO 4:** Create an academic environment that encourages multidisciplinary learning, professional engagement, and lifelong learning through collaborations with industry and professional bodies.

**PEO 5:** Develop research aptitude among students to advance knowledge in cutting-edge technologies, solve real-world problems, and offer technical consultancy services.

**Program Outcomes**

### **Program Outcomes (POs)**

**PO1:** Apply knowledge of mathematics, science, and engineering, along with fundamental principles, to solve complex computing problems.

**PO2:** Identify, formulate, and solve engineering problems effectively.

**PO3:** Design systems, components, or processes to meet desired needs in Computer Science and Engineering while considering realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability.

**PO4:** Design and conduct experiments, analyze and interpret data, and apply engineering principles for problem-solving.

**PO5:** Utilize modern techniques, skills, and tools necessary for system design in Computer Science and Engineering with embedded engineering practices.

**PO6:** Apply contextual knowledge to assess societal, health, safety, legal, and cultural issues, demonstrating responsibility in professional engineering practice.

**PO7:** Acquire a broad education to understand the impact of engineering solutions in global, economic, environmental, and societal contexts.

**PO8:** Develop a strong sense of professional and ethical responsibility.

**PO9:** Work effectively as a member or leader in multidisciplinary teams.

**PO10:** Communicate effectively in both technical and non-technical contexts.

**PO11:** Demonstrate knowledge of engineering and management principles and apply them to one's work as a member or leader in a team, managing projects in multidisciplinary environments.

**PO12:** Recognize the need for lifelong learning and engage in continuous self-improvement.

**Program Specific Outcomes**

**PSO1** Equip students with industry-relevant skills by providing training in cutting-edge technologies.

**PSO2** Develop the ability to apply core computing concepts and optimization techniques to design efficient and effective computing solutions.

**PSO3** Prepare graduates to uphold societal and ethical responsibilities in their professional careers while actively contributing to the industry and society.

# Syllabus

**SKILL DEVELOPMENT COURSE (NODE JS/ REACT JS/ DJANGO)**

**B.Tech. II Year II Sem.** **L T P C** **0 0 2 1**

### **Prerequisites:** Object Oriented Programming through Java, HTML Basics. **Course Objectives:**

The course aims to:

1. Implement static web pages using HTML and perform client-side validation using JavaScript.
2. Design and work with databases using Java.
3. Develop an end-to-end application using Java full stack.
4. Introduce Node.js for server-side programming.
5. Experiment with single-page application development using React.

##### **Course Outcomes:**

At the end of the course, students will be able to:

1. At the end of the course, students will be able to:
2. Build a custom website using HTML, CSS, Bootstrap, and JavaScript.
3. Demonstrate advanced JavaScript features and understand JDBC.
4. Develop server-side implementations using Java technologies.
5. Implement server-side development using Node.js.
6. Design a Single Page Application (SPA) using React.
7. Create complex web applications using Node.js, React.js, or Django.

##### **Exercises** **HTML / JavaScript / CSS:**

1. Build a responsive web application for a shopping cart with registration, login, catalog, and cart pages using CSS3 features, flex, and grid.
2. Enhance the above web application to be responsive using the Bootstrap framework.
3. Implement client-side validation using JavaScript for the pages developed in Experiments 1 and 2.
4. Explore ES6 features such as arrow functions, callbacks, promises, and async/await. Implement an application to read weather information from openweathermap.org and display it in graphical form on a web page.
5. Develop a Java standalone application that connects to a database (Oracle/MySQL) and performs CRUD operations on database tables.
6. Design a controller using Servlets to interact with the application from Experiment 1 and the database from Experiment 5.

##### Explore various session tracking mechanisms (Cookies, HTTP Sessions) to maintain user transaction history Node.js

1. Create a custom server using the http module and explore other Node.js modules such as OS, path, and events.
2. Develop an Express web application that interacts with a REST API to perform CRUD operations on student data. (Use Postman for testing).
3. Implement authentication and create authorized endpoints for the above application using JWT (JSON Web Token).

##### React.js:

1. Create a React application for a Student Management System with registration, login, contact, and about pages, implementing React Router for navigation.
2. Develop a React service to fetch weather data from openweathermap.org and display current and historical weather information using Chart.js.
3. Build a TODO application using React with necessary components and deploy it on GitHub.

##### Django:

1. Create a Django application for a Student Management System with registration, login, contact, and about pages, implementing Django routing for navigation.
2. Develop a Django application that fetches weather data from openweathermap.org and displays current and historical weather information using Chart.js.
3. Build a TODO application using Django with necessary components and deploy it on GitHub.

#### Reference Books:

1. Jon Duckett, Beginning HTML, XHTML, CSS, and JavaScript, Wrox Publications, 2010.
2. Bryan Basham, Kathy Sierra, Bert Bates, Head First Servlets and JSP, O'Reilly Media, 2nd Edition, 2008.
3. Vasan Subramanian, Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node, 2nd Edition, Apress.

# 

# 

# 

# Index

| **S.No.** | **List of Experiments** |
| --- | --- |
| **1** | Build a responsive web application for a shopping cart with registration, login, catalog, and cart pages using CSS3 features, flex, and grid. |
| **2** | Make the above web application responsive using the Bootstrap framework. |
| **3** | Use JavaScript for client-side validation of the pages implemented in Experiment 1 and Experiment 2. |
| **4** | Explore ES6 features like arrow functions, callbacks, promises, and async/await. Implement an application that fetches weather information from openweathermap.org and displays it in a graph. |
| **5** | Develop a Java standalone application that connects with a database (Oracle/MySQL) and performs CRUD operations on database tables. |
| **6** | Design a controller using Servlets to interact with the application from Experiment 1 and the database from Experiment 5. |
| **7** | Explore various session tracking mechanisms (Cookies, HTTP Sessions) for maintaining transactional history of users. |
| **8** | Create a custom server using the **http** module in Node.js and explore other modules like OS, path, and events. |
| **9** | Develop an Express web application that interacts with a REST API to perform CRUD operations on student data. (Use Postman). |
| **10** | Implement JWT-based authentication and create authorized endpoints for the application in Experiment 9. |
| **11** | Create a React application for a **Student Management System** with registration, login, contact, and about pages, implementing **React Router** for navigation. |
| **12** | Develop a React service to fetch weather data from **openweathermap.org** and display current and historical weather information using **Chart.js**. |
| **13** | Build a **TODO application** using React with necessary components and deploy it on **GitHub**. |
| **14** | Create a **Django application** for a **Student Management System** with registration, login, contact, and about pages, implementing **Django routing** for navigation. |
| **15** | Develop a Django application that fetches weather data from **openweathermap.org** and displays current and historical weather information using **Chart.js**. |
| **16** | Build a **TODO application** using Django with necessary components and deploy it on **GitHub**. |

# Index (Detailed)

[1.Department Vision, Mission, POs and PSOs 2](#_heading=h.94h6aeo74ovz)

[Program Educational Objectives (PEOs) 3](#_heading=h.9w0fe6ogeak0)

[Program Outcomes (POs) 3](#_heading=h.7iovbcu5sa4y)

[Syllabus 5](#_heading=h.oywvn7jx0274)

[Prerequisites: Object Oriented Programming through Java, HTML Basics.  
Course Objectives: 5](#_heading=h.udtr3kqvlixv)

[Course Outcomes: 5](#_heading=h.p5ewa6sav7a)

[Exercises  
 HTML / JavaScript / CSS: 5](#_heading=h.fev7x8rtjxkl)

[7. Explore various session tracking mechanisms (Cookies, HTTP Sessions) to maintain user transaction history  
  
Node.js 6](#_heading=h.3avgo69tdsfv)

[React.js: 6](#_heading=h.5k5lmi7kjsjl)

[Django: 6](#_heading=h.2hb802akj88p)

[Reference Books: 6](#_heading=h.vipevgtz9sfe)

[**Index 7**](#_heading=h.rpuakntp0xrp)

[**Index (Detailed) 8**](#_heading=h.ng1uz9cjwwiz)

[**Program 1 - Responsive Shopping Cart Web Application 17**](#_heading=h.3wue1hin6d7l)

[1.1 Solution : 17](#_heading=h.vea72zwav8su)

[1.2 AIM :  
To design and develop a responsive shopping cart web application with user registration, login, product catalog, and cart functionalities using HTML, CSS3 (Flexbox & Grid), and JavaScript for enhanced user experience. 17](#_heading=h.hc0z2tfds1ew)

[Steps to Implement the Solution: 17](#_heading=h.timyfsdd6jkw)

[1.3 Create the Required HTML Pages: 17](#_heading=h.nq7pu4cyz5cb)

[1.4 Code Implementation 19](#_heading=h.6r27de6zafb0)

[1.4.1 Index.html 19](#_heading=h.cl2gukswwkk3)

[1.4.2 top.html 20](#_heading=h.83wdskw0fqx6)

[1.4.3 left.html 21](#_heading=)

[1.4.4 Right.html 22](#_heading=)

[1.4.5 Login.html 23](#_heading=)

[1.4.5 Registration.html 26](#_heading=)

[1.4.6 Userprofile.html 30](#_heading=)

[1.4.7 Cse.html 33](#_heading=)

[1.4.8 Ece.html 36](#_heading=)

[1.4.9 It.html 39](#_heading=)

[1.4.10 Cart.html 41](#_heading=)

[1.4.10 catalogue.html 44](#_heading=h.get29ayj9dq0)

[1.4.11 Apply CSS3 Features for Responsiveness : CSS Code (style.css): 48](#_heading=h.tg8zs13iinur)

[**Program 2: Make the above web application responsive web application using Bootstrap framework 50**](#_heading=h.858uq64bo5q2)

[2.1 AIM: Make the above web application responsive web application using Bootstrap framework 50](#_heading=h.eqqvjehg3gcj)

[2.2 File Structure: 50](#_heading=h.esowump5i4ix)

[2.2.1 Folder 50](#_heading=h.esowump5i4ix)

[2.2.2 Subfolder / File 50](#_heading=h.esowump5i4ix)

[2.3 Code Implementation 50](#_heading=h.qsa6tq74ci7m)

[2.3.1 Paybycreditcard.html 50](#_heading=h.34m2rp95hpx4)

[2.3.2 Paybycreditcard.css 52](#_heading=)

[2.3.3 Orderconform.html 53](#_heading=h.d74q7a92cjla)

[2.4 Output : 57](#_heading=)

[2.4.1 Home Page (index.html) – Displaying the homepage layout 57](#_heading=)

[2.4.2 Registration Page (registration.html, registration\_success.html) – Form fields and successful registration message 57](#_heading=)

[2.4.3 Login Page (login.html) – Username and password fields 58](#_heading=)

[2.4.4 Catalog Pages (catalogue.html, cse.html, ece.html, it.html) – Product categories and layout using Flexbox/Grid 59](#_heading=)

[2.4.5 Cart Page (cart.html) – Cart items, total price, checkout button 60](#_heading=)

[2.4.6 Order Confirmation Page (orderconfirm.html) – Final order summary 61](#_heading=h.5ghmxolhf1z6)

[**Program 3: Client-Side Validation Using JavaScript 61**](#_heading=h.rvzyy32wr4kw)

[3.1 Aim: 61](#_heading=h.4xb13gg64fsf)

[3.2 Requirements: 61](#_heading=h.2k9pja46rc5q)

[3.3 Theory: 61](#_heading=h.7rrk43l8m9k0)

[3.4 Experiment / Steps: 61](#_heading=h.8ov4fcdxp9e6)

[3.4.1 Registration Form Validation: 61](#_heading=h.mphu1aygb1v6)

[3.4.2 Shopping Cart Validation: 62](#_heading=h.slkabjwz0wxs)

[3.5 Code Implementation: 62](#_heading=h.g5mqvqmzslfs)

[3.5.1 HTML Code for Registration Form 62](#_heading=h.n859s1mdtmcs)

[3.5.2 validation.js 62](#_heading=h.y80xl3lglvwu)

[// Display Logged-in Username (Fixing the error) 65](#_heading=h.4d5fgjon2ls1)

[// Logout Function 65](#_heading=h.ewb25fuuzq0d)

[3.6 Output Screenshots: 65](#_heading=h.819wrl3saxfb)

[3.6.1 Successful form validation 66](#_heading=h.p5jmcbcuytss)

[3.6.2 Shopping cart validation errors 66](#_heading=h.1vp9r89p41wb)

[3.7 Conclusion: 66](#_heading=h.3jfx8kjzfre6)

[3.8 Viva Questions: 67](#_heading=h.8ge58ipv1mmb)

[**Program4: Weather Information Visualization Using ES6 Features 68**](#_heading=h.ued4ewkqltuv)

[4.1 Aim: 68](#_heading=h.meksyvrgi991)

[4.2 Requirements: 68](#_heading=h.d3phxuwbi7sp)

[4.3 Theory: 68](#_heading=h.d7ixybnsgbp6)

[4.4 Code Implementations: 68](#_heading=h.tfxgvdfd9f77)

[4.5 Explanation of Concepts Covered in This Experiment 70](#_heading=h.i8cutlpjy6l8)

[4.5.1 HTML Structure: 70](#_heading=h.m5xzu84we8w4)

[4.5.2 Displaying the Current Date (JavaScript DOM Manipulation) 71](#_heading=h.wwxevi1oat2m)

[4.5.3 Taking User Input (City Name) : Code: 71](#_heading=h.2t4oqrcpwj3v)

[4.5.4 Button to Trigger Weather Fetching 71](#_heading=h.hz2fsoc52po6)

[Code: 71](#_heading=h.ihpvrvjtacu7)

[4.6 (script.js) Fetching Weather Data (Handled in script.js) 71](#_heading=h.1zj2i6app4vh)

[4.6.1 code implementation 72](#_heading=h.e4tdjiw8v4ak)

[4.7 Expected Output 74](#_heading=h.ih0q3w4ka02)

[● 4.7.1 Display Today’s Date. 74](#_heading=h.mhtxtgdqb9ua)

[● 4.7.2 City Name Input from the user. 74](#_heading=h.ucwv58q78ras)

[● 4.7.3 Weather Details (Temperature, Humidity, etc.) when the button is clicked. 74](#_heading=h.q1me0bl21rjb)

[4.8 Conclusion 74](#_heading=h.py2twpbt7g8w)

[4.9 Viva Questions 74](#_heading=h.5m5t2ohsv7rg)

[**Program 5: Java Standalone Application with MySQL (CRUD Operations) 75**](#_heading=h.5yw7xkmn9nb)

[5.1 Objective:  
Develop a Java Standalone Application that connects to a MySQL database and performs CRUD operations (Create, Read, Update, Delete) on database tables. 75](#_heading=h.k5gunixwncpj)

[5.2 Required Software & Setup 75](#_heading=h.m4jkjbfup3dm)

[5.3 Database Code Implementation: 75](#_heading=h.z8o45ffhglyq)

[5.3.1 Run the following SQL commands in MySQL Workbench or the MySQL terminal: 75](#_heading=h.weq7naqxph3i)

[5.3.2 Step 1: Create the Database & Table 75](#_heading=h.ay7nyo9vy529)

[5.4 Java Code Implementation 77](#_heading=h.zao3swspquzc)

[5.4.1 Java Code for CRUD Operations 77](#_heading=h.uld7r5wov23j)

[5.5 Step : Compile & Run the Program 80](#_heading=h.unyh50x4s4cs)

[5.5.1 Compile the Java Program 80](#_heading=h.b03qcyle52jd)

[5.5. 2 Run the Java Program 81](#_heading=h.vuahbqjt3z4o)

[5.5.2.1 Explanation (Step-by-Step) 81](#_heading=h.uin35w9gt7c0)

[5.5.2.2 What Happens When You Run This? 81](#_heading=h.p5490tj9kolr)

[Troubleshooting 82](#_heading=h.4p8wy9h4s7bh)

[5.6 Outputs: 82](#_heading=h.a19g3zuhspmy)

[5.6.1 MySQL Workbench: 82](#_heading=h.sh1op7p3iiaf)

[5.6.2 Executing Java Program 82](#_heading=h.ue74b7l8lez7)

[5.6.3 Program Output: 83](#_heading=h.ex30hovakfa9)

[5.7 Viva Questions for Experiment 5: Java Standalone Application with MySQL (CRUD Operations) 86](#_heading=h.2ev0xvt4oo87)

[5.7.1. General Questions on Java and JDBC 86](#_heading=h.gftonmjdwk17)

[5.7.2. Questions on CRUD Operations 87](#_heading=h.4m2e863r41l6)

[5.7.3. Questions on MySQL and Database Concepts 87](#_heading=h.llb0yucdskt8)

[5.7.4. Practical Implementation Questions 87](#_heading=h.ft5efmtdbrsi)

[5.8 Conclusion 88](#_heading=h.1qp4emh7v4vw)

[**Program 6: Designing a Servlet Controller for Application and Database Interaction 89**](#_heading=h.g6xeexp2uxd)

[6.1 Aim: 89](#_heading=h.3ivu94rlbggw)

[6.2 Overview: 89](#_heading=h.scincizf9f4n)

[6.2.1 What is a Servlet Controller? 89](#_heading=h.9qwbmcjx2k2e)

[6.2.1 Why Use Servlets for Controller Design? 89](#_heading=h.bcke3iub57b3)

[6.2.1 Core Features Used in This Experiment: 89](#_heading=h.tlybjzkl4dc5)

[6.3 Steps to Implement the Servlet Controller 90](#_heading=h.uvyla69o5rte)

[6.4 Code Implementation 90](#_heading=h.xvo232ie7b7b)

[6.4.1 Setting Up Database (MySQL/Oracle Schema) 90](#_heading=h.c6vkhytzrwgn)

[6.4.2 Purpose: Creates tables for users and products. 90](#_heading=h.hm1cxf6u3ffk)

[6.5 Creating the Servlet Controller (ShoppingController.java) 91](#_heading=h.5f46jfxbxcnx)

[6.5.2 Explanation: 92](#_heading=h.p6jfk7t3ovw7)

[6.6 Configuring Servlet Mapping (web.xml) 93](#_heading=h.ogpsmcueur30)

[6.6 How to Run and Test the Servlet? 93](#_heading=h.reemz718isjt)

[6.6.1 Set Up Environment: 93](#_heading=h.zaurv6t8db1t)

[6.6.2 Deploy and Run: 93](#_heading=h.51sd79pt9nzd)

[6.6.3 Test with Postman: 94](#_heading=h.xje0svmgzver)

[6.6.3.1 Register a User 94](#_heading=h.m8y4vd3jc6e8)

[6.7 Conclusion 94](#_heading=h.5082x8uucgd8)

[6.8 Viva Questions 95](#_heading=h.m5ae4lfliai4)

[**Program 7: Servlet Controller with Session Tracking 95**](#_heading=h.j4wlkybgj3gs)

[7.1 Aim: 95](#_heading=h.ukiwszd793f)

[7.2 Overview: 95](#_heading=h.jcf8apj3nio3)

[7.2.1 What is a Servlet Controller? 95](#_heading=h.v65vg6ikljfz)

[7.2.2 Why Session Tracking? 95](#_heading=h.ssjip7so2mfk)

[7.2.3 Types of Session Tracking Used in this Experiment: 96](#_heading=h.nwo3rd62ikyo)

[7.3 Steps to Implement Servlet Controller with Session Tracking 96](#_heading=h.5pz86ypxp4zq)

[7.4 Code Implementation 96](#_heading=h.r25zvaxi02xn)

[7.4.1 HTML Form (index.html) 96](#_heading=h.9pfco3ltck5w)

[Purpose: Collects user input and sends it to the Servlet. 96](#_heading=h.uzpogg6pjwne)

[7.4.2 Explanation: 96](#_heading=h.eszpl4q2rjfr)

[7.5 Servlet (LoginServlet.java) with Session Tracking 97](#_heading=h.mqvh4knkehm)

[7.5.1 Explanation: 97](#_heading=h.i8qmtahddrrp)

[7.6 Servlet (TransactionServlet.java) for Viewing History 97](#_heading=h.t5wvzq1iqxj4)

[7.6.1 Explanation (TransactionServlet.java): 98](#_heading=h.xbgp9rvrwqmv)

[7.7 Deployment Descriptor (web.xml) 99](#_heading=h.sbla04itbpps)

[7.7.1 Expected Output 99](#_heading=h.7fc5inmzk2xp)

[7.8 Conclusion 100](#_heading=h.qmgl04n7w4p6)

[7.9 Viva Questions 100](#_heading=h.depuelem65m9)

[**Program 8: Building a RESTful API using Java Servlets 100**](#_heading=h.9b99zmyneyps)

[8.1 Aim 100](#_heading=h.4jccgt99u8hz)

[8.2 Overview 100](#_heading=h.3sawouqt791g)

[8.2.1 What is a RESTful API? 100](#_heading=h.j1efv8syo4af)

[8.2.2 Why use Servlets for REST API? 101](#_heading=h.eqtegox2z2b3)

[8.3 Steps to Implement a REST API using Servlets 101](#_heading=h.tuqj2fa9iqju)

[8.4 Code Implementation 101](#_heading=h.7lwh7zgjg4d9)

[Create MySQL Database and Table 101](#_heading=h.eohs4veiobqe)

[Run this SQL query to set up the database. 101](#_heading=h.wcp3pzhve72)

[2 Servlet: StudentAPI.java (Handles CRUD operations) 102](#_heading=h.78f1rmpy327q)

[Explanation: 103](#_heading=h.5a1dj35p53hd)

[Deployment Descriptor (web.xml) 104](#_heading=h.aas48r8b4p4p)

[Explanation: 104](#_heading=h.1tilsgjktnfp)

[Testing the API using Postman 104](#_heading=h.io9ui8o03he3)

[1. Get All Students 104](#_heading=h.cl36rj6to6ux)

[● HTTP Method : `GET` 104](#_heading=h.1atlrnvk4rx7)

[● URL: http://localhost:8080/StudentAPI 104](#_heading=h.cux6c7aw5ito)

[● Body : None 104](#_heading=h.o2kmy44l3opr)

[Steps: 104](#_heading=h.laip1c20n0t2)

[2. Add Student 104](#_heading=h.g51qiw66i538)

[Steps: 105](#_heading=h.z5oujkozeeco)

[## 3. Update Student 105](#_heading=h.vv5re1aqqf4z)

[Steps: 105](#_heading=h.r6n4erkqpweu)

[4. Delete Student 105](#_heading=h.bk7p8uphlyqr)

[Summary of Operations 106](#_heading=h.c00uixkr6wss)

[10.6.1 Use Postman to test API endpoints. 106](#_heading=h.c9y0hpc2jsfw)

[Expected Output (GET request): 106](#_heading=h.bqij3evlwyhz)

[Notes 106](#_heading=h.fgodrks92jnx)

[8.6 Conclusion 106](#_heading=h.oavewpybrrzn)

[8.7 Viva Questions 106](#_heading=h.qzi32ybyx0x8)

[8.8 Final Steps 107](#_heading=h.c0lmgr5iqu2t)

[**PROGRAM 9: Building a RESTful API using Node.js and Express 107**](#_heading=h.alflqz41l5n3)

[9.1 Aim 107](#_heading=h.a4gj32pecfti)

[9.2 Overview 107](#_heading=h.ymtncxkdefcc)

[What is a RESTful API? 107](#_heading=h.hxsnzhey7g5s)

[Why Use Node.js and Express? 107](#_heading=h.grq8cjwf93v3)

[9.3 Steps to Implement 108](#_heading=h.rqh4ktyfz45n)

[9.4 Code Implementation 108](#_heading=h.a7h4ohm81bs6)

[Install Node.js and Express 108](#_heading=h.x0tldsovlp1m)

[Open a terminal and run: 108](#_heading=h.8ywlvn39rmwe)

[Explanation: 108](#_heading=h.5t066lrms0g4)

[2 Create MySQL Database and Table 108](#_heading=h.3h9rmlkfp4p5)

[Run this SQL query to set up the database: 108](#_heading=h.vcmodqbbhjat)

[3 Create the Express Server (server.js) 108](#_heading=h.3kbbpwsvyuuh)

[server.js – Main entry point for the API 109](#_heading=h.o364898qsp9d)

[// Routes 109](#_heading=h.og51iir1807r)

[// 1 Fetch all students (GET) 109](#_heading=h.j5bmuayq1qec)

[// 2 Add a new student (POST) 109](#_heading=h.bdbq7o61k32y)

[// 3 Update a student (PUT) 109](#_heading=h.jwk4do684p56)

[// 4 Delete a student (DELETE) 109](#_heading=h.7fvimqxnlqnt)

[Explanation: 110](#_heading=h.h6rvhcbcti6l)

[9.5 Running the API 110](#_heading=h.na5clf6k3lrp)

[9.6 Testing the API using Postman 110](#_heading=h.54eo1eoena99)

[9.6.1 Use Postman to test API endpoints. 111](#_heading=h.mtcxbfhx65dl)

[Expected Output (GET request): 111](#_heading=h.t8aywavdlg99)

[9.7 Conclusion 111](#_heading=h.wkxmq3nj33bz)

[9.9 Final Steps 111](#_heading=h.1t6r1xnmqgc0)

[**Program 10: Building a Full-Stack Application using React.js and Node.js 112**](#_heading=h.26wdc9xrcxlt)

[10.1 Aim 112](#_heading=h.xshmywr2oup5)

[10.2 Overview : 112](#_heading=h.zh90mfeyq168)

[10.2.1 What is a Full-Stack Application? 112](#_heading=h.7a0duvni6ee5)

[A full-stack application includes: 112](#_heading=h.18p95bvxsqs8)

[10.2.2 Why Use React.js, Node.js, and MySQL? 112](#_heading=h.mlxdwccyqy5j)

[10.3 Steps to Implement 112](#_heading=h.4ruf4auk738o)

[10.4 Code Implementation 113](#_heading=h.i3k5x74xmjfa)

[10.4.1 Backend - Node.js + Express.js (server.js) 113](#_heading=h.oy8az9g6g8jx)

[10.4.2 Create server.js to handle API requests: 113](#_heading=h.aa52injdddo5)

[// Add a student 114](#_heading=h.th6tn6eq1pvp)

[// Start server 114](#_heading=h.x4lmrr35c8uh)

[10.4.3 Explanation: 114](#_heading=h.z4hnxwmmixj7)

[10.5 Frontend - React.js (App.js) 114](#_heading=h.14exoz39jmhp)

[10.5.1 Create a React app: 114](#_heading=h.jmp94c90yd2m)

[10.5.2 Modify src/App.js to display student data: 114](#_heading=h.hnv0e1838g9)

[// Fetch students from backend 114](#_heading=h.qx7xzn3u98o1)

[// Add a student 114](#_heading=h.uxaes6j2txsw)

[10.5.6 Explanation: 115](#_heading=h.uf4bv2sks01v)

[10.6 Running the Application 115](#_heading=h.zbh4jcbve93c)

[10.6.1 Start the Backend 115](#_heading=h.ood5rvhxtgz3)

[node server.js 115](#_heading=h.pn8avygjstq0)

[10.6.2 Start the React Frontend 115](#_heading=h.hjvpz8fmj5rq)

[10.6.3 Expected Output: 115](#_heading=h.h8r60syx063p)

[10.6 Testing the API using Postman 116](#_heading=h.n7dqod2r0z8c)

[10.6.1 Use Postman to test API endpoints. 116](#_heading=h.2vmhj7jfyddw)

[Expected Output (GET request): 116](#_heading=h.rbtfsds9wl1l)

[10.8 Conclusion 116](#_heading=h.vkkxcgwjvh6i)

[10.8 Viva Questions 116](#_heading=h.tfchqued287i)

[10.9 Final Steps  
  
 Run the full-stack app and take screenshots.  
 Ensure the database operations work properly. 117](#_heading=h.2altae1i6uwz)

[**Experiment 11: Building a TODO Application using React.js and Django 117**](#_heading=h.mlltwxyiuhdu)

[11.1 Aim 117](#_heading=h.3f275z2tq4ai)

[11.2 Overview 117](#_heading=h.3fugle9pmqhp)

[11.3 Steps to Implement 118](#_heading=h.n3i7jkj9z7yj)

[11.4 Code Implementation 118](#_heading=h.jws9wkabcp2x)

[Backend - Django + DRF (API) 118](#_heading=h.2abnteogf5ni)

[Define the Task Model (models.py): 119](#_heading=h.k1d5aov2ij1)

[Create API Serializer (serializers.py): 119](#_heading=h.fh3xo8d1h3vv)

[Define API Views (views.py): 119](#_heading=h.3x2v2lmmrb0m)

[Set up URL Routing (urls.py in todo app): 119](#_heading=h.rpc8rw74pzjl)

[Migrate Database and Run Server: 119](#_heading=h.r2xctbwdxd5d)

[Frontend - React.js (TODO App UI) 120](#_heading=h.d3cintbwtq1b)

[Create a React app: 120](#_heading=h.jztf8krp8tk6)

[Modify src/App.js to interact with the API: 120](#_heading=h.6czc0za10av3)

[// Fetch tasks from backend 120](#_heading=h.tpa46lpgfgfa)

[// Add a new task 120](#_heading=h.aaaxmsc63rtj)

[// Toggle task completion 120](#_heading=h.13xvnk4vp9va)

[// Delete a task 120](#_heading=h.yhhq8r45t6ha)

[Explanation: 121](#_heading=h.ga0lpferq5lm)

[11.5 Running the Application 121](#_heading=h.u5zc2icf29lw)

[Start the Django Backend 121](#_heading=h.bx5dinqir4s)

[Start the React Frontend 121](#_heading=h.1khs6z5j2brg)

[Expected Output: 122](#_heading=h.vlj9p37p43m)

[11.6 Testing the API using Postman 122](#_heading=h.axx30hsqf8ro)

[Expected API Response (GET request) 122](#_heading=h.wshy993g8xa7)

[11.7 Conclusion 122](#_heading=h.r67ohxx6xv84)

[11.8 Viva Questions 122](#_heading=h.bvflm33ybj9r)

[Final Steps 123](#_heading=h.6hfd9crm6bff)

[**Experiment 12: Weather API Integration using React.js 123**](#_heading=h.i6hu2kz4znhx)

[12.1 Aim 123](#_heading=h.8sozudok4vac)

[12.2 Overview 123](#_heading=h.2psbp4rcxgcj)

[What is a Weather API? 123](#_heading=h.qty1lcui5a2r)

[Why Use React.js? 123](#_heading=h.q4nqqpj7qkqa)

[12.3 Steps to Implement 124](#_heading=h.v7e5yfofude1)

[12.4 Code Implementation 124](#_heading=h.ukdbm37ppdj5)

[Get an API Key from OpenWeatherMap 124](#_heading=h.6mezxx0lpa9)

[Set up React.js Project 124](#_heading=h.8mk1zpvv1knq)

[Fetch Weather Data in App.js 124](#_heading=h.79z6llwca8y)

[Modify src/App.js: 124](#_heading=h.qxyum281vbkh)

[Explanation: 125](#_heading=h.qj8v077ibu09)

[12.5 Running the Application 125](#_heading=h.8hhv5mohc47)

[12.5.1 Start React App 125](#_heading=h.qy7qbsmnehdi)

[Expected Output: 126](#_heading=h.c7ynyley1kil)

[12.6 Testing the API using Postman 126](#_heading=h.pooio3l7fn8t)

[Operation 126](#_heading=h.f7p97pjwei49)

[HTTP Method 126](#_heading=h.f7p97pjwei49)

[URL 126](#_heading=h.f7p97pjwei49)

[Get Weather Data 126](#_heading=h.f7p97pjwei49)

[GET 126](#_heading=h.f7p97pjwei49)

[https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR\_API\_KEY&units=metric 126](#_heading=h.f7p97pjwei49)

[Expected API Response: 126](#_heading=h.cg7nrxq4gqw6)

[12.7 Conclusion 126](#_heading=h.alpd62d7pjiy)

[12.8 Viva Questions 126](#_heading=h.x6enfwgkqc8r)

[12.9 Final Steps 127](#_heading=h.6anbqqdk2coc)

[Experiment 13: Student Management System using React.js & Django 128](#_heading=h.xm8s6qd0yozz)

[13.1 Aim : To develop a Student Management System using 128](#_heading=h.jjhk10o0hbtq)

[13.2 Overview 128](#_heading=h.kskv4zrryji0)

[What is a Student Management System? 128](#_heading=h.teibk9mb5ev6)

[Why Use React.js & Django? 128](#_heading=h.qi1ldtrik9s2)

[13.3 Steps to Implement 128](#_heading=h.ho86wqfe0171)

[13.4 Code Implementation 129](#_heading=h.11vlw145fbj6)

[13.4.1 Set up Django Backend with MySQL 129](#_heading=h.1xo6fkbfwavr)

[Install Django & Django REST Framework 129](#_heading=h.zfw3jzlqxqzi)

[Create a Django project and app 129](#_heading=h.hx0juvwryp4y)

[Define Student Model (models.py) 129](#_heading=h.wptgdvux94qd)

[Modify students/models.py: 129](#_heading=h.xzhxvqj6pye8)

[Explanation: 129](#_heading=h.x3nk6lhbfk3b)

[Create API using Django REST Framework 129](#_heading=h.92bgwuh75n43)

[Modify students/serializers.py: 129](#_heading=h.7rmxwk5cabes)

[Modify students/views.py: 130](#_heading=h.1dhb6tgkw124)

[Modify students/urls.py: 130](#_heading=h.ai2of6he6i2m)

[Modify student\_management/settings.py: 130](#_heading=h.uot5lwe2wzc3)

[Apply Migrations 130](#_heading=h.yg1tlcc9uloz)

[Run Django Server 130](#_heading=h.feqftinfball)

[API Endpoint: 130](#_heading=h.eghnwfz1bbda)

[Set up React.js Frontend 130](#_heading=h.977ebcvruzg)

[Modify src/App.js: 130](#_heading=h.55b6vz363eom)

[Running the Application 132](#_heading=h.m1vwe9bm3ihs)

[Start Django Backend 132](#_heading=h.p197sasr82la)

[Start React Frontend  
 npm start 132](#_heading=h.8ml6z1jg4enr)

[Expected Output: 132](#_heading=h.e7xvkopmhjel)

[13.5 Testing API using Postman 132](#_heading=h.r3tr74xcrl97)

[13.6 Conclusion 133](#_heading=h.wc3dr1m9j5z3)

[13.7 Viva Questions 133](#_heading=h.ikgq9uc7thg9)

[13.8 Final Steps 133](#_heading=h.ci8s5zsstlri)

[**Program 14: Student Management System using Django 134**](#_heading=h.b3fagsvx7rz4)

[14.1 Aim: 134](#_heading=h.d8d959duevrf)

[14.2 Overview 134](#_heading=h.5czn1uep7nqq)

[14.2.1 What is a Student Management System? 134](#_heading=h.yjiijikzbwd8)

[14.2.2 Why Use Django? 134](#_heading=h.sot2e9vpa4rl)

[14.3 Steps to Implement: 134](#_heading=h.822nygfys280)

[14.4 Code Implementation 135](#_heading=h.uq4r36lfjbqq)

[Set Up Django Project & Application 135](#_heading=h.i2hxs3vuh1pz)

[Install Django 135](#_heading=h.8l63rju2kud2)

[Create Django Project & App 135](#_heading=h.9hxn5k9okk4m)

[Add `students` to `INSTALLED\_APPS` in `settings.py` 135](#_heading=h.vomyk9ql7a3z)

[Configure URL Routing for Navigation 135](#_heading=h.gq52g1lfn7e)

[Modify `student\_management/urls.py`: 135](#_heading=h.dz2lwcpxzmhx)

[Implement User Registration & Login 135](#_heading=h.qfsxg9ik9cwc)

[Modify `students/models.py`: 136](#_heading=h.gjl3ymql6d6l)

[Apply Migrations 136](#_heading=h.cm9xmh83qsvq)

[Modify `students/forms.py`: 136](#_heading=h.olhsulk208bs)

[Modify `students/views.py`: 136](#_heading=h.sqky0bvpw7f1)

[Create Contact & About Pages 137](#_heading=h.8hm8q8q9ksb)

[Create `templates/students/home.html` 137](#_heading=h.y8ooba30fx9u)

[Create `templates/students/register.html` 138](#_heading=h.qm52uakm6y0n)

[Create `templates/students/login.html` 138](#_heading=h.jy77hnb3lr8d)

[Create `templates/students/contact.html` 138](#_heading=h.4is2n7ebom1l)

[Create `templates/students/about.html` 138](#_heading=h.r7l2ga42coii)

[Test the Application 138](#_heading=h.3j7g0ykabgu9)

[Run the Django Server 138](#_heading=h.dw1n841valw6)

[14.4.1 Access the application in a browser: 138](#_heading=h.5njy1sliks1l)

[14.5 Expected Output 139](#_heading=h.krdqoxmnykcq)

[14.6 Viva Questions 139](#_heading=h.25c0plwyn7o8)

[**Program 15: Weather Application using Django & OpenWeatherMap API 139**](#_heading=h.ail4x17z78xl)

[Objective 139](#_heading=h.o65lqyi36bt4)

[Software & Tools Required 139](#_heading=h.21r4x0cooy3d)

[Step 1: Install Required Packages 139](#_heading=h.ravlqemvp1h5)

[Step 2: Create Django Project and App 140](#_heading=h.sgrysdqniuul)

[Create Django Project 140](#_heading=h.c285wd8ma0va)

[Create Django App 140](#_heading=h.yejqxlazj00c)

[Step 3: Add weather App to Django Settings 140](#_heading=h.lnzvz663pnli)

[Step 4: Get OpenWeatherMap API Key 141](#_heading=h.xj3b9cqjzx9m)

[Step 5: Create a Weather Model 141](#_heading=h.ys0kap5hhnmt)

[Run Migrations 141](#_heading=h.rpowik1nfsf5)

[Step 6: Fetch Weather Data from API 142](#_heading=h.vedzmwe8vi6d)

[Step 7: Configure URLs 143](#_heading=h.1abj0raafjhr)

[Modify weather/urls.py 143](#_heading=h.ccw9ekx0xz93)

[Modify experiment15/urls.py 144](#_heading=h.23y9fi45l393)

[Step 8: Create HTML Page 144](#_heading=h.b1sxcrxmlpt7)

[Step 9: Run the Django Server 146](#_heading=h.hha8g6azgxvu)

[Conclusion 146](#_heading=h.thpibgpc1lr3)

[**Experiment 16: TODO Application using Django & GitHub Deployment 147**](#_heading=h.ilp37f21dtf1)

[16.1 Aim : 147](#_heading=h.byrjlmxxsywg)

[16.2 Overview 147](#_heading=h.165xg23ccenk)

[16.2.1 Why Use Django for a TODO App? 147](#_heading=h.5xfu75icqwu2)

[16.2.2 Why Deploy on GitHub? 147](#_heading=h.ehcb5bhgnp1l)

[16.3 Install Required Software: 147](#_heading=h.estnmg34djlc)

[16.3.1 Python (Download from [python.org](https://www.python.org/)) : 148](#_heading=h.goeca9mmxwof)

[16.3.2 Visual Studio Code : 148](#_heading=h.93qnmzc4b09r)

[16.3.4 Git (Download from [git-scm.com](https://git-scm.com/)) 148](#_heading=h.n33qrcu7v4ot)

[16.4 Steps to Implement 149](#_heading=h.oa38cnp5oas2)

[16.5 Code Implementation 149](#_heading=h.df2wor4ipchx)

[16.5.1 Install Django & Create Project 149](#_heading=h.42ylktrcjkfz)

[16.5.2 Create Models for Task Management 149](#_heading=h.5x4ka5jbtna2)

[Modify `todo/models.py` 149](#_heading=h.1t4rq7mhcgz7)

[16.5.3 Implement Views & Templates 150](#_heading=h.4xt8pnvxn6v)

[Modify `todo/views.py`   
D:\College\ACE College\LABS\SDC LAB\Experiment16\DjangoTODO\todo\admin.py 150](#_heading=h.sjxxmpey94ei)

[Create Forms & Templates 151](#_heading=h.tyze2uhhjczy)

[Create `templates/todo/task\_list.html` 151](#_heading=h.khu4k7u1h7dj)

[Deploy the Project on GitHub 152](#_heading=h.22z9g2ei4wvs)

[Create a GitHub Repository 152](#_heading=h.7uy93t27vrzg)

[2. Run the following commands to push code: 152](#_heading=h.bfuslrkpxhtv)

[Ignore Sensitive Files 152](#_heading=h.hmo97o47r6li)

[16.5 Expected Output 153](#_heading=h.7woht3x28lku)

[16.6 Viva Questions 155](#_heading=h.nrirhyd2zuq1)

# 

# 

# 

# 

# Program 1 - Responsive Shopping Cart Web Application

Build a responsive web application for a shopping cart with registration, login, catalog, and cart pages using CSS3 features, flex, and grid.

## 1.1 Solution :

## 1.2 AIM : To design and develop a responsive shopping cart web application with user registration, login, product catalog, and cart functionalities using HTML, CSS3 (Flexbox & Grid), and JavaScript for enhanced user experience.

##### Steps to Implement the Solution:

### 1.3 Create the Required HTML Pages:

The web application consists of the following HTML pages:

| **Folder/File** | **Description** |
| --- | --- |
| ShoppingCartApp | Root directory of the application |
| public | Public assets folder |
| public/css | Stylesheets folder |
| public/css/style.css | Main stylesheet |
| public/images | Image assets folder |
| public/js | JavaScript files folder |
| public/js/script.js | Main script file |
| public/html | HTML pages folder |
| public/html/index.html | Home Page |
| public/html/top.html | Header Section |
| public/html/left.html | Left Sidebar - Navigation |
| public/html/right.html | Right Sidebar |
| public/html/login.html | User Login Page |
| public/html/registration.html | User Registration Page |
| public/html/userprofile.html | User Profile Page |
| public/html/cse.html | CSE Book Section |
| public/html/ece.html | ECE Book Section |
| public/html/it.html | IT Book Section |
| public/html/cart.html | Shopping Cart Page |
| public/html/catalogue.html | Book Catalog Page |
| public/html/payment.html | Payment Page |
| public/html/order-confirmation.html | Order Confirmation Page |
| server.js | Node.js Express Server |
| package.json | Project dependencies and scripts |
| README.md | Project documentation |

### 

### 1.4 Code Implementation

#### 1.4.1 Index.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<!-- Metadata and Page Title -->**

**<meta charset="UTF-8"> <!-- Ensures proper text encoding -->**

**<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Enables responsiveness -->**

**<title>Welcome to ACE e-Book's Website</title> <!-- Title shown in the browser tab -->**

**</head>**

**<!-- Using <frameset> to divide the webpage into multiple sections (Deprecated in HTML5) -->**

**<frameset rows="35%, \*">**

**<!-- Top Frame (Header Section) -->**

**<frame name="top" src="top.html" noresize frameborder="0" scrolling="no" />**

**<!-- Nested Frameset: Dividing Remaining Space into Two Columns -->**

**<frameset cols="15%, \*">**

**<!-- Left Frame (Navigation Sidebar) -->**

**<frame name="left" src="left.html" noresize frameborder="0" scrolling="no">**

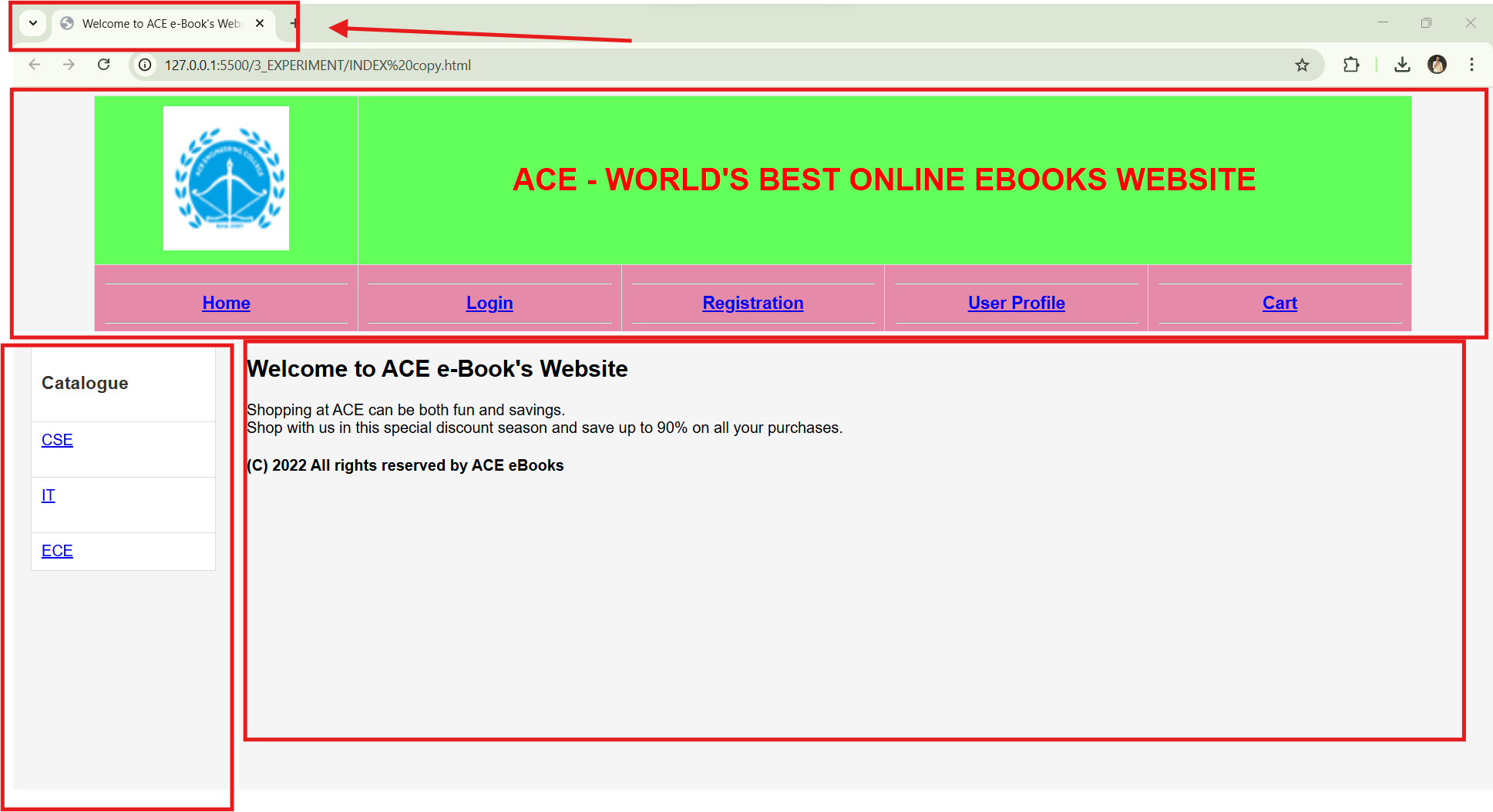
**<!-- Right Frame (Main Content Area) -->**

**<frame name="right" src="right.html" noresize frameborder="0">**

**</frameset>**

**</frameset>**

**<!-- Closing the HTML tag -->**

**</html>**  


### 1.4.2 top.html

**<!DOCTYPE html>  
<html lang="en"><head>  
<meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Welcome to ACE e-Book's Website</title>**

**<link rel="stylesheet" href="styles.css"> <!-- Linking an external CSS file for styling -->**

**</head>**

**<body>**

**<!-- Navigation Table (Full Width) -->**

**<table width="100%" align="center" cellpadding="0" cellspacing="2">**

**<tr>**

**<!-- Website Logo -->**

**<th width="20%">**

**<img src="books/logo.png" alt="ACE LOGO" width="130" height="150"/>**

**</th>**

**<!-- Website Title -->**

**<th colspan="4">**

**<h1 style="color:red;">ACE - WORLD'S BEST ONLINE EBOOKS WEBSITE</h1>**

**</th>**

**</tr>**

**<!-- Navigation Menu -->**

**<tbody align="center" bgcolor="#e68aaa" style="font-weight:bold; font-size:18px;">**

**<tr>**

**<!-- Navigation Links -->**

**<td width="20%"><hr><a href="right.html" target="right">Home</a><hr></td>**

**<td width="20%"><hr><a href="login.html" target="right">Login</a><hr></td>**

**<td width="20%"><hr><a href="registration.html" target="right">Registration</a><hr></td>**

**<td width="20%"><hr><a href="userprofile.html" target="right">User Profile</a><hr></td>**

**<td width="20%"><hr><a href="cart.html" target="right">Cart</a><hr></td>**

**</tr>**

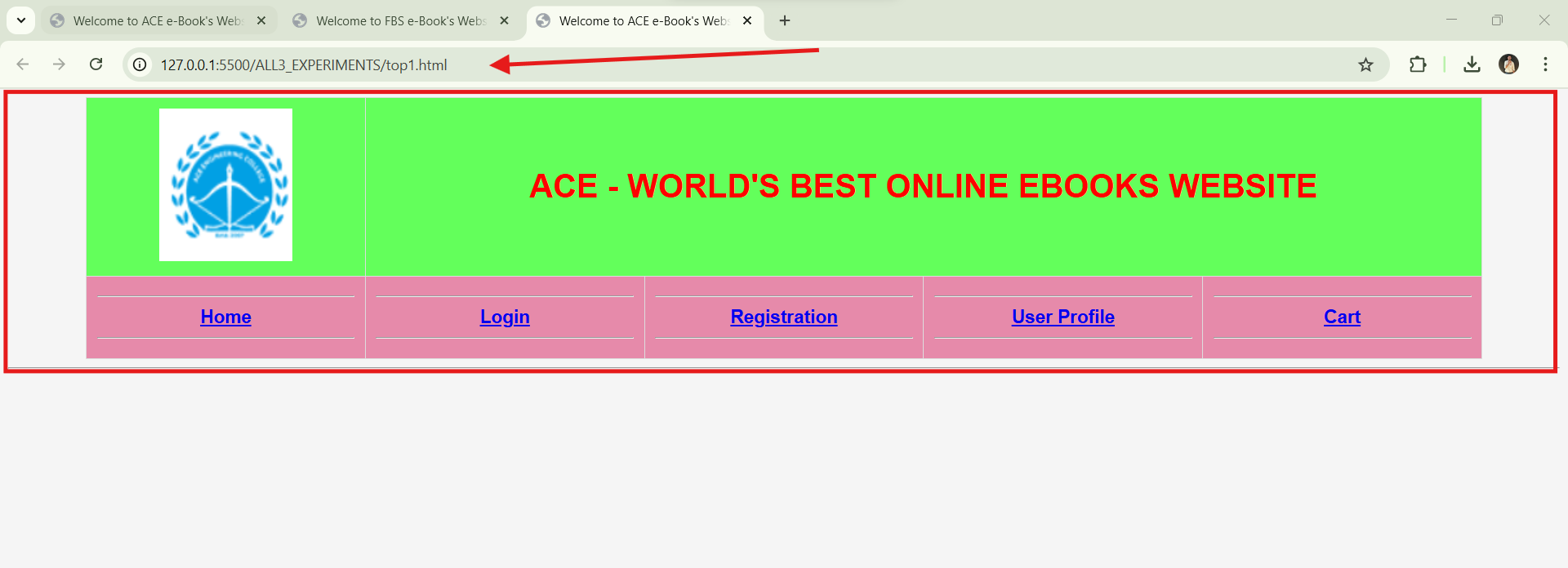
**</tbody>**

**</table>**

**<hr/> <!-- Horizontal line for separation -->**

**</body>**

**</html>**



### 1.4.3 left.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<title>Catalogue</title>**

**<link rel="stylesheet" href="styles.css" />**

**<!-- Linking external CSS -->**

**</head>**

**<body bgcolor="#D6E9EF">**

**<!-- Background color of the sidebar -->**

**<!-- Sidebar Menu Table -->**

**<table align="center">**

**<tr>**

**<td><h3>Catalogue</h3></td>**

**<!-- Heading for the sidebar -->**

**</tr>**

**<tr>**

**<td><a href="cse.html" target="right">CSE</a><br /><br /></td>**

**<!-- Link to CSE Books -->**

**</tr>**

**<tr>**

**<td><a href="it.html" target="right">IT</a><br /><br /></td>**

**<!-- Link to IT Books -->**

**</tr>**

**<tr>**

**<td><a href="ece.html" target="right">ECE</a></td>**

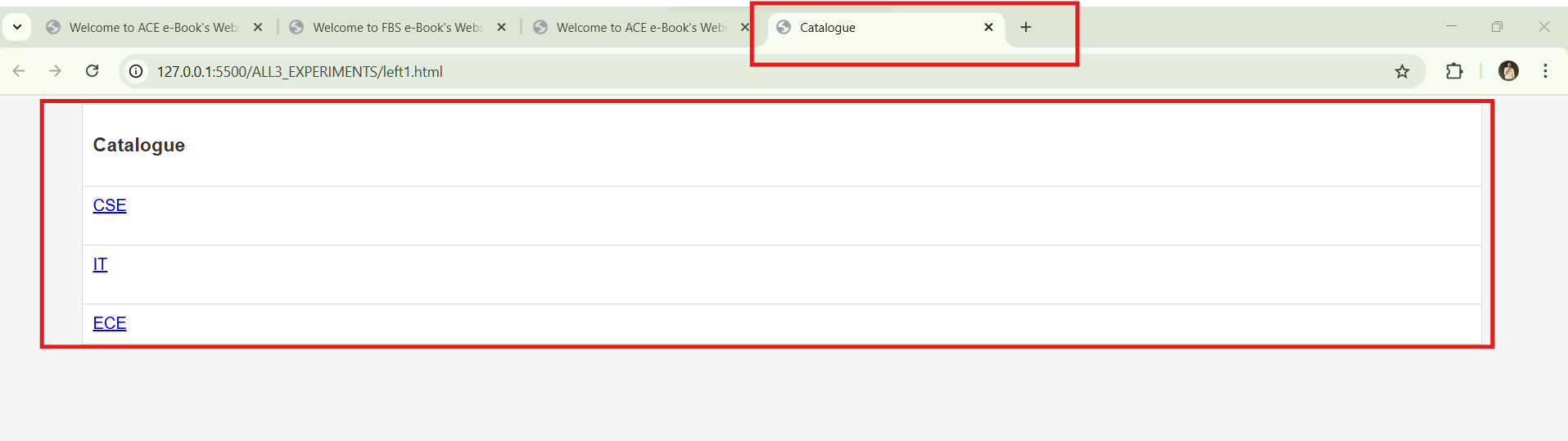
**<!-- Link to ECE Books -->**

**</tr>**

**</table>**

**</body>**

**</html>**



### 1.4.4 Right.html

**<!DOCTYPE html>**

**<!-- Defines the document type as HTML5 -->**

**<html lang="en">**

**<!-- The root element of the HTML document, with English as the language -->**

**<head>**

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

**<!-- Defines the character encoding as UTF-8 (supports all characters) -->**

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<!-- Makes the website responsive for all devices -->**

**<title>Welcome to ACE e-Book's Website</title>**

**<!-- Sets the title of the web page (shown in the browser tab) -->**

**<link rel="stylesheet" href="styles.css" />**

**<!-- Links an external CSS file for styling -->**

**</head>**

**<body>**

**<!-- The body contains all visible content of the webpage -->**

**<div class="container">**

**<!-- A container div to wrap the content -->**

**<h2>Welcome to ACE e-Book's Website</h2>**

**<!-- Heading for the page -->**

**<p>**

**<!-- Paragraph with highlighted text -->**

**Shopping at <span class="highlight">ACE</span> can be both**

**<span class="highlight">fun</span> and**

**<span class="highlight">savings</span>.<br />**

**Shop with us in this special**

**<span class="highlight">discount</span> season and save up to**

**<span class="highlight">90%</span> on all your purchases.<br />**

**</p>**

**<div class="spacer"></div>**

**<!-- Empty div used for spacing -->**

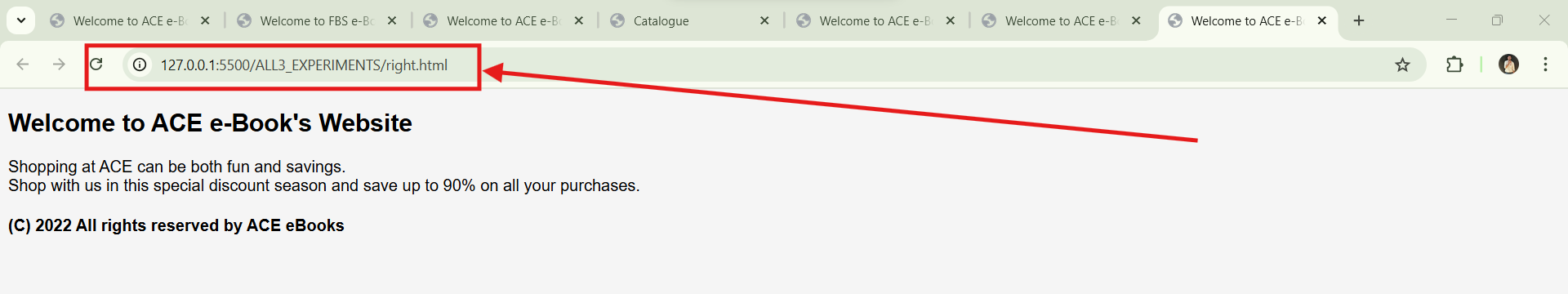
**<!-- Footer section -->**

**<h4 class="footer-text">(C) 2022 All rights reserved by ACE eBooks</h4>**

**</div>**

**</body>**

**</html>**



### 1.4.5 Login.html

**<html>**

**<head>**

**<title>Welcome to ACE e-Book's Website</title>**

**<!-- Sets the title of the webpage -->**

**<script>**

**function validateLoginForm() {**

**var username = document.forms["f1"]["username"].value; // Gets the username input value**

**var password = document.forms["f1"]["password"].value; // Gets the password input value**

**if (username.trim() === "") {**

**// Checks if the username is empty**

**alert("Username cannot be empty"); // Shows an alert if empty**

**return false; // Prevents form submission**

**}**

**if (password.length < 6) {**

**// Checks if the password length is less than 6 characters**

**alert("Password must be at least 6 characters long"); // Shows an alert if too short**

**return false; // Prevents form submission**

**}**

**return true; // Submits the form if validations pass**

**}**

**</script>**

**</head>**

**<body>**

**<center>**

**<!-- Centers all the content inside -->**

**<br />**

**<h3>Login Details</h3>**

**<!-- Heading for the login section -->**

**<br />**

**<form name="f1" onsubmit="return validateLoginForm()">**

**<!-- Form with validation function -->**

**<table align="center">**

**<!-- Creates a table layout for the form -->**

**<tr>**

**<td>Username :</td>**

**<td><input type="text" name="username" /></td>**

**<!-- Input field for username -->**

**</tr>**

**<tr>**

**<td><br /></td>**

**<!-- Empty row for spacing -->**

**</tr>**

**<tr>**

**<td>Password :</td>**

**<td><input type="password" name="password" /></td>**

**<!-- Input field for password -->**

**</tr>**

**<tr>**

**<td><br /></td>**

**<!-- Empty row for spacing -->**

**</tr>**

**<tr>**

**<td></td>**

**<td>**

**<input type="submit" value="SUBMIT" />**

**<!-- Button to submit the form -->**

**<input type="reset" value="RESET" />**

**<!-- Button to reset form fields -->**

**</td>**

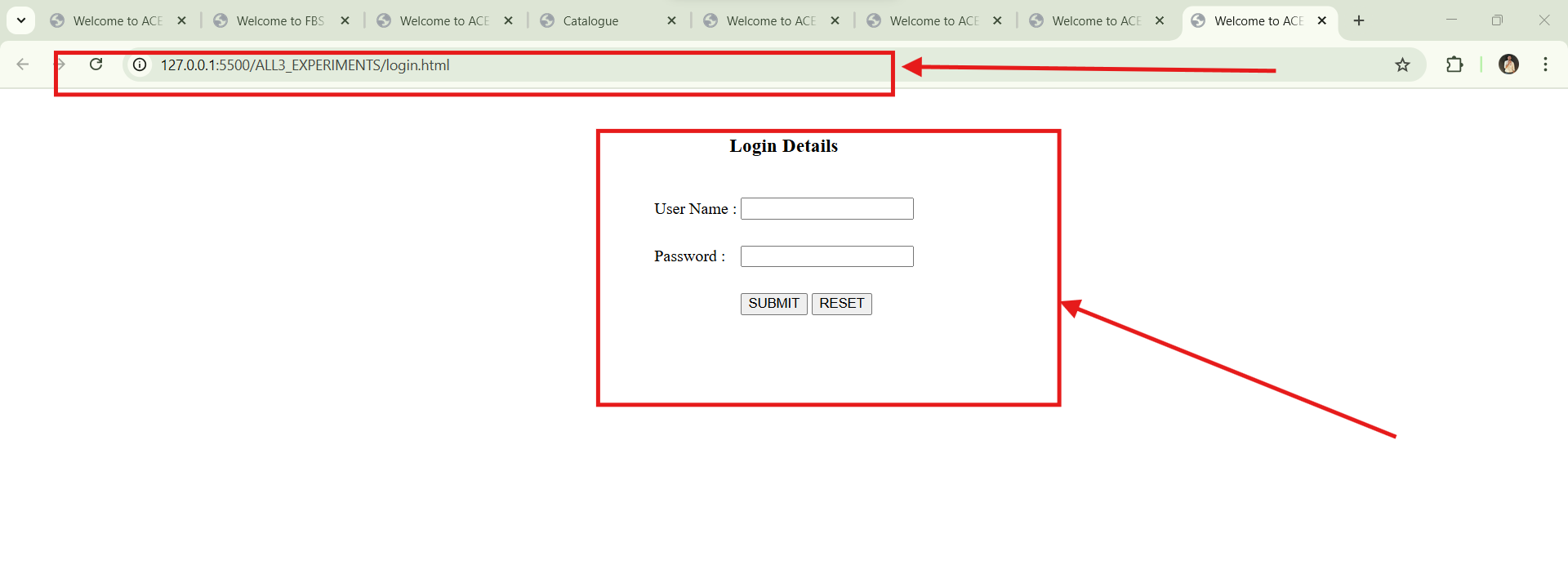
**</tr>**

**</table>**

**</form>**

**</center>**

**</body>**

**</html>**

### 1.4.5 Registration.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

**<!-- Specifies character encoding as UTF-8 for better compatibility with different languages -->**

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<!-- Ensures the page is responsive on all devices -->**

**<title>Welcome to ACE e-Book's Website</title>**

**<!-- Title of the webpage -->**

**<script>**

**function validateForm() {**

**// Fetching input values from the form using document.forms**

**let username = document.forms["f1"]["username"].value; // Getting username input value**

**let password = document.forms["f1"]["password"].value; // Getting password input value**

**let email = document.forms["f1"]["email"].value; // Getting email input value**

**let phone = document.forms["f1"]["phno"].value; // Getting phone number input value**

**let gender = document.querySelector('input[name="gender"]:checked'); // Fetching the selected radio button for gender**

**let languages = document.querySelectorAll(**

**'input[name="languages[]"]:checked'**

**); // Fetching all checked checkboxes for languages**

**let address = document.forms["f1"]["address"].value; // Getting address input value**

**// Regular expressions for validation**

**let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Email should be in the format: example@domain.com**

**let phonePattern = /^\d{10}$/; // Phone number should be exactly 10 digits**

**let passwordPattern = /^(?=.\*[A-Za-z])(?=.\*\d)[A-Za-z\d]{6,}$/; // Password should have at least 6 characters, 1 letter, and 1 digit**

**// Validation checks for each field**

**if (username.trim().length < 3) {**

**alert("Name must be at least 3 characters long.");**

**return false; // Stops form submission**

**}**

**if (!passwordPattern.test(password)) {**

**alert(**

**"Passwords must be at least 6 characters long and contain at least one letter and one number."**

**);**

**return false;**

**}**

**if (!emailPattern.test(email)) {**

**alert("Enter a valid email ID.");**

**return false;**

**}**

**if (!phonePattern.test(phone)) {**

**alert("Phone number must be exactly 10 digits.");**

**return false;**

**}**

**if (!gender) {**

**// If no gender is selected**

**alert("Please select a gender.");**

**return false;**

**}**

**if (languages.length === 0) {**

**// If no language is selected**

**alert("Please select at least one language.");**

**return false;**

**}**

**if (address.trim().length < 10) {**

**alert("Address must be at least 10 characters long.");**

**return false;**

**}**

**return true; // If all validations pass, allow form submission**

**}**

**</script>**

**</head>**

**<body>**

**<center>**

**<h3>Registration Form</h3>**

**<!-- Heading of the form -->**

**<form name="f1" onsubmit="return validateForm()">**

**<!-- Form with validation on submission -->**

**<table cellpadding="5" align="center">**

**<!-- Table layout for better alignment -->**

**<!-- Name Field -->**

**<tr>**

**<td>Name:\*</td>**

**<td><input type="text" name="username" required /></td>**

**<!-- Required field -->**

**</tr>**

**<!-- Password Field -->**

**<tr>**

**<td>Password:\*</td>**

**<td><input type="password" name="password" required /></td>**

**<!-- Required field -->**

**</tr>**

**<!-- Email Field -->**

**<tr>**

**<td>Email ID:\*</td>**

**<td><input type="text" name="email" required /></td>**

**<!-- Required field -->**

**</tr>**

**<!-- Phone Number Field -->**

**<tr>**

**<td>Phone Number:\*</td>**

**<td><input type="text" name="phno" required /></td>**

**<!-- Required field -->**

**</tr>**

**<!-- Gender Selection -->**

**<tr>**

**<td>Gender:\*</td>**

**<td>**

**<input type="radio" name="gender" value="Male" /> Male**

**<!-- Radio button for Male -->**

**<input type="radio" name="gender" value="Female" /> Female**

**<!-- Radio button for Female -->**

**</td>**

**</tr>**

**<!-- Language Selection -->**

**<tr>**

**<td>Language Known:\*</td>**

**<td>**

**<input type="checkbox" name="languages[]" value="English" />**

**English**

**<input type="checkbox" name="languages[]" value="Telugu" /> Telugu**

**<input type="checkbox" name="languages[]" value="Hindi" /> Hindi**

**<input type="checkbox" name="languages[]" value="Tamil" /> Tamil**

**</td>**

**</tr>**

**<!-- Address Field -->**

**<tr>**

**<td>Address:\*</td>**

**<td><textarea name="address" required></textarea></td>**

**<!-- Required textarea field -->**

**</tr>**

**<!-- Submit & Reset Buttons -->**

**<tr>**

**<td></td>**

**<td>**

**<input type="submit" value="Submit" />**

**<!-- Submit button -->**

**<input type="reset" value="Reset" />**

**<!-- Reset button -->**

**</td>**

**</tr>**

**<!-- Mandatory Fields Notice -->**

**<tr>**

**<td colspan="2">**

**\* <span style="color: red">Fields are mandatory</span>**

**<!-- Highlighting required fields -->**

**</td>**

**</tr>**

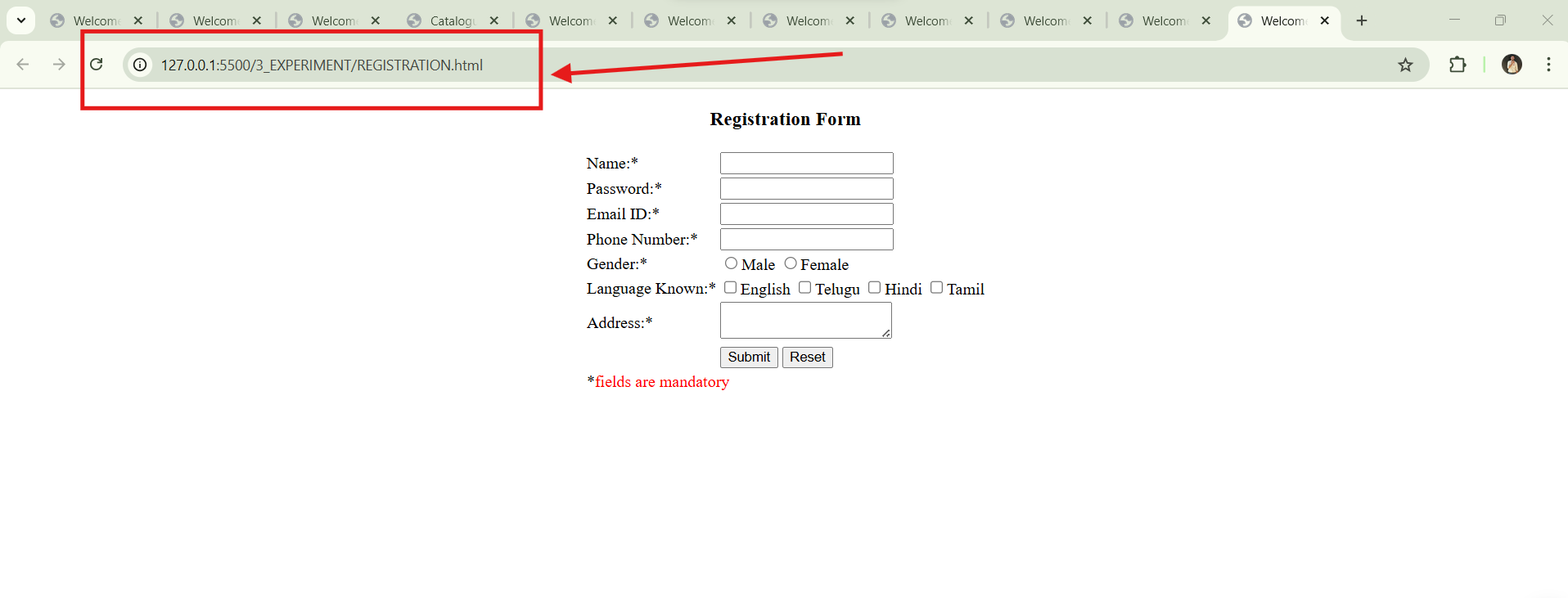
**</table>**

**</form>**

**</center>**

**</body>**

**</html>**



### 1.4.6 Userprofile.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<title>User Profile - Shopping Cart</title>**

**<style>**

**/\* Basic styling for layout and form \*/**

**body {**

**font-family: Arial, sans-serif;**

**background-color: #f4f4f4;**

**text-align: center;**

**}**

**.container {**

**width: 50%; /\* Container width is 50% of the viewport \*/**

**margin: auto; /\* Centers the container horizontally \*/**

**background: white;**

**padding: 20px;**

**box-shadow: 0px 0px 10px #ccc; /\* Adds a shadow effect \*/**

**border-radius: 10px; /\* Rounds the corners \*/**

**}**

**h1 {**

**color: #333; /\* Dark color for heading \*/**

**}**

**table {**

**width: 100%; /\* Table occupies full width \*/**

**}**

**td {**

**padding: 10px; /\* Adds spacing inside table cells \*/**

**}**

**input,**

**textarea {**

**width: 100%; /\* Makes inputs and textareas full width \*/**

**padding: 8px; /\* Adds padding for better readability \*/**

**margin-top: 5px;**

**}**

**button {**

**background-color: #28a745; /\* Green background color \*/**

**color: white; /\* White text color \*/**

**padding: 10px;**

**border: none;**

**cursor: pointer; /\* Changes cursor to pointer when hovered \*/**

**border-radius: 5px; /\* Rounds button corners \*/**

**}**

**button:hover {**

**background-color: #218838; /\* Darker green when hovered \*/**

**}**

**.profile-img {**

**width: 120px; /\* Profile picture size \*/**

**border-radius: 50%; /\* Circular profile image \*/**

**}**

**.nav {**

**margin-bottom: 20px; /\* Space between navigation and form \*/**

**}**

**.nav a {**

**margin: 10px;**

**text-decoration: none;**

**color: #007bff; /\* Blue color for links \*/**

**font-size: 16px;**

**}**

**</style>**

**</head>**

**<body>**

**<div class="container">**

**<h1>User Profile</h1>**

**<!-- Navigation Bar -->**

**<div class="nav">**

**<a href="CART.html">Shopping Cart</a> |**

**<a href="OrderConfirm.html">Order History</a> |**

**<a href="RIGHT.html">Home</a>**

**</div>**

**<!-- User Profile Form -->**

**<form id="profileForm">**

**<table>**

**<tr>**

**<td><strong>First Name:</strong></td>**

**<td><input type="text" id="firstName" value="Satish" /></td>**

**</tr>**

**<tr>**

**<td><strong>Last Name:</strong></td>**

**<td><input type="text" id="lastName" value="T" /></td>**

**</tr>**

**<tr>**

**<td><strong>Email ID:</strong></td>**

**<td><input type="email" id="email" value="Satish@gmail.com" /></td>**

**</tr>**

**<tr>**

**<td><strong>Phone Number:</strong></td>**

**<td><input type="tel" id="phone" value="9988665544" /></td>**

**</tr>**

**<tr>**

**<td><strong>Address:</strong></td>**

**<td>**

**<textarea id="address" rows="3">**

**Main Street, Good Area, Popular City, Best Country</textarea**

**>**

**</td>**

**</tr>**

**<tr>**

**<td><strong>Profile Picture:</strong></td>**

**<td>**

**<!-- Profile image display with default fallback -->**

**<img**

**src="images/user.PNG"**

**class="profile-img"**

**id="profilePic"**

**onerror="this.src='default.png'"**

**/>**

**<br />**

**<!-- File input for uploading a new profile image -->**

**<input type="file" id="profileImage" />**

**</td>**

**</tr>**

**<tr>**

**<td colspan="2">**

**<button type="button" onclick="updateProfile()">**

**Update Profile**

**</button>**

**</td>**

**</tr>**

**</table>**

**</form>**

**</div>**

**<script>**

**// Function to update user profile details**

**function updateProfile() {**

**let firstName = document.getElementById("firstName").value;**

**let lastName = document.getElementById("lastName").value;**

**let email = document.getElementById("email").value;**

**let phone = document.getElementById("phone").value;**

**let address = document.getElementById("address").value;**

**// Validation: Ensure all fields are filled before updating**

**if (!firstName || !lastName || !email || !phone || !address) {**

**alert("Please fill in all fields.");**

**return;**

**}**

**alert("Profile updated successfully!");**

**// In a real-world scenario, this data can be stored in a database or localStorage**

**}**

**// Function to preview profile image before upload**

**document**

**.getElementById("profileImage")**

**.addEventListener("change", function (event) {**

**let file = event.target.files[0]; // Get the selected file**

**if (file) {**

**let reader = new FileReader(); // Create FileReader object**

**reader.onload = function (e) {**

**document.getElementById("profilePic").src = e.target.result; // Update profile picture preview**

**};**

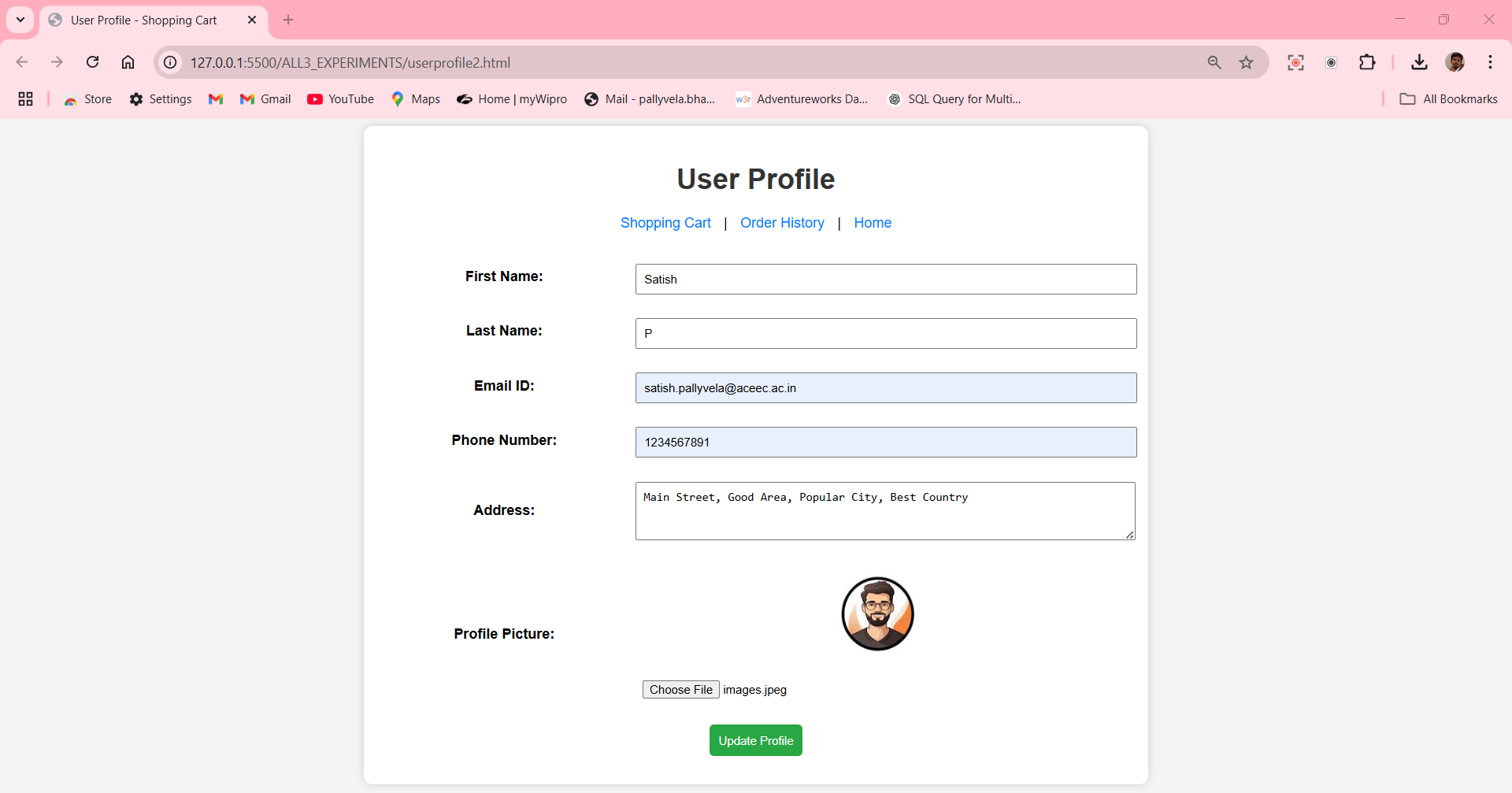
**reader.readAsDataURL(file); // Read the file as a Data URL**

**}**

**});**

**</script>**

**</body>**

**</html>**

### 1.4.7 Cse.html

**<html>**

**<head>**

**<title>Welcome to ACE e-Book's website</title>**

**<!-- The <title> tag sets the title of the webpage, which appears on the browser tab -->**

**</head>**

**<body>**

**<center>**

**<br />**

**<h2>Computer Science & Engineering Catalogue</h2>**

**<!-- The <h2> tag defines a heading for the page, indicating the catalog section -->**

**<table width="90%" align="center" bgcolor="#ffe1bf">**

**<!-- The <table> tag is used to create a table.**

**- width="90%" makes the table take 90% of the page width.**

**- align="center" centers the table on the page.**

**- bgcolor="#ffe1bf" sets the background color of the table to a light orange. -->**

**<tbody bgcolor="#ff6600">**

**<!-- The <tbody> tag groups table rows.**

**- bgcolor="#ff6600" sets the background color of this section (header) to orange. -->**

**<th width="25%">**

**<hr />**

**BOOK**

**<hr />**

**</th>**

**<th width="35%">**

**<hr />**

**DETAILS**

**<hr />**

**</th>**

**<th width="20%">**

**<hr />**

**PRICE**

**<hr />**

**</th>**

**<th width="20%">**

**<hr />**

**OPTION**

**<hr />**

**</th>**

**<!-- These <th> (table headers) define the column names: BOOK, DETAILS, PRICE, OPTION. -->**

**</tbody>**

**<tbody align="center">**

**<!-- This <tbody> tag groups book entries and aligns their content to the center. -->**

**<!-- First Book Entry: C - Programming -->**

**<tr>**

**<td align="center">**

**<img**

**src="books/Title\_C\_Programming.jpeg"**

**width="90"**

**height="120"**

**/>**

**<!-- The <img> tag displays an image of the book cover. -->**

**</td>**

**<td align="center">**

**Title: C - Programming<br />**

**Author: E. Balaguru Swamy<br />**

**Publisher: Tata McGraw-Hill**

**<!-- Displays book details with line breaks <br> for better formatting. -->**

**</td>**

**<td align="center">Rs. 1200/-</td>**

**<td align="center">**

**<input type="button" value="Add to cart" />**

**<!-- A button allowing users to add the book to their cart. -->**

**</td>**

**</tr>**

**<!-- Second Book Entry: Computer Networks -->**

**<tr>**

**<td align="center">**

**<img**

**src="books/COMPUTER\_NETWORKS\_B.jpg"**

**width="90"**

**height="120"**

**/>**

**</td>**

**<td align="center">**

**Title: Computer Networks<br />**

**Author: M. Ananda Rao<br />**

**Publisher: ARS Publications**

**</td>**

**<td align="center">Rs. 1000/-</td>**

**<td align="center">**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**<!-- Third Book Entry: Java Programming -->**

**<tr>**

**<td align="center">**

**<img src="books/Java\_SJS\_23\_24.jpeg" width="90" height="120" />**

**</td>**

**<td align="center">**

**Title: Java Programming<br />**

**Author: Jhanaki Ramaya<br />**

**Publisher: Matches Publications**

**</td>**

**<td align="center">Rs. 2300/-</td>**

**<td align="center">**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**<!-- Fourth Book Entry: Web Technologies -->**

**<tr>**

**<td align="center">**

**<img src="books/web\_technologies.jpg" width="90" height="120" />**

**</td>**

**<td align="center">**

**Title: Web Technologies<br />**

**Author: Uttam K. Roy<br />**

**Publisher: Oxford Publications**

**</td>**

**<td align="center">Rs. 3000/-</td>**

**<td align="center">**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

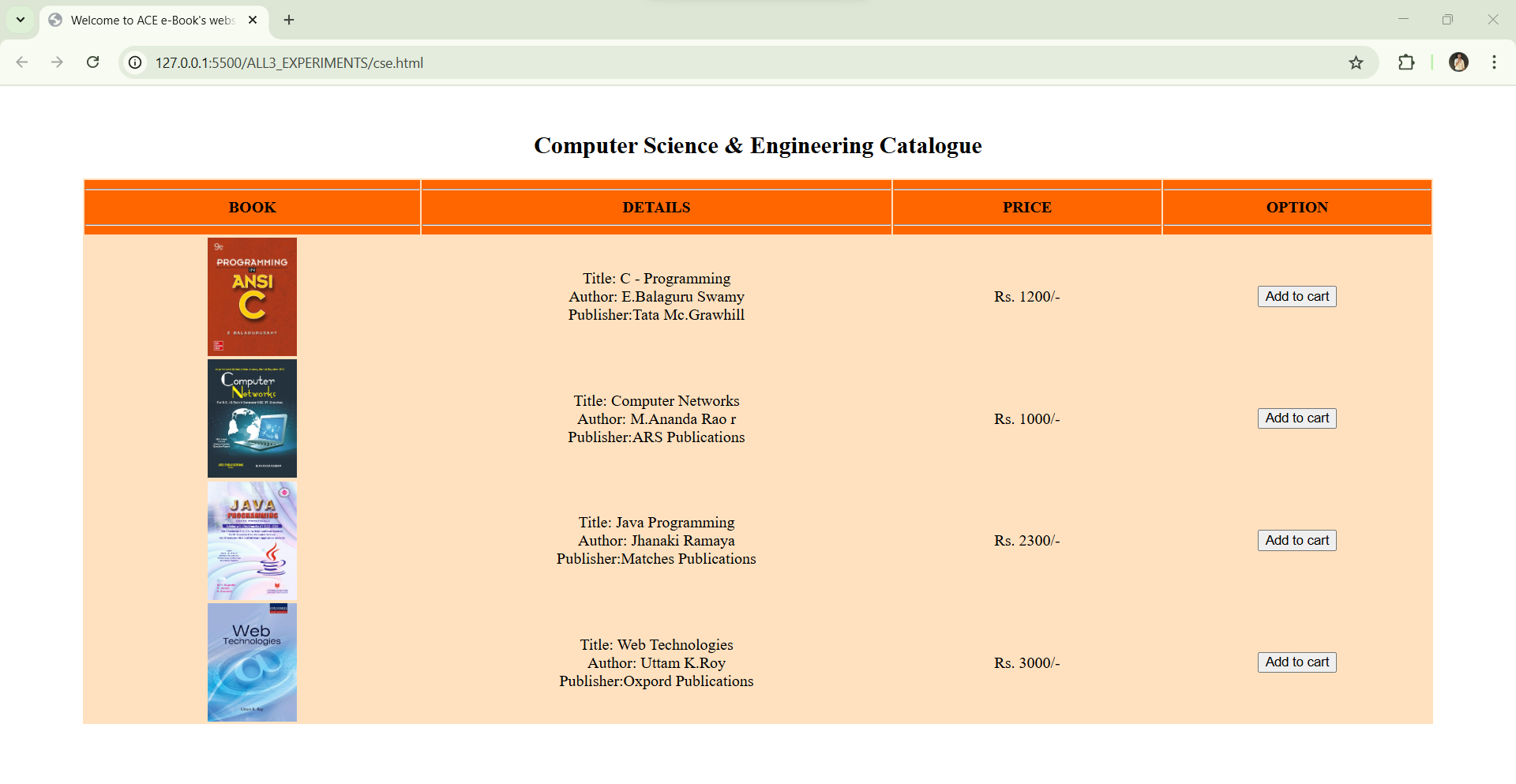
**</tbody>**

**</table>**

**</center>**

**</body>**

**</html>**



### 1.4.8 Ece.html

**<html>**

**<head>**

**<!-- Title of the webpage displayed on the browser tab -->**

**<title>Welcome to ACE e-Book's website</title>**

**</head>**

**<body>**

**<center>**

**<br />**

**<!-- Main heading for the Electronics & Communication Engineering book catalog -->**

**<h2>Electronics & Communication Engineering Catalogue</h2>**

**<!-- Table to display books, details, price, and purchase option -->**

**<table width="90%" align="center" bgcolor="#c4c4ff">**

**<!-- Table header section with blue background -->**

**<tbody bgcolor="#5b5bff">**

**<!-- First column: Book cover images -->**

**<th width="25%">**

**<hr />**

**BOOK**

**<hr />**

**</th>**

**<!-- Second column: Book details like title, author, publisher -->**

**<th width="35%">**

**<hr />**

**DETAILS**

**<hr />**

**</th>**

**<!-- Third column: Price of the book -->**

**<th width="20%">**

**<hr />**

**PRICE**

**<hr />**

**</th>**

**<!-- Fourth column: Button to add the book to cart -->**

**<th width="20%">**

**<hr />**

**OPTION**

**<hr />**

**</th>**

**</tbody>**

**<!-- Table body section containing book entries -->**

**<tbody align="center">**

**<tr>**

**<!-- First book entry -->**

**<td align="center">**

**<!-- Displays book cover image -->**

**<img src="books/Title\_Analog\_Digital\_Communications\_ECE.jpg" width="90" height="120" />**

**</td>**

**<td align="center">**

**<!-- Displays book title, author, and publisher -->**

**Title: Analog Digital Communications <br />**

**Author: Meru Babu <br />**

**Publisher: Schaum Outlines**

**</td>**

**<td align="center">Rs. 1200/-</td>**

**<td align="center">**

**<!-- Button for adding the book to cart -->**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**<tr>**

**<!-- Second book entry -->**

**<td align="center">**

**<!-- Displays book cover image -->**

**<img src="books/Title\_Anntena\_Theory\_ECE.webp" width="90" height="120" />**

**</td>**

**<td align="center">**

**<!-- Displays book title, author, and publisher -->**

**Title: Antenna Theory <br />**

**Author: M. Dileep Kumar <br />**

**Publisher: BRS Publications**

**</td>**

**<td align="center">Rs. 800/-</td>**

**<td align="center">**

**<!-- Button for adding the book to cart -->**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**<tr>**

**<!-- Third book entry -->**

**<td align="center">**

**<!-- Displays book cover image -->**

**<img src="books/Title\_Gate\_Study\_Metireal\_ECE.webp" width="90" height="120" />**

**</td>**

**<td align="center">**

**<!-- Displays book title, author, and publisher -->**

**Title: GATE Study Material <br />**

**Author: Bill Joy <br />**

**Publisher: Pearson Publications**

**</td>**

**<td align="center">Rs. 8200/-</td>**

**<td align="center">**

**<!-- Button for adding the book to cart -->**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**</tbody>**

**</table>**

**</center>**

**</body>**

**</html>**



### 1.4.9 It.html

**<html>**

**<head>**

**<!-- Title of the webpage displayed on the browser tab -->**

**<title>Welcome to ACE e-Book's website</title>**

**</head>**

**<body>**

**<center>**

**<br />**

**<!-- Main heading of the page -->**

**<h2>Information Technologies Catalogue</h2>**

**<!-- Creating a table to display book details -->**

**<table width="90%" align="center" bgcolor="#bfffbf">**

**<!-- Table header row with different columns -->**

**<tbody bgcolor="#03f515">**

**<th width="25%">**

**<hr />**

**BOOK**

**<hr />**

**</th>**

**<th width="35%">**

**<hr />**

**DETAILS**

**<hr />**

**</th>**

**<th width="20%">**

**<hr />**

**PRICE**

**<hr />**

**</th>**

**<th width="20%">**

**<hr />**

**OPTION**

**<hr />**

**</th>**

**</tbody>**

**<!-- Table body containing book entries -->**

**<tbody align="center">**

**<tr>**

**<!-- First book entry -->**

**<td align="center">**

**<!-- Book image -->**

**<img src="books/IT\_Let\_Us\_C.jpg" width="90" height="120" />**

**</td>**

**<td align="center">**

**<!-- Book details: Title, Author, Publisher -->**

**Title: Let Us C <br />**

**Author: Yashwant Kanetkar <br />**

**Publisher: Tata McGraw-Hill**

**</td>**

**<td align="center">Rs. 4200/-</td>**

**<td align="center">**

**<!-- Add to Cart button -->**

**<input type="button" value="Add to cart" />**

**</td>**

**</tr>**

**<tr>**

**<!-- Second book entry -->**

**<td align="center">**

**<img src="books/PYTHON\_programmingit.jpg" width="90" height="120" />**

**</td>**

**<td align="center">**

**Title: Python Programming<br />**

**Author: M. Kiran Kumar <br />**

**Publisher: ARS Publications**

**</td>**

**<td align="center">Rs. 1100/-</td>**

**<td align="center"><input type="button" value="Add to cart" /></td>**

**</tr>**

**<tr>**

**<!-- Third book entry -->**

**<td align="center">**

**<img src="books/web\_technologies.jpg" width="90" height="120" />**

**</td>**

**<td align="center">**

**Title: Web Programming<br />**

**Author: Uttam K. Roy <br />**

**Publisher: Oxford Publications**

**</td>**

**<td align="center">Rs. 6200/-</td>**

**<td align="center"><input type="button" value="Add to cart" /></td>**

**</tr>**

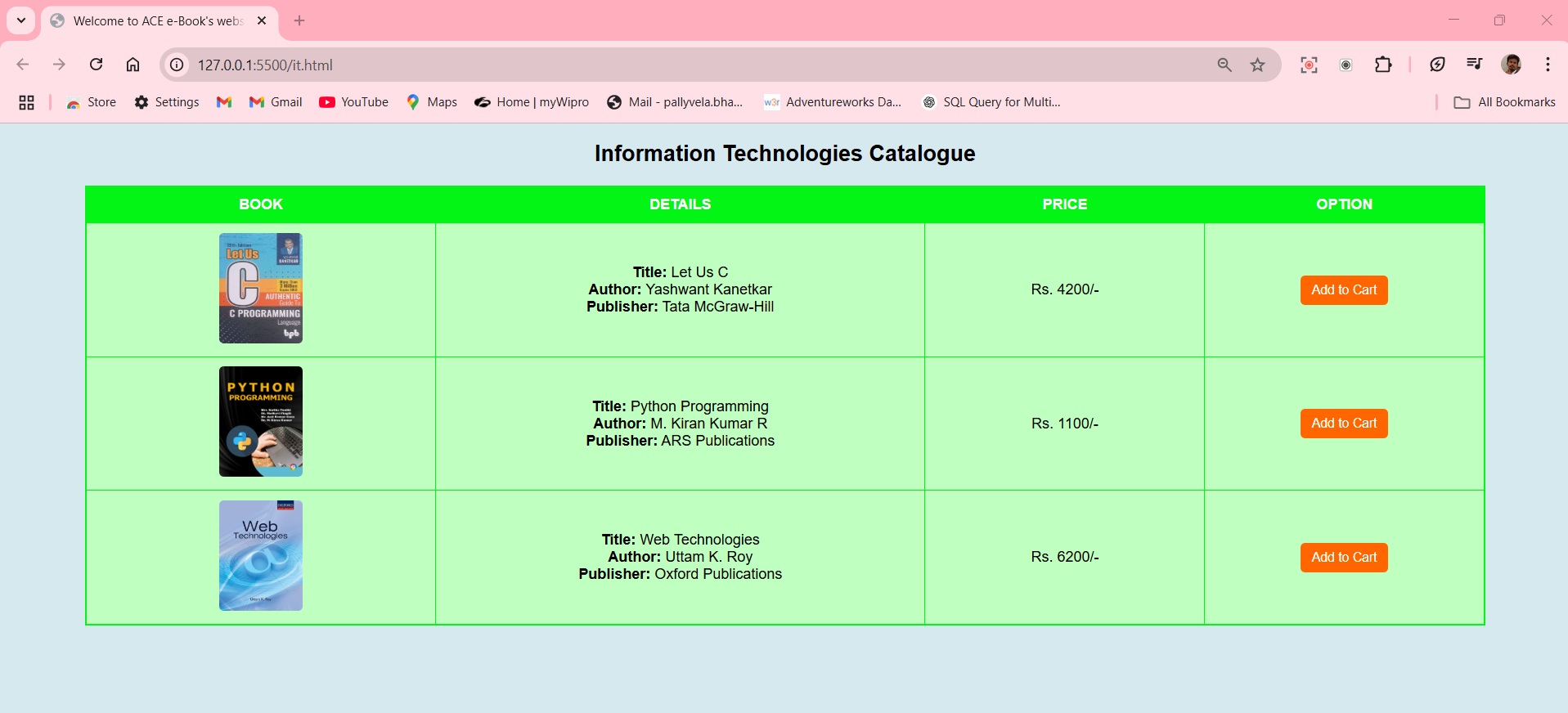
**</tbody>**

**</table>**

**</center>**

**</body>**

**</html>**



### 1.4.10 Cart.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>ACE e-Book Store - Cart</title>**

**<!-- Favicon Fix -->**

**<link rel="icon" href="ace-icon.png" type="image/png">**

**<!-- FontAwesome for Cart Icon -->**

**<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">**

**<style>**

**/\* General Styles \*/**

**body {**

**font-family: Arial, sans-serif;**

**background-color: #f4f4f4;**

**text-align: center;**

**margin: 0;**

**padding: 0;**

**}**

**/\* Navigation Bar Styling \*/**

**.navbar {**

**background-color: #333;**

**overflow: hidden;**

**padding: 10px 20px;**

**display: flex;**

**justify-content: space-between; /\* Moves cart to the right \*/**

**align-items: center;**

**}**

**.navbar-links {**

**display: flex;**

**gap: 15px;**

**}**

**.navbar a {**

**color: white;**

**text-decoration: none;**

**padding: 14px 20px;**

**display: inline-block;**

**}**

**.navbar a:hover {**

**background-color: rgb(242, 34, 34);**

**}**

**/\* Cart Icon Styling \*/**

**.cart-container {**

**position: relative;**

**display: flex;**

**align-items: center;**

**}**

**.cart-icon {**

**color: white;**

**font-size: 22px;**

**margin-left: 5px;**

**}**

**#cart-count {**

**background-color: red;**

**color: white;**

**font-size: 12px;**

**border-radius: 50%;**

**padding: 4px 8px;**

**position: absolute;**

**top: -5px;**

**right: -10px;**

**}**

**/\* Table Styling \*/**

**table {**

**width: 90%;**

**background-color: #ffe479;**

**margin-top: 20px;**

**border-collapse: collapse;**

**}**

**th {**

**background-color: #ff7c5b;**

**padding: 10px;**

**color: white;**

**}**

**td {**

**padding: 10px;**

**}**

**hr {**

**border: 1px solid #996600;**

**}**

**.total {**

**color: #996600;**

**font-weight: bold;**

**}**

**/\* Pay Button Styling \*/**

**.pay-button {**

**background-color: #ff7c5b;**

**color: white;**

**border: none;**

**padding: 10px 15px;**

**cursor: pointer;**

**border-radius: 5px;**

**}**

**.pay-button:hover {**

**background-color: #ff5722;**

**}**

**</style>**

**</head>**

**<body>**

**<!-- Navigation Bar (Cart moved to the right with an icon) -->**

**<div class="navbar">**

**<div class="navbar-links">**

**<a href="index.html">Home</a>**

**<a href="catalog.html">Catalog</a>**

**<a href="login.html">Login</a>**

**</div>**

**<div class="cart-container">**

**<a href="cart.html">**

**<i class="fas fa-shopping-cart cart-icon"></i>**

**<span id="cart-count">0</span>**

**</a>**

**</div>**

**</div>**

**<!-- Shopping Cart Table -->**

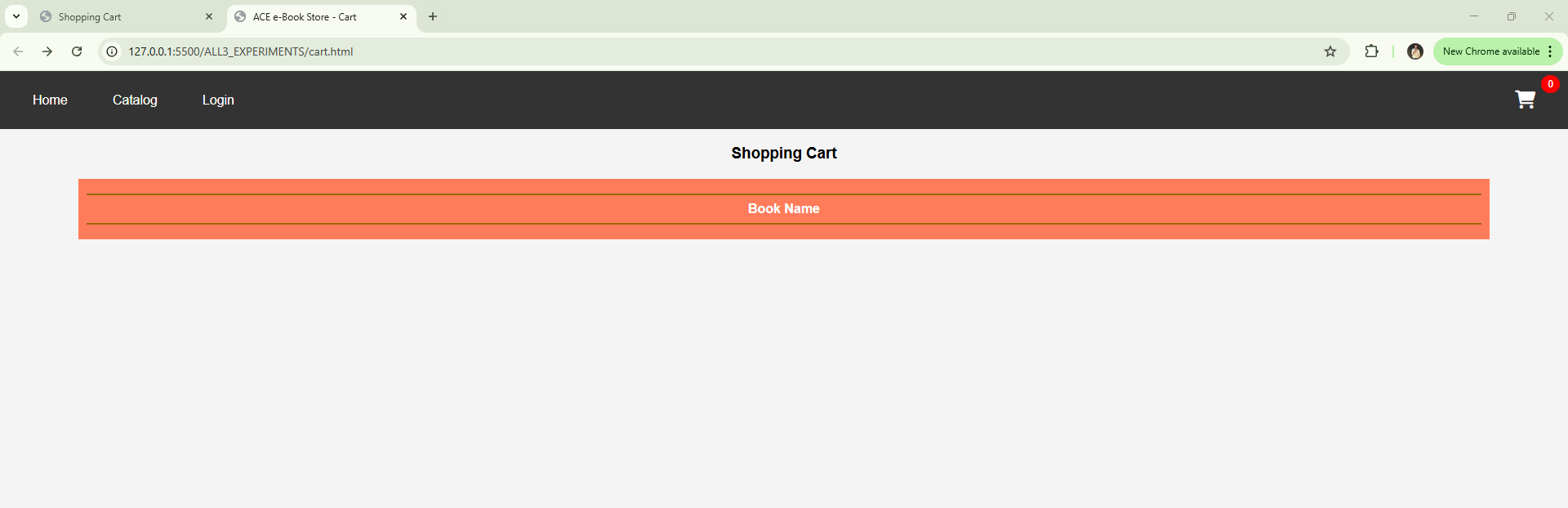
**<h3>Shopping Cart</h3>**

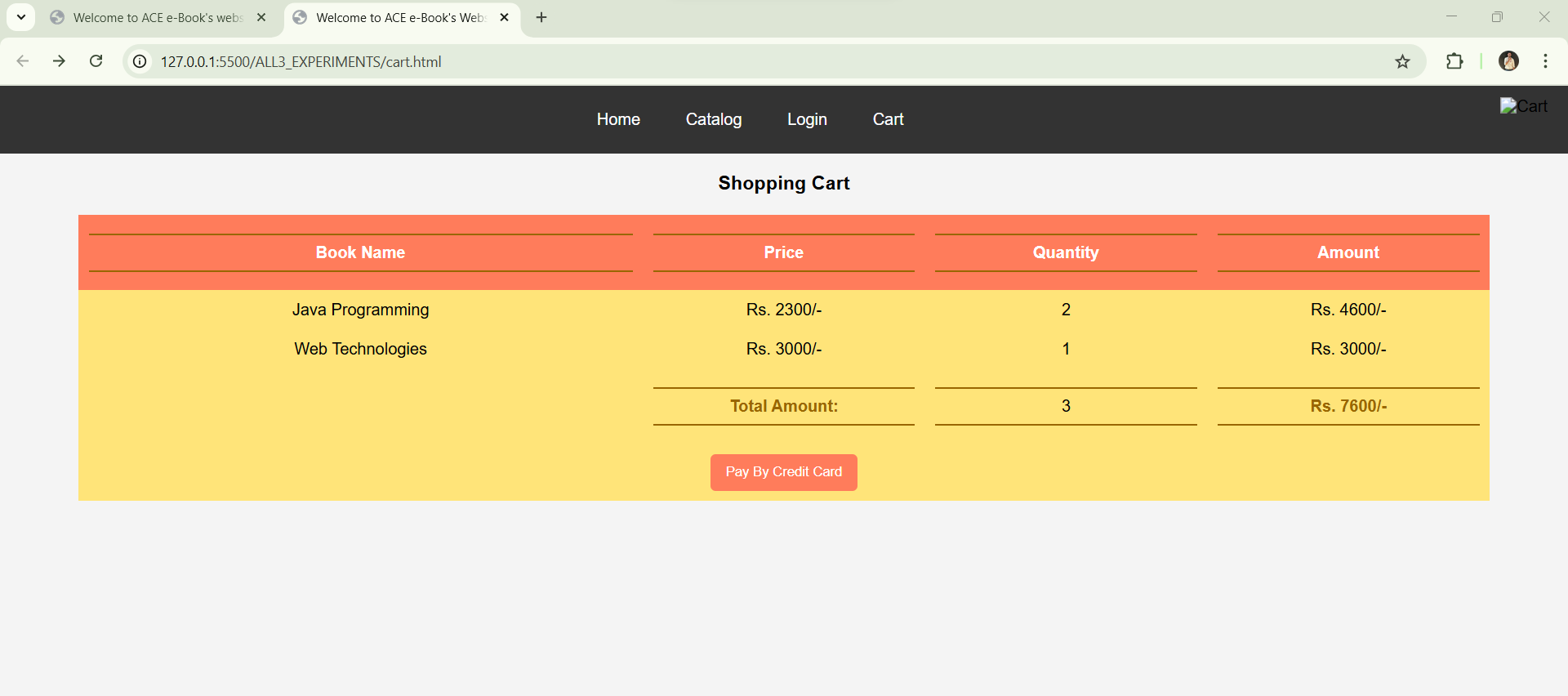
**<table align="center">**

**<tbody>**

**<tr>**

**<th width="40%"><hr>Book Name<hr></th>**





### 1.4.10 catalogue.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Catalogue</title>**

**<link rel="stylesheet" href="styles.css">**

**</head>**

**<body>**

**<header>**

**<h1>Product Catalogue</h1>**

**</header>**

**<section class="catalogue">**

**<!-- CSE Books -->**

**<div class="category">**

**<h2>CSE Books</h2>**

**<div class="product">**

**<img src="books/COMPUTER\_NETWORKS\_B.jpg" alt="Computer Networks">**

**<h2>Computer Networks</h2>**

**<p>$10.00</p>**

**<button onclick="addToCart('Computer Networks', 10)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_C\_Programming.jpeg" alt="C Programming">**

**<h2>C Programming</h2>**

**<p>$12.00</p>**

**<button onclick="addToCart('C Programming', 12)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_Artificial\_Intelligence\_A\_Modern\_Approach.jpg" alt="Artificial Intelligence">**

**<h2>Artificial Intelligence</h2>**

**<p>$20.00</p>**

**<button onclick="addToCart('Artificial Intelligence', 20)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_Operating\_System\_Concepts.jpeg" alt="Operating System Concepts">**

**<h2>Operating System Concepts</h2>**

**<p>$18.00</p>**

**<button onclick="addToCart('Operating System Concepts', 18)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/web\_technologies.jpg" alt="Web Technologies">**

**<h2>Web Technologies</h2>**

**<p>$15.00</p>**

**<button onclick="addToCart('Web Technologies', 15)">Add to Cart</button>**

**</div>**

**</div>**

**<!-- IT Books -->**

**<div class="category">**

**<h2>IT Books</h2>**

**<div class="product">**

**<img src="books/IT\_Let\_Us\_C.jpg" alt="Let Us C">**

**<h2>Let Us C</h2>**

**<p>$15.00</p>**

**<button onclick="addToCart('Let Us C', 15)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Java\_SJS\_23\_24.jpeg" alt="Java Programming">**

**<h2>Java Programming</h2>**

**<p>$20.00</p>**

**<button onclick="addToCart('Java Programming', 20)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/PYTHON\_programmingit.jpg" alt="Python Programming">**

**<h2>Python Programming</h2>**

**<p>$22.00</p>**

**<button onclick="addToCart('Python Programming', 22)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_ Data\_Structures\_and\_Algorithms.webp" alt="Data Structures & Algorithms">**

**<h2>Data Structures & Algorithms</h2>**

**<p>$25.00</p>**

**<button onclick="addToCart('Data Structures & Algorithms', 25)">Add to Cart</button>**

**</div>**

**</div>**

**<!-- ECE Books -->**

**<div class="category">**

**<h2>ECE Books</h2>**

**<div class="product">**

**<img src="books/Title\_Analog\_Digital\_Communications\_ECE.jpg" alt="Analog & Digital Communications">**

**<h2>Analog & Digital Communications</h2>**

**<p>$18.00</p>**

**<button onclick="addToCart('Analog & Digital Communications', 18)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_Anntena\_Theory\_ECE.webp" alt="Antenna Theory">**

**<h2>Antenna Theory</h2>**

**<p>$14.00</p>**

**<button onclick="addToCart('Antenna Theory', 14)">Add to Cart</button>**

**</div>**

**<div class="product">**

**<img src="books/Title\_Gate\_Study\_Metireal\_ECE.webp" alt="GATE Study Material">**

**<h2>GATE Study Material</h2>**

**<p>$30.00</p>**

**<button onclick="addToCart('GATE Study Material', 30)">Add to Cart</button>**

**</div>**

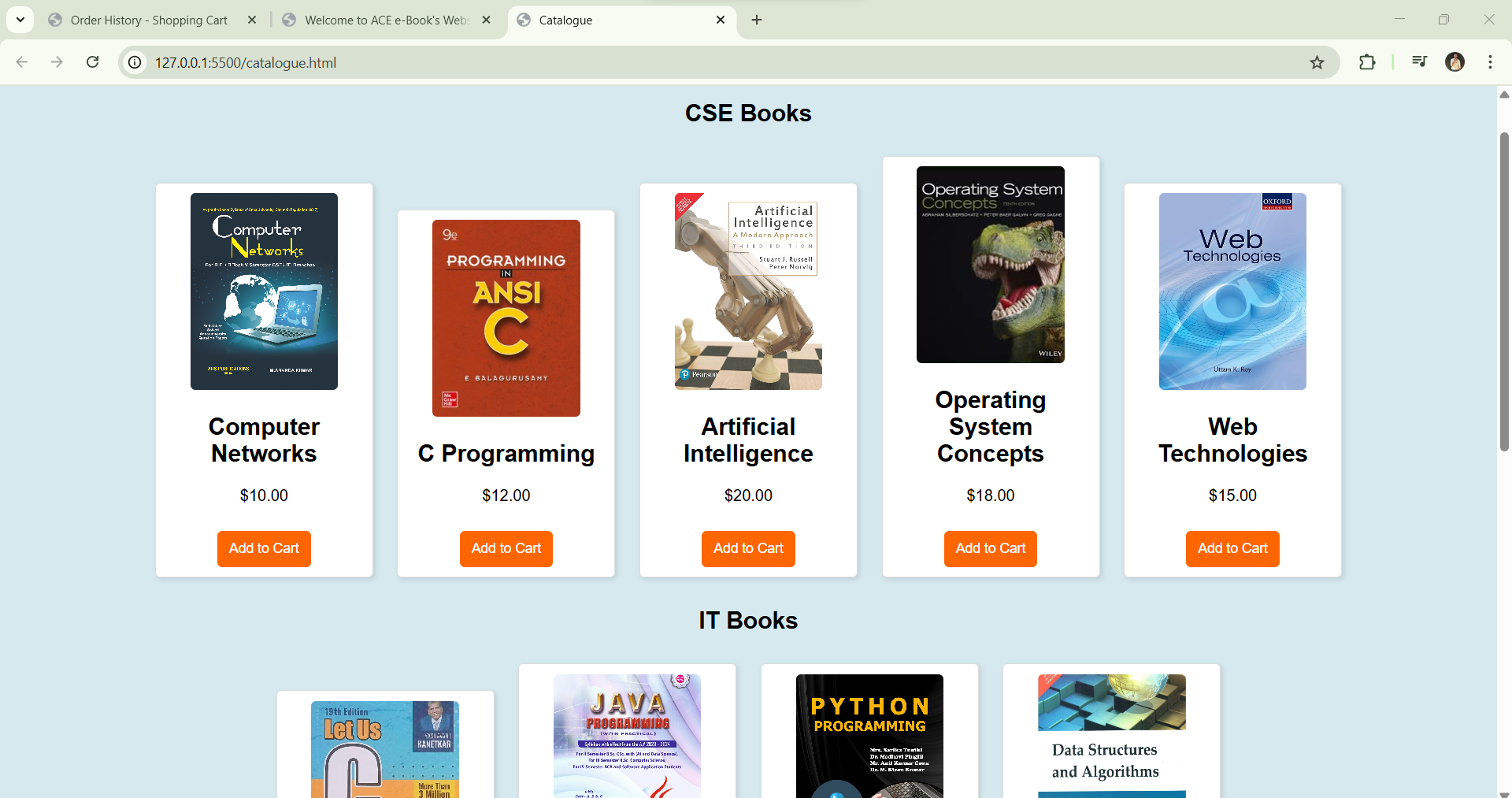
**</div>**

**</section>**

**<script src="cart.js"></script>**

**</body>**

**</html>**



### 1.4.11 Apply CSS3 Features for Responsiveness : CSS Code (style.css):

To ensure responsiveness, CSS3 Flexbox and Grid should be used.

**/\* Reset default styles \*/**

**\* {**

**margin: 0;**

**padding: 0;**

**box-sizing: border-box;**

**}**

**/\* Flexbox Layout for Navbar \*/**

**.navbar {**

**display: flex;**

**justify-content: space-between;**

**background: #333;**

**padding: 15px;**

**}**

**.navbar a {**

**color: white;**

**text-decoration: none;**

**padding: 10px;**

**} /\* Grid Layout for Catalog Page \*/**

**.catalog-grid {**

**display: grid;**

**grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));**

**gap: 20px;**

**padding: 20px;**

**}**

**.book-item {**

**border: 1px solid #ddd;**

**padding: 15px;**

**text-align: center;**

**}**

# Program 2: Make the above web application responsive web application using Bootstrap framework

### 2.1 AIM: Make the above web application responsive web application using Bootstrap framework

### 2.2 File Structure:

| 2.2.1 Folder | 2.2.2 Subfolder / File |
| --- | --- |
| ShoppingCartApp |  |
| public |  |
| css | paybycreditcard.css |
| html | paybycreditcard.html |

### 2.3 Code Implementation

### 2.3.1 Paybycreditcard.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8" />  
 <!-- Ensures proper text encoding to support international characters -->**

**<meta name="viewport" content="width=device-width, initial-scale=1.0" />**

**<!-- Makes the page responsive and compatible with different screen sizes -->**

**<title>Payment By Credit Card</title>**  
 **<!-- Linking external CSS file →  
 <link rel="stylesheet" href="../css/paybycreditcard.css" />  
 </head>  
<body>**

**<!-- Payment form container -->**

**<form class="payment-form" action="orderconfirm.html" method="post">**

**<h1>Payment By Credit Card</h1>**

**<!-- Card Number Input -->**

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

**<label for="card-number">Card Number:</label>**

**<input**

**type="text"**

**id="card-number"**

**name="card-number"**

**placeholder="xxxx-xxxx-xxxx-xxxx"**

**required**

**pattern="\d{4}-\d{4}-\d{4}-\d{4}"**

**title="Enter a valid 16-digit card number (xxxx-xxxx-xxxx-xxxx)"**

**/>**

**<!-- Ensures users enter a 16-digit card number in the correct format -->**

**</div>**

**<!-- Expiry Date Input -->**

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

**<label for="exp-date">Exp Date:</label>**

**<input**

**type="text"**

**id="exp-date"**

**name="exp-date"**

**placeholder="MM/YY"**

**required**

**pattern="(0[1-9]|1[0-2])/[0-9]{2}"**

**title="Enter a valid expiry date (MM/YY)"**

**/>**

**<!-- Ensures the date is in MM/YY format and validates months (01-12) -->**

**</div>**

**<!-- CVV Input -->**

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

**<label for="cvv">CVV:</label>**

**<input**

**type="password"**

**id="cvv"**

**name="cvv"**

**placeholder="xxx"**

**required**

**pattern="\d{3}"**

**title="Enter a valid 3-digit CVV"**

**/>**

**<!-- Uses 'password' type for security, expects a 3-digit CVV code -->**

**</div>**

**<!-- Cardholder's Name Input -->**

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

**<label for="card-name">Name on the Card:</label>**

**<input**

**type="text"**

**id="card-name"**

**name="card-name"**

**placeholder="Name of the card"**

**required**

**/>**

**<!-- Requires the user to enter the cardholder's name** →

**</div>**

**<!-- Payment & Cancel Buttons -->**

**<div class="button-group">**

**<input type="submit" value="Pay" />**

**<input type="reset" value="Cancel" />**

**<!-- Submit button processes the payment, Reset button clears input fields-->**

**</div>**

**</form>**

**</body>**

**</html>**

### 2.3.2 Paybycreditcard.css

**/\* General page styling \*/**

**body {**

**font-family: Arial, sans-serif; /\* Sets a clean, readable font \*/**

**display: flex; /\* Uses flexbox for centering \*/**

**justify-content: center; /\* Centers the form horizontally \*/**

**align-items: center; /\* Centers the form vertically \*/**

**height: 100vh; /\* Makes the body take the full viewport height \*/**

**background-color: #f4f4f4; /\* Light grey background for a soft look \*/**

**}**

**/\* Payment form container \*/**

**.payment-form {**

**background: white; /\* Sets a white background for the form \*/**

**padding: 20px; /\* Adds padding inside the form for spacing \*/**

**border-radius: 10px; /\* Rounds the corners of the form \*/**

**box-shadow: 0px 0px 10px 0px rgba(0, 0, 0, 0.1); /\* Adds a light shadow for a subtle 3D effect \*/**

**width: 350px; /\* Sets the width of the form \*/**

**}**

**/\* Form title \*/**

**.payment-form h1 {**

**text-align: center; /\* Centers the title \*/**

**}**

**/\* Form groups for inputs \*/**

**.form-group {**

**margin-bottom: 15px; /\* Adds spacing between input fields \*/**

**}**

**/\* Labels for inputs \*/**

**.form-group label {**

**display: block; /\* Makes the label appear on top of the input \*/**

**font-weight: bold; /\* Makes the text bold \*/**

**}**

**/\* Styling input fields \*/**

**.form-group input {**

**width: 100%; /\* Makes the input field take full width of the container \*/**

**padding: 8px; /\* Adds padding for better readability \*/**

**border: 1px solid #ccc; /\* Adds a light border \*/**

**border-radius: 5px; /\* Rounds the edges of input fields \*/**

**}**

**/\* Button container \*/**

**.button-group {**

**text-align: center; /\* Aligns buttons to the center \*/**

**}**

**/\* Styling buttons \*/**

**.button-group input {**

**padding: 10px 20px; /\* Adds padding inside buttons \*/**

**margin: 5px; /\* Adds spacing between buttons \*/**

**border: none; /\* Removes default button border \*/**

**cursor: pointer; /\* Changes cursor to pointer on hover \*/**

**border-radius: 5px; /\* Rounds the button edges \*/**

**}**

**/\* Submit (Pay) button \*/**

**.button-group input[type="submit"] {**

**background-color: #28a745; /\* Green background for submit \*/**

**color: white; /\* White text for contrast \*/**

**}**

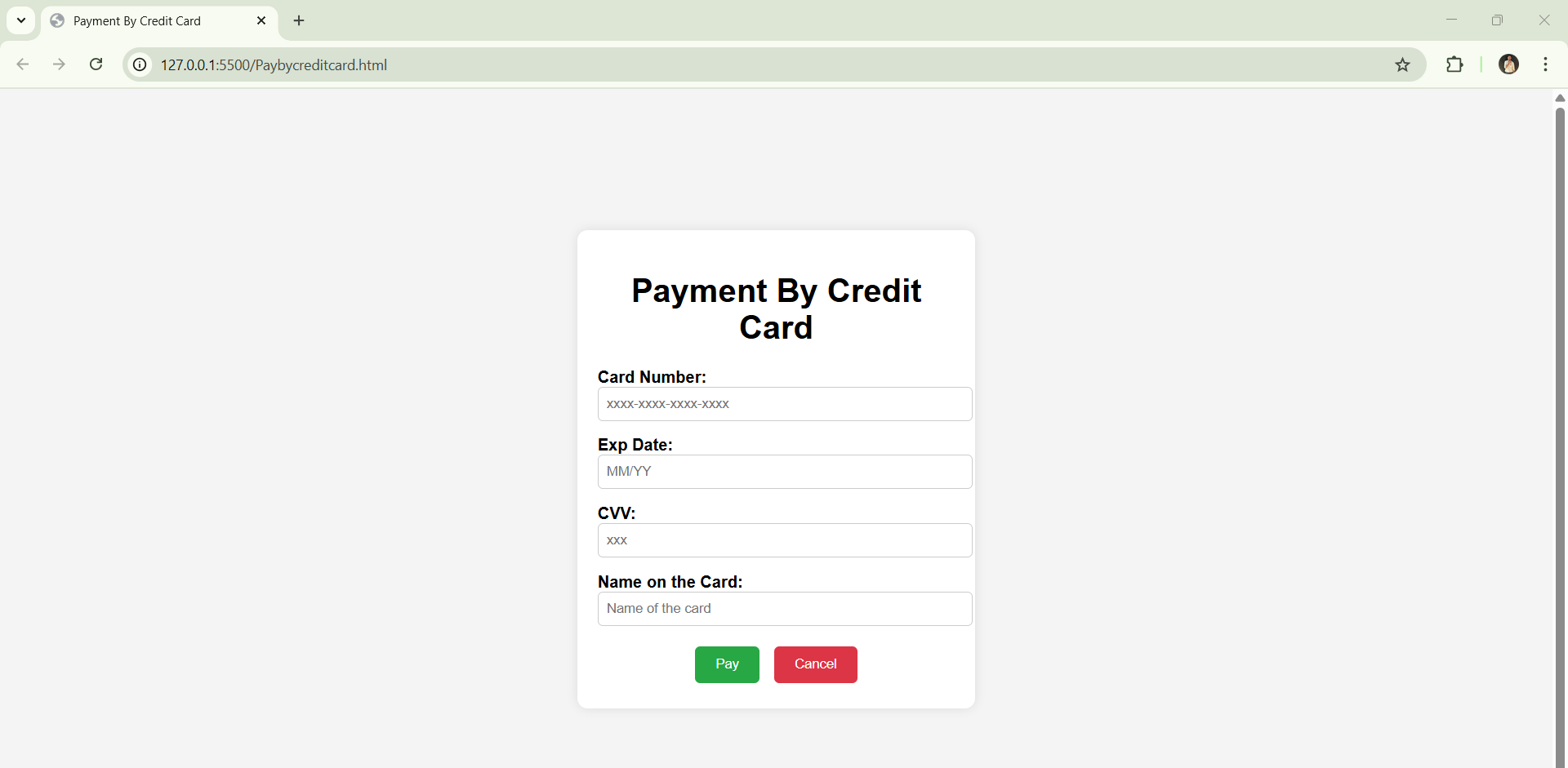
**/\* Reset (Cancel) button \*/**

**.button-group input[type="reset"] {**

**background-color: #dc3545; /\* Red background for reset \*/**

**color: white; /\* White text for contrast \*/**

**}**



### 2.3.3 Orderconform.html

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Order History - Shopping Cart</title>**

**<style>**

**/\* Basic styling for better layout \*/**

**body {**

**font-family: Arial, sans-serif;**

**background-color: #f4f4f4;**

**text-align: center;**

**}**

**.container {**

**width: 60%;**

**margin: auto;**

**background: white;**

**padding: 20px;**

**box-shadow: 0px 0px 10px #ccc;**

**border-radius: 10px;**

**}**

**h1 {**

**color: #333;**

**}**

**.nav {**

**margin-bottom: 20px;**

**}**

**.nav a {**

**margin: 10px;**

**text-decoration: none;**

**color: #007bff;**

**font-size: 16px;**

**}**

**table {**

**width: 100%;**

**border-collapse: collapse;**

**margin-top: 20px;**

**}**

**th, td {**

**border: 1px solid #ddd;**

**padding: 10px;**

**text-align: center;**

**}**

**th {**

**background-color: #28a745;**

**color: white;**

**}**

**.empty {**

**color: red;**

**font-size: 18px;**

**}**

**</style>**

**</head>**

**<body>**

**<div class="container">**

**<h1>Order History</h1>**

**<!-- Navigation Bar -->**

**<div class="nav">**

**<a href="cart2.html">Shopping Cart</a> |**

**<a href="INDEX2.html">Home</a> |**

**<a href="userprofile1.html">Profile</a>**

**</div>**

**<!-- Order History Table -->**

**<table>**

**<thead>**

**<tr>**

**<th>Order ID</th>**

**<th>Items</th>**

**<th>Total Amount</th>**

**<th>Date</th>**

**</tr>**

**</thead>**

**<tbody id="orderHistory">**

**<!-- Orders will be loaded here dynamically -->**

**</tbody>**

**</table>**

**<p id="emptyMsg" class="empty">No orders found.</p>**

**</div>**

**<script>**

**// Function to load order history from localStorage**

**function loadOrderHistory() {**

**let orders = JSON.parse(localStorage.getItem("orderHistory")) || []; // Get stored orders or empty array**

**let orderTable = document.getElementById("orderHistory");**

**let emptyMessage = document.getElementById("emptyMsg");**

**if (orders.length === 0) {**

**// Show "No orders found" message if there are no orders**

**emptyMessage.style.display = "block";**

**} else {**

**emptyMessage.style.display = "none"; // Hide empty message**

**orders.forEach(order => {**

**let row = `<tr>**

**<td>${order.id}</td>**

**<td>${order.items}</td>**

**<td>Rs. ${order.total}</td>**

**<td>${order.date}</td>**

**</tr>`;**

**orderTable.innerHTML += row;**

**});**

**}**

**}**

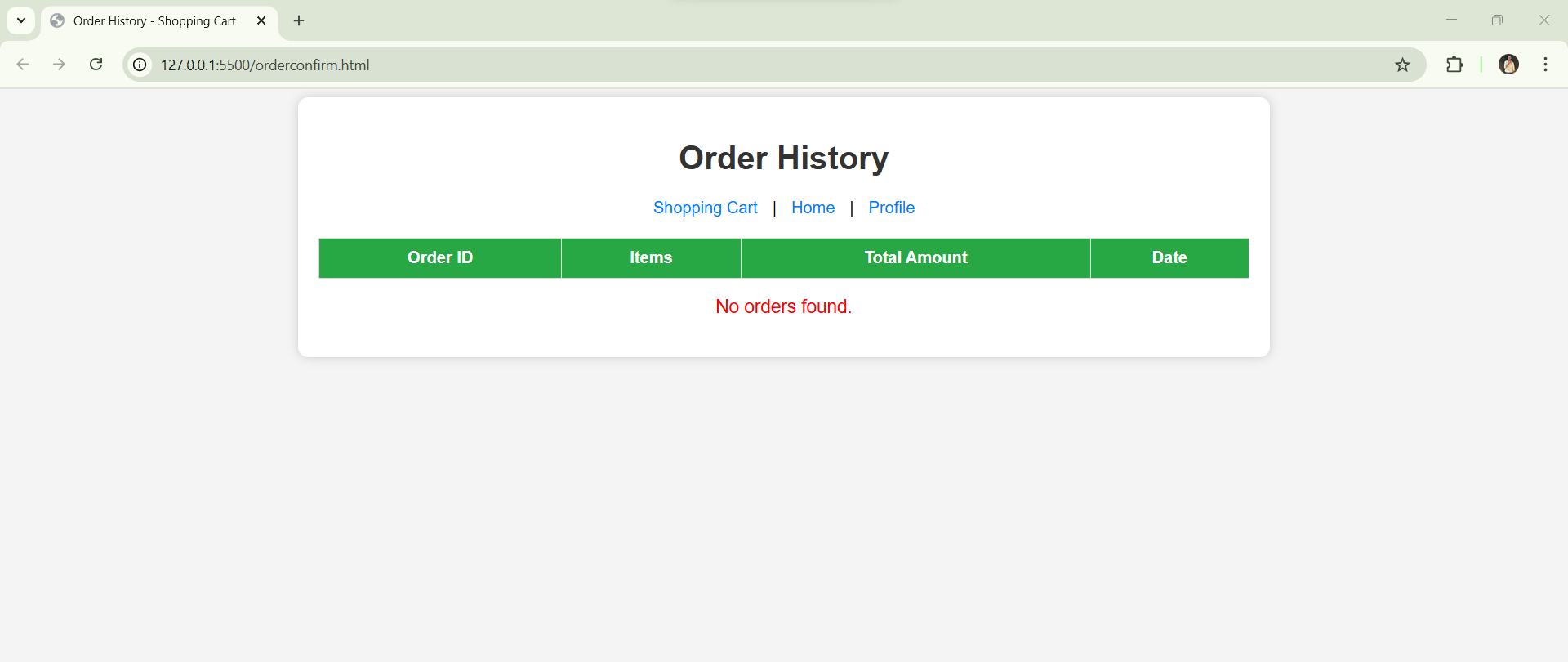
**// Load order history when the page loads**

**window.onload = loadOrderHistory;**

**</script>**

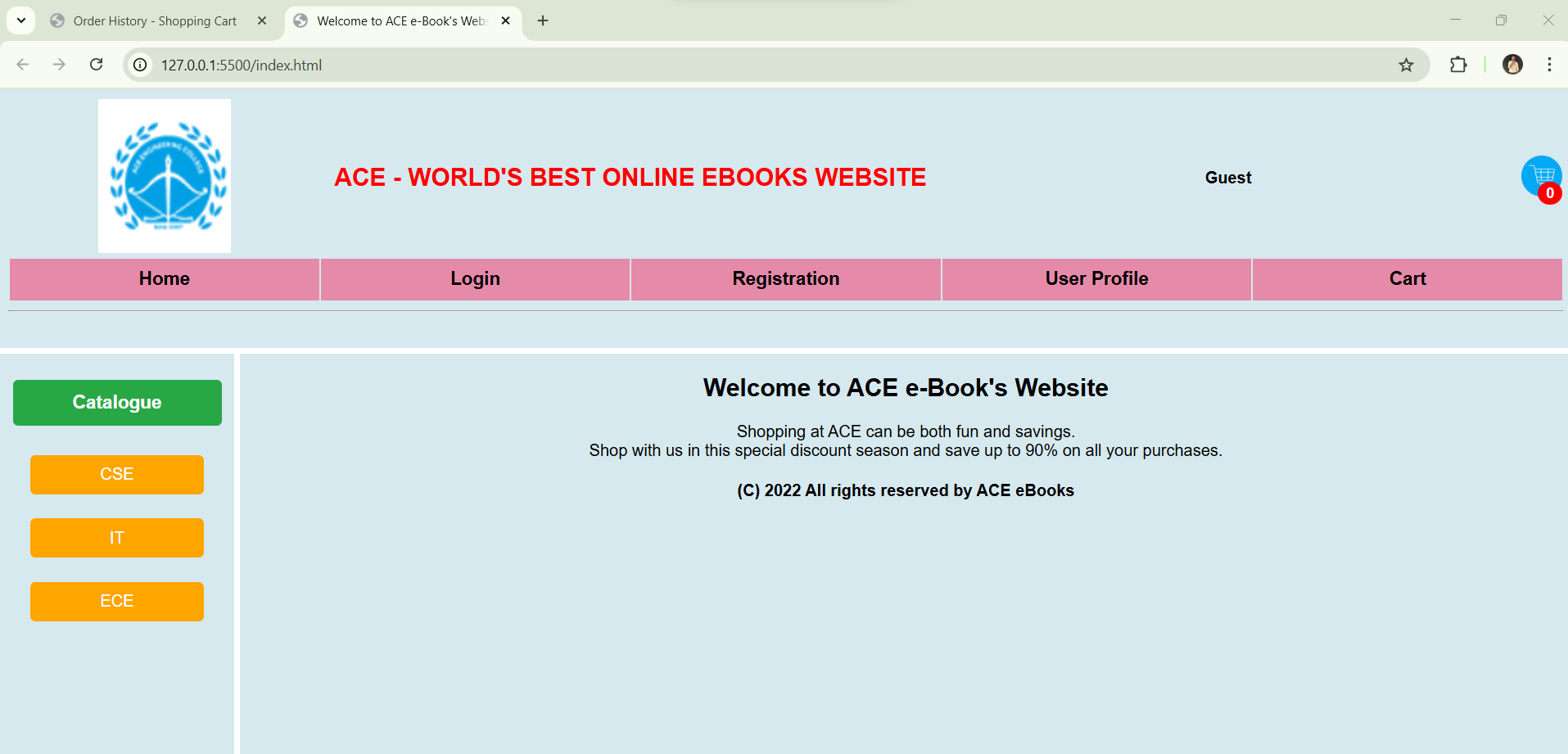
**</body>**

**</html>**



### 2.4 Output :

### 2.4.1 Home Page (index.html) – Displaying the homepage layout

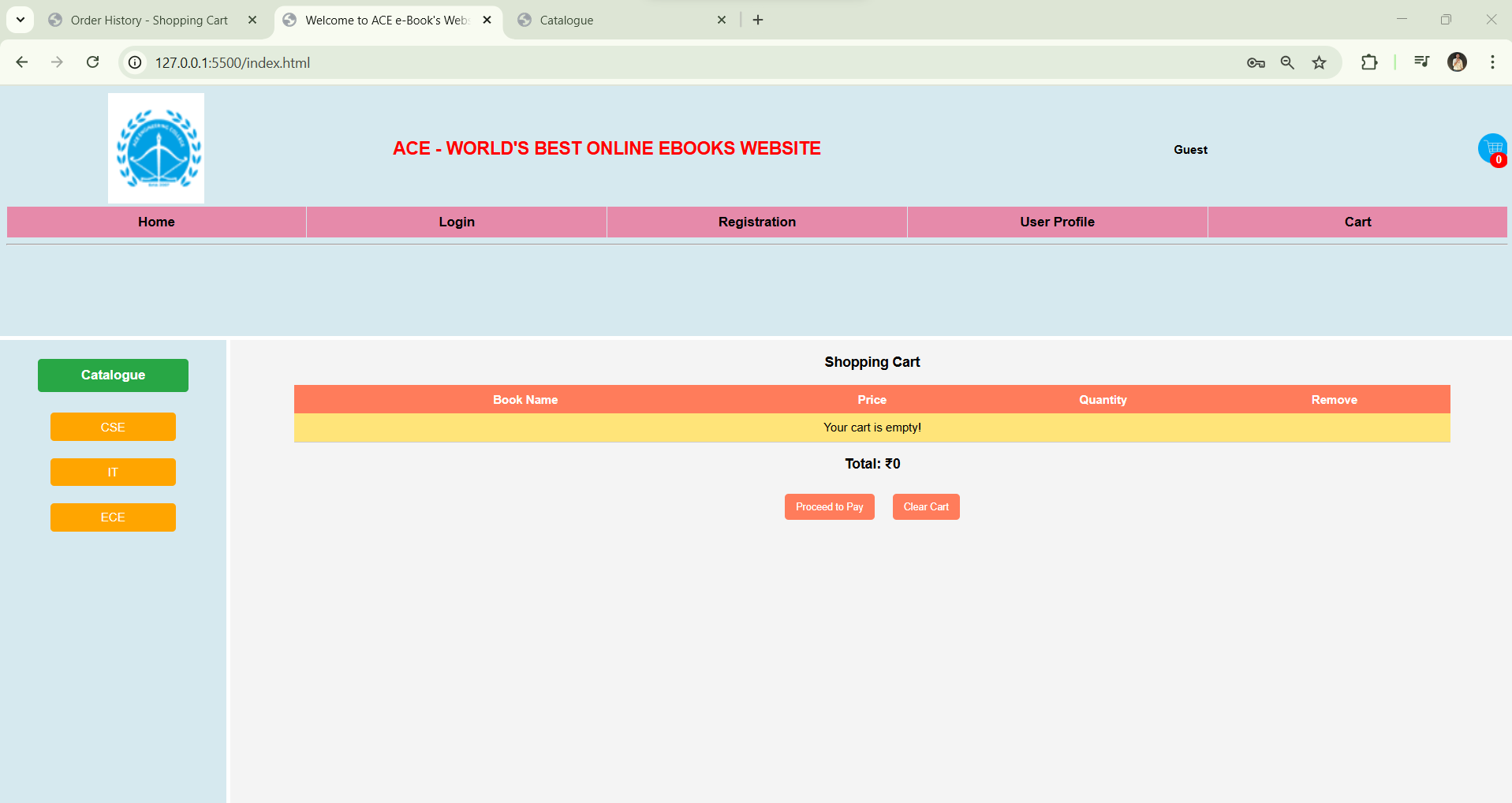


### 2.4.2 Registration Page (registration.html, registration\_success.html) – Form fields and successful registration message

### 2.4.3 Login Page (login.html) – Username and password fields

### 2.4.4 Catalog Pages (catalogue.html, cse.html, ece.html, it.html) – Product categories and layout using Flexbox/Grid

### 2.4.5 Cart Page (cart.html) – Cart items, total price, checkout button



### 2.4.6 Order Confirmation Page (orderconfirm.html) – Final order summary

# Program 3: Client-Side Validation Using JavaScript

### 3.1 Aim:

To implement client-side validation using JavaScript for the registration form and shopping cart pages.

### 3.2 Requirements:

* Software: Google Chrome, VS Code / Any Text Editor
* Technologies: HTML, CSS, JavaScript

### 3.3 Theory:

Client-side validation ensures that form inputs meet the required format before being sent to the server. JavaScript functions are used to check user input dynamically.

### 3.4 Experiment / Steps:

#### 3.4.1 Registration Form Validation:

1. Get input values (name, email, password).
2. Check if fields are empty.
3. Validate email using a regular expression.
4. Check password length (minimum 6 characters).
5. If valid, allow submission. Otherwise, show an alert.

#### **3.4.**2Shopping Cart Validation:

1. Retrieve cart items from localStorage.
2. Validate quantity (must be > 0 and a number).
3. Display an alert for incorrect input.
4. Update total price dynamically.

### 3.5 Code Implementation:

#### 3.5.1 HTML Code for Registration Form

**html**

**<form id="registrationForm">**

**<input type="text" id="nameInput" placeholder="Name" required>**

**<input type="email" id="emailInput" placeholder="Email" required>**

**<input type="password" id="passwordInput" placeholder="Password" required>**

**<button type="submit">Register</button>**

**</form>**

#### 3.5.2 validation.js

**// Ensure the script runs only after DOM is fully loaded**

**document.addEventListener("DOMContentLoaded", function () {**

**// Attach event listener to the registration form**

**const registrationForm = document.getElementById("registrationForm");**

**if (registrationForm) {**

**registrationForm.addEventListener("submit", function (event) {**

**if (!validateRegistrationForm()) {**

**event.preventDefault();**

**}**

**});**

**}**

**// Attach event listener to the login form**

**const loginForm = document.getElementById("loginForm");**

**if (loginForm) {**

**loginForm.addEventListener("submit", function (event) {**

**if (!validateLoginForm()) {**

**event.preventDefault();**

**}**

**});**

**}**

**// Display the logged-in user's name**

**displayUserProfile();**

**});**

**// Validate and Store Registration Data**

**function validateRegistrationForm() {**

**let username = document.getElementById("username").value.trim();**

**let password = document.getElementById("password").value.trim();**

**let email = document.getElementById("email").value.trim();**

**let phone = document.getElementById("phno").value.trim();**

**let address = document.getElementById("address").value.trim();**

**let gender = document.querySelector('input[name="gender"]:checked');**

**let languages = document.querySelectorAll('input[name="language"]:checked');**

**// Email Validation**

**let emailPattern = /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;**

**// Phone Number Validation (Must be 10 Digits)**

**let phonePattern = /^[0-9]{10}$/;**

**// Error messages**

**if (username === "") {**

**alert("Username is required.");**

**return false;**

**}**

**if (password.length < 6) {**

**alert("Password must be at least 6 characters long.");**

**return false;**

**}**

**if (!emailPattern.test(email)) {**

**alert("Enter a valid email address.");**

**return false;**

**}**

**if (!phonePattern.test(phone)) {**

**alert("Enter a valid 10-digit phone number.");**

**return false;**

**}**

**if (!gender) {**

**alert("Please select a gender.");**

**return false;**

**}**

**if (languages.length === 0) {**

**alert("Please select at least one language.");**

**return false;**

**}**

**if (address === "") {**

**alert("Address is required.");**

**return false;**

**}**

**// Store user data**

**let userData = {**

**username: username,**

**password: password,**

**email: email,**

**phone: phone,**

**gender: gender.value,**

**languages: Array.from(languages).map((lang) => lang.value),**

**address: address,**

**};**

**localStorage.setItem(username, JSON.stringify(userData)); // Store user data**

**alert("Registration successful!");**

**window.location.href = "login.html"; // Redirect to login page**

**return true;**

**}**

**// Validate Login Credentials**

**function validateLoginForm() {**

**let username = document.getElementById("loginUsername").value.trim();**

**let password = document.getElementById("loginPassword").value.trim();**

**let storedUser = localStorage.getItem(username);**

**if (!storedUser) {**

**alert("User not found! Please register first.");**

**return false;**

**}**

**let userData = JSON.parse(storedUser);**

**if (userData.password !== password) {**

**alert("Incorrect password!");**

**return false;**

**}**

**// Save logged-in user session**

**localStorage.setItem("loggedInUser", username);**

**alert("Login successful!");**

**window.location.href = "index.html"; // Redirect to homepage**

**return true;**

**}**

##### // Display Logged-in Username (Fixing the error)

**function displayUserProfile() {**

**let loggedInUser = localStorage.getItem("loggedInUser");**

**let profileElement = document.getElementById("userProfile");**

**if (loggedInUser && profileElement) {**

**profileElement.innerText = "Welcome, " + loggedInUser + "!";**

**}**

**}**

##### **// Logout Function**

**function logout() {**

**localStorage.removeItem("loggedInUser");**

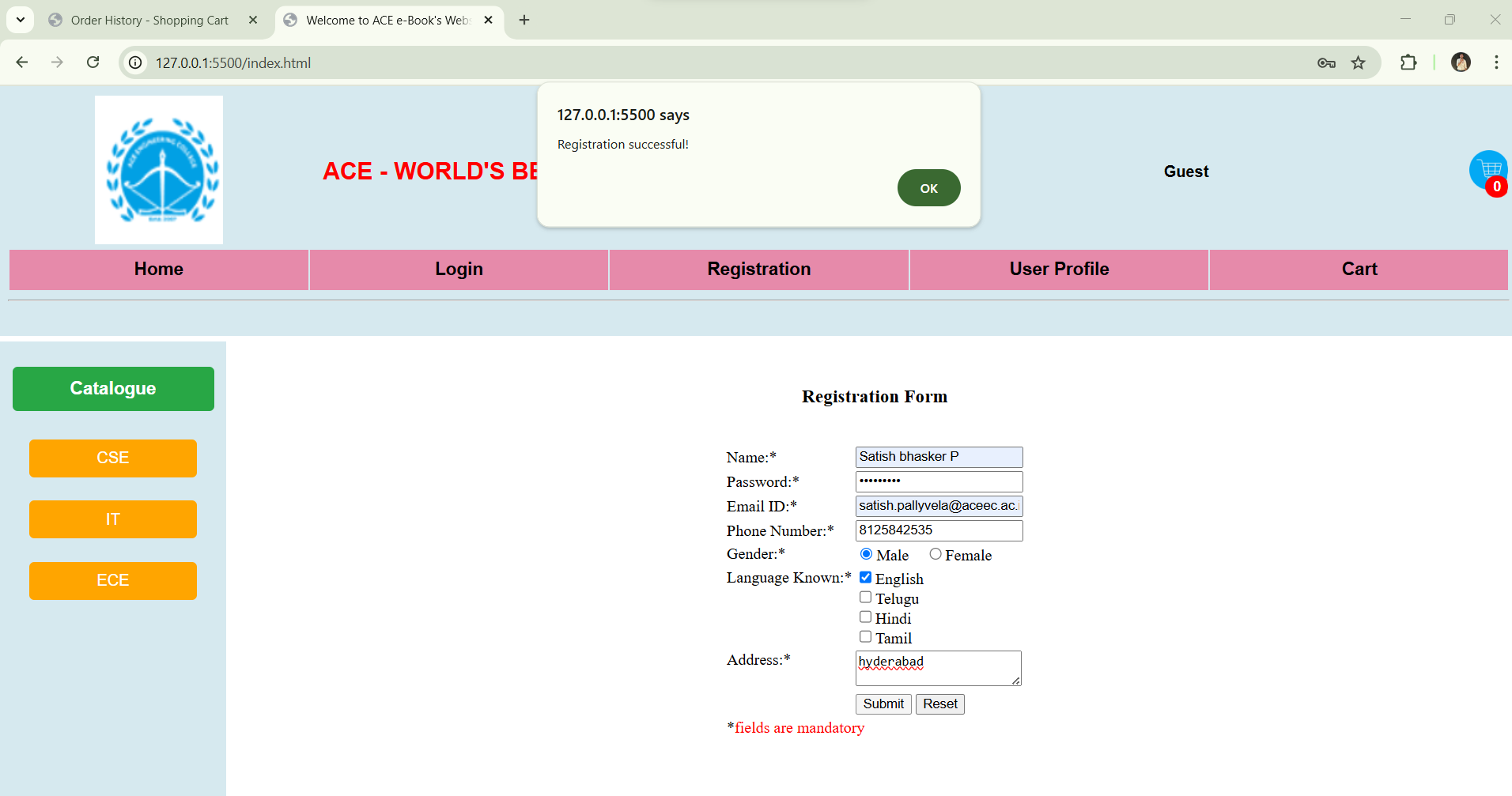
**alert("Logged out successfully!");**

**window.location.href = "login.html";**

**}**

### 3.6 Output Screenshots:

#### 3.6.1 Successful form validation

****

#### **3.6.2** Shopping cart validation errors

### 3.7 Conclusion:

This experiment demonstrated how JavaScript can validate form inputs and shopping cart entries before submission, improving the user experience and preventing incorrect data entry.

### 3.8 Viva Questions:

1. What is client-side validation?
2. Why is JavaScript used for validation?
3. What are the advantages of validating data on the client side?
4. What are regular expressions, and how do they help in form validation?
5. How does localStorage store shopping cart data?

# Program4: Weather Information Visualization Using ES6 Features

### 4.1 Aim:

To explore and implement ES6 features such as arrow functions, callbacks, promises, and async/await by developing a weather application that fetches real-time weather data from the OpenWeatherMap API. The application will utilize client-side validation and display weather information graphically using Chart.js for enhanced visualization.

### 4.2 Requirements:

* Software: Google Chrome, VS Code
* Technologies Used: HTML, CSS, JavaScript
* APIs Used: OpenWeatherMap API

### 4.3 Theory:

* **JavaScript DOM Manipulation:** Updating elements dynamically
* **User Input Validation:** Ensuring valid city names before API requests
* **Fetching Data from APIs:** Using fetch() to retrieve real-time weather data
* **Displaying Current Date:** Using JavaScript to format and show today's date
* **Styling with Google Fonts & External CSS:** Enhancing UI/UX

### 4.4 Code Implementations:

**4.4.1** (index.html) HTML Code for Weather App

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Weather-app</title>**

**<!-- Google Fonts for better styling -->**

**<link href="https://fonts.googleapis.com/css2?family=Alkatra&display=swap" rel="stylesheet">**

**<link href="https://fonts.googleapis.com/css2?family=Domine&display=swap" rel="stylesheet">**

**<link href="https://fonts.googleapis.com/css2?family=Alegreya&family=Domine&display=swap" rel="stylesheet">**

**<link href="https://fonts.googleapis.com/css2?family=Quicksand:wght@700&display=swap" rel="stylesheet">**

**<!-- Favicon for website branding -->**

**<link rel="icon" href="cloudy.png"/>**

**<!-- External CSS file for styling -->**

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

**</head>**

**<body>**

**<!-- Header Section -->**

**<header>**

**<h2 class="heading">Weather Application</h2>**

**</header>**

**<!-- Main Content Container -->**

**<div class="container">**

**<!-- Section to Display Current Date -->**

**<div class="date-container">**

**<p id="current-date"></p> <!-- JavaScript will update this dynamically -->**

**</div>**

**<!-- Input Section for City Name -->**

**<div class="container1">**

**<p>Enter the city name:</p>**

**<input type="text" id="city" placeholder="Enter city name">**

**</div>**

**<!-- Button to Fetch Weather Data -->**

**<div class="button">**

**<button id="btn">Get Weather</button>**

**</div>**

**<!-- Result Section to Display Weather Data -->**

**<div id="result"></div>**

**</div>**

**<!-- External JavaScript File -->**

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

**<!-- Footer Section -->**

**<footer class="footer">**

**<span> Ace Engineering College </span>**

**</footer>**

**<!-- JavaScript to Display the Current Date -->**

**<script>**

**document.addEventListener("DOMContentLoaded", () => {**

**// Get the current date**

**let today = new Date();**

**let formattedDate = today.toDateString(); // Example: "Mon Mar 10 2025"**

**// Display the date inside the HTML page**

**document.getElementById("current-date").innerText = "Today's Date: " + formattedDate;**

**});**

**</script>**

**</body>**

**</html>**

### 4.5 Explanation of Concepts Covered in This Experiment

Below is a detailed breakdown of each feature included in the index.html file:

#### 4.5.1 HTML Structure:

* The document follows proper **HTML5 standards** (<!DOCTYPE html>).
* The **head section** contains:
  + **Meta tags** for responsive design (viewport ensures mobile compatibility).
  + **Google Fonts** for custom typography.
  + **External CSS** for styling.
  + **Favicon** (cloudy.png) for branding.

#### 4.5.2 Displaying the Current Date (JavaScript DOM Manipulation)

Code:

**<!-- JavaScript to Display the Current Date -->**

**<script>**

**document.addEventListener("DOMContentLoaded", () => {**

**// Get the current date**

**let today = new Date();**

**let formattedDate = today.toDateString(); // Example: "Mon Mar 10 2025"**

**// Display the date inside the HTML page**

**document.getElementById("current-date").innerText = "Today's Date: " + formattedDate;**

**});**

**</script>**

This script automatically **retrieves the current date** and **updates the page dynamically** when loaded.

#### 4.5.3 Taking User Input (City Name) : Code:

**<!-- Input Section for City Name -->**

**<div class="container1">**

**<p>Enter the city name:</p>**

**<input type="text" id="city" placeholder="Enter city name">**

**</div>**

This **text input** allows users to enter the name of a city for weather lookup.

#### 4.5.4 Button to Trigger Weather Fetching

#### **Code:**

**<!-- Button to Fetch Weather Data -->**

**<div class="button">**

**<button id="btn">Get Weather</button>**

**</div>**

When clicked, this button will call a JavaScript function in **script.js** to fetch weather details.

#### 4.6 (script.js) Fetching Weather Data (Handled in script.js)

The JavaScript file (script.js) will:

* Fetch weather details from OpenWeatherMap API.
* Validate user input before making the API request.
* Display temperature, humidity, and other weather conditions dynamically.

#### 4.6.1 code implementation

**// 🔹 Select the button element from HTML by its ID**

**const btn = document.getElementById("btn");**

**// 🔹 Add an event listener to the button to trigger weather fetching when clicked**

**btn.addEventListener("click", async () => {**

**// Using async/await to fetch weather data asynchronously**

**const city = document.getElementById("city").value;   
// Get city name entered by the user**

**const apiKey = "171e009cae7e126fcbcec49c3a5cee6a";   
// OpenWeatherMap API Key**

**const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`;**

**try {**

**const response = await fetch(url);   
// 🔹 Fetch data from the API (returns a Promise)**

**const data = await response.json(); // 🔹 Convert the response to JSON format**

**// 🔹 Check if city is not found (API returns "404" when city is invalid)**

**if (data.cod === "404") {**

**document.getElementById("result").innerHTML =**

**"City not found. Try again!";**

**return; // Stop execution if city is invalid**

**}**

**displayWeather(data); // 🔹 Call function to display weather details**

**displayGraph(data.main.temp, data.main.feels\_like, data.main.humidity); // 🔹 Call function to display a bar chart**

**} catch (error) {**

**console.error("Error fetching weather:", error); // 🔹 Handle any errors**

**}**

**});**

**// 🔹 Function to display weather information dynamically in HTML**

**const displayWeather = (data) => {**

**const result = document.getElementById("result");**

**result.innerHTML = `**

**<h2 style="font-family: 'Alkatra', serif;">Weather of ${data.name}</h2>**

**<h4>Temperature: ${Math.round(**

**data.main.temp - 273.15**

**)}°C</h4> <!-- Convert Kelvin to Celsius -->**

**<h4>Feels like: ${Math.round(data.main.feels\_like - 273.15)}°C</h4>**

**<h4>Humidity: ${data.main.humidity}%</h4>**

**<h4>Pressure: ${data.main.pressure} hPa</h4>**

**<h4>Wind speed: ${data.wind.speed} m/s</h4>**

**<h4>Wind direction: ${data.wind.deg} degrees</h4>**

**`;**

**};**

**// 🔹 Function to create and display a weather chart using Chart.js**

**const displayGraph = (temp, feelsLike, humidity) => {**

**const ctx = document.getElementById("weatherChart").getContext("2d"); // 🔹 Get the canvas element from HTML**

**new Chart(ctx, {**

**type: "bar", // 🔹 Define the chart type as a bar graph**

**data: {**

**labels: ["Temperature (°C)", "Feels Like (°C)", "Humidity (%)"], // 🔹 Labels for X-axis**

**datasets: [**

**{**

**label: "Weather Data",**

**data: [**

**Math.round(temp - 273.15), // Convert temperature from Kelvin to Celsius**

**Math.round(feelsLike - 273.15),**

**humidity,**

**],**

**backgroundColor: ["red", "orange", "blue"], // 🔹 Colors for bars**

**borderColor: ["darkred", "darkorange", "darkblue"], // 🔹 Border colors**

**borderWidth: 1, // 🔹 Thickness of the bar border**

**},**

**],**

**},**

**options: {**

**scales: {**

**y: {**

**beginAtZero: true, // 🔹 Ensure Y-axis starts from 0**

**},**

**},**

**},**

**});**

**};**

**// 🔹 Function to display the current date dynamically on page load**

**document.addEventListener("DOMContentLoaded", () => {**

**let today = new Date(); // Get the current date**

**let formattedDate = today.toDateString(); // Format date as "Mon Mar 10 2025"**

**document.getElementById("current-date").innerText =**

**"Today's Date: " + formattedDate; // Insert formatted date into the HTML**

**})**

### 4.7 Expected Output

The application should:

#### 4.7.1 Display Today’s Date.

#### 4.7.2 City Name Input from the user.

#### 4.7.3 Weather Details (Temperature, Humidity, etc.) when the button is clicked.

### 4.8 Conclusion

This experiment demonstrates:

* Fetching real-time weather data using APIs.
* JavaScript DOM Manipulation to update content dynamically.
* Client-Side Validation before making API requests.
* Styling using Google Fonts "jdbc:mysql://localhost:3306/book\_management";& External CSS.

### 4.9 Viva Questions

1. What is the purpose of DOMContentLoaded in JavaScript?
2. How does fetch() work in JavaScript?
3. Why is it important to validate user input before making API requests?
4. How can you improve this weather app?
5. What are **Google Fonts**, and how do they enhance a webpage?

# Program 5: Java Standalone Application with MySQL (CRUD Operations)

### 5.1 Objective: Develop a Java Standalone Application that connects to a MySQL database and performs CRUD operations (Create, Read, Update, Delete) on database tables.

### 5.2 Required Software & Setup

1. **Install MySQL Server** (Ensure MySQL is running)
2. **Install MySQL Workbench** (For managing the database)
3. **Install OpenJDK / Java Development Kit (JDK)**
4. **Install Visual Studio Code** (VS Code)
5. **Download MySQL Connector/J** (JDBC driver for Java-MySQL connectivity

### 

### 5.3 Database Code Implementation:

#### 5.3.1 Run the following SQL commands in MySQL Workbench or the MySQL terminal:

#### 5.3.2 Step 1: Create the Database & Table

**CREATE DATABASE book\_management;**

**USE book\_management;**

**CREATE TABLE books (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**title VARCHAR(255) NOT NULL,**

**author VARCHAR(255) NOT NULL,**

**publish\_year INT NOT NULL,**

**status VARCHAR(20) DEFAULT 'Available'**

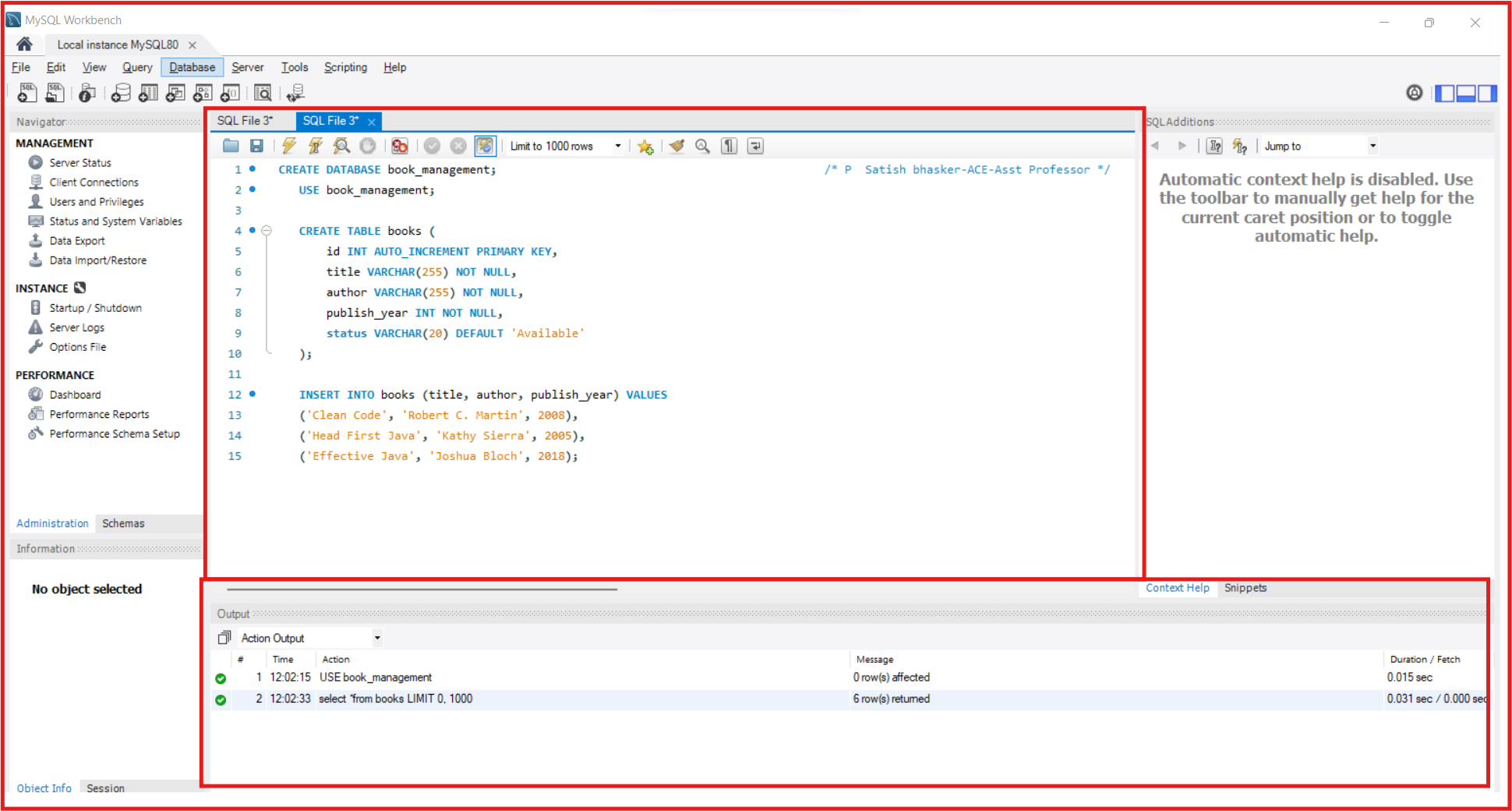
**);**

**INSERT INTO books (title, author, publish\_year) VALUES**

**('Clean Code', 'Robert C. Martin', 2008),**

**('Head First Java', 'Kathy Sierra', 2005),**

**('Effective Java', 'Joshua Bloch', 2018);**

****

**This creates a books table** with fields:

* id → Unique book ID
* title → Book name
* author → Author name
* publish\_year → Year of publication
* status → Status (Available or Borrowed)

### 5.4 Java Code Implementation

#### 5.4.1 Java Code for CRUD Operations

// Import necessary Java SQL classes for database operations

**import java.sql.\*;**

**import java.util.Scanner;** // Import Scanner to take user input

// Main class for the Book Management System

**public class MySQLConnection {**

// Database connection details

**private static final String URL =**  // Database URL  
 **private static final String USER = "root";** // MySQL username

**private static final String PASSWORD = "Aish@96786";** // MySQL password

// Main function (Entry point of the program)  
 **public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

// Scanner object to take user input

// Infinite loop to keep showing menu options until user chooses to exit

**while (true) {**

// Display menu options

**System.out.println("\n Book Management System");**

**System.out.println("1. Add Book");**

**System.out.println("2. Show All Books");**

**System.out.println("3. Borrow Book");**

**System.out.println("4. Return Book");**

**System.out.println("5. Exit");**

**System.out.print("Enter your choice: ")**;

**int choice = scanner.nextInt();** // Read user choice

**scanner.nextLine();** // Consume the newline left after entering the choice  
 // Execute the appropriate function based on user choice

**switch (choice) {**

**case 1:**

**addBook(scanner);** // Call function to add a book

break;

**case 2:**

**showAllBooks();** // Call function to display books

**break;**

**case 3:**

**borrowBook(scanner);** // Call function to borrow a book

break;

case 4:

returnBook(scanner); // Call function to return a book

**break;**

**case 5:**

**System.out.println("Exiting...");** // Exit the program

**System.exit(0);**

**default:**

**System.out.println("Invalid choice! Try again.");** // Handle invalid inputs

}

}

}

// Function to Add a Book to the Database

**public static void addBook(Scanner scanner) {**

**// Ask user for book details**

**System.out.print("Enter Book Title: ");**

**String title = scanner.nextLine();**

**System.out.print("Enter Author Name: ");**

**String author = scanner.nextLine();**

**System.out.print("Enter Publish Year: ");**

**int publishYear = scanner.nextInt();**

// Try to insert book details into the database

**try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);**

**PreparedStatement stmt = conn.prepareStatement("INSERT INTO books (title, author, publish\_year) VALUES (?, ?, ?)")) {**

// Set values in the SQL query

**stmt.setString(1, title);**

**stmt.setString(2, author);**

**stmt.setInt(3, publishYear);**

**stmt.executeUpdate();** // Execute the query  
 **System.out.println("✅ Book Added Successfully!");** // Success message **} catch (SQLException e) {System.out.println("❌ Error: " + e.getMessage());** // Print error if something goes wrong  
 }  
 }

// Function to Show All Books

**public static void showAllBooks() {**

// Try to fetch all books from the database

**try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);**

**Statement stmt = conn.createStatement();**

**ResultSet rs = stmt.executeQuery("SELECT \* FROM books")) {**

// Print table header

**System.out.println("\n📖 List of Books:");**

**System.out.printf("%-5s %-25s %-20s %-10s %-10s%n", "ID", "Title", "Author", "Year", "Status");**

**System.out.println("------------------------------------------------------------");**

// Loop through each book record and print details

**while (rs.next()) {**

**System.out.printf("%-5d %-25s %-20s %-10d %-10s%n",**

**rs.getInt("id"),**

**rs.getString("title"),**

**rs.getString("author"),**

**rs.getInt("publish\_year"),**

**rs.getString("status"));**

**}**

**} catch (SQLException e) {**

**System.out.println("❌ Error: " + e.getMessage());** // Print error if something goes wrong

**}**

**}**

// Function to Borrow a Book

**public static void borrowBook(Scanner scanner) {**

// Ask user for the book ID to borrow

**System.out.print("Enter Book ID to Borrow: ");**

**int bookId = scanner.nextInt();**

// Try to update the book's status to 'Borrowed'

**try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);**

**PreparedStatement stmt = conn.prepareStatement("UPDATE books SET status = 'Borrowed' WHERE id = ? AND status = 'Available'")) {**

**stmt.setInt(1, bookId);**

**int rowsAffected = stmt.executeUpdate();** // Execute the update query

// Check if the book was successfully borrowed

**if (rowsAffected > 0) {**

**System.out.println("📘 Book Borrowed Successfully!");**

**} else {**

**System.out.println("❌ Book Not Available or Invalid ID!");**

**}**

**} catch (SQLException e) {**

**System.out.println("❌ Error: " + e.getMessage());** // Print error if something goes wrong

**}**

**}**

// 📌 Function to Return a Book

public static void returnBook(Scanner scanner) {  
 // Ask user for the book ID to return

System.out.print("Enter Book ID to Return: ");

int bookId = scanner.nextInt();  
   
// Try to update the book's status back to 'Available'  
  **try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);  
PreparedStatement stmt = conn.prepareStatement("UPDATE books SET status = 'Available' WHERE id = ? AND status = 'Borrowed'")) {**

**stmt.setInt(1, bookId);**

**int rowsAffected = stmt.executeUpdate();** // Execute the update query

// Check if the book was successfully returned

**if (rowsAffected > 0) {**

**System.out.println("📗 Book Returned Successfully!");**

**} else {**

**System.out.println("❌ Book Not Borrowed or Invalid ID!");**

**}**

**} catch (SQLException e) {**

**System.out.println("❌ Error: " + e.getMessage());** // Print error if something goes wrong

**}**

**}**

**}**

### 5.5 Step : Compile & Run the Program

1. Open VS Code Terminal

#### 5.5.1 Compile the Java Program

**javac -cp "lib/mysql-connector-j-9.2.0.jar;src" -d bin src/MySQLConnection.java**

This command compiles your Java program (MySQLConnection.java) and includes the MySQL JDBC driver for database connectivity. Here's a simple breakdown:

* javac → Java compiler (used to compile Java files).  
  -cp "lib/mysql-connector-j-9.2.0.jar;src" →
* -cp means classpath (tells Java where to find external libraries).
* "lib/mysql-connector-j-9.2.0.jar" is the MySQL driver needed to connect Java to MySQL.
* src is the folder where your Java files are located.
* -d bin → Stores compiled .class files inside the bin directory.
* src/MySQLConnection.java → The Java file to compile (your database connection program).

#### 5.5. 2 Run the Java Program

**java -cp "lib/mysql-connector-j-9.2.0.jar;bin" MySQLConnection**

##### **5.5.2.1 Explanation (Step-by-Step)**

1. **java** → Runs the Java program.
2. **-cp (Classpath)** → Specifies where to find compiled files and external libraries.
3. **"lib/mysql-connector-j-9.2.0.jar;bin"** →
   * **lib/mysql-connector-j-9.2.0.jar** → Specifies the MySQL JDBC driver required for database connectivity.
   * **bin** → Points to the directory containing compiled .class files.
4. **MySQLConnection** → Runs the MySQLConnection.class file (without the .class extension).

### 5.5.2.2 What Happens When You Run This?

* The Java Virtual Machine (JVM) starts.
* It loads required libraries (MySQL JDBC driver).
* It executes the main method inside MySQLConnection.
* If the database connection is successful, it prints:  
   "Connected to Database Successfully!"
* If there is an issue, it prints an appropriate error message.

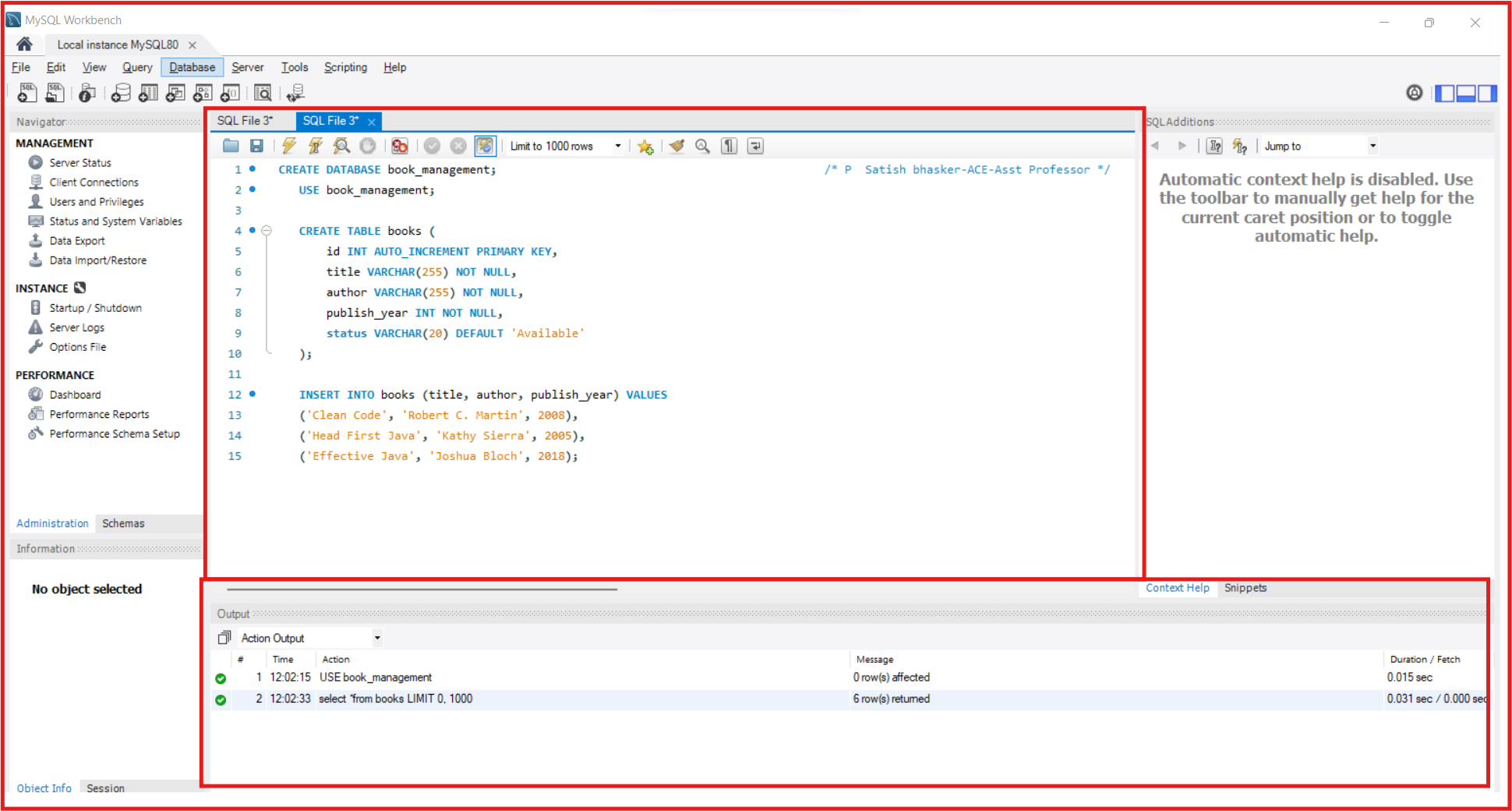
### Troubleshooting

* Ensure the JAR file path is correct.
* Check if the bin folder contains compiled .class files.
* Verify that the main class name is correct.

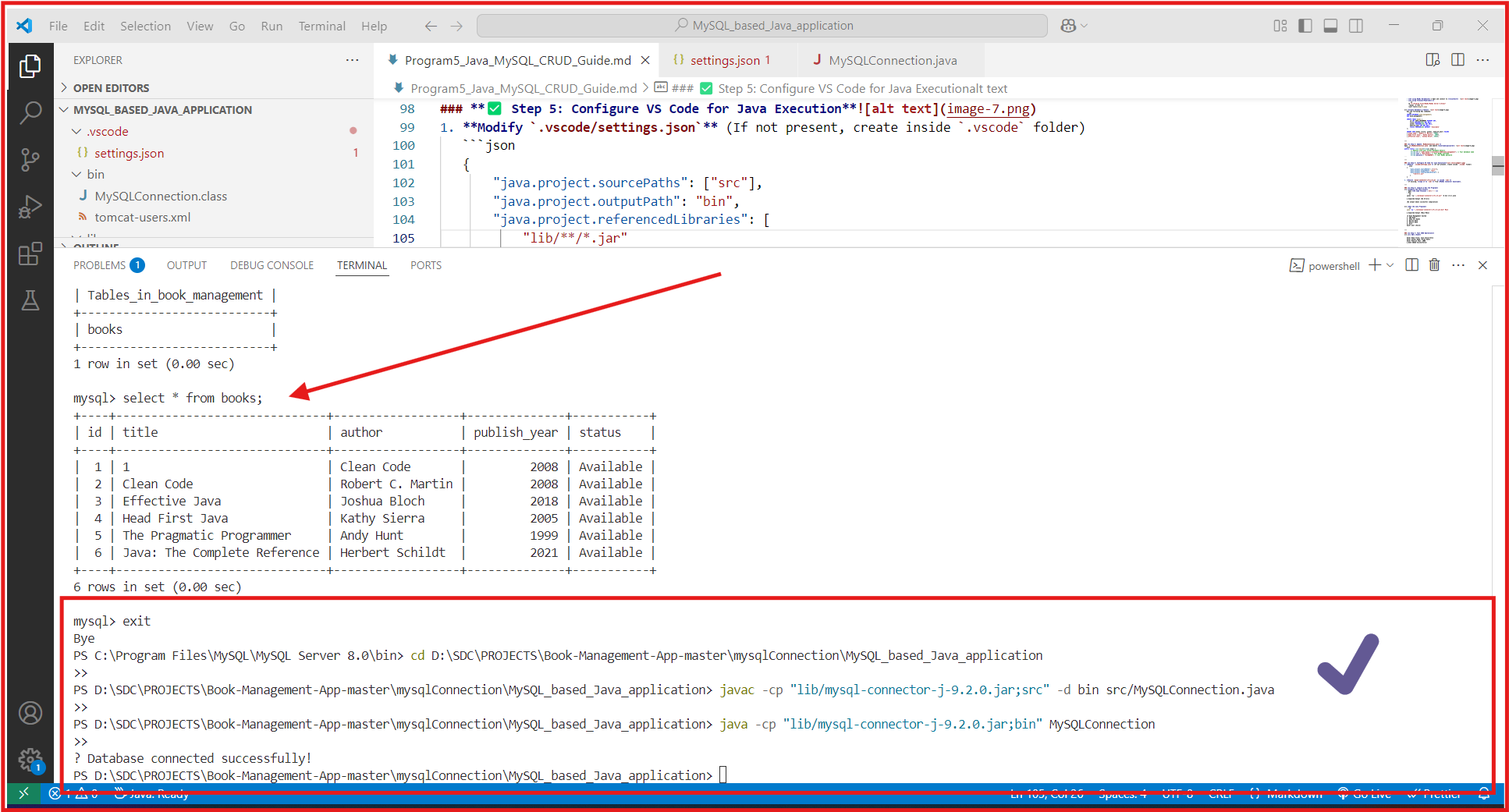
### 5.6 Outputs:

#### **5.6.1** MySQL Workbench:

* Database book\_management created.
* books table created.

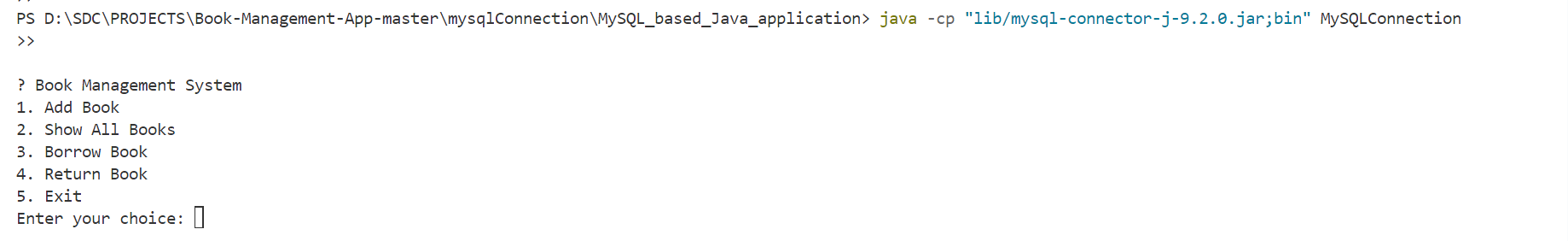


#### 5.6.2 Executing Java Program

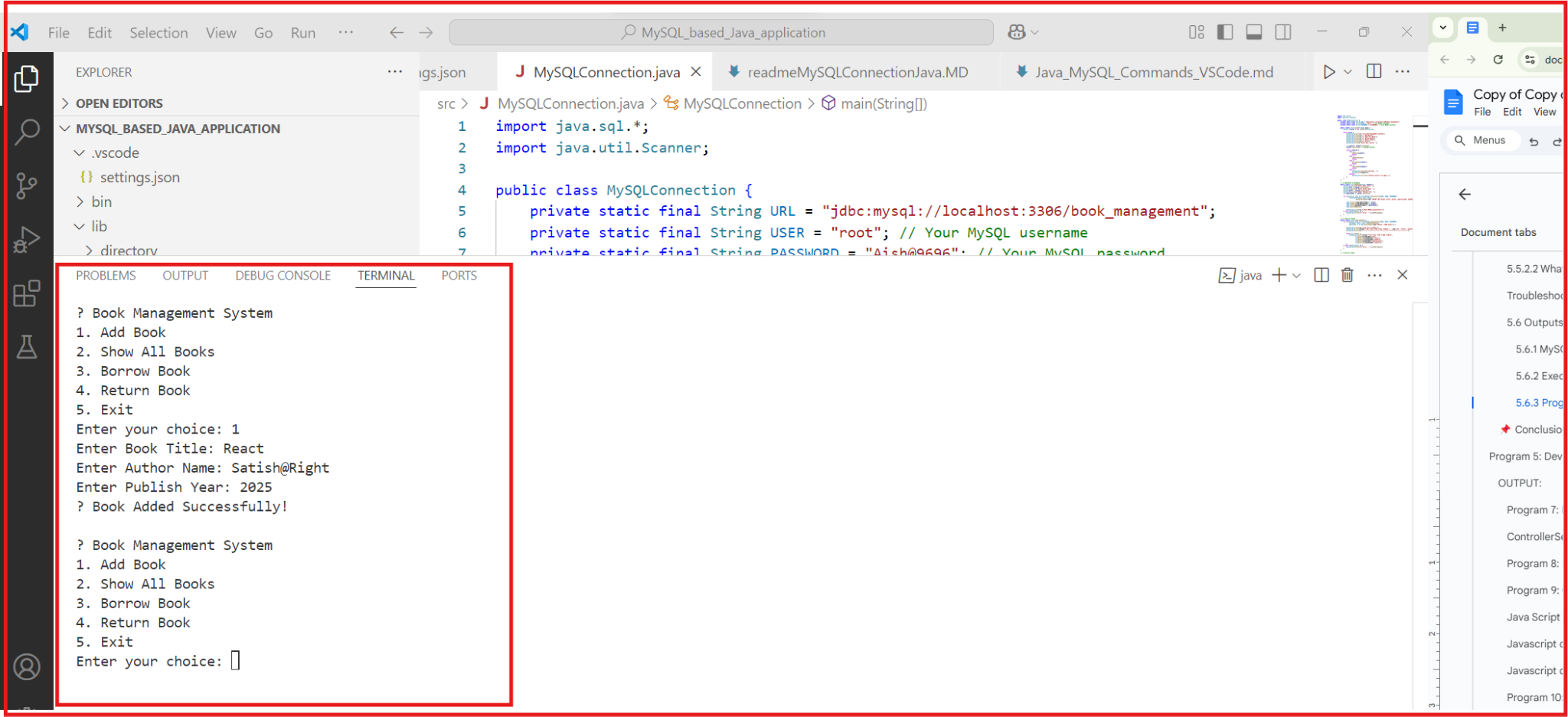
Output of Compiling Java Code (javac) and Running Java Program (java)  
 ****

#### **5.6.3** Program Output:

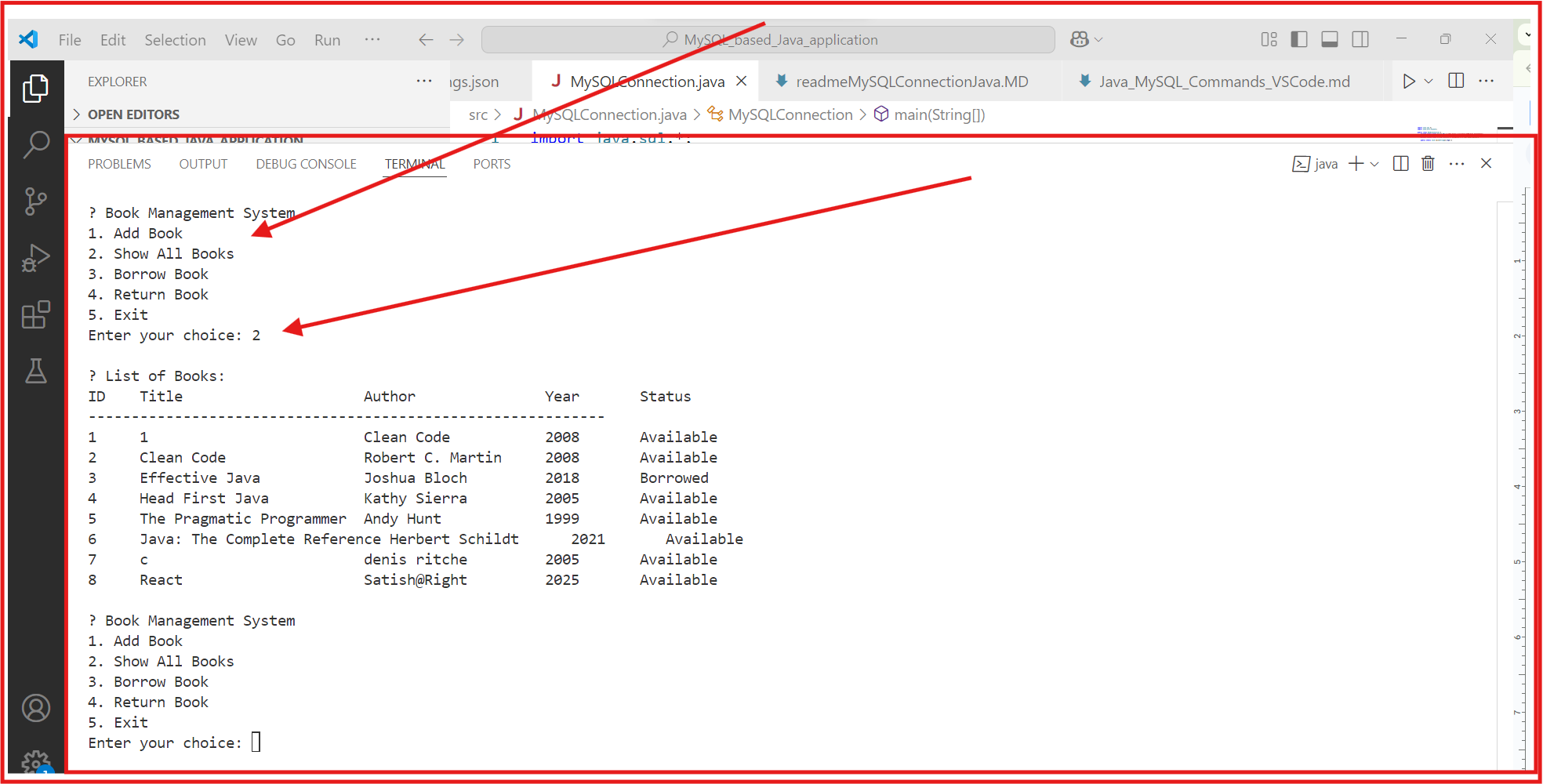
* + - Main Menu

****

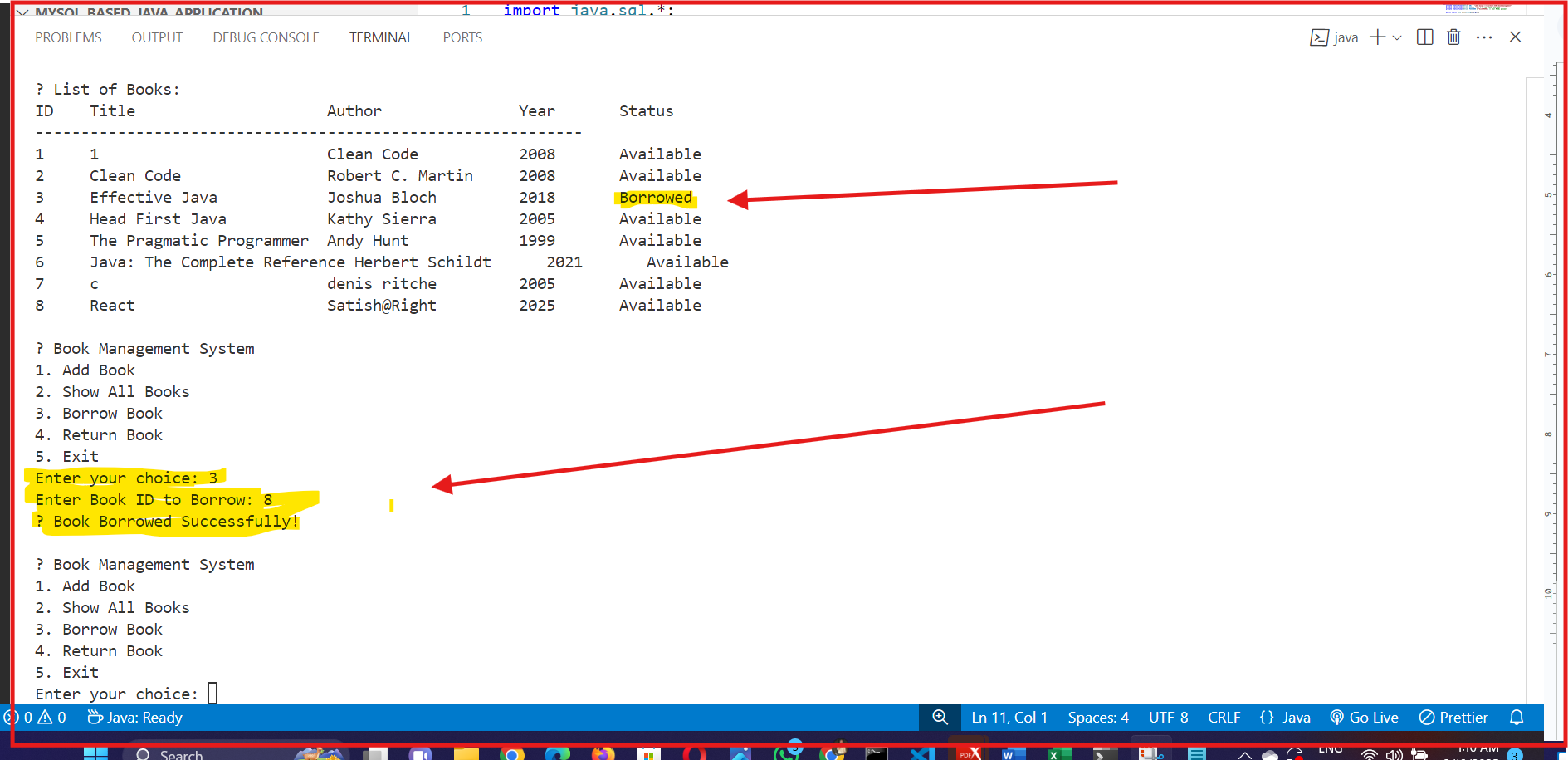
* + - **Adding a Book1**

****

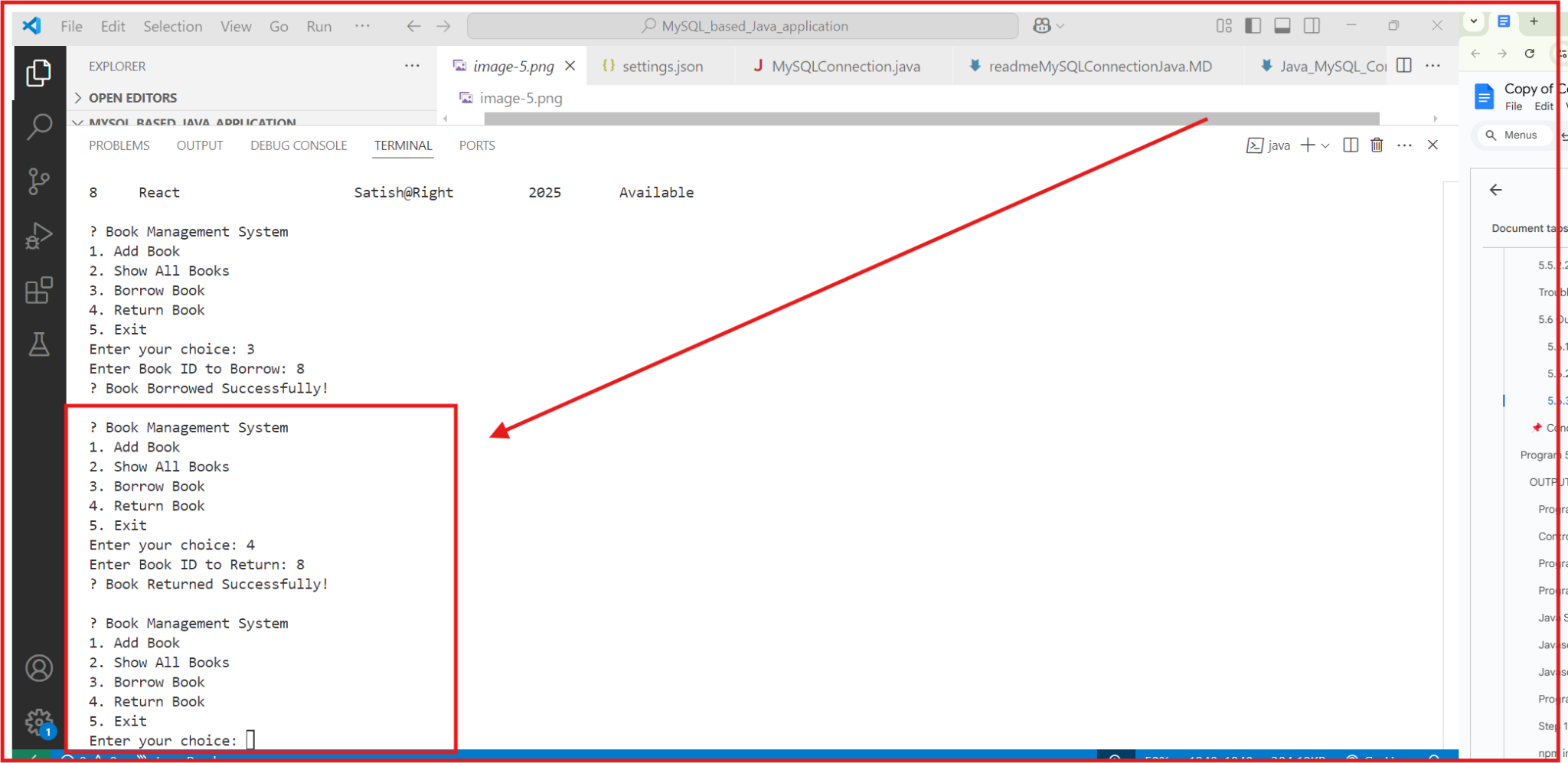
* + - **Displaying Books**

****

* + - **Borrowing a Book**

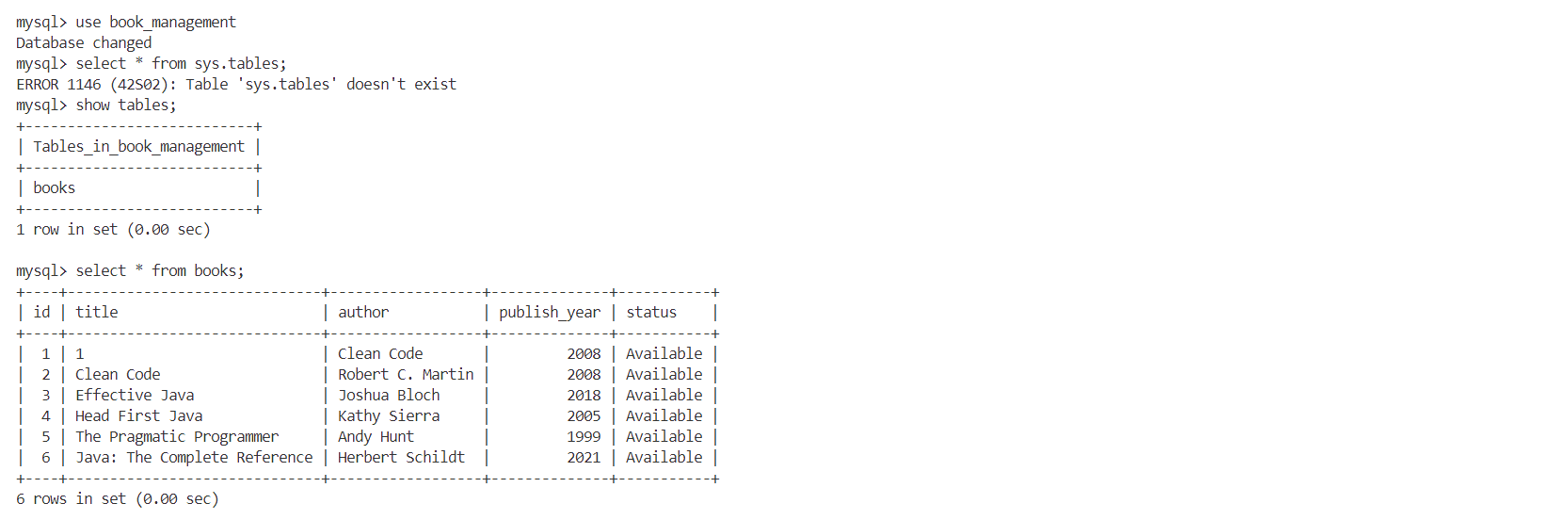
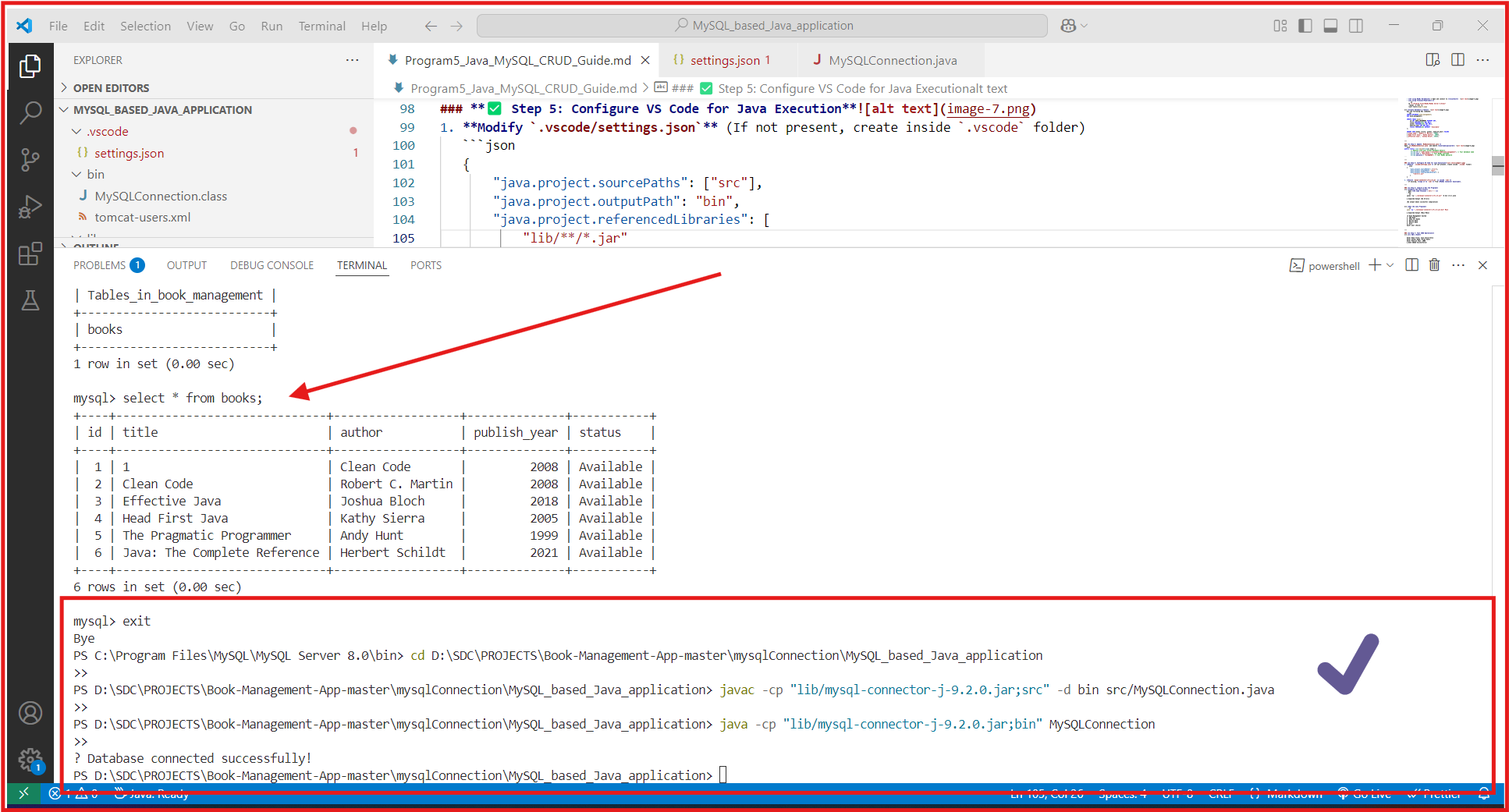
****

* + - **Returning a Book**

****

* **Checking MySQL Database**

**SELECT \* FROM books; output before and after operations.**

### 5.7 Viva Questions for Experiment 5: Java Standalone Application with MySQL (CRUD Operations)

#### 5.7.1. General Questions on Java and JDBC

1. What is JDBC, and why is it used?
2. Explain the steps to establish a database connection in Java using JDBC.
3. What are the different types of JDBC drivers?
4. What is the difference between Statement, PreparedStatement, and CallableStatement?
5. Why is PreparedStatement preferred over Statement?
6. What is the purpose of the ResultSet in JDBC?
7. What is the role of Class.forName() in JDBC?
8. What is the difference between executeQuery(), executeUpdate(), and execute()?
9. How do you handle exceptions in JDBC?
10. What is SQL Injection, and how can you prevent it?

#### 5.7.2. Questions on CRUD Operations

1. What are CRUD operations?
2. How do you insert a record into a database using JDBC?
3. How do you retrieve data from a database using JDBC?
4. How do you update a record in a database using JDBC?
5. How do you delete a record from a database using JDBC?
6. What is the purpose of auto-commit in JDBC, and how can you disable it?
7. How can you implement transactions in JDBC?

#### 5.7.3. Questions on MySQL and Database Concepts

1. What is the difference between VARCHAR and TEXT in MySQL?
2. What is the difference between PRIMARY KEY and UNIQUE KEY?
3. What is the use of AUTO\_INCREMENT in MySQL?
4. What is the difference between DELETE, TRUNCATE, and DROP in MySQL?
5. What is indexing in MySQL, and how does it improve performance?
6. What are foreign keys, and why are they important?
7. What is normalization, and why is it important in database design?
8. What are ACID properties in databases?

#### 5.7.4. Practical Implementation Questions

1. How did you configure the MySQL database connection in your project?
2. What issues did you face while setting up the MySQL connection, and how did you resolve them?
3. How does your Java application handle errors when connecting to the database?
4. How do you ensure data integrity when performing CRUD operations?
5. What improvements can be made to your Java standalone application?

### 5.8 Conclusion

* Successfully **connected Java with MySQL**.
* Performed **CRUD operations** (Create, Read, Update).
* Learned **how to compile & run Java programs in VS Code**.
* Improved **MySQL database handling using JDBC**.

# Program **6:** Designing a Servlet Controller for Application and Database Interaction

### 6.1 Aim:

To design a **Servlet Controller** that interacts with the **shopping cart application** (Experiment 1) and the **database** (Experiment 5) to manage user and product transactions.

### 6.2 Overview:

#### 6.2.1 What is a Servlet Controller?

* A **Servlet** is a Java program that runs on a server and handles web requests.
* A **Servlet Controller** acts as a central point to process user requests and interact with the **database**.
* Follows the **MVC (Model-View-Controller)** architecture:
  + **Model** → Database (MySQL/Oracle)
  + **View** → Frontend (HTML, CSS, JSP)
  + **Controller** → Servlet (Handles requests and responses)

#### 6.2.1 Why Use Servlets for Controller Design?

* Handles **dynamic web requests**.
* Manages **database interactions** for CRUD operations.
* Works as a **bridge** between frontend and backend.

#### 6.2.1 Core Features Used in This Experiment:

* **HttpServlet** → Processes HTTP GET & POST requests.
* **JDBC (Java Database Connectivity)** → Connects to MySQL/Oracle.
* **Session Management** → Stores user sessions.
* **Servlet Mapping (web.xml)** → Defines URL patterns.

### 6.3 Steps to Implement the Servlet Controller

1. Set up a Java web project in Eclipse/NetBeans.
2. Create a MySQL/Oracle database for products & users.
3. Develop a Servlet to handle registration, login, and cart management.
4. Use JDBC to interact with the database.
5. Map the Servlet in web.xml and test using Postman.

### 6.4 Code Implementation

### 6.4.1 Setting Up Database (MySQL/Oracle Schema)

#### 6.4.2 **Purpose:** Creates tables for users and products.

**CREATE DATABASE ShoppingCart;**

**USE ShoppingCart;**

**CREATE TABLE users (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**username VARCHAR(50) UNIQUE NOT NULL,**

**password VARCHAR(255) NOT NULL,**

**email VARCHAR(100) NOT NULL**

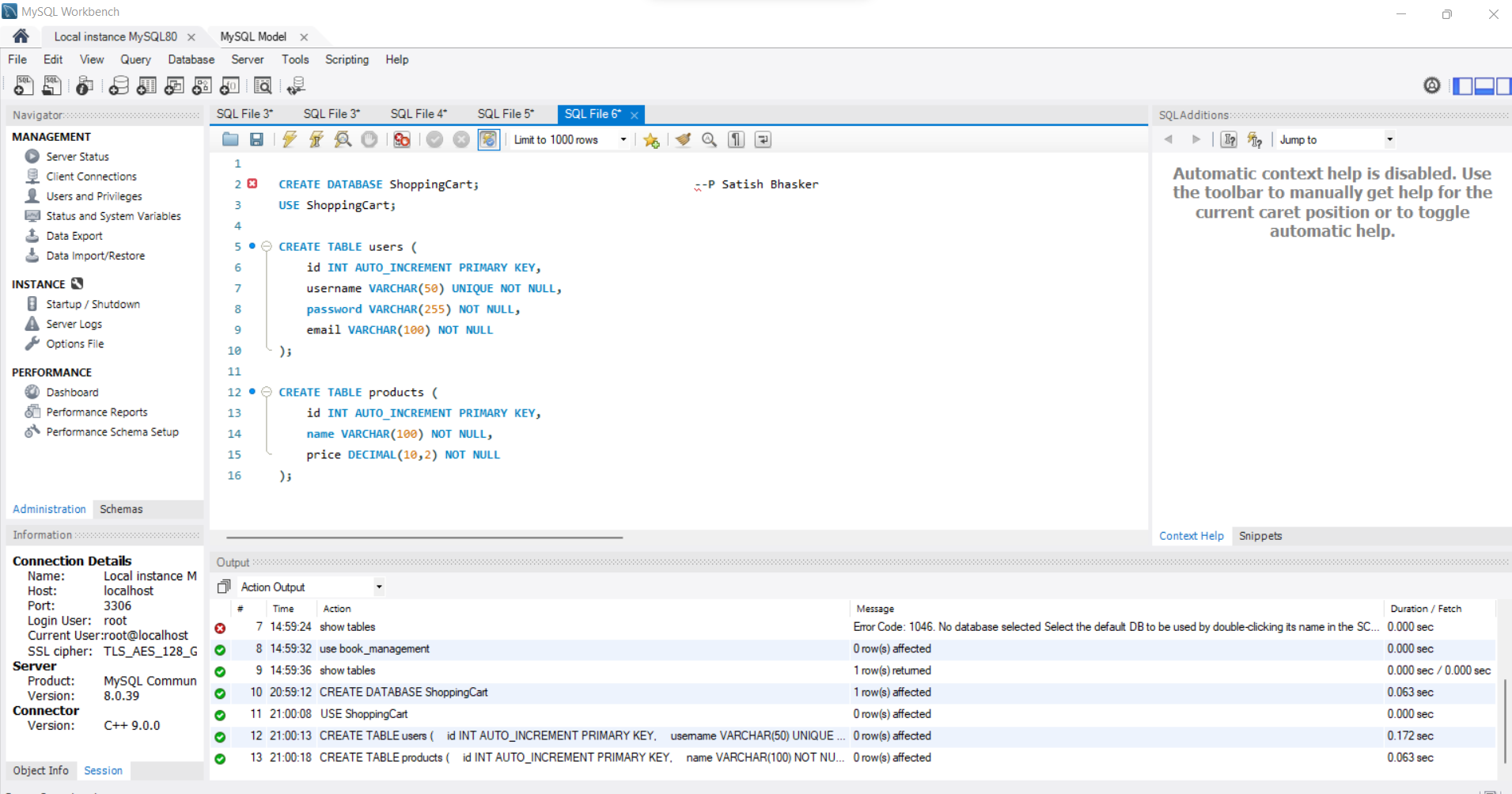
**);**

**CREATE TABLE products (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**price DECIMAL(10,2) NOT NULL**

**);**

Explanation for above Database tables:

* Defines **users** table with **username, password, and email**.
* Defines **products** table with **name and price**.

### 6.5 Creating the Servlet Controller (ShoppingController.java)

**6.5.1 Purpose:** Manages user login, product retrieval, and cart handling.

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import java.sql.\*;**

**import javax.servlet.ServletException;**

**import javax.servlet.annotation.WebServlet;**

**import javax.servlet.http.\*;**

**@WebServlet("/ShoppingController")**

**public class ShoppingController extends HttpServlet {**

**protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String action = request.getParameter("action");**

**try {**

**Class.forName("com.mysql.cj.jdbc.Driver");**

**Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ShoppingCart", "root", "password");**

**if ("register".equals(action)) {**

**String username = request.getParameter("username");**

**String password = request.getParameter("password");**

**PreparedStatement ps = con.prepareStatement("INSERT INTO users (username, password, email) VALUES (?, ?, ?)");**

**ps.setString(1, username);**

**ps.setString(2, password);**

**ps.setString(3, request.getParameter("email"));**

**ps.executeUpdate();**

**out.println("Registration Successful!");**

**}**

**else if ("login".equals(action)) {**

**String username = request.getParameter("username");**

**String password = request.getParameter("password");**

**PreparedStatement ps = con.prepareStatement("SELECT \* FROM users WHERE username=? AND password=?");**

**ps.setString(1, username);**

**ps.setString(2, password);**

**ResultSet rs = ps.executeQuery();**

**if (rs.next()) {**

**HttpSession session = request.getSession();**

**session.setAttribute("user", username);**

**out.println("Login Successful!");**

**} else {**

**out.println("Invalid Credentials");**

**}**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

#### 6.5.2 Explanation:

* Uses **JDBC** to connect to MySQL.
* Handles **user registration and login**.
* Uses **HttpSession** to maintain login sessions.

### 6.6 Configuring Servlet Mapping (web.xml)

**Purpose:** Defines the URL pattern for Servlet.

**<web-app>**

**<servlet>**

**<servlet-name>ShoppingController</servlet-name>**

**<servlet-class>ShoppingController</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>ShoppingController</servlet-name>**

**<url-pattern>/ShoppingController</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**Explanation:**

* Defines ShoppingController Servlet.
* Maps it to /ShoppingController URL.

## 6.6 How to Run and Test the Servlet?

### 6.6.1 Set Up Environment:

* Install **Eclipse/NetBeans**.
* Install **Tomcat Server**.
* Create a **Dynamic Web Project**.

### 6.6.2 Deploy and Run:

* Save **ShoppingController.java** in src/.
* Run web.xml configuration.
* Start the Tomcat server.

### 6.6.3 Test with Postman:

#### 6.6.3.1 **Register a User**

* **Method:** POST
* **URL:** http://localhost:8080/ShoppingController?action=register
* **Body:**

{

"username": "john\_doe",

"password": "12345",

"email": "john@example.com"

}

* **Response:** Registration Successful!

**Login a User**

* **Method:** POST
* **URL:** http://localhost:8080/ShoppingController?action=login
* **Body:**

{

"username": "john\_doe",

"password": "12345"

}

* **Response:** Login Successful!

## 6.7 Conclusion

* Successfully **designed a Servlet Controller** for application & database interaction.
* Implemented **user registration and login using JDBC**.
* Managed **sessions for authentication**.
* Configured Servlet in **web.xml**.

## 6.8 Viva Questions

1. What is a **Servlet Controller**?
2. What is the **role of JDBC** in database interaction?
3. How does **HttpSession** help in session management?
4. Why do we use **web.xml**?
5. How can we extend this Servlet to **handle product purchases**?

# Program 7: Servlet Controller with Session Tracking

### 7.1 Aim:

To design a Servlet Controller that interacts with the application from Experiment 1 and the database from Experiment 5, while implementing session tracking mechanisms (Cookies, HTTP Sessions) to maintain user transaction history.

### 7.2 Overview:

#### 7.2.1 What is a Servlet Controller?

A Servlet Controller handles:

* User requests from a web application.
* Processes data by interacting with a MySQL database.
* Manages session tracking to maintain transaction history.

#### 7.2.2 Why Session Tracking?

Session tracking ensures that:

* The server remembers user interactions.
* Users can maintain login status across pages.
* Transaction history can be stored without losing data.

#### 7.2.3 Types of Session Tracking Used in this Experiment:

1 Cookies → Stores small data in the user's browser.  
2 HTTP Session → Maintains user state on the server side.

### 7.3 Steps to Implement Servlet Controller with Session Tracking

1. Create a Web Form (HTML) to Collect User Input.
2. Develop a Servlet (Controller) to Handle Requests.
3. Use Session Tracking (Cookies & HTTP Sessions).
4. Connect the Servlet to a MySQL Database.
5. Retrieve and Display Transaction History.
6. Deploy and Test the Application on Tomcat.

### 7.4 Code Implementation

#### 7.4.1 HTML Form (index.html)

#### **Purpose:** Collects user input and sends it to the Servlet.

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>User Login</title>**

**</head>**

**<body>**

**<h2>Login</h2>**

**<form action="LoginServlet" method="post">**

**Username: <input type="text" name="username" required><br>**

**<input type="submit" value="Login">**

**</form>**

**</body>**

**</html>**

#### 7.4.2 Explanation:

- When submitted, data is sent to LoginServlet.

### 7.5 Servlet (LoginServlet.java) with Session Tracking

Purpose: Handles user login, tracks sessions using Cookies & HTTP Sessions.

***import java.io.\*;***

***import javax.servlet.\*;***

***import javax.servlet.http.\*;***

***import java.sql.\*;***

**public class LoginServlet extends HttpServlet {**

**protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**

**String username = request.getParameter("username");**

**HttpSession session = request.getSession(); // Start Session**

**session.setAttribute("username", username); // Store Username in Session**

**// Create Cookie to store username**

**Cookie userCookie = new Cookie("user", username);**

**userCookie.setMaxAge(60 \* 60); // Cookie expires in 1 hour**

**response.addCookie(userCookie);**

**PrintWriter out = response.getWriter();**

**response.setContentType("text/html");**

**out.println("<h3>Welcome, " + username + "</h3>");**

**out.println("<a href='TransactionServlet'>View Transaction History</a>");**

**}**

**}**

### 7.5.1 Explanation:

- Stores username in a session.

- Creates a cookie to store user data temporarily.

- Redirects to TransactionServlet for history retrieval.

### 7.6 Servlet (TransactionServlet.java) for Viewing History

Purpose: Retrieves transaction history from the database using the session username.

**java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**import java.sql.\*;**

**public class TransactionServlet extends HttpServlet {**

**protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**

**HttpSession session = request.getSession(false);**

**String username = (String) session.getAttribute("username");**

**PrintWriter out = response.getWriter();**

**response.setContentType("text/html");**

**if (username == null) {**

**out.println("<h3>No active session. Please <a href='index.html'>login</a> again.</h3>");**

**return;**

**}**

**try {**

**Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase", "root", "password");**

**String query = "SELECT \* FROM transactions WHERE username=?";**

**PreparedStatement ps = con.prepareStatement(query);**

**ps.setString(1, username);**

**ResultSet rs = ps.executeQuery();**

**out.println("<h2>Transaction History for " + username + "</h2>");**

**while (rs.next()) {**

**out.println("<p>Transaction: " + rs.getString("details") + "</p>");**

**}**

**con.close();**

**} catch (Exception e) {**

**out.println("<h3>Error: " + e.getMessage() + "</h3>");**

**}**

**}**

**}**

#### 7.6.1 Explanation (TransactionServlet.java):

- Retrieves username from session.

- Queries the transaction history from MySQL.

- Displays history if a session exists, else prompts re-login.

### 7.7 Deployment Descriptor (web.xml)

Purpose: Maps Servlets to URLs.

**<?xml version="1.0" encoding="UTF-8"?>**

**<web-app>**

**<servlet>**

**<servlet-name>LoginServlet</servlet-name>**

**<servlet-class>LoginServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>LoginServlet</servlet-name>**

**<url-pattern>/LoginServlet</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>TransactionServlet</servlet-name>**

**<servlet-class>TransactionServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>TransactionServlet</servlet-name>**

**<url-pattern>/TransactionServlet</url-pattern>**

**</servlet-mapping>**

**</web-app>**

#### 7.7.1 Expected Output

* **Browser Output after Login:**  
   Welcome, JohnDoe  
   [View Transaction History]
* **Transaction History Output:**

Transaction: Purchased Book - Java Programming

Transaction: Added Balance - ₹500

### 

### 7.8 Conclusion

* Implemented a Servlet Controller to handle user requests.
* Used Cookies & HTTP Sessions for session tracking.
* Retrieved and displayed transaction history dynamically.

### 7.9 Viva Questions

1. What is session tracking and why is it used?
2. How does HTTP Session differ from Cookies?
3. What is the role of doPost() and doGet() in Servlets?
4. How does `web.xml` help in Servlet deployment?
5. What happens when a session expires?

# Program 8: Building a RESTful API using Java Servlets

### 8.1 Aim

To develop a RESTful API using Java Servlets that can perform CRUD (Create, Read, Update, Delete) operations on a MySQL database.

### 8.2 Overview

#### 8.2.1 What is a RESTful API?

A RESTful API (Representational State Transfer API) allows communication between a client (like a web app or mobile app) and a server. It follows these principles:

* Uses HTTP Methods (GET, POST, PUT, DELETE)
* Returns data in JSON format
* Stateless (Each request is independent)

#### 8.2.2 Why use Servlets for REST API?

* Java Servlets handle HTTP requests directly.
* They can connect to a database and perform operations.
* They return JSON responses, which makes them suitable for APIs.

### 8.3 Steps to Implement a REST API using Servlets

1. 1 Set up the MySQL database and table
2. Create a Servlet to handle API requests
3. Use HTTP methods (GET, POST, PUT, DELETE) for CRUD operations
4. Connect to MySQL using JDBC
5. Return responses in JSON format
6. Test the API using Postman

## 8.4 Code Implementation

### Create MySQL Database and Table

#### Run this SQL query to set up the database.

**CREATE DATABASE studentdb;**

**USE studentdb;**

**CREATE TABLE students (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100),**

**email VARCHAR(100)**

**);**

### 2 Servlet: StudentAPI.java (Handles CRUD operations)

This servlet will handle API requests and return JSON responses.

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**import java.sql.\*;**

**import org.json.\*;**

**public class StudentAPI extends HttpServlet {**

**private Connection connectDB() throws SQLException {**

**return DriverManager.getConnection("jdbc:mysql://localhost:3306/studentdb", "root", "password");**

**}**

**protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**response.setContentType("application/json");**

**PrintWriter out = response.getWriter();**

**JSONArray studentsArray = new JSONArray();**

**try (Connection con = connectDB()) {**

**Statement stmt = con.createStatement();**

**ResultSet rs = stmt.executeQuery("SELECT \* FROM students");**

**while (rs.next()) {**

**JSONObject student = new JSONObject();**

**student.put("id", rs.getInt("id"));**

**student.put("name", rs.getString("name"));**

**student.put("email", rs.getString("email"));**

**studentsArray.put(student);**

**}**

**out.print(studentsArray);**

**} catch (Exception e) {**

**out.print("{\"error\":\"" + e.getMessage() + "\"}");**

**}**

**}**

**protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**String name = request.getParameter("name");**

**String email = request.getParameter("email");**

**try (Connection con = connectDB()) {**

**PreparedStatement ps = con.prepareStatement("INSERT INTO students (name, email) VALUES (?, ?)");**

**ps.setString(1, name);**

**ps.setString(2, email);**

**ps.executeUpdate();**

**response.getWriter().print("{\"message\":\"Student added successfully\"}");**

**} catch (Exception e) {**

**response.getWriter().print("{\"error\":\"" + e.getMessage() + "\"}");**

**}**

**}**

**protected void doPut(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**int id = Integer.parseInt(request.getParameter("id"));**

**String name = request.getParameter("name");**

**String email = request.getParameter("email");**

**try (Connection con = connectDB()) {**

**PreparedStatement ps = con.prepareStatement("UPDATE students SET name=?, email=? WHERE id=?");**

**ps.setString(1, name);**

**ps.setString(2, email);**

**ps.setInt(3, id);**

**ps.executeUpdate();**

**response.getWriter().print("{\"message\":\"Student updated successfully\"}");**

**} catch (Exception e) {**

**response.getWriter().print("{\"error\":\"" + e.getMessage() + "\"}");**

**}**

**}**

**protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws IOException {**

**int id = Integer.parseInt(request.getParameter("id"));**

**try (Connection con = connectDB()) {**

**PreparedStatement ps = con.prepareStatement("DELETE FROM students WHERE id=?");**

**ps.setInt(1, id);**

**ps.executeUpdate();**

**response.getWriter().print("{\"message\":\"Student deleted successfully\"}");**

**} catch (Exception e) {**

**response.getWriter().print("{\"error\":\"" + e.getMessage() + "\"}");**

**}**

**}**

**}**

### Explanation:

* doGet() → Fetches all students.
* doPost() → Adds a new student.
* doPut() → Updates student details.
* doDelete() → Deletes a student.

#### Deployment Descriptor (web.xml)

**<?xml version="1.0" encoding="UTF-8"?>**

**<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee**

**http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"**

**version="3.1">**

**<servlet>**

**<servlet-name>StudentAPI</servlet-name>**

**<servlet-class>StudentAPI</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>StudentAPI</servlet-name>**

**<url-pattern>/StudentAPI</url-pattern>**

**</servlet-mapping>**

**</web-app>**

#### Explanation:

* **Maps /StudentAPI to StudentAPI servlet.**

### Testing the API using Postman

This section provides instructions on how to test the `StudentAPI` using Postman, including the operations, HTTP methods, URLs, and request bodies.

#### 1. Get All Students

##### HTTP Method : `GET`

##### URL:[**http://localhost:8080/StudentAPI**](http://localhost:8080/StudentAPI)

##### Body : None

#### Steps:

1. Open Postman.

2. Select `GET` from the dropdown menu.

3. Enter the URL.

4. Click on `Send`.

##### 

##### 2. Add Student

* HTTP Method: `POST`
* URL:<http://localhost:8080/StudentAPI>
* Body:json {   
   "name": "Satish",  
   "email": "[satish.pallyvela@aceec.ac.in](mailto:satish.pallyvela@aceec.ac.in)"  
   }

#### Steps:

1. Open Postman.

2. Select `POST` from the dropdown menu.

3. Enter the URL.

4. Go to the `Body` tab.

5. Select `raw` and choose `JSON` from the dropdown on the right.

6. Paste the JSON body.

7. Click on `Send`.

##### ## 3. Update Student

**HTTP Method: `PUT`**

**URL: http://localhost:8080/StudentAPI?id=1**

**Body:json**

**{**

**"name": "satish",**

**"email": "satish.pallyvela@aceec.ac.in"**

**}**

#### Steps:

1. Open Postman.

2. Select `PUT` from the dropdown menu.

3. Enter the URL (replace `1` with the actual student ID you want to update).

4. Go to the `Body` tab.

5. Select `raw` and choose `JSON` from the dropdown on the right.

6. Paste the JSON body.

7. Click on `Send`.

#### 

##### 4. Delete Student

* **HTTP Method: `DELETE`**
* **URL: http://localhost:8080/StudentAPI?id=1**
* **Body: None**

**\*\*Steps\*\*:**

1. Open Postman.

2. Select `DELETE` from the dropdown menu.

3. Enter the URL (replace `1` with the actual student ID you want to delete).

4. Click on `Send`.

#### Summary of Operations

| **Operation** | **HTTP Method** | **URL** | **Body (if needed)** |
| --- | --- | --- | --- |
| Get all students | GET | http://localhost:3000/students | None |
| Add student | POST | http://localhost:3000/students | { "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" } |

#### 10.6.1 Use Postman to test API endpoints.

##### Expected Output (GET request):

**[**

**{ "id": 1, "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" }**

**] |**

#### Notes

* Ensure your server is running and the database is properly set up before testing.
* Adjust the student ID in the update and delete requests as needed.
* Check the responses in Postman to verify the success or failure of each operation.

## 8.6 Conclusion

* **Created a RESTful API using Java Servlets.**
* **Implemented CRUD operations with MySQL.**
* **Used JSON format for responses.**
* **Tested API using Postman.**

## 8.7 Viva Questions

* What is a RESTful API?
* Explain GET, POST, PUT, DELETE methods.
* How does a Servlet handle HTTP requests?
* What is JSON, and why is it used in APIs?
* How do we test APIs in Postman?

### 8.8 Final Steps

Run the API on Tomcat.

Take Postman screenshots and attach them.

# PROGRAM 9: Building a RESTful API using Node.js and Express

## 9.1 Aim

**To develop a RESTful API using Node.js and Express.js that can perform CRUD (Create, Read, Update, Delete) operations on a MySQL database.**

## 9.2 Overview

### What is a RESTful API?

**A RESTful API allows communication between a client (web app, mobile app) and a server. It follows:**

* **Uses HTTP Methods: GET, POST, PUT, DELETE**
* **Returns JSON data**
* **Stateless Architecture**

### Why Use Node.js and Express?

* **Express.js simplifies handling requests.**
* **Node.js is lightweight and fast.**
* **MySQL stores data permanently.**
* **API returns JSON responses (widely used in web and mobile applications).**

## 9.3 Steps to Implement

* Install Node.js and Express
* Set up MySQL database
* Create an Express server
* Implement CRUD operations
* Test the API using Postman

## 9.4 Code Implementation

#### Install Node.js and Express

##### Open a terminal and run:

**npm init -y**

**npm install express mysql body-parser cors**

##### Explanation:

* **express → Handles HTTP requests**
* **mysql → Connects to MySQL database**
* **body-parser → Parses incoming JSON data**
* **cors → Allows cross-origin requests**

### 2 Create MySQL Database and Table

#### Run this SQL query to set up the database:

**CREATE DATABASE studentdb;**

**USE studentdb;**

**CREATE TABLE students (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100),**

**email VARCHAR(100)**

**);**

### 3 Create the Express Server (server.js)

### **server.js – Main entry point for the API**

**const express = require("express");**

**const mysql = require("mysql");**

**const bodyParser = require("body-parser");**

**const cors = require("cors");**

**const app = express();**

**app.use(bodyParser.json());**

**app.use(cors());**

**// Database Connection**

**const db = mysql.createConnection({**

**host: "localhost",**

**user: "root",**

**password: "password",**

**database: "studentdb",**

**});**

**db.connect(err => {**

**if (err) throw err;**

**console.log("Connected to MySQL Database!");**

**});**

##### // Routes

##### // 1 Fetch all students (GET)

**app.get("/students", (req, res) => {**

**db.query("SELECT \* FROM students", (err, result) => {**

**if (err) throw err;**

**res.json(result);**

**});**

**});**

##### // 2 Add a new student (POST)

**app.post("/students", (req, res) => {**

**const { name, email } = req.body;**

**db.query("INSERT INTO students (name, email) VALUES (?, ?)", [name, email], (err, result) => {**

**if (err) throw err;**

**res.json({ message: "Student added successfully!" });**

**});**

**});**

##### // 3 Update a student (PUT)

**app.put("/students/:id", (req, res) => {**

**const { id } = req.params;**

**const { name, email } = req.body;**

**db.query("UPDATE students SET name=?, email=? WHERE id=?", [name, email, id], (err, result) => {**

**if (err) throw err;**

**res.json({ message: "Student updated successfully!" });**

**});**

**});**

##### // 4 Delete a student (DELETE)

**app.delete("/students/:id", (req, res) => {**

**const { id } = req.params;**

**db.query("DELETE FROM students WHERE id=?", [id], (err, result) => {**

**if (err) throw err;**

**res.json({ message: "Student deleted successfully!" });**

**});**

**});**

**// Start Server**

**app.listen(3000, () => {**

**console.log("Server running on port 3000");**

**});**

##### Explanation:

* **GET /students → Fetches all students.**
* **POST /students → Adds a new student.**
* **PUT /students/:id → Updates a student’s details.**
* **DELETE /students/:id → Deletes a student.**

### 9.5 Running the API

1. Start MySQL Server  
   Ensure MySQL is running.
2. Run the Node.js Server  
    Open a terminal in the project folder and run:

**node server.js**

**✅ Expected Output:**

**Server running on port 3000**

**Connected to MySQL Database!**

## 9.6 Testing the API using Postman

| **Operation** | **HTTP Method** | **URL** | **Body (if needed)** |
| --- | --- | --- | --- |
| Get all students | GET | http://localhost:3000/students | None |
| Add student | POST | http://localhost:3000/students | { "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" } |
| Update student | PUT | http://localhost:3000/students/1 | { "name": "Satish Pallyvela", "email": "satish.pallyvela@aceec.ac.in" } |
| Delete student | DELETE | http://localhost:3000/students/1 | None |

#### 9.6.1 Use Postman to test API endpoints.

##### Expected Output (GET request):

**[**

**{ "id": 1, "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" }**

**]**

### 9.7 Conclusion

* Created a RESTful API using Node.js and Express.js.
* Connected to MySQL database.
* Implemented CRUD operations.
* Tested the API using Postman.

9.8 Viva Questions

1. What is a RESTful API?
2. What are GET, POST, PUT, DELETE methods used for?
3. How does Express.js handle HTTP requests?
4. What is CORS, and why is it needed?
5. How do you connect Node.js to MySQL?

### 9.9 Final Steps

* Run the API and take Postman screenshots.
* Test database operations.

**Would you like any modifications or additional explanations? 🚀**

**Here’s a structured version of Experiment 10: Building a Full-Stack Application using React.js and Node.js in simple language:**

# Program 10: Building a Full-Stack Application using React.js and Node.js

### 10.1 Aim

To develop a full-stack web application using:

* React.js (Frontend) → Handles user interface
* Node.js + Express.js (Backend) → Handles API requests
* MySQL (Database) → Stores application data

### 10.2 Overview :

#### 10.2.1 What is a Full-Stack Application?

##### A full-stack application includes:

* Frontend (React.js) → Displays the UI and interacts with the backend
* Backend (Node.js + Express.js) → Processes requests and connects to the database
* Database (MySQL) → Stores and retrieves data

#### 10.2.2 Why Use React.js, Node.js, and MySQL?

* React.js → Component-based UI, fast rendering
* Node.js + Express.js → Handles backend logic efficiently
* MySQL → Structured data storage

### 10.3 Steps to Implement

* let up the Backend (Node.js + Express + MySQL)
* Create RESTful APIs for CRUD operations
* Build the Frontend using React.js
* Connect the React Frontend to the Node.js Backend
* Run and Test the Application

### 10.4 Code Implementation

#### 10.4.1 Backend - Node.js + Express.js (server.js)

**Install dependencies:**

**npm init -y**

**npm install express mysql cors body-parser**

##### 10.4.2 Create server.js to handle API requests:

**const express = require("express");**

**const mysql = require("mysql");**

**const cors = require("cors");**

**const bodyParser = require("body-parser");**

**const app = express();**

**app.use(cors());**

**app.use(bodyParser.json());**

**// Database connection**

**const db = mysql.createConnection({**

**host: "localhost",**

**user: "root",**

**password: "password",**

**database: "studentdb",**

**});**

**db.connect((err) => {**

**if (err) throw err;**

**console.log("Connected to MySQL Database!");**

**});**

**// API Routes**

**// 1️⃣ Get all students**

**app.get("/students", (req, res) => {**

**db.query("SELECT \* FROM students", (err, result) => {**

**if (err) throw err;**

**res.json(result);**

**});**

**});**

##### // Add a student

**app.post("/students", (req, res) => {**

**const { name, email } = req.body;**

**db.query("INSERT INTO students (name, email) VALUES (?, ?)", [name, email], (err, result) => {**

**if (err) throw err;**

**res.json({ message: "Student added successfully!" });**

**});**

**});**

##### // Start server

**app.listen(5000, () => {**

**console.log("Server running on port 5000");**

**});**

##### 10.4.3 Explanation:

* Connects to MySQL database.
* Implements GET and POST API routes for students.

### 10.5 Frontend - React.js (App.js)

##### 10.5.1 Create a React app:

**npx create-react-app student-app**

**cd student-app**

**npm install axios**

##### 10.5.2 Modify src/App.js to display student data:

**import React, { useEffect, useState } from "react";**

**import axios from "axios";**

**function App() {**

**const [students, setStudents] = useState([]);**

**const [name, setName] = useState("");**

**const [email, setEmail] = useState("");**

##### // Fetch students from backend

**useEffect(() => {**

**axios.get("http://localhost:5000/students").then((response) => {**

**setStudents(response.data);**

**});**

**}, []);**

##### // Add a student

**const addStudent = () => {**

**axios.post("http://localhost:5000/students", { name, email }).then(() => {**

**alert("Student added!");**

**window.location.reload();**

**});**

**};**

**return (**

**<div>**

**<h2>Student Management</h2>**

**<input**

**type="text"**

**placeholder="Name"**

**onChange={(e) => setName(e.target.value)}**

**/>**

**<input**

**type="email"**

**placeholder="Email"**

**onChange={(e) => setEmail(e.target.value)}**

**/>**

**<button onClick={addStudent}>Add Student</button>**

**<h3>Student List</h3>**

**<ul>**

**{students.map((student) => (**

**<li key={student.id}>**

**{student.name} - {student.email}**

**</li>**

**))}**

**</ul>**

**</div>**

**);**

**}**

**export default App;**

##### 10.5.6 Explanation:

* Fetches student data from backend API.
* Allows adding new students using a form.
* Displays the student list.

### 10.6 Running the Application

#### 10.6.1 Start the Backend

##### node server.js

##### 10.6.2 Start the React Frontend

**cd student-app**

**npm start**

##### 10.6.3 Expected Output:

* A form to add students
* A list of students fetched from the backend

### 10.6 Testing the API using Postman

| **Operation** | **HTTP Method** | **URL** | **Body (if needed)** |
| --- | --- | --- | --- |
| **Get all students** | **GET** | **http://localhost:3000/students** | **None** |
| **Add student** | **POST** | **http://localhost:3000/students** | **{ "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" }** |

#### 10.6.1 Use Postman to test API endpoints.

##### Expected Output (GET request):

**[**

**{ "id": 1, "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" }**

**]**

### 10.8 Conclusion

* Developed a full-stack React.js + Node.js web application.
* Connected frontend and backend using REST API.
* Used MySQL database for data storage.
* Implemented CRUD operations.

### 10.8 Viva Questions

1. What is React.js, and why is it used?
2. What is the purpose of Express.js in backend development?
3. How does React fetch data from the backend?
4. How do you handle POST requests in React?
5. What are the benefits of full-stack development?

### 10.9 Final Steps **Run the full-stack app and take screenshots. Ensure the database operations work properly.**

**Would you like any modifications or additional explanations?**

**Here’s a structured version of Experiment 11: Building a TODO Application using React.js and Django in simple language:**

# Program 11: Building a TODO Application using React.js and Django

## 11.1 Aim

**To develop a full-stack TODO application using:**

* **React.js (Frontend) → Handles user interface**
* **Django + Django REST Framework (Backend) → Handles API requests**
* **SQLite (Database) → Stores task data**

## 11.2 Overview

What is a TODO Application?

**A TODO application allows users to:  
 Add tasks  
 View tasks  
 Mark tasks as completed  
 Delete tasks**

Why Use React.js and Django?

* **React.js → Builds an interactive UI.**
* **Django + DRF → Provides a robust backend API.**
* **SQLite → A lightweight database for storing tasks.**

## 11.3 Steps to Implement

1. Set up the Backend (Django + Django REST Framework + SQLite)
2. Create API Endpoints for CRUD operations
3. Build the Frontend using React.js
4. Connect the React Frontend to the Django Backend
5. Run and Test the Application

## 11.4 Code Implementation

### Backend - Django + DRF (API)

**Install Django and Django REST Framework:**

**pip install django djangorestframework cors-headers**

**📌 Create a new Django project:**

**django-admin startproject todo\_backend**

**cd todo\_backend**

**django-admin startapp todo**

**Modify settings.py to add installed apps:**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'rest\_framework',

'corsheaders',

'todo',

]

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

**CORS\_ALLOW\_ALL\_ORIGINS = True**

##### Define the Task Model (models.py):

from django.db import models

class Task(models.Model):

title = models.CharField(max\_length=200)

completed = models.BooleanField(default=False)

def \_\_str\_\_(self):

return self.title

##### Create API Serializer (serializers.py):

from rest\_framework import serializers

from .models import Task

class TaskSerializer(serializers.ModelSerializer):

class Meta:

model = Task

**fields = '\_\_all\_\_'**

##### Define API Views (views.py):

from rest\_framework import viewsets

from .models import Task

from .serializers import TaskSerializer

class TaskViewSet(viewsets.ModelViewSet):

queryset = Task.objects.all().order\_by('-id')

serializer\_class = TaskSerializer

##### Set up URL Routing (urls.py in todo app):

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import TaskViewSet

router = DefaultRouter()

router.register(r'tasks', TaskViewSet)

urlpatterns = [

path('api/', include(router.urls)),

]

#### Migrate Database and Run Server:

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

API is now accessible at: http://127.0.0.1:8000/api/tasks/

### Frontend - React.js (TODO App UI)

##### **Create a React app:**

npx create-react-app todo-frontend

cd todo-frontend

npm install axios

##### Modify src/App.js to interact with the API:

import React, { useState, useEffect } from "react";

import axios from "axios";

const API\_URL = "http://127.0.0.1:8000/api/tasks/";

function App() {

const [tasks, setTasks] = useState([]);

const [title, setTitle] = useState("");

##### // Fetch tasks from backend

useEffect(() => {

axios.get(API\_URL).then((response) => {

setTasks(response.data);

});

}, []);

##### // Add a new task

const addTask = () => {

axios.post(API\_URL, { title, completed: false })

.then(() => {

alert("Task added!");

window.location.reload();

});

};

##### // Toggle task completion

const toggleComplete = (task) => {

axios.patch(`${API\_URL}${task.id}/`, { completed: !task.completed })

.then(() => {

window.location.reload();

});

};

##### // Delete a task

const deleteTask = (taskId) => {

axios.delete(`${API\_URL}${taskId}/`)

.then(() => {

window.location.reload();

});

};

return (

<div>

<h2>TODO Application</h2>

<input type="text" placeholder="Task Title" onChange={(e) => setTitle(e.target.value)} />

<button onClick={addTask}>Add Task</button>

<h3>Task List</h3>

<ul>

{tasks.map((task) => (

<li key={task.id}>

<span style={{ textDecoration: task.completed ? "line-through" : "none" }}>

{task.title}

</span>

<button onClick={() => toggleComplete(task)}>

{task.completed ? "Undo" : "Complete"}

</button>

<button onClick={() => deleteTask(task.id)}>Delete</button>

</li>

))}

</ul>

</div>

);

}

export default App;

##### Explanation:

* Fetches task data from Django API.
* Allows users to add, complete, and delete tasks.

## 11.5 Running the Application

##### Start the Django Backend

python manage.py runserver

##### Start the React Frontend

cd todo-frontend

npm start

##### Expected Output:

* **A form to add new tasks**
* **A list of tasks with complete and delete options**

## 11.6 Testing the API using Postman

| **Operation** | **HTTP Method** | **URL** | **Body (if needed)** |
| --- | --- | --- | --- |
| Get all tasks | GET | http://127.0.0.1:8000/api/tasks/ | None |
| Add a task | POST | http://127.0.0.1:8000/api/tasks/ | { "title": "Buy groceries", "completed": false } |
| Update a task | PATCH | http://127.0.0.1:8000/api/tasks/1/ | { "completed": true } |
| Delete a task | DELETE | http://127.0.0.1:8000/api/tasks/1/ | None |

##### Expected API Response (GET request)

[

{ "id": 1, "title": "Buy groceries", "completed": false },

{ "id": 2, "title": "Complete assignment", "completed": true }

]

## 11.7 Conclusion

* Developed a React.js + Django TODO application.
* Implemented CRUD operations.
* Integrated React frontend with Django REST API.

## 11.8 Viva Questions

1. What is Django REST Framework (DRF)?
2. How does React fetch data from a REST API?
3. What is CORS, and why is it needed?
4. How do you update data using Django REST API?
5. What happens when an API request fails?

## Final Steps

* Copy this structured format into your lab manual.
* Run and test the TODO app.
* Ensure database operations work correctly.

# Experiment 12: Weather API Integration using React.js

### 12.1 Aim

To develop a Weather Forecast Application using:

* React.js (Frontend UI)
* OpenWeatherMap API (Data Source)

### 12.2 Overview

### What is a Weather API?

A Weather API provides real-time weather information such as:

* Temperature
* Weather conditions (Sunny, Rainy, Cloudy, etc.)
* Humidity and Wind Speed

### 

### Why Use React.js?

* Dynamic UI → Updates weather info instantly.
* Fetch API/axios → Retrieves weather data easily.
* Component-based → Modular and reusable code.

### 12.3 Steps to Implement

1. Get an API Key from OpenWeatherMap
2. Set up a React.js project
3. Fetch Weather Data from OpenWeatherMap API
4. Display Weather Information on the UI
5. Allow Users to Search for Any City’s Weather

### 12.4 Code Implementation

### Get an API Key from OpenWeatherMap

* Go to<https://openweathermap.org/>
* Sign up and get a free API key
* **API URL format:** https://api.openweathermap.org/data/2.5/weather?q=CityName&appid=YOUR\_API\_KEY&units=metric

### Set up React.js Project

Create a new React app:

npx create-react-app weather-app

cd weather-app

npm install axios

### Fetch Weather Data in App.js

##### Modify src/App.js:

import React, { useState } from "react";

import axios from "axios";

const API\_KEY = "YOUR\_API\_KEY"; // Replace with your actual API key

const API\_URL = "https://api.openweathermap.org/data/2.5/weather";

function App() {

const [city, setCity] = useState("");

const [weather, setWeather] = useState(null);

// Fetch Weather Data

const fetchWeather = async () => {

if (city === "") {

alert("Please enter a city name!");

return;

}

try {

const response = await axios.get(API\_URL, {

params: { q: city, appid: API\_KEY, units: "metric" },

});

setWeather(response.data);

} catch (error) {

alert("City not found! Please try again.");

}

};

return (

<div style={{ textAlign: "center", padding: "20px" }}>

<h2>Weather Forecast App</h2>

<input

type="text"

placeholder="Enter city name"

onChange={(e) => setCity(e.target.value)}

/>

<button onClick={fetchWeather}>Get Weather</button>

{weather && (

<div style={{ marginTop: "20px" }}>

<h3>{weather.name}, {weather.sys.country}</h3>

<p>Temperature: {weather.main.temp}°C</p>

<p>Condition: {weather.weather[0].description}</p>

<p>Humidity: {weather.main.humidity}%</p>

<p>Wind Speed: {weather.wind.speed} m/s</p>

</div>

)}

</div>

);

}

export default App;

##### Explanation:

* axios.get(API\_URL, { params }) → Fetches weather data.
* setWeather(response.data) → Updates UI dynamically.
* Handles errors if the city is not found.

### 12.5 Running the Application

##### 12.5.1 Start React App

**npm start**

#### Expected Output:

* A search bar for city input.
* Displays weather details when a city is entered.

### 12.6 Testing the API using Postman

| **Operation** | **HTTP Method** | **URL** |
| --- | --- | --- |
| Get Weather Data | GET | https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR\_API\_KEY&units=metric |

### 

### **Expected API Response:**

**{**

**"name": "London",**

**"sys": { "country": "GB" },**

**"main": { "temp": 10.5, "humidity": 85 },**

**"weather": [{ "description": "cloudy" }],**

**"wind": { "speed": 3.5 }**

**}**

### 12.7 Conclusion

* Integrated OpenWeatherMap API with React.js.
* Implemented API fetching using axios.
* Created a dynamic UI to display weather data.

### 12.8 Viva Questions

1. What is an API key, and why is it needed?
2. How does axios work in React?
3. What does "units=metric" do in the API request?
4. How can you handle API errors in React?
5. How can this project be improved (e.g., adding icons, improving UI)?

### 

### 12.9 Final Steps

1. Copy this structured format into your lab manual.
2. Run and test the Weather App.
3. Check API responses for different cities.

## 

## 

## 

## Experiment 13: Student Management System using React.js & Django

### 13.1 Aim : To develop a **Student Management System** using

* **React.js (Frontend UI)**
* **Django (Backend API)**
* **MySQL (Database)**

### 13.2 Overview

### What is a Student Management System?

A Student Management System (SMS) allows:

* Adding new students
* Viewing student details
* updating student information
* deleting student records

### Why Use React.js & Django?

* **React.js** → Creates a fast, interactive UI.
* **Django** → Handles backend logic & database operations.
* **REST API** → Connects React frontend with Django backend.

### 13.3 Steps to Implement

* Set up Django Backend with MySQL
* Create REST API using Django REST Framework
* Develop a React.js Frontend
* Connect React with Django API
* Perform CRUD operations (Create, Read, Update, Delete)

### 13.4 Code Implementation

## 13.4.1 Set up Django Backend with MySQL

##### Install Django & Django REST Framework

pip install django django rest framework mysqlclient

##### Create a Django project and app

django-admin startproject student\_management

cd student\_management

django-admin startapp students

### Define Student Model (models.py)

##### Modify students/models.py:

from django.db import models

class Student(models.Model):

name = models.CharField(max\_length=100)

email = models.EmailField(unique=True)

age = models.IntegerField()

course = models.CharField(max\_length=50)

def \_\_str\_\_(self):

return self.name

##### Explanation:

* Defines **Student model** with fields → name, email, age, course.
* \_\_str\_\_ → Returns student name when printed.

### Create API using Django REST Framework

##### **Modify students/serializers.py:**

from rest\_framework import serializers

from .models import Student

class StudentSerializer(serializers.ModelSerializer):

class Meta:

model = Student

fields = '\_\_all\_\_'

##### **Modify students/views.py:**

from rest\_framework import viewsets

from .models import Student

from .serializers import StudentSerializer

class StudentViewSet(viewsets.ModelViewSet):

queryset = Student.objects.all()

serializer\_class = StudentSerializer

##### **Modify students/urls.py:**

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import StudentViewSet

router = DefaultRouter()

router.register(r'students', StudentViewSet)

urlpatterns = [

path('api/', include(router.urls)),

]

##### **Modify student\_management/settings.py:**

INSTALLED\_APPS = [

'rest\_framework',

'students',

]

##### **Apply Migrations**

python manage.py makemigrations

python manage.py migrate

##### **Run Django Server**

python manage.py runserver

##### **API Endpoint:**

* http://127.0.0.1:8000/api/students/ (Lists all students)
* http://127.0.0.1:8000/api/students/<id>/ (CRUD operations on a student)

### Set up React.js Frontend

**Create React App:**

npx create-react-app student-app

cd student-app

npm install axios react-router-dom

### **Modify src/App.js:**

import React, { useState, useEffect } from "react";

import axios from "axios";

const API\_URL = "http://127.0.0.1:8000/api/students/";

function App() {

const [students, setStudents] = useState([]);

const [name, setName] = useState("");

const [email, setEmail] = useState("");

const [age, setAge] = useState("");

const [course, setCourse] = useState("");

useEffect(() => {

fetchStudents();

}, []);

const fetchStudents = async () => {

const response = await axios.get(API\_URL);

setStudents(response.data);

};

const addStudent = async () => {

await axios.post(API\_URL, { name, email, age, course });

fetchStudents();

};

return (

<div style={{ textAlign: "center", padding: "20px" }}>

<h2>Student Management System</h2>

<input type="text" placeholder="Name" onChange={(e) => setName(e.target.value)} />

<input type="email" placeholder="Email" onChange={(e) => setEmail(e.target.value)} />

<input type="number" placeholder="Age" onChange={(e) => setAge(e.target.value)} />

<input type="text" placeholder="Course" onChange={(e) => setCourse(e.target.value)} />

<button onClick={addStudent}>Add Student</button>

<h3>Students List</h3>

<ul>

{students.map((student) => (

<li key={student.id}>{student.name} - {student.course}</li>

))}

</ul>

</div>

);

}

export default App;

**Explanation:**

* Fetches student data from Django API.
* Allows adding new students.
* Displays student list dynamically.

### Running the Application

##### Start Django Backend

python manage.py runserver

##### Start React Frontend npm start

#### Expected Output:

* **Form** to add student details.
* **List of students** retrieved from MySQL.

### 13.5 Testing API using Postman

| **Operation** | **HTTP Method** | **URL** | **Body (if needed)** |
| --- | --- | --- | --- |
| Get all students | GET | http://localhost:3000/students | None |
| Add student | POST | http://localhost:3000/students | { "name": "Satish", "email": "satish.pallyvela@aceec.ac.in" } |
| Update student | PUT | http://localhost:3000/students/1 | { "name": "Satish Pallyvela", "email": "satish.pallyvela@aceec.ac.in" } |
| Delete student | DELETE | http://localhost:3000/students/1 | None |

#### 

**Expected JSON Response:**

**[**

**{**

**"id": 1,**

**"name": "Satish",**

**"email": "Satish@example.com",**

**"age": 20,**

**"course": "Computer Science"**

**}**

**]**

#### 13.6 Conclusion

1. Created **Student Management System** using **React & Django**.
2. Implemented **REST API** for CRUD operations.
3. Connected **React frontend** with **Django backend**.

#### 13.7 Viva Questions

1. What is Django REST Framework, and why is it used?
2. How does React fetch API data?
3. Explain CRUD operations in Django.
4. How is MySQL integrated with Django?
5. What is the role of axios in React?

#### 13.8 Final Steps

1. Test the Student Management System.
2. Improve UI with Bootstrap or Material-UI.

# 

# 

# Program 14: Student Management System using Django

### 14.1 Aim:

To develop a **Student Management System** using Django, implementing:

* User Registration & Login
* Navigation (Contact & About Pages)
* Django Routing for Page Navigation

### 14.2 Overview

#### 14.2.1 What is a Student Management System?

A **Student Management System** is a web application that allows users to:

* **Register** a new student account
* **Login** using valid credentials
* **Navigate to Contact & About pages** using Django Routing

##### 14.2.2 Why Use Django?

* **Built-in User Authentication** → Secure registration & login.
* **Django Routing System** → Efficient page navigation.

**Django ORM** → Easy database handling.

#### 14.3 Steps to Implement:

* Set Up Django Project & Application
* Configure URL Routing for Navigation
* Implement User Registration & Login
* Create Contact & About Pages
* Test the Application

---

### 14.4 Code Implementation

#### Set Up Django Project & Application

#### Install Django

**pip install django**

##### Create Django Project & App

**django-admin startproject student\_management**

**cd student\_management**

**django-admin startapp students**

##### **Add `students` to `INSTALLED\_APPS` in `settings.py`**

**INSTALLED\_APPS = [**

**'django.contrib.admin',**

**'django.contrib.auth',**

**'django.contrib.contenttypes',**

**'django.contrib.sessions',**

**'django.contrib.messages',**

**'django.contrib.staticfiles',**

**'students',**

**]**

### Configure URL Routing for Navigation

##### **Modify `student\_management/urls.py`:**

**from django.contrib import admin**

**from django.urls import path**

**from students import views**

**urlpatterns = [**

**path('admin/', admin.site.urls),**

**path('', views.home, name='home'),**

**path('register/', views.register, name='register'),**

**path('login/', views.login\_user, name='login'),**

**path('contact/', views.contact, name='contact'),**

**path('about/', views.about, name='about'),**

**path('logout/', views.logout\_user, name='logout'),**

**]**

### Implement User Registration & Login

##### **Modify `students/models.py`:**

**from django.db import models**

**from django.contrib.auth.models import User**

**class StudentProfile(models.Model):**

**user = models.OneToOneField(User, on\_delete=models.CASCADE)**

**phone = models.CharField(max\_length=15)**

**address = models.TextField()**

##### Apply Migrations

**python manage.py makemigrations**

**python manage.py migrate**

##### Modify `students/forms.py`:

**from django import forms**

**from django.contrib.auth.models import User**

**from .models import StudentProfile**

**class RegisterForm(forms.ModelForm):**

**password = forms.CharField(widget=forms.PasswordInput)**

**class Meta:**

**model = User**

**fields = ['username', 'email', 'password']**

**class StudentProfileForm(forms.ModelForm):**

**class Meta:**

**model = StudentProfile**

**fields = ['phone', 'address']**

##### Modify `students/views.py`:

**from django.shortcuts import render, redirect**

**from django.contrib.auth import authenticate, login, logout**

**from django.contrib.auth.models import User**

**from .forms import RegisterForm, StudentProfileForm**

**from .models import StudentProfile**

**def home(request):**

**return render(request, 'students/home.html')**

**def register(request):**

**if request.method == 'POST':**

**user\_form = RegisterForm(request.POST)**

**profile\_form = StudentProfileForm(request.POST)**

**if user\_form.is\_valid() and profile\_form.is\_valid():**

**user = user\_form.save(commit=False)**

**user.set\_password(user\_form.cleaned\_data['password'])**

**user.save()**

**profile = profile\_form.save(commit=False)**

**profile.user = user**

**profile.save()**

**return redirect('login')**

**else:**

**user\_form = RegisterForm()**

**profile\_form = StudentProfileForm()**

**return render(request, 'students/register.html', {'user\_form': user\_form, 'profile\_form': profile\_form})**

**def login\_user(request):**

**if request.method == 'POST':**

**username = request.POST['username']**

**password = request.POST['password']**

**user = authenticate(request, username=username, password=password)**

**if user:**

**login(request, user)**

**return redirect('home')**

**return render(request, 'students/login.html')**

**def logout\_user(request):**

**logout(request)**

**return redirect('home')**

**def contact(request):**

**return render(request, 'students/contact.html')**

**def about(request):**

**return render(request, 'students/about.html')**

### Create Contact & About Pages

##### Create `templates/students/home.html`

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Home</title>**

**</head>**

**<body>**

**<h2>Welcome to Student Management System</h2>**

**<a href="{% url 'register' %}">Register</a> |**

**<a href="{% url 'login' %}">Login</a> |**

**<a href="{% url 'contact' %}">Contact</a> |**

**<a href="{% url 'about' %}">About</a>**

**</body>**

**</html>**

##### **Create `templates/students/register.html`**

**<form method="post">**

**{% csrf\_token %}**

**{{ user\_form.as\_p }}**

**{{ profile\_form.as\_p }}**

**<button type="submit">Register</button>**

**</form>**

##### Create `templates/students/login.html`

**<form method="post">**

**{% csrf\_token %}**

**<input type="text" name="username" placeholder="Username">**

**<input type="password" name="password" placeholder="Password">**

**<button type="submit">Login</button>**

**</form>**

##### Create `templates/students/contact.html`

**<h2>Contact Page</h2>**

**<p>Email: admin@university.com</p>**

**<p>Phone: +123456789</p>**

##### Create `templates/students/about.html`

**<h2>About Page</h2>**

**<p>Student Management System built using Django.</p>**

#### Test the Application

##### Run the Django Server

**python manage.py runserver**

##### 14.4.1 Access the application in a browser:

* Home Page: `http://127.0.0.1:8000/`
* Registration Page: `http://127.0.0.1:8000/register/`
* Login Page: `http://127.0.0.1:8000/login/`
* Contact Page: `http://127.0.0.1:8000/contact/`
* About Page: `http://127.0.0.1:8000/about/`

#### 14.5 Expected Output

1. User successfully registers and logs in.
2. Navigation works between Home, Contact, and About pages.
3. Django handles routing and authentication properly.

#### 14.6 Viva Questions

1. What is Django routing, and why is it important?
2. How does Django handle user authentication?
3. What is the difference between `redirect()` and `render()` in Django?
4. How do you create a one-to-one relationship in Django models?
5. How can you add custom validation in Django forms?

# Program 15: Weather Application using Django & OpenWeatherMap API

### Objective

To develop a Weather Application using Django, OpenWeatherMap API, and Chart.js, which fetches real-time weather data and stores historical weather records in a database.

## Software & Tools Required

* Python (latest version) - [Download](https://www.python.org/downloads/)
* Django Framework
* VS Code (or any Python-supported IDE)
* OpenWeatherMap API Key
* SQLite (default database for Django)

## Step 1: Install Required Packages

1. Open **VS Code** and go to the terminal.

Install Django and requests package using the command:  
 pip install django requests

## Step 2: Create Django Project and App

### Create Django Project

django-admin startproject Django\_1

cd Django\_1

### Create Django App

django-admin startapp weather

## Step 3: Add weather App to Django Settings

Open experiment15/settings.py and add 'weather' in INSTALLED\_APPS:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'weather', # Add this

]

## Step 4: Get OpenWeatherMap API Key

1. Sign up at [OpenWeatherMap](https://home.openweathermap.org/users/sign_up).
2. Copy the API Key and add it to settings.py:

OPENWEATHER\_API\_KEY = "your\_api\_key\_here"

## Step 5: Create a Weather Model

Modify weather/models.py:

from django.db import models

class WeatherData(models.Model):

city = models.CharField(max\_length=100)

temperature = models.FloatField()

humidity = models.IntegerField()

description = models.CharField(max\_length=255)

date = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"{self.city} - {self.temperature}°C ({self.date})"

### Run Migrations

python manage.py makemigrations

python manage.py migrate

## Step 6: Fetch Weather Data from API

Modify weather/views.py:

import requests

from django.shortcuts import render

from django.conf import settings

from .models import WeatherData

def get\_weather(city):

api\_key = settings.OPENWEATHER\_API\_KEY

url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}&units=metric"

response = requests.get(url)

if response.status\_code == 200:

data = response.json()

return {

'city': city,

'temperature': data['main']['temp'],

'humidity': data['main']['humidity'],

'description': data['weather'][0]['description'],

'icon': data['weather'][0]['icon'],

}

return None

def weather\_view(request):

city = request.GET.get('city', 'New York')

weather = get\_weather(city)

if weather:

WeatherData.objects.create(

city=weather['city'],

temperature=weather['temperature'],

humidity=weather['humidity'],

description=weather['description']

)

history = WeatherData.objects.filter(city=city).order\_by('-date')[:10]

return render(request, 'weather/weather.html', {'weather': weather, 'history': history})

## Step 7: Configure URLs

### Modify weather/urls.py

from django.urls import path

from .views import weather\_view

urlpatterns = [

path('', weather\_view, name='weather'),

]

### Modify experiment15/urls.py

from django.urls import path, include

urlpatterns = [

path('', include('weather.urls')),

path('admin/', admin.site.urls),

]

## Step 8: Create HTML Page

Create weather/templates/weather/weather.html:

<!DOCTYPE html>

<html>

<head>

<title>Weather App</title>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>

<h2>Weather Information</h2>

<form method="GET">

<input type="text" name="city" placeholder="Enter city" required>

<button type="submit">Get Weather</button>

</form>

{% if weather %}

<h3>Weather in {{ weather.city }}</h3>

<p>Temperature: {{ weather.temperature }}°C</p>

<p>Humidity: {{ weather.humidity }}%</p>

<p>Description: {{ weather.description }}</p>

<img src="http://openweathermap.org/img/w/{{ weather.icon }}.png" alt="Weather icon">

{% endif %}

<h2>Historical Weather Data</h2>

<canvas id="weatherChart"></canvas>

<script>

var ctx = document.getElementById('weatherChart').getContext('2d');

var weatherChart = new Chart(ctx, {

type: 'line',

data: {

labels: [{% for record in history %}"{{ record.date|date:'M d, H:i' }}",{% endfor %}],

datasets: [{

label: 'Temperature (°C)',

data: [{% for record in history %}{{ record.temperature }},{% endfor %}],

borderColor: 'blue',

borderWidth: 2,

fill: false

}]

},

options: {

responsive: true,

scales: {

x: { display: true },

y: { display: true }

}

}

});

</script>

</body>

</html>

## Step 9: Run the Django Server

Run the following command in the terminal:

python manage.py runserver

Now, open [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/) in a web browser.

### Conclusion

In this experiment, we successfully built a Weather Application using Django, integrated OpenWeatherMap API to fetch real-time weather data, and used Chart.js to display historical weather data. This experiment helps students understand Django models, views, templates, API integration, and database operations.

# Experiment 16: TODO Application using Django & GitHub Deployment

### 16.1 Aim :

To develop a **Django TODO Application** that:

* Allows users to **add, update, delete, and mark tasks as completed.**
* Uses **Django Models, Views, and Templates** for a structured approach.
* Deploys the project on **GitHub** for version control and collaboration.

### 16.2 Overview

#### 16.2.1 Why Use Django for a TODO App?

* Simple & Scalable: Django’s ORM makes database handling easy.
* Modular Approach: Supports MVC (Model-View-Controller)pattern.
* User Authentication: Secure login/logout for personalized TODO lists.

#### 16.2.2 Why Deploy on GitHub?

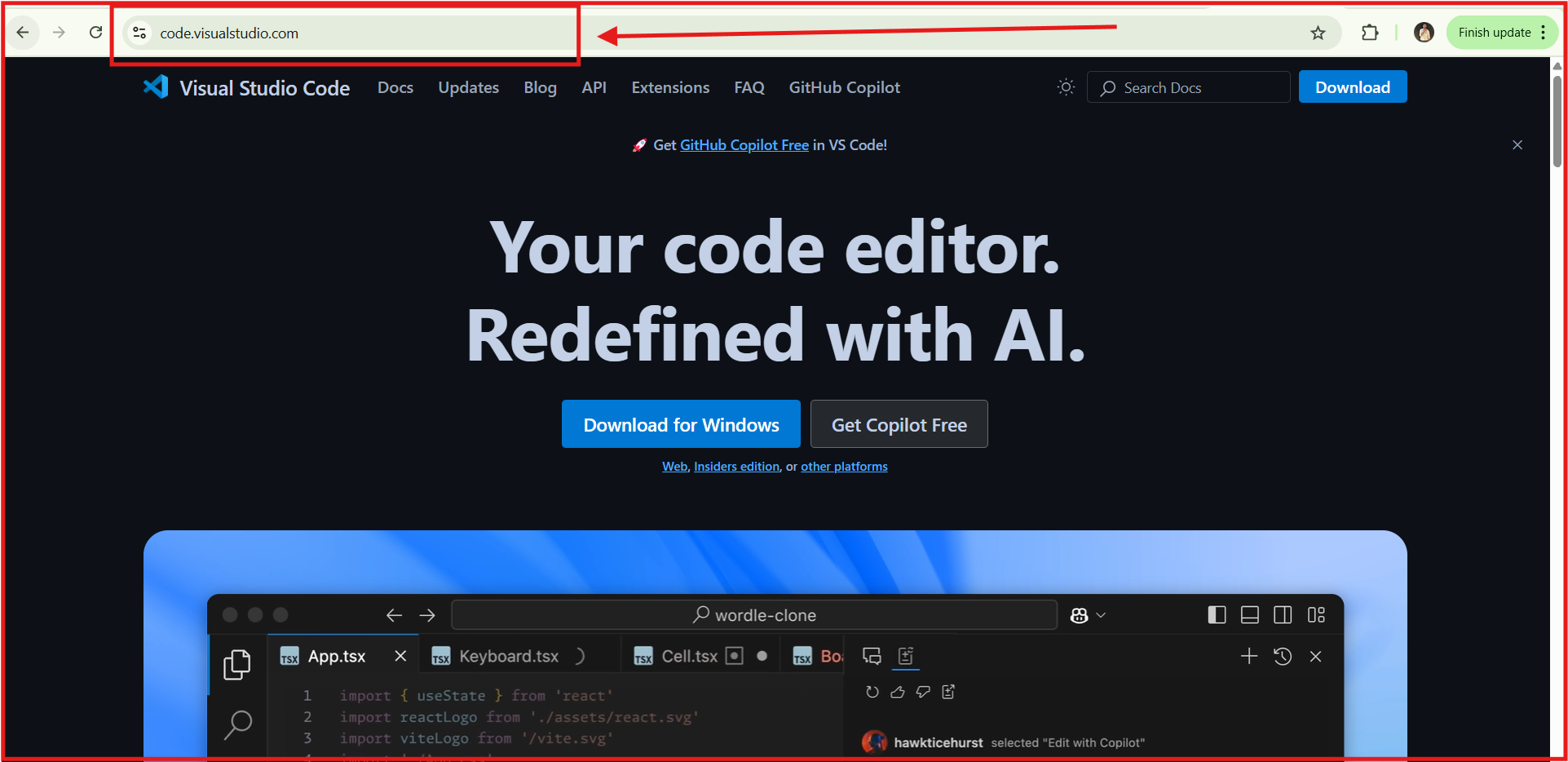
* **Version Control**: Track changes & collaborate.
* **Open Source Contribution**: Share your work.
* **Easy Deployment**: Later deploy on **Heroku/Vercel/Railway**.

### 16.3 Install Required Software:

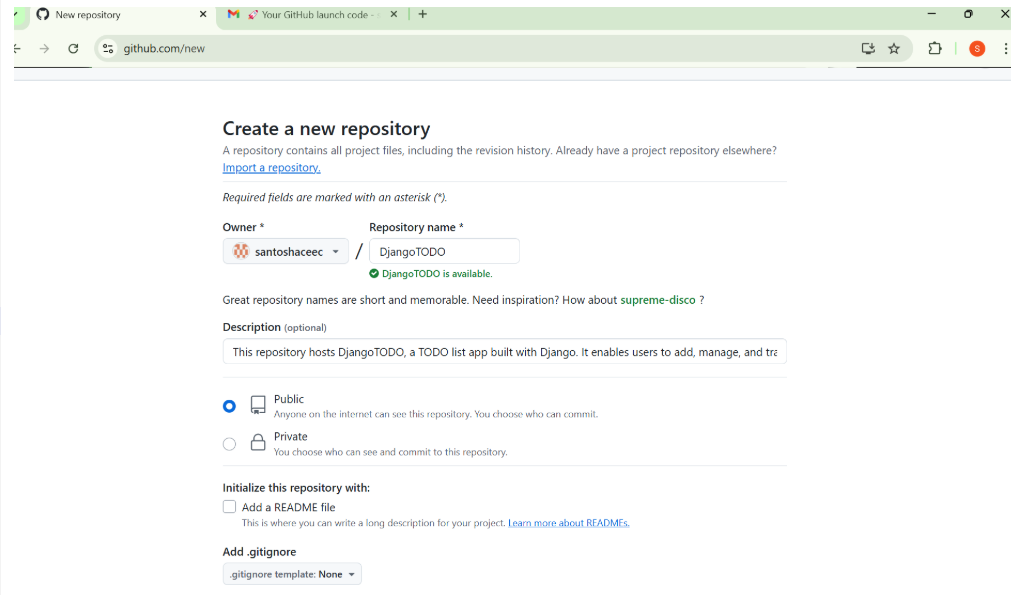
* Before starting, make sure you have the following installed:

#### 16.3.1 Python (Download from [python.org](<https://www.python.org/>)) :

#### 16.3.2 **Visual Studio Code** :

(Download from [code.visualstudio.com](https://code.visualstudio.com/))  


#### 16.3.4 Git (Download from [git-scm.com](https://git-scm.com/))



### 16.4 Steps to Implement

* Set Up Django Project & App
* Create Models for Task Management
* Implement Views & Templates
* Handle User Authentication (Login/Logout)
* Deploy the Project on GitHub

### 16.5 Code Implementation

#### 16.5.1 Install Django & Create Project

**pip install django**

**django-admin startproject todo\_project**

**cd todo\_project**

**django-admin startapp todo**

**\*\*Add `todo` to `INSTALLED\_APPS` in `settings.py`\*\* —> python**

**INSTALLED\_APPS = [**

**'django.contrib.admin',**

**'django.contrib.auth',**

**'django.contrib.contenttypes',**

**'django.contrib.sessions',**

**'django.contrib.messages',**

**'django.contrib.staticfiles',**

**'todo',**

**]**

**Apply Migrations & Run Server**

**python manage.py migrate**

**python manage.py runserver**

### 16.5.2 Create Models for Task Management

##### Modify `todo/models.py`

# Create your models here.

**from django.db import models**

**from django.contrib.auth.models import User**

**class Task(models.Model):**

**user = models.ForeignKey(User, on\_delete=models.CASCADE)**

**title = models.CharField(max\_length=255)**

**completed = models.BooleanField(default=False)**

**created\_at = models.DateTimeField(auto\_now\_add=True)**

**def \_\_str\_\_(self):**

**return self.title**

**Apply Migrations**

**python manage.py makemigrations**

**python manage.py migrate**

#### 16.5.3 Implement Views & Templates

##### **Modify `todo/views.py`** D:\College\ACE College\LABS\SDC LAB\Experiment16\DjangoTODO\todo\admin.py

**from django.shortcuts import render, redirect**

**from django.contrib.auth.decorators import login\_required**

**from .models import Task**

**from .forms import TaskForm**

**@login\_required**

**def task\_list(request):**

**tasks = Task.objects.filter(user=request.user)**

**form = TaskForm()**

**if request.method == "POST":**

**form = TaskForm(request.POST)**

**if form.is\_valid():**

**task = form.save(commit=False)**

**task.user = request.user**

task.save()

return redirect('task\_list')

**return render(request, 'todo/task\_list.html', {'tasks': tasks, 'form': form})**

**@login\_required**

**def delete\_task(request, task\_id):**

**Task.objects.get(id=task\_id, user=request.user).delete()**

**return redirect('task\_list')**

**@login\_required**

**def toggle\_task(request, task\_id):**

**task = Task.objects.get(id=task\_id, user=request.user)**

**task.completed = not task.completed**

**task.save()**

**return redirect('task\_list')**

#### Create Forms & Templates

**Modify `todo/forms.py`**

**from django import forms**

**from .models import Task**

**class TaskForm(forms.ModelForm):**

**class Meta:**

**model = Task**

**fields = ['title']**

**Modify `todo/urls.py`** → python

**from django.urls import path**

**from .views import task\_list, delete\_task, toggle\_task**

**urlpatterns = [**

**path('', task\_list, name='task\_list'),**

**path('delete/<int:task\_id>/', delete\_task, name='delete\_task'),**

**path('toggle/<int:task\_id>/', toggle\_task, name='toggle\_task'),**

**]**

#### **Create `templates/todo/task\_list.html`**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>TODO App</title>**

**</head>**

**<body>**

**<h2>TODO List</h2>**

**<form method="POST">**

**{% csrf\_token %}**

**{{ form.as\_p }}**

**<button type="submit">Add Task</button>**

**</form>**

**<ul>**

**{% for task in tasks %}**

**<li>**

**<span {% if task.completed %} style="text-decoration: line-through;" {% endif %}>**

**{{ task.title }}**

**</span>**

**<a href="{% url 'toggle\_task' task.id %}">✔</a>**

**<a href="{% url 'delete\_task' task.id %}">❌</a>**

**</li>**

**{% endfor %}**

**</ul>**

**<a href="{% url 'logout' %}">Logout</a>**

**</body>**

**</html>**

### Deploy the Project on GitHub

**Initialize Git Repository**

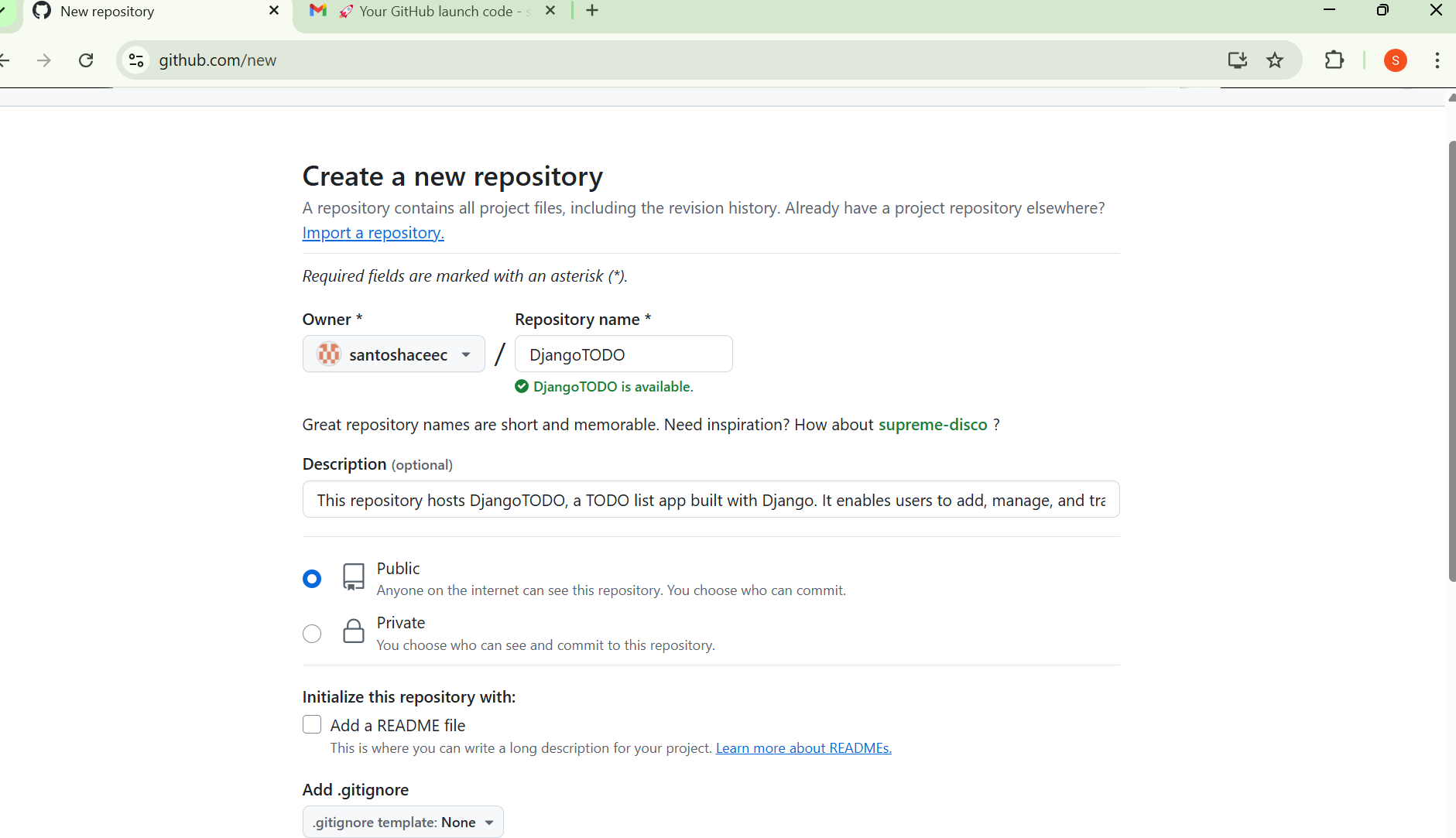
**git init**

**git add .**

**git commit -m "Initial commit - Django TODO App"**

### Create a GitHub Repository

1. Go to **\*\*[GitHub](**https://github.com/**)\*\*** and create a new repository.

****

#### 2. Run the following commands to push code:

**git remote add origin https://github.com/your-username/todo-app.git**

**git branch -M main**

**git push -u origin main**

#### Ignore Sensitive Files

**Create `.gitignore` and add:**

**.pyc**

**\_\_pycache\_\_/**

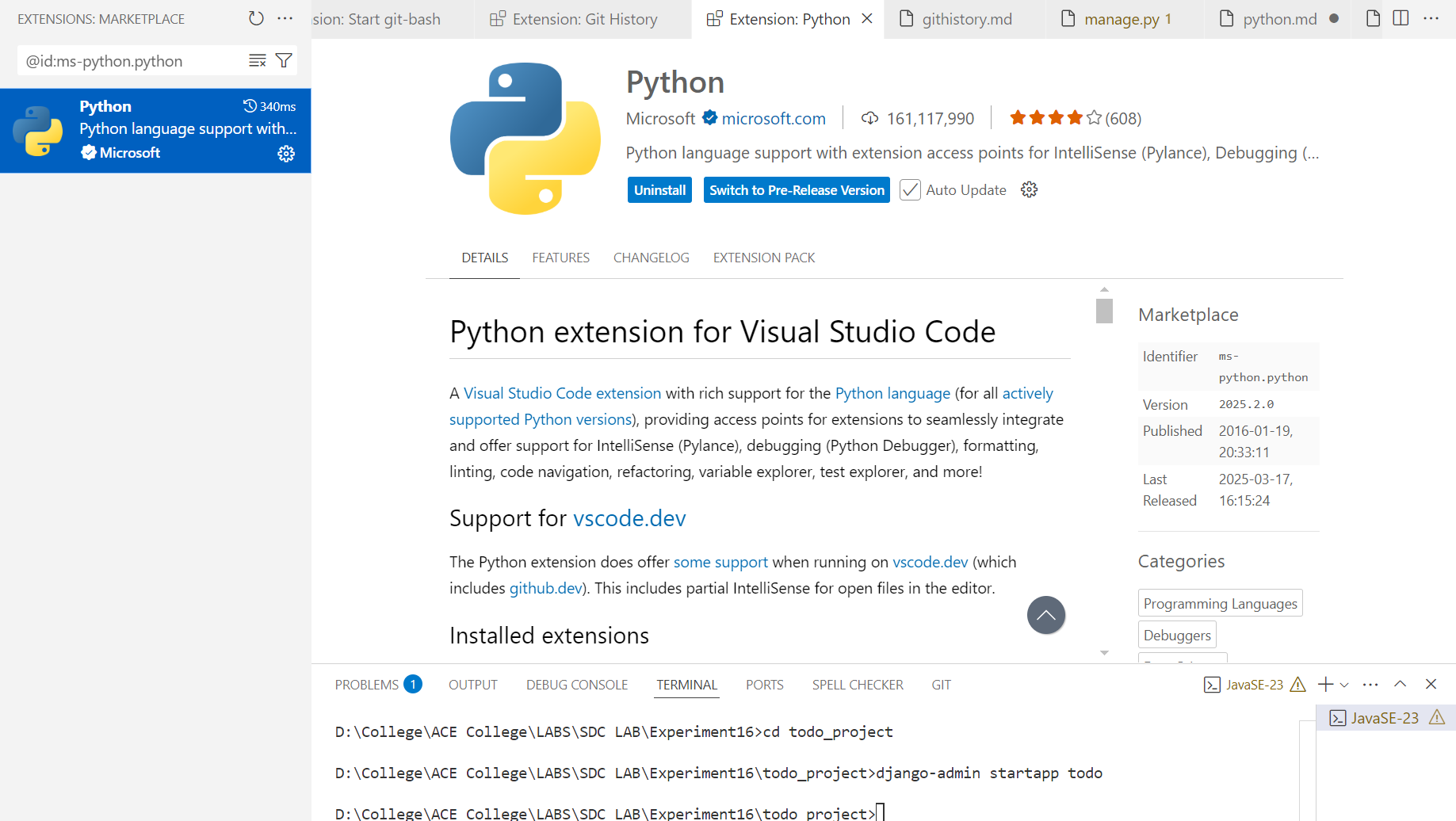
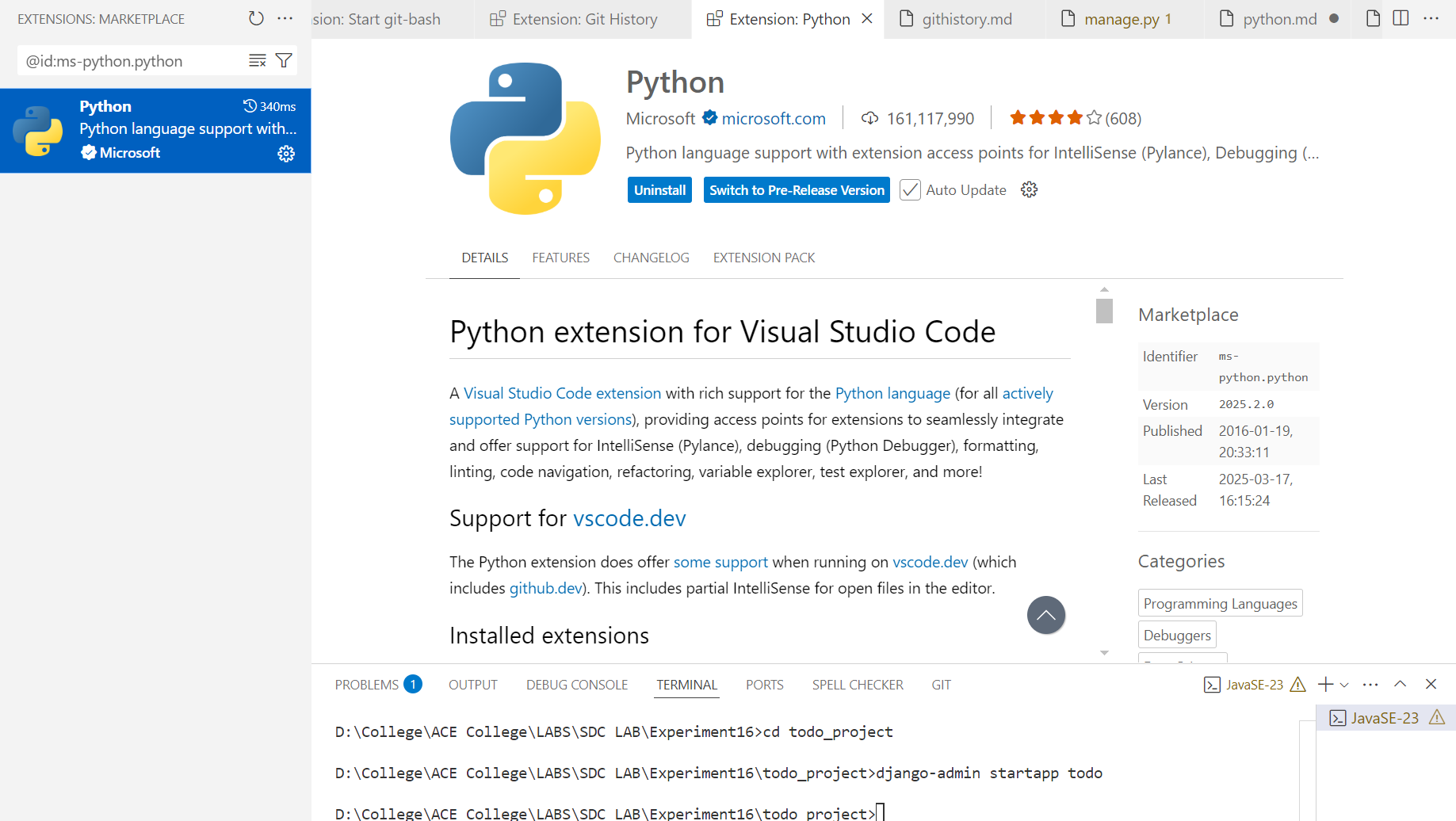
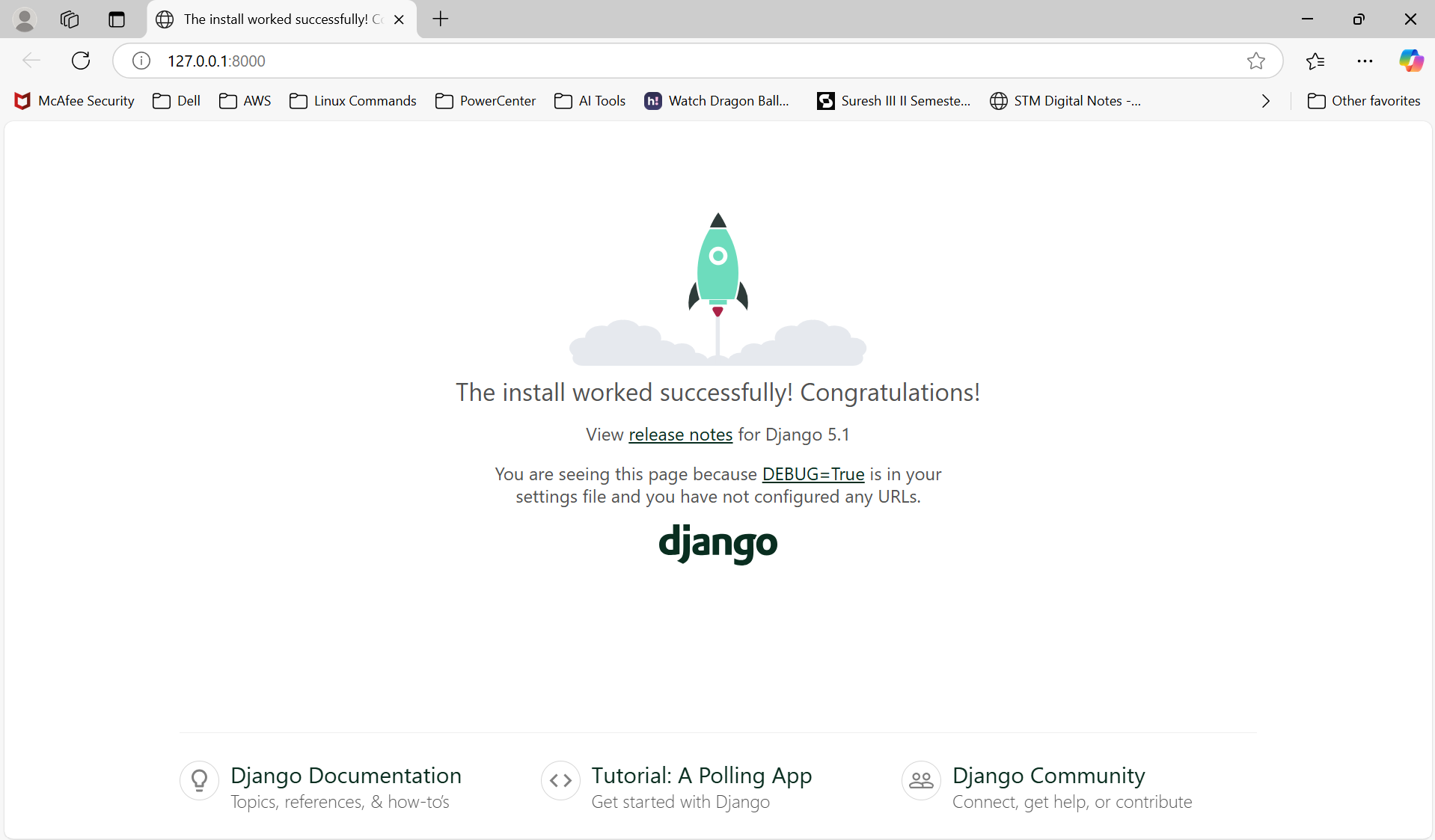
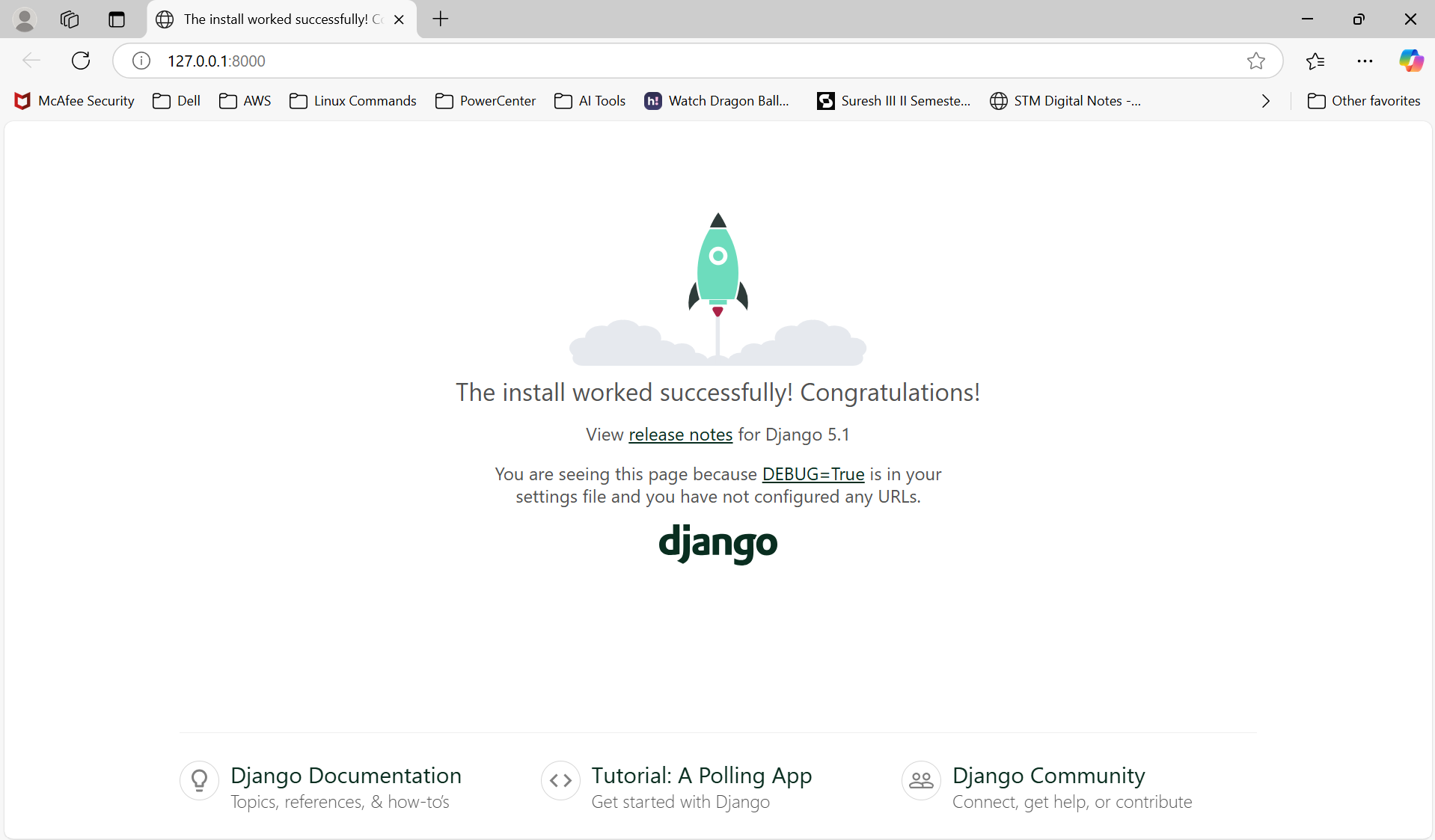
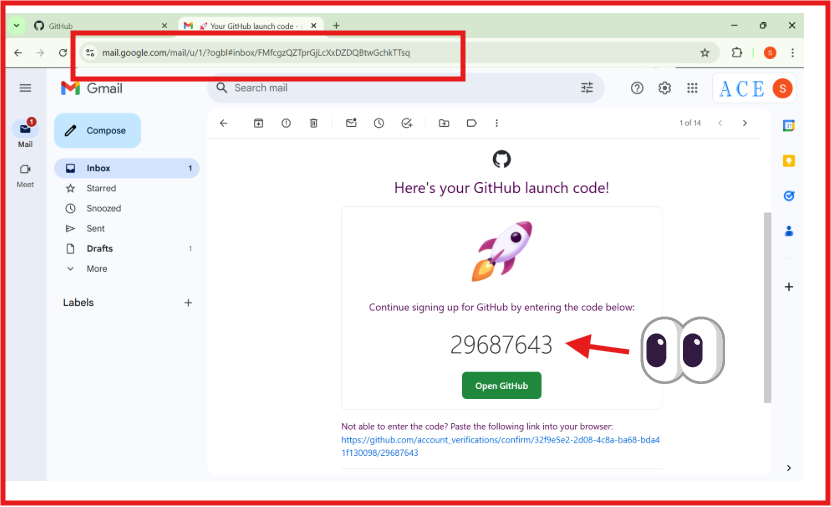
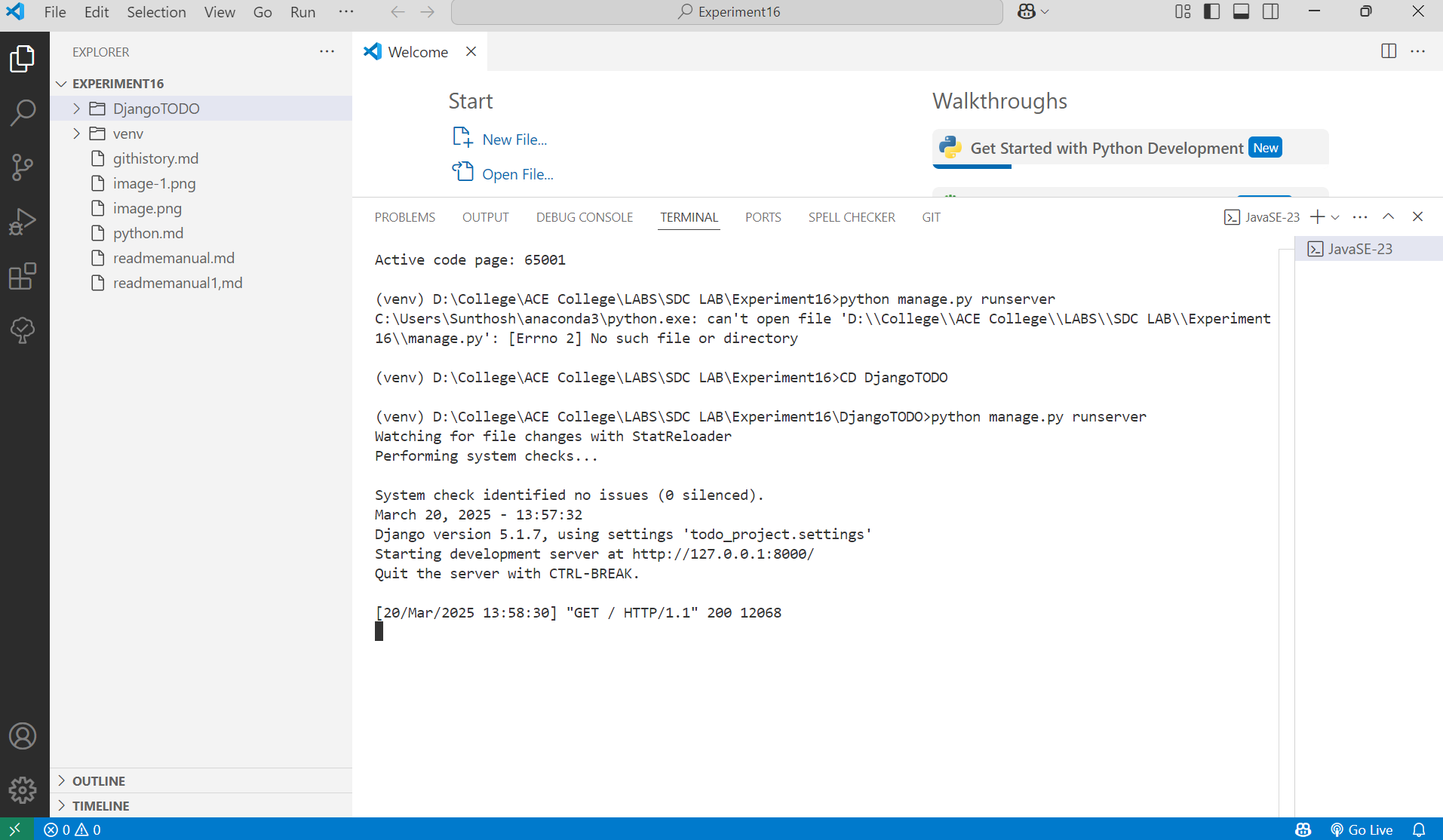
**db.sqlite3**

**.env**

### 

### 16.5 Expected Output

* Users can add, complete, and delete tasks.
* Tasks are stored in the database.
* App is deployed on GitHub for easy access.

### 16.6 Viva Questions

* What is Django ORM, and why is it used?
* How does Django handle authentication?
* What is the purpose of `ForeignKey` in Django models?
* Why use `.gitignore` while deploying on GitHub?
* How do you handle static files in Django?