MIT Art, Design and Technology University
MIT School of Computing, Pune

Department of Computer Science and Engineering

Lab File

Practical – Full Stack Development

Class - L.Y. (SEM-I), SMAD

Name of the Practical Coordinator

Prof. Rohini Bhosale

Prepared By
Gurneet Juneja

A.Y. 2025 – 2026 (SEM-I)

COURSE STRUCTURE

Full Stack Development **SEMESTER – VII**			
Course Code:		**Course Credits:**	**04**
Teaching Hours / Week (L:T:P):	**2:0:4**	**CA Marks:**	**40**
Total Number of Teaching Hours:	**--**	**END-SEM Marks:**	**60**

Course Pre-requisites: Web Technology

Course Description:

a. This course offers a comprehensive foundation in full stack web development using Node.js, Express.js, EJS, and MySQL.
b. Students will learn to build dynamic and responsive web applications by integrating front-end and back-end technologies.
c. The course emphasizes hands-on project-based learning and real-world application development.
d. Learners will design RESTful APIs and manage databases for seamless client-server interaction.
e. Through practical exercises, students will gain industry-ready skills for scalable web application development.

Course Learning Objectives: This course will enable the students to:

1. Introduce students to server-side development using Node.js and its ecosystem.
2. Equip learners with the ability to develop scalable and efficient web servers using Express.js.
3. Enable the use of EJS for dynamic web page rendering and data binding.
4. Provide hands-on experience in integrating relational databases like MySQL with Node.js.
5. Facilitate the development of full stack applications through project-based learning using REST APIs and client-server integration.

Course Outcome:

1. CO1: Build responsive front-end user interfaces using Bootstrap layout and component system.
2. CO2: Analyze the core principles, architecture, and modular structure of Node.js for robust server-side application development.
3. CO3: Construct and deploy scalable web servers using Node.js and Express.js to efficiently manage HTTP protocols and client-server communication.
4. CO4: Design and synthesize dynamic, data-driven user interfaces using EJS templating integrated with backend services.
5. CO5: Implement and manage CRUD operations by interfacing Node.js with MySQL and engineer RESTful APIs for structured data exchange.

UNIT – I	**Responsive Interfaces with Bootstrap Components**	**14 Hours**

Main Topic-1: Introduction to Media Queries, Responsive Web Design Using Media Queries, Syntax, Uses of Media Queries, Benefits of Responsive Design, Elements of a Responsive Design, Working of Responsive Design Main Topic-2: Introduction to Bootstrap: Bootstrap Concept, Uses of Bootstrap, Bootstrap library source, Bootstrap Basics- container, jumbotron, page header, buttons, tables; Main Topic-3: Bootstrap Grid System, Bootstrap Images, Bootstrap Forms, Bootstrap Flexbox- horizontal direction, vertical direction.		
Pedagogy	ICT Teaching / Power Point Presentation and Videos	
	Self-study / Do it yourself	
	Experiential Learning Topics:	
	Case Study / PBL - Project Based Learning	
UNIT – II	Backend Development Fundamentals with Node.js	10 Hours
Main Topic-1: Introduction to Node.js: What is Node.js? Why use it? Capabilities and file structure Main Topic-2: Setting up Node.js Development Environment – Installing Node.js and npm, NPM and project creation, Node.js modules, Creating a basic web server, Node.js console		
Pedagogy	ICT Teaching / Power Point Presentation and Videos	
	Self-study / Do it yourself	
	Experiential Learning Topics:	
	Case Study / PBL - Project Based Learning	
UNIT – III	Building Web Servers using Express.js	12 Hours
Main Topic-1: Introduction to Express.js – Features, benefits, use cases Main Topic-2: Express Server Setup – Installing Express.js, project setup, configuration, building Express - based web server		
Pedagogy	ICT Teaching / Power Point Presentation and Videos	
	Self-study / Do it yourself	
	Experiential Learning Topics:	
	Case Study / PBL - Project Based Learning	
UNIT – IV	Templating and Dynamic Views with EJS	12 Hours
Main Topic-1: Integrating EJS Templates, Creating EJS partials and views, Passing data to views, rendering variables, Looping through data in templates Main Topic-2: Building dynamic front-ends using EJS with Express		
Pedagogy	ICT Teaching / Power Point Presentation and Videos	
	Self-study / Do it yourself	
	Experiential Learning Topics:	
	Case Study / PBL - Project Based Learning	
UNIT – V	Database Integration and REST API Development	12 Hours

Main Topic-1: Connecting Node.js with MySQL, Installing MySQL, setting up the connection, Performing CRUD operations using Node.js and MySQL **Main Topic-2: Building RESTful APIs using Express.js, HTTP methods (GET, POST, PUT, DELETE), Handling requests and responses using Express, Using body-parser for form data**	
Pedagogy	**ICT Teaching / Power Point Presentation and Videos**
	Self-study / Do it yourself
	Experiential Learning Topics:
	Case Study / PBL - Project Based Learning

Text Books:

1. Andrew Mead, |Learning Node.js Development: Learn the fundamentals of Node.js, and deploy and test Node.js applications on the web|, Packt Publishing, ISBN-10 : 1788395549, ISBN-13 : 978-1788395540
2. Ethan Brown, |Web Development with Node and Express, 2e: Leveraging the JavaScript Stack|, O'Reilly; 2nd edition, ISBN-10 : 1492053511, ISBN-13 : 978-1492053514

Reference Books:

1. Mario Casciaro and Luciano Mammino, |Node.js Design Patterns|, Packt Publishing, ISBN: 978-1839214110
2. Russell Dyer, |Learning MySQL and MariaDB|, O'Reilly Media, ISBN: 978-1449362904
3. Leonard Richardson and Mike Amundsen,|RESTful Web APIs|, O'Reilly Media, ISBN: 978-1449358068
4. Ben Frain, Responsive Web Design with HTML5 and CSS (3e), Packt Publishing, ISBN: 978-1839211560

URLs (Optional) - List of Online Courses

1. **Full-Stack Web Development with React Specialization – Coursera (by The Hong Kong University of Science and Technology)**
   Covers: Node.js, Express.js, MongoDB, React, REST APIs
   Platform: Coursera
   Level: Intermediate – Project-based

2. **The Complete Node.js Developer Course (3rd Edition) – Udemy (by Andrew Mead & Rob Percival)**
   Covers: Node.js, Express, MongoDB, REST APIs, Authentication
   Platform: Udemy
   Level: Beginner to Advanced

3. **Web Development with Node.js and Express – Coursera (by University of London)**
   Covers: Node.js, Express.js, REST APIs
   Platform: Coursera
   Level: Intermediate

4. **MySQL for Data Analytics and Business Intelligence – Udemy**
   Covers: MySQL Basics to Advanced (joins, CRUD, queries)
   Platform: Udemy
   Level: Beginner to Intermediate

5. **Full Stack Open 2024 – University of Helsinki (Free)**

List of Assignments:

Sr. No.	Assignment Title
1.	**Create a Static Homepage Layout Using Bootstrap** 1.1 Design a responsive homepage using Bootstrap components like container, jumbotron, headers, and buttons. 1.2 Use the grid system to create a structured and adaptive layout suitable for any web app.
2.	**Design a Grid-Based Information Page** 2.1 Develop a multi-column content layout using Bootstrap Grid. 2.2 Display items such as cards, product previews, or blog entries with consistent alignment and spacing.
3.	**Set Up a Node.js Project with Express Server** 3.1 Initialize a new Node.js project using npm init. 3.2 Install Express.js and set up a basic web server with routes like Home (/), About (/about), and Contact (/contact).
4.	**Implement Modular Routing and Middleware Logging** 4.1 Organize server logic using separate route modules (e.g., userRoutes, productRoutes). 4.2 Add middleware to log request URLs, methods, and timestamps for all incoming traffic.
5.	**Create a Data Submission Form and Handle POST Requests** 5.1 Build an HTML form (e.g., for adding items or submitting feedback) and handle its POST request using Express. 5.2 Display the submitted data dynamically on a confirmation view.
6.	**Serve Dynamic Content Using Server-Side Rendering** 6.1 Render a list of items (e.g., products, posts, services) dynamically using EJS with hardcoded or temporary in-memory data. 6.2 Route should accept data from the server and render it in the view using loops.
7.	**Create Dynamic Detail Pages Using URL Parameters** 7.1 Use route parameters to render individual item details dynamically via EJS. 7.2 Display properties like title, description, and date using passed route data.

8. **Build a Reusable Layout Using EJS Partials**

   8.1 Create common EJS partials (header, footer, navbar) and include them across views.

   8.2 Maintain consistency across all pages by using a shared layout template.

9. **Connect Node.js to MySQL and Perform Basic CRUD**

   9.1 Create MySQL tables and connect to them using a MySQL Node.js package.

   9.2 Implement basic CRUD operations from Express routes to the database.

10. **Develop RESTful APIs for Data Access and Manipulation**

    10.1 Build a set of RESTful APIs using Express.js for performing operations.

    10.2 Test the endpoints using Postman and ensure they return structured JSON responses.

Assignments

Assignment 1:

Aim:

To design a responsive static homepage for a web application using Bootstrap, implementing Bootstrap's container, grid system, jumbotron/hero section, headers, and buttons to achieve a professional, adaptive layout

---

Theory:

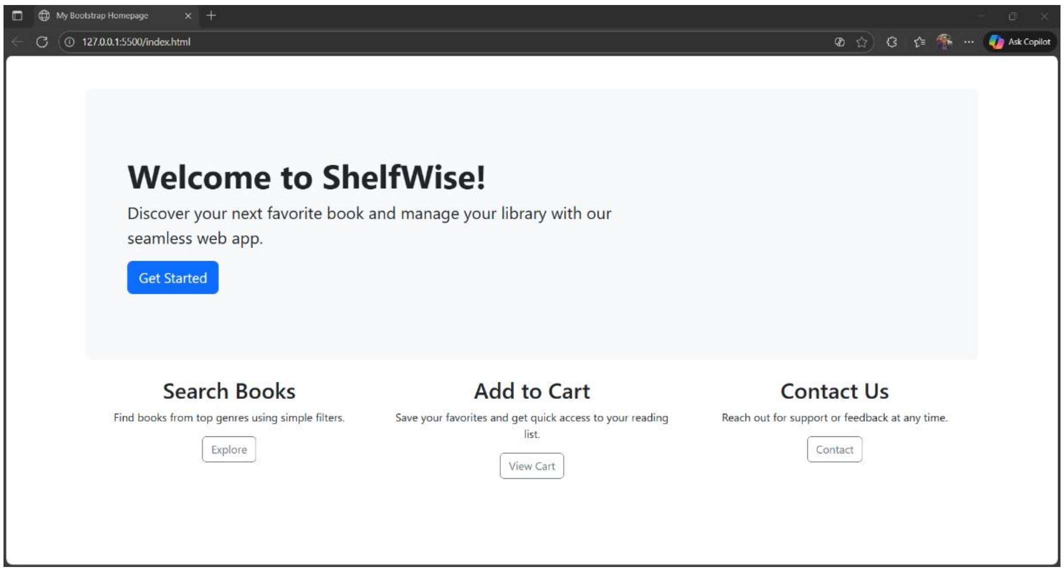
Bootstrap is a popular front-end framework for developing responsive and mobile-first web apps. It provides pre-defined classes and utilities for grids, containers, typography, buttons, banners (jumbotron/hero), and much more. The grid system (container, row, col-) enables layouts that adapt gracefully to any screen size. Using Bootstrap not only speeds up UI development but ensures the website remains visually consistent and responsive.

Code (index.ejs/static index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>My Bootstrap Homepage</title>
  <!-- Bootstrap CDN -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">

</head>
<body>
  <div class="container my-5">
    <!-- Hero Section -->
    <div class="p-5 mb-4 bg-light rounded-3">
      <div class="container-fluid py-5">
        <h1 class="display-5 fw-bold">Welcome to ShelfWise!</h1>
        <p class="col-md-8 fs-4">Discover your next favorite book and manage your library with our
seamless web app.</p>
        <button class="btn btn-primary btn-lg" type="button">Get Started</button>
      </div>
    </div>

    <!-- Grid Feature Section -->
    <div class="row text-center mb-4">
      <div class="col-md-4">
        <h2>Search Books</h2>
        <p>Find books from top genres using simple filters.</p>
```

```
<button class="btn btn-outline-secondary">Explore</button>
      </div>
      <div class="col-md-4">
        <h2>Add to Cart</h2>
        <p>Save your favorites and get quick access to your reading list.</p>
        <button class="btn btn-outline-secondary">View Cart</button>
      </div>
      <div class="col-md-4">
        <h2>Contact Us</h2>
        <p>Reach out for support or feedback at any time.</p>
        <button class="btn btn-outline-secondary">Contact</button>
      </div>
    </div>
  </div>
  <!-- Bootstrap JS Bundle -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Output:



Welcome to ShelfWise!

Discover your next favorite book and manage your library with our seamless web app.

Get Started

Search Books

Find books from top genres using simple filters.

Explore

Add to Cart

Save your favorites and get quick access to your reading list.

View Cart

Contact Us

Reach out for support or feedback at any time.

Contact

This code includes:

- Responsive container and grid layout.
- A "hero"/jumbotron-style welcome section.
- Three structured grid columns with headings and buttons.w3schools+1
- All layout elements automatically adapt to different device sizes.geeksforgeeks

---

Conclusion:

By using Bootstrap's grid, container, hero section, and button components, a fully responsive static homepage was created.

The layout adapts to all device sizes, looks visually appealing, and is suitable for any modern web application. Bootstrap's utility classes and structural components simplify and standardize responsive design.

Assignment 2

Aim:

To create an information page featuring a consistent, grid-based, multi-column layout by using Bootstrap's grid and card components to display items such as product previews, cards, or blog entries, each aligned and spaced responsively.

Theory:

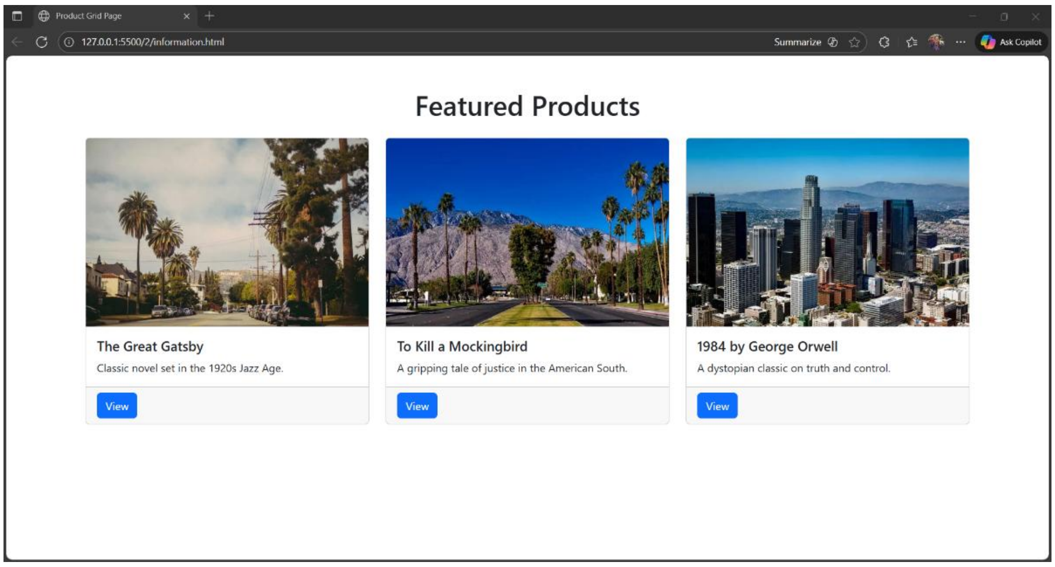
Bootstrap's grid system is based on containers, rows, and columns powered by CSS Flexbox.
It enables you to create complex and responsive layouts easily, ensuring that content is well-aligned and adapts to all device sizes.
Card components offer flexible content containers with options for images, text, headers, footers, and more, making them ideal for displaying repeated information blocks like product previews or blogs.w3schools+2

Code (information.html / EJS section):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Product Grid Page</title>
  <!-- Bootstrap CDN -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
  <div class="container py-5">
    <h1 class="mb-4 text-center">Featured Products</h1>
    <div class="row row-cols-1 row-cols-md-3 g-4">

      <!-- Card 1 -->
      <div class="col">
        <div class="card h-100">
          <img src="https://mdbcdn.b-cdn.net/img/new/standard/city/041.webp" class="card-img-top"
alt="Hollywood Sign" />
          <div class="card-body">
            <h5 class="card-title">The Great Gatsby</h5>
            <p class="card-text">Classic novel set in the 1920s Jazz Age.</p>
          </div>
          <div class="card-footer">
            <a href="#" class="btn btn-primary">View</a>
          </div>
```

```
 </div>
      </div>

      <!-- Card 2 -->
      <div class="col">
        <div class="card h-100">
<img src="https://mdbcdn.b-cdn.net/img/new/standard/city/042.webp" class="card-img-top"
alt="Palm Springs Road" />
          <div class="card-body">
            <h5 class="card-title">To Kill a Mockingbird</h5>
            <p class="card-text">A gripping tale of justice in the American South.</p>
          </div>
          <div class="card-footer">
            <a href="#" class="btn btn-primary">View</a>
          </div>
        </div>
      </div>

      <!-- Card 3 -->
      <div class="col">
        <div class="card h-100">
          <img src="https://mdbcdn.b-cdn.net/img/new/standard/city/043.webp" class="card-img-top"
alt="Skyscrapers" />
          <div class="card-body">
            <h5 class="card-title">1984 by George Orwell</h5>
            <p class="card-text">A dystopian classic on truth and control.</p>
          </div>
          <div class="card-footer">
            <a href="#" class="btn btn-primary">View</a>
          </div>
        </div>
      </div>

    </div>
  </div>
  <!-- Bootstrap JS Bundle -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Output:



Featured Products

The Great Gatsby
Classic novel set in the 1920s Jazz Age.
View

To Kill a Mockingbird
A gripping tale of justice in the American South.
View

1984 by George Orwell
A dystopian classic on truth and control.
View

- Uses Bootstrap’s grid: .row, .col, and .row-cols-* for auto-columns.
- Each product/blog entry is shown in a card with image, title, description, and action button.
- Responsive on all devices, with spacing managed by g-4.
- Can be expanded to include more cards by copy-pasting the <div class="col">...</div> block

---

Conclusion:

By leveraging Bootstrap’s grid and card components, a visually attractive and responsive multi-column information layout was achieved.
Each item is consistently spaced, aligned, and adapts perfectly to different screen sizes, making this pattern ideal for product listings, previews, or blogs in any web application.

Assignment 3

Aim
To initialize a basic Node.js project using npm and set up a web server with Express that responds to Home (/), About (/about), and Contact (/contact) routes.

Theory:

Node.js is an open-source runtime for server-side JavaScript.
Express is a minimalist web framework for Node that simplifies creating web servers and handling routes, middleware, and responses.
The typical workflow starts by initializing the project with npm, installing Express via npm, and writing code that sets up routes and HTTP responses.image.jpg

---

Steps and Sample Code:

Step 1: Initialize Node.js Project
bash
npm init -y
This command creates a package.json file with default settings.image.jpg

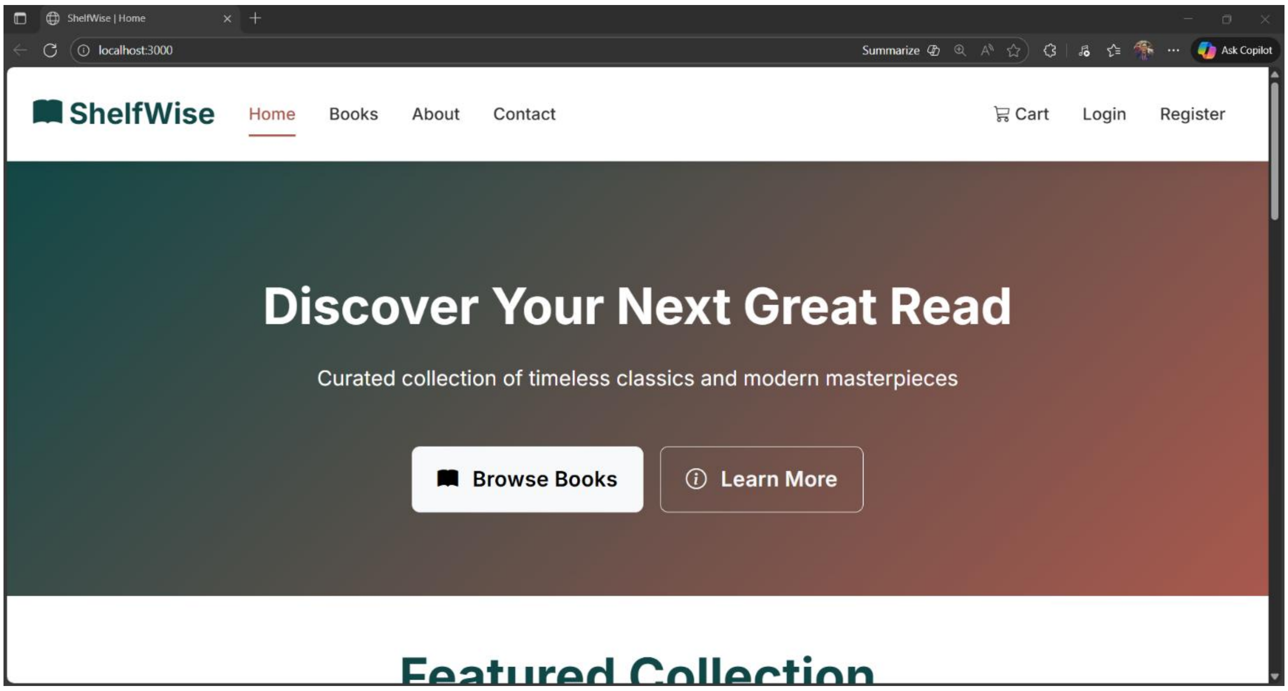
Step 2: Install Express
bash
npm install express

Step 3: Create app.js and set up server/routes

```
app.get('/', (req, res) => {
  try {
    const userPreferences = Array.isArray(req.session.user?.preferred_genres)
      ? req.session.user.preferred_genres
      : [];

    const sortedBooks = [...books];

    if (userPreferences.length > 0) {
      sortedBooks.sort((a, b) => {
        const prefIndexA = userPreferences.indexOf(a.genre);
        const prefIndexB = userPreferences.indexOf(b.genre);
        const scoreA = prefIndexA === -1 ? userPreferences.length : prefIndexA;
        const scoreB = prefIndexB === -1 ? userPreferences.length : prefIndexB;
        if (scoreA !== scoreB) {
          return scoreA - scoreB;
        }
        return b.rating - a.rating;
      });
    } else {


   sortedBooks.sort((a, b) => b.rating - a.rating);
    }

    const featuredBooks = sortedBooks.slice(0, 6);

    res.render('index', { title: 'ShelfWise | Home', books: featuredBooks, currentPage: 'home' });
  } catch (err) {
    console.error(err);
    setError(req, 'Failed to load home page.');
    res.render('index', { title: 'ShelfWise | Home', books: [], currentPage: 'home' });
  }
});

// About page
app.get('/about', (req, res) => {
  res.render('about', { title: 'About Us | ShelfWise', currentPage: 'about' });
});

// Contact page
app.get('/contact', (req, res) => {
  res.render('contact', { title: 'Contact Us | ShelfWise', currentPage: 'contact' });
});

// Contact form submission (CRUD - Create)
app.post('/contact', (req, res) => {
  const { name, email, message } = req.body;

  if (!name || !email || !message) {
    setError(req, 'All fields are required.');
    return res.redirect('/contact');
  }

  try {
    // Add contact to in-memory storage
    contacts.push({
      id: contactIdCounter++,
      name,
      email,
      message,
      created_at: new Date()
    });
    setSuccess(req, 'Your inquiry has been submitted successfully!');
  } catch (err) {
    console.error(err);
    setError(req, 'Failed to submit your inquiry. Please try again.');
  }

  res.redirect('/contact');
```

```
});
```

Output:



Conclusion:

A Node.js project was successfully initialized and set up with Express.
Essential routes—including Home, About, and Contact—were implemented, and the web server runs and responds correctly to requests.
This forms the backbone for further development and integration of dynamic web pages or APIs.

Assignment 4

Aim:

To modularize Express.js server code using route modules and enhance server observability by adding middleware for logging every incoming request's URL, method, and timestamp.

Theory:

Separating routes into modules (files like userRoutes.js, productRoutes.js) keeps your project organized and easier to maintain.
Middleware in Express is a function that runs for every request in the pipeline; it can handle logging, authentication, or any custom logic.
Logging middleware tracks activity and simplifies debugging.

Code:

routes/userRoutes.js:

```
const express = require('express');
const router = express.Router();

router.get('/users', (req, res) => {
  res.send('User list');
});

router.get('/users/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});

module.exports = router;
```

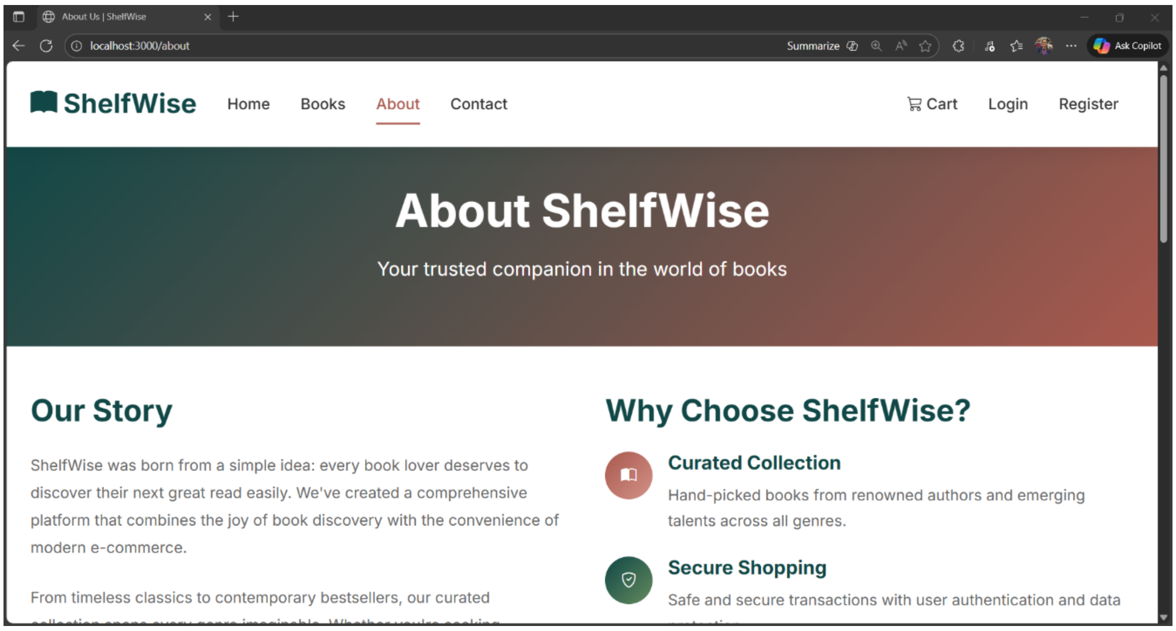
routes/productRoutes.js

```
const express = require('express');
const router = express.Router();

router.get('/products', (req, res) => {
  res.send('Product list');
});

router.get('/products/:id', (req, res) => {
  res.send(`Product ID: ${req.params.id}`);
});

module.exports = router;
```

app.js

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

// Logging Middleware
app.use((req, res, next) => {
  const now = new Date().toISOString();
  console.log(`[${now}] ${req.method} ${req.url}`);
  next();
});

// Import Routes
const userRoutes = require('./routes/userRoutes');
const productRoutes = require('./routes/productRoutes');

// Use Route Modules
app.use(userRoutes);
app.use(productRoutes);

// Basic pages
app.get('/', (req, res) => res.send('Home Page'));
app.get('/about', (req, res) => res.send('About Page'));
app.get('/contact', (req, res) => res.send('Contact Page'));

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Output:



About ShelfWise

Your trusted companion in the world of books

Our Story

ShelfWise was born from a simple idea: every book lover deserves to discover their next great read easily. We've created a comprehensive platform that combines the joy of book discovery with the convenience of modern e-commerce.

From timeless classics to contemporary bestsellers, our curated

Why Choose ShelfWise?

Curated Collection

Hand-picked books from renowned authors and emerging talents across all genres.

Secure Shopping

Safe and secure transactions with user authentication and data

- Place app.js in your project folder.
- Place the two route files in a folder called routes.
- Visit /users, /products, /users/1, /products/2, etc.
- Every request logs method, URL, and timestamp.

Conclusion:

By splitting logic into route modules and adding request logging middleware, the Express server is now well-organized and provides crucial request traceability for development and debugging. This setup is scalable for larger applications.

Assignment 5

Aim:

To build an HTML feedback/contact form that submits data using POST, and display the submitted data dynamically on a confirmation or success page.

Theory:

Express allows you to handle form submissions by registering a POST route and using body parsing middleware to read form fields.
When a user submits data, the server processes it and can render a confirmation view with the submitted information, improving UX.
In ShelfWise, the /contact form works this way, demonstrating modern web patterns.

Code (from ShelfWise):

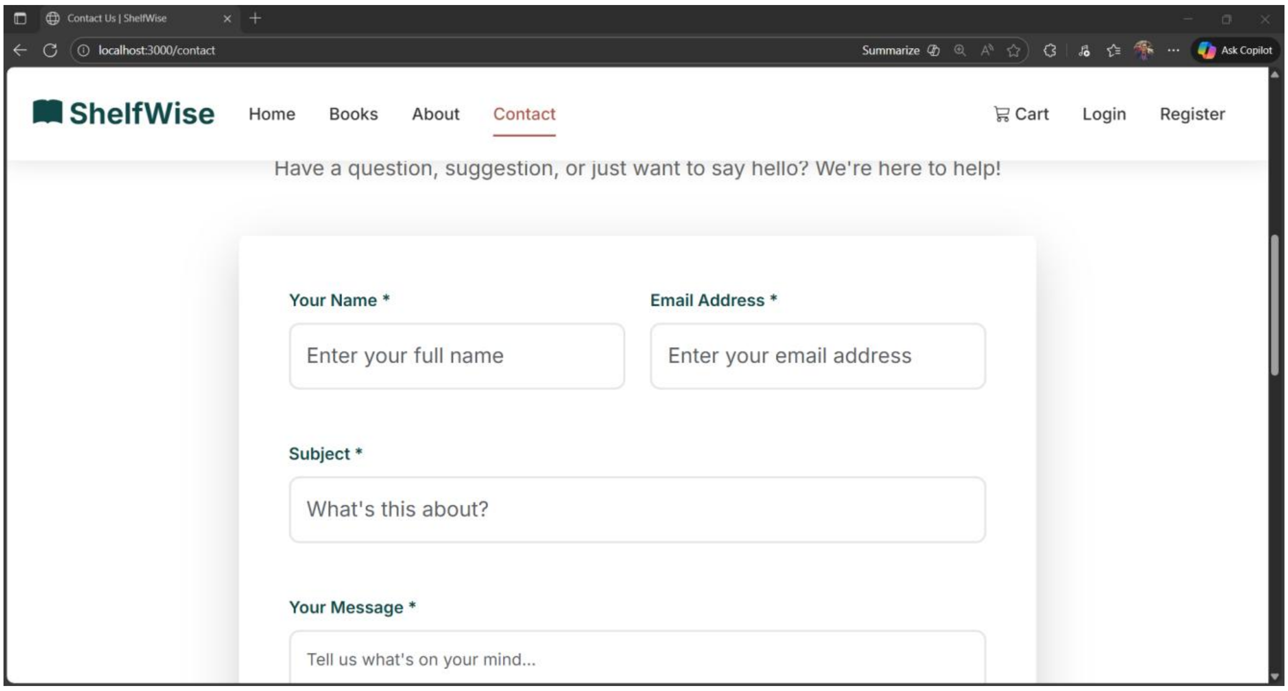
contact.ejs (HTML form — cut down, uses POST)

```
<form action="/contact" method="POST">
  <div>
    <label for="name">Name:</label>
    <input id="name" name="name" type="text" required />
  </div>
  <div>
    <label for="email">Email:</label>
    <input id="email" name="email" type="email" required />
  </div>
  <div>
    <label for="message">Message:</label>
    <textarea id="message" name="message" required></textarea>
  </div>
  <button type="submit">Send</button>
</form>
```

app.js (handle POST & show confirmation):

```
// Middleware already present: app.use(express.urlencoded({ extended: true }))

// Contact - GET
app.get('/contact', (req, res) => {
  res.render('contact', { title: 'Contact Us | ShelfWise', currentPage: 'contact' });
});

// Contact - POST (handle submission logic)
app.post('/contact', (req, res) => {
  const { name, email, message } = req.body;
```

```
 if (!name || !email || !message) {
  setError(req, 'All fields are required.');
  return res.redirect('/contact');
 }
 try {
  // Normally, store in DB
  contacts.push({ id: contactIdCounter++, name, email, message, created_at: new Date() });
  setSuccess(req, 'Your inquiry has been submitted successfully!');
  // Show confirmation view and data
  res.render('contact-confirm', { name, email, message });
 } catch (err) {
  console.error(err);
  setError(req, 'Failed to submit your inquiry. Please try again.');
  res.redirect('/contact');
 }
});
```



Have a question, suggestion, or just want to say hello? We're here to help!

Your Name *	Email Address *
Enter your full name	Enter your email address
Subject *	
What's this about?	
Your Message *	
Tell us what's on your mind...	

Conclusion:

The ShelfWise contact feature demonstrates a complete Express POST handling workflow: user data is captured via a form, processed on the server, and shown dynamically as a confirmation view, meeting all assignment criteria for robust, user-friendly submission handling.

Assignment 6

Aim:

To render a dynamic list of products on a web page using EJS, receiving product data from the Express server, and generate the view with a loop.

Theory:

Server-side rendering (SSR) enables Node.js (with Express and EJS) to send HTML containing dynamically generated content before it reaches the client.
The view engine (EJS) uses its templating tags to iterate and display object lists passed as variables from the server, updating the web page whenever server data changes.

Code (from ShelfWise project):

In-memory data in app.js:

```
let books = [
  { id: 1, title: 'The Great Gatsby', author: 'F. Scott Fitzgerald', price: 299.00 },
  { id: 2, title: 'To Kill a Mockingbird', author: 'Harper Lee', price: 349.00 },
  // ...more book objects
];
```

Route renders EJS view with books list:

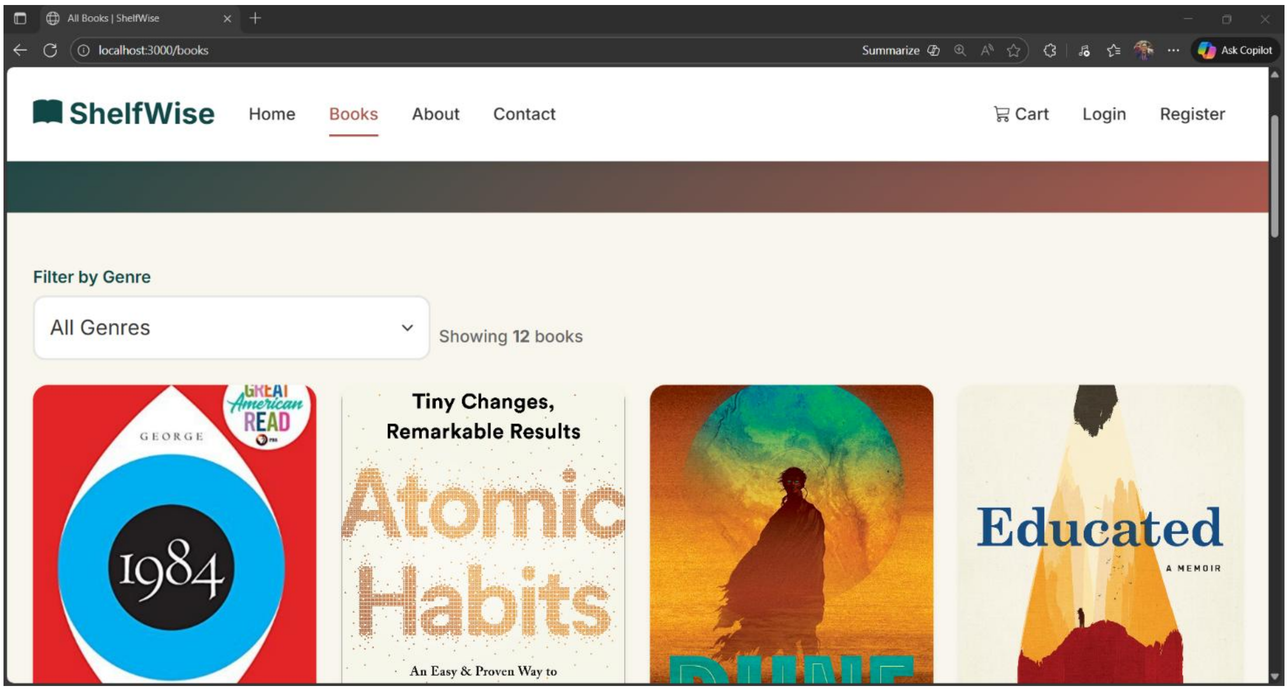
```
// Home page route - renders featured books
app.get('/', (req, res) => {
  const featuredBooks = books.slice(0, 6);
  res.render('index', { title: 'ShelfWise | Home', books: featuredBooks });
});
```

or for all books:

```
app.get('/books', (req, res) => {
  res.render('books', { title: 'All Books', books });
});
```

EJS view ("index.ejs" or "books.ejs"):

```
<h2>Featured Books</h2>
<ul>
  <% books.forEach(function(book) { %>
    <li>
      <strong><%= book.title %></strong> by <%= book.author %> — ₹<%= book.price %>
    </li>
  <% }); %>
</ul>
```



- The server passes the books array to the view.
- EJS loops over books and renders each item as HTML.

Conclusion:

The project meets all requirements for dynamic, server-side rendering:
it passes in-memory lists to EJS templates and uses loops to build HTML, providing fresh content from the server on every request.

Assignment 7

Aim:

To render item detail pages by passing the item's id as a URL parameter, and display properties like title, description, and date using server data inside EJS templates.

Theory:

Express route parameters (e.g. /books/:id) allow you to extract an identifier from the URL and use it to find and serve dynamic content for that item.
Passing that item to EJS enables rendering its details in a template with variables and formatting.

Code Example (From ShelfWise Project):

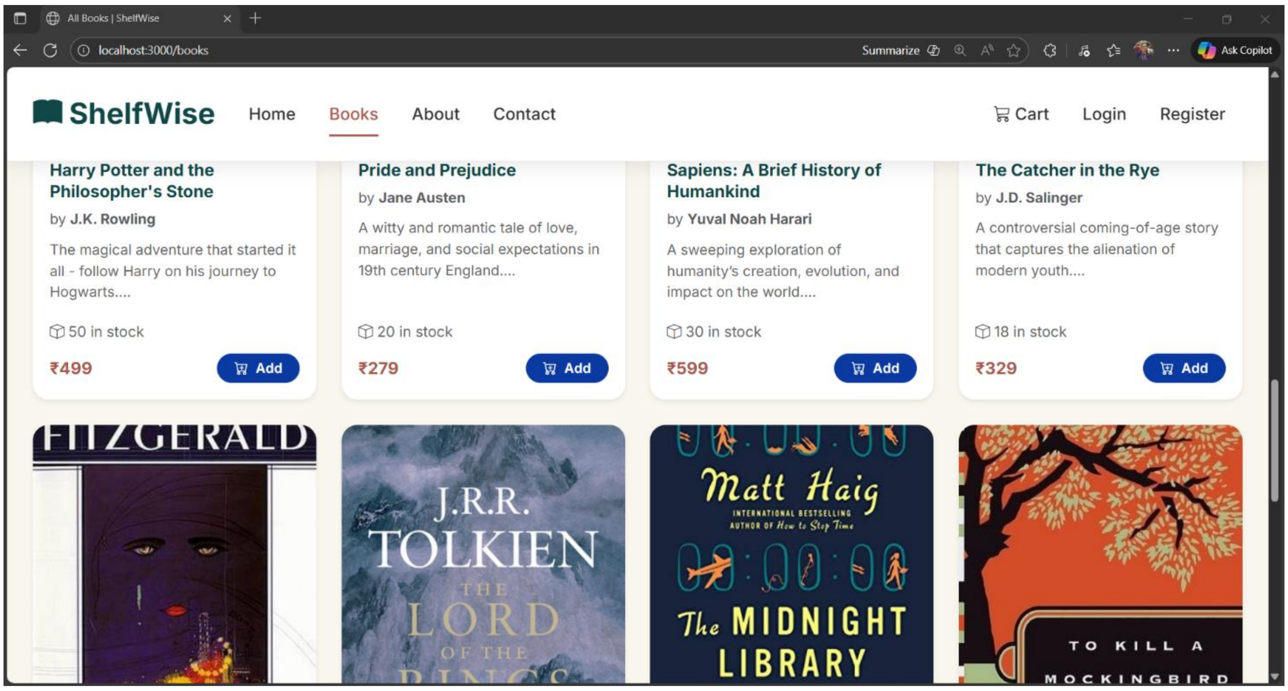
Add to app.js:

```
// Book Details Route
app.get('/books/:id', (req, res) => {
  const bookId = parseInt(req.params.id, 10);
  const book = books.find(b => b.id === bookId);
  if (!book) {
    return res.status(404).render('404', { title: 'Book Not Found', currentPage: '404' });
  }
  res.render('book-detail', { title: book.title, book });
});
```

Example book-detail.ejs (create in your views/ folder):

```
<h1><%= book.title %></h1>
<p><strong>Author:</strong> <%= book.author %></p>
<p><strong>Genre:</strong> <%= book.genre %></p>
<p><strong>Description:</strong> <%= book.description %></p>
<p><strong>Price:</strong> ₹<%= book.price %></p>
<p><strong>Available Stock:</strong> <%= book.stock_quantity %></p>
<p><em>Rendered at: <%= new Date().toLocaleString() %></em></p>
<a href="/books">Back to Books</a>
```

Output:


All Books | ShelfWise

localhost:3000/books

ShelfWise Home Books About Contact Cart Login Register

Harry Potter and the Philosopher's Stone
by J.K. Rowling
The magical adventure that started it all - follow Harry on his journey to Hogwarts....
50 in stock
₹499 Add

Pride and Prejudice
by Jane Austen
A witty and romantic tale of love, marriage, and social expectations in 19th century England....
20 in stock
₹279 Add

Sapiens: A Brief History of Humankind
by Yuval Noah Harari
A sweeping exploration of humanity's creation, evolution, and impact on the world....
30 in stock
₹599 Add

The Catcher in the Rye
by J.D. Salinger
A controversial coming-of-age story that captures the alienation of modern youth....
18 in stock
₹329 Add

Conclusion:

With route parameters and EJS, ShelfWise now dynamically serves item detail pages for any book, showing individual info with a dedicated URL.
This is scalable for products, posts, or services in any Express app.app.js+1

Assignment 8

Aim:

To create common layout partials (header, footer, navbar) for Express/EJS, and include them in every view for consistent site structure.

Theory:

EJS supports include statements, allowing code reuse and easier maintenance of headers, footers, and navigation bars. Partial templates help standardize the look and reduce repetition across pages. A shared layout ensures brand consistency and improves user experience.

Code (Folder Structure & Example):

/views/partials/header.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title><%= title %></title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
```

/views/partials/navbar.ejs

```
<nav class="navbar navbar-expand-lg navbar-light bg-light mb-4">
  <a class="navbar-brand" href="/">ShelfWise</a>
  <ul class="navbar-nav">
    <li class="nav-item"><a class="nav-link" href="/">Home</a></li>
    <li class="nav-item"><a class="nav-link" href="/books">Books</a></li>
    <li class="nav-item"><a class="nav-link" href="/about">About</a></li>
    <li class="nav-item"><a class="nav-link" href="/contact">Contact</a></li>
  </ul>
</nav>
```

/views/partials/footer.ejs:

```
<footer class="bg-dark text-light py-3 mt-5 text-center">
  <p>&copy; <%= new Date().getFullYear() %> ShelfWise. All rights reserved.</p>
</footer>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
```

```
</html>
```

Example of a page (e.g., index.ejs or books.ejs) using partials:

```
<%- include('partials/header', { title: 'Home | ShelfWise' }) %>
<%- include('partials/navbar') %>

<div class="container">
  <h1>Featured Books</h1>
  <ul>
    <% books.forEach(book => { %>
      <li><%= book.title %> by <%= book.author %></li>
    <% }) %>
  </ul>
</div>

<%- include('partials/footer') %>
```

Conclusion:

Using EJS include for common layout parts, your pages remain visually and structurally consistent. Maintaining the site and adding new pages becomes much easier—changes to headers, navbars, or footers are applied site-wide from a single file.

Assignment 9

Aim:

To connect Node.js to MySQL with a suitable package, create MySQL tables, and perform Create, Read, Update, and Delete (CRUD) operations from Express routes.

Theory:

The mysql2 package allows connecting Node.js applications to a MySQL database for executing SQL queries.
Basic CRUD in web apps means:

- **Create:** Add new records with INSERT
- **Read:** Fetch data with SELECT
- **Update:** Modify data with UPDATE
- **Delete:** Remove records with DELETE

These map to POST, GET, PUT, and DELETE routes in Express apps for manipulating database data with full persistence.

Code (from ShelfWise, simplified example):

1. MySQL Table Example

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  favorite_author VARCHAR(100),
  reading_frequency VARCHAR(50),
  preferred_genres TEXT,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

2. Database Connection (app.js)

```
const mysql = require('mysql2');
const db = mysql.createPool({
  host: 'localhost',
  user: 'gurneet',
  password: '1234567890',
  database: 'shelfwise',
  waitForConnections: true,
  connectionLimit: 10
}).promise();
```

3. Basic CRUD Routes

Create (Insert User):

```
app.post('/register', async (req, res) => {
  const { name, email, password } = req.body;
  const hash = await bcrypt.hash(password, 10);
  await db.execute(
    `INSERT INTO users (name, email, password_hash) VALUES (?, ?, ?)`,
    [name, email, hash]
  );
  res.send('User registered!');
});
```

Read (Get All Users):

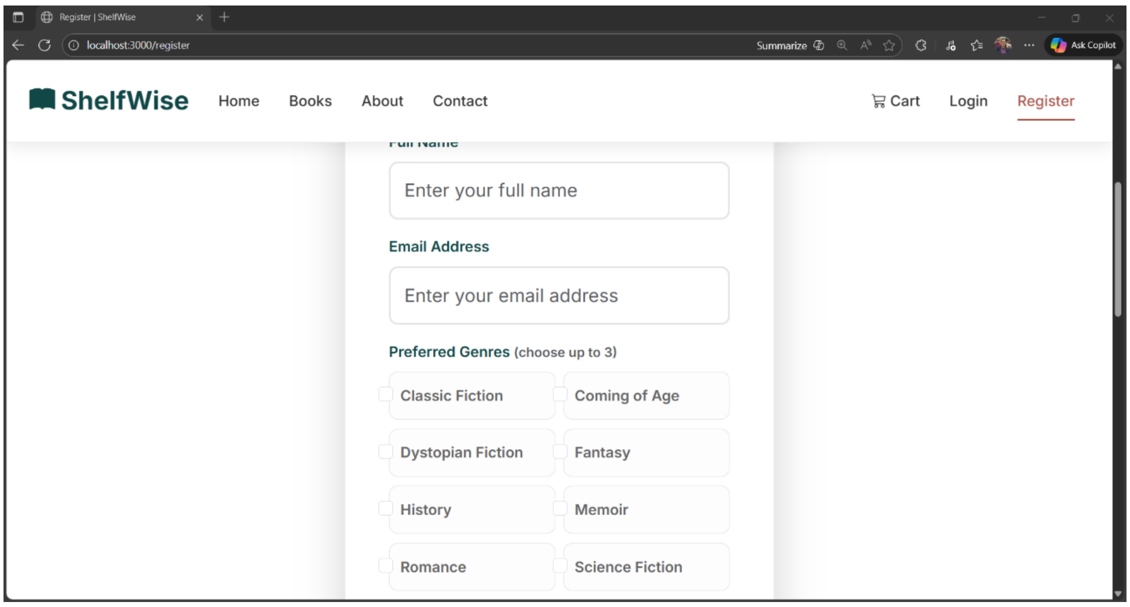
```
app.get('/users', async (req, res) => {
  const [users] = await db.execute('SELECT id, name, email FROM users');
  res.json(users);
});
```

Update (Update User's Name):

```
app.post('/users/:id/update', async (req, res) => {
  const { name } = req.body;
  await db.execute('UPDATE users SET name = ? WHERE id = ?', [name, req.params.id]);
  res.send('User updated!');
});
```

Delete (Delete User):

```
app.post('/users/:id/delete', async (req, res) => {
  await db.execute('DELETE FROM users WHERE id = ?', [req.params.id]);
  res.send('User deleted!');
});
```



Full Name
Enter your full name
Email Address
Enter your email address
Preferred Genres (choose up to 3)
- Classic Fiction
- Coming of Age
- Dystopian Fiction
- Fantasy
- History
- Memoir
- Romance
- Science Fiction

Conclusion:

Your code connects Node.js to MySQL, creates a user table, and implements all CRUD operations through Express routes and SQL, fulfilling standard persistent data management for web applications.

Assignment 10

Aim:

To implement RESTful JSON APIs for core operations on the app’s resources and verify correct output using Postman or a similar client.

Theory:

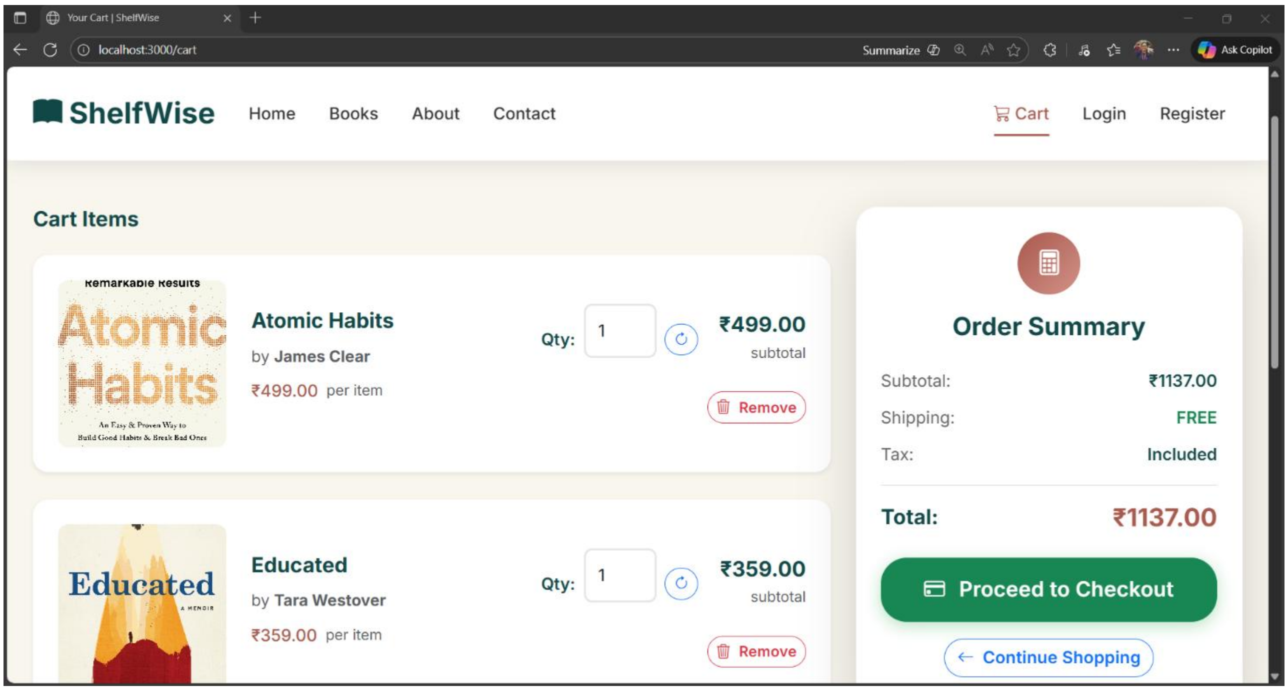
RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) mapped to CRUD operations and return structured JSON for interoperability. Using Express’s routing and JSON middleware, a resource can be listed, created, updated, or deleted by calling the corresponding endpoint.

API Implementation (Example for Books):

Add to app.js or an api/books.js file:

```
const express = require('express');
const app = express();
app.use(express.json()); // for parsing application/json

// In-memory data
let books = [
  { id: 1, title: "The Great Gatsby", author: "F. Scott Fitzgerald" },
  { id: 2, title: "Dune", author: "Frank Herbert" }
];

// GET all books
app.get('/api/books', (req, res) => {
  res.json(books);
});

// GET book by ID
app.get('/api/books/:id', (req, res) => {
  const book = books.find(b => b.id === parseInt(req.params.id));
  if (!book) return res.status(404).json({ error: "Book not found" });
  res.json(book);
});

// CREATE book
app.post('/api/books', (req, res) => {
  const { title, author } = req.body;
  const newBook = { id: books.length + 1, title, author };
  books.push(newBook);
  res.status(201).json(newBook);
});

// UPDATE book
```

```
app.put('/api/books/:id', (req, res) => {
  const book = books.find(b => b.id === parseInt(req.params.id));
  if (!book) return res.status(404).json({ error: "Book not found" });
  book.title = req.body.title;
  book.author = req.body.author;
  res.json(book);
});

// DELETE book
app.delete('/api/books/:id', (req, res) => {
  const index = books.findIndex(b => b.id === parseInt(req.params.id));
  if (index === -1) return res.status(404).json({ error: "Book not found" });
  const deleted = books.splice(index, 1);
  res.json(deleted[0]);
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`API server running on port ${PORT}`));
```



Cart Items

Atomic Habits
by James Clear
₹499.00 per item
Qty: 1
₹499.00
subtotal

Educated
by Tara Westover
₹359.00 per item
Qty: 1
₹359.00
subtotal

Order Summary

Subtotal:	₹1137.00
Shipping:	FREE
Tax:	Included
Total:	**₹1137.00**

Conclusion
ShelfWise's server now supports RESTful APIs for books, exposes clear JSON data, and is testable with tools like Postman to validate all CRUD operations.
You can adapt this pattern for other resources or connect endpoints to your MySQL database for real persistence