

WEEK-1:

HTML Frames:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Online Book Store</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header class="header">
    <h1>Online BookStore</h1>
  </header>

  <nav class="sidebar" aria-label="Book Categories">
    <ul>
      <li><b>Categories</b></li>
      <li id="aids">
        <a href="#aids">AI&DS</a>
        <ol>
          <li><a href="#aids1">1st Year</a></li>
          <li><a href="#aids2">2nd Year</a></li>
          <li><a href="#aids3">3rd Year</a></li>
          <li><a href="#aids4">4th Year</a></li>
        </ol>
      </li>
      <li id="ds">
        <a href="#ds">Data Science</a>
        <ol>
          <li><a href="#ds1">1st Year</a></li>
          <li><a href="#ds2">2nd Year</a></li>
          <li><a href="#ds3">3rd Year</a></li>
          <li><a href="#ds4">4th Year</a></li>
        </ol>
      </li>
      <li id="cybersecurity">
        <a href="#cybersecurity">Cybersecurity</a>
      </li>
    </ul>
  </nav>
</body>
</html>
```

```
<ol>
<li><a href="#cybersecurity1">1st Year</a></li>
    <li><a href="#cybersecurity2">2nd Year</a></li>
    <li><a href="#cybersecurity3">3rd Year</a></li>
    <li><a href="#cybersecurity4">4th Year</a></li>

</ol>

</li>
</ul>
</nav>

<main class="content">
    <h2>Welcome to the Online Bookstore</h2>

    <!-- AI&DS - 1st Year -->
    <section id="aids1" class="book-display-area">
        <h3>AI&DS - 1st Year</h3>
        <article class="book-card">
            <h4>AI for Beginners</h4>
            <p>by John Doe</p>
            <div class="card-body">
                <p>A comprehensive introduction to Artificial Intelligence for beginners.</p>
            </div>
        </article>
        <article class="book-card">
            <h4>Machine Learning 101</h4>
            <p>by Jane Smith</p>
            <div class="card-body">
                <p>A detailed guide to understanding machine learning algorithms and
concepts.</p>
            </div>
        </article>
    </section>

    <!-- AI&DS - 2nd Year -->
    <section id="aids2" class="book-display-area">
        <h3>AI&DS - 2nd Year</h3>
        <article class="book-card">
            <h4>Advanced AI Techniques</h4>
            <p>by Michael Jones</p>
            <div class="card-body">
```

```
<p>An advanced course in AI techniques for students with prior knowledge.</p>
    </div>
</article>
</section>
<!-- AI&DS - 3rd Year -->
<section id="aids3" class="book-display-area">
    <h3>AI&DS - 3rd Year</h3>
<article class="book-card">
    <h4>Advanced AI Techniques</h4>
    <p>by Michael Jones</p>
    <div class="card-body">
        <p>An advanced course in AI techniques for students with prior knowledge.</p>
    </div>
</article>
</section>
<!-- AI&DS - 4th Year -->
<section id="aids4" class="book-display-area">
    <h3>AI&DS - 4th Year</h3>
    <article class="book-card">
        <h4>Advanced AI Techniques</h4>
        <p>by Michael Jones</p>
        <div class="card-body">
            <p>An advanced course in AI techniques for students with prior knowledge.</p>
        </div>
    </article>
</section>

<!-- Data Science - 1st Year -->
<section id="ds1" class="book-display-area">
    <h3>Data Science - 1st Year</h3>
    <article class="book-card">
        <h4>Data Science with Python</h4>
        <p>by Peter Gray</p>
        <div class="card-body">
            <p>Master the basics of Data Science with Python programming language.</p>
        </div>
    </article>
</section>

<!-- Data Science - 2nd Year -->
<section id="ds2" class="book-display-area">
```

```
<h3>Data Science - 2nd Year</h3>
  <article class="book-card">
    <h4>Data Science with Python</h4>
    <p>by Peter Gray</p>
    <div class="card-body">
      <p>Master the basics of Data Science with Python programming language.</p>
    </div>
  </article>
</section>
<!-- Data Science - 3rd Year -->
<section id="ds3" class="book-display-area">
  <h3>Data Science - 3rd Year</h3>
  <article class="book-card">
    <h4>Data Science with Python</h4>
    <p>by Peter Gray</p>
    <div class="card-body">
      <p>Master the basics of Data Science with Python programming language.</p>
    </div>
  </article>
</section>
<!-- Data Science - 4th Year -->
<section id="ds4" class="book-display-area">
  <h3>Data Science - 4th Year</h3>
  <article class="book-card">
    <h4>Data Science with Python</h4>
    <p>by Peter Gray</p>
    <div class="card-body">
      <p>Master the basics of Data Science with Python programming language.</p>
    </div>
  </article>
</section>
<!-- Cybersecurity - 1st Year -->
<section id="cybersecurity1" class="book-display-area">
  <h3>Cybersecurity - 1st Year</h3>
  <article class="book-card">
    <h4>Introduction to Cybersecurity</h4>
    <p>by Anna Lee</p>
    <div class="card-body">
      <p>An introductory guide to the fundamentals of cybersecurity.</p>
```

```
</div>
    </article>
</section>
<!-- Cybersecurity - 2nd Year -->
<section id="cybersecurity2" class="book-display-area">
    <h3>Cybersecurity - 2nd Year</h3>
    <article class="book-card">
<h4>Network Security Basics</h4>
        <p>by Robert Brown</p>
        <div class="card-body">
            <p>A deep dive into network security protocols and techniques.</p>
        </div>
    </article>
</section>
<!-- Cybersecurity - 3rd Year -->
<section id="cybersecurity3" class="book-display-area">
    <h3>Cybersecurity - 3rd Year</h3>
    <article class="book-card">
        <h4>Advanced Cybersecurity</h4>
        <p>by Sarah White</p>
        <div class="card-body">
            <p>Explores advanced topics in ethical hacking and system defense.</p>
        </div>
    </article>
</section>
<!-- Cybersecurity - 4th Year -->
<section id="cybersecurity4" class="book-display-area">
    <h3>Cybersecurity - 4th Year</h3>
    <article class="book-card">
        <h4>Ethical Hacking and Penetration Testing</h4>
        <p>by James Black</p>
        <div class="card-body">
            <p>An advanced guide to ethical hacking, penetration testing, and security systems.</p>
        </div>
    </article>
</section>
</main>
</body>
</html>
```

OUTPUT

Online BookStore

- Categories
- AI&DS
 - 1. 1st Year
 - 2. 2nd Year
 - 3. 3rd Year
 - 4. 4th Year
- Data Science
 - 1. 1st Year
 - 2. 2nd Year
 - 3. 3rd Year
 - 4. 4th Year
- Cybersecurity
 - 1. 1st Year
 - 2. 2nd Year
 - 3. 3rd Year
 - 4. 4th Year

Welcome to the Online Bookstore

AI&DS - 1st Year

AI for Beginners

by John Doe

A comprehensive introduction to Artificial Intelligence for beginners.

Machine Learning 101

by Jane Smith

A detailed guide to understanding machine learning algorithms and concepts.

AI&DS - 2nd Year

Advanced AI Techniques

by Michael Jones

An advanced course in AI techniques for students with prior knowledge.

AI&DS - 3rd Year

Advanced AI Techniques

by Michael Jones

An advanced course in AI techniques for students with prior knowledge.

AI&DS - 4th Year

Advanced AI Techniques

by Michael Jones

An advanced course in AI techniques for students with prior knowledge.

Data Science - 1st Year

Data Science with Python

by Peter Gray

Master the basics of Data Science with Python programming language.

Data Science - 2nd Year

Data Science with Python

by Peter Gray

Master the basics of Data Science with Python programming language.

Data Science - 3rd Year

Data Science with Python

by Peter Gray

Master the basics of Data Science with Python programming language.

Data Science - 4th Year

Data Science with Python

Data Science - 4th Year

Data Science with Python

by Peter Gray

Master the basics of Data Science with Python programming language.

Cybersecurity - 1st Year

Introduction to Cybersecurity

by Anna Lee

An introductory guide to the fundamentals of cybersecurity.

Cybersecurity - 2nd Year

Network Security Basics

by Robert Brown

A deep dive into network security protocols and techniques.

Cybersecurity - 3rd Year

Advanced Cybersecurity

by Sarah White

Explores advanced topics in ethical hacking and system defense.

Cybersecurity - 4th Year

Ethical Hacking and Penetration Testing

by James Black

An advanced guide to ethical hacking, penetration testing, and security systems.

WEEK-2: CSS Styling

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 60px 0 0;
}

.header {
    background-color: aqua;
    padding: 10px 0;
    color: black;
    text-align: center;
    width: 100%;
    position: fixed;
    top: 0;
    left: 0;
    height: 50px;
    z-index: 10;
}

.sidebar {
    background-color: #f05555;
    padding: 20px;
    width: 150px;
    position: fixed;
    top: 50px;
    left: 0;
    height: calc(100vh - 50px);
    overflow-y: auto;
}

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

.sidebar li {
    margin: 10px 0;
}

.sidebar a {
    text-decoration: none;
    color: black;
    font-size: 18px;
    display: block;
    padding: 5px 0;
}

/* Hide all submenus by default */
.sidebar ol {
    display: none;
```

```
list-style-type: none;
padding-left: 20px;
margin-top: 5px;
}

/* Show submenu when parent is targeted */
.sidebar li:target > ol {
  display: block;
}

.content {
  margin-left: 200px;
  padding: 20px;
  background-color: #fff;
  min-height: calc(100vh - 50px);
  overflow-y: auto;
}

/* Hide all book sections by default */
.book-display-area {
  display: none;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

/* Show book section when targeted */
.book-display-area:target {
  display: grid;
}

.book-card {
  padding: 20px;
  background-color: #f4f4f4;
  border: 1px solid #ccc;
  border-radius: 8px;
  text-align: center;
}

.book-card h4 {
  margin: 0 0 10px 0;
}

.book-card p {
  margin: 0;
}

.card-body {
  margin-top: 10px;
  font-size: 14px;
  color: #555;
}
```



```
/* Style for active links */
```

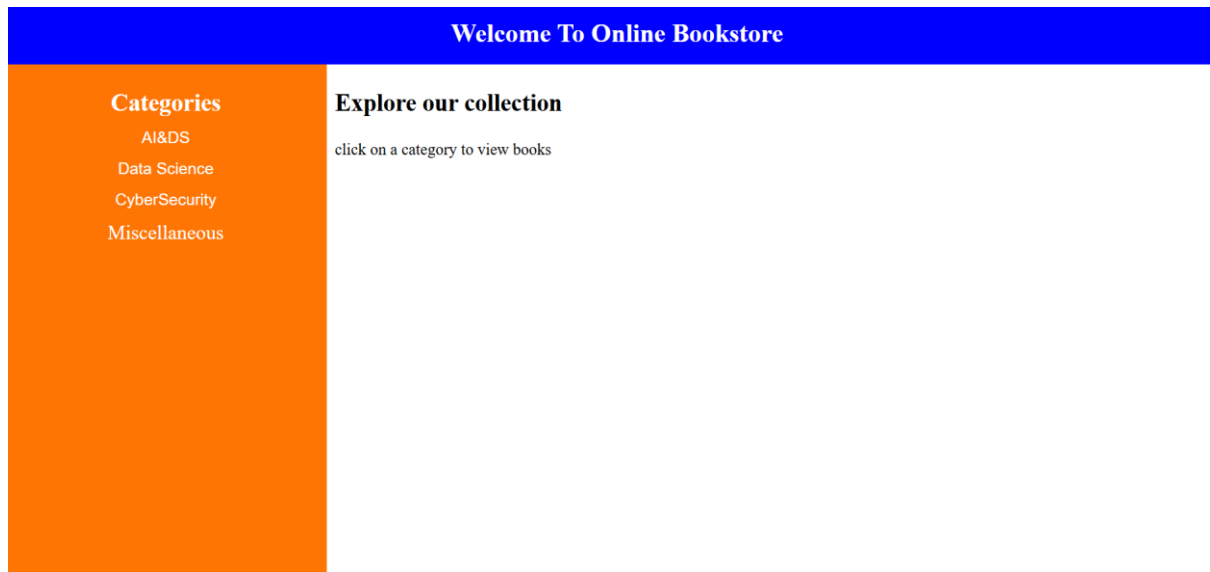
```
.sidebar a:target {  
    background-color: #ff7777;  
    padding: 5px;  
    border-radius: 4px;  
}
```

```
/* Add hover effects for better interactivity */
```

```
.sidebar a:hover {  
    background-color: #ff7777;  
    border-radius: 4px;  
}
```

```
.book-card:hover {  
    transform: translateY(-2px);  
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);  
    transition: all 0.3s ease;  
}
```

Output:



WEEK-3: JavaScript:**a) User Registration Form****Code:****HTML:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login Page</title>
  <link rel="stylesheet" href="./style.css">
</head>
<body>
  <div class="container" id="loginBox">
    <h2>Login</h2>
    <input type="text" id="loginUser" placeholder="Username">
    <input type="password" id="loginPass" placeholder="Password">
    <button onclick="login()">Login</button>
    <p class="message" id="loginMessage"></p>
    <p>Don't have an account? <a href="#" onclick="showRegister()">Create
one</a></p>
  </div>

  <div class="container" id="registerBox" style="display: none;">
    <h2>Create Account</h2>
    <input type="text" id="registerUser" placeholder="New Username">
    <input type="password" id="registerPass" placeholder="New Password">
    <button onclick="register()">Register</button>
    <p class="message" id="registerMessage"></p>
    <p>Already have an account? <a href="#"
onclick="showLogin()">Login</a></p>
  </div>

  <script src="./script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-color: #f4f4f4;
```

```
}

.container {
  background: white;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
  text-align: center;
  width: 300px;
}

input {
  display: block;
  width: 85%;
  margin: 10px 0;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin-left: 20px;
}

button {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  border: none;
  cursor: pointer;
  width: 100%;
  border-radius: 5px;
}

button:hover {
  background-color: #218838;
}

.message {
  margin-top: 10px;
  font-size: 14px;
}

.success {
  color: green;
}

.error {
  color: red;
}
```

Output:

Login

<input type="text" value="Username"/>	<input type="text" value="Password"/>	<input type="button" value="Login"/>
---------------------------------------	---------------------------------------	--------------------------------------

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

Welcome to Online Bookstore

Name:	<input type="text"/>
Password:	<input type="text"/>
<input type="button" value="Submit"/>	

a) Forgot Password Form

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>User Registration</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f4f4f4;
      display: flex;
      align-items: center;
      justify-content: center;
      height: 100vh;
    }
    .container {
      background: #fff;
      padding: 20px 30px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
      width: 300px;
    }
    input {
      width: 100%;
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    button {
      width: 100%;
      padding: 10px;
      background: #28a745;
      border: none;
      color: #fff;
      border-radius: 4px;
      cursor: pointer;
    }
    button:hover {
      background: #218838;
    }
    .links {
      text-align: center;
      margin-top: 10px;
    }
  </style>
</html>
```

```
.links a {
  text-decoration: none;
  color: #007bff;
}
.links a:hover {
  text-decoration: underline;
}
</style>
</head>
<body>
  <div class="container">
    <h2>Register</h2>
    <form id="registrationForm">
      <input type="text" id="username" placeholder="Username" required />
      <input type="email" id="email" placeholder="Email" required />
      <input type="password" id="password" placeholder="Password" required />
      <input type="password" id="confirmPassword" placeholder="Confirm
Password" required />
      <button type="submit">Register</button>
    </form>
    <div class="links">
      <!-- Link to Forgot Password page -->
      <p><a href="fp.html">Forgot Password?</a></p>
    </div>
  </div>

  <script>
    document.getElementById("registrationForm").addEventListener("submit",
function(e) {
    e.preventDefault();

    // Get values
    const username = document.getElementById("username").value.trim();
    const email = document.getElementById("email").value.trim();
    const password = document.getElementById("password").value;
    const confirmPassword =
document.getElementById("confirmPassword").value;

    // Validate Username: not empty and at least 5 characters
    if (username === "" || username.length < 5) {
      alert("Username must be at least 5 characters long.");
      return;
    }

    // Validate Password: not empty and at least 8 characters
    if (password === "" || password.length < 8) {
      alert("Password must be at least 8 characters long.");
      return;
    }
  }
  </script>
```

```
    }

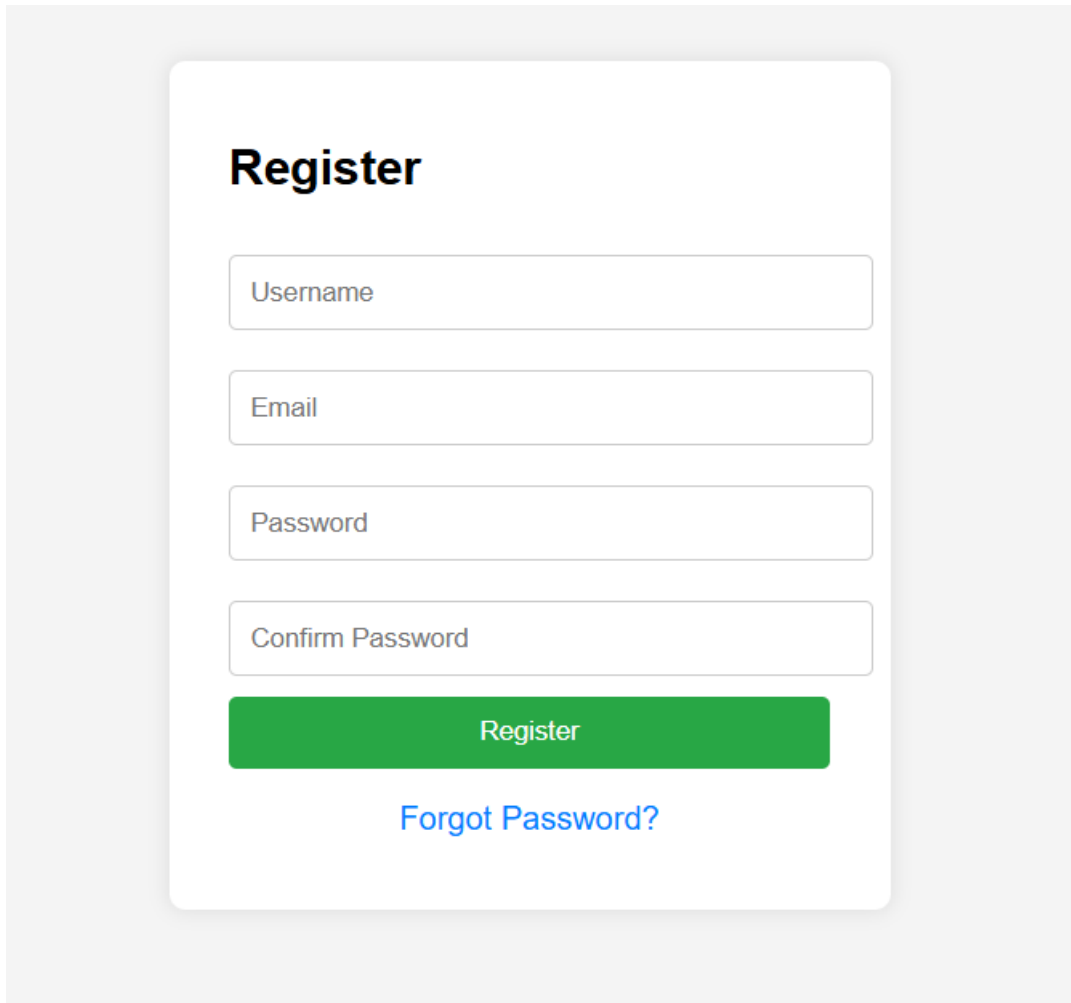
    // Validate Confirm Password: should match Password
    if (password !== confirmPassword) {
        alert("Passwords do not match.");
        return;
    }

    // Store user data in localStorage (keyed by email)
    const user = { username, email, password };
    localStorage.setItem(email, JSON.stringify(user));

    alert("Registration successful!");

    // Redirect to index.html after successful registration
    window.location.href = "index.html";
});
</script>
</body>
</html>
```


Output:



Register

Username

Email

Password

Confirm Password

Register

[Forgot Password?](#)

b) Dynamically Content Loading

Html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="./index.css" />
  <title>Welcome To Online Bookstore</title>
</head>
<style>
  /* Basic styles for layout */
  body {
    font-family: Arial, sans-serif;
    background: #f4f4f4;
    margin: 0;
    padding: 20px;
  }
  h1 {
    text-align: center;
    margin-bottom: 20px;
  }
  .container {
    display: flex;
    max-width: 1000px;
    margin: 0 auto;
    background: #fff;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
  }
  .left {
    width: 30%;
    padding: 20px;
    border-right: 1px solid #ddd;
  }
  .right {
    width: 70%;
    padding: 20px;
  }
  .left h3 {
    margin-top: 0;
  }
  .dropdown, .dropdown1, .dropdown2, .misc {
    margin-bottom: 15px;
  }
  button, .misc {
    width: 100%;
    padding: 10px;
    border: none;
  }
```

```
background: #007bff;
color: white;
font-size: 16px;
cursor: pointer;
border-radius: 4px;
text-align: left;
}
button:hover, .misc:hover {
background: #0056b3;
}
/* Book list styles */
.book-list {
list-style-type: none;
padding: 0;
}
.book-list li {
margin-bottom: 20px;
padding: 10px;
border-bottom: 1px solid #eee;
}
.book-list li:last-child {
border-bottom: none;
}
.book-title {
font-weight: bold;
margin-bottom: 5px;
}
.book-image {
width: 150px;
height: auto;
display: block;
}
</style>
</head>
<body>
<h1>Welcome To Online Bookstore</h1>
<div class="container">
<div class="left">
<h3>Categories</h3>
<!-- AI & DS Category -->
<div class="dropdown">
<button id="ai_ds">AI & DS</button>
</div>
<!-- Data Science Category -->
<div class="dropdown1">
<button id="data_science">Data Science</button>
</div>
<!-- CyberSecurity Category -->
```

```
<div class="dropdown2">
  <button id="cyber_security">CyberSecurity</button>
</div>
<!-- Miscellaneous Category -->
<div class="misc">
  <button id="miscellaneous">Miscellaneous</button>
</div>
</div>
<div class="right">
  <h2>Explore our collection</h2>
  <p>Click on a category to view books</p>
  <ul id="bookList" class="book-list"></ul>
</div>
</div>

<script>
  // Sample book data for each category (each book has a title and an image
URL)
  const booksData = {
    ai_ds: [
      {
        title: "Artificial Intelligence: A Modern Approach",
        image: "https://m.media-amazon.com/images/I/61-
6TTTBZel._AC_UF1000,1000_QL80_.jpg"
      },
      {
        title: "Data Structures & Algorithms in Java",
        image: "https://m.media-
amazon.com/images/I/91s8Z61R87L._AC_UF1000,1000_QL80_.jpg"
      },
      {
        title: "Python for Data Analysis",
        image: "https://m.media-amazon.com/images/I/6161h-ZkhuL.jpg"
      }
    ],
    data_science: [
      {
        title: "Data Science from Scratch",
        image: "https://m.media-amazon.com/images/I/715wQxqDKTL.jpg"
      },
      {
        title: "Python Data Science Handbook",
        image: "https://m.media-amazon.com/images/I/91Yqv5wWuPL.jpg"
      },
      {
```

```
        title: "Big Data: Principles and Best Practices",
        image: "https://m.media-
amazon.com/images/I/51Zr1+i+brL._UF1000,1000_QL80_.jpg"
    },
],
cyber_security: [

    {
        title: "Hacking: The Art of Exploitation",
        image: "https://m.media-amazon.com/images/I/81VrvsAwAmL.jpg"
    },
    {
        title: "Cybersecurity Essentials",
        image: "https://m.media-
amazon.com/images/I/7109NPQA5cL._AC_UF1000,1000_QL80_.jpg"
    },
    {
        title: "Network Security Fundamentals",
        image: "https://m.media-
amazon.com/images/I/710tLdh2etL._AC_UF1000,1000_QL80_.jpg"
    },
],
miscellaneous: [
    {
        title: "The Great Gatsby",
        image: "https://m.media-
amazon.com/images/I/81TLiZrasVL._UF1000,1000_QL80_.jpg"
    },
    {
        title: "To Kill a Mockingbird",
        image: "https://m.media-
amazon.com/images/I/81gepf1eMqL._AC_UF1000,1000_QL80_.jpg"
    },
    {
        title: "Pride and Prejudice",
        image: "https://m.media-
amazon.com/images/I/81Scutrtj4L._UF1000,1000_QL80_.jpg"
    },
]
];

// Function to display books in the right panel
function displayBooks(category) {
    const bookListEl = document.getElementById("bookList");
    // Clear previous list
    bookListEl.innerHTML = "";
```

```
// Get random books from the selected category
const books = booksData[category];

// Shuffle the books array to randomize order
const shuffledBooks = books.sort(() => 0.5 - Math.random());
// Display 3 random books (or fewer if there are less than 3)
const booksToShow = shuffledBooks.slice(0, 3);

booksToShow.forEach(book => {
  const li = document.createElement("li");

  // Create an element for the book title
  const titleEl = document.createElement("div");
  titleEl.className = "book-title";
  titleEl.textContent = book.title;

  // Create an image element for the book cover
  const imgEl = document.createElement("img");
  imgEl.className = "book-image";
  imgEl.src = book.image;
  imgEl.alt = book.title;

  // Append title and image to the list item
  li.appendChild(titleEl);
  li.appendChild(imgEl);

  bookListEl.appendChild(li);
});
}

// Add event listeners to the category buttons
document.getElementById("ai_ds").addEventListener("click", function() {
  displayBooks("ai_ds");
});
document.getElementById("data_science").addEventListener("click",
function() {
  displayBooks("data_science");
});
document.getElementById("cyber_security").addEventListener("click",
function() {
  displayBooks("cyber_security");
});
document.getElementById("miscellaneous").addEventListener("click",
function() {
  displayBooks("miscellaneous");
});
});
</script>
```

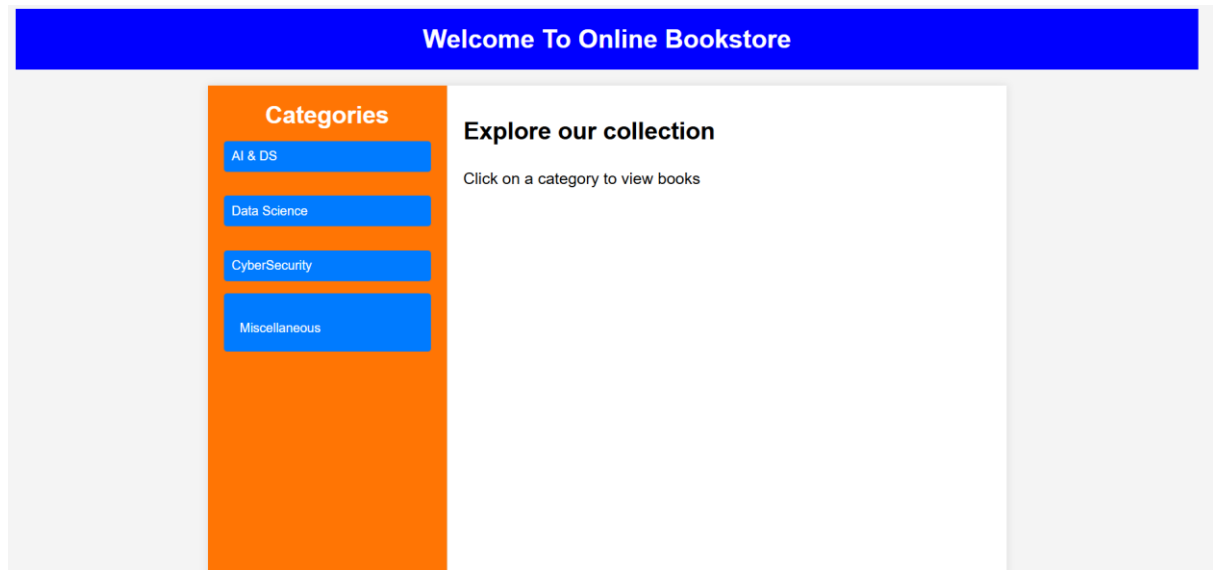
```
</body>
</html>
```

CSS:

```
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
h1{
    background-color: blue;
    color: white;
    padding: 20px;
    text-align: center;
}
.container{
    display: grid;
    grid-template-columns: 400px 1fr;
}
.left{
    background-color: rgb(255, 117, 4);
    height: 100vh;
    text-align: center;
}
.right{
    padding-left: 10px;
    padding-top: 10px;
}
h2{
    margin-top: 20px;
    margin-bottom: 30px;
    font-size: 30px;
}
p{
    font-size: 20px;
}
h3{
    font-size: 30px;
    color: white;
    margin-top: 30px;
}
button{
    border: none;
    font-size: 20px;
    text-align: center;
    background-color: rgb(255, 117, 4) ;
    color: white;
```

```
margin-top: 15px;
cursor: pointer;
}
.mis{
  color: white;
  margin-top: 15px;
  font-size: 25px;
}
.dropdown{
  position: relative;
}
.dropdown-menu{
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 80px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}
.dropdown-menu a{
  display: block;
  text-decoration: none;
  color: white;
  background-color: blue;
}
```


Output:



WEEK-4: Shopping Cart Functionality:

```
<!DOCTYPE html><html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shopping Cart</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .container {
      display: flex;
      gap: 20px;
    }
    .box {
      border: 1px solid #ccc;
      padding: 10px;
      width: 45%;
    }
    button {
      margin-left: 10px;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <h1>Simple Shopping Cart</h1>
  <div class="container">
    <div class="box">
      <h2>Available Books</h2>
      <div id="book-list"></div>
    </div>
    <div class="box">
      <h2>Shopping Cart</h2>
      <div id="cart"></div>
    </div>
  </div><script>
const books = [
  { id: 1, title: "The Great Gatsby", price: 10 },
  { id: 2, title: "1984", price: 15 },
  { id: 3, title: "To Kill a Mockingbird", price: 12 }
];
let cart = JSON.parse(localStorage.getItem("cart")) || [];

function addToCart(bookId) {
  const book = books.find(b => b.id === bookId);
  if (book) {
    cart.push(book);
    localStorage.setItem("cart", JSON.stringify(cart));
    updateCartDisplay();
  }
}
```

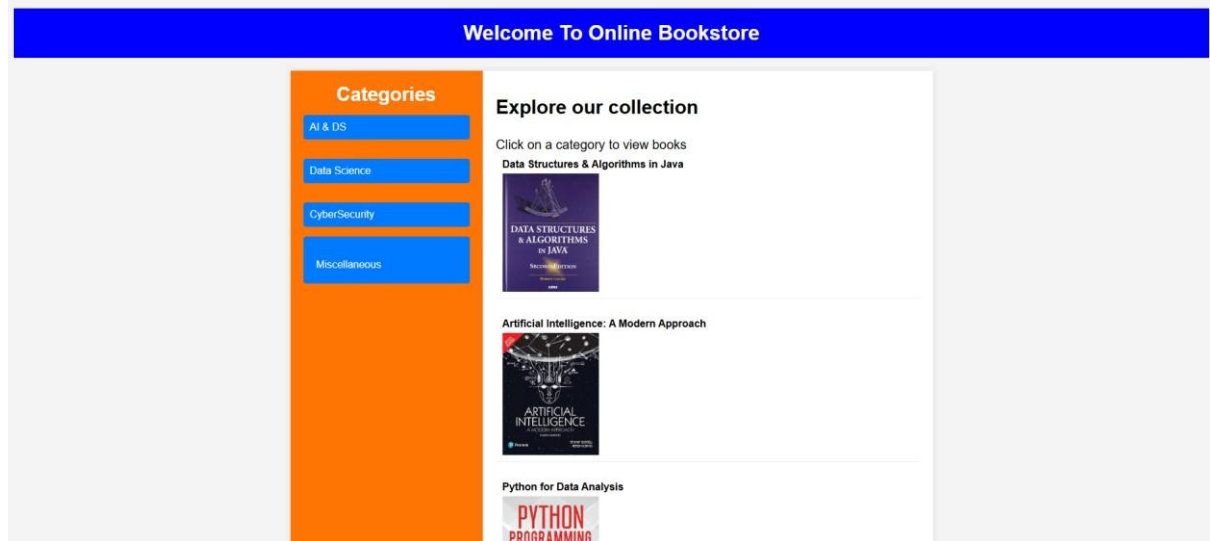
```
    }

    function updateCartDisplay() {
        const cartContainer = document.getElementById("cart");
        cartContainer.innerHTML = "";
        cart.forEach(item => {
            const cartItem = document.createElement("div");
            cartItem.textContent = `${item.title} - ${item.price}`;
            cartContainer.appendChild(cartItem);
        });
    }

    document.addEventListener("DOMContentLoaded", () => {
        const bookList = document.getElementById("book-list");
        books.forEach(book => {
            const bookItem = document.createElement("div");
            bookItem.innerHTML = `${book.title} - ${book.price} <button
onclick="addToCart(${book.id})">Add to Cart</button>`;
            bookList.appendChild(bookItem);
        });
        updateCartDisplay();
    });
</script>

</body>
</html>
```

Output:



WEEK-5: XML Structure and Design

```
<?xml version="1.0" encoding="UTF-8"?>
<book_catalog>
  <book id="101">
    <title>The Alchemist</title>
    <author>Paulo Coelho</author>
    <genre>Fiction</genre>
    <price currency="USD">9.99</price>
    <availability>In Stock</availability>
  </book>

  <book id="102">
    <title>Sapiens</title>
    <author>Abli</author>
    <genre>History</genre>
    <price currency="USD">14.99</price>
  </book>
</book_catalog>
```

Output:

The Alchemist

Author: Paulo Coelho

Genre: Fiction

Price: 9.99 USD

Availability: In Stock

Sapiens

Author: Abli

Genre: History

Price: 14.99 USD

WEEK-6: User Account XML

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <book>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <genre>Fiction</genre>
    <price currency="USD">18.99</price>
    <availability>In Stock</availability>
  </book>
  <book>
    <title>1984</title>
    <author>George Orwell</author>
    <genre>Dystopian</genre>
    <price currency="USD">15.99</price>
    <availability>Out of Stock</availability>
  </book>
  <book>
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
    <genre>Fiction</genre>
    <price currency="USD">12.50</price>
    <availability>In Stock</availability>
  </book>
  <book>
    <title>Harry Potter and the Sorcerer's Stone</title>
    <author>J.K. Rowling</author>
    <genre>Fantasy</genre>
    <price currency="USD">22.99</price>
    <availability>In Stock</availability>
  </book>
  <book>
    <title>Becoming</title>
    <author>Michelle Obama</author>
    <genre>Biography</genre>
    <price currency="USD">14.99</price>
    <availability>In Stock</availability>
  </book>
</catalog>
```

STORING USER ACCOUNT INFORMATION

```
<?xml version="1.0" encoding="UTF-8"?>
<UserAccounts>
  <User>
    <Username>john_doe</Username>
    <Password
encrypted="true">a94a8fe5ccb19ba61c4c0873d391e987982fbbd3</Password>

    <Email>john.doe@example.com</Email>
    <PurchaseHistory>
      <Purchase>
        <Item>Smartphone</Item>
        <Price currency="USD">699.99</Price>
        <Date>2025-03-20</Date>
      </Purchase>
      <Purchase>
        <Item>Laptop</Item>
        <Price currency="USD">1299.99</Price>
        <Date>2025-02-15</Date>
      </Purchase>
    </PurchaseHistory>
  </User>
</UserAccounts>
```

WEEK-7: Install and configure Apache Tomcat server in Eclipse

```
C:\Users\Vinay>java --version
java 16.0.1 2021-04-20
Java(TM) SE Runtime Environment (build 16.0.1+9-24)
Java HotSpot(TM) 64-Bit Server VM (build 16.0.1+9-24, mixed mode, sharing)

C:\Users\Vinay>
```

Prerequisites

- **Eclipse IDE for Java EE Developers** (Download from [Eclipse Official Site](#))
- **Apache Tomcat Server** (Download from [Apache Tomcat Official Site](#))
- **Java Development Kit (JDK)** installed and configured

Step 1: Download and Install Apache Tomcat

1. Visit the [Apache Tomcat download page](#).
2. Download the latest **Tomcat 9** or **Tomcat 10** binary distribution (.zip or .tar.gz).
3. Extract the downloaded file to a directory of your choice (e.g., C:\apache-tomcat-9.0.XX on Windows or /opt/tomcat on Linux/Mac).

Step 2: Install Eclipse IDE

1. Download **Eclipse IDE for Java EE Developers** from [Eclipse Downloads](#).
2. Install and launch Eclipse.

Step 3: Configure Apache Tomcat in Eclipse

1. **Open Eclipse** and go to the "Servers" tab.
 - If the **Servers** tab is not visible:
 - Click on **Window** → **Show View** → **Servers**.
2. **Add a New Server**
 - Right-click inside the **Servers** tab and select **New** → **Server**.
 - Expand **Apache** and select **Tomcat v9.0 Server** (or your installed version).
 - Click **Next**.
3. **Specify Tomcat Installation Directory**
 - Click **Browse** and select the folder where Tomcat is extracted.
 - Click **Finish**.

4. Verify Configuration

- Right-click on **Tomcat v9.0 Server** in the **Servers** tab and select **Open**.
- Under **Server Locations**, select **Use Tomcat installation (takes control of Tomcat installation)**.
- Save the configuration.

Step 4: Start and Test Tomcat Server

1. Start the Server

- Right-click on **Tomcat v9.0 Server** in the **Servers** tab and click **Start**.
- If the server starts successfully, the configuration is correct.

2. Test the Server

- Open a web browser and visit: `http://localhost:8080`
- You should see the **Apache Tomcat welcome page**.

Step 5: Deploy a Web Application

1. Create a Dynamic Web Project

- Go to **File** → **New** → **Dynamic Web Project**.
- Enter a project name (e.g., `MyWebApp`).
- Select **Tomcat v9.0** as the Target Runtime.
- Click **Finish**.

2. Add a JSP Page for Testing

- Inside the `WebContent` folder, right-click → **New** → **JSP File**.
- Name it `index.jsp` and add the following content:
 - `<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>`
 - `<html>`
 - `<head><title>Welcome</title></head>`
 - `<body>`
 - `<h2>Hello, Apache Tomcat is Running!</h2>`

- `</body>`

- `</html>`

3. Deploy the Project to Tomcat

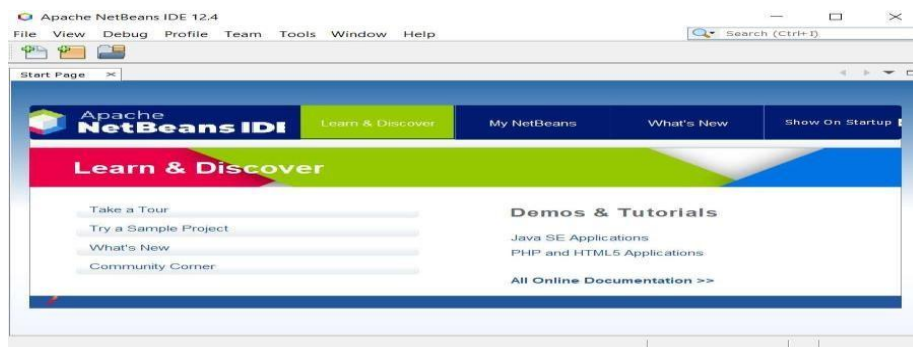
- Right-click on the project → **Run As** → **Run on Server**.
- Choose **Tomcat v9.0** and click **Finish**.

4. Test the Web Application

- Open a browser and visit: `http://localhost:8080/MyWebApp/index.jsp`
- You should see the message "**Hello, Apache Tomcat is Running!**".

Step 6: Stop and Restart Tomcat

- **Stop:** Right-click on **Tomcat v9.0 Server** → Click **Stop**.
- **Restart:** Right-click on **Tomcat v9.0 Server** → Click **Restart**.



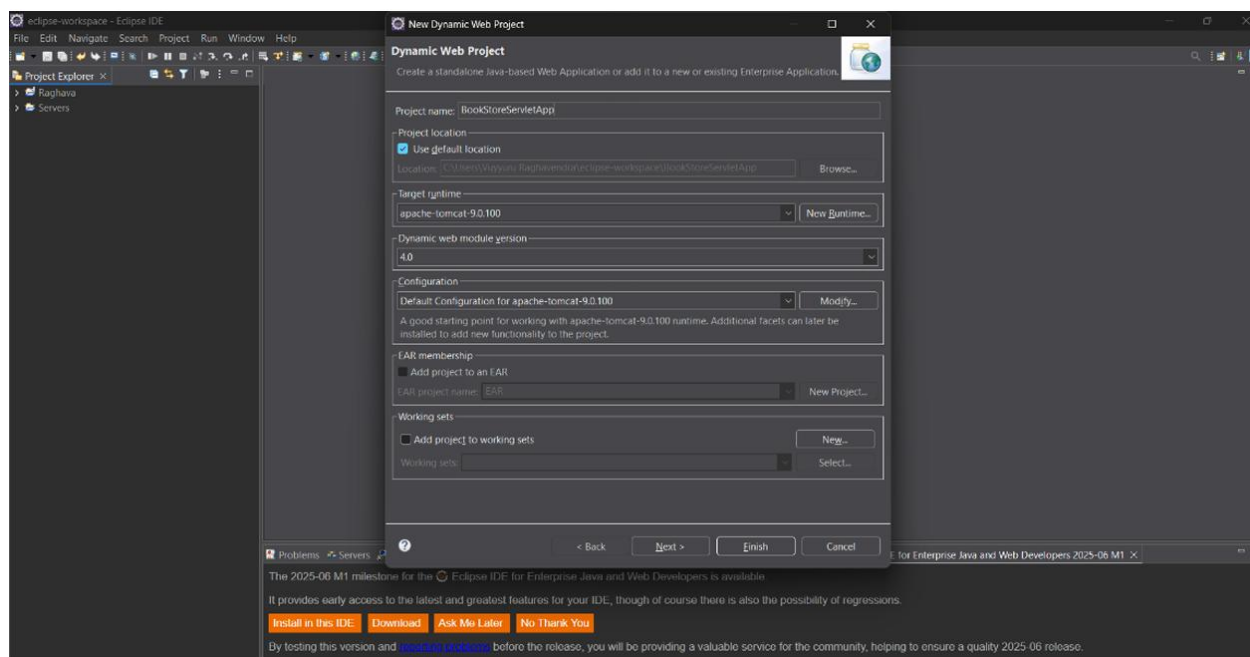
First look of Apache NetBeans IDE

Week 8: Servlet for Book Retrieval - Develop a servlet that retrieves book information from a database and sends it to the content frame.

1. Setup Project

• Create a New Dynamic Web Project:

- o Open Eclipse and select File > New > Dynamic Web Project.
- o Name the project (e.g., BookStoreServletApp).
- o Select Apache Tomcat as the runtime environment.
- o Finish creating the project.



2. Install Apache Tomcat Server

• Install Apache Tomcat if not already done. Follow the steps for setting up Tomcat in Eclipse:

- o Go to Window > Preferences > Server > Runtime Environments.
- o Add Apache Tomcat if it's not listed. Select the path where Apache Tomcat is installed.

3. Configure MySQL Database

• Create Database and Table:

- o Use MySQL Workbench or command-line client to create a database bookstore:

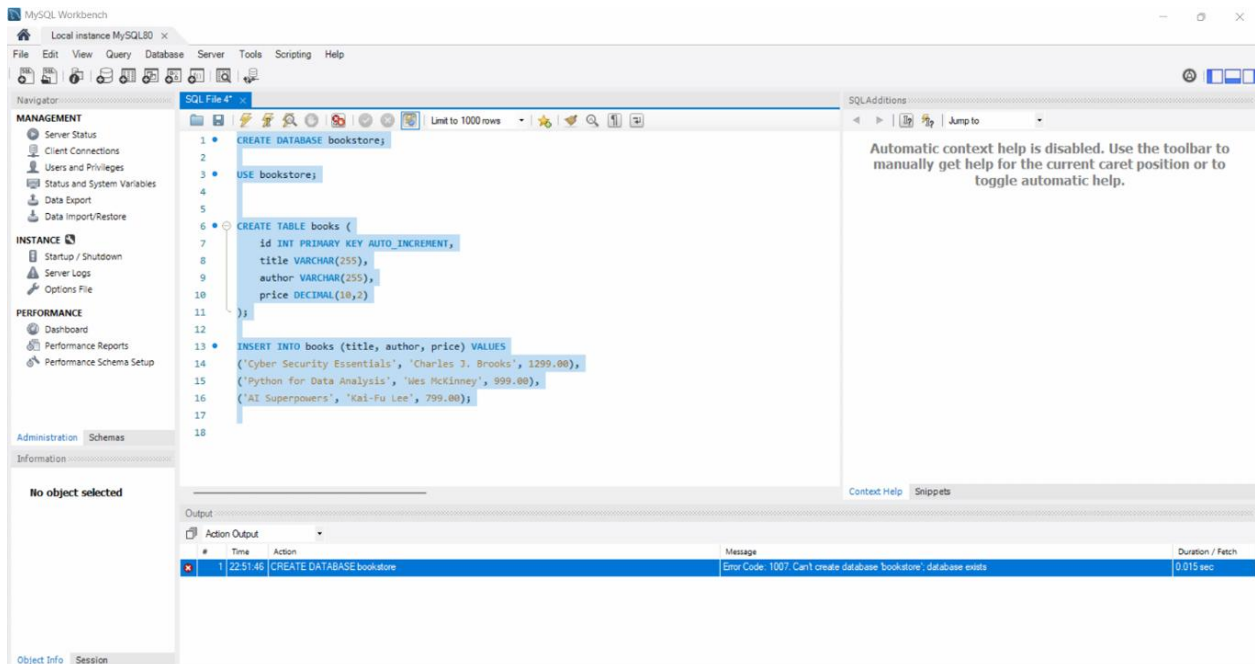
```
CREATE DATABASE bookstore;
```

```
USE bookstore;
```

```
CREATE TABLE books (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    title VARCHAR(255),  
    author VARCHAR(255),  
    price DECIMAL(10,2)  
);
```

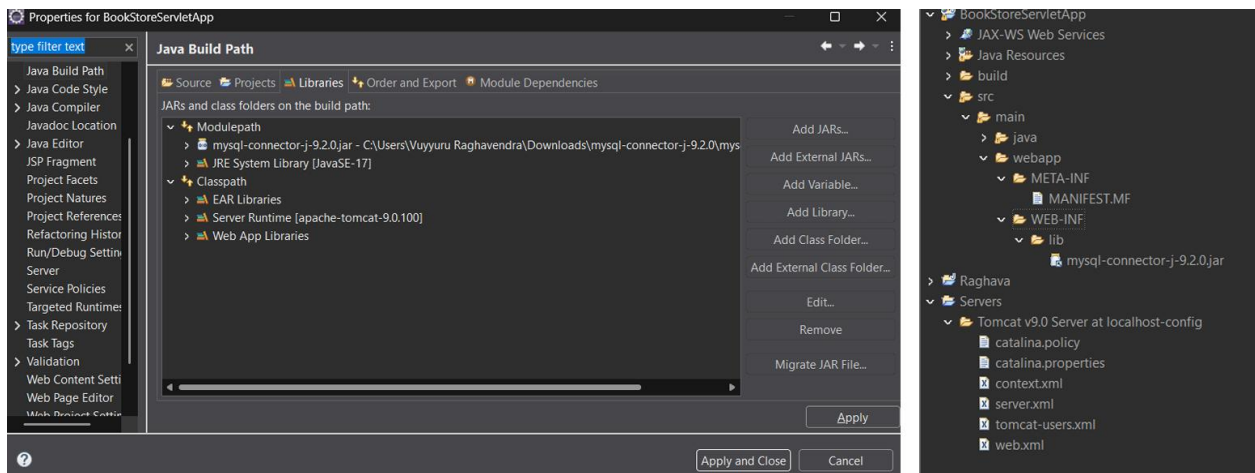
```
INSERT INTO books (title, author, price) VALUES
```

('Cyber Security Essentials', 'Charles J. Brooks', 1299.00),
('Python for Data Analysis', 'Wes McKinney', 999.00),
('AI Superpowers', 'Kai-Fu Lee', 799.00);



4. Add MySQL JDBC Driver

- o Download the MySQL Connector/J from the official website or from Maven.
- o Right-click the project > Build Path > Configure Build Path.
- o Go to the Libraries tab and click Add External JARs to add the .jar file.



5. Create the Servlet

• Write the BookServlet Code:

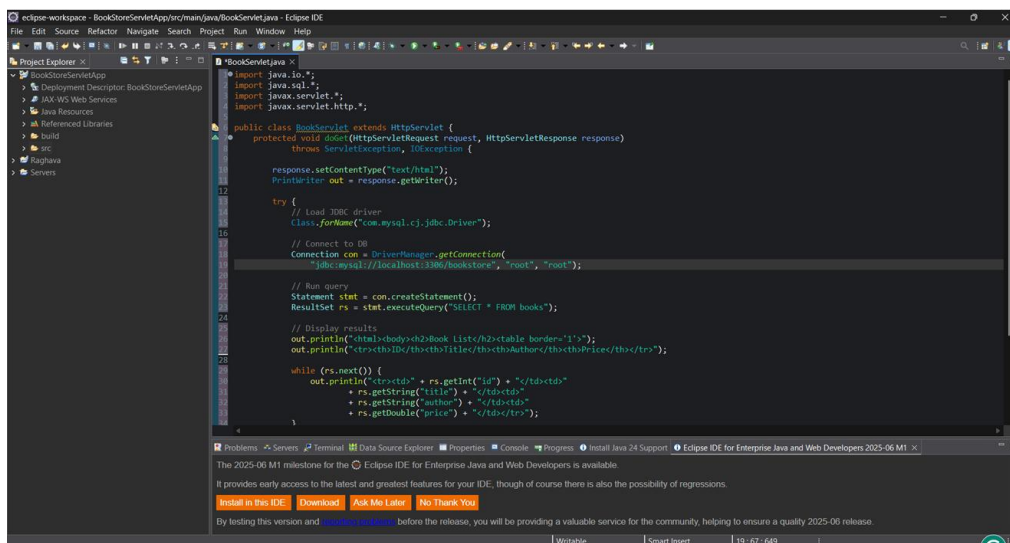
o Use the following code:

o Create a BookServlet.java file under src in your project.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/books")

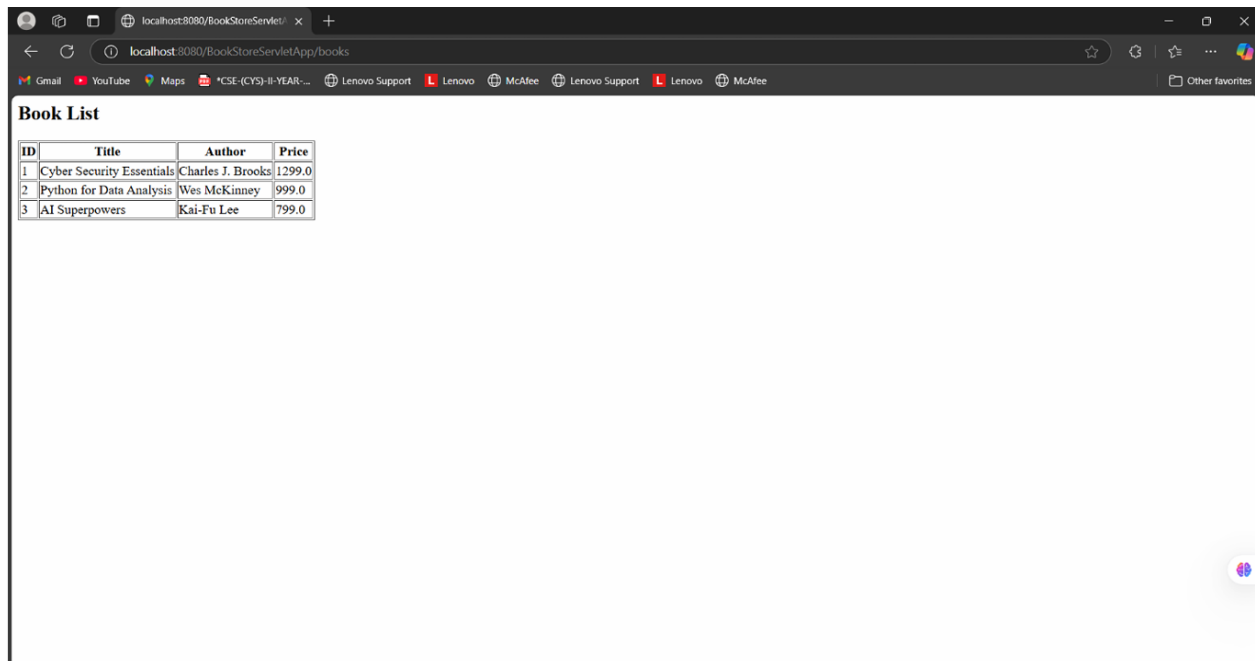
public class BookServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/bookstore", "root", "root");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM books");
            out.println("<html><body><h2>Book List</h2><table border='1'>");
            out.println("<tr><th>ID</th><th>Title</th><th>Author</th><th>Price</th></tr>");
            while (rs.next()) {
                out.println("<tr><td>" + rs.getInt("id") + "</td><td>"
                    + rs.getString("title") + "</td><td>"
                    + rs.getString("author") + "</td><td>"
                    + rs.getDouble("price") + "</td></tr>");
            }
            out.println("</table></body></html>");
            con.close();
        } catch (Exception e) {
            out.println("<p>Error: " + e.getMessage() + "</p>");
        }
    }
}
```



6. Test the Servlet

- Run the Project on Tomcat:
 - o Right-click on your project > Run on Server.
- <http://localhost:8080/YourProjectName/books>
- o You should see a table displaying the books from the database.

Output



WEEK-9: User Authentication: Implement user authentication using servlets and JSP.

Create a login page and restrict access to certain pages for authenticated users.

1. Set Up Project

First, make sure your project from Week 8 is set up correctly, and that you can run servlets.

2. Create the Login Page (JSP)

We'll start by creating a simple login page using JSP.

1. Create a new JSP page:

o Go to WebContent and create a new file named login.jsp.

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Login Page</title>
```

```
</head>
```

```
<body>
```

```
<h2>Login</h2>
```

```
<form action="LoginServlet" method="POST">
```

```
<label for="username">Username:</label>
```

```
<input type="text" name="username" id="username" required><br><br>
```

```
<label for="password">Password:</label>
```

```
<input type="password" name="password" id="password" required><br><br>
```

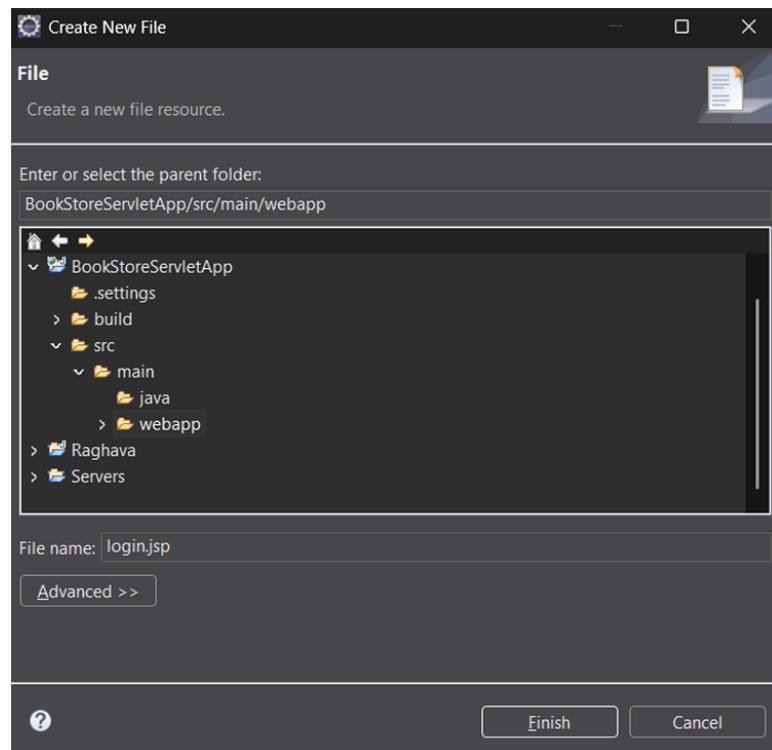
```
<button type="submit">Login</button>
```

```
</form>
```

```
</body>
```

```
</html>
```

o This page will collect the user's credentials and submit them to a servlet (LoginServlet) for authentication.



3. Create the Login Servlet

Create a new servlet (LoginServlet.java) that will handle the login logic.

1. Create LoginServlet.java:

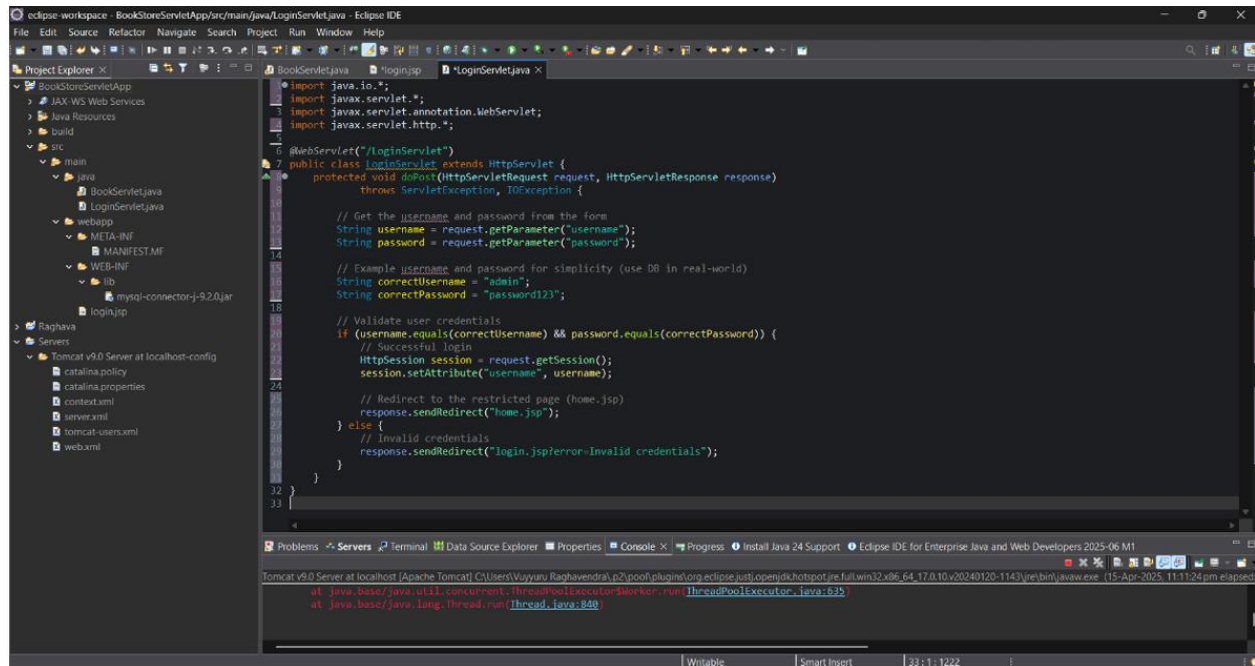
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Get the username and password from the form
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        // Example username and password for simplicity (use DB in real-world)
        String correctUsername = "admin";
        String correctPassword = "password123";
    }

    // Validate user credentials
    if (username.equals(correctUsername) && password.equals(correctPassword)) {
        // Successful login
        HttpSession session = request.getSession();
        session.setAttribute("username", username);
        // Redirect to the restricted page (home.jsp)
        response.sendRedirect("home.jsp");
    } else {
        // Invalid credentials
        response.sendRedirect("login.jsp?error=Invalid credentials");
    }
}
```

o Explanation:

- When the user submits the form, the servlet checks if the username and password match the correct credentials.
- If the login is successful, the username is stored in the session.
- If login fails, the user is redirected back to the login page with an error message.

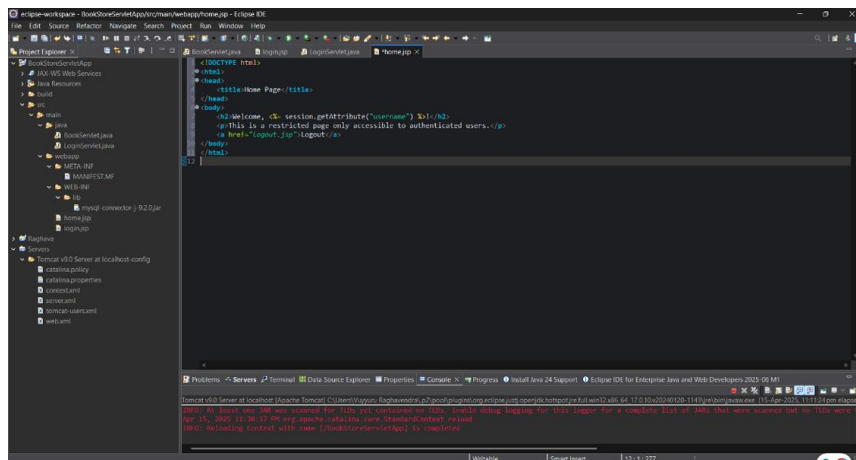


4. Create a Restricted Page (home.jsp)

1. Create home.jsp:

Next, create a restricted page that only authenticated users should be able to access.

```
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>
</head>
<body>
<h2>Welcome, <%= session.getAttribute("username") %>!</h2>
<p>This is a restricted page only accessible to authenticated users.</p>
<a href="logout.jsp">Logout</a>
</body>
</html>
```



o This page displays a welcome message using the username stored in the session.

5. Create the Logout Page (logout.jsp)

To allow users to log out, create a simple logout page that will invalidate their session.

1. Create logout.jsp:

```
<%
```

```
// Invalidate the session to log out the user
```

```
session.invalidate();
```

```
%>
```

o This page displays a welcome message using the username stored in the session.

```
<html>
```

```
<body>
```

```
<h2>You have successfully logged out.</h2>
```

```
<a href="login.jsp">Login Again</a>
```

```
</body>
```

```
</html>
```

o This page invalidates the session, effectively logging the user out, and provides a link to log in again.

6. Protect the Restricted Page (Servlet Filter)

To prevent unauthorized access to the restricted page, you can use a filter to check if the user is logged in before they access the page.

1. Create AuthFilter.java:

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.annotation.WebFilter;
```

```
import javax.servlet.http.*;
```

```
@WebFilter("/home.jsp") // Protect home.jsp with this filter
```

```
public class AuthFilter implements Filter {
```

```
    public void init(FilterConfig fConfig) throws ServletException {
```

```
    }
```

```
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
```

```
        HttpServletRequest req = (HttpServletRequest) request;
```

```
        HttpServletResponse resp = (HttpServletResponse) response;
```

```
        // Check if the user is logged in
```

```
        HttpSession session = req.getSession(false);
```

```
        if (session == null || session.getAttribute("username") == null) {
```

```
            // Redirect to login page if not logged in
```

```
            resp.sendRedirect("login.jsp");
```

```
        } else {
```

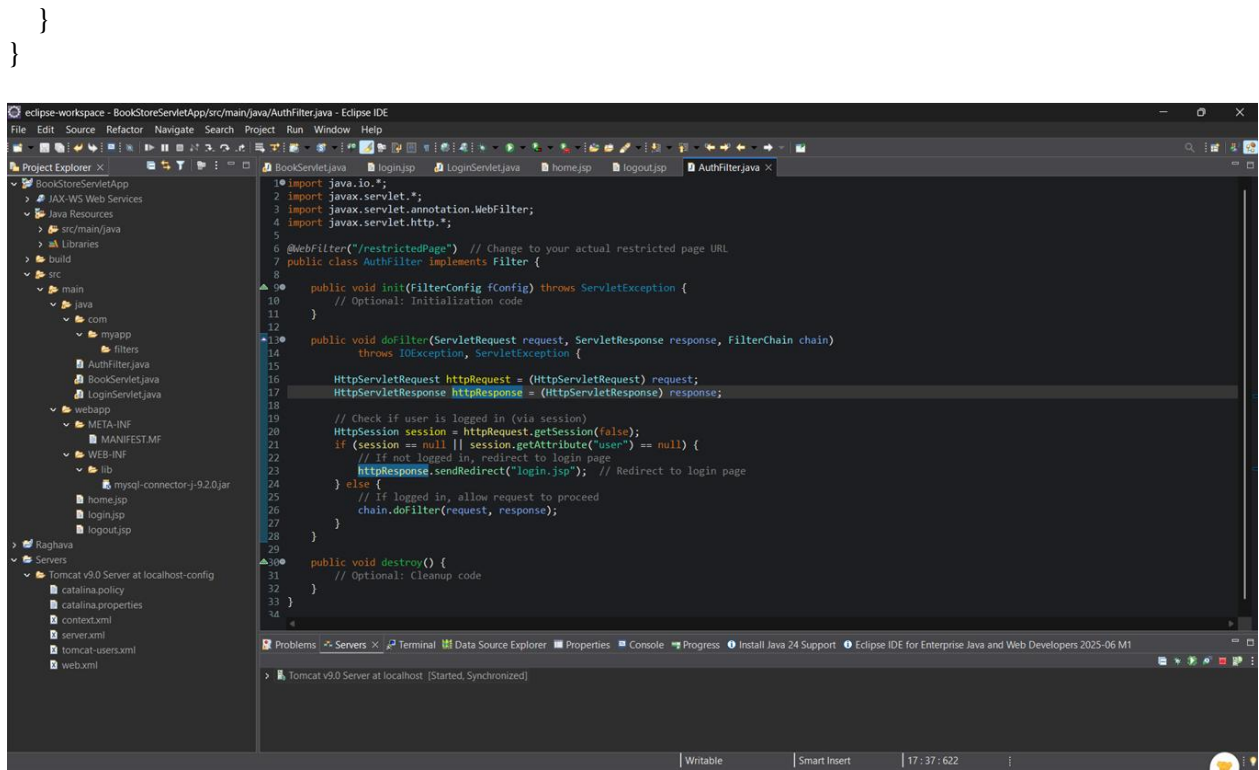
```
            // Allow the request to proceed to home.jsp
```

```
            chain.doFilter(request, response);
```

```
        }
```

```
    }
```

```
    public void destroy() {
```



7. Test the Application

1. Run the application on Tomcat.

2. Access the login page:

o Navigate to <http://localhost:8080/YourProjectName/login.jsp>.

o Try logging in with the username admin and password password123.

3. Test authentication:

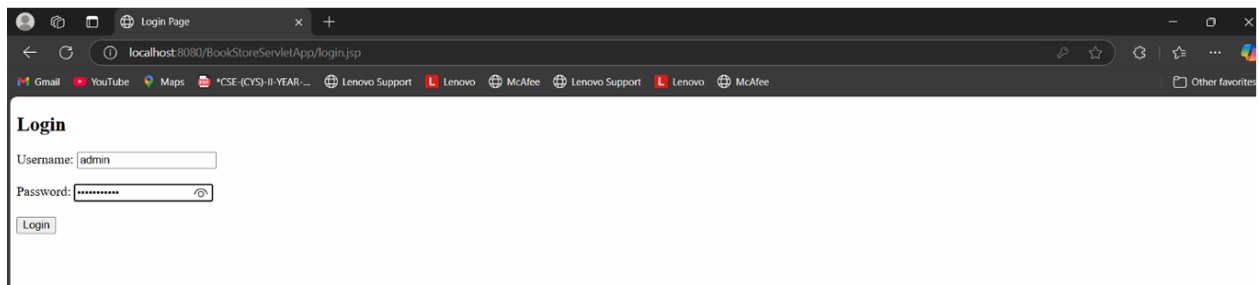
o Once logged in, you should be redirected to home.jsp, where you see a welcome message.

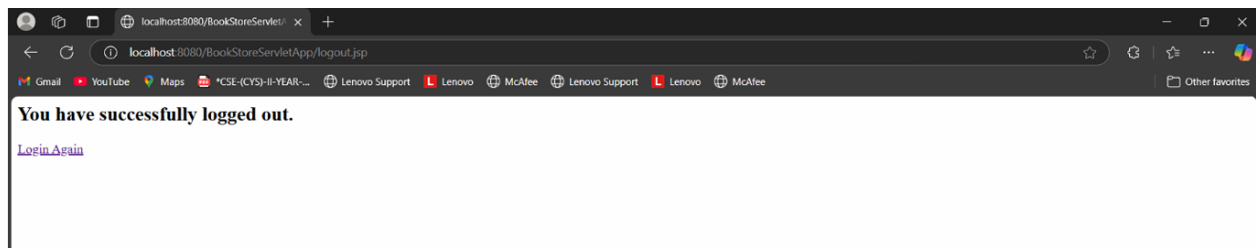
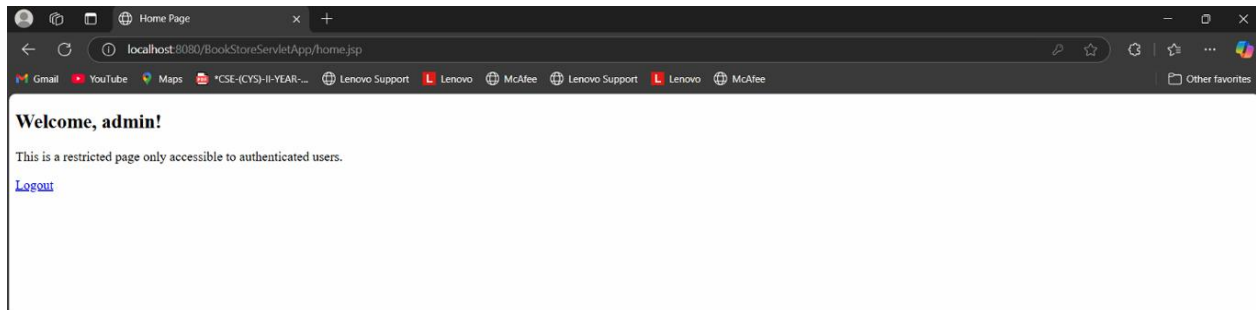
o If you try to access home.jsp without logging in, you should be redirected back to the login page.

4. Test logout:

o Click on the logout link, which should log you out and redirect you to the login page.

Output:

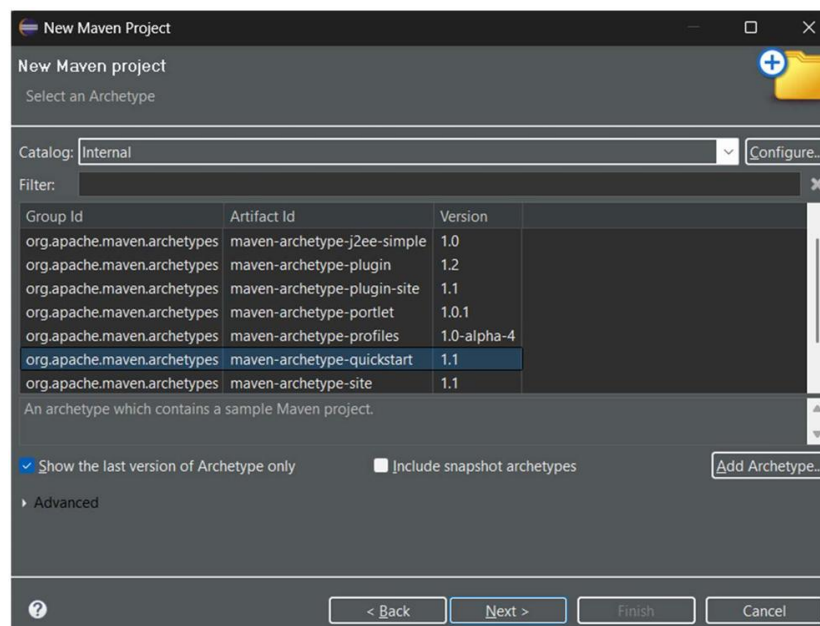
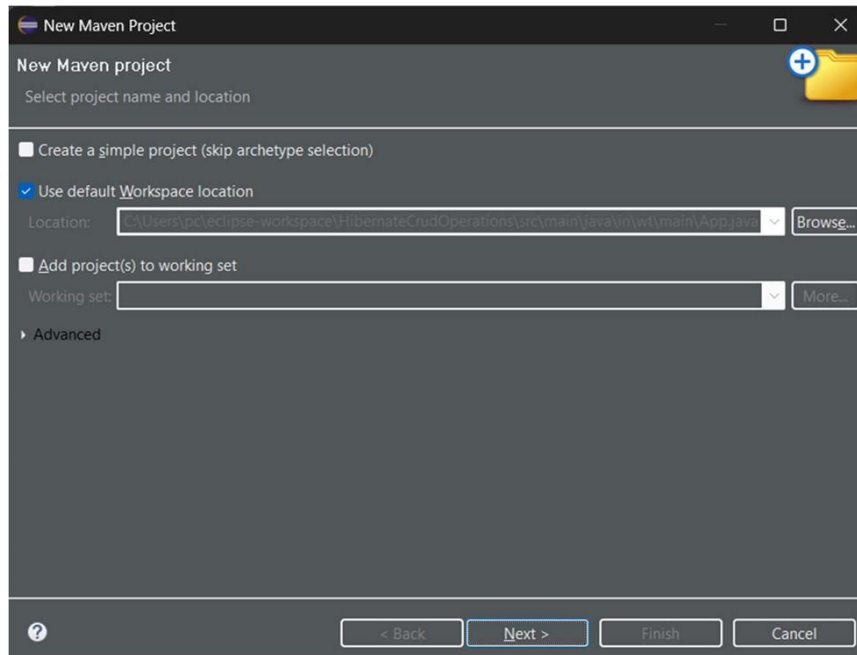


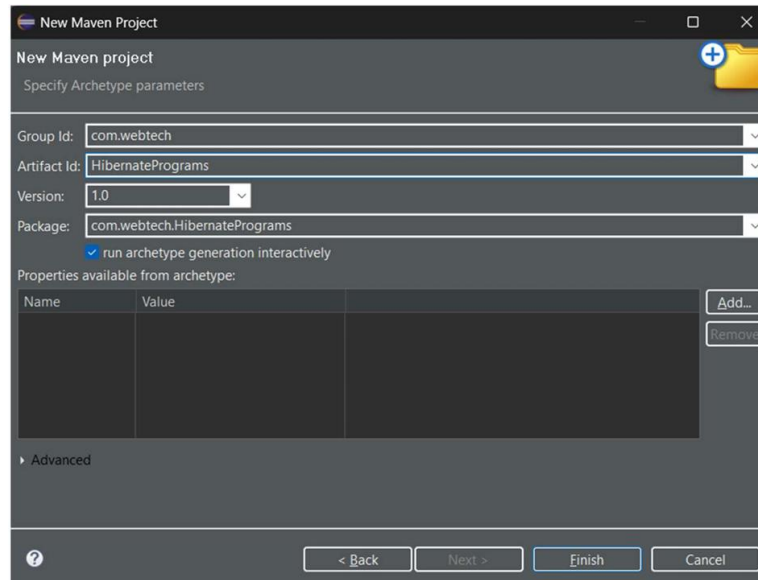


WEEK-10:

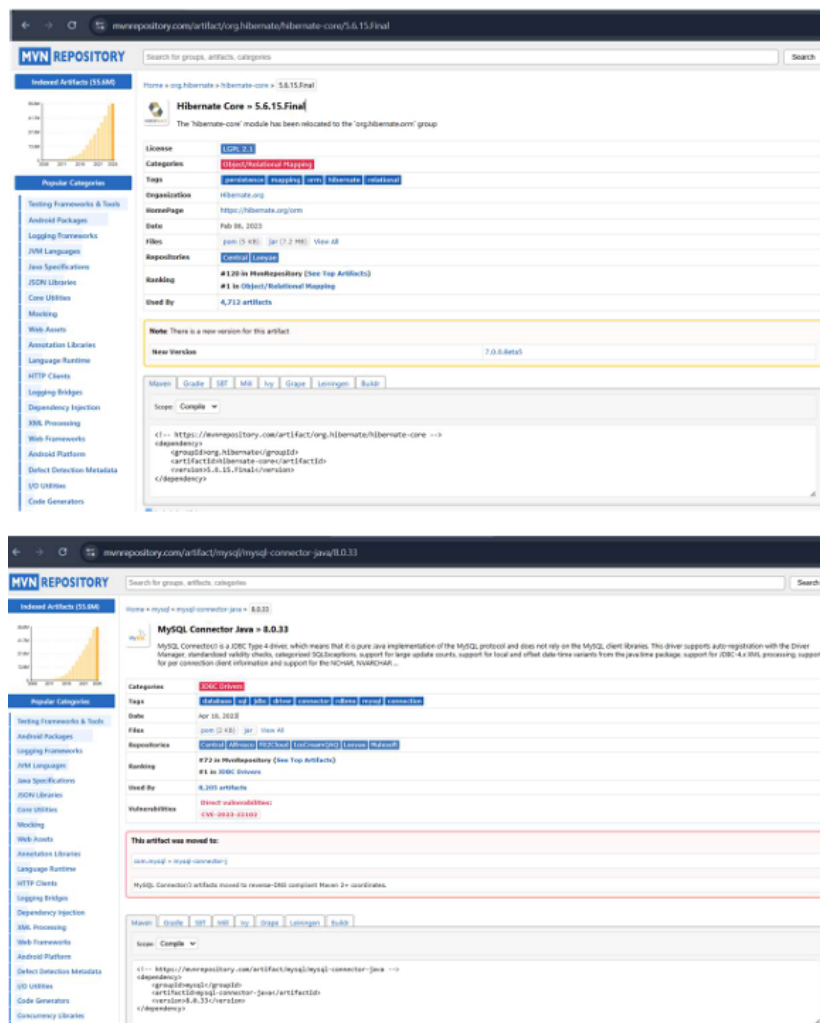
Hibernate: Hibernate Entity Mapping: Create Hibernate entity classes to represent books and users. Map these entities to corresponding database tables.

1. Create a Maven Project with GroupId (global package) and ArtifactId (application name)





2. Add the hibernate and mysql dependencies from maven repository into the pom.xml file.



```
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.webtech</groupId>
<artifactId>HibernatePrograms</artifactId>
<version>1.0</version>
<packaging>jar</packaging>

<name>HibernatePrograms</name>
<url>http://maven.apache.org</url>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>

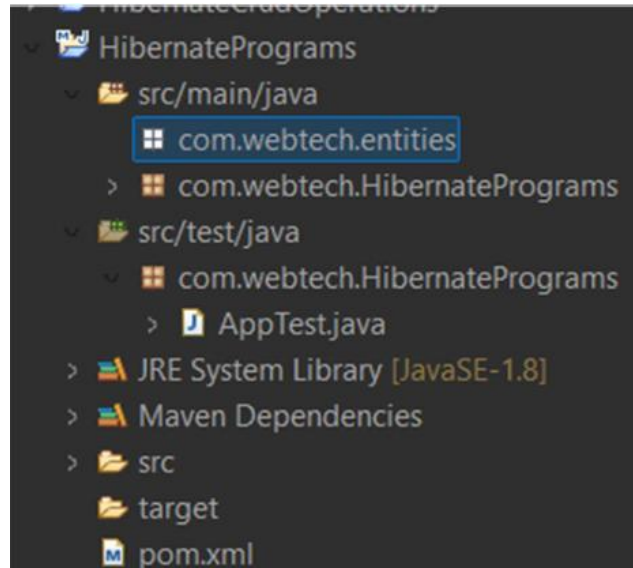
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.6.15.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>

<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.33</version>
</dependency>

</dependencies>
</project>
```

3. Create a package (entities) in src/main/java and add user and book classes.



```
user.java
package com.webtech.entities;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table
public class user {
    @Id
    @Column
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column
    private String name;
    @Column
    private String email;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```



```
}
    public int getId() {

        return id;
    }
    public void setId(int id) {

        this.id = id;
    }
    public user(String name, String email, int id) {
        super();
        this.name = name;
        this.email = email;
        this.id = id;
    }
    public user() {
        super();
    }
}

books.java

package com.webtech.entities;

import javax.persistence.*;

@Entity
@Table
public class books {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column
    private String title;

    @Column
    private String author;

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

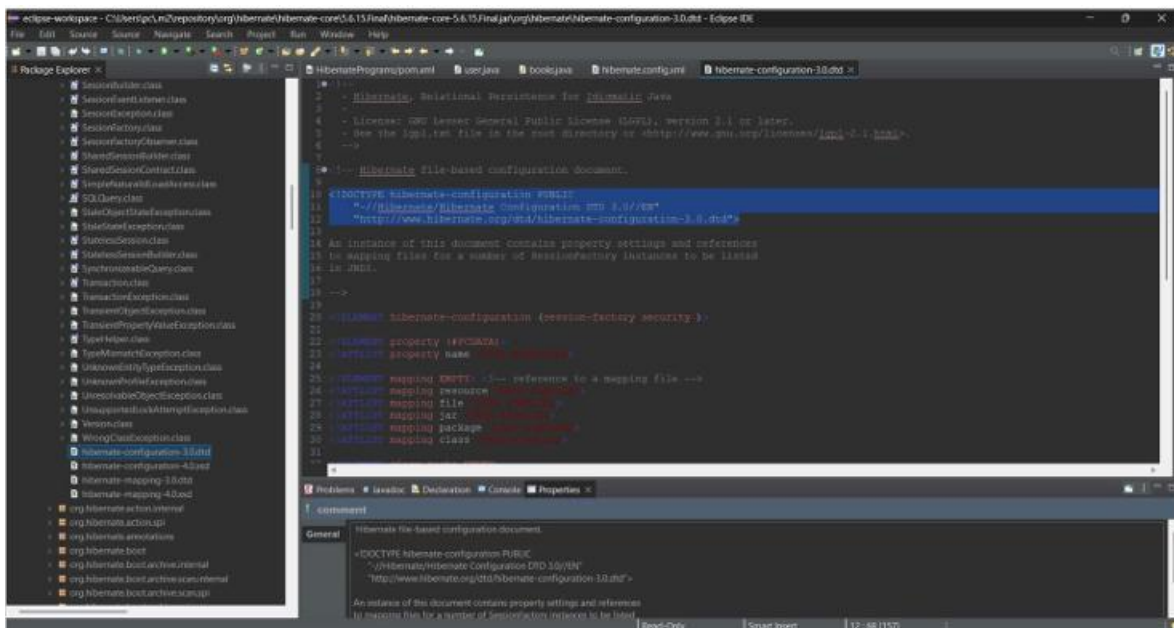
    public books() {}

    public books(String title, String author) {
```

```
this.title = title;
this.author = author;
}
}
```

4. Create a configuration file in src/main/java which holds the database connection and entity mapping details.

5. In maven dependencies go to hibernate core.jar and open the org.hibernate package and copy the schema from hibernate-configuration-3.0.dtd file .



6. Copy the schema into hibernate.config.xml file.

hibernate.config.xml

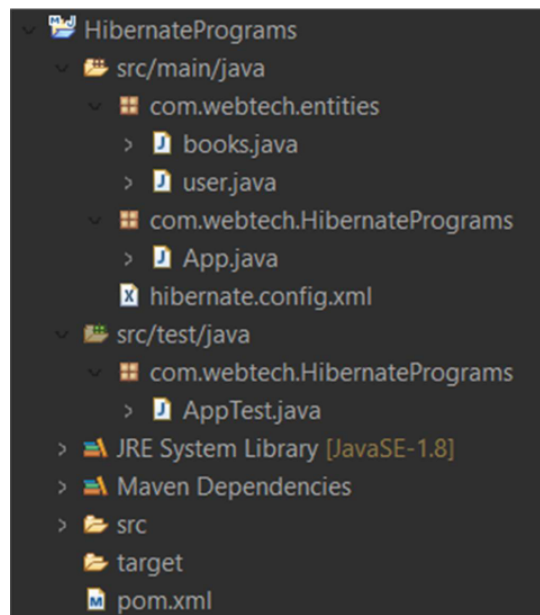
```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernate</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>
```

```
<property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
  <property name="show_sql">true</property>
  <property name="hbm2ddl.auto">update</property>

  <mapping class="com.webtech.entities.user"/>
  <mapping class="com.webtech.entities.books"/>
</session-factory>
</hibernate-configuration>
```

Directory-structure:



WEEK-11: CRUD Operations: Implement CRUD operations using Hibernate for managing book information in the database.

App.java

```
package com.webtech.HibernatePrograms;
import org.hibernate.cfg.Configuration;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import com.webtech.entities.user;
import com.webtech.entities.books;
import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        Configuration cfg = new Configuration();
        cfg.configure("hibernate.config.xml");
        cfg.addAnnotatedClass(user.class);
        cfg.addAnnotatedClass(books.class);
        SessionFactory sf = cfg.buildSessionFactory();
        Session session = sf.openSession();
        Scanner sc = new Scanner(System.in);
        int choice = 0;
        while (choice != -1) {
            System.out.println("\nChoose an option:");
            System.out.println("1. Create User");
            System.out.println("2. Create Book");
            System.out.println("3. Read User");
            System.out.println("4. Read Book");
            System.out.println("5. Update User Email");
            System.out.println("6. Update Book Author");
            System.out.println("7. Delete User");
            System.out.println("8. Delete Book");
            System.out.println("-1. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline
            int id;
            String name, email, title, author;
            switch (choice) {
                case 1:
                    System.out.print("Enter name: ");
                    name = sc.nextLine();
                    System.out.print("Enter email: ");
                    email = sc.nextLine();
                    createUser(session, name, email);
                    break;
                case 2:
                    System.out.print("Enter title: ");
                    title = sc.nextLine();
                    System.out.print("Enter author: ");
                    author = sc.nextLine();
```

```
        createBook(session, title, author);
        break;
    case 3:
        System.out.print("Enter user ID: ");
        id = sc.nextInt();
        readUser(session, id);
        break;
    case 4:
        System.out.print("Enter book ID: ");
        id = sc.nextInt();
        readBook(session, id);
        break;
    case 5:
        System.out.print("Enter user ID: ");
        id = sc.nextInt();
        sc.nextLine(); // consume newline
        System.out.print("Enter new email: ");
        email = sc.nextLine();
        updateUserEmail(session, id, email);
        break;
    case 6:
        System.out.print("Enter book ID: ");
        id = sc.nextInt();
        sc.nextLine(); // consume newline
        System.out.print("Enter new author: ");
        author = sc.nextLine();
        updateBookAuthor(session, id, author);
        break;
    case 7:
        System.out.print("Enter user ID to delete: ");
        id = sc.nextInt();
        deleteUser(session, id);
        break;
    case 8:
        System.out.print("Enter book ID to delete: ");
        id = sc.nextInt();
        deleteBook(session, id);
        break;
    case -1:
        System.out.println("Exiting program.");
        break;
    default:
        System.out.println("Invalid choice. Please try again.");
}
}
session.close();
sf.close();
sc.close();
}

static int createUser(Session session, String name, String email) {
    Transaction tx = session.beginTransaction();
```

```
        user user1 = new user();
        user1.setName(name);
        user1.setEmail(email);

int id = (int) session.save(user1);
    tx.commit();
    System.out.println("User created: " + name + " with ID: " + id);
    return id;
}

static void readUser(Session session, int id) {
    user user1 = session.get(user.class, id);
    if (user1 != null)
        System.out.println("User: " + user1.getName() + ", Email: " +
user1.getEmail());
    else
        System.out.println("User not found with id: " + id);
}

static void updateUserEmail(Session session, int id, String newEmail) {
    Transaction tx = session.beginTransaction();
    user user1 = session.get(user.class, id);
    if (user1 != null) {
        user1.setEmail(newEmail);
        session.merge(user1);
        tx.commit();
        System.out.println("Updated user email to: " + newEmail);
    } else {
        System.out.println("User not found for update.");
        tx.rollback();
    }
}

static void deleteUser(Session session, int id) {
    Transaction tx = session.beginTransaction();
    user user1 = session.get(user.class, id);
    if (user1 != null) {
        session.delete(user1);
        tx.commit();
        System.out.println("User deleted.");
    } else {
        System.out.println("User not found for deletion.");
        tx.rollback();
    }
}

static int createBook(Session session, String title, String author) {
    Transaction tx = session.beginTransaction();
    books book = new books();
    book.setTitle(title);
    book.setAuthor(author);
    int id = (int) session.save(book);
```

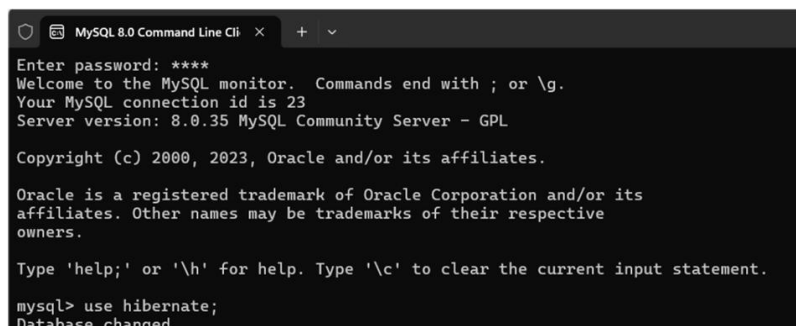
```
        tx.commit();
        System.out.println("Book created: " + title + " with ID: " + id);
        return id;
    }

    static void readBook(Session session, int id) {
        books book = session.get(books.class, id);
        if (book != null)
            System.out.println("Book: " + book.getTitle() + ", Author: " +
book.getAuthor());
        else
            System.out.println("Book not found with id: " + id);
    }

    static void updateBookAuthor(Session session, int id, String newAuthor) {
        Transaction tx = session.beginTransaction();
        books book = session.get(books.class, id);
        if (book != null) {
            book.setAuthor(newAuthor);
            session.update(book);
            tx.commit();
            System.out.println("Updated book author to: " + newAuthor);
        } else {
            System.out.println("Book not found for update.");
            tx.rollback();
        }
    }

    static void deleteBook(Session session, int id) {
        Transaction tx = session.beginTransaction();
        books book = session.get(books.class, id);
        if (book != null) {
            session.delete(book);
            tx.commit();
            System.out.println("Book deleted.");
        } else {
            System.out.println("Book not found for deletion.");
            tx.rollback();
        }
    }
}
```

Output:



```
MySQL 8.0 Command Line Cli x + v
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 23
Server version: 8.0.35 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use hibernate;
Database changed
```

Insert:

```
Problems * Javadoc Declaration Console x
App [Java Application] C:\Users\pc\Downloads\openlogic-openjdk-jre-8u442-b06-windows-x64\openlogic-openjdk-jre-8u442-b06

Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 1
Enter name: sid
Enter email: sid@gmail.com
Hibernate: insert into user (email, name) values (?, ?)
User created: sid with ID: 1

Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 2
Enter title: Pride and Prejudice
Enter author: Jane Austen
Hibernate: insert into books (author, title) values (?, ?)
Book created: Pride and Prejudice with ID: 1
```

```
mysql> show tables;
+-----+
| Tables_in_hibernate |
+-----+
| books                |
| user                 |
+-----+
2 rows in set (0.00 sec)

mysql> select * from books;
+-----+
| id | author      | title                |
+-----+
| 1  | Jane Austen | Pride and Prejudice |
+-----+
1 row in set (0.00 sec)

mysql> select * from user;
+-----+
| id | email          | name |
+-----+
| 1  | sid@gmail.com  | Sid  |
+-----+
1 row in set (0.00 sec)
```

Read:

```
Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 3
Enter user ID: 1
User: sid, Email: sid@gmail.com

Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 4
Enter book ID: 1
Book: Pride and Prejudice, Author: Jane Austen
```


Update:

```
Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 5
Enter user ID: 1
Enter new email: siddhu@gmail.com
Hibernate: select user0_.id as id1_1_0_, user0_.email as email2_1_0_, user0_.name as name3_1_0_ from user user0_ where user0_.id=?
Updated user email to: siddhu@gmail.com

Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 6
Enter book ID: 1
Enter new author: J Austen
Hibernate: select books0_.id as id1_0_0_, books0_.author as author2_0_0_, books0_.title as title3_0_0_ from books books0_ where books0_.id=?
Hibernate: update books set author=?, title=? where id=?
Updated book author to: J Austen
```

```
mysql> select * from user;
+----+-----+-----+
| id | email                | name |
+----+-----+-----+
| 1  | siddhu@gmail.com    | sid  |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from books;
+----+-----+-----+
| id | author              | title                |
+----+-----+-----+
| 1  | J Austen            | Pride and Prejudice |
+----+-----+-----+
1 row in set (0.00 sec)
```

Delete:

```
Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 7
Enter user ID to delete: 1
Hibernate: select user0_.id as id1_1_0_, user0_.email as email2_1_0_, user0_.name as name3_1_0_ from user user0_ where user0_.id=?
Hibernate: delete from user where id=?
User deleted.

Choose an option:
1. Create User
2. Create Book
3. Read User
4. Read Book
5. Update User Email
6. Update Book Author
7. Delete User
8. Delete Book
-1. Exit
Enter your choice: 8
Enter book ID to delete: 1
Hibernate: delete from books where id=?
Book deleted.
```

```
mysql> select * from books;
Empty set (0.00 sec)

mysql> select * from user;
Empty set (0.00 sec)
```

WEEK-12: Full-Stack Integration: Integrate JavaScript with Servlets: Enhance the shopping cart functionality by integrating JavaScript with servlets. Allow users to update the cart without page refresh.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Shopping Cart</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
    }
    table {
      width: 60%;
      border-collapse: collapse;
      margin-bottom: 20px;
    }
    th, td {
      padding: 10px;
      border: 1px solid #ccc;
      text-align: center;
    }
    button {
      padding: 5px 10px;
      background-color: #4CAF50;
      color: white;
      border: none;
      cursor: pointer;
    }
    #message {
      font-weight: bold;
      color: green;
    }
  </style>
</head>
<body>
<h1>Shopping Cart</h1>
<table id="cart-table">
  <tr>
    <th>Item</th>
    <th>Quantity</th>
    <th>Action</th>
  </tr>
  <tr>
    <td>Product A</td>
    <td><input type="number" id="qty-1" value="1" min="1"></td>
    <td><button onclick="updateCart(1)">Update</button></td>
  </tr>
  <tr>
    <td>Product B</td>
    <td><input type="number" id="qty-2" value="2" min="1"></td>
```

```
<td><button onclick="updateCart(2)">Update</button></td>

</tr>
</table>
<div id="message"></div>
<!-- Simulated XML for products (Normally server sends this) -->
<xml id="products-xml">
<products>
  <product id="1" name="Product A" price="100" />
  <product id="2" name="Product B" price="200" />
</products>
</xml>
<script>
  function updateCart(productId) {
    let quantity = document.getElementById('qty-' + productId).value;
    // Simulate AJAX call
    setTimeout(function() {
      // Simulate XML parsing (just for the fun)
      let xmlData = document.getElementById("products-xml").innerHTML;
      // Normally we parse XML using DOMParser, skipping that for simplicity
      // Simulate SQL update
      let simulatedSQL = `UPDATE cart SET quantity = ${quantity} WHERE product_id =
${productId}`;
      console.log("[Simulated AJAX Request]");
      console.log("SQL Executed: " + simulatedSQL);
      // Update the message dynamically
      document.getElementById('message').innerText =
        "Cart updated! Product ID: " + productId + ", Quantity: " + quantity;
    }, 500); // Simulate network delay
  }
</body>
</html>
```

OUTPUT:

Shopping Cart

Item	Quantity	Action
Product A	<input type="text" value="5"/>	<button>Update</button>
Product B	<input type="text" value="2"/>	<button>Update</button>

Cart updated! Product ID: 1, Quantity: 5

WEEK-13: Checkout Process: Implement a checkout process using servlets and JSP. Allow users to review their cart, enter shipping information, and complete the purchase.

```
CheckoutServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class CheckoutServlet extends HttpServlet {
    // Database connection details
    private static final String URL = "jdbc:mysql://localhost:3306/shop";
    private static final String USER = "root";
    private static final String PASSWORD = "yourpassword";
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // Display cart and shipping form
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Checkout</title>")
    // CSS
    out.println("<style>");
    out.println("body { font-family: Arial; background-color: #f0f0f0; padding: 20px; }");
    out.println("h1 { color: #4CAF50; }");
    out.println("form { background: white; padding: 20px; border-radius: 5px; }");
    out.println("input, textarea { width: 100%; padding: 10px; margin-top: 10px; }");
    out.println("button { margin-top: 20px; background-color: #4CAF50; color: white; padding: 10px
20px; border: none; }");
    out.println("</style>");
    // JS
    out.println("<script>");
    out.println("function validateForm() {");
    out.println("    var name = document.forms['checkoutForm']['name'].value;");
    out.println("    if (name == '') { alert('Name must be filled out'); return false; }");
    out.println("}");
    out.println("</script>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Review Cart</h1>");
    out.println("<ul>");
    out.println("<li>Item 1 - $100</li>");
    out.println("<li>Item 2 - $50</li>");
    out.println("</ul>");

    out.println("<h1>Enter Shipping Info</h1>");
    out.println("<form name='checkoutForm' action='CheckoutServlet' method='post'
onsubmit='return validateForm()'>");
    out.println("Name: <input type='text' name='name' required><br>");
    out.println("Address: <textarea name='address' required></textarea><br>");
    out.println("Phone: <input type='text' name='phone' required><br>");
    out.println("<button type='submit'>Complete Purchase</button>");
```

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

out.println("</body>");
out.println("</html>");
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String name = request.getParameter("name");
    String address = request.getParameter("address");
    String phone = request.getParameter("phone");
    try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD)) {
        String sql = "INSERT INTO orders (name, address, phone) VALUES (?, ?, ?)";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, name);
        ps.setString(2, address);
        ps.setString(3, phone);
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Confirmation</title>");
    out.println("<style>body { font-family: Arial; padding: 20px; background: #dff0d8; }</style>");
    out.println("</head><body>");
    out.println("<h1>Order Confirmed!</h1>");
    out.println("<p>Thank you, " + name + ".</p>");
    out.println("<p>Your order will be shipped to: " + address + ".</p>");
    out.println("<p>We will contact you at: " + phone + ".</p>");
</body>
</html>");
}
}
```

Output:

Review Cart

- Item 1 - \$100
- Item 2 - \$50

Enter Shipping Info

Name:

Address:

Phone:

Complete Purchase

WEEK-14: Deployment and Advanced Features: Web Deployment: Deploy the online bookstore application to a servlet container (e.g., Apache Tomcat) on a cloud platform.

1. Package Your Application: This initial phase is critical. You need to take your developed online bookstore application and package it into a format that the servlet container (like Apache Tomcat) on the cloud can understand and execute. If you're using Maven, this typically involves running the command `mvn clean package`. This process compiles your Java code, gathers all the necessary dependencies (libraries, frameworks), and structures them according to the Java web application standard. The output will be a `.war` (Web Application Archive) file, which is essentially a zipped archive containing your Servlets, JSPs, static content (HTML, CSS, JavaScript, images), and deployment descriptors (like `web.xml`). For Spring Boot applications configured for Tomcat deployment, you might need to ensure the packaging is set to `war` in your `pom.xml` or `build.gradle` file. Thoroughly verify that this `.war` file contains everything your application needs to run independently within a Tomcat environment. This packaged artifact is what you will ultimately upload to the cloud platform.

2. Choose a Cloud Platform: Selecting the right cloud platform is a foundational decision with long-term implications. Consider factors like your budget, required scalability, technical expertise, existing infrastructure, and the specific services offered by each provider. Popular choices include Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and Heroku. Each platform has its strengths and weaknesses. AWS offers a vast array of services with granular control, while GCP is known for its strengths in data analytics and machine learning. Azure integrates well with Microsoft ecosystems, and Heroku provides a simpler, developer-centric Platform-as-a-Service (PaaS) experience. Research the pricing models, regional availability (considering latency for users in Hyderabad and elsewhere), service-level agreements (SLAs), and community support for each platform before making your decision. Your choice here will dictate the subsequent steps and the specific tools and interfaces you'll interact with.

3. Select a Deployment Service: Within your chosen cloud platform, you'll have various services for deploying web applications. Platform-as-a-Service (PaaS) options like AWS Elastic Beanstalk, GCP App Engine, Azure App Service, and Heroku offer a higher level of abstraction, managing the underlying infrastructure (servers, operating systems, etc.) for you. This simplifies deployment and scaling but might offer less granular control. Infrastructure-as-a-Service (IaaS) options like AWS EC2, GCP Compute Engine, and Azure Virtual Machines provide virtual servers where you have more control over the environment, including installing and configuring Apache Tomcat yourself. Consider your comfort level with server administration, the scalability requirements of your online bookstore, and your budget when making this choice. PaaS is often quicker to get started with, while IaaS offers greater flexibility for complex configurations.

4. Create an Application/Environment: Once you've chosen your deployment service, you'll need to create a dedicated space for your online bookstore application. In a PaaS environment, this might involve creating a new "application" or "environment" through the platform's web console or command-line interface (CLI). You'll typically provide a name for your application and might need to specify the region where you want it to be hosted (consider a region geographically closer to your primary user base for better performance, though Hyderabad itself might not always be a direct option, so consider nearby regions). For IaaS, this step involves launching a virtual machine instance in your chosen region. This virtual server will be the host for your Tomcat installation and your application.

5. Configure for Java/Tomcat: This step ensures that the cloud environment is set up to run your Java web application within Apache Tomcat. In a PaaS environment, you'll usually select a Java platform version and the specific version of Tomcat you want to use during the environment creation or configuration process. The platform will then provision the necessary resources with these components pre-installed or ready to be used. For IaaS, you'll need to manually install the Java Development Kit (JDK) and then download and install Apache Tomcat on your virtual machine. You might also need to configure environment variables and adjust Tomcat's settings (like port numbers or memory allocation) to suit your application's needs and the cloud environment.

6. Deploy Your .war File: With the environment configured, you'll now deploy your packaged .war file. PaaS platforms typically offer several ways to do this, such as uploading the file through a web interface, using a CLI command, or integrating with your version control system (like Git) for automated deployments. The platform will then take your .war file and deploy it to the Tomcat instance(s) within your environment. For IaaS, you'll usually need to transfer your .war file to the webapps directory within your Tomcat installation on the virtual machine. Tomcat will automatically detect the new .war file and deploy your application. Monitor the deployment process for any errors or logs that might indicate issues.

7. Configure Environment Variables: Cloud environments often provide mechanisms to configure environment variables. This is crucial for externalizing configuration settings like database connection URLs, API keys, and other sensitive information that should not be hardcoded into your application. PaaS platforms usually have a dedicated section in their management console or CLI to set these variables. Your application can then read these variables at runtime. For IaaS, you might set environment variables at the operating system level or within Tomcat's configuration files. Using environment variables enhances security and makes it easier to manage different configurations for various deployment stages (development, testing, production).

8. Test Your Deployment: Once your application is deployed, the critical next step is thorough testing. Access your application using the public URL provided by the cloud platform. Navigate through all the core functionalities of your online bookstore – browsing books, user registration and login, adding items to the cart, placing orders, etc. Check for any errors, broken links, or unexpected behavior. Test under different scenarios and with various data inputs. Review the application logs (provided by the cloud platform or Tomcat) for any exceptions or warnings. Thorough testing in the actual deployment environment is essential to ensure that your application is functioning correctly in the cloud.

9. Set Up Monitoring: After successful deployment and initial testing, it's vital to set up monitoring for your application. Cloud platforms offer various monitoring tools that allow you to track key metrics like CPU utilization, memory usage, network traffic, response times, and error rates. Configure alerts to notify you if any critical thresholds are breached. This proactive approach helps you identify and address potential issues before they impact your users. Monitoring provides insights into your application's performance and resource consumption, enabling you to optimize it for better efficiency and reliability. Consider using logging services provided by the cloud platform to centralize and analyze your application logs.

OUTPUT:

A fully accessible website at a specific web address (URL). Users visiting this URL will see your online bookstore's homepage, be able to browse through books, and interact with its features (like searching, viewing details, and potentially adding to a cart). The site will be visually presented according to your design, and its various functionalities will be operational for anyone with internet access.