Name:Prathmesh Pattewar
Assignment No:9

Title: Design and develop a website using REACT, NodeJS and MySql.

Problem Statement: Design and develop a responsive website for an online book store using REACT, Node JS/ PHP and MySQL/ MongoDB/Oracle having 1) Home Page2) Login Page 3) Catalogue Page: 4) Registration Page: (database)

Course Objective: To learn REST API based enterprise website development using REACT, Node JS, Spring Boot with different database technologies.

Course Outcome: Write Web API/RESTful API application programming interface to communicate with Springboot as a server side technology.

Tools Required: VSCode, MySql, Firefox/GoogleChrome

Theory:

1. Introduction to Node JS

1.1 What is Node.js?

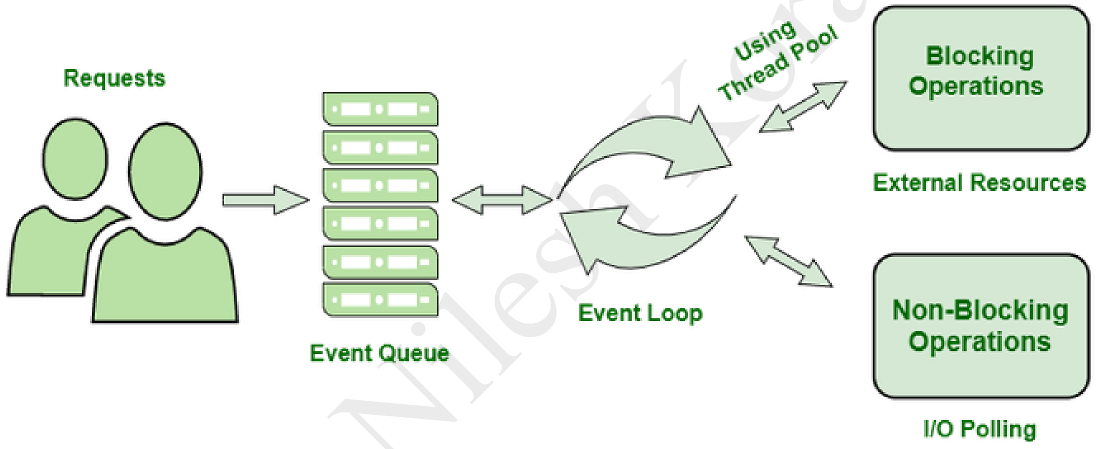
- **Node.js** is an **open-source, cross-platform, JavaScript runtime environment** that allows developers to **run JavaScript code outside of a web browser**.
- It is built on **Google Chrome's V8 JavaScript Engine**.
- Node.js is primarily used for **building fast, scalable, and server-side applications**.

1.2 Key Features of NodeJS

- **Server-Side JavaScript:** NodeJS allows JavaScript to run outside the browser, enabling backend development.
- **Asynchronous & Non-Blocking:** Uses an event-driven architecture to handle multiple requests without waiting, improving performance.
- **Single-Threaded Event Loop:** Efficiently manages concurrent tasks using a single thread, avoiding thread overhead.
- V8 Chrome Engine: High performance, Open-Source Javascript web assembly engine developed by google and rewoned for compiling JS into native machine code to execute it with exceptional speed.
- **Scalable & Lightweight:** Ideal for building microservices and handling high-traffic applications efficiently.
- **Rich NPM Ecosystem:** Access to thousands of open-source libraries through Node Package Manager (NPM) for faster development.

1.3 Components of the Node.js Architecture

- **Requests:** Depending on the actions that a user needs to perform, the requests to the server can be either blocking (complex) or non-blocking (simple).
- **Node.js Server:** The Node.js server accepts user requests, processes them, and returns results to the users.
- **Event Queue:** The main use of Event Queue is to store the incoming client requests and pass them sequentially to the Event Loop.
- **Thread Pool:** The Thread pool in a Node.js server contains the threads that are available for performing operations required to process requests.
- **Event Loop:** Event Loop receives requests from the Event Queue and sends out the responses to the clients.
- **External Resources:** In order to handle blocking client requests, external resources are used. They can be of any type (computation, storage, etc).



- Users send requests (blocking or non-blocking) to the server for performing operations.
- The requests enter the Event Queue first at the server-side.
- The Event queue passes the requests sequentially to the event loop. The event loop checks the nature of the request (blocking or non-blocking).
- Event Loop processes the non-blocking requests which do not require external resources and returns the responses to the corresponding clients
- For blocking requests, a single thread is assigned to the process for completing the task by using external resources.
- After the completion of the operation, the request is redirected to the Event Loop which delivers the response back to the client

1.4 Advantages of Node.js

- High performance and scalability.
- Suitable for real-time applications (e.g., chat apps, streaming apps).

- Large ecosystem with npm.
- Easy to learn for JavaScript developers.

1.5 Applications of Node.js

- Real-time web applications (Chat, Gaming).
- RESTful APIs and backend services.
- Streaming and data-intensive applications.
- IoT and Microservices-based systems.

1.6 Limitations of Node.js

- Not suitable for CPU-intensive tasks.
- Callback hell (complex nested callbacks).
- Single-threaded nature may limit performance for heavy computation.

2. Installation on Windows

Step 1: Download Node.js

- Visit the official Node.js website: https://nodejs.org
- You'll see two versions:
  - **LTS (Long Term Support):** Stable version (recommended)
  - **Current:** Latest features (for developers who need cutting-edge features)

Choose **LTS version** and download the **.msi installer** for Windows.

Step 2: Run the Installer

1. Open the downloaded .msi file.
2. Follow the setup wizard:
   - Accept the License Agreement.
   - Choose installation folder (default is fine).
   - Check the box "Add to PATH" (important for running Node from Command Prompt).
3. Click **Install** and wait for completion.

Step 3: Verify Installation

Open **Command Prompt (CMD)** or **PowerShell** and type:

node -v This will show your Node.js version.

npm -v This will show your npm version.

3. Node.js Core Modules List

Module Name	Description / Purpose	Commonly Used Methods / Functions
fs (File System)	Used to handle file operations like read, write, update, delete, etc.	fs.readFile(), fs.writeFile(), fs.appendFile(), fs.unlink()

Module Name	**Description / Purpose**	**Commonly Used Methods / Functions**
http	Used to create HTTP servers and clients.	http.createServer(), http.request(), http.get()
https	Used to create secure HTTP (SSL/TLS) servers.	https.createServer()
os (Operating System)	Provides information about the operating system.	os.platform(), os.type(), os.freemem(), os.totalmem()
path	Used for handling and transforming file paths.	path.join(), path.resolve(), path.basename(), path.extname()
url	Provides utilities for URL resolution and parsing.	url.parse(), url.format()
querystring	Helps parse and format URL query strings.	querystring.parse(), querystring.stringify()
events	Allows working with custom events using EventEmitter.	on(), emit(), once()
stream	Used to handle streaming data like reading/writing large files or video/audio streams.	stream.Readable, stream.Writable, stream.pipe()
buffer	Used for handling binary data directly.	Buffer.alloc(), Buffer.from(), Buffer.concat()
crypto	Provides cryptographic functionalities such as hashing, encryption, and decryption.	crypto.createHash(), crypto.createCipheriv()
net	Creates TCP servers and clients for network applications.	net.createServer(), net.connect()
dns	Used to perform DNS lookup and name resolution.	dns.lookup(), dns.resolve(), dns.reverse()
child_process	Enables running other system processes from Node.js.	exec(), spawn(), fork()
readline	Allows reading input from command line one line at a time.	readline.createInterface()
timers	Provides functions for scheduling execution.	setTimeout(), setInterval(), setImmediate()
assert	Used for testing expressions, helps in unit testing.	assert.equal(), assert.strictEqual(), assert.deepEqual()

Module Name	Description / Purpose	Commonly Used Methods / Functions
zlib	Used to compress or decompress files using Gzip/Deflate.	zlib.gzip(), zlib.unzip()
tty	Provides classes for handling text terminals (used in CLI apps).	tty.isatty()
v8	Provides access to V8 engine statistics and features.	v8.getHeapStatistics()
util	Contains utility functions for debugging and formatting.	util.format(), util.promisify(), util.inspect()
cluster	Enables load balancing by running multiple Node.js processes.	cluster.fork(), cluster.isMaster
repl	Provides a Read-Eval-Print Loop for interactive programming.	repl.start()
perf_hooks	Used to measure performance and latency in Node.js apps.	performance.now()
inspector	Provides debugging and profiling tools for Node.js.	inspector.open(), inspector.close()

4. Node Package Manager (NPM)

- **NPM** stands for **Node Package Manager**.
- It is the **default package manager for Node.js**, automatically installed when Node.js is installed.
- NPM helps **install, manage, update, and remove packages (modules)** that add functionality to Node.js applications.
- It also allows developers to **share and reuse code** efficiently.

4.1 Key Functions of NPM

Function	Description
Package Installation	Download and install libraries or modules for Node.js.
Dependency Management	Keeps track of all installed packages and their versions in package.json.
Version Control	Allows installing specific versions of packages.
Global & Local Installation	Install packages globally (available system-wide) or locally (specific project).
Publishing Packages	Developers can publish their own packages to the NPM registry.

4.2 NPM Directory Structure

- Every Node.js project can have a **node_modules/** folder containing all installed packages.
- The **package.json** file keeps track of the dependencies and project metadata.

```
myapp/
 ├── node_modules/
 ├── package.json
 └── app.js
```

4.3 Important NPM Commands

Command	**Description**	**Example**
npm -v	Check NPM version	npm -v
npm init	Initialize a new Node.js project and create package.json file	npm init
npm init -y	Quickly create package.json with default values	npm init -y
npm install <package>	Install a package locally in project	npm install express
npm install -g <package>	Install package globally	npm install -g nodemon
npm uninstall <package>	Uninstall a package	npm uninstall express
npm update <package>	Update an installed package	npm update express
npm list	List installed packages	npm list
npm outdated	Show outdated packages	npm outdated
npm run <script>	Run a script defined in package.json	npm run start
npm publish	Publish a package to npm registry	npm publish

4.4 package.json File

It is the **heart of any Node.js project** — stores information about the project and its dependencies.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "My first Node.js project",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.0.3"
  }
}
```

Key Fields:

Field	**Description**
name	Name of your project/package
version	Version number
description	Short description of project
main	Entry point file (usually app.js or index.js)
scripts	Commands that can be run using npm run
dependencies	List of required modules for the project
devDependencies	Modules required only during development
author	Developer’s name
license	License type (MIT, ISC, etc.)

Commonly Used NPM Packages

Package Name	**Purpose**
express	Web framework for building REST APIs
mongoose	MongoDB object modeling tool
nodemon	Automatically restarts app when files change
cors	Enable Cross-Origin Resource Sharing
dotenv	Load environment variables from .env file

Package Name	Purpose
body-parser	Parse incoming request bodies
jsonwebtoken	Implement authentication using JWT
bcryptjs	Encrypt passwords
multer	Handle file uploads
axios	Make HTTP requests
chalk	Add colors in terminal output

Conclusion: Hence, we have Successfully designed a responsive website for an online book store using REACT, Node JS and MySQL.

Source Code:

MySQL Setup

```
CREATE DATABASE bookstore;
USE bookstore;
CREATE TABLE users (id INT AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(255), email VARCHAR(255) UNIQUE, password VARCHAR(255));

CREATE TABLE books (id INT AUTO_INCREMENT PRIMARY KEY, title
VARCHAR(255), author VARCHAR(255), price DECIMAL(10,2));
INSERT INTO books (title, author, price) VALUES('The Alchemist', 'Paulo Coelho', 10.99),
('1984', 'George Orwell', 8.99),('Rich Dad Poor Dad', 'Robert Kiyosaki', 12.50);
```

Backend Setup

Step 1: Create Project Folders in VSCode [frontend and backend]

Step 2: Initialize Node.js project

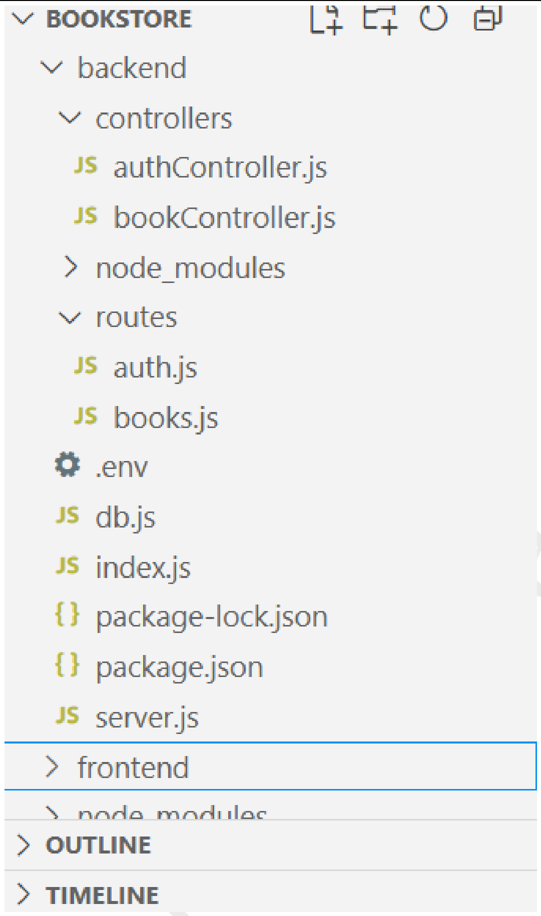
```
cd bookstore
mkdir backend
cd backend
npm init -y
```

Step 3: Install dependencies

```
npm install express mysql2 cors body-parser bcryptjs jsonwebtoken dotenv
```

Step 4: Create Controllers and routes folder

Step 5: Create db.js, .env, authController.js, bookController.js, auth.js, books.js files


```
BOOKSTORE
  backend
    controllers
      authController.js
      bookController.js
    node_modules
    routes
      auth.js
      books.js
    .env
    db.js
    index.js
    package-lock.json
    package.json
    server.js
  frontend
  node_modules
OUTLINE
TIMELINE
```

Backend Folder structure

Database Connection [db.js]

```
// Path: backend/db.js
const mysql = require('mysql2');
const dotenv = require('dotenv');
dotenv.config();

const db = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  port : 3309,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});

db.connect((err) => {
  if (err) throw err;
  console.log('MySQL Connected...');
});
module.exports = db;
```

.env

```
// Path:  backend/.env
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=root
DB_NAME=bookstore
JWT_SECRET=mySuperSecretKey123!
```

Main server file: index.js

```
// Path: backend/index.js
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');

const authRoutes = require('./routes/auth');
const bookRoutes = require('./routes/books');

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

app.use('/api/auth', authRoutes);
app.use('/api/books', bookRoutes);

const PORT = 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Auth Routes: auth.js

```
// Path:  backend/routes/auth.js
const express = require('express');
const db = require('../db');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');
```

```
dotenv.config();
const router = express.Router();

// REGISTER ROUTE
router.post('/register', (req, res) => {
  const { name, email, password } = req.body;
  const hashedPassword = bcrypt.hashSync(password, 8);

  db.query(
    'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
    [name, email, hashedPassword],
    (err, result) => {
      if (err) {
        console.error('Database error:', err);
        return res.status(500).send({ error: 'Database error' });
      }
      res.status(201).send({ message: 'User registered successfully' });
    }
  );
});

// LOGIN ROUTE
router.post('/login', (req, res) => {
  const { email, password } = req.body;

  db.query('SELECT * FROM users WHERE email = ?', [email], (err, results) => {
    if (err) return res.status(500).send({ error: 'Database error' });

    if (results.length === 0)
      return res.status(404).send({ message: 'User not found' });

    const user = results[0];
    const isPasswordValid = bcrypt.compareSync(password, user.password);

    if (!isPasswordValid)
      return res.status(401).send({ message: 'Invalid password' });

    const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, { expiresIn: '1h' });
    res.send({
      token,
      user: { id: user.id, name: user.name, email: user.email },
    });
  });
});

module.exports = router;
```

Book Routes: books.js

```
// Path:  backend/routes/books.js
const express = require('express');
const router = express.Router();
const db = require('../db');
const { getBooks } = require('../controllers/bookController');

router.get('/', (req, res) => {
  db.query('SELECT * FROM books', (err, results) => {
    if (err) return res.status(500).send({ error: 'Database error' });
    res.send(results);
  });
});
module.exports = router;
```

Book Controller: bookController.js

```
// Path:  backend/ controllers/bookController.js
const db = require('../db');

const getBooks = (req, res) => {
  db.query('SELECT * FROM books', (err, results) => {
    if (err) return res.status(500).send(err);
    res.send(results);
  });
};
module.exports = { getBooks };
```

Auth Controller: authController.js

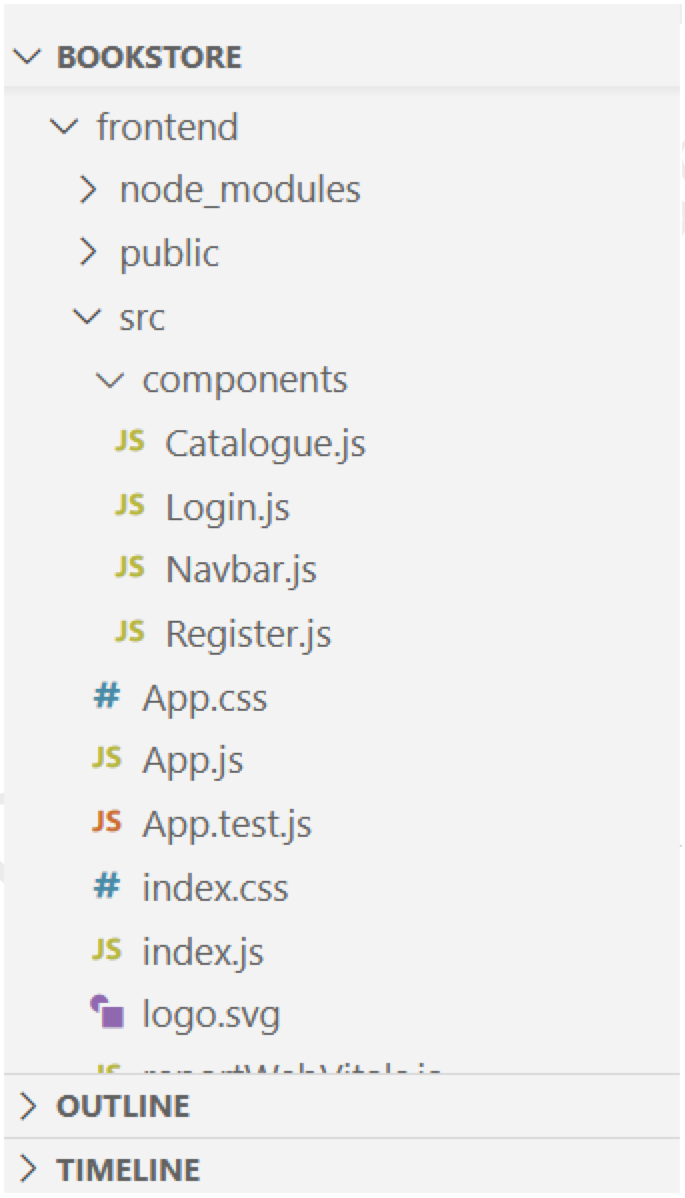
```
// Path:  backend/ controllers/ authController.js
const express = require('express');
const db = require('../db');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');

dotenv.config();
const router = express.Router();
router.post('/register', (req, res) => {
  const { name, email, password } = req.body;
  const hashedPassword = bcrypt.hashSync(password, 8);

  // Insert user into database
  db.query(
    'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
    [name, email, hashedPassword],
    (err, result) => {
      if (err) {
        console.error('Database error:', err);
        return res.status(500).send({ error: 'Database error' });
      }
      res.status(201).send({ message: 'User registered successfully' });
    }
  );
});

router.post('/login', (req, res) => {
  const { email, password } = req.body;

  db.query('SELECT * FROM users WHERE email = ?', [email], (err, results) => {
    if (err) {
      console.error('Database error:', err);
      return res.status(500).send({ error: 'Database error' });
    }

    if (results.length === 0) {
      return res.status(404).send({ message: 'User not found' });
    }

    const user = results[0];
    const isPasswordValid = bcrypt.compareSync(password, user.password);

    if (!isPasswordValid) {
      return res.status(401).send({ message: 'Invalid password' });
    }

    // Generate JWT token
    const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, { expiresIn: '1h' });

    res.send({
      token,
      user: { id: user.id, name: user.name, email: user.email },
    });
  });
});
module.exports = router;
```

Frontend Setup

Step 1: Create React App

```
cd bookstore
npx create-react-app frontend
cd frontend
npm install axios react-router-dom
```

Step 2: Create component folder and Catlog.js,Login.js,Navbar.js, Register.js file



```
BOOKSTORE
  frontend
    node_modules
    public
    src
      components
        Catalogue.js
        Login.js
        Navbar.js
        Register.js
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
OUTLINE
TIMELINE
```

Frontend Folder Structure

App.js

```
import React from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Login from "./components/Login";
import Catalogue from "./components/Catalogue";

function App() {
  return (
    <Router>
      <div>
        <h1 style={{ textAlign: "center" }}>Online Book Store</h1>
        <Routes>
          {/* Home page: show Register and Login */}
          <Route
            path="/"
            element={
              <div>
                <Login />
              </div>
            }
          />

          {/* Catalogue page */}
          <Route path="/catalogue" element={<Catalogue />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

Navbar.js

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar = () => (
  <nav>
    <Link to="/">Home</Link> |
    <Link to="/catalogue">Catalogue</Link> |
    <Link to="/login">Login</Link> |
    <Link to="/register">Register</Link>
  </nav>
);

export default Navbar;
```

```
Login.js
import React, { useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import Register from "./Register";

function Login() {
  const [showRegister, setShowRegister] = useState(false);
  const [user, setUser] = useState({ email: "", password: "" });
  const [message, setMessage] = useState("");
  const navigate = useNavigate();

  const handleChange = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post("http://localhost:5000/api/auth/login", user);
      localStorage.setItem("token", res.data.token);
      setMessage("Login successful!");
      navigate("/catalogue");
    } catch (err) {
      setMessage("Login failed: " + (err.response?.data?.message || "Server error"));
    }
  };

  const handleRegisterClick = () => {
    setShowRegister(true);
  };

  if (showRegister) {
    return <Register />;
  }

  return (
    <div style={styles.container}>
      <h2>Login</h2>
      <form onSubmit={handleSubmit} style={styles.form}>
        <input type="email" name="email" placeholder="Email" onChange={handleChange}
required />
        <input type="password" name="password" placeholder="Password"
onChange={handleChange} required />
        <button type="submit">Login</button>
        <button type="button" onClick={handleRegisterClick}>Register</button>
      </form>
      <p>{message}</p>
    </div>
  );
}

const styles = {
  container: { textAlign: "center", padding: "20px" },
  form: { display: "flex", flexDirection: "column", alignItems: "center", gap: "10px", width:
"250px", margin: "auto" }
};

export default Login;
```

Register.js

```
import React, { useState } from "react";
import axios from "axios";

function Register() {
  const [user, setUser] = useState({ name: "", email: "", password: "" });
  const [message, setMessage] = useState("");

  const handleChange = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
    const res = await axios.post("http://localhost:5000/api/auth/register", user);
      setMessage(res.data.message || "Registration successful!");
    } catch (err) {
      setMessage("Registration failed: " + (err.response?.data?.message || "Server error"));
    }
  };

  return (
    <div style={styles.container}>
      <h2>Register</h2>
      <form onSubmit={handleSubmit} style={styles.form}>
        <input type="text" name="name" placeholder="Name" onChange={handleChange}
required />
        <input type="email" name="email" placeholder="Email" onChange={handleChange}
required />
        <input type="password" name="password" placeholder="Password"
onChange={handleChange} required />
        <button type="submit">Register</button>
      </form>
      <p>{message}</p>
    </div>
  );
}

const styles = {
  container: { textAlign: "center", padding: "20px" },
  form: { display: "flex", flexDirection: "column", alignItems: "center", gap: "10px", width:
"250px", margin: "auto" }
};

export default Register;
```

Catalogue.js

```
import React, { useEffect, useState } from "react";
import axios from "axios";

function Catalogue() {
  const [books, setBooks] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Fetch all books from backend
    const fetchBooks = async () => {
      try {
        const res = await axios.get("http://localhost:5000/api/books");
        setBooks(res.data);
      } catch (err) {
        console.error("Error fetching books:", err);
      } finally {
        setLoading(false);
      }
    };

    fetchBooks();
  }, []);

  if (loading) return <p style={{ textAlign: "center" }}>Loading books...</p>;

  return (
    <div style={{ textAlign: "center" }}>
      <h2>Book Catalogue</h2>
      {books.length === 0 ? (
        <p>No books available</p>
      ) : (
        <div style={{ display: "flex", flexWrap: "wrap", justifyContent: "center", gap: "20px"
}}>
          {books.map((book) => (
            <div
              key={book.id} // assuming your books table has 'id'
              style={{
                border: "1px solid #ccc",
          padding: "10px",
          width: "200px",
          borderRadius: "5px",
         }}
        >
         <h3>{book.title}</h3>
         <p>Author: {book.author}</p>
         <p>Price: ${book.price}</p>
        </div>
      ))}
     </div>
    )}
   </div>
 );
}

export default Catalogue;
```

Execution:

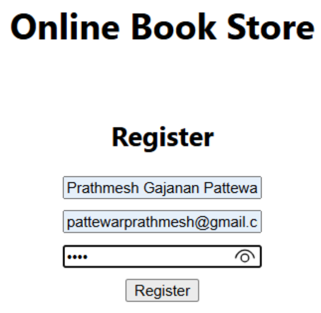
Start Backend

```
C:\VSCode\ReactJS\bookstore>cd backend
C:\VSCode\ReactJS\bookstore\backend> node server.js
```

Start FrontEnd

```
C:\VSCode\ReactJS\bookstore> cd frontend
C:\VSCode\ReactJS\bookstore\frontend> npm start
```

OutPut


Online Book Store

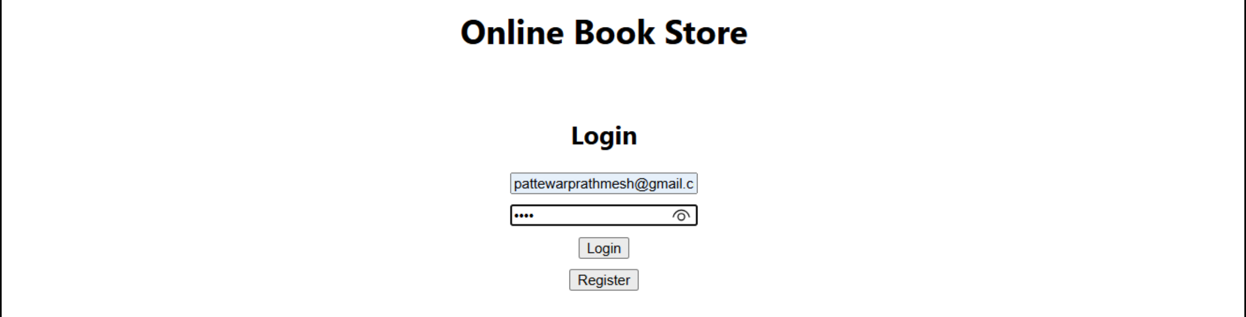
Register

Prathmesh Gajanan Pattewa

pattewarprathmesh@gmail.c

••••

Register Page



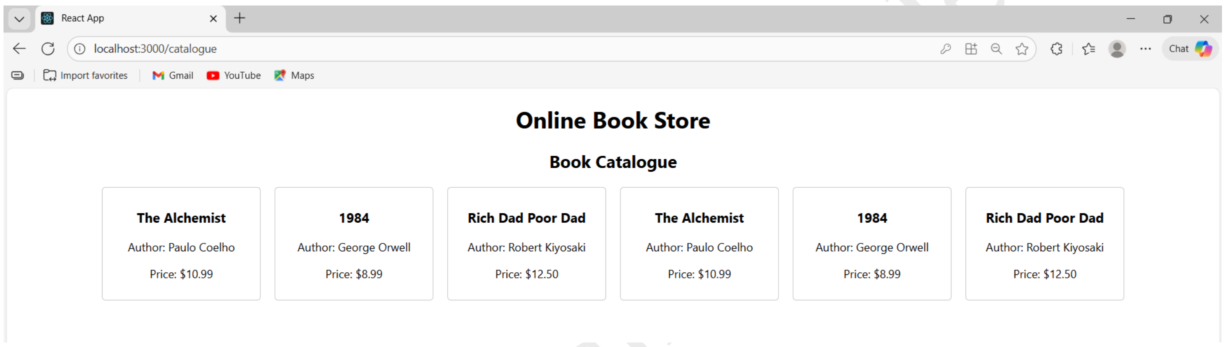
Online Book Store

Login

pattewarprathmesh@gmail.c

••••

LoginPage



Online Book Store

Book Catalogue

The Alchemist	**1984**	**Rich Dad Poor Dad**	**The Alchemist**	**1984**	**Rich Dad Poor Dad**
Author: Paulo Coelho	Author: George Orwell	Author: Robert Kiyosaki	Author: Paulo Coelho	Author: George Orwell	Author: Robert Kiyosaki
Price: $10.99	Price: $8.99	Price: $12.50	Price: $10.99	Price: $8.99	Price: $12.50

Catalogue Page