

Flightfinder: navigating your air travel options

INTRODUCTION

FlySmart is your go-to destination for convenient, fast, and reliable flight bookings. With an intuitive interface and access to a wide range of domestic and international airlines, planning your journey has never been easier.

Effortlessly search and compare flights, check real-time seat availability, read verified traveler reviews, and unlock exclusive airfare deals.

Enjoy a secure, quick booking experience and receive instant ticket confirmations. For airlines and agents, our partner dashboard offers streamlined booking management and insightful analytics to drive operational efficiency.

Key Features

- Effortless Flight Search & Filters
- Seamless Booking & Payment Process
- Personalized Travel Recommendations
- Efficient Booking Management for Airlines & Agents
- Real-Time Analytics to Boost Sales and Performance

Scenario: Raj's Last-Minute Business Trip

Raj, a busy marketing executive, is informed about an urgent business meeting in another city—scheduled for the very next day. With no time to lose, he needs to book a reliable flight quickly.

1. Effortless Flight Search

Raj opens FlySmart on his phone and selects the "Last-Minute Flights" tab. Instantly, he sees a curated list of available flights sorted by price, duration, and earliest departure. Using filters like "Direct Flights Only" and "Under ₹5000," he quickly finds a perfect option.

2. Personalized Travel Suggestions

As he browses, Raj notices a section titled "Recommended for Your Frequent Routes." Based on his past bookings, FlySmart suggests flights he often takes for business, as well as seat upgrades and airport lounge options.

3. Seamless Booking & Payment

In just a few taps, Raj selects a window seat, adds 20kg of extra baggage, and chooses UPI for payment. Thanks to FlySmart's fast, secure booking process, his flight is confirmed in under a minute.

4. Instant Ticket Confirmation

Raj immediately receives his e-ticket via email and SMS, along with boarding time, gate info, and a live flight status link. With peace of mind, he packs his bags and prepares for his meeting.

5. Airline Booking Management

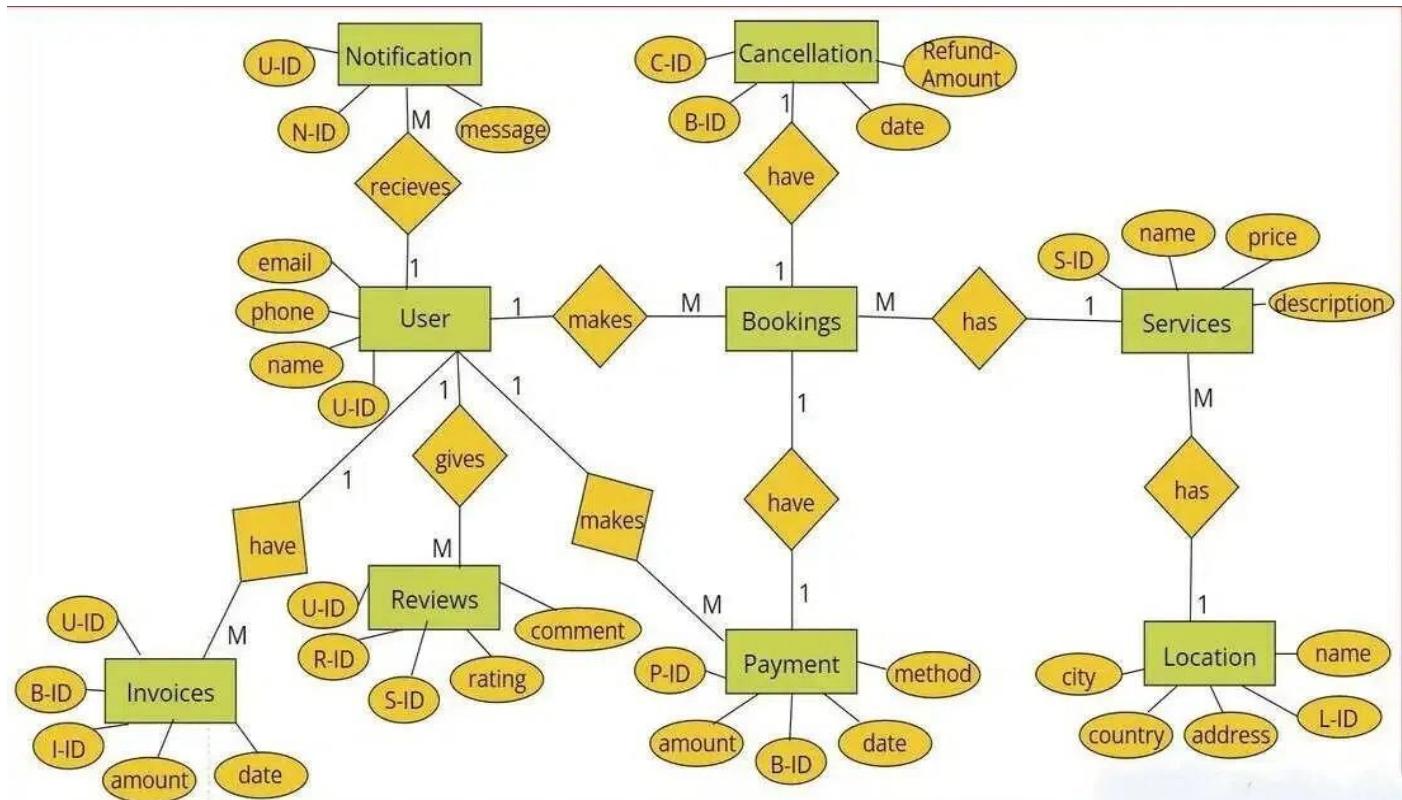
Meanwhile, the airline receives Raj's booking via the FlySmart Airline Dashboard. The system highlights the urgent booking, allowing staff to update the manifest and notify the gate crew in real time.

6. A Smooth Business Trip

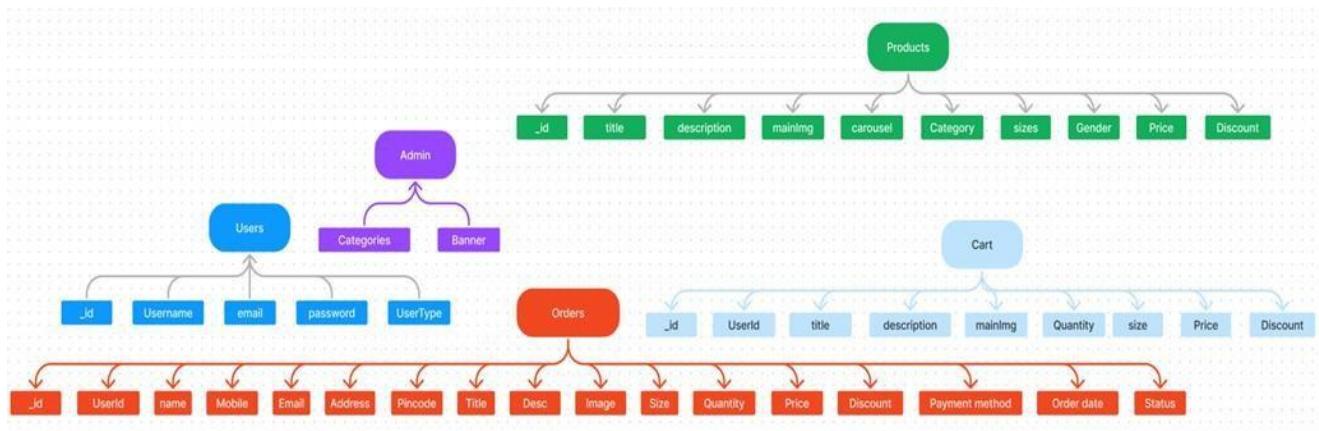
Raj boards on time, enjoys a smooth flight, and arrives ready for his presentation. The last-minute booking, once a source of stress, was handled effortlessly—thanks to FlySmart.

FlySmart makes air travel planning stress-free, intelligent, and efficient—for both passengers and providers.

ER DIAGRAM:



The Database section represents the database that stores collections for Users, Admin, booking, services and payment.



The ShopEZ ER-diagram represents the entities and relationships involved in an e-commerce system. It illustrates how users, products, cart, and orders are interconnected. Here is a breakdown of the entities and their relationships:

USER: Represents the individuals or entities who are registered in the platform.

Admin: Represents a collection with important details such as Banner image and

Categories. **Products:** Represents a collection of all the products available in the platform.

Cart: This collection stores all the products that are added to the cart by users. Here, the elements in the cart are differentiated by the user Id.

Orders: This collection stores all the orders that are made by the users in the platform.

Features:

- Comprehensive Product Catalog:** ShopEZ boasts an extensive catalog of products, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect items for your needs.
- Shop Now Button:** Each product listing features a convenient "Shop Now" button. When you find a product that aligns with your preferences, simply click on the button to initiate the purchasing process.
- Order Details Page:** Upon clicking the "Shop Now" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.
- Secure and Efficient Checkout Process:** ShopEZ guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as

possible.

5. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

In addition to these user-centric features, ShopEZ provides a robust seller dashboard, offering sellers an array of functionalities to efficiently manage their products and sales. With the seller dashboard, sellers can add and oversee multiple product listings, view order history, monitor customer activity, and access order details for all purchases.

ShopEZ is designed to elevate your online shopping experience by providing a seamless and user-friendly way to discover and purchase products. With our efficient checkout process, comprehensive product catalog, and robust seller dashboard, we ensure a convenient and enjoyable online shopping experience for both shoppers and sellers alike.

PREREQUISITES:

To develop a full-stack e-commerce app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.
• Download: <https://nodejs.org/en/download/>
• Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

<https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link: •

Link: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

To run the existing ShopEZ App project downloaded from github: Follow below steps:

Clone the repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the e-commerce app.
- Execute the following command to clone the repository:

Git clone: <https://github.com/harsha-vardhan-reddy-07/shopEZ--e-commerce-MERN>

Install Dependencies:

- Navigate into the cloned repository directory:

```
cd ShopEZ—e-commerce-App-MERN
```

- Install the required dependencies by running the following command: **npm install**

Start the Development Server:

- To start the development server, execute the following command:

```
npm run dev or npm run start
```

- The e-commerce app will be accessible at <http://localhost:3000> by default.
You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the ShopEZ app on your local machine. You can now proceed with further customization, development, and testing as needed.

USER & ADMIN FLOW:

1. User Flow:

- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform.
- Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment details.
- After ordering, they can check them in the profile section.

2. Admin Flow:

- Admins start by logging in with their credentials.

- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.,

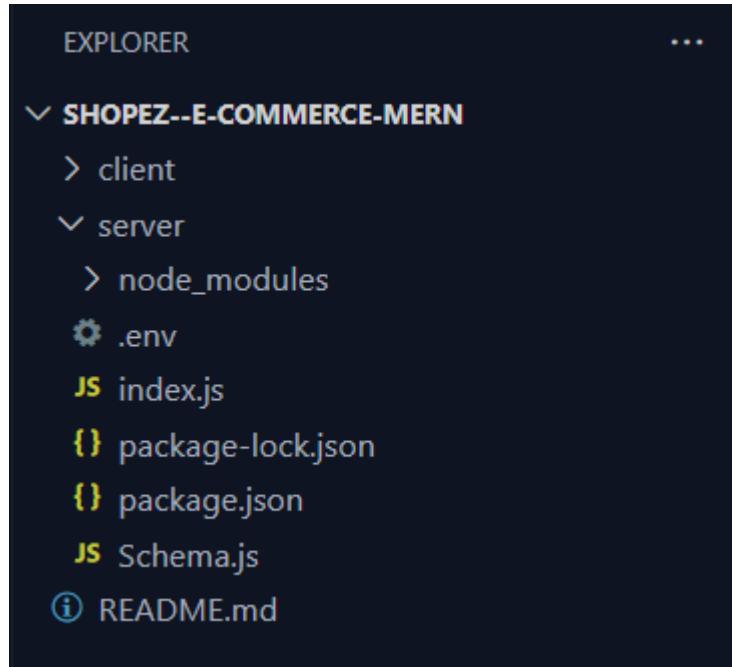
PROJECT STRUCTURE:

```

▽ SHOPEZ--E-COMMERCE-MERN
  ▽ client
    > node_modules
    > public
    ▽ src
      > components
      > context
      > images
      > pages
      > styles
      # App.css
      JS App.js
      JS App.test.js
      # index.css
      JS index.js
      logo.svg
      JS reportWebVitals.js
      JS setupTests.js
    {} package-lock.json
    {} package.json
    ⓘ README.md
    > server
    ⓘ README.md
  
```

This structure assumes a React app and follows a modular approach. Here's a brief explanation of the main directories and files:

- `src/components`: Contains components related to the application such as, `register`, `login`, `home`, etc.,
- `src/pages` has the files for all the pages in the application.



Project Flow:

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Node.js.

Reference Article: <https://www.geeksforgeeks.org/installation-of-node-js-on-windows/>

- Git.

Reference Article: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

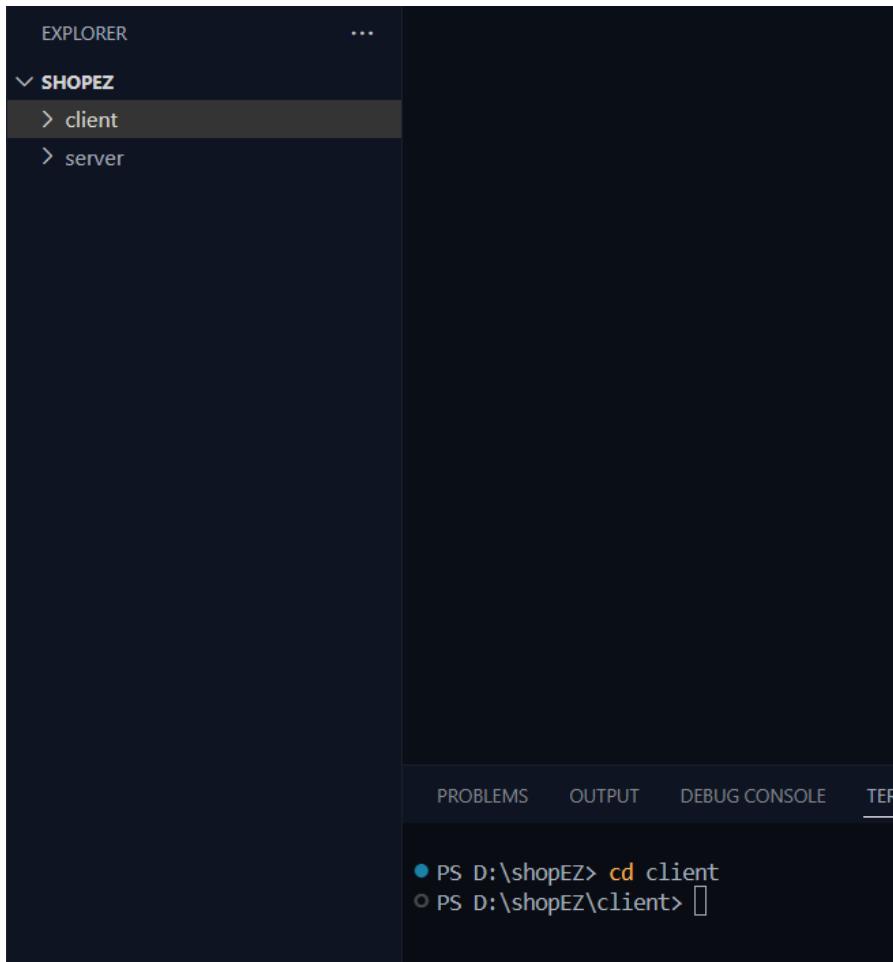
2. Create project folders and files:

- Client folders.
- Server folders

Referral Video Link:

https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QIjTpnyQ/view?usp=sharing

Referral Image:



Milestone 2: Backend Development:

1. Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API's

Reference Video:

https://drive.google.com/file/d/1-uKMIcrok_ROHyZI2vRORggrYRi02qXS/view?usp=sharing

Reference Image:

The screenshot shows the VS Code interface. The Explorer sidebar on the left shows a project structure with a 'server' folder containing 'index.js'. The 'index.js' file is open in the editor tab, displaying the following code:

```
1 import express from "express";
2
3 const app = express();
4 app.use(express.json());
5
6 app.listen(3001, () => {
7   console.log("App server is running on port 3001");
8 });
9
```

The terminal tab at the bottom shows the command line output:

- PS D:\shopEZ> cd server
- PS D:\shopEZ\server> node index.js

The message "App server is running on port 3001" is displayed in the terminal.

Now your express is successfully created.

2. Configure MongoDB:

Create database in cloud video link:-

<https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLPOh-Bu2bXhq7A3/view>

- Install Mongoose.
- Create database connection.

Reference Video of connect node with mongoDB database:

<https://drive.google.com/file/d/1cTS3 -EOAAvDctkibG5zVikrTdmoY2Aq/view?usp=sharing>

Reference Article: <https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/>

Reference Image:

```

EXPLORER      ...
JS index.js  X  .env
SHOPEZ        ...
client
server
> node_modules
.env
JS index.js
{} package-lock.json
{} package.json

server > JS index.js > ...
1 import express from "express";
2 import mongoose from "mongoose";
3 import cors from "cors";
4 import dotenv from "dotenv";
5
6 dotenv.config({ path: "./.env" });
7
8 const app = express();
9 app.use(express.json());
10 app.use(cors());
11
12 app.listen(3001, () => {
13   console.log("App server is running on port 3001");
14 });
15
16 const MongoUri = process.env.DRIVER_LINK;
17 const connectToMongo = async () => {
18   try {
19     await mongoose.connect(MongoUri);
20     console.log("Connected to your MongoDB database successfully");
21   } catch (error) {
22     console.log(error.message);
23   }
24 };
25
26 connectToMongo();
27

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\shopEZ> cd server
 ○ PS D:\shopEZ\server> node index.js
 App server is running on port 3001
 bad auth : authentication failed
 ○ PS D:\shopEZ\server> node index.js
 App server is running on port 3001
 Connected to your MongoDB database successfully

> OUTLINE > TIMELINE > NPM SCRIPTS

3. Implement API endpoints:

- Implement CRUD operations.
- Test API endpoints.

Backend:

1. Set Up Project Structure:

- Create a new directory for your project and set up a package.json file using the npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other

required packages.

Reference Video:

<https://drive.google.com/file/d/19df7NU-gQK3D06wr7ooAfJYI0wnemZoF/view?usp=sharing>

Reference Image:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure for 'SHOPEZ' with 'client' and 'server' folders, and files like 'package-lock.json' and 'package.json'. The 'package.json' file is open in the center editor, showing its contents. Below the editor is a terminal window showing npm installation logs for 'express', 'mongoose', 'body-parser', and 'dotenv' packages. The bottom status bar shows icons for outline, timeline, and other development tools.

```
1 {  
2   "name": "server",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"  
8   },  
9   "keywords": [],  
10  "author": "",  
11  "license": "ISC",  
12  "dependencies": {  
13    "bcrypt": "^5.1.1",  
14    "body-parser": "^1.20.2",  
15    "cors": "^2.8.5",  
16    "dotenv": "^16.4.5",  
17    "express": "^4.19.1",  
18    "mongoose": "^8.2.3"  
19  }  
20}  
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS D:\shopEZ\server> npm install express mongoose body-parser dotenv  
added 85 packages, and audited 86 packages in 11s  
14 packages are looking for funding  
  run `npm fund` for details  
  found 0 vulnerabilities  
● PS D:\shopEZ\server> npm i bcrypt cors  
added 61 packages, and audited 147 packages in 9s
```

2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and

cors for handling cross-origin requests.

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

Schema use-case:

1. User Schema:

- Schema: userSchema
- Model: 'User'
- The User schema represents the user data and includes fields such as username, email, and password.
- It is used to store user information for registration and authentication purposes.
 - The email field is marked as unique to ensure that each user has a unique email address

2. Product Schema:

- Schema: productSchema
- Model: 'Product'
- The Product schema represents the data of all the products in the platform.
- It is used to store information about the product details, which will later be useful for ordering .

3. Orders Schema:

- Schema: ordersSchema
- Model: 'Orders'
- The Orders schema represents the orders data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the orders made by users.
- The user Id field is a reference to the user who made the order.

4. Cart Schema:

- Schema: cartSchema
- Model: 'Cart'
- The Cart schema represents the cart data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the products added to the cart by users.
- The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

- Schema: adminSchema
- Model: 'Admin'
- The admin schema has essential data such as categories, banner.

Code Explanation:

Schemas:

Now let us define the required schemas

```
JS Schema.js X
server > JS Schema.js > [0] productSchema
1  import mongoose from "mongoose";
2
3  const userSchema = new mongoose.Schema({
4    username: {type: String},
5    password: {type: String},
6    email: {type: String},
7    usertype: {type: String}
8  });
9
10 const adminSchema = new mongoose.Schema({
11   banner: {type: String},
12   categories: {type: Array}
13 });
14
15 const productSchema = new mongoose.Schema({
16   title: {type: String},
17   description: {type: String},
18   mainImg: {type: String},
19   carousel: {type: Array},
20   sizes: {type: Array},
21   category: {type: String},
22   gender: {type: String},
23   price: {type: Number},
24   discount: {type: Number}
25 })
26
```

```

JS Schema.js X
server > JS Schema.js > ⚡ productSchema
27 const orderSchema = new mongoose.Schema({
28   userId: {type: String},
29   name: {type: String},
30   email: {type: String},
31   mobile: {type: String},
32   address: {type: String},
33   pincode: {type: String},
34   title: {type: String},
35   description: {type: String},
36   mainImg: {type: String},
37   size: {type: String},
38   quantity: {type: Number},
39   price: {type: Number},
40   discount: {type: Number},
41   paymentMethod: {type: String},
42   orderDate: {type: String},
43   deliveryDate: {type: String},
44   orderStatus: {type: String, default: 'order placed'}
45 })
46
47 const cartSchema = new mongoose.Schema({
48   userId: {type: String},
49   title: {type: String},
50   description: {type: String},
51   mainImg: {type: String},
52   size: {type: String},
53   quantity: {type: String},
54   price: {type: Number},
55   discount: {type: Number}
56 })
57
58
59 export const User = mongoose.model('users', userSchema);
60 export const Admin = mongoose.model('admin', adminSchema);
61 export const Product = mongoose.model('products', productSchema);
62 export const Orders = mongoose.model('orders', orderSchema);
63 export const Cart = mongoose.model('cart', cartSchema);
64

```

User Authentication:

- Backend

Now, here we define the functions to handle http requests from the client for authentication.

```

JS index.js X
server > JS index.js > ⚡ then() callback > ⚡ app.post('/login') callback
21
22 app.post('/register', async (req, res) => {
23   const { username, email, usertype, password } = req.body;
24   try {
25
26     const existingUser = await User.findOne({ email });
27     if (existingUser) {
28       return res.status(400).json({ message: 'User already exists' });
29     }
30
31     const hashedPassword = await bcrypt.hash(password, 10);
32     const newUser = new User({
33       username, email, usertype, password: hashedPassword
34     });
35     const userCreated = await newUser.save();
36     return res.status(201).json(userCreated);
37
38   } catch (error) {
39     console.log(error);
40     return res.status(500).json({ message: 'Server Error' });
41   }
42 });
43

```

```
JS index.js X
server > JS index.js > ⚡ then() callback > ⚡ app.get('/fetch-banner') callback
46     app.post('/login', async (req, res) => {
47       const { email, password } = req.body;
48       try {
49         const user = await User.findOne({ email });
50         if (!user) {
51           return res.status(401).json({ message: 'Invalid email or password' });
52         }
53         const isMatch = await bcrypt.compare(password, user.password);
54         if (!isMatch) {
55           return res.status(401).json({ message: 'Invalid email or password' });
56         } else{
57           return res.json(user);
58         }
59       } catch (error) {
60         console.log(error);
61         return res.status(500).json({ message: 'Server Error' });
62       }
63     });
64   
```

In the backend, we fetch all the products and then filter them on the client side.

```
JS index.js X
server > JS index.js > ⚡ then() callback > ⚡ app.get('/fetch-banner') callback
100
101    // fetch products
102
103    app.get('/fetch-products', async(req, res)=>{
104      try{
105        const products = await Product.find();
106        res.json(products);
107
108      }catch(err){
109        res.status(500).json({ message: 'Error occurred' });
110      }
111    })
112  
```

Milestone 3: Web Development:

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2. Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Video Link:

<https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing>

Reference Article Link:

https://www.w3schools.com/react/react_getstarted.asp

Reference Image:

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists a project structure for a React application named 'SHOPEZ'. The 'client' folder contains 'node_modules', 'public', and 'src' subfolders. The 'src' folder contains files like '# App.css', 'App.js', 'App.test.js', '# index.css', 'index.js', 'logo.svg', 'reportWebVitals.js', 'setupTests.js', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The 'server' folder is also listed. The 'TERMINAL' tab at the bottom shows the command 'webpack' was run, resulting in a successful compilation with the message 'webpack compiled successfully'.

File Edit Selection View Go Run Terminal Help ← → 🔍 shopEZ

EXPLORER

SHOPEZ

- client
 - node_modules
 - public
- src
 - # App.css
 - App.js
 - App.test.js
 - # index.css
 - index.js
 - logo.svg
 - reportWebVitals.js
 - setupTests.js
 - .gitignore
 - package-lock.json
 - package.json
 - README.md
- server

JS App.js M X JS index.js M

```
client > src > JS App.js > App
1 import logo from "./logo.svg";
2 import "./App.css";
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Compiled successfully!

You can now view **client** in the browser.

Compiled successfully!

You can now view **client** in the browser.

Local: http://localhost:3000
On Your Network: http://192.168.29.151:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

< master* ⌂ ⌂ 0 △ 0 ⌂ 0

Code Explanation:

· Frontend

Login:

```
JS GeneralContext.js U X
client > src > context > JS GeneralContext.js > [o] GeneralContextProvider > [o] register > [o] then() callback
  46   const login = async () =>{
  47     try{
  48       const loginInputs = {email, password}
  49       await axios.post('http://localhost:6001/login', loginInputs)
  50       .then( async (res)=>{
  51
  52         localStorage.setItem('userId', res.data._id);
  53         localStorage.setItem('userType', res.data.usertype);
  54         localStorage.setItem('username', res.data.username);
  55         localStorage.setItem('email', res.data.email);
  56         if(res.data.usertype === 'customer'){
  57           navigate('/');
  58         } else if(res.data.usertype === 'admin'){
  59           navigate('/admin');
  60         }
  61       }).catch((err) =>{
  62         alert("login failed!!!");
  63         console.log(err);
  64       });
  65     }catch(err){
  66       console.log(err);
  67     }
  68   }
  69 }
```

Register:

```
JS GeneralContext.js U X
client > src > context > JS GeneralContext.js > [o] GeneralContextProvider > [o] logout
  71   const inputs = {username, email, usertype, password};
  72
  73   const register = async () =>{
  74     try{
  75       await axios.post('http://localhost:6001/register', inputs)
  76       .then( async (res)=>{
  77         localStorage.setItem('userId', res.data._id);
  78         localStorage.setItem('userType', res.data.usertype);
  79         localStorage.setItem('username', res.data.username);
  80         localStorage.setItem('email', res.data.email);
  81
  82         if(res.data.usertype === 'customer'){
  83           navigate('/');
  84         } else if(res.data.usertype === 'admin'){
  85           navigate('/admin');
  86         }
  87       }).catch((err) =>{
  88         alert("registration failed!!!");
  89         console.log(err);
  90       });
  91     }catch(err){
  92       console.log(err);
  93     }
  94   }
  95 }
```

logout:

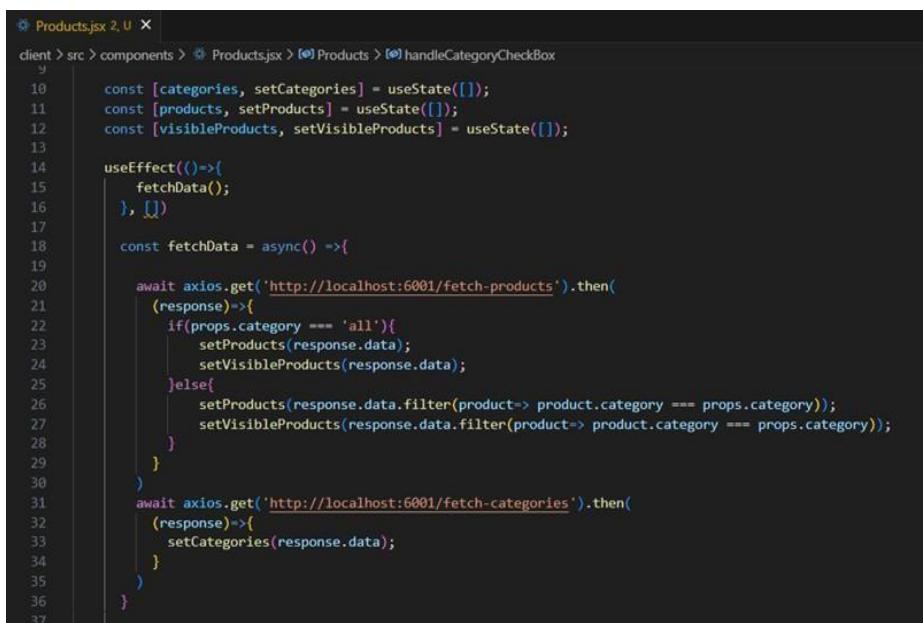


```
client > src > context > GeneralContext.jsx > GeneralContextProvider > login >
/2
73  const logout = async () =>{
74
75    localStorage.clear();
76    for (let key in localStorage) {
77      if (localStorage.hasOwnProperty(key)) {
78        localStorage.removeItem(key);
79      }
80    }
81
82    navigate('/');
83  }
84
85
```

All Products (User):

In the home page, we'll fetch all the products available in the platform along with the filters.

Fetching products:



```
client > src > components > Products.jsx > handleCategoryCheckBox >
9
10  const [categories, setCategories] = useState([]);
11  const [products, setProducts] = useState([]);
12  const [visibleProducts, setVisibleProducts] = useState([]);
13
14  useEffect(()=>{
15    fetchData();
16  }, [ ])
17
18  const fetchData = async() =>{
19
20    await axios.get('http://localhost:6001/fetch-products').then(
21      (response)=>{
22        if(props.category === 'all'){
23          setProducts(response.data);
24          setVisibleProducts(response.data);
25        }else{
26          setProducts(response.data.filter(product=> product.category === props.category));
27          setVisibleProducts(response.data.filter(product=> product.category === props.category));
28        }
29      }
30    )
31    await axios.get('http://localhost:6001/fetch-categories').then(
32      (response)=>{
33        setCategories(response.data);
34      }
35    )
36  }
37
```

Filtering products:

```
Products.jsx 2.0 X
client > src > components > Products.jsx > [e] Products > [e] useEffect() callback
  38 |   const [sortFilter, setSortFilter] = useState('popularity');
  39 |   const [categoryFilter, setCategoryFilter] = useState([]);
  40 |   const [genderFilter, setGenderFilter] = useState([]);
  41 |
  42 |   const handleCategoryCheckBox = (e) =>{
  43 |     const value = e.target.value;
  44 |     if(e.target.checked){
  45 |       setCategoryFilter([...categoryFilter, value]);
  46 |     }else{
  47 |       setCategoryFilter(categoryFilter.filter(size=> size !== value));
  48 |     }
  49 |   }
  50 |
  51 |   const handleGenderCheckBox = (e) =>{
  52 |     const value = e.target.value;
  53 |     if(e.target.checked){
  54 |       setGenderFilter([...genderFilter, value]);
  55 |     }else{
  56 |       setGenderFilter(genderFilter.filter(size=> size !== value));
  57 |     }
  58 |   }
  59 |
  60 |   const handleSortFilterChange = (e) =>{
  61 |     const value = e.target.value;
  62 |     setSortFilter(value);
  63 |     if(value === 'low-price'){
  64 |       setVisibleProducts(visibleProducts.sort((a,b)=> a.price - b.price))
  65 |     } else if (value === 'high-price'){
  66 |       setVisibleProducts(visibleProducts.sort((a,b)=> b.price - a.price))
  67 |     }else if (value === 'discount'){
  68 |       setVisibleProducts(visibleProducts.sort((a,b)=> b.discount - a.discount))
  69 |     }
  70 |   }
  71 |
  72 |   useEffect(()=>[
  73 |
  74 |     if (categoryFilter.length > 0 && genderFilter.length > 0){
  75 |       setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category) && genderFilter.includes(product.gender) ));
  76 |     }else if(categoryFilter.length === 0 && genderFilter.length > 0){
  77 |       setVisibleProducts(products.filter(product=> genderFilter.includes(product.gender) ));
  78 |     } else if(categoryFilter.length > 0 && genderFilter.length === 0){
  79 |       setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category)));
  80 |     }else{
  81 |       setVisibleProducts(products);
  82 |     }
  83 |
  84 |   ], [categoryFilter, genderFilter])
  85 |
  86 |
```

Add product to cart:

Here, we can add the product to the cart or can buy directly.

```

IndividualProduct.jsx 2, U X
client > src > pages > customer > IndividualProduct.jsx > IndividualProduct
  61 const buyNow = async() =>{
  62   await axios.post('http://localhost:6001/buy-product',{userId, name, email, mobile, address,
  63   pincode, title: productName, description: productDescription,
  64   mainImg: productMainImg, size, quantity: productQuantity, price: productPrice,
  65   discount: productDiscount, paymentMethod: paymentMethod, orderDate: new Date()}).then(
  66     (response)=>{
  67       alert('Order placed!!');
  68       navigate('/profile');
  69     }
  70   ).catch((err)=>{
  71     alert("Order failed!!");
  72   })
  73 }
  74
  75 const handleAddToCart = async() =>{
  76   await axios.post('http://localhost:6001/add-to-cart', {userId, title: productName, description: productDescription,
  77   mainImg: productMainImg, size, quantity: productQuantity, price: productPrice,
  78   discount: productDiscount}).then(
  79     (response)=>{
  80       alert("product added to cart!!!");
  81       navigate('/cart');
  82     }
  83   ).catch((err)=>{
  84     alert("Operation failed!!");
  85   })
  86 }
  87

```

- Backend: In the backend, if we want to buy, then with the address and payment method, we process buying. If we need to add the product to the cart, then we add the product details along with the user Id to the cart collection.

Buy product:

```

index.js X
server > index.js > then() callback > app.post('/add-to-cart') callback
  234 // buy product
  235
  236 app.post('/buy-product', async(req, res)=>{
  237   const {userId, name, email, mobile, address, pincode, title,
  238   description, mainImg, size, quantity, price,
  239   discount, paymentMethod, orderDate} = req.body;
  240   try{
  241     const newOrder = new Orders({userId, name, email, mobile, address,
  242     pincode, title, description, mainImg, size, quantity, price,
  243     discount, paymentMethod, orderDate})
  244     await newOrder.save();
  245     res.json({message: 'order placed'});
  246   }catch(err){
  247     res.status(500).json({message: "Error occurred"});
  248   }
  249 }
  250

```

Add product to cart:

```
JS index.js X
server > JS index.js > ⓘ then() callback > ⓘ app.put('/increase-cart-quantity') callback
301 // add cart item
302
303 app.post('/add-to-cart', async(req, res)=>{
304
305     const {userId, title, description, mainImg, size, quantity, price, discount} = req.body
306     try{
307         const item = new Cart({userId, title, description, mainImg, size, quantity, price, discount});
308         await item.save();
309         res.json({message: 'Added to cart'});
310     }catch(err){
311         res.status(500).json({message: "Error occurred"});
312     }
313 }
314
```

Order products:

Now, from the cart, let's place the order

- Frontend

```
JS Cart.jsx 3, U X
client > src > pages > customer > ⓘ Cart.jsx > [o] Cart
83 const [name, setName] = useState('');
84 const [mobile, setMobile] = useState('');
85 const [email, setEmail] = useState('');
86 const [address, setAddress] = useState('');
87 const [pincode, setPincode] = useState('');
88 const [paymentMethod, setPaymentMethod] = useState('');
89
90 const userId = localStorage.getItem('userId');
91 const placeOrder = async() =>{
92     if(cartItems.length > 0){
93         await axios.post('http://localhost:6001/place-cart-order', {userId, name, mobile,
94                                         email, address, pincode, paymentMethod, orderDate: new Date()}).then(
95             (response)=>{
96                 alert('Order placed!!');
97                 setName('');
98                 setMobile('');
99                 setEmail('');
100                setAddress('');
101                setPincode('');
102                setPaymentMethod('');
103                navigate('/profile');
104            }
105        )
106    }
107 }
```

- Backend

In the backend, on receiving the request from the client, we then place the order for the products in the cart with the specific user Id.

```

JS index.js X
server > JS index.js > ⚡ then() callback
359
360     // Order from cart
361
362     app.post('/place-cart-order', async(req, res)=>{
363         const {userId, name, mobile, email, address, pincode, paymentMethod, orderDate} = req.body;
364         try{
365             const cartItems = await Cart.find({userId});
366             cartItems.map(async (item)=>{
367
368                 const newOrder = new Orders({userId, name, email, mobile, address,
369                     pincode, title: item.title, description: item.description,
370                     mainImg: item.mainImg, size:item.size, quantity: item.quantity,
371                     price: item.price, discount: item.discount, paymentMethod, orderDate})
372                 await newOrder.save();
373                 await Cart.deleteOne({_id: item._id})
374             })
375             res.json({message: 'Order placed'});
376         }catch(err){
377             res.status(500).json({message: "Error occurred"});
378         }
379     })
380

```

Add new product:

Here, in the admin dashboard, we will add a new product.

o Frontend:

```

@ NewProduct.jsx U ✘
client > src > pages > admin > @ NewProduct.jsx > @ NewProduct
47
48     const handleNewProduct = async() =>{
49         await axios.post('http://localhost:6001/add-new-product', {productName, productDescription, productMainImg,
50                         productCarousel: [productCarouselImg1, productCarouselImg2, productCarouselImg3], productSizes,
51                         productGender, productCategory, productNewCategory, productPrice, productDiscount}).then(
52             (response)=>{
53                 alert("product added");
54                 setProductName('');
55                 setProductDescription('');
56                 setProductMainImg('');
57                 setProductCarouselImg1('');
58                 setProductCarouselImg2('');
59                 setProductCarouselImg3('');
60                 setProductSizes([]);
61                 setProductGender('');
62                 setProductCategory('');
63                 setProductNewCategory('');
64                 setProductPrice(0);
65                 setProductDiscount(0);
66
67                 navigate('/all-products');
68             }
69         )
70     }

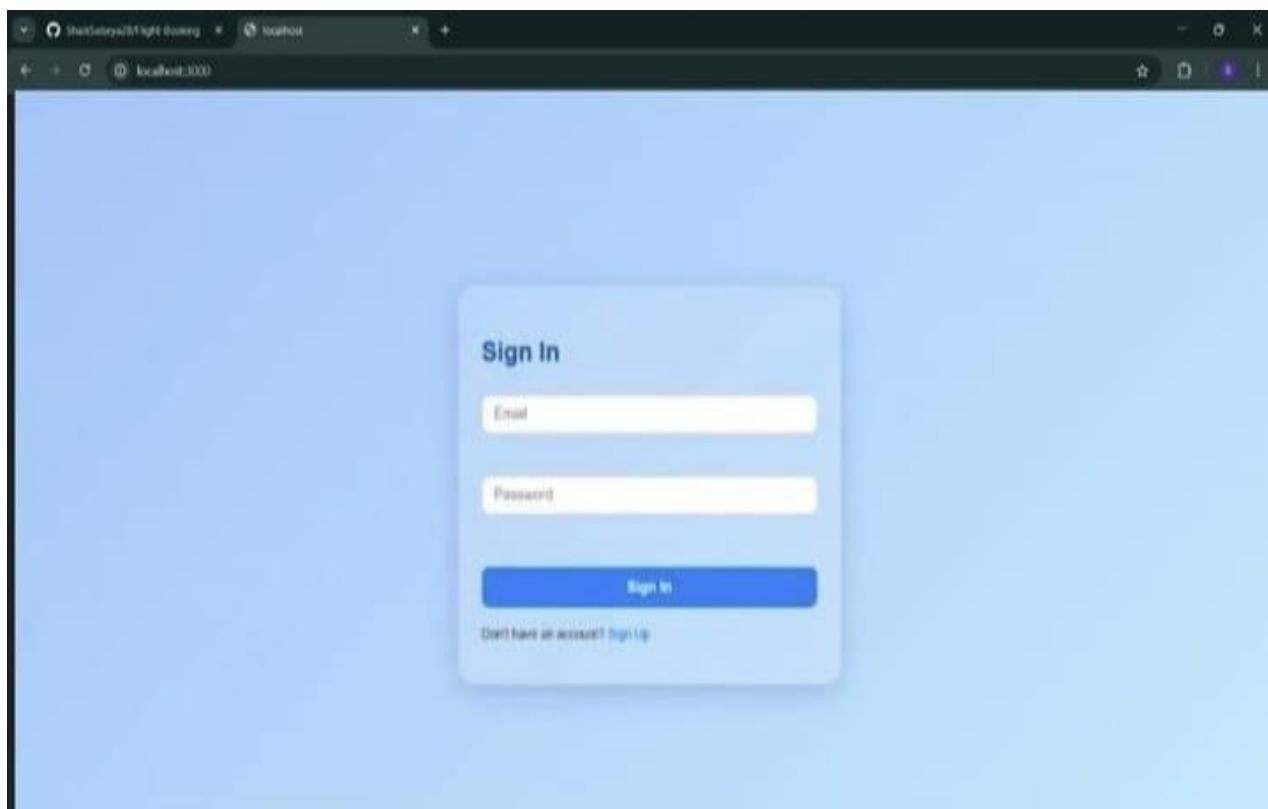
```

o Backend:

```
JS index.js x
server > JS index.js > then0 callback > app.put('/update-product/:id') callback
143
144     // Add new product
145
146     app.post('/add-new-product', async(req, res)=>{
147         const {productName, productDescription, productMainImg, productCarousel,
148                 productSizes, productGender, productCategory, productNewCategory,
149                 productPrice, productDiscount} = req.body;
150
151     try{
152         if(productCategory === 'new category'){
153             const admin = await Admin.findOne();
154             admin.categories.push(productNewCategory);
155             await admin.save();
156             const newProduct = new Product({title: productName, description: productDescription,
157                     mainImg: productMainImg, carousel: productCarousel, category: productNewCategory,
158                     sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
159             await newProduct.save();
160         } else{
161             const newProduct = new Product({title: productName, description: productDescription,
162                     mainImg: productMainImg, carousel: productCarousel, category: productCategory,
163                     sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
164             await newProduct.save();
165         }
166         res.json({message: "product added!!"});
167     }catch(err){
168         res.status(500).json({message: "Error occurred"});
169     }
170 })
```

Demo UI images:

- #### · Login page:



Search flights:

The screenshot shows a flight search results page titled "Search Flights". At the top, there are input fields for "From" (Mumbai), "To" (Hyderabad), and a date range ("2025-07-10" to "2025-07-11"). A green "Search" button is to the right. Below the search bar, there are eight flight cards arranged in two rows of four. Each card displays a route, date, time, airline, and price, along with a "Book Now" button.

From	To	Date	Time	Airline	Price
Mumbai	Hyderabad	2025-07-10	12:30 AM	Air India	₹10500
Hyderabad	Delhi	2025-07-10	10:30 AM	Air India Indigo	₹3500
Delhi	Hyderabad	2025-07-11	8:30 AM	Spirit Jet	₹4500
Mumbai	Bangalore	2025-07-10	5:30 AM	Air India	₹9500
Bangalore	Vizag	2025-07-11	12:30 AM	Air India	₹12500
Delhi	Goa	2025-07-11	9:30 AM	Air India	₹11500
Pune	Goa	2025-07-11	6:30 AM	Spirit Jet	₹10500
Mumbai	Delhi	2025-07-12	5:00 AM	Air India express	₹5500
Bangalore	Mumbai	2025-07-12	10:00 AM	Air India express	₹5500
Pune	Mumbai	2025-07-13	15:00	Air India express	₹7500
Dubai	Hyderabad	2025-07-13	7:00 PM	Alliance Air	₹9500

View bookings:

The screenshot shows a "Your Bookings" page. The title "Your Bookings" is centered at the top. Below it, there are two separate sections, each representing a flight booking from Air India. Each section contains details like route, date, time, seat, passenger information, and booking timestamp.

Flight Details	Passenger	Booking Date
Route: Mumbai → Hyderabad Date: 2025-07-10 Time: 12:30 AM Seat: 82	goutham Email: 1070@gmail.com Phone: 9899999999	Booked At: 25/06/2025, 12:40 AM
Route: Mumbai → Hyderabad Date: 2025-07-11 Time: 12:30 AM Seat: 54	pranav Email: 1070@gmail.com Phone: 9899999999	Booked At: 25/06/2025, 11:24 AM

For any further doubts or help, please consider the GitHub repo

<https://github.com/ShaikSabeya28/Flight-Booking>

The demo of the app is available at:

<https://1drv.ms/v/c/8d6c4ff8df2a482e/ES-RUw5RfjZJrvJsHicksIEBEYUe7WB7u4OfJ5oVHuxpGw?e=ZzSSI9>