

HOUSE RENT WEB APPLICATION USING MERN

SMART INTERNZ (NAAN MUDHALVAN)

PROJECT REPORT

Submitted by

- | | |
|----------------------------|---------------------|
| 1. DERIN SHYLO D S | 211121104011 |
| 2. KABILAN M | 211121104026 |
| 3. JEFFREY SAMSON R | 211121104025 |
| 4. ABISHEK S | 211121104003 |
| 5. SRITHAR | 2221110036 |

Project report submitted in fulfilment for the requirement of the degree

Of

**BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND
ENGINEERING**



**MADHA ENGINEERING COLLEGE
KUNDRATHUR CHENNAI 600 069**

NOV 2024

ABSTRACT

This project focuses on developing a user-friendly House Rent Application using the MERN (MongoDB, Express.js, React.js, and Node.js) stack. The application includes key features such as property listings with detailed descriptions, user authentication for secure account management, real-time search and filter functionalities, and an intuitive interface for uploading and managing property details. Landlords can easily list their properties by providing essential details like location, rent amount, and amenities, while tenants can browse and filter available properties based on their preferences. Backend services are implemented using Node.js and Express.js, ensuring scalable and efficient server-side operations.

The proposed application has significant potential for further enhancements, including advanced features like payment integration, landlord-tenant communication tools, and AI-powered recommendations for properties. This project demonstrates the practical application of the MERN stack in solving real-world problems and contributing to the digital transformation of the real estate industry.

Table of Contents

S. No	Topics	Page No
1	Introduction	1
2	Purposes and Features	2
2.1	Key Features	2
3	Technical Architecture	5
4	Prerequisites	7
5	User and Admin Flow	9
6	Folder Structure	11
7	Program	13
8	API Documentation	35
9	Authentication Mechanism	38
10	User Interface Screenshots	39
11	Testing Strategy	46
12	Known Issues	47
13	Future Enhancement	51
14	Conclusion	52
15	Reference	53

Fig	Fig Name	Page No
1.1	Functioning of MERN	5
2.2	Frontend	13
3.1	Index code	13
3.2	Adminhome code	13
3.3	AllBookings code	15
3.4	AllProperty code	17
3.5	AllUsers Code	18
3.6	Home code	20
3.7	Login code	22
3.8	Register Code	23
3.9 a)	Admin Controller code	24
3.9 b)	OwnerController Code	25
3.9 c)	UserController Code	27
3.10a)	AdminRoutes code	27
3.10 b)	OwnerRoutes Code	28
3.10 c)	UserRoutes code	29
3.11 a)	BookingModel Code	30

3.11 b)	PropertyModel code	31
3.11 c)	UserModel code	32
3.12	Connect code	32
3.13	Index code	33
3.14	Package code	33
4.1	Home Page	40
4.2	User Login Page	41
4.3	User Signup Page	42
4.4	Owner Login Page	43
4.5	Register as Owner Page	44
4.6	Property For Rent In app Page	44

1. INTRODUCTION

A **house rent application** is a modern digital platform designed to streamline and enhance the process of renting residential properties. It bridges the gap between landlords and tenants, offering an efficient and user-friendly solution to traditional renting challenges. By integrating technology into the renting process, these applications simplify property discovery, enhance communication, and make transactions more transparent

The application transforms the conventional property renting process, which often involves intermediaries, physical visits, and extensive paperwork, into a streamlined digital experience. Landlords can easily showcase their properties, while tenants can explore options from the comfort of their homes. This virtual approach saves time and effort, providing an accessible and convenient solution for individuals in different locations or with busy schedules.

In addition to addressing logistical challenges, a house rent app fosters trust and transparency in the rental ecosystem. By enabling clear communication and providing comprehensive property details upfront, it eliminates potential misunderstandings or disputes. This digital environment ensures that both landlords and tenants have all the necessary information to make informed decisions, thereby reducing the likelihood of conflicts or dissatisfaction.

It allows landlords to list properties with details like location, price, and amenities, while tenants can search, filter, and book rentals easily. Features such as secure payment gateways, chat systems, and map integration enhance the user experience. The app also supports digital rental agreements, making the process more efficient and transparent. By saving time and offering convenience, house rent apps are transforming the traditional rental market.

Overall, a house rent application redefines the renting process, offering convenience, efficiency, and reliability. It not only modernizes the way properties are rented but also creates a more structured and accountable system for both parties. As the rental market grows increasingly competitive, such applications play a pivotal role in ensuring a smooth and satisfactory experience for everyone involved.

2. PURPOSE AND FEATURES

The primary purpose of a **house rent application** is to simplify and digitize the process of renting residential properties. It aims to create a platform where landlords and tenants can connect easily, eliminating inefficiencies associated with traditional renting methods. The app facilitates property discovery, enables clear communication, and ensures secure transactions, benefiting both parties involved. Additionally, it aims to promote transparency and trust by providing detailed information about properties, rental terms, and user reviews, while saving time and effort for users.

2.1 KEY FEATURES:

1. User Registration and Profiles:

- Separate accounts for landlords and tenants.
- Profile management with detailed information about properties or rental preferences.

2. Property Listings:

- Landlords can list properties with descriptions, photos, rent details, and amenities.
- Tenants can browse properties with comprehensive details.

3. Search and Filters:

- Advanced search options to find properties based on location, budget, size, type, and additional features.
- Filters for pet-friendly properties, furnished options, and proximity to specific landmark

4. Interactive Maps:

- Integrated maps to display property locations and nearby facilities like schools, hospitals, and public transport.

5. In-App Communication:

- Secure chat or messaging features for landlords and tenants to discuss rental terms or schedule visits.

6. Rent Payment Integration:

- Secure payment gateways for tenants to pay rent, security deposits, or application fees directly through the app.

7. Booking and Scheduling:

- Features for scheduling property viewings or virtual tours.
- Notifications for confirmed appointments or changes.

8. Document Management:

- Upload and share essential documents such as lease agreements, identity proofs, and property certifications.

9. Review and Rating System:

- Tenants can rate landlords and properties, and landlords can review tenants to ensure accountability and trust.

10. Notifications and Alerts:

- Automated alerts for new property listings, rent due dates, or updates on tenant inquiries.

11. Favorites and Wishlist:

- Tenants can save preferred properties to review later.

12. Property Analytics (for Landlords):

- Tenants can rate landlords and properties, and landlords can review tenants to ensure accountability and trust.
- Insights on property performance, number of views, and tenant inquiries.

13. Multi-Language Support:

- Options for users to interact with the app in different languages to enhance accessibility.

14. Customer Support:

- Dedicated in-app support or chatbot assistance for quick issue resolution.

3. TECHNICAL ARCHITECTURE

Functioning of MERN

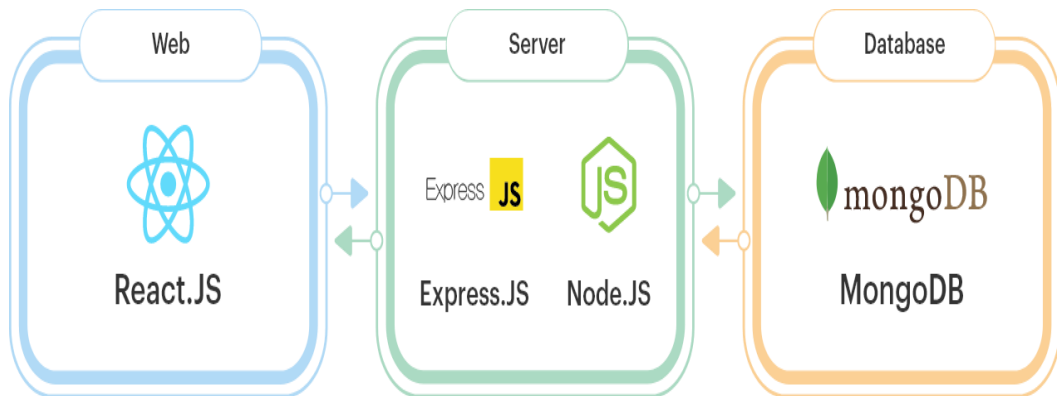


Fig 1.1

1. Frontend (React.js):

- React is used to build the user interface (UI) where users (landlords and tenants) interact with the app.
- Components like property listings, search filters, and payment pages are created here.
- The frontend communicates with the backend using APIs (RESTful or GraphQL).

2. Backend (Node.js with Express.js):

- Node.js and Express.js handle the app's logic and manage user requests.
- The backend processes API requests (e.g., property searches, user registration, payment) and communicates with the database.
- It also manages user authentication and payment transactions.

3. Database (MongoDB):

- MongoDB stores app data like user profiles, property listings, transactions, and reviews.
- Mongoose is used to manage data models and perform CRUD operations on the database.

4. APIs (Express.js):

- RESTful APIs are created using Express.js to handle requests like fetching properties, adding listings, and processing payments.

5. Security:

- JWT (JSON Web Tokens) ensures secure user authentication.
- Data is encrypted and secure payment gateways (like Stripe) are integrated for rent transactions.

Workflow:

- The tenant searches for properties through the React frontend, which makes requests to the Node.js backend.
- The backend fetches data from MongoDB and returns it to the frontend.
- Tenants can make payments, which are processed through APIs and integrated payment gateways.
- The app uses JWT for authentication, ensuring secure access.

4. PREREQUISITES

To develop a full-stack house rent 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/package-manager>

Installation Instruction: <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 Instruction: <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://docs.github.com/en/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop>

Development Environment:

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

To Connect the Database with NodeJS go through the below provided link:

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

5. ADMIN AND USER FLOW

USER FLOW

1. User Registration/Login

- Sign up with email, phone number, or social accounts.
- Login with credentials for returning users.

2. Search for Properties

- Browse or use filters like location, price range, property type, and amenities.
- View property details, photos, and reviews.

3. Property Interaction

- Schedule a visit or book the property online.
- Communicate with the landlord via in-app chat or call.

4. Payment

- Pay rent, deposit, or service fees securely through the app.
- Receive payment confirmations and transaction history.

5. Feedback and Support

- Leave reviews for properties and landlords.
- Contact customer support for queries or complaints.

ADMIN FLOW

1. Admin Dashboard

- Monitor user activities, property listings, and app performance.

- Manage user accounts (landlords, tenants, and agents).

2. Property Moderation

- Approve or reject property listings based on app guidelines.
- Ensure property details and images are accurate and appropriate.

3. Payment Management

- Track transactions, commission, and payouts to landlords.
- Resolve payment disputes or errors.

4. User Support

- Respond to complaints or queries from users.
- Take action against rule violations, like fraudulent listings or improper behavior.

5. Analytics and Reports

- Generate reports on app usage, revenue, and user engagement.
- Use analytics to improve the app and implement new features.

The user flow of a house rent app involves registering or logging in, searching for properties using filters, viewing property details, scheduling visits or booking rentals, making secure payments, and providing reviews or seeking support. The admin flow includes managing user accounts and property listings, approving or rejecting listings, handling payments and disputes, addressing user complaints, and analyzing app performance to implement improvements. Both flows ensure a seamless and efficient rental experience for users and administrators.

6. FOLDER STRUCTURE

6.1 BACKEND

- Config
- Connect.js
- Controller.js
 1. Admin controller.js
 2. OwnerController.js
 3. UserController.js
- Middleware.js
- Routes
 1. adminRoutes
 2. OwnerRoutes
 3. UserRoutes
- Schemas
 1. BookingModel
 2. PropertyModel
 3. UserModel
- Index
- Package-lock.json
- Package.json

6.2 FRONTEND

- Public
- Src
- .gitignore
- Package - lock.json
- Package.json

7. RUNNING THE APPLICATION

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import "./App.css";
import Home from "../modules/common/Home";
import Login from "../modules/common/Login";
import Register from "../modules/common/Register";
import ForgotPassword from "../modules/common/ForgotPassword";
import { createContext, useEffect, useState } from "react";
import AdminHome from "../modules/admin/AdminHome";
import OwnerHome from "../modules/user/Owner/OwnerHome";
import RenterHome from "../modules/user/renter/RenterHome";

export const UserContext = createContext();

function App() {
  const date = new Date().getFullYear();
  const [userData, setUserData] = useState();
  const [userLoggedIn, setUserLoggedIn] = useState(false)
  const getData = async () => {
    try {
      const user = await JSON.parse(localStorage.getItem("user"));
      if (user && user !== undefined) {
        setUserData(user);
        setUserLoggedIn(true)
      }
    } catch (error) {
      console.log(error);
    }
  };

  useEffect(() => {
    getData();
  }, []);

  // const userLoggedIn = !!localStorage.getItem("user");
  return (
    <UserContext.Provider value={{ userData, userLoggedIn }}>
      <div className="App">
        <Router>
          <div className="content">
            <Routes>
              <Route path="/" element={ <Home /> } />
              <Route path="/login" element={ <Login /> } />
              <Route path="/register" element={ <Register /> } />
              <Route path="/forgotpassword" element={ <ForgotPassword /> } />
            {userLoggedIn ? (
              <

```

8. PROGRAM OF FRONTEND PAGES

8.1 Index.Html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

8.2 AdminHome.jsx

```
function CustomTabPanel(props) {
  const { children, value, index, ...other } = props;

  return (
    <div
      role="tabpanel"
      hidden={value !== index}
      id={`simple-tabpanel-${index}`}
      aria-labelledby={`simple-tab-${index}`}
      {...other}
    >
      {value === index && (
        <Box sx={{ p: 3 }}>
          <Typography>{children}</Typography>
        </Box>
      )}
    </div>
  );
}
```

```
CustomTabPanel.propTypes = {
  children: PropTypes.node,
```

```

index: PropTypes.number.isRequired,
value: PropTypes.number.isRequired,
};

function allProps(index) {
  return {
    id: `simple-tab-${index}`,
    'aria-controls': `simple-tabpanel-${index}`,
  };
}

const AdminHome = () => {
  const user = useContext(UserContext)
  const [value, setValue] = useState(0);

  const handleChange = (event, newValue) => {
    setValue(newValue);
  };

  const handleLogOut = () => {
    localStorage.removeItem('token')
    localStorage.removeItem('user')
  }

  if (!user) {
    return null;;
  }

  return (
    <div>
      <Navbar expand="lg" className="bg-body-tertiary">
        <Container fluid>
          <Navbar.Brand><h2>RentEase</h2></Navbar.Brand>
          <Navbar.Toggle aria-controls="navbarScroll" />
          <Navbar.Collapse id="navbarScroll">
            <Nav
              className="me-auto my-2 my-lg-0"
              style={{ maxHeight: '100px' }}
              navbarScroll
            >
            </Nav>
            <Nav>
              <h5 className="mx-3">Hi {user.userData.name}</h5>
              <Link onClick={handleLogOut} to={"/"}>Log Out</Link>
            </Nav>

          </Navbar.Collapse>
        </Container>
      </Navbar>
    </div>
  );
}

```

```

<Box sx={{ width: '100%' }}>
  <Box sx={{ borderBottom: 1, borderColor: 'divider' }}>
    <Tabs value={value} onChange={handleChange} aria-label="basic tabs example">
      <Tab label="All Users" {...allYProps(0)} />
      <Tab label="All Properties" {...allYProps(1)} />
      <Tab label="All Bookings" {...allYProps(2)} />
    </Tabs>
  </Box>
  <CustomTabPanel value={value} index={0}>
    <AllUsers />
  </CustomTabPanel>
  <CustomTabPanel value={value} index={1}>
    <AllProperty />
  </CustomTabPanel>
  <CustomTabPanel value={value} index={2}>
    <AllBookings />
  </CustomTabPanel>
</Box>
</div>
)
}

export default AdminHome

```

8.3 AllBookings.jsx

```

const AllBookings = () => {
  const [allBookings, setAllBookings] = useState([]);

  const getAllBooking = async () => {
    try {
      const response = await axios.get('http://localhost:8001/api/admin/getallbookings', {
        headers: { 'Authorization': `Bearer ${localStorage.getItem("token")}` }
      });

      if (response.data.success) {
        setAllBookings(response.data.data);
      } else {
        message.error(response.data.message);
      }
    } catch (error) {
      console.log(error);
    }
  };
};

```

```

useEffect(() => {
  getAllBooking();
}, []);

return (
  <div>
    <TableContainer component={Paper}>
      <Table sx={{ minWidth: 650 }} aria-label="simple table">
        <TableHead>
          <TableRow>
            <TableCell>Booking ID</TableCell>
            <TableCell align="center">Owner ID</TableCell>
            <TableCell align="center">Property ID</TableCell>
            <TableCell align="center">Tenent ID</TableCell>
            <TableCell align="center">Tenent Name</TableCell>
            <TableCell align="center">Tenent Contact</TableCell>
            <TableCell align="center">Booking Status</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {allBookings.map((booking) => (
            <TableRow
              key={booking._id}
              sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
            >
              <TableCell component="th" scope="row">
                {booking._id}
              </TableCell>
              <TableCell align="center">{booking.ownerID}</TableCell>
              <TableCell align="center">{booking.propertyId}</TableCell>
              <TableCell align="center">{booking.userID}</TableCell>
              <TableCell align="center">{booking.userName}</TableCell>
              <TableCell align="center">{booking.phone}</TableCell>
              <TableCell align="center">{booking.bookingStatus}</TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  </div>
);
};

export default AllBookings;

```

8.4 AllProperty.jsx

```
const AllProperty = () => {
  const [allProperties, setAllProperties] = useState([]);

  const getAllProperty = async () => {
    try {
      const response = await axios.get('http://localhost:8001/api/admin/getallproperties', {
        headers: { 'Authorization': `Bearer ${localStorage.getItem("token")}` }
      });

      if (response.data.success) {
        setAllProperties(response.data.data);
      } else {
        message.error(response.data.message);
      }
    } catch (error) {
      console.log(error);
    }
  };

  useEffect(() => {
    getAllProperty();
  }, []);

  return (
    <div>
      <TableContainer component={Paper}>
        <Table sx={{ minWidth: 650 }} aria-label="simple table">
          <TableHead>
            <TableRow>
              <TableCell>Property ID</TableCell>
              <TableCell align="center">Owner ID</TableCell>
              <TableCell align="center">Property Type</TableCell>
              <TableCell align="center">Property Ad Type</TableCell>
              <TableCell align="center">Property Address</TableCell>
              <TableCell align="center">Owner Contact</TableCell>
              <TableCell align="center">Property Amt</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {allProperties.map((property) => (
              <TableRow
                key={property._id}
                sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
              >
                <TableCell component="th" scope="row">
```

```

        {property._id}
      </TableCell>
      <TableCell align="center">{property.ownerId}</TableCell>
      <TableCell align="center">{property.propertyType}</TableCell>
      <TableCell align="center">{property.propertyType}</TableCell>
      <TableCell align="center">{property.propertyAddress}</TableCell>
      <TableCell align="center">{property.ownerContact}</TableCell>
      <TableCell align="center">{property.propertyAmt}</TableCell>
    </TableRow>
  ))}
</TableBody>
</Table>
</TableContainer>
</div>
);
};

export default AllProperty;

```

8.5 AllUsers.jsx

```

const AllUsers = () => {
  const [allUser, setAllUser] = useState([]);

  useEffect(() => {
    getAllUser();
  }, []);

  const getAllUser = async () => {
    try {
      const response = await axios.get('http://localhost:8001/api/admin/getallusers', {
        headers: { 'Authorization': `Bearer ${localStorage.getItem("token")}` }
      });

      if (response.data.success) {
        setAllUser(response.data.data);
      } else {
        message.error(response.data.message);
      }
    } catch (error) {
      console.log(error);
    }
  };
};

```

```

const handleStatus = async (userid, status) => {
  try {
    await axios.post('http://localhost:8001/api/admin/handlestatus', { userid, status }, {
      headers: { 'Authorization': `Bearer ${localStorage.getItem("token")}` }
    }).then((res) => {
      if (res.data.success) {
        getAllUser();
      }
    });
  } catch (error) {
    console.log(error);
  }
};

return (
  <div>
    <TableContainer component={Paper}>
      <Table sx={{ minWidth: 650 }} aria-label="simple table">
        <TableHead>
          <TableRow>
            <TableCell>User ID</TableCell>
            <TableCell align="center">Name</TableCell>
            <TableCell align="center">Email</TableCell>
            <TableCell align="center">Type</TableCell>
            <TableCell align="center">Granted (for Owners users only)</TableCell>
            <TableCell align="center">Actions</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {allUser.map((user) => (
            <TableRow
              key={user._id}
              sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
            >
              <TableCell component="th" scope="row">
                {user._id}
              </TableCell>
              <TableCell align="center">{user.name}</TableCell>
              <TableCell align="center">{user.email}</TableCell>
              <TableCell align="center">{user.type}</TableCell>
              <TableCell align="center">{user.granted}</TableCell>
              <TableCell align="center">
                {user.type === 'Owner' && user.granted === 'ungranted' ? (
                  <Button onClick={() => handleStatus(user._id, 'granted')} size='small'
                    variant="contained" color="success">
                      Granted
                    </Button>
                ) : user.type === 'Owner' && user.granted === 'granted' ? (

```



```

                <Button onClick={() => handleStatus(user._id, 'ungranted')} size='small'
variant="outlined" color="error">
                    Ungranted
                </Button>
            ) : null}
        </TableCell></TableRow>
    )}
</TableBody>
</Table>
</TableContainer>
</div>
);
};

export default AllUserproperty

```

8.6 Home.jsx:

```

const Home = () => {
    const [index, setIndex] = useState(0);

    const handleSelect = (selectedIndex) => {
        setIndex(selectedIndex);
    };

    return (
        <
            <Navbar expand="lg" className="bg-body-tertiary">
                <Container fluid>
                    <Navbar.Brand><h2>RentEase</h2></Navbar.Brand>
                    <Navbar.Toggle aria-controls="navbarScroll" />
                    <Navbar.Collapse id="navbarScroll">
                        <Nav
                            className="me-auto my-2 my-lg-0"
                            style={{ maxHeight: '100px' }}
                            navbarScroll
                        >
                            </Nav>
                        <Nav>
                            <Link to={'/'}>Home</Link>
                            <Link to={'/login'}>Login</Link>
                            <Link to={'/register'}>Register</Link>
                        </Nav>
                    </Navbar.Collapse>
                </Container>
            </Navbar>
        >
    )
}

```

```

        </Navbar.Collapse>
      </Container>
    </Navbar>

    <div className='home-body'>
      <Carousel activeIndex={index} onSelect={handleSelect}>
        <Carousel.Item>
          <img
            src={p1}
            alt="First slide"
          />
        </Carousel.Item>
        <Carousel.Item>
          <img
            src={p2}
            alt="Second slide"
          />
        </Carousel.Item>
        <Carousel.Item>
          <img
            src={p3}
            alt="Third slide"
          />
        </Carousel.Item>
        <Carousel.Item>
          <img
            src={p4}
            alt="Fourth slide"
          />
        </Carousel.Item>
      </Carousel>
    </div>

    <div className='property-content'>
      <div className='text-center'>
        <h1 className='m-1 p-5'>All Properties that may you look for</h1>
        <p style={{fontSize: 15, fontWeight: 800}}>Want to post your Property? <Link
to={'/register'}><Button variant='outline-info'>Register as Owner</Button></Link></p>
      </div>

      <Container>
        <AllPropertiesCards />
      </Container>
    </div>
  </>
)
}

```

export default Home

8.7 Login.jsx

```
const handleSubmit = (e) => {
  e.preventDefault();

  if (!data?.email || !data?.password) {
    return alert("Please fill all fields");
  } else {
    axios.post('http://localhost:8001/api/user/login', data)
      .then((res) => {
        if (res.data.success) {
          message.success(res.data.message);

          localStorage.setItem("token", res.data.token);
          localStorage.setItem("user", JSON.stringify(res.data.user));
          const isLoggedIn = JSON.parse(localStorage.getItem("user"));

          switch (isLoggedIn.type) {
            case "Admin":
              navigate("/adminhome");
              break;
            case "Renter":
              navigate("/renterhome");
              break;
            case "Owner":
              if (isLoggedIn.granted === 'ungranted') {
                message.error('Your account is not yet confirmed by the admin');
              } else {
                navigate("/ownerhome");
              }
              break;
            default:
              navigate("/login");
              break;
          }
          setTimeout(()=>{
            window.location.reload()
          },1000)
        } else {
          message.error(res.data.message);
        }
      })
      .catch((err) => {
        if (err.response && err.response.status === 401) {
```

```

        alert("User doesn't exist");
    }
    navigate("/login");
  });
}
};

```

8.8 Register.jsx

```

const Register = () => {

  const navigate = useNavigate()
  const [data, setData] = useState({
    name: "",
    email: "",
    password: "",
    type: ""
  })

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

  const handleSubmit = (e) => {
    e.preventDefault()
    if (!data?.name || !data?.email || !data?.password || !data?.type ) return alert("Please fill all fields");
    else {
      axios.post('http://localhost:8001/api/user/register', data)
        .then((response) => {
          if (response.data.success) {
            message.success(response.data.message);
            navigate('/login')

          } else {
            message.error(response.data.message)
          }
        })
        .catch((error) => {
          console.log("Error", error);
        });
    }
  };

  return (
    <

```

```

<Navbar expand="lg" className="bg-body-tertiary">
  <Container fluid>
    <Navbar.Brand><h2>RentEase</h2></Navbar.Brand>
    <Navbar.Toggle aria-controls="navbarScroll" />
    <Navbar.Collapse id="navbarScroll">
      <Nav
        className="me-auto my-2 my-lg-0"
        style={{ maxHeight: '100px' }}
        navbarScroll
      >
      </Nav>
      <Nav>
        <Link to={'/'}>Home</Link>
        <Link to={'/login'}>Login</Link>
        <Link to={'/register'}>Register</Link>
      </Nav>

    </Navbar.Collapse>
  </Container>
</Navbar>

<Container component="main" >
  <Box
    sx={{
      marginTop: 8,
      marginBottom: 4,
      display: 'flex',
      flexDirection: 'column',
      alignItems: 'center',
    }}
  >

```

Backend Program Pages:

8.9 a) AdminController.jsx

```

const propertySchema = require("../schemas/propertyModel");

const userSchema = require("../schemas/userModel");
const bookingSchema = require("../schemas/bookingModel");

/////////getting all users/////////
const getAllUsersController = async (req, res) => {
  try {
    const allUsers = await userSchema.find({});
    if (!allUsers) {
      return res.status(401).send({
        success: false,

```

```

        message: "No users presents",
    });
    } else {
    return res.status(200).send({
        success: true,
        message: "All users",
        data: allUsers,
    });
    }
    } catch (error) {
    console.log("Error in get All Users Controller ", error);
    }
    };

//////////handling status for owner//////////
const handleStatusController = async (req, res) => {
    const { userid, status } = req.body;
    try {
        const user = await userSchema.findByIdAndUpdate(
            userid,
            { granted: status },
            { new: true }
        );
        return res.status(200).send({
            success: true,
            message: `User has been ${status}`,
        });
    } catch (error) {
        console.log("Error in get All Users Controller ", error);
    }
};

```

8.9 b) OwnerController.jsx

```

const bookingSchema = require("../schemas/bookingModel");
const propertySchema = require("../schemas/propertyModel");
const userSchema = require("../schemas/userModel");

//////////adding property by owner//////////
const addPropertyController = async (req, res) => {
    try {
        let images = [];
        if (req.files) {
            images = req.files.map((file) => ({
                filename: file.filename,
                path: `~/uploads/${file.filename}`,
            }));
        }
    }
};

```

```

    }

    const user = await userSchema.findById({ _id: req.body.userId });

    const newPropertyData = new propertySchema({
      ...req.body,
      propertyImage: images,
      ownerId: user._id,
      ownerName: user.name,
      isAvailable: "Available",
    });

    await newPropertyData.save();

    return res.status(200).send({
      success: true,
      message: "New Property has been stored",
    });
  } catch (error) {
    console.log("Error in get All Users Controller ", error);
  }
};

//////////all properties of owner//////////
const getAllOwnerPropertiesController = async (req, res) => {
  const { userId } = req.body;
  try {
    const getAllProperties = await propertySchema.find();
    const updatedProperties = getAllProperties.filter(
      (property) => property.ownerId.toString() === userId
    );
    return res.status(200).send({
      success: true,
      data: updatedProperties,
    });
  } catch (error) {
    console.error(error);
    return res
      .status(500)
      .send({ message: "Internal server error", success: false });
  }
};

```

8.9 c) UserController.jsx

```
const registerController = async (req, res) => {

  try {
    let granted = "";
    const existsUser = await userSchema.findOne({ email: req.body.email });
    if (existsUser) {
      return res
        .status(200)
        .send({ message: "User already exists", success: false });
    }
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);
    req.body.password = hashedPassword;

    if (req.body.type === "Owner") {
      granted = "ungranted";
      const newUser = new userSchema({ ...req.body, granted });
      await newUser.save();
    } else {
      const newUser = new userSchema(req.body);
      await newUser.save();
    }

    ///////////aur you can do this/////////
    //   if (req.body.type === "Owner") {
    //     newUser.set("granted", "pending", { strict: false });
    //   }
    /////////// for this, then you need to remove strict keyword from schema/////////

    return res.status(201).send({ message: "Register Success", success: true });
  } catch (error) {
    console.log(error);
    return res
      .status(500)
      .send({ success: false, message: `${error.message}` });
  }
};
```

8.9a) AdminRoutes.jsx

```
const express = require("express");

const authMiddleware = require("../middlewares/authMiddleware");
```



```

const { getAllUsersController, handleStatusController, getAllPropertiesController,
getAllBookingsController } = require("../controllers/adminController");

const router = express.Router()

router.get('/getallusers', authMiddleware, getAllUsersController)

router.post('/handlestatus', authMiddleware, handleStatusController)

router.get('/getallproperties', authMiddleware, getAllPropertiesController)

router.get('/getallbookings', authMiddleware, getAllBookingsController)

module.exports = router

```

8.10 b) OwnerRoutes.js

```

const express = require("express");

const multer = require("multer");

const authMiddleware = require("../middlewares/authMiddleware");

const {
  addPropertyController,
  getAllOwnerPropertiesController,
  deletePropertyController,
  updatePropertyController,
  getAllBookingsController,
  handleAllBookingstatusController,
} = require("../controllers/ownerController");

const router = express.Router();

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./uploads/");
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage: storage });

```

```

router.post(
  "/postproperty",
  upload.array("propertyImages"),
  authMiddleware,
  addPropertyController
);

router.get("/getallproperties", authMiddleware, getAllOwnerPropertiesController);

router.get("/getallbookings", authMiddleware, getAllBookingsController);

router.post("/handlebookingstatus", authMiddleware, handleAllBookingstatusController);

router.delete(
  "/deleteproperty/:propertyid",
  authMiddleware,
  deletePropertyController
)

```

8.10 c) UserRoutes.js

```

const express = require("express");
const authMiddleware = require("../middlewares/authMiddleware");

const {
  registerController,
  loginController,
  forgotPasswordController,
  authController,
  getAllPropertiesController,
  bookingHandleController,
  getAllBookingsController,
} = require("../controllers/userController");

const router = express.Router();
router.post("/register", registerController);

router.post("/login", loginController);

router.post("/forgotpassword", forgotPasswordController);

router.get('/getAllProperties', getAllPropertiesController)

router.post("/getuserdata", authMiddleware, authController);

router.post("/bookinghandle/:propertyid", authMiddleware, bookingHandleController);

router.get('/getallbookings', authMiddleware, getAllBookingsController)

```

```
module.exports = router;
```

8.11 a) bookingModel.js

```
const mongoose = require("mongoose");
```

```
const bookingModel = mongoose.Schema(  
  {  
    propertId: {  
      type: mongoose.Schema.Types.ObjectId,  
      ref: "propertySchema",  
    },  
    ownerId: {  
      type: mongoose.Schema.Types.ObjectId,  
      ref: "user",  
    },  
    userID: {  
      type: mongoose.Schema.Types.ObjectId,  
      ref: "user",  
    },  
    userName: {  
      type: String,  
      required: [true, "Please provide a User Name"],  
    },  
    phone: {  
      type: Number,  
      required: [true, "Please provide a Phone Number"],  
    },  
    bookingStatus: {  
      type: String,  
      required: [true, "Please provide a booking Type"],  
    },  
  },  
  {  
    strict: false,  
  }  
);
```

```
const bookingSchema = mongoose.model("bookingschema", bookingModel);
```

```
module.exports = bookingSchema;
```

8.11 b) propertyModel.js

```
const mongoose = require('mongoose')

const propertyModel = mongoose.Schema({
  ownerId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'user'
  },
  propertyType: {
    type: String,
    required: [true, 'Please provide a Property Type']
  },
  propertyAdType: {
    type: String,
    required: [true, 'Please provide a Property Ad Type']
  },
  propertyAddress: {
    type: String,
    required: [true, "Please Provide an Address"]
  },
  ownerContact: {
    type: Number,
    required: [true, 'Please provide owner contact']
  },
  propertyAmt: {
    type: Number,
    default: 0,
  },
  propertyImage: {
    type: Object
  },
  additionalInfo: {
    type: String,
  },
  ownerName: {
    type: String,
  }
}, {
  strict: false,
})

const propertySchema = mongoose.model('propertyschema', propertyModel)

module.exports = propertySchema
```

8.11 c) UserModel.js

```
const mongoose = require("mongoose");
const userModel = mongoose.Schema({
  name: {
    type: String,
    required: [true, "Name is required"],
    set: function (value) {
      return value.charAt(0).toUpperCase() + value.slice(1);
    },
  },
  email: {
    type: String,
    required: [true, "email is required"],
  },
  password: {
    type: String,
    required: [true, "password is required"],
  },
  type: {
    type: String,
    required: [true, "type is required"],
  },
}, {
  strict: false,
});

const userSchema = mongoose.model("user", userModel);
module.exports = userSchema;
```

8.12 Connect.js

```
const mongoose = require('mongoose');

const connectionOfDb = () => {

  mongoose
    .connect(process.env.MONGO_DB, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    .then(() => {
      console.log('Connected to MongoDB');
    })
    .catch((err) => {
      throw new Error(`Could not connect to MongoDB: ${err}`);
    });
}; module.exports = connectionOfDb;
```

8.13 Index.js

```
const express = require("express");
const dotenv = require("dotenv");
const cors = require("cors");
const connectionofDb = require("./config/connect.js");
const path = require("path");

const app = express();

/////dotenv config/////////////////////////////////
dotenv.config();

/////connection to DB/////////////////////////////////
connectionofDb();

//////////port number/////////////////////////////////
const PORT = process.env.PORT || 8000;

//////////middlewares/////////////////////////////////
app.use(express.json());
app.use(cors());

//////////routes/////////////////////////////////
app.use("/uploads", express.static(path.join(__dirname, "uploads")));

app.use('/api/user', require('./routes/userRoutes.js'))
app.use('/api/admin', require('./routes/adminRoutes'))
app.use('/api/owner', require('./routes/ownerRoutes'))

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

8.14 Package json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
```

```
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "bcryptjs": "^2.4.3",  
    "cors": "^2.8.5",  
    "dotenv": "^16.3.1",  
    "express": "^4.18.2",  
    "jsonwebtoken": "^9.0.1",  
    "mongoose": "^7.4.3",  
    "multer": "^1.4.5-lts.1",  
    "nodemon": "^3.0.1"  
  }  
}
```

9. API DOCUMENTATION

1. User Registration

1. The user submits their details via a POST request.
2. The server validates the inputs (e.g., checks if the email is unique and the password meets security standards).
3. If valid, the server stores the user's information (typically hashed passwords for security) in the database.
4. A success message or user ID is returned.

2. User Login:

1. The user sends their email and password via a POST request.
2. The server verifies the email and password combination (passwords are matched using a secure hash function).
3. If valid, the server generates a JWT (JSON Web Token) or similar token and sends it back to the client.
4. The client stores the token (in local storage or cookies) and attaches it to requests for accessing protected resources.

3. Get User Details:

1. The client includes their JWT token in the request header.
2. The server verifies the token to ensure the request is coming from an authenticated user.
3. If valid, the server retrieves the user's details from the database and sends them back.

4. Update User Profile:

1. The client sends the updated data (e.g., new name or password) in a PUT request.
2. The server verifies the user's token and checks if the new data meets validation rules.
3. If valid, the server updates the database with the new details.
4. A success message is returned.

Purpose: Allow users to maintain and modify their account information.

5. Delete User Account

1. The client sends a DELETE request with their token for authentication.
2. The server verifies the token and checks if the user has the authority to delete the account.
3. If valid, the server removes the user's data from the database.
4. A confirmation message is returned.

Admin API Steps:

1. User Management

- The Admin API helps administrators handle user-related tasks to ensure the platform remains safe and functional.
- Create, Update, or Delete User Accounts: Admins can register new users manually, update details, or delete accounts of users who violate policies.

2. Listing Management

- Admins oversee and control property listings to maintain the quality and authenticity of content.
- Create or Modify Listings: Add or edit rental property details to ensure accuracy.

3. Content Moderation

- Content moderation ensures the platform maintains a high standard of safety and compliance.
- Handle Reports: Respond to user complaints about listings, messages, or profiles.

4. Analytics and Insights

- The Admin API provides data and metrics to help manage the platform effectively.
- Track User Growth: Monitor the number of new users, active users, and churn rates.

5. Notifications and Communication

- The Admin API is used to communicate effectively with users and enforce policies.
- Broadcast Announcements: Notify users about updates, new features, or policy changes.

10. AUTHENTICATION MECHANISM

1. User Registration

During the first interaction, users (tenants or landlords) register by providing their details.

Required Information:

- Full Name
- Email Address (or Mobile Number)
- Password (hashed and salted before storage)

Steps:

- Email/phone verification (OTP or confirmation link).
- Securely hash and salt passwords using algorithms like bcrypt, Argon2, or PBKDF2.
- Store the data in a secure database.

2. User Login

Once registered, users log in to access their accounts.

Steps:

- email/phone number and password.
- Authenticate credentials using:
- Password comparison with stored hash.
- Email/phone number verification if required.

Multi-factor Authentication (MFA)

Add a second layer of authentication for enhanced security.

- **SMS/Email OTP:** Send a One-Time Password to the user's registered number/email.

- **Authenticator Apps:** Use apps like Google Authenticator to generate time-based codes.
- **Biometric Authentication:** Leverage device biometrics like fingerprints or facial recognition.

3. Social Login:

Allow users to log in via their social media accounts.

Methods:

- OAuth2-based login using platforms like Google, Facebook, or Apple.
- Fetch basic user profile data securely

4. Password Management

Forgot Password:

Users request a password reset link or OTP via email/phone.

Reset link should expire after a short time.

Change Password:

Users can update their password after logging in, with verification via the old password or OTP.

5. Session Management

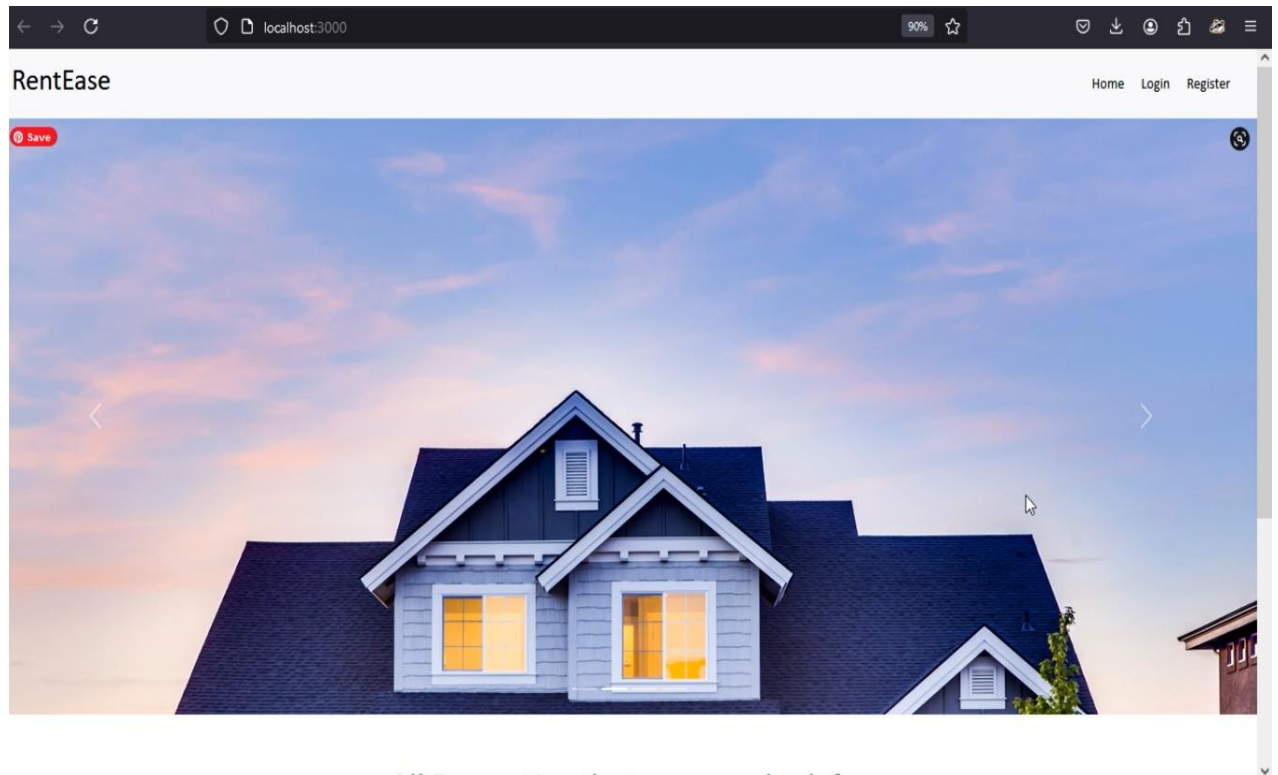
- **Token-based Authentication:**

Use JWT (JSON Web Token) for session management. Tokens contain user data, are signed with a secret key, and expire after a set time.

- **Session Persistence:** Store tokens in HTTP-only cookies to protect against XSS attack.

11. USER INTERFACE

Home Page



The homepage serves as the entry point to the system. It should be user-friendly and visually appealing.

Features:

- Home
- Property List
- Login (for users and owners)
- Contact Us/About Us
- A search bar to filter properties by location, type, or rent range.
- Featured properties section showcasing popular or recently added properties.
- Footer with basic links like privacy policy and social media handles.

User Login

← → ↻ localhost:3000/login 90% ☆

RentEase Home Login Register

Sign In

Email Address

Password

SIGN UP

forgot password? [Click here](#) Have an account? [Sign Up](#)

© 2024 Copyright: RentEase

This page is for tenants looking for properties to rent.


Features:

- Profile Management: Edit personal details, view application history.
- Search & Filters: Search properties by price, location, type.
- Rent Applications: Apply for properties directly.
- Payment Status: View and manage rental payments.

User Signup


← → ↻ localhost:3000/login 90% ☆

RentEase Home Login Register

 Sign In

Email Address
jhon123@gmail.com

Password
•••••

 SIGN UP

forgot password? [Click here](#) Have an account? [Sign Up](#)

© 2024 Copyright: RentEase

Features of the Signup Page

1. Collect basic user details:

- Full Name
- Email
- Password
- Confirm Password
- Contact Number

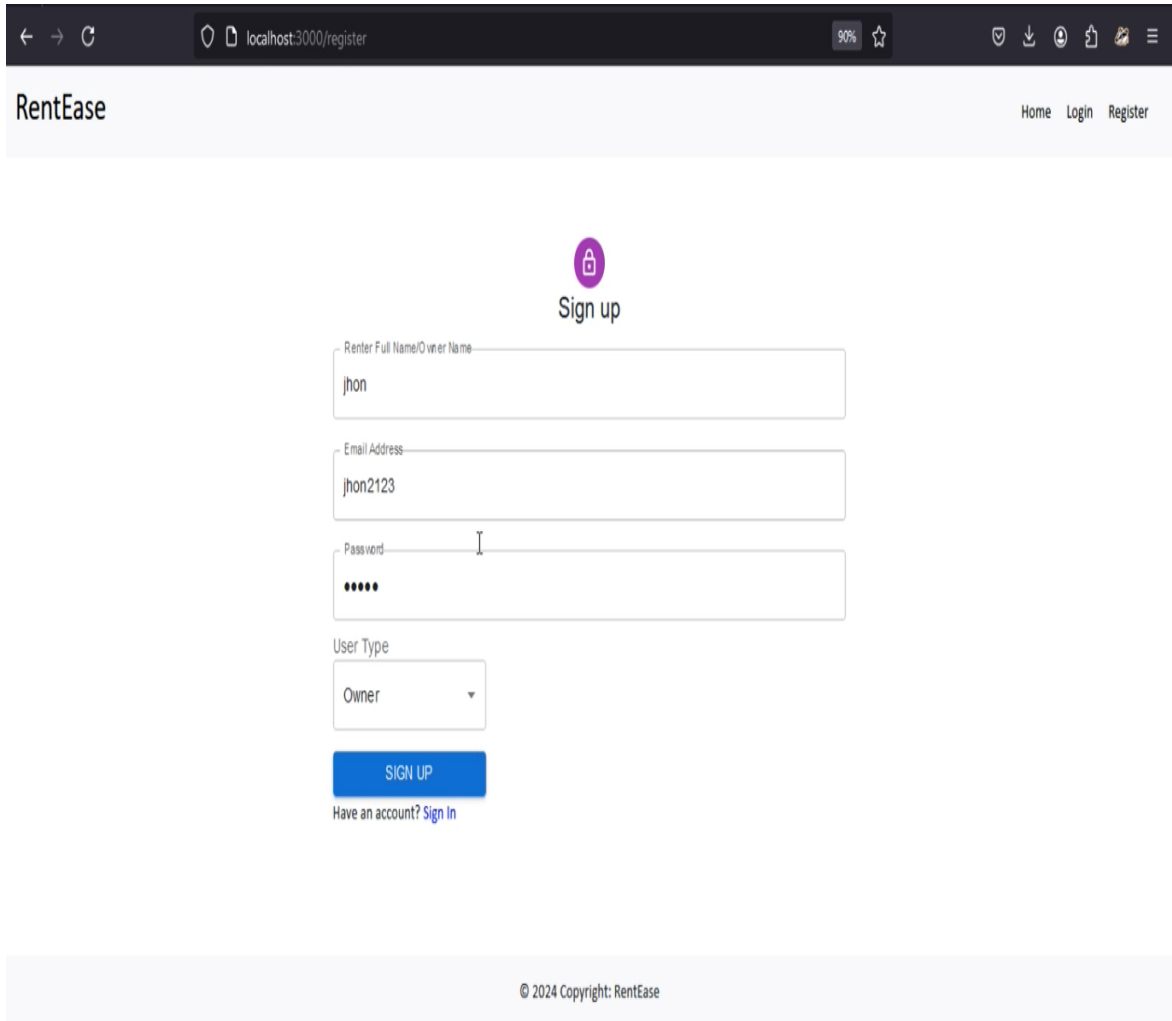
2. Input validation for:

- Email format
- Password strength
- Matching passwords

3. A submit button to save user details in the database.

4. Option to redirect to the login page after successful signup.

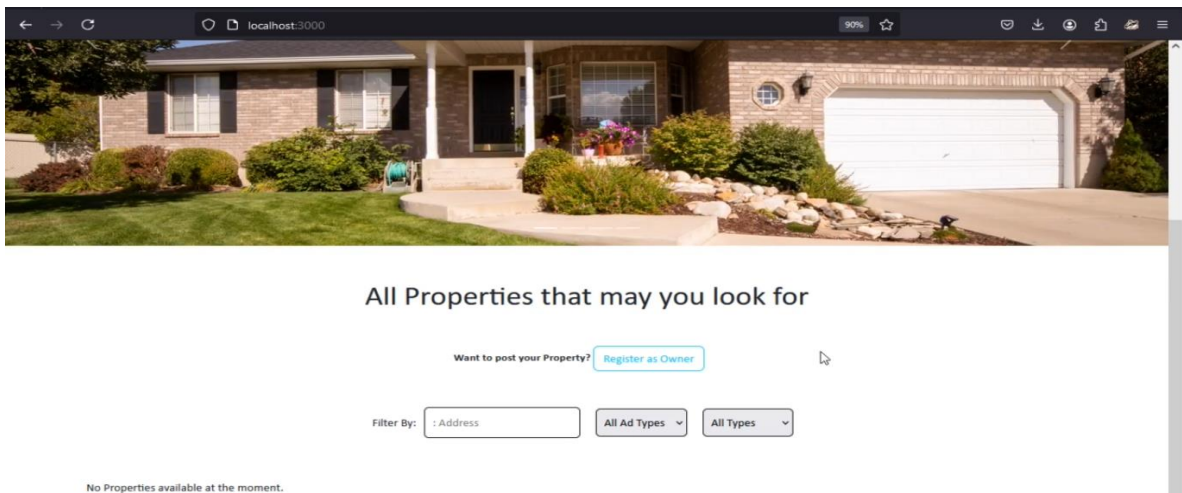
Owner Login page



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/register'. The page title is 'RentEase'. In the top right corner, there are links for 'Home', 'Login', and 'Register'. The main content area features a 'Sign up' heading with a purple lock icon. Below this, there are three input fields: 'Renter Full Name/Owner Name' containing 'Jhon', 'Email Address' containing 'Jhon2123', and 'Password' containing five dots. A 'User Type' dropdown menu is set to 'Owner'. A blue 'SIGN UP' button is positioned below the dropdown. At the bottom of the form, there is a link that says 'Have an account? Sign In'. The footer of the page contains the copyright notice '© 2024 Copyright: RentEase'.

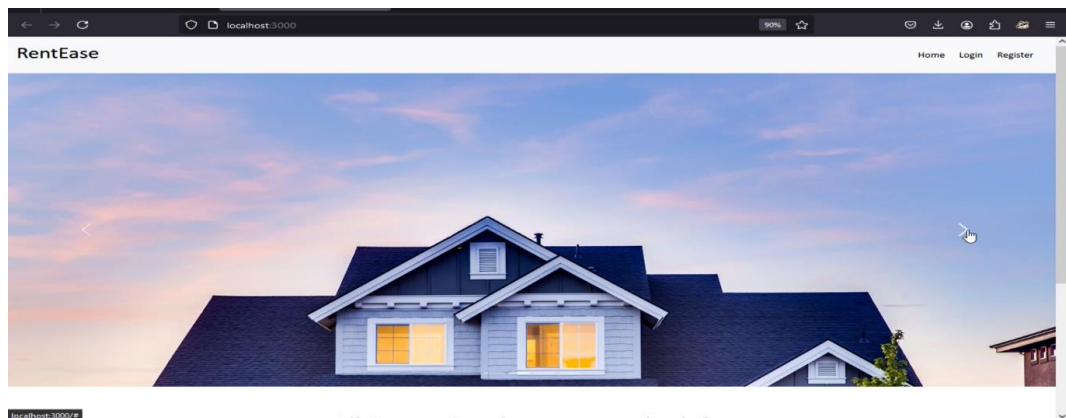
1. Allow property owners to log in using their email and password.
2. Validate credentials against the database.
3. Redirect successful logins to the Owner Dashboard.

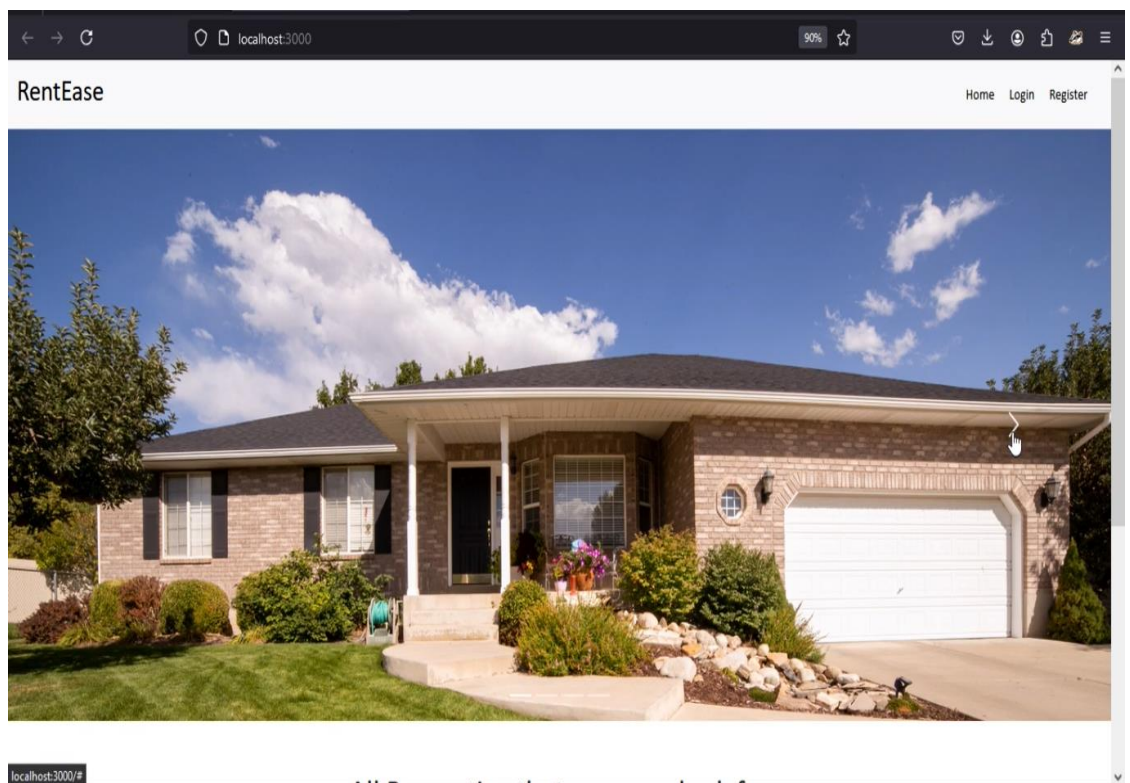
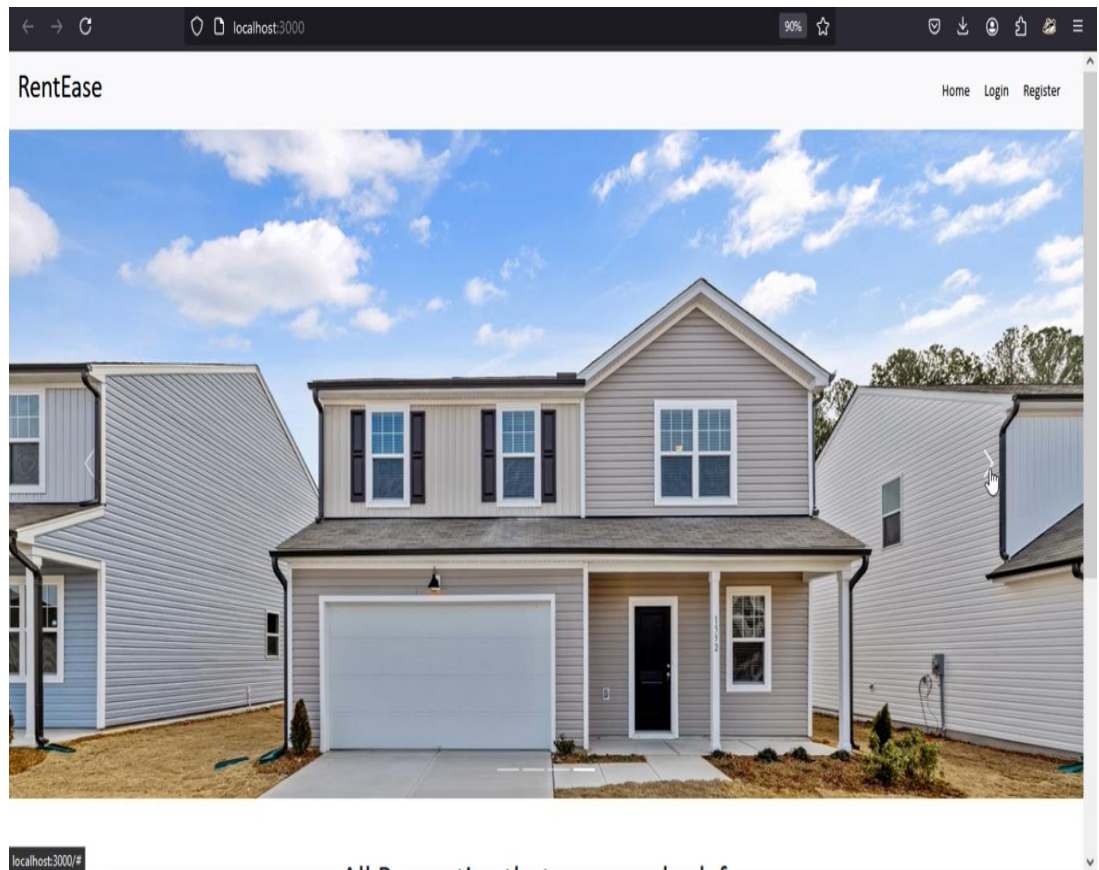
Register as owner page



1. Allow new property owners to create an account.
2. Validate user inputs (e.g., email, phone number, password).
3. Store owner details securely in the database.
4. Send a verification email to confirm the owner's identity.
5. Provide a user-friendly, responsive interface.

Property for Rent in App





12. TESTING STRATEGY

1. Performance Testing

- High traffic during peak rental seasons.
- Multiple tenants searching for properties simultaneously.
- Tools: Apache JMeter, LoadRunner.

2. Security Testing

- Ensure passwords are stored as hashed and salted values.
- Test for SQL injection, XSS, and CSRF vulnerabilities.
- Verify payment data is transmitted over secure channels (TL)

3.Frontend (React) Testing Tools

- Jest: For unit testing React components.
- React Testing Library: For testing UI components and interactions.
- Cypress: For end-to-end testing of the user interface.
- Playwright: For cross-browser UI testing.

4.Backend (Node.js/Express) Testing Tools

- Mocha: For testing backend routes and logic.
- Chai: Assertion library used with Mocha.
- Supertest: For testing HTTP requests and API endpoints.

5.Database (MongoDB) Testing Tools

- MongoDB Memory Server: For in-memory database testing.
- Mongoose: For validating schema operations.
- End-to-End Testing Tools
- Cypress or Puppeteer: For simulating real-world scenarios like user login, property search, and booking.

13.KNOWN ISSUES

1. Functional Issues

- **Search Failures:** Bugs in search functionality, such as irrelevant results or incorrect sorting, reduce usability.
- **Booking Errors:** Issues with booking or reserving properties, such as duplicate bookings or failed confirmations, can lead to user distrust.
- **Notification Glitches:** Delays or spam-like notifications may irritate users or cause them to miss important updates.
- **Payment Failures:** Problems with online payments, including delays, errors, or lack of secure gateways, pose significant risks.
- **Account Management Problems:** Issues with login, password resets, or data syncing can frustrate user.

2. Property Listing Issues

- **Fake Listings:** Presence of fraudulent or outdated property listings harms credibility.
- **Incomplete Listings:** Missing photos or essential details create a negative impression.

3. Technical Issues

- **Compatibility Problems:** App may not work well on certain devices, operating systems, or browser versions.

4. Monetization Issues

- **High Service Fees:** Excessive charges for tenants or landlords may reduce app usage.

- **In-App Ads:** Intrusive or irrelevant advertisements can disrupt user experience.

4.API Performance:

Problem: Slow response times when querying large datasets in MongoDB (e.g., all property listings).

Cause: Inefficient database queries or lack of indexing.

Solution:

Implement proper indexes in MongoDB for frequently queried fields (e.g., location, price).

Use pagination for API responses.

5.Data Validation

Problem: Improper or incomplete user input leads to inconsistent database entries.

Cause: Insufficient input validation.

Solution:

- Use a library like Joi or Validator.js for backend input validation.
- Validate fields at both frontend and backend levels.

Scalability Bottlenecks

Problem: App crashes or slows down under high traffic.

Cause: Single-threaded nature of Node.js and lack of load balancing.

Solution:

- Use clustering or a process manager like PM2.

- Implement caching using Redis for frequently accessed data (e.g., popular property listings).

Security Vulnerabilities

Problem: Vulnerabilities such as SQL/NoSQL injection, broken authentication, or insecure APIs.

Cause: Insufficient input sanitization or token verification.

Solution:

- Use libraries like Helmet for HTTP header security.
- Sanitize user inputs using mongoose-sanitize.
- Implement robust token-based authentication (e.g., JWT with proper expiration).

6. Frontend Issues (React)

State Management

Problem: Inconsistent data or UI state across components.

Cause: Improper handling of global states in larger apps.

Solution:

- Use a state management library like Redux or Context API.
- Avoid prop drilling by restructuring components.

7. Database Issues (MongoDB)

Data Redundancy

Problem: Duplicate property or user entries.

Cause: Poor schema design or lack of constraints.

Solution:

- Use unique fields for critical data (e.g., email, property ID).
- Implement checks before inserting new records.

8. Schema Scalability

Problem: Schema design doesn't scale well with added features (e.g., adding reviews, amenities).

Cause: Overly simplistic initial schema.

Solution:

- Design schemas with scalability in mind using embedded or referenced documents.
- Regularly review and refactor the schema as the app grows.

9. Authentication and Authorization Issues**Session Management**

Problem: User sessions don't expire or are not secure.

Cause: Poor implementation of JWT or cookie ma...

Solution:

- Implement short-lived tokens with refresh token support.
- Use HttpOnly and Secure flags for cookies.

10. Password Security

Problem: Plaintext passwords are stored in the database.

Cause: Lack of password hashing.

14. FUTURE ENHANCEMENTS

1. Advanced Search & Personalization:

- **AI-Powered Recommendations:**

Use machine learning to suggest properties based on user preferences, search history, and location trends.

- **Voice Search:**

Allow users to search properties using voice commands.

- **Augmented Reality (AR):**

Enable virtual walk-throughs using AR for better visualization of properties.

2. Improved User Experience:

- **Gamification:**

Rewards for users (e.g., discounts or points for early rent payments, referrals, etc.).

- **Smart Notifications:**

Personalized alerts for new properties, rent due dates, or lease expiry.

- **Offline Functionality:**

Save property details for offline access when users have limited internet connectivity.

3. Financial Integrations:

- **Credit Score Integration:**

Allow users to check their eligibility for renting based on their credit scores.

- **EMI or Loan Options:** Collaborate with financial institutions to offer loans for security deposits or long-term leases.

CONCLUSION

The House Rent App built using the MERN stack demonstrates the power and versatility of modern web development technologies. By utilizing MongoDB as a NoSQL database, the app ensures high scalability and flexibility in handling user data, property listings, and transactions. The Express.js backend provides a robust and efficient framework for handling HTTP requests and managing user authentication, while Node.js enables seamless server-side operations with asynchronous processing for improved performance. The app also integrates secure payment gateways and interactive map features to allow users to easily view property locations and complete transactions securely. The MERN stack's full-stack nature allows for easy maintenance and future scalability, making it a perfect choice for building a high-performance rental platform. House Rent App built with the MERN stack offers a seamless and efficient solution for the rental market, with the potential for rapid growth and easy adaptation to emerging technologies.

REFERENCE

1. **React Documentation** - <https://reactjs.org/docs>
2. **Node.js Documentation** - <https://nodejs.org/en/docs>
3. **Express.js Guide** - <https://expressjs.com/en/guide>
4. **MongoDB Documentation** - <https://www.mongodb.com/docs>
5. **JWT Authentication** - <https://jwt.io/introduction>
6. **MERN Stack Tutorials** – <https://www.freecodecamp.org>
7. **Testing** - <https://www.cypress.io>
8. **Bootstrap** - <https://getbootstrap.com/docs>