PROJECT TITLE

HOUSE HUNT:FINDING YOUR PERFECT RENTAL HOME

Full Stack Development Project

Team Leader : Ambati Shiva sai
Team member : Abdul Mueed
Team member : Adapala Harshini
Team member :Abdul Kafeel

Phase 1: Brainstorming & Ideation
Objective:
To build a user-friendly, full-stack house rental platform that simplifies property listings, booking, communication, and administrative approvals with role-based access for Renters, Owners, and Admins.
Key Points:

- Full-stack MERN (MongoDB, Express, React, Node.js) based web app
- Role-based login: Renter, Owner, Admin
- Owner property listings and booking management
- Renter messaging and booking system
- Admin dashboard for account approval
- Secure authentication with JWT & bcrypt
- Real-time communication via in-app messaging

Problem Statement:
Rental processes are often fragmented across platforms, with limited trust, poor communication, and no centralized approval system. Property owners lack visibility, renters can’t connect directly, and there's no admin oversight.
Proposed Solution:
A centralized application with clear user roles, secure login, listing management, direct messaging, and booking capabilities, all moderated by an admin interface.
Target Users:

- Property Owners
- Renters (individuals/families)
- Admin (platform manager)

Expected Outcomes:

- Simplified property listing and renting
- Seamless communication and booking process
- Admin-controlled approval for verified activity
- Scalable and user-friendly architecture

Phase 2: Requirement Analysis
Objective:
To analyze technical and functional requirements for building a reliable rental platform supporting communication, listings, bookings, and role-based access.
Key Features & Requirements:
Technical Requirements:
- **Frontend**: React.js + Bootstrap/Material UI
- **Backend**: Node.js + Express
- **Database**: MongoDB
- **Authentication**: JWT tokens & bcrypt encryption
- **Image Upload**: File system or Cloudinary
- **Messaging System**: MongoDB-based threaded messages
- **Booking System**: Status-tracked booking model
- **Email/Notifications**: Nodemailer (planned)

Functional Requirements:
- Register/Login by role
- Owner: Create/Edit/Delete listings, View messages, Manage bookings
- Renter: Browse listings, Filter by rent/sale, Book property, View history
- Admin: Approve owner accounts, Monitor platform
- Renter-to-Owner chat with history
- Booking request & approval system
- Admin dashboard for verification

Constraints & Challenges:
- Ensuring secure file uploads
- User state management after login
- Synchronizing messages and bookings across users
- Handling duplicate actions (e.g., multiple bookings)
- Admin flow integration without interrupting UX

Phase 3: Project Design
Objective:
To define architectural, database, and interface design supporting all user roles with scalability and clean separation of concerns.
Key Points:
- React-based UI with role-based dashboard redirection

- RESTful API structure for clean backend operations
- ER Diagram with models: User, Property, Message, Booking
- Modular pages: Login, Register, Owner Dashboard, Renter Dashboard, Admin Panel

System Architecture:

User Flow

- User logs in → redirected to role-specific dashboard
- Owners create/edit listings
- Renters browse/filter listings and send booking requests
- Owners view and approve/reject requests
- Admin reviews owners for approval

UI/UX Considerations

- Simple layout with top header and right sidebar
- Card-style listing for properties
- Modal for messaging
- Form validation and alerts
- Admin table with pending owner approval

Phase 4: Project Planning (Agile Sprints)

Objective:

To structure the project development using Agile sprint-based planning for modular, testable, and scalable delivery.

Key Points:

- Agile Scrum methodology with iterative 1-week sprints
- Task-based collaboration: frontend, backend, UI/UX, admin
- Continuous testing and integration
- Feedback-driven improvements during sprint retrospectives
- Deployment planning during later stages

Sprint Plan:

Sprint	Goal	Deliverables
Sprint 1	Setup & Authentication	Project structure, JWT auth, role-based register/login
Sprint 2	Role-based Dashboard UI	Login flow with conditional redirection, Renter/Owner/Admin dashboards
Sprint 3	Property Listing System	Owner can add, edit, delete properties with image upload
Sprint 4	Renter Browsing & Filtering	Renter views properties, filters rent/sale

Sprint	Goal	Deliverables
Sprint 5	Messaging Module	Renter sends messages, Owner replies, chat thread
Sprint 6	Booking System	Booking request by renter, approval/rejection by owner
Sprint 7	Admin Panel	View/approve pending owners, manage user roles
Sprint 8	Booking History + UI Fixes	Renter booking history, owner booking requests UI cleanup
Sprint 9	Email & Notification System	Email alerts using Nodemailer, basic in-app notifications
Sprint 10	Testing, Bug Fixing, Final Polish	Full testing, validation, deployment preparation

---

Task Allocation:

Role	Responsibility
Frontend Developer	React pages, form validation, message & booking UI
Backend Developer	Node.js APIs, MongoDB schema, JWT auth, booking & messaging logic
Admin	Owner verification, user moderation features
UI/UX Designer	Styling with Bootstrap/Material UI, responsive layout
Tester	Manual and functional testing of each sprint deliverable

---

Timeline & Milestones:

- **Week 1**: Authentication + Role-based routing
- **Week 2**: Dashboards (Renter, Owner, Admin)
- **Week 3**: Property listing CRUD + image upload
- **Week 4**: Renter messaging → Owner inbox
- **Week 5**: Booking system implementation
- **Week 6**: Admin approval and control features
- **Week 7**: Booking history & cleanup
- **Week 8**: Notifications + email alerts
- **Week 9**: Testing and final review

Phase 5: Project Development

Objective:
To implement the HouseHunt platform in modular, testable components using the MERN stack with clean code, secure logic, and user-friendly interfaces.
Tech Stack:

- **Frontend**: React, Bootstrap, Material UI
- **Backend**: Node.js, Express
- **Database**: MongoDB
- **Authentication**: JWT, bcrypt
- **Deployment**: (Localhost / optional cloud)
- **Additional Tools**: Moment.js, Nodemailer, Multer for file upload

Modules Completed:
✅ User registration/login with role separation
✅ Owner Dashboard: Add/Edit/Delete property with image
✅ Renter Dashboard: Filter listings, view & send messages
✅ Messaging: Renter-to-Owner with threaded inbox
✅ Booking system: Status tracking, owner action
✅ Admin approval system: Owner verification
✅ Booking history for renter
✅ Notifications + Email trigger (in progress)

Phase 6: Testing, Deployment & Conclusion
Objective:
To test all features thoroughly, fix bugs, polish UI, and prepare the project for deployment.
Testing Strategy:

- Unit testing for APIs
- Manual testing for all user flows
- Booking edge cases (duplicate booking, invalid input)
- Auth testing (expired tokens, role misuse)

Deployment Plan (Optional):

- Backend: Node.js on Render/Heroku
- Frontend: Vercel or Netlify
- Database: MongoDB Atlas
- Email service: Gmail SMTP via Nodemailer

Final Outcome:

- ✅ Functional rental platform for Renters, Owners, and Admins
- ✅ Real-time messaging and booking
- ✅ Secure login and role-specific features

- ✅ Admin moderation and control
- ✅ Modern, responsive UI
- 🔄 Future Scope: Notifications center, advanced search, payment integration

CODE:

Fornt-end:

```
// src/pages/OwnerDashboard.js
import React from 'react';
import { useNavigate } from 'react-router-dom';

export default function OwnerDashboard() {
  const navigate = useNavigate();
  const name = localStorage.getItem('userName') || 'Owner';

  const buttonStyle = {
    padding: '15px 30px',
    fontSize: '16px',
    margin: '10px',
    cursor: 'pointer',
    backgroundColor: '#007bff',
    color: '#fff',
    border: 'none',
    borderRadius: '5px'
  };

  return (
```

```
    <div className="container" style={{ textAlign: 'center', marginTop: '50px' }}>
      <h2>Welcome, {name}!</h2>
      <p>Manage your properties and connect with renters</p>

      <div style={{ marginTop: '30px' }}>
        <button onClick={() => navigate('/owner/property')} style={buttonStyle}>
          🏠 Enter Property Details
        </button>
        <button onClick={() => navigate('/owner/inbox')} style={buttonStyle}>
          💬 Message Box
        </button>
        <button onClick={() => navigate('/owner/bookings')} style={buttonStyle}>
          📩 Booking Requests
        </button>
      </div>
    </div>
  );
}


import React, { useEffect, useState } from 'react';
import API from '../services/api';
import { useNavigate } from 'react-router-dom';

export default function RenterDashboard() {
  const [properties, setProperties] = useState([]);
```

```
const [filter, setFilter] = useState('');
const [selectedProperty, setSelectedProperty] = useState(null);
const [message, setMessage] = useState('');
const [bookingNote, setBookingNote] = useState('');

const navigate = useNavigate();

useEffect(() => {
  const fetchProperties = async () => {
    try {
      const res = await API.get('/properties/all');
      setProperties(res.data);
    } catch (err) {
      console.error(err);
      alert('Failed to load properties');
    }
  };
  fetchProperties();
}, []);

const filteredProperties = filter
  ? properties.filter((p) => p.adType === filter)
  : properties;

const handleSendMessage = async () => {
  if (!message.trim()) return alert("Message cannot be empty");
```

```
  try {
    await API.post('/messages/send', {
      propertyId: selectedProperty._id,
      content: message,
    }, {
      headers: { Authorization: `Bearer ${localStorage.getItem('token')}` },
    });
    alert("Message sent");
    setMessage('');
    setSelectedProperty(null);
  } catch (err) {
    alert("Failed to send message");
  }
};

const handleBooking = async () => {
 if (!bookingNote.trim()) {
  alert("Please enter a booking message.");
  return;
 }

 try {
  const res = await API.post('/bookings/book', {
    propertyId: selectedProperty._id,
    message: bookingNote,
  }, {
```

```
      headers: {
        Authorization: `Bearer ${localStorage.getItem('token')}`,
      }
    });
    alert(res.data.message || 'Booking successful');
    setBookingNote('');
    setSelectedProperty(null);
  } catch (err) {
    console.error('Booking error:', err.response?.data || err.message);
    alert("Booking failed");
  }
};

  return (
    <div className="renter-container">
      <h2>Hello {localStorage.getItem('userName')}</h2>

      <div style={{ margin: '10px 0' }}>
        <button onClick={() => setFilter('Rent')}>🏠 For Rent</button>
        <button onClick={() => setFilter('Sale')}>🏘 On Sale</button>
      </div>

      {selectedProperty && (
        <div style={{ border: '2px solid #333', padding: 20, marginBottom: 20 }}>
          <h3>{selectedProperty.type} - ₹{selectedProperty.amount}</h3>
          <p><strong>Address:</strong> {selectedProperty.address}</p>
```

```
<p><strong>Description:</strong> {selectedProperty.info}</p>
<p><strong>Owner:</strong> {selectedProperty.owner?.name}</p>

{/* Messaging */}
<textarea
  value={message}
  onChange={(e) => setMessage(e.target.value)}
  placeholder="Send a message to the owner"
  rows={2}
  style={{ width: '100%', marginBottom: '10px' }}
/><br />
<button onClick={handleSendMessage}>Send Message</button>{' '}
<button onClick={() =>
navigate(`/messages/history/${selectedProperty._id}`)}>View Message
History</button>

{/* Booking */}
<hr />
<textarea
  value={bookingNote}
  onChange={(e) => setBookingNote(e.target.value)}
  placeholder="Write booking note or request"
  rows={2}
  style={{ width: '100%' }}
/><br />
<button onClick={handleBooking}>Book this Property</button>{' '}
<button onClick={() => setSelectedProperty(null)}>Close</button>
```

```
      </div>
    )}
<button onClick={() => navigate('/bookings/my')}>
  📖 View My Bookings
</button>



    <div>
      {filteredProperties.map((prop) => (
        <div
          key={prop._id}
          style={{
            border: '1px solid #ccc',
            padding: 15,
            marginBottom: 10,
            cursor: 'pointer',
          }}
          onClick={() => setSelectedProperty(prop)}


        >
          <h4>{prop.type} - ₹{prop.amount}</h4>
          <p>{prop.address}</p>
          {prop.image && (
            <img
              src={`http://localhost:5000${prop.image}`}
              alt="Property"
```

```
            width="200"
           />
          )}
         </div>
        ))}
      </div>
    </div>
  );
}
```

Back-end:

```
const express = require('express');
const router = express.Router();
const Message = require('../models/Message');
const Property = require('../models/Property');
const auth = require('../middleware/authMiddleware');

// ✅ Renter sends a message to the property owner
router.post('/send', auth, async (req, res) => {
  try {
    const { propertyId, content } = req.body;
    const property = await Property.findById(propertyId);
    if (!property) return res.status(404).json({ message: 'Property not found' });

    const message = new Message({
      property: propertyId,
      sender: req.user.id,
```

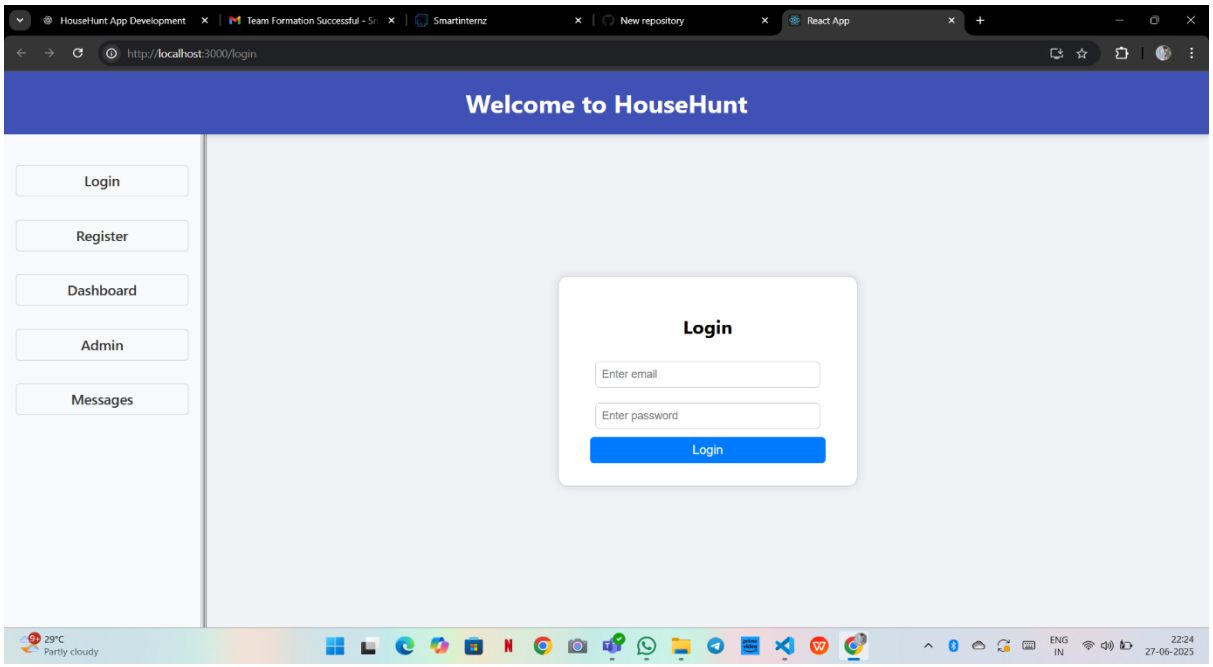
```
      receiver: property.owner,
      content,
    });

    await message.save();
    res.status(201).json(message);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Failed to send message' });
  }
});

// ✅ Owner views inbox
router.get('/inbox', auth, async (req, res) => {
  try {
    const messages = await Message.find({ receiver: req.user.id })
      .populate('sender', 'name email')
      .populate('property', 'type address');
    res.json(messages);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Failed to load inbox' });
  }
});

// ✅ Owner replies
```

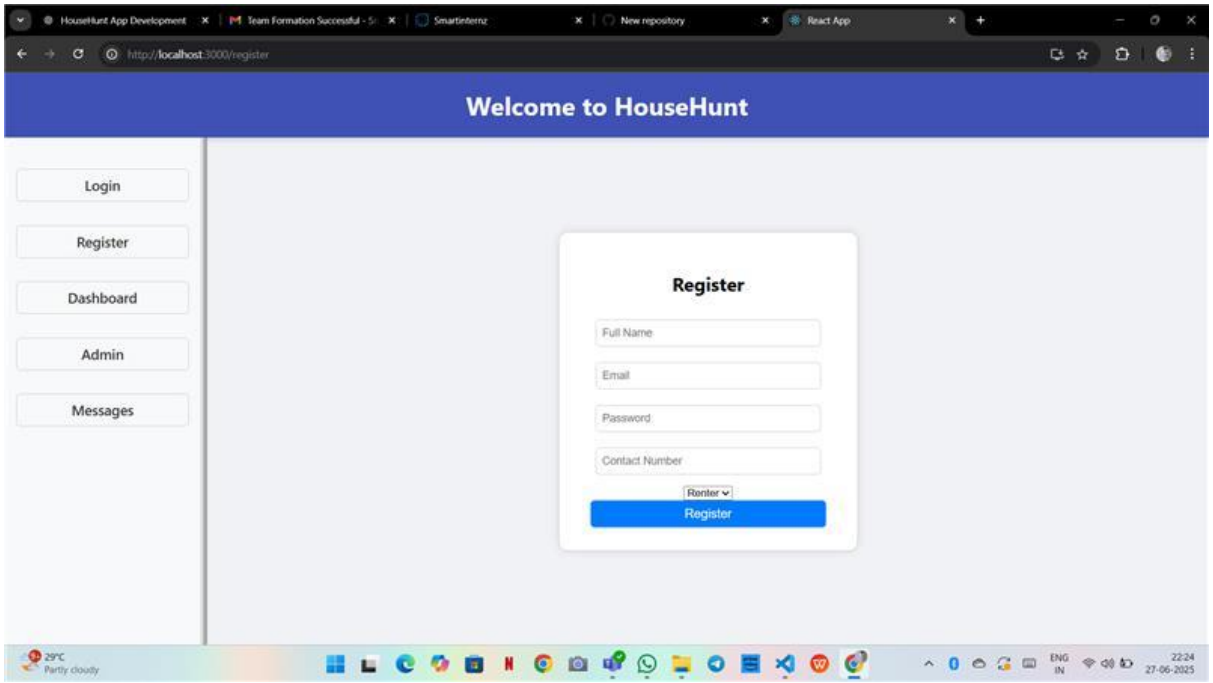
```
router.post('/reply/:id', auth, async (req, res) => {
  try {
    const message = await Message.findById(req.params.id);
    if (!message) return res.status(404).json({ message: 'Message not found' });

    message.reply = req.body.reply;
    await message.save();
    res.json({ message: 'Reply sent', data: message });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Failed to send reply' });
  }
});

// ✅ Renter sees their chat history per property
router.get('/thread/:propertyId', auth, async (req, res) => {
  try {
    const messages = await Message.find({
      property: req.params.propertyId,
      sender: req.user.id
    })
      .populate('receiver', 'name email')
      .sort({ createdAt: 1 });
    res.json(messages);
  } catch (err) {
    console.error(err);
```

```
    res.status(500).json({ message: 'Failed to load thread' });
  }
});

module.exports = router;



const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const connectDB = require('./config/db');
const bookingRoutes = require('./routes/bookingRoutes');

const authRoutes = require('./routes/authRoutes');
const propertyRoutes = require('./routes/propertyRoutes');
const messageRoutes = require('./routes/messageRoutes');
const adminRoutes = require('./routes/adminRoutes'); // ✅ make sure this import is below express()

const app = express(); // ✅ DEFINE app before using it
dotenv.config();
connectDB();

app.use(cors());
app.use(express.json());
app.use('/uploads', express.static('uploads'));
```

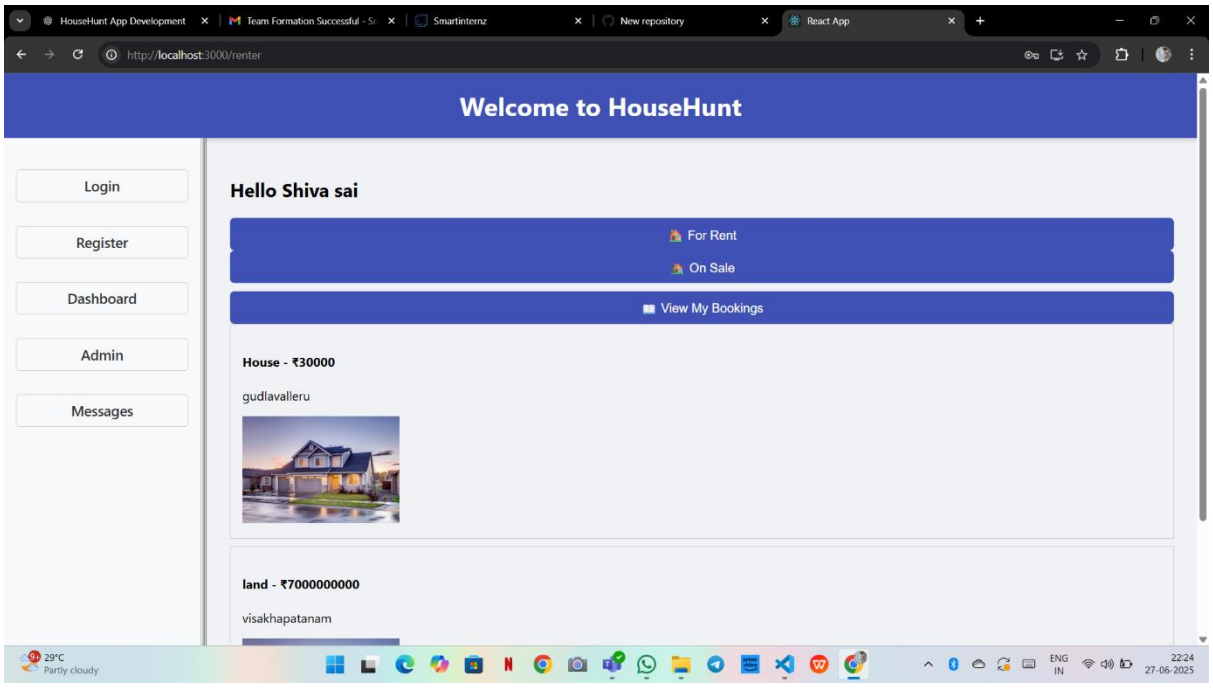
```
// ✅ All routes go after `app` is initialized
app.use('/api/auth', authRoutes);
app.use('/api/properties', propertyRoutes);
app.use('/api/messages', messageRoutes);
app.use('/api/admin', adminRoutes); // ✅ Moved here
app.use('/api/bookings', bookingRoutes);


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

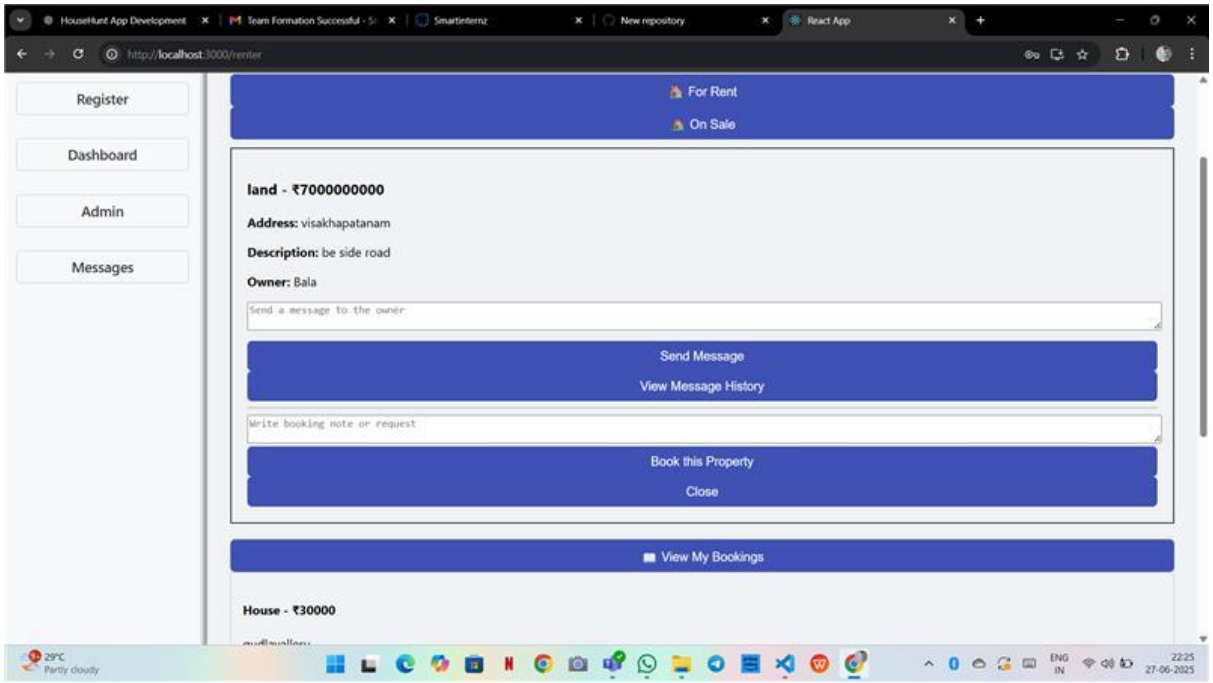
OUTPUT SCREENSHOTS:



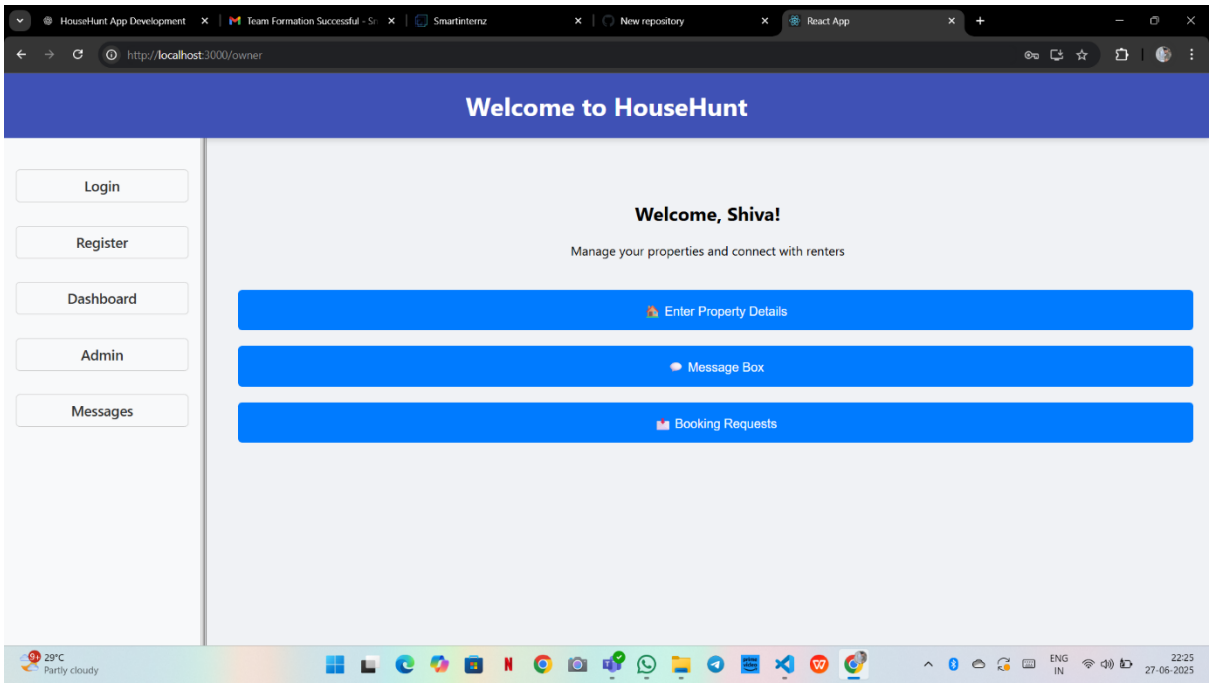


RENTER LOGIN PAGE:

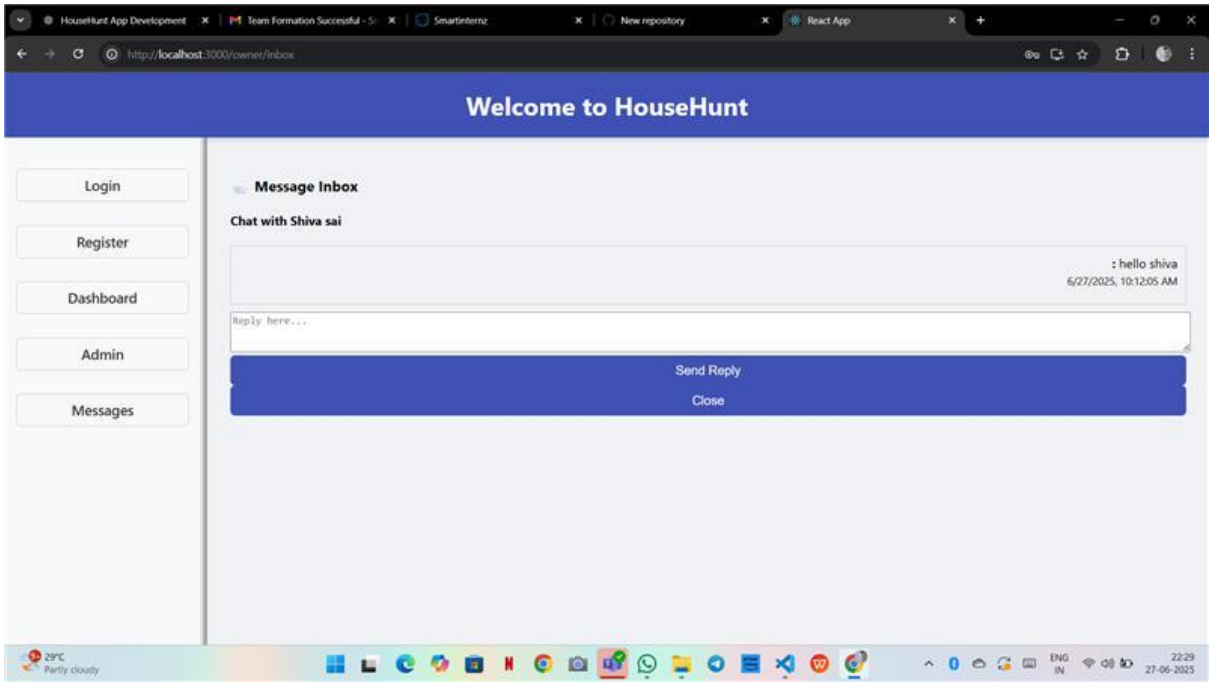




OWNER LOGIN PAGE:



OWNER MESSAGE:



Renter message history: