Harini M **22N219**
Pavithra E **22N236**
Sneha Rajesh **22N252**
Sri Sai Varshini B **22N255**
Pranaya V **22N263**

19N511 - APPLICATION DEVELOPMENT LABORATORY

RENTAL HOUSE MANAGEMENT SYSTEM

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING
(Artificial Intelligence and Machine Learning)

Of Anna University

NOVEMBER 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641004

PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)

COIMBATORE – 641 004

Bonafide record of work done by

Harini M	22N219
Pavithra E	22N236
Sneha Rajesh	22N252
Sri Sai Varshini B	22N255
Pranaya V	22N263

This is to certify that the project work titled "Rental House Management" is the original and bonafide work of the students listed, who carried out the project under my supervision. The project work has been submitted in partial fulfillment of the requirements for the Application Development Lab in the Department of Computer Science and Engineering at PSG College of Technology during the academic year 2024-25.

The project embodies the work completed by the students as part of their coursework and adheres to the standards of the institution.

…...……..… ……..…….

Internal Examiner External Examiner

The candidate was examined in the review held on 29/10/2024

TABLE OF CONTENTS

ABSTRACT

This project introduces a comprehensive house management system designed to enhance the connection between house owners and tenants. It features advanced search options with filters for location, number of bedrooms, budget, and move-in date, making it easier for users to find suitable properties. Central to the platform is an ML-based recommendation system, which provides personalized property suggestions guided by user preferences and activity. Additionally, secure authentication and seamless property uploads ensure users can efficiently manage and list properties. Altogether, these functionalities streamline the house-hunting process, aiming to reduce search time and improve the user experience for both owners and prospective tenants. This platform not only simplifies property searches but also fosters trust and convenience by providing reliable and secure interactions. With its user-centric approach, the system aims to redefine the housing search experience, making it faster, smarter, and more intuitive for everyone involved.

1 PROBLEM STATEMENT

1.1 INTRODUCTION

Often renters come face to face with the seriousness of the housing problem even in cases when the real estate is not fully occupied and every person in need of housing wants to rent housing. The ways we always used a few years back like newspaper property advertisements and property agents' charter were not of any help and they would take a lot of time. With the advancements of modern online real estate systems, the speed and efficiency of the house hunting process significantly improved. Despite the progress, the problem still remains in the fields like the incapability of filtering and lack of personalized recommendations that lead to incorrect tenant-property matches.

According to the latest figures, almost 10% of rental properties are empty each year and the average time to find a good home goes beyond three months. This kind of imbalance among the different roles of homeowners, and potential tenants or buyers creates empty spaces, as well as higher costs, frustration, and delays for the apartment seekers. The major objective of the project is to come up with an alternative that solves this and is more efficient and customer-oriented especially by narrowing the gap in the real estate market.

1.2 OBJECTIVES

- Develop a user-centric platform for house hunting.
- Improve filtering capabilities for property searches.
- Provide personalized recommendations based on user preferences.
- Reduce the average search time for tenants.
- Minimize vacancy rates in rental properties.
- Enhance communication between tenants and homeowners.
- Gather and analyze data to improve platform offering.

2 REQUIREMENT ANALYSIS

2.1 FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS	DESCRIPTION
User Authentication	The system offers smooth authentication of users by registering them in the database first and logging in later. The system also provides facilities for users to reset passwords and logging out.
Property search	Advanced search filters based on location, rating, and budget are offered by the system so that the user can find his/her preferred property in a short period of time.
Property listings	Owners can add new properties with description and images. For every property, the owner should add their house document which is authenticated using a deep learning model by the system.This ensures that the properties are credible and are not fake.
Recommendation engine	Personalized suggestions based on user preferences are displayed using machine learning models. The model uses the K-Nearest Neighbours algorithm to find the most suitable recommendation for a user.
Bridging communication	The system offers direct contact of house owners via email and phone number, thus bridging the gap between house owners and tenants.

2.2 NON-FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENTS	DESCRIPTION
Performance	The system should offer quick page loading within 3 ms for better user experience. The system should support concurrent users up to 50 users.
Scalability	The system should be able to accommodate a large number of users and data without a lot of changes.
Security	The system should perform user authentication using secure hashing algorithms. Provide data encryption to protect sensitive information.
Accessibility	The system should provide an intuitive and user-friendly interface for all users. Tenant and house owners must be able to seamlessly work with the platform without any inconvenience.
Reliability	The platform aims to achieve 99.9% uptime and must implement regular backups and disaster recovery.

3 TECHNOLOGY STACK

3.1 OVERVIEW

CATEGORIES	TECH STACK
Front End	React
Back End	Node JS, Express JS
Database	MongoDB
AI & ML	Scikit-Learn, PyTorch

FRONT END:

React, a prevalent JavaScript library and front-end development tool, is chosen for the construction of the application's looks.It is based on the component-oriented architecture that developers can generate a quite responsive and dynamic user interface, enabling users to easily navigate and filter their search results in real-time.The reason behind the use of the virtual DOM technique is to get the application to render more quickly, thus, having a high-quality smooth experience when a user triggers a web application to query complex datasets and search property listings.Its facility to manage the state of a component effectively is the propounded technology for personal recommendations and interactive maps, thus, the users get a responsive and engaging platform.

BACK END:

Node.js and Express.js are the platform’s backend. Node.js uses a model that is fully event-driven and has no blocking I/O operations.It means that the system can interact smoothly with several user requests at the same time, thus leading to the system being scalable and efficient.Express.js, in addition to being a web application framework for Node.js, makes the server-side development more understandable by allowing a developer a robust and flexible environment to build his or her own APIs, to manage the data of the users, and to interact with the database. This achieved real-time data processing and communication between the client and the server, which resulted in smooth operation and fast responses to user actions.

DATABASE:

MongoDB, a NoSQL database, is an ideal choice for this project due to its flexibility and scalability, which are essential for handling varied and unstructured property listings that may include both text and images. Unlike traditional relational databases, MongoDB allows for dynamic schema design, meaning it can easily accommodate different data formats and property attributes that may change over time. Additionally, MongoDB's horizontal scaling capabilities allow the system to manage large volumes of data efficiently, making it suitable for applications that anticipate high traffic and growing datasets.

AI & ML:

Artificial intelligence and machine learning play a significant role in improving the platform’s recommendation system. Scikit-Learn is employed for building and deploying machine learning models that analyze user behavior, preferences, and historical data to generate personalized property recommendations. By learning from user interactions, the system continuously improves its matching accuracy. PyTorch, a deep learning framework, is used for more complex tasks such as image recognition for property photos, natural language processing for analyzing property descriptions, and other AI-driven features. These tools enable the platform to offer highly relevant suggestions, reduce the time tenants spend searching, and ensure a better match between properties and tenants.

3.2 SYSTEM ARCHITECTURE

3.2.1 Activity Diagram

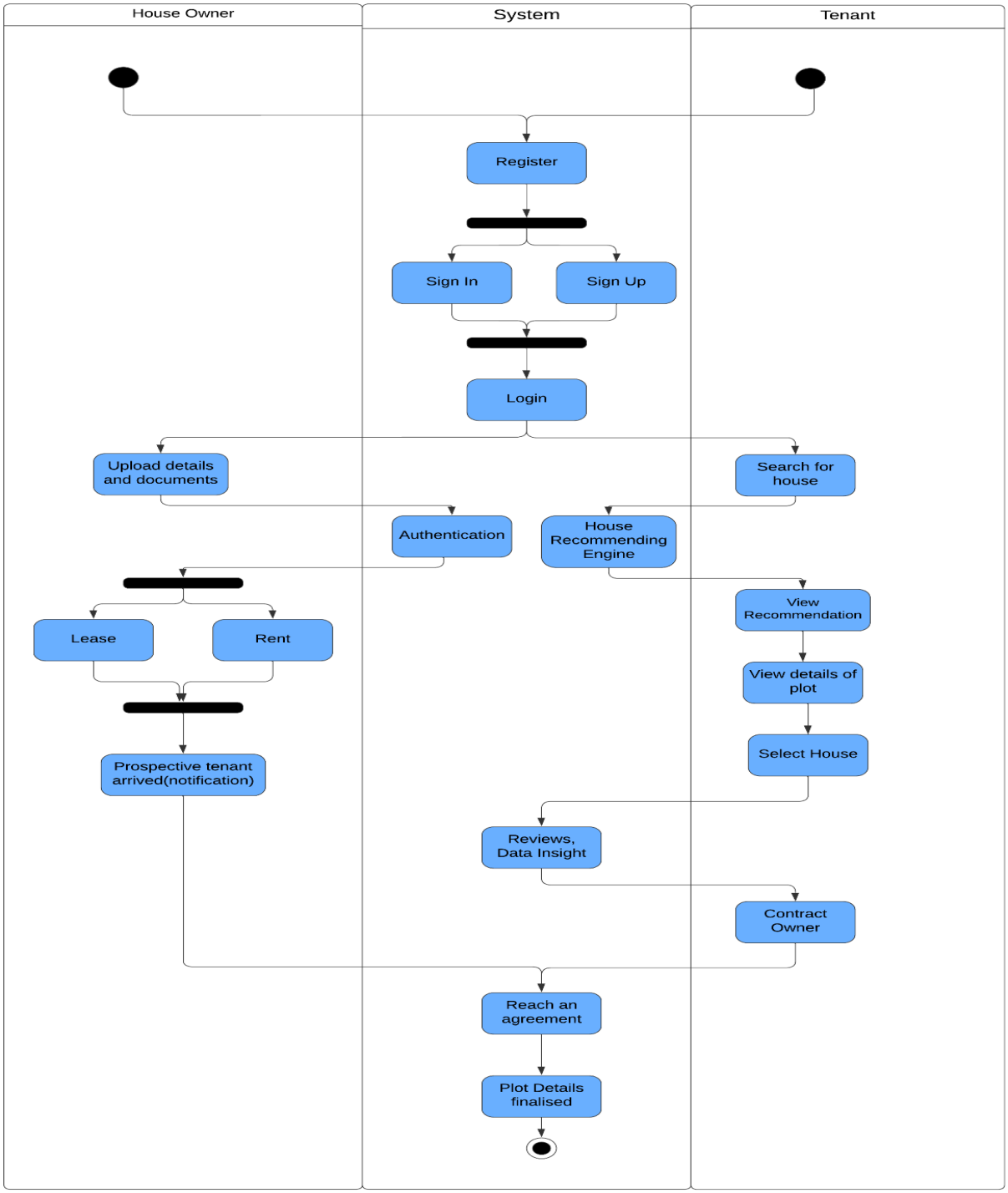

Fig.3.2.1 Activity Diagram

3.2.2 System Flow Diagram

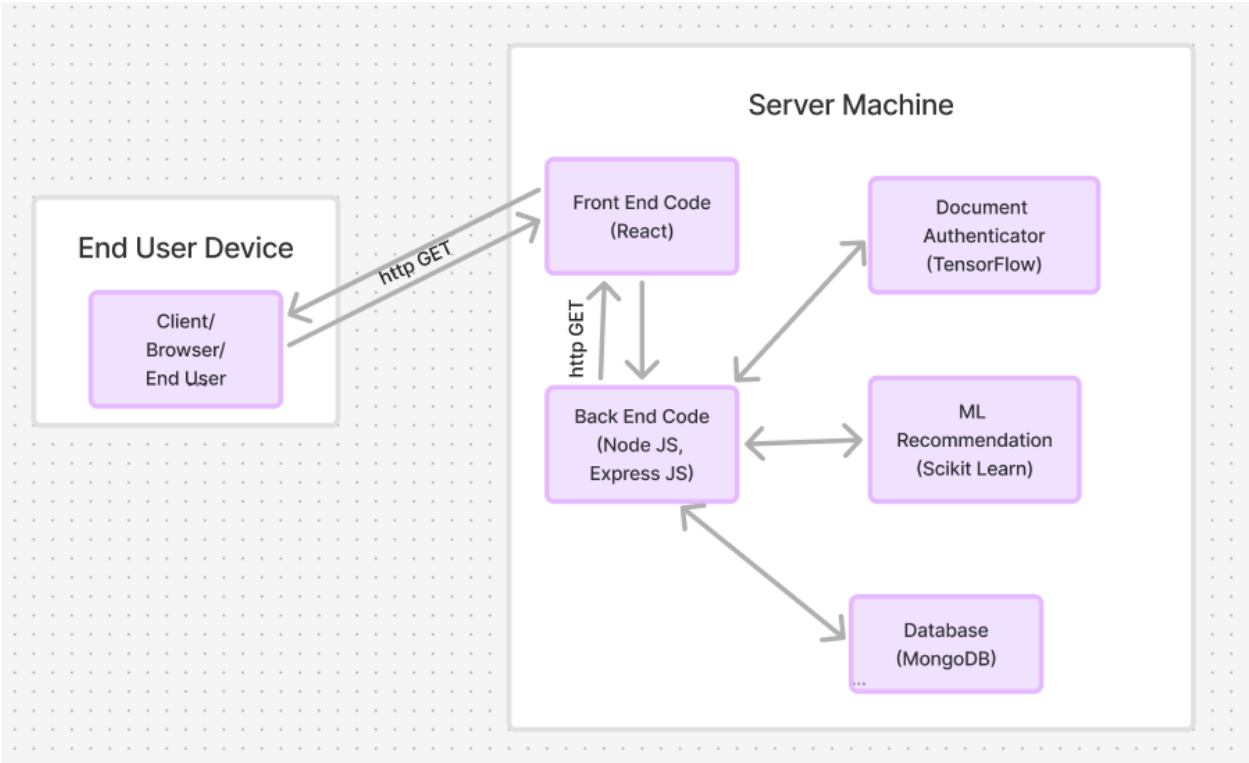

Fig.3.2.2 System Flow Diagram

4 MODULES

This section summarizes the key modules of the application, detailing their functionalities and interconnections, and emphasizing their contributions to the overall system.

4.1 User Authentication Module

The system encompasses user registration, operation of which involves new users the possibility to register granting by providing necessary information, together with secure login and logout functionalities to guarantee authorized access. The users are allowed to change their passwords whenever they require or they can reset them, and the role-based access is supported by the platform, i.e., allowing different access levels to tenants and property owners. The user credentials are collected through forms which are located on the frontend and validation is done at the backend where session management occurs for the secure authentication.

User profiles and passwords are stored in a database, which is encoded so that the data is safeguarded and the access to it is limited to rightful users. This type approach strengthens data security by not only infusing confidence in people of the platform but also making sure the privacy is protected through efficient data protection.

4.2 Property Listings and Search Module

The module provides landowners with full control over their property listings, enabling them to add, edit, or delete details through both image and text inputs. Occupants, meanwhile, can conduct advanced searches based on variables such as location, price, and lease duration. On the frontend, users are presented with a clean and intuitive search interface, where they can view search results and existing listings, dynamically generated from backend property data and search queries.

4.3 Property Upload Module

This module enables property owners to seamlessly list their properties on the platform, with all information stored securely in a cloud database. Users can easily input essential details such as property descriptions, pricing, location, and key features, alongside uploading high-quality images. Once submitted, all data is automatically stored in the cloud database, allowing for efficient retrieval and management. This cloud-based approach ensures scalability, accessibility, and data security, enabling property listings to be easily updated or modified as needed. Here, authentication of house documents is integrated enabling verification of each property.

4.4 Recommendation and Insights Module

The system offers personalized property recommendations by leveraging algorithms that align with each user's unique preferences and behavior. It continuously analyzes user interactions, refining and enriching suggestions based on real-time data insights. The frontend displays tailored property recommendations and insights, while the backend processes user data and applies a machine learning model to enhance recommendation accuracy. Ultimately, this feature makes browsing more engaging and personalized, presenting users with properties that best match their search history, interests, and tastes.

4.5 Authentication of Documents Module

This module utilizes deep learning algorithms to ensure the legitimacy and security of all uploaded property-related documents on the platform. By employing advanced techniques such as convolutional neural networks (CNNs) for image analysis and natural language processing (NLP) for text extraction, the module effectively verifies the authenticity of documents submitted by property owners, including ownership records, lease agreements, and property certifications. This deep learning approach allows the system to detect anomalies, inconsistencies, or signs of forgery, enhancing user trust and reducing the risk of fraud. By leveraging deep learning, this feature significantly improves the accuracy and efficiency of document authentication, contributing to a more reliable property management experience for all users.

5 KEY CODE SEGMENTS

This section highlights significant code snippets that demonstrate the core functionalities of the application, explaining their roles in achieving the project objectives.

5.1 TenantHome Page:

Fetches property listings and displays them alongside filter options for tenants to customize their search.

```
// Fetch properties from the API
useEffect(() => {
  const fetchProperties = async () => {
    try {
      const response = await fetch('http://localhost:5000/api/properties');
      const data = await response.json();
      setProperties(data);
      setFilteredProperties(data);
    } catch (error) {
      console.error('Error fetching properties:', error);
    }
  };
  fetchProperties();
}, []); // Run once when the component mounts
```

Fig 5.1 Tenant Home Page

5.2 House Recommendation Model(Functions):

This code defines a function to recommend houses based on user preferences by normalizing the input, finding nearest neighbors using a KNN model, and retrieving serialized property data from MongoDB.

```
def serialize_property(property_data):
    if property_data is None:
        return None
    property_data['_id'] = str(property_data['_id'])
    return property_data
def recommend_houses(user_preferences):
    user_normalized = scaler.transform([user_preferences]) # Normalize user preferences
    distances, indices = model.kneighbors(user_normalized) # Find nearest neighbors
    # Fetch recommended properties from MongoDB using their indices
    recommended_properties = []
    for index in indices[0]:
        property_data = properties_collection.find_one({"id": int(index)})
        if property_data:
            recommended_properties.append(serialize_property(property_data))
    return recommended_properties
```

Fig 5.2 House Recommendation Model(Functions)

5.3 Backend request to Recommendation Model:

This code defines a Flask route that processes user input for house recommendations and returns suitable properties as a JSON response.

```python
@app.route('/recommend', methods=['POST'])
def recommend():
    user_data = request.json # Extract user input from the request
    user_input = np.array([
        user_data['price'],                    # Price of the house
        user_data['size'],                     # Size of the house
        user_data['rooms'],                    # Number of bedrooms
        user_data['bathrooms']                 # Number of bathrooms
    ])
    recommended_properties = recommend_houses(user_input)
    response = {
        'recommended_properties': recommended_properties   # Return the properties
    }
    return jsonify(response)
```

Fig 5.3 Backend request to Recommendation Model

5.4 API endpoint for Register

Allows new users to create an account by providing their details, which are then saved to the database.

```javascript
// User registration route
app.post('/api/auth/register', async (req, res) => {
  const { name, email, password, role } = req.body;
  try {
    // Check if user already exists
    const userExists = await User.findOne({ email });
    if (userExists) {
      return res.status(400).json({ message: 'User already exists' });
    }
    // Hash password and create new user
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({ name, email, password: hashedPassword, role });
    // Save user to the database
    await user.save();
    res.status(201).json({ message: 'User registered', role });
  } catch (error) {
    res.status(500).json({ message: 'Server error', error });
  }
});
```

Fig 5.4 API endpoint for Register

5.5 API endpoint for Reset Password

Verifies the reset token, updates the user's password, and clears the reset token data once the password is changed.

```
// Reset password route
app.post('/api/auth/reset-password', async (req, res) => {
  const { token, password } = req.body;
  try {
    // Find user by reset token and check expiration
    const user = await User.findOne({
      resetToken: token,
      resetTokenExpiration: { $gt: Date.now() },
    });
    if (!user) {
      return res.status(400).json({ success: false, message: 'Invalid or expired token.' });
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    user.password = hashedPassword;
    user.resetToken = undefined;
    user.resetTokenExpiration = undefined;
    await user.save();
    res.json({ success: true, message: 'Password has been reset successfully.' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'An error occurred. Please try again later.' });
  }
});
```

Fig 5.5 API endpoint for Rest Password

5.6 RDCN(Residual Deformable Convolutional Network) Model

This code defines a CNN model for document fraud detection. It classifies documents as authentic or fraudulent based on binary cross-entropy loss.

```
# Define the RDCN model
def create_rdcn_model(input_shape):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Fig 5.6 RDCN Model

6 RESULTS AND DISCUSSIONS

6.1 OUTPUT SCREENSHOTS

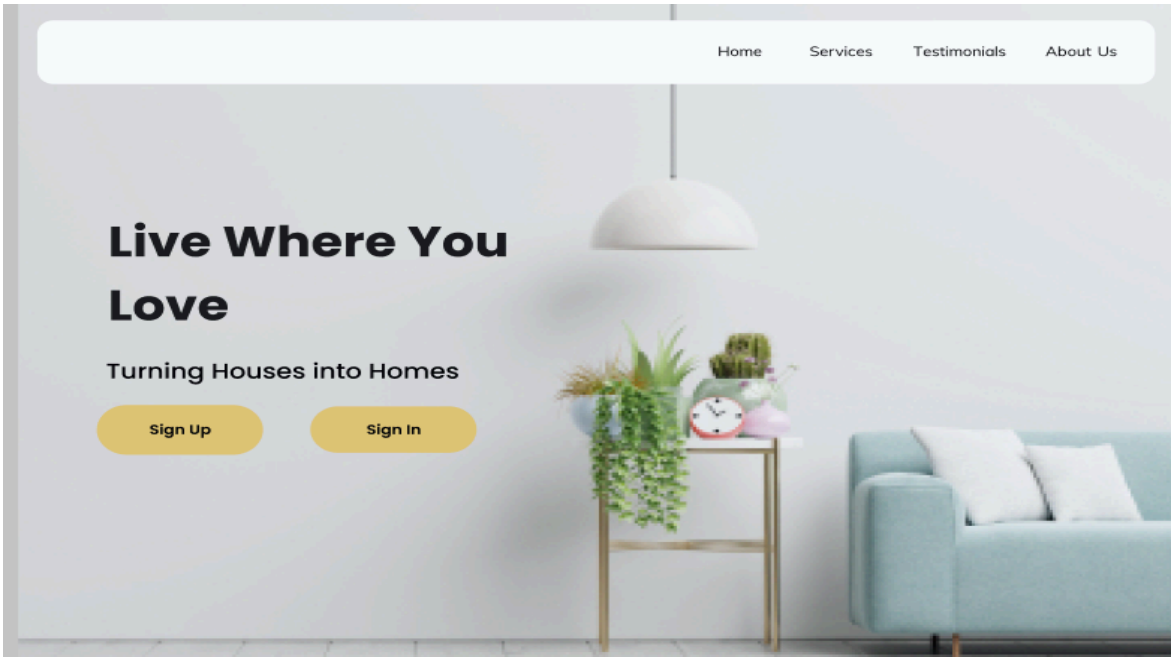

Fig.6.1.1 Home Page 1

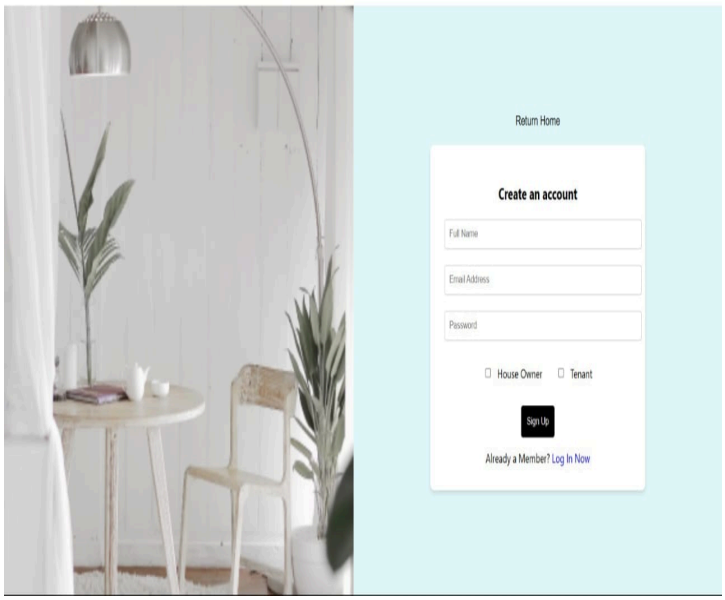

Fig.6.1.2 Sign Up

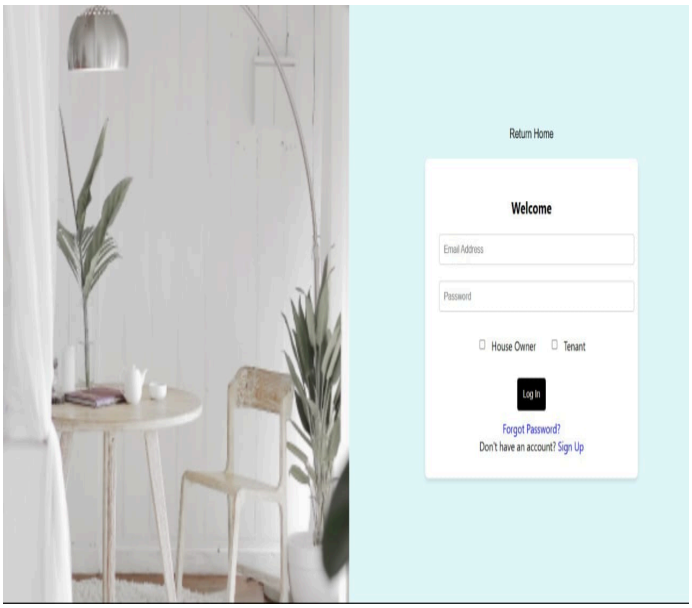

Fig.6.1.3 Log In

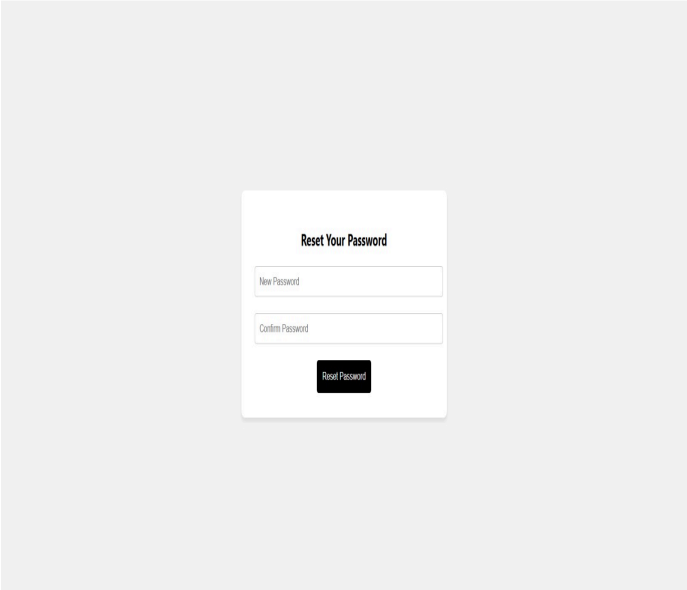

Fig.6.1.4 Reset Password Page

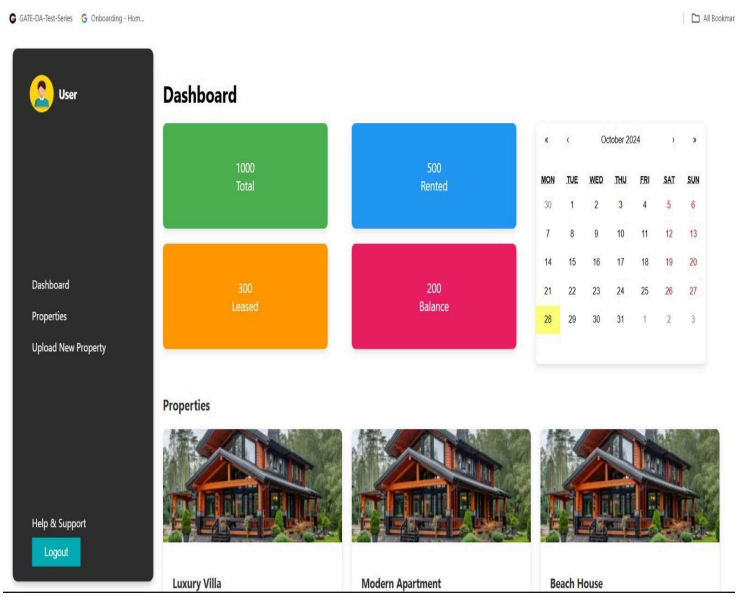

Fig 6.1.5 Dashboard for house owners

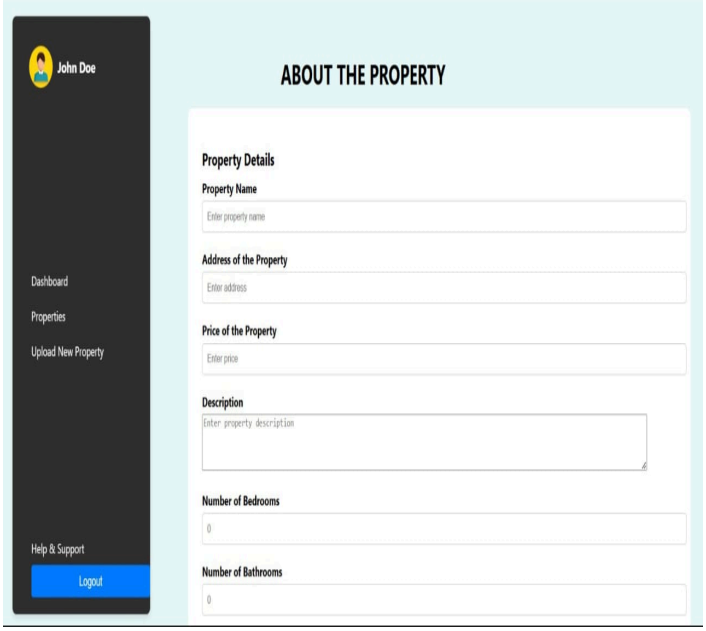

Fig.6.1.6 Property Upload 1

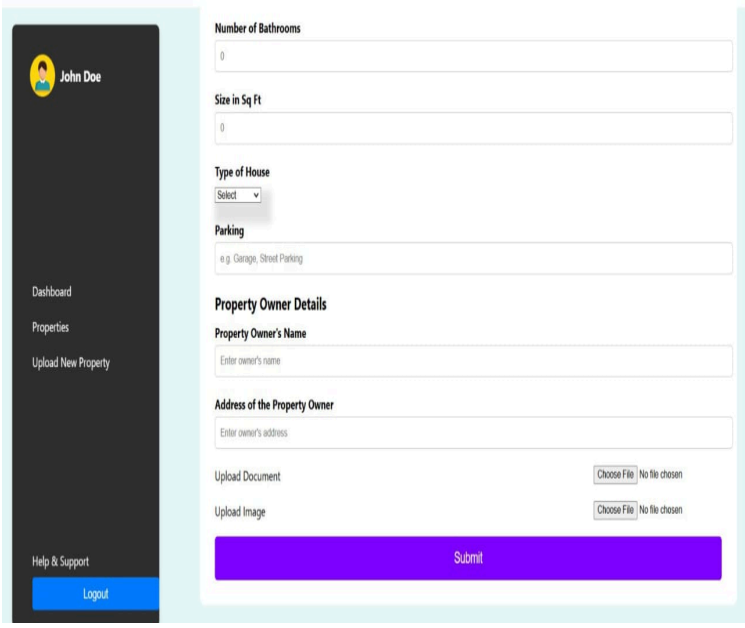

Fig 6.1.7 Property Upload 2

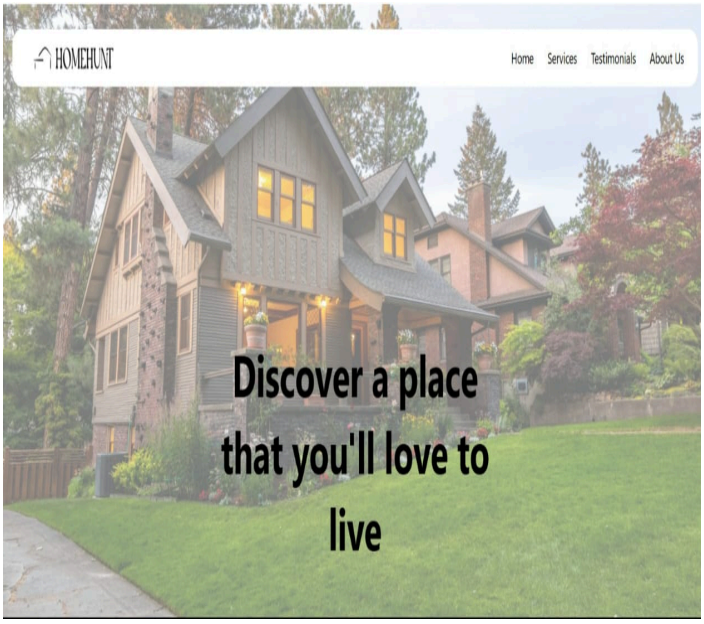

Fig 6.1.8 Testimonials Page 1

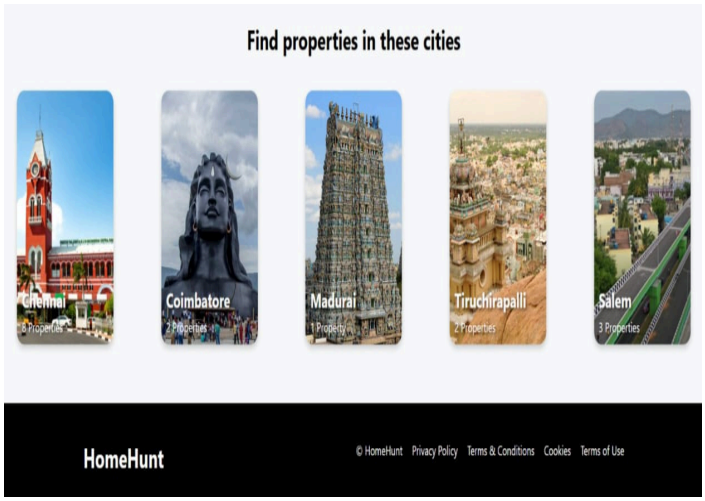

Fig 6.1.9 Testimonials Page 2

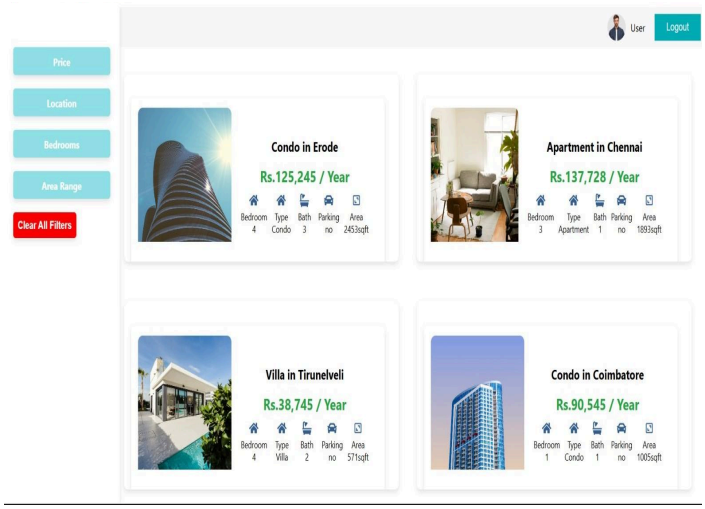

Fig 6.1.10 Tenant Homepage

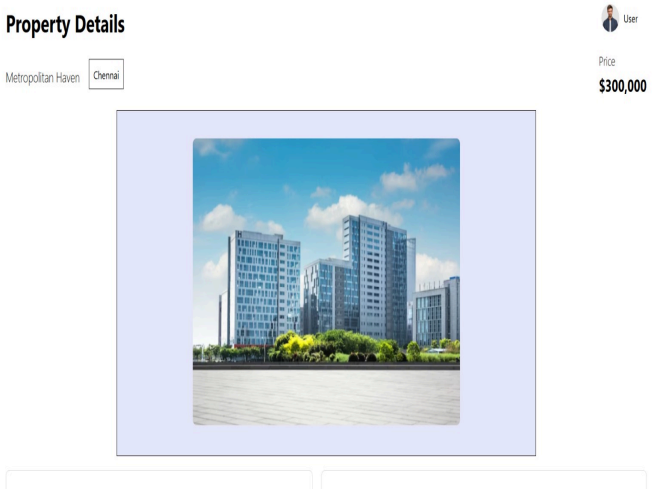

Fig 6.1.11 Tenant Homepage

6.2 PERFORMANCE METRICS

1. **Network Time**: The time taken to transfer data between the client and server.
2. **Response Time**: Total time taken for the server to process and respond to a request.
3. **TTFB (Time to First Byte)**: The time from the initial request to the moment the first byte of data is received.
4. **Response Code**: HTTP status code indicating the outcome of the request (e.g., 200 OK means success).
5. **Latency**: The delay from the moment a request is sent to the moment a response is received, including network and processing times.

Host	Network Time	Response Time	TTFB	Response Code	Latency
http://localhost:3000 (Frontend)	10 ms	10 ms	3.92 ms	200 OK	10
http://localhost:5000 (Backend)	13 ms	13 ms	6.64 ms	200 OK	13

Proof for performance metrics:

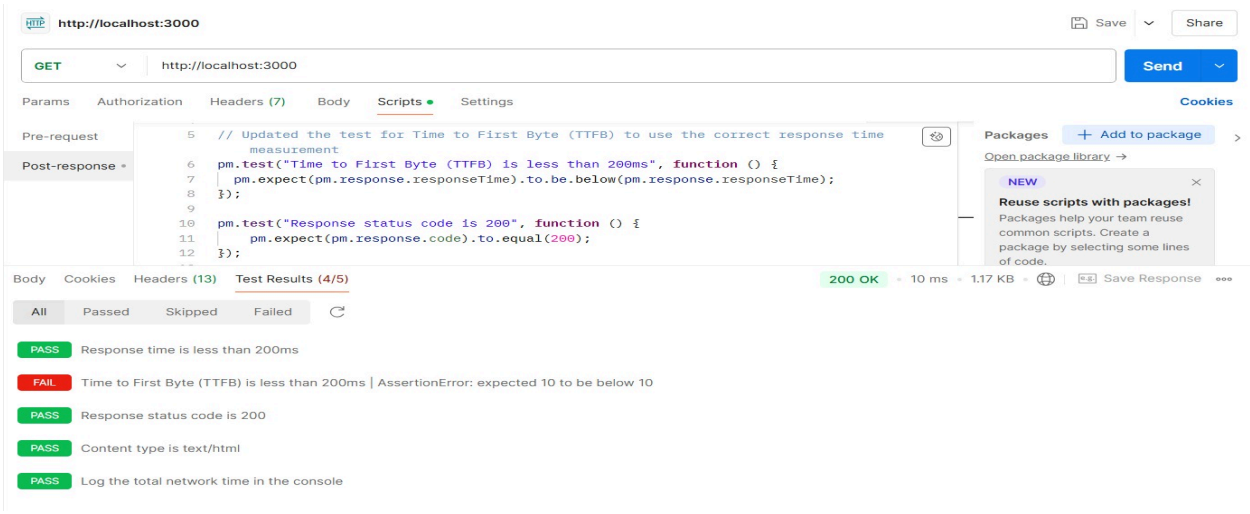

Fig 6.2.1 Frontend Performance Metric Screenshot

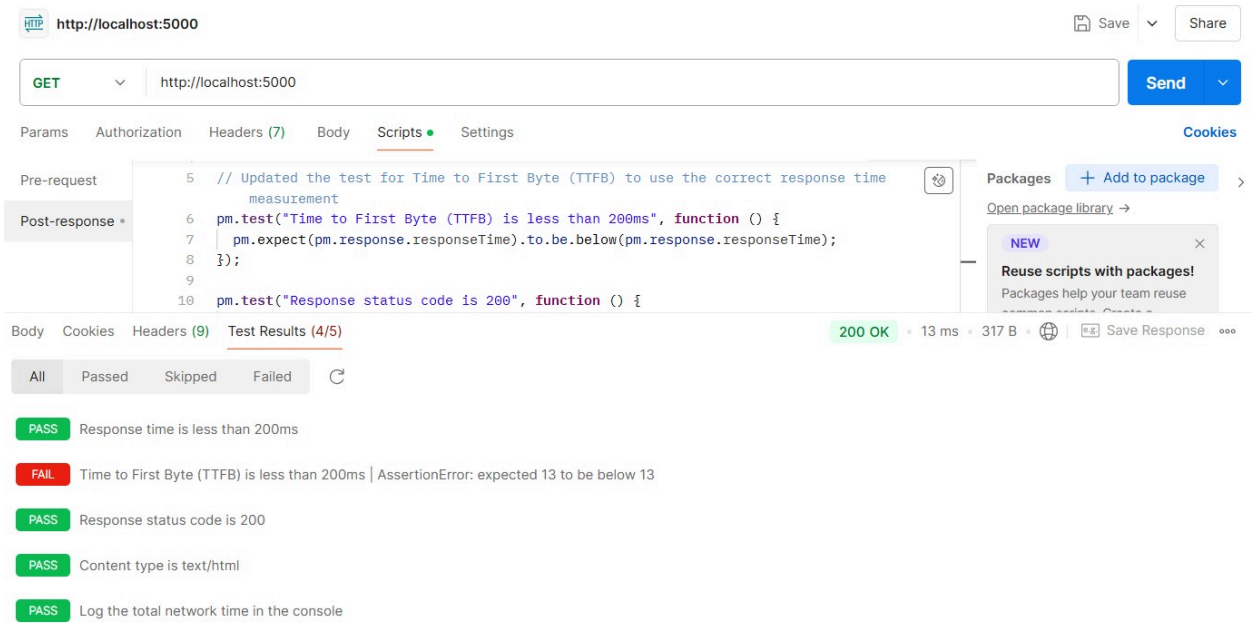

Fig 6.2.2 Backend Performance Metric Screenshot

CHALLENGES FACED

1. **Efficient Property Listing and Matching:**
   The system faced significant challenges with advanced search, filtering and personalized recommendations, which directly impacted the user experience for tenants.
2. **Authentication of House Documents:**
   Due to the lack of a real-time dataset for classifying documents as genuine or fake, the deep learning model could not be integrated into the system, impacting automated verification capabilities.

FUTURE WORKS

1. **Search functionality:**
   Future work includes implementing a search bar with automatic suggestions to enhance user experience, allowing for quicker and more accurate property searches.
2. **Authentication of House Documents:**
   The goal is to build a real-time dataset and integrate a deep learning model to classify documents as authentic or fake, thereby enhancing automated verification.

CONCLUSION

The project successfully created a tenant-property matching system that is user-friendly and much more efficient than the previous approach. React was utilized for frontend development, contributing to a smooth and responsive user interface, while the integration of machine learning models like Scikit-Learn and PyTorch improved the accuracy of property recommendations. These features have led to increased user satisfaction and reduced search time. However, challenges were encountered in achieving the required scalability to manage high data volumes and heavy user activity during peak hours. Proprietary learning systems were built to support frequent upgrades, ensuring recommendations stay accurate as user behavior evolves. Looking forward, the project aims to enhance the platform by further optimizing machine learning models, refining filters and location-based services, and introducing a more powerful infrastructure to address scalability and traffic challenges effectively.

REFERENCES

[1] React Documentation
Available at: https://reactjs.org/docs/getting-started.html

[2] "A Survey on Machine Learning Algorithms for Smart Real Estate Applications"
This paper explores the use of machine learning in real estate, discussing the application of recommendation systems, predictive modeling, and personalized matching.
Available at: https://ieeexplore.ieee.org/document/9125070

[3] "Recommender Systems: An Overview by Aggarwal, C. C. (2016)."
This paper provides a comprehensive overview of recommender systems, including collaborative filtering and content-based methods, which are key to developing personalized property recommendations.
DOI: 10.1007/978-3-319-29659-3_1