Find Your Dream Home
EstateEdge
Project Report

By

Harsh Parab – AF04957496
Vikas Vishwakarma – AF04957501

ABSTRACT

Abstract – EstateEdge

EstateEdge is a comprehensive full-stack real estate management platform that bridges the gap between buyers, sellers, and administrators in the digital property market. Developed with Node.js, Express, MySQL, and Bootstrap 5, the platform offers an intuitive and responsive interface for property browsing, searching, and filtering.

For buyers, EstateEdge provides easy access to detailed property listings with images, pricing, and availability status. Sellers can manage their properties through full CRUD operations (create, read, update, delete) and track the status of their listings (available, pending, sold). Administrators benefit from a dedicated dashboard featuring user management, property oversight, transaction monitoring, and payment tracking.

Security is ensured through JWT-based authentication with password encryption and role-based access control, enabling personalized experiences for each type of user. By combining modern web technologies with scalable architecture, EstateEdge delivers a seamless solution for streamlining real estate operations and enhancing user trust in the property marketplace.

ACKNOWLEDGEMENT

Acknowledgement-EstateEdge

Acknowledgement The project “Online News Portal with Sentiment Analysis” is the Project work carried out by

Name	Enrollment No
Harsh Parab	**AF04957496**
Vikas Vishwakarma	**AF04957501**

We would like to express our deepest gratitude to all those who contributed to the successful development of *EstateEdge*. This project would not have been possible without the continuous support, guidance, and encouragement we received throughout its journey.

First and foremost, we extend our sincere thanks to our mentors and faculty members for their valuable insights, constructive feedback, and motivation, which helped shape this project into a comprehensive real estate management platform.

We are also grateful to our peers and colleagues who provided constant encouragement, shared ideas, and assisted in testing the application to ensure its effectiveness.

Finally, we acknowledge the open-source community and the developers of the frameworks, libraries, and tools such as Node.js, Express, MySQL, and Bootstrap, which served as the backbone of this project. Their contributions to the developer ecosystem enabled us to build a robust, scalable, and user-friendly solution.

This accomplishment is the result of collective effort, and we sincerely thank everyone who played a role in making *EstateEdge* a reality.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

1.1 Background

The real estate industry has experienced rapid digital transformation in recent years, with increasing demand for online platforms that simplify property discovery, buying, selling, and management. Traditional methods of real estate transactions often involve manual processes, limited accessibility, and lack of transparency, which can cause delays and inefficiencies. To address these challenges, modern web technologies are being leveraged to create platforms that connect buyers, sellers, and administrators in a seamless and efficient way.

EstateEdge is designed as a full-stack solution to bridge this gap by offering an intuitive, secure, and scalable real estate management platform. With the integration of Node.js, Express, MySQL, and Bootstrap, EstateEdge provides a responsive and user-friendly environment for property transactions.

1.2 Objectives

The main objectives of the EstateEdge project are:

- To develop a digital platform that simplifies real estate management for buyers, sellers, and administrators.
- To provide a secure and role-based authentication system for different types of users.
- To enable property listings with search, filter, and CRUD (Create, Read, Update, Delete) functionalities.
- To design an administrative dashboard for monitoring users, properties, sales, and payments.
- To ensure scalability, responsiveness, and reliability through modern full-stack development practices.

1.3 Purpose & Scope

The purpose of EstateEdge is to **digitize and streamline the real estate transaction process** by creating a platform that is accessible, secure, and efficient. It is aimed at reducing manual efforts, enhancing transparency, and providing a structured environment for property-related operations.

The scope of the project includes:

- **For Buyers:** Easy property search, filtering, and access to detailed property information including images, pricing, and status.
- **For Sellers:** A streamlined interface to add, update, or manage property listings and track their availability status.
- **For Administrators:** A comprehensive dashboard for user management, property oversight, transaction monitoring, and payment tracking.
- **Technology Scope:** Implementation of a responsive web application using **Node.js, Express, MySQL, JWT authentication, and Bootstrap** for frontend design.

By combining these features, EstateEdge aims to serve as a **complete real estate management ecosystem**, adaptable to future enhancements such as integration of online payments, AI-driven property recommendations, and advanced analytics.

Chapter 2

SYSTEM ANALYSIS

2.1 Existing System

The traditional real estate process is largely manual, involving physical visits to properties, paper-based documentation, and reliance on intermediaries such as brokers or agents. This leads to:

- Lack of **transparency** in transactions.
- **Time-consuming** processes for property discovery and verification.
- Difficulty in **managing large numbers of listings** efficiently.
- Limited **user accessibility**, especially for remote buyers and sellers.
- No unified platform for administrators to monitor and regulate property transactions.

Although some online portals exist, many are either region-specific or lack comprehensive features like role-based access, property status tracking, or integrated dashboards for administration.

2.2 Proposed System

EstateEdge addresses these challenges by offering a **modern, centralized, and secure platform** for managing real estate operations. The system introduces:

- **Digital Property Listings:** Buyers can search, filter, and view properties with detailed information and images.
- **Multi-role Authentication:** Buyers, sellers, and administrators have distinct roles with specific access privileges.
- **Property Management:** Sellers can add, update, or delete properties and track their availability status.

- **Administrative Dashboard:** Admins can oversee users, properties, sales, and payments in real-time.
- **Secure Transactions:** JWT authentication, password hashing, and role-based access control ensure data integrity and security.

This proposed system not only digitizes the workflow but also enhances efficiency, accessibility, and trust in property transactions.

2.3 Requirement Analysis

To successfully develop and deploy EstateEdge, both **functional** and **non-functional** requirements are considered:

Functional Requirements

- User registration and login with secure authentication.
- Role-based access control for buyers, sellers, and administrators.
- Property management with CRUD operations.
- Search and filter options for property listings.
- Admin dashboard for monitoring sales, payments, and user activities.

Non-Functional Requirements

- **Security:** JWT authentication, password encryption.
- **Performance:** Fast search and property retrieval using optimized queries.
- **Scalability:** Capable of handling growing user base and property listings.
- **Usability:** Responsive design for accessibility across devices.

2.4 Hardware Requirements

- **Processor:** Intel i3 or higher (or equivalent AMD).
- **RAM:** Minimum 4 GB (8 GB recommended for development).
- **Storage:** 20 GB free disk space.
- **Display:** 1366x768 resolution or higher.
- **Network:** Stable internet connection for database and server communication.

2.5 Software Requirements

- **Operating System:** Windows 10 / Linux (Ubuntu) / macOS.
- **Backend:** Node.js with Express framework.
- **Database:** MySQL.
- **Frontend:** HTML5, CSS3, Bootstrap 5, JavaScript.
- **Authentication:** JWT (JSON Web Tokens).
- **Development Tools:** VS Code, Git, Postman for API testing.
- **Package Management:** npm or pnpm.
- **Server Environment:** Vite for frontend build and Node.js runtime.

2.6 Justification of Platform

The chosen **hardware and software platforms** for EstateEdge are justified as follows:

- **Hardware Suitability:** Moderate hardware specifications (Intel i3, 4 GB RAM) are sufficient to run Node.js, MySQL, and Bootstrap-based applications, making the project feasible on standard development and deployment machines.
- **Software Suitability:**
    - **Node.js & Express**: Efficient for building scalable and lightweight server-side applications.
    - **MySQL**: Reliable relational database system for managing structured property and user data.
    - **Bootstrap 5**: Ensures responsive design across different devices, enhancing usability.
    - **JWT Authentication**: Provides secure and scalable authentication suitable for multi-role systems.
    - **Vite & npm/pnpm**: Modern build tools ensure faster development and deployment cycles.

Together, this combination of hardware and software ensures that *EstateEdge* is **secure, efficient, scalable, and user-friendly**, aligning with the project objectives.

Chapter 3

SYSTEM DESIGN

System Design is a crucial phase that translates the gathered requirements into a blueprint for constructing the actual system. In this chapter, the overall architecture, modular breakdown, data structure, and process flows of the Estateedge Webpage are illustrated through various models and descriptions.

3.1 Module Division

The *EstateEdge* system is divided into the following key modules:

1. **User Management Module**
    - Handles user registration, login, authentication, and role-based access (Buyer, Seller, Admin).
2. **Property Management Module**
    - Manages CRUD operations for property listings.
    - Includes property details such as title, price, images, location, and status (available, pending, sold).
3. **Search & Filter Module**
    - Allows buyers to browse properties with search and filter functionalities based on location, price, and status.

4. **Admin Dashboard Module**
    - Provides statistics and control over properties, users, sales, and payments.
5. **Transaction & Payment Tracking Module**
    - Records property sales and maintains payment details for admin monitoring.
6. **Security Module**
    - Implements JWT authentication, password hashing, and access control.

3.2 Data Dictionary

Field Name	Data Type	Description
User_ID	INT (PK)	Unique identifier for each user
Username	VARCHAR(50)	Name of the user
Password	VARCHAR(255)	Encrypted password for authentication
Role	ENUM	Role of the user (Buyer/Seller/Admin)
Property_ID	INT (PK)	Unique identifier for each property

Field Name	Data Type	Description
Title	VARCHAR(100)	Property title
Description	TEXT	Property details
Price	DECIMAL(10,2)	Property price
Status	ENUM	Available, Pending, Sold
Image_URL	VARCHAR(255)	Path to property image
Sale_ID	INT (PK)	Unique identifier for each transaction
Buyer_ID	INT (FK)	Reference to Buyer (User_ID)
Seller_ID	INT (FK)	Reference to Seller (User_ID)
Payment_Status	ENUM	Pending, Completed

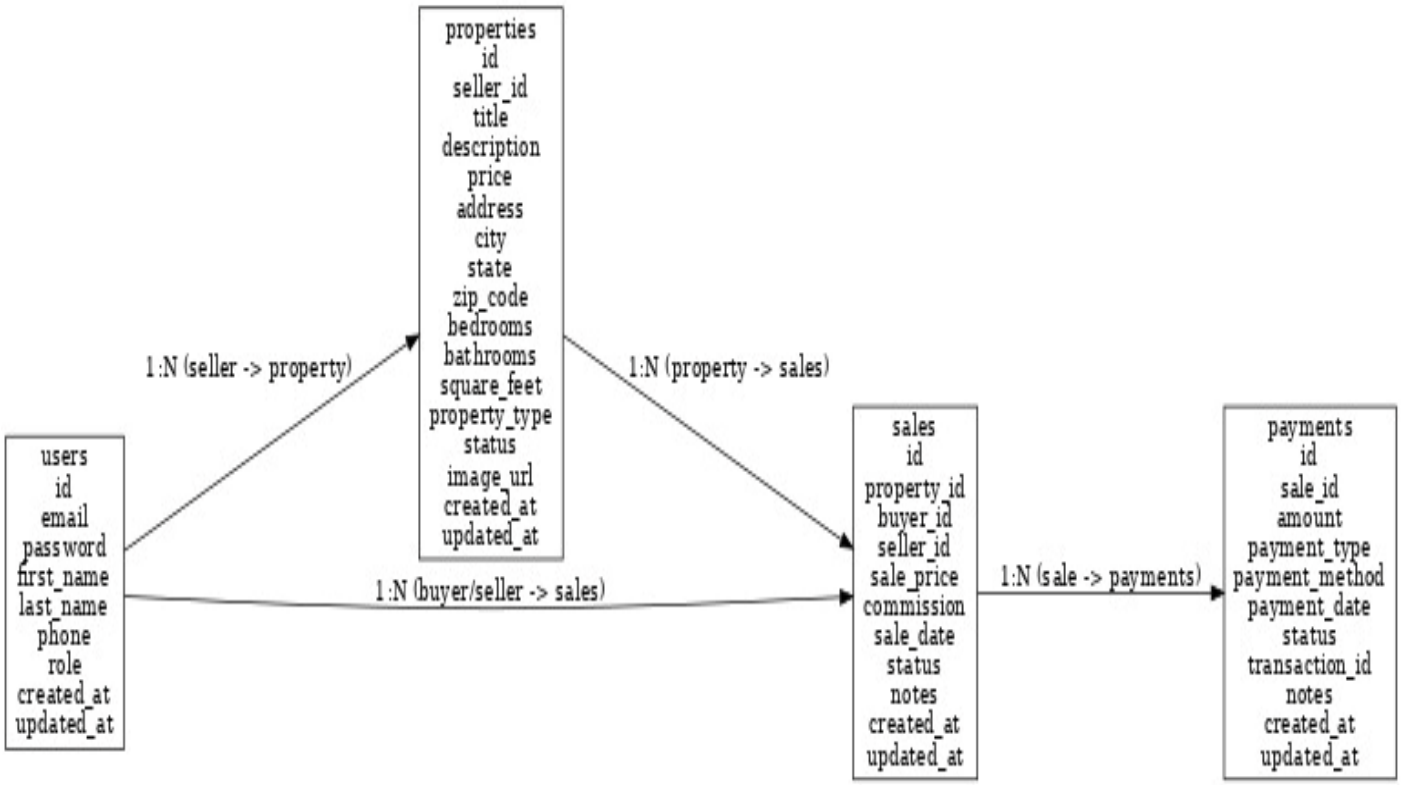
3.3 E-R Diagram

Entities:

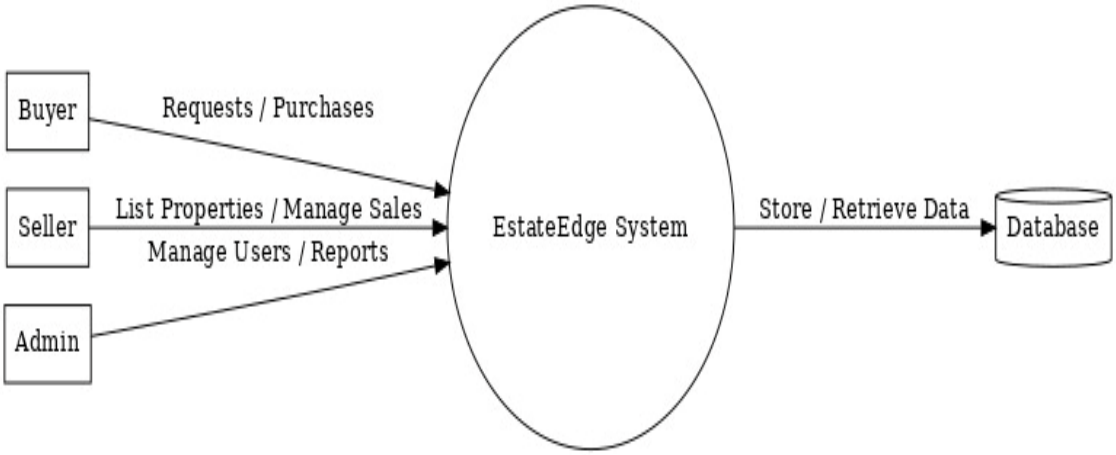
- **User** (User_ID, Username, Password, Role)
- **Property** (Property_ID, Title, Description, Price, Status, Image_URL, Seller_ID)
- **Sale** (Sale_ID, Buyer_ID, Property_ID, Payment_Status, Date)

Relationships:

- A **User (Seller)** can list multiple **Properties**.
- A **User (Buyer)** can purchase multiple **Properties**.
- A **Sale** is linked to a Buyer, Seller, and Property.



3.4 Data Flow Diagrams (DFD) / UML

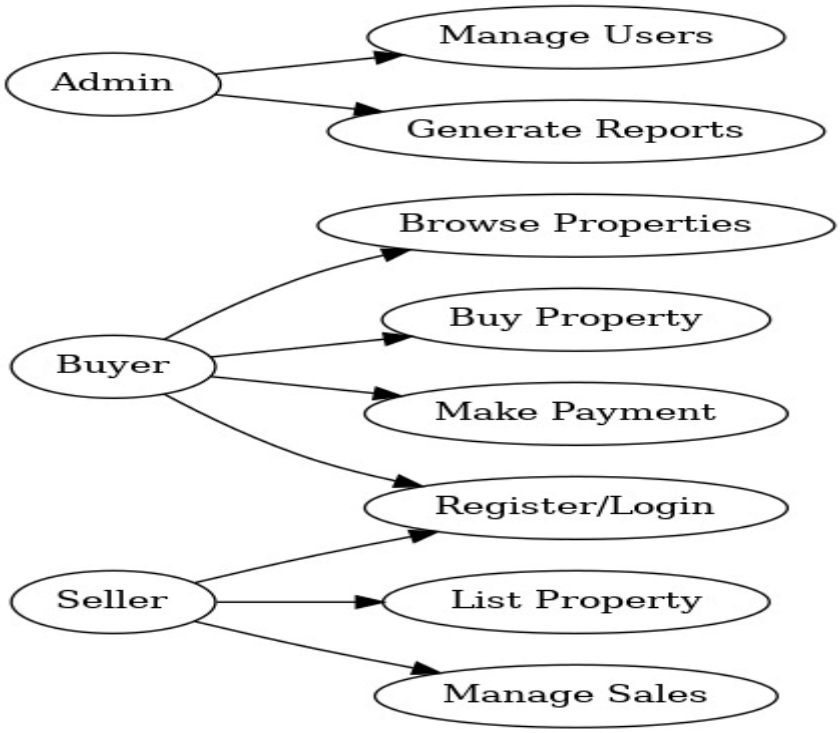


3.5 UML Use Case Diagram

Actors: Buyer, Seller, Admin.

Use Cases:

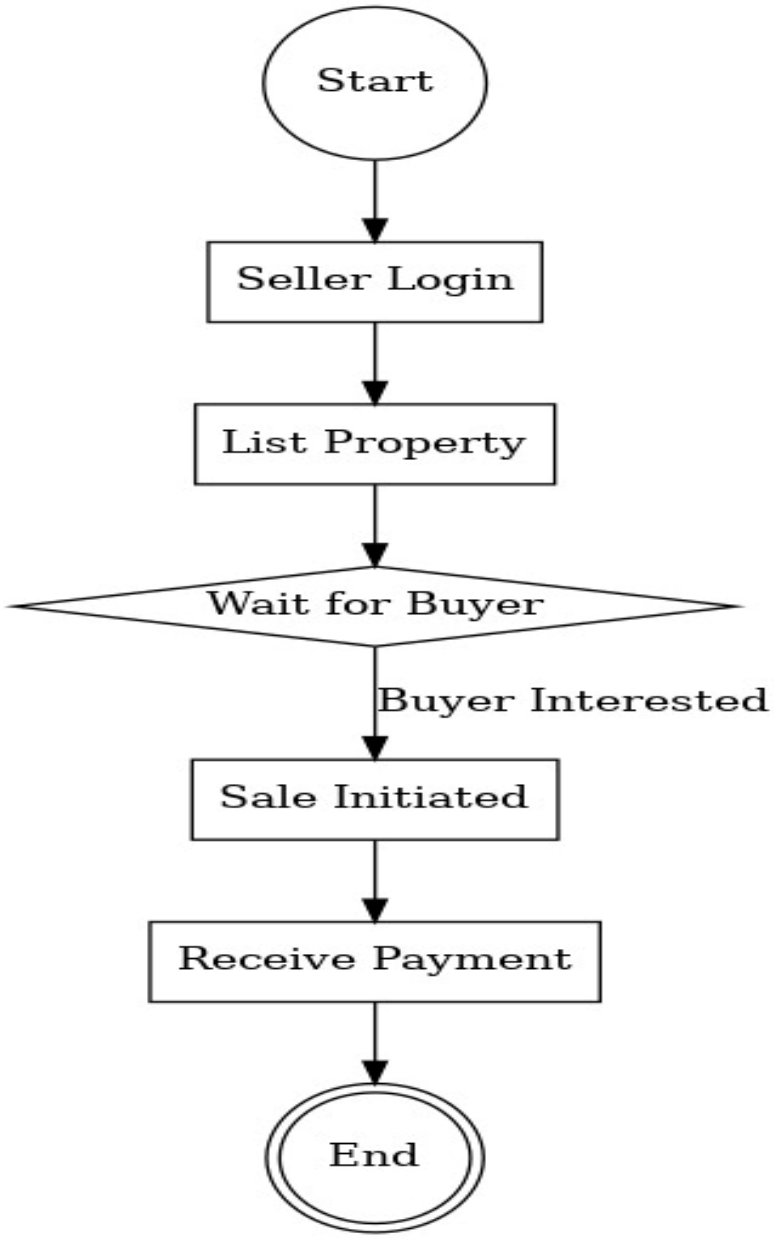
- Buyer: Register/Login, Search Property, View Property, Buy Property.
- Seller: Register/Login, Add Property, Update Property, Delete Property.
- Admin: Manage Users, Manage Properties, Track Sales & Payments.



3.6 UML Activity Diagram (Order Flow)

Flow:

1. Buyer logs in → Searches property.
2. Buyer selects property → Initiates purchase.
3. System records sale → Updates property status.
4. Admin reviews and verifies transaction → Updates payment status.



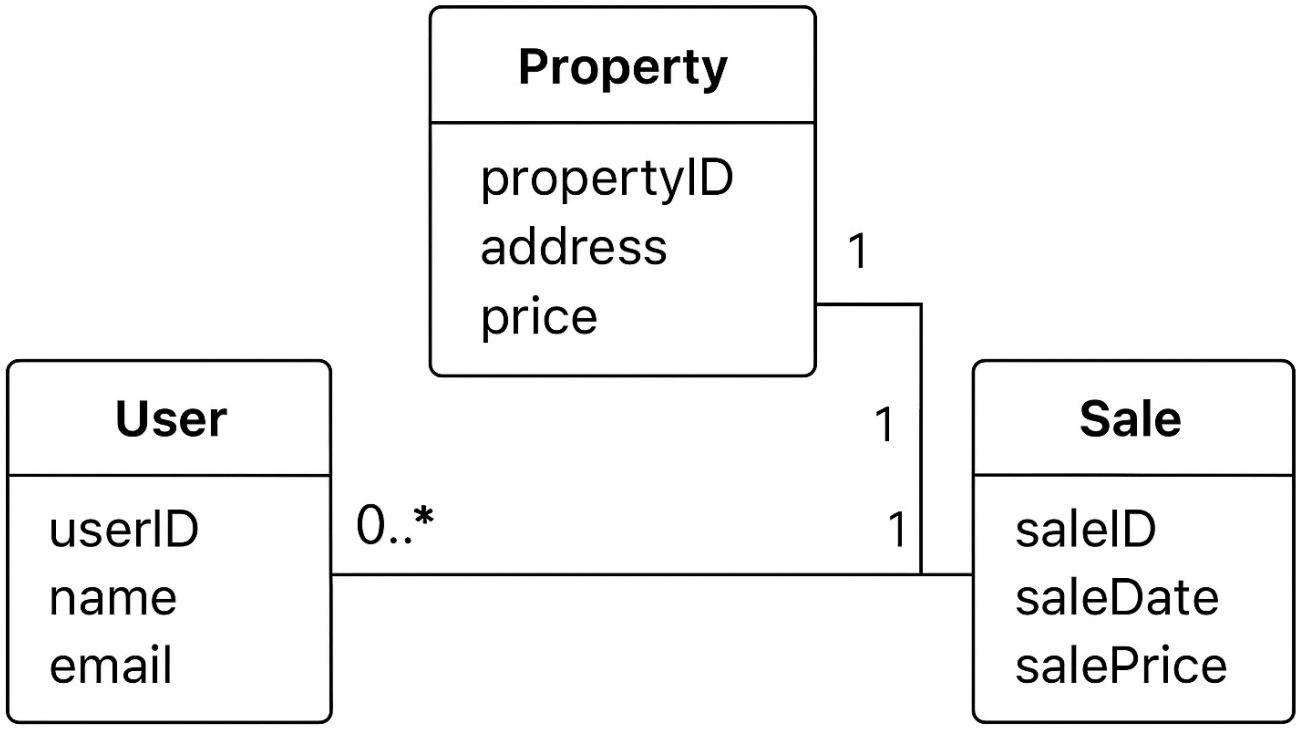
3.7 UML Class Diagram

Classes:

- **User** (User_ID, Username, Password, Role).
- **Property** (Property_ID, Title, Price, Status, Seller_ID).
- **Sale** (Sale_ID, Buyer_ID, Property_ID, Payment_Status).

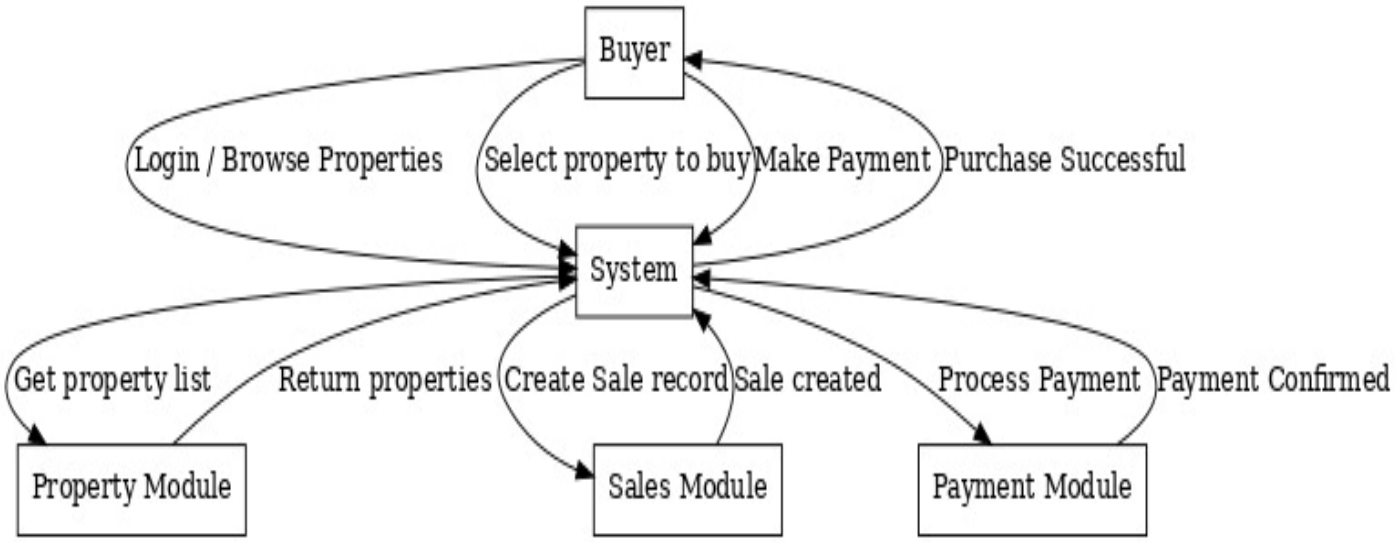
  Relationships:

- User (Seller) → Property (1-to-many).
- User (Buyer) → Sale (1-to-many).
- Property → Sale (1-to-1).



3.8 UML Sequence Diagram (Property Purchase)

1. Buyer → System: Login.
2. System → DB: Verify credentials.
3. Buyer → System: Search property.
4. System → DB: Retrieve property list.
5. Buyer → System: Select property & initiate purchase.
6. System → DB: Record transaction & update property status.
7. Admin → System: Review transaction & confirm payment.



CHAPTER 4

IMPLEMENTATION AND TESTING

4.1 Code (Core Segments)

To implement *EstateEdge*, **Node.js with Express** was used for the backend, **MySQL** for the database, and **Bootstrap** for the frontend. Below are some core code snippets demonstrating critical functionalities.

a) Database Connection (config/database.js)

```
const mysql = require('mysql2');

const db = mysql.createConnection({
   host: 'localhost',
   user: 'root',
   password: '',
   database: 'estateedge'
});

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

module.exports = db;
```

b) User Authentication (middleware/auth.js)

```
const jwt = require('jsonwebtoken');

function authenticateToken(req, res, next) {
   const token = req.headers['authorization']?.split(' ')[1];
```

```
    if (!token) return res.sendStatus(401);

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
        if (err) return res.sendStatus(403);
        req.user = user;
        next();
    });
}

module.exports = authenticateToken;
```

c) Property Routes (server.js excerpt)

```
const express = require('express');
const router = express.Router();
const db = require('./config/database');

// Get all properties
router.get('/properties', (req, res) => {
    db.query("SELECT * FROM properties", (err, results) => {
        if (err) throw err;
        res.json(results);
    });
});

// Add new property
router.post('/properties', (req, res) => {
    const { title, description, price, seller_id } = req.body;
    const sql = "INSERT INTO properties (title, description, price, seller_id, status) VALUES (?, ?, ?, ?,
'available')";
    db.query(sql, [title, description, price, seller_id], (err, result) => {
        if (err) throw err;
        res.json({ message: "Property added successfully" });
    });
});
module.exports = router;
```

d) JWT Login Example (server.js excerpt)

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

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

   db.query("SELECT * FROM users WHERE username = ?", [username], (err, results) => {
      if (err) throw err;
      if (results.length === 0) return res.status(401).json({ message: "User not found" });

      const user = results[0];
      bcrypt.compare(password, user.password, (err, match) => {
         if (!match) return res.status(401).json({ message: "Invalid credentials" });

         const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT_SECRET, { expiresIn: "1h"
});
         res.json({ token });
      });
   });
});
```

4.2 Testing Approach

Testing was carried out in multiple phases to ensure correctness, reliability, and performance of *EstateEdge*.

a) Unit Testing

- Each module (User Management, Property Management, Authentication) was tested individually.
- Example: Testing the login() function with valid/invalid credentials.

b) Integration Testing

- Verified interaction between modules, e.g., ensuring that a seller can add a property and that it reflects in the buyer's search results.
- Tested JWT authentication with restricted routes.

c) System Testing

- End-to-end testing of workflows such as:
    - User Registration → Login → Add Property → Search → Purchase.
    - Admin login → View Dashboard → Manage Users/Properties.

d) User Acceptance Testing (UAT)

- Conducted with sample users (buyers, sellers, admins).
- Ensured the system met requirements: easy navigation, accurate property status updates, secure login.

e) Tools Used

- **Postman** for API testing.
- **MySQL Workbench** for database validation.
- **Jest/Mocha (optional)** for unit tests.
- Manual testing of UI components.

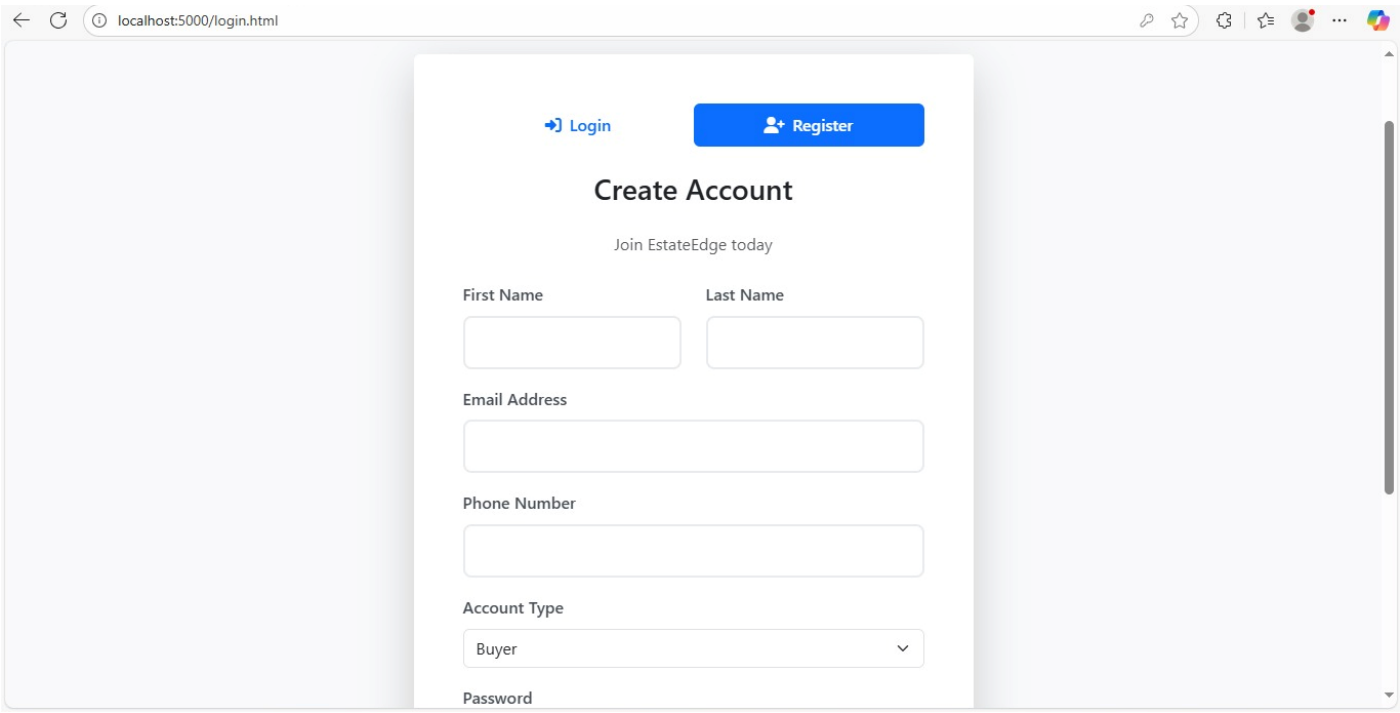
Chapter 5

Results and Discussions (Output Screens)

The implementation of *EstateEdge* was completed successfully, and the system provides all the planned functionalities. The following sections show the key user interfaces and their roles in fulfilling system objectives.

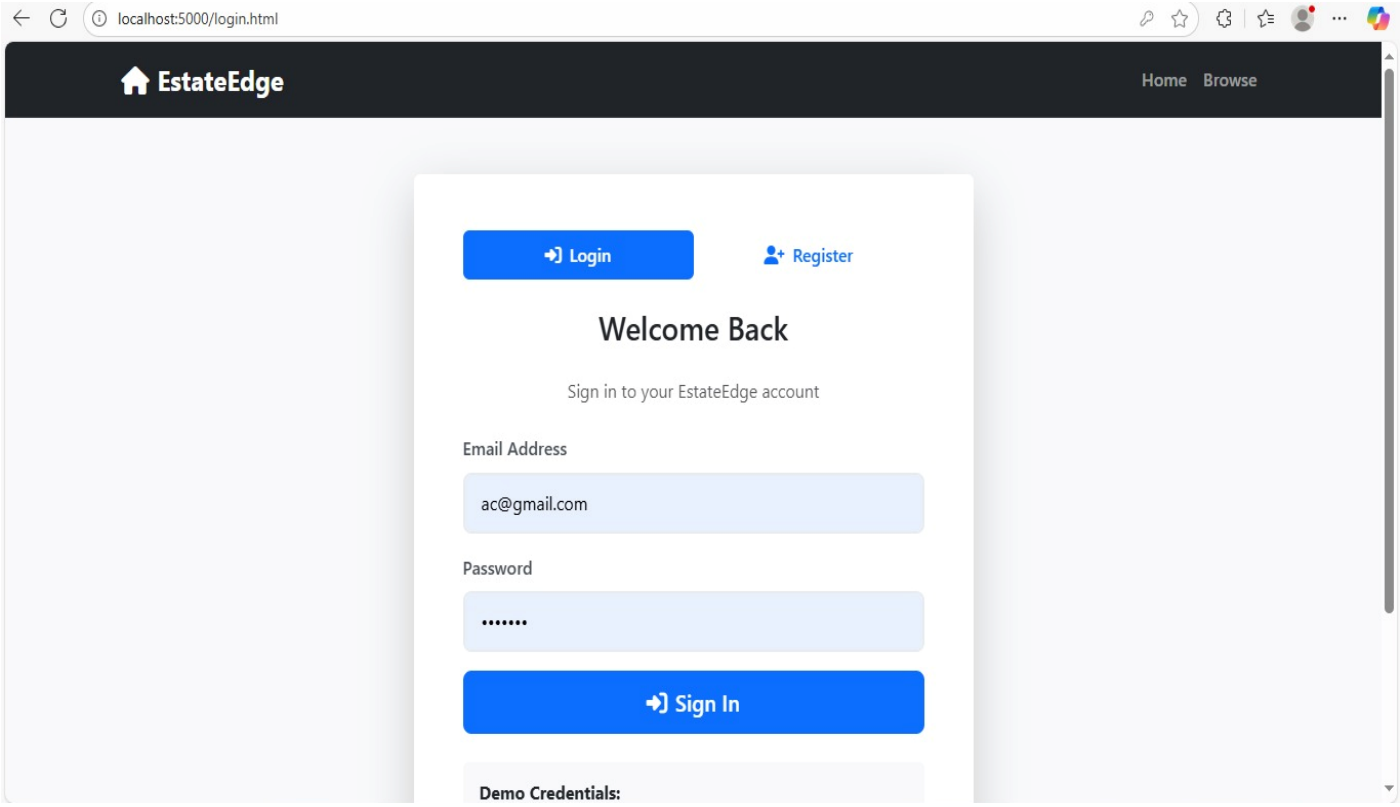
5.1 User Registration Screen

- Allows new users (Buyer, Seller, Admin) to register by entering details such as username, email, password, and role selection.
- Ensures password encryption and input validation.
- Helps in onboarding new users securely.



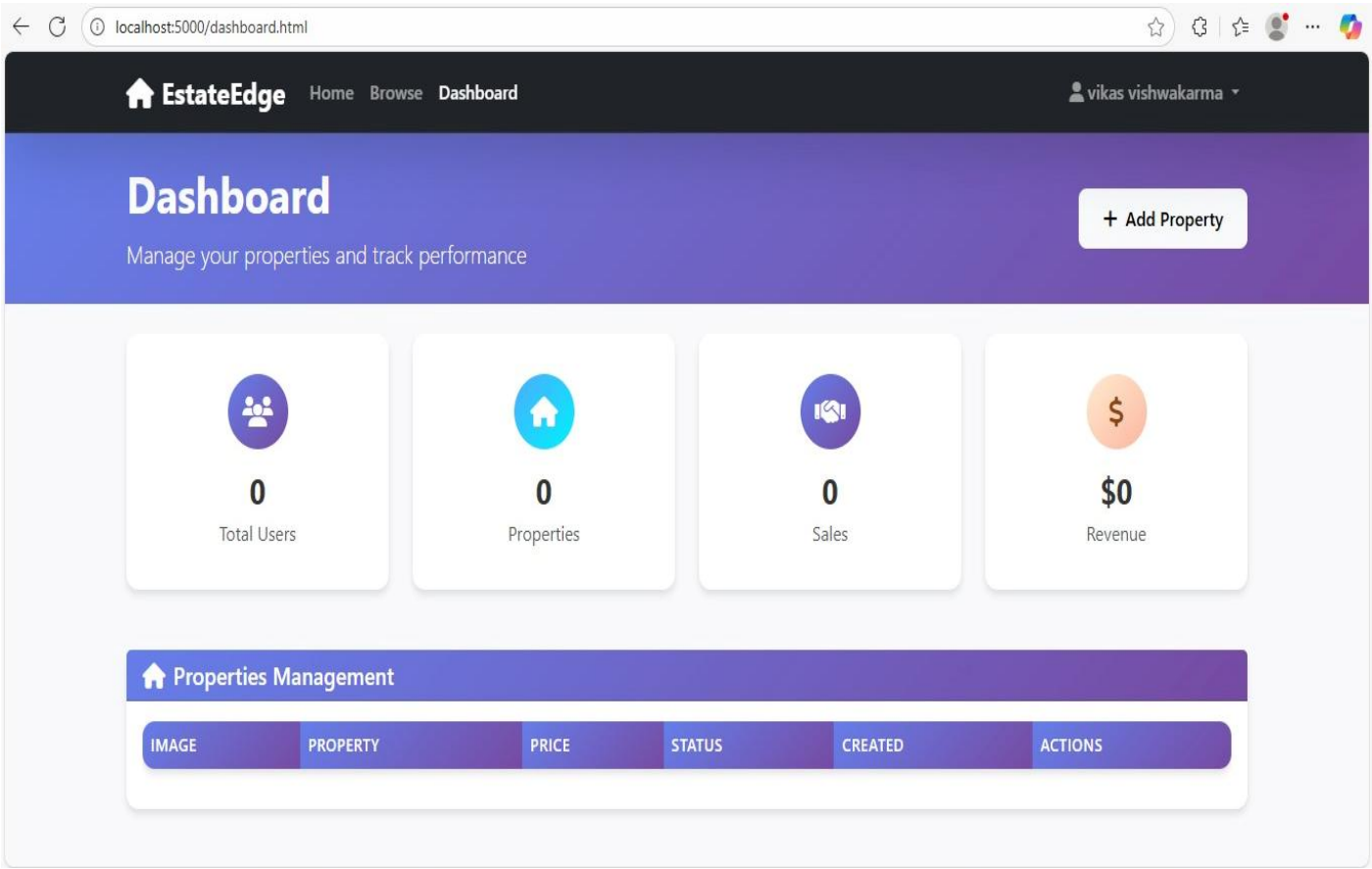
5.2 Login Screen

- Provides secure login using username and password.
- Implements JWT authentication for session management.
- Role-based access ensures that Buyers, Sellers, and Admins are redirected to their respective dashboards.



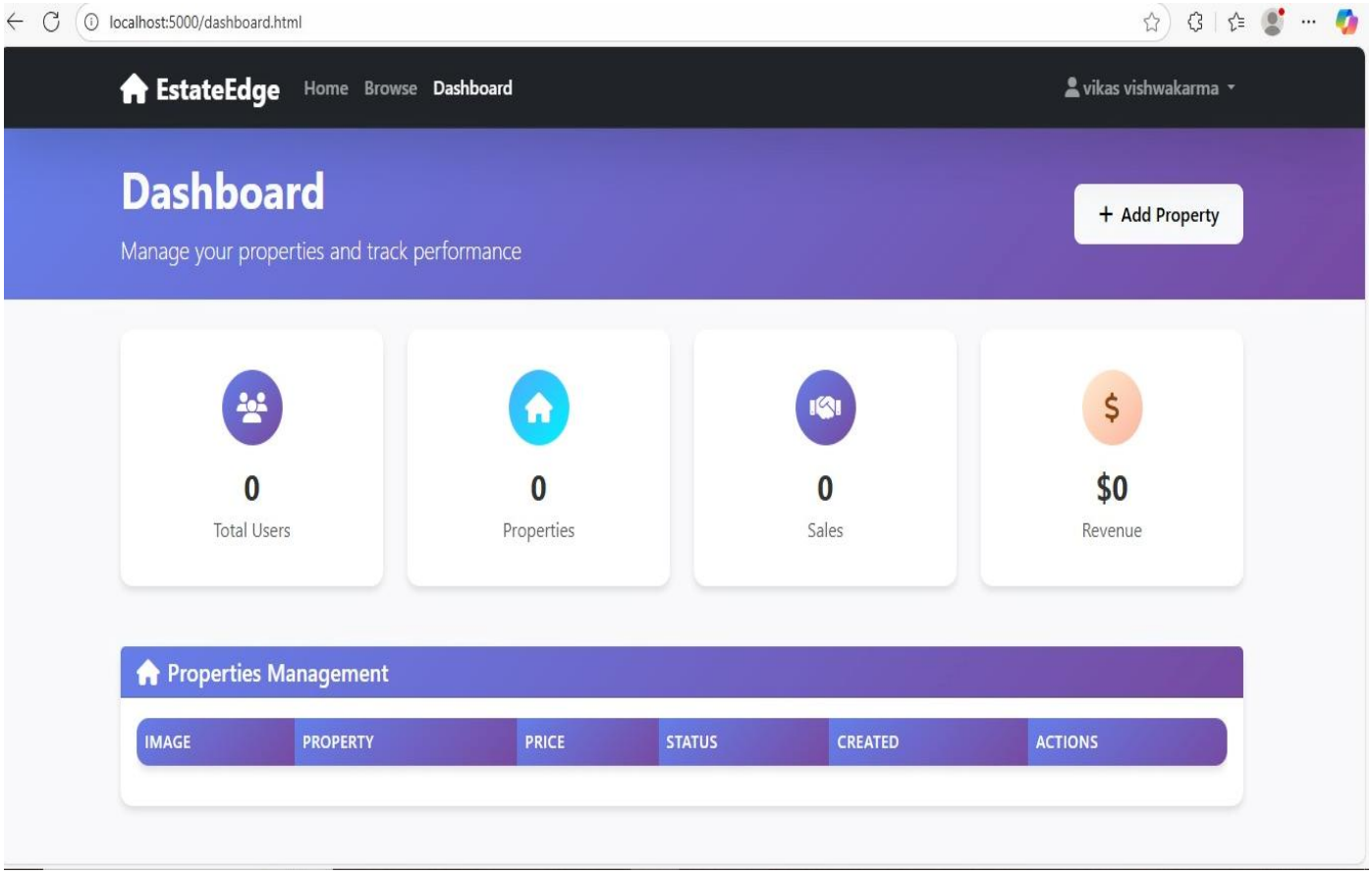
5.3 Buyer Dashboard

- Buyers can browse properties, use search and filter options (location, price, status).
- Displays property details including images, descriptions, and prices.
- Buyers can initiate a purchase request, which updates the property's status.



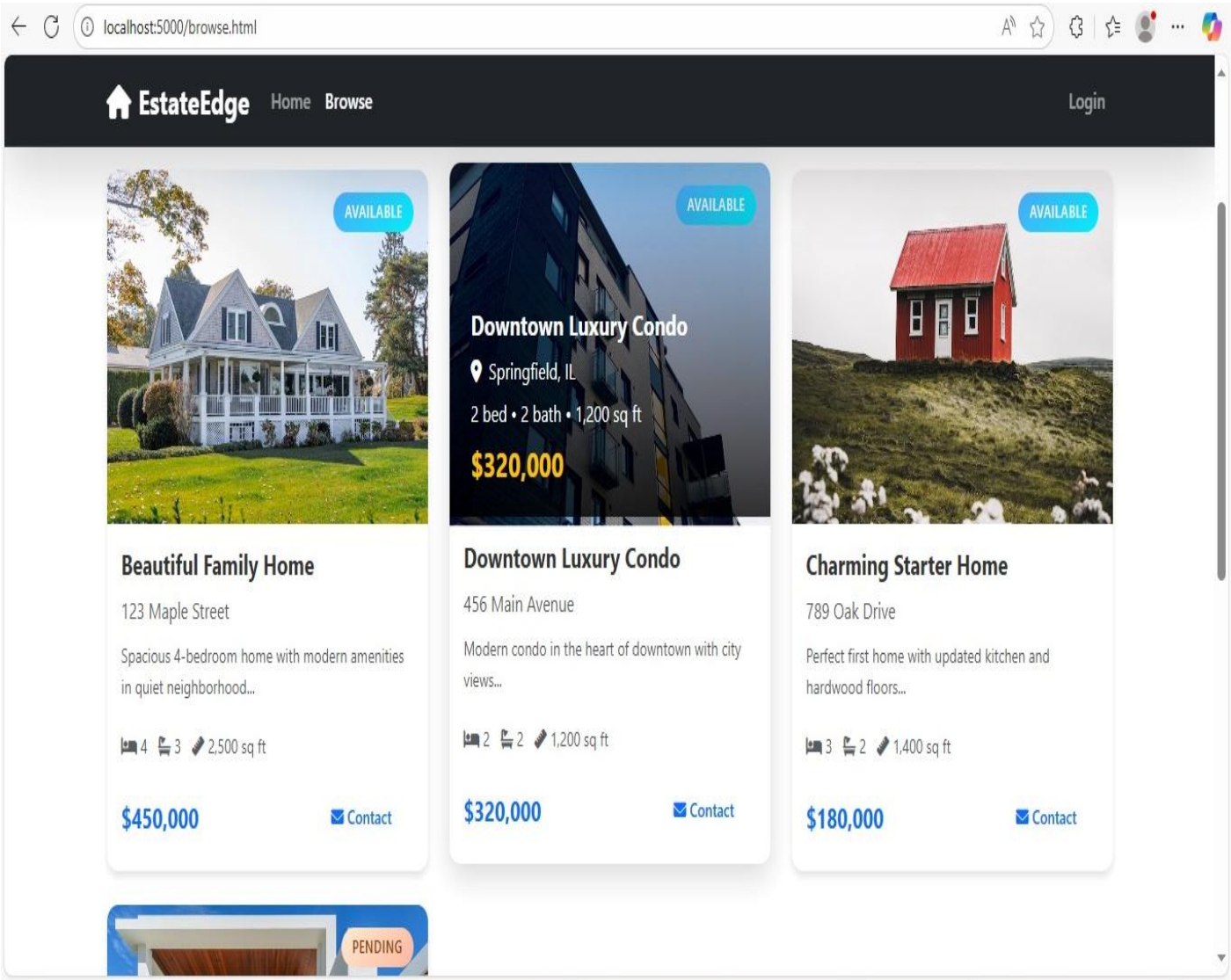
5.4 Seller Dashboard

- Sellers can add new properties with details such as title, price, location, and images.
- Provides options to update or delete listings.
- Displays status of properties (Available, Pending, Sold).



5.6 Property Management Dashboard (Admin)

- Admins have full access to manage users and property listings.
- Provides statistics such as total users, total properties, sales, and revenue.
- Allows monitoring of fraudulent listings or inactive accounts.



✅ **Discussion:**

The results confirm that *EstateEdge* meets the objectives defined in earlier chapters:

- Buyers can easily search and purchase properties.
- Sellers can manage their listings efficiently.
- Admins have full oversight and control over the platform.
- The system ensures **security**, **role-based access**, and **responsive design**, making it a complete solution for digital real estate management.

Chapter 6
CONCLUSION

6.1 Summary of the Project

The *EstateEdge* platform was developed as a full-stack solution to streamline real estate operations by connecting buyers, sellers, and administrators on a single digital platform. The system successfully integrates **user authentication, property management, search and filter functionality, transaction tracking, and an admin dashboard**. By leveraging modern technologies such as **Node.js, Express, MySQL, JWT authentication, and Bootstrap 5**, the project delivers a secure, responsive, and scalable real estate management solution.

6.2 Key Achievements

- Developed a **multi-role authentication system** with secure JWT-based login.
- Implemented **CRUD operations** for property management with real-time status tracking (Available, Pending, Sold).
- Designed a **buyer dashboard** for browsing and purchasing properties with search and filter options.
- Built a **seller dashboard** for managing property listings.
- Created an **admin panel** for monitoring users, properties, transactions, and payments.
- Ensured system-wide **security, performance, and responsiveness** using modern development tools.

6.3 Benefits of the System

- **Efficiency:** Simplifies property discovery and management through a centralized online platform.
- **Transparency:** Provides clear tracking of property status and transactions.
- **Security:** Uses password hashing and JWT-based authentication to protect user data.
- **Accessibility:** Responsive design allows access across devices.
- **Scalability:** Can handle increasing numbers of users and properties without significant performance issues.

6.4 Limitations

- Currently, the system does not support **real-time payment gateway integration** (only admin-tracked payment status).
- Limited to **basic property filtering** (location, price, availability); lacks advanced AI-based recommendations.
- The platform currently functions as a **web application only**, with no dedicated mobile app.
- Requires stable internet connectivity and a server for hosting.

6.5 Future Enhancements

To further improve and expand *EstateEdge*, the following enhancements are proposed:

- **Payment Gateway Integration:** Direct integration with secure online payment systems.
- **AI/ML Recommendations:** Intelligent property suggestions based on buyer preferences and history.
- **Mobile Application Development:** Native Android/iOS apps for better accessibility.
- **Geolocation Services:** Map-based property search and visualization.
- **Advanced Analytics:** Admin dashboards with predictive insights on sales, demand, and property trends.
- **Chat/Support Module:** Enabling direct communication between buyers, sellers, and agents.

Chapter 7

References

The following references were consulted and utilized during the research, design, development, testing, and documentation phases of the **Estateegge Website** project:

7.1 Tools and Platforms Used

The development of *EstateEdge* utilized several tools, frameworks, and platforms that contributed to building a robust and scalable system:

- **Node.js** – Backend runtime environment.
- **Express.js** – Web framework for handling routes and APIs.
- **MySQL** – Relational database management system for storing user and property data.
- **Bootstrap 5** – Frontend framework for responsive UI design.
- **Vite** – Frontend build tool for faster development and deployment.
- **Postman** – API testing tool.
- **Git & GitHub** – Version control and collaborative development.
- **Visual Studio Code (VS Code)** – Primary code editor.

7.2 AI Assistance (If Used)

During the development and documentation of *EstateEdge*, AI-based tools such as **ChatGPT** were used to:

- Draft technical documentation (Abstract, Acknowledgement, Chapters).
- Generate structured content for system analysis and design.
- Provide guidance on UML diagrams, ER diagrams, and DFD descriptions.
- Assist in code explanation and best practices for implementation.

AI assistance was not directly involved in coding but was leveraged for **knowledge support, documentation clarity, and content structuring**.

7.3 Online Tutorials & Documentation

The following resources were referenced during the implementation of *EstateEdge*:

- **Node.js Documentation:** https://nodejs.org/docs
- **Express.js Guide:** https://expressjs.com
- **MySQL Documentation:** https://dev.mysql.com/doc
- **Bootstrap 5 Docs:** https://getbootstrap.com/docs
- **JWT Authentication Guide:** https://jwt.io/introduction
- **W3Schools (JavaScript, SQL, HTML, CSS):** https://www.w3schools.com
- **Stack Overflow Community:** https://stackoverflow.com

These references provided the necessary **technical guidance, examples, and best practices** that supported the successful completion of the project.