CERTIFICATE

Certified that **Saumya Sharma 2300290140164, Shrishti Pandey 2300290140177** have carried out the project work having "**EstateVista**" (**Mini Project-KCA353**) for **Master of Computer Applications** from Dr. A.P.J. Abdul Kalam TechnicalUniversity (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the students and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Saumya Sharma 2300290140164

Shrishti Pandey 2300290140177

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Apoorv Jain	**Dr. Arun Kumar Tripathi**
Assistant Professor	**Head of Department**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

EstateVista

ABSTRACT

In today's fast-paced real estate market, the need for a user-friendly and efficient property search platform has never been greater. This project aims to develop a comprehensive real estate website designed to meet the needs of property buyers, sellers, and renters. The platform combines advanced search functionalities, detailed property insights, and seamless connectivity with real estate agents, offering a streamlined and intuitive experience for all users.

The website features an extensive database of property listings, including residential, commercial, and rental options, presented with high-quality images, comprehensive descriptions, and detailed specifications. Users can filter their searches based on criteria such as location, price range, property type, and amenities, ensuring tailored results that align with their preferences.

The platform also includes interactive tools such as a map-based property search. These features enable users to make well-informed decisions by understanding property values and the surrounding area better. Additionally, users can save searches, bookmark properties.

Connecting users directly with verified real estate agents is a key aspect of the website, fostering trust and transparency. The platform facilitates property visit scheduling, inquiries, and document submissions, streamlining the real estate process for all parties involved.

In conclusion, this real estate website offers a holistic solution to property search and transactions by integrating robust tools, detailed information, and streamlined communication channels. It serves as a one-stop destination for navigating the complexities of the real estate market with ease and confidence.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Apoorv Jain** for his guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directlyand indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Saumya Sharma

Shrishti Pandey

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview

EstateVista is a modern and versatile real estate platform designed to streamline property discovery, management, and transactions for buyers, sellers, and renters. Built with the MERN stack and Prisma, EstateVista delivers an intuitive, feature-rich experience that caters to the diverse needs of real estate enthusiasts. With its robust functionalities and user-friendly design, the platform empowers users to navigate the complexities of the real estate market effortlessly.

At its core, EstateVista offers secure user authentication powered by Auth0, ensuring a safe and personalized experience for every user. The platform enables users to list properties with ease, upload high-quality images, and showcase precise location details through an interactive map interface. This map-based integration provides users with a comprehensive view of property locations and their surroundings, facilitating informed decision-making.

The property listing module aggregates diverse property options submitted by users, offering a centralized and dynamic repository for residential, commercial, and rental properties. Advanced search filters allow users to refine their searches based on location, price range, property type, and amenities, ensuring they can find properties that match their specific preferences.

EstateVista also features a "Favourites" functionality, enabling users to bookmark properties of interest for quick reference and comparison. Additionally, the platform's visit booking system allows users to schedule property viewings directly, simplifying the coordination process between buyers and sellers.

A standout aspect of EstateVista is its focus on user engagement through visually enriched and data-driven tools. The interactive map not only displays property locations but also provides valuable insights into neighbourhood dynamics, proximity to services, and other geographical details, helping users evaluate options with confidence.

EstateVista' s responsive design ensures seamless accessibility across all devices, allowing users to explore, list, and manage properties from desktops, tablets, or smartphones. The platform's clean interface and streamlined navigation make it an ideal choice for both experienced real estate professionals and first-time users.

In conclusion, EstateVista redefines the real estate journey by integrating advanced functionalities such as property listing, interactive mapping, visit scheduling, and personalized features. It serves as a one-stop destination for property enthusiasts, delivering an efficient and enjoyable experience while bridging the gap between property seekers and sellers in today's fast-paced market.

1.2 Motivation

The development of EstateVista was inspired by the need to modernize and streamline the property management process for our PG owner who also operates a real estate business. Before the creation of this platform, our client relied on traditional methods to manage property listings, communicate with clients, and organize transactions. These conventional approaches, though functional, were often time-consuming, inefficient, and unable to meet the growing demands of the modern real estate market.

Our primary motivation was to provide our client with a digital solution that could simplify and enhance the way properties are listed, discovered, and managed. By creating EstateVista, we aimed to replace outdated methods with a dynamic, feature-rich platform that not only improves operational efficiency but also offers a more engaging experience for users.

The need to adapt to a fast-paced market and increasing client expectations was another driving force behind this project. Traditional methods lacked the ability to provide quick, detailed, and personalized property information. EstateVista addresses this gap by offering an intuitive platform that enables users to list properties, upload images, and showcase locations on an interactive map. The inclusion of search filters, favourites functionality, and visit booking options further ensures that property seekers can explore and interact with listings effortlessly.

Another critical aspect of our motivation was the recognition that our client's real estate business required a scalable and modernized approach to meet the growing competition in the industry. EstateVista provides our client with tools to attract more clients, manage properties effectively, and improve their market reach through a professional and user-friendly digital presence.

Finally, this project was driven by our commitment to empowering our client with technology that simplifies their workload while enhancing the experience for property seekers. By building EstateVista using the MERN stack and Prisma, we delivered a robust, responsive, and secure platform that aligns with contemporary digital standards and ensures long-term scalability.

In summary, the development of EstateVista was motivated by the desire to transform our client's traditional real estate practices into a modern, efficient, and engaging digital platform. By addressing the challenges of traditional methods and embracing the potential of technology, EstateVista provides a tailored solution that empowers our client to thrive in the competitive real estate market.

1.3 Problem Statement

The traditional methods of property search and management in real estate often fall short of meeting the needs of property owners, buyers, renters, and agents, especially in a dynamic and fast-paced market. Our client, a PG owner who also runs a real estate business, was previously relying on outdated manual processes that were inefficient, time-consuming, and lacked scalability. These conventional methods posed several significant challenges that hindered accessibility, efficiency, and personalization.

Key Issues:

1. **Inefficient Property Listing and Discovery:**
   Traditional approaches require extensive manual work to list properties and share information. This often results in delays, errors, and missed opportunities. Buyers and renters also struggle to find suitable properties due to limited search options and outdated listings.

2. **Lack of Real-Time Updates and Dynamic Customization:**
   Manual systems do not offer real-time updates or dynamic adjustments based on user interactions and preferences. This results in conflicts in scheduling, missed visits, and overall inconvenience for buyers, sellers, and agents.

3. **Complex Communication and Scheduling:**
   Coordinating property visits, managing inquiries, and handling document exchanges are cumbersome in conventional systems. This fragmented communication process creates delays, misunderstandings, and inefficiencies for all parties involved.

4. **Limited User Engagement and Personalization:**
   Existing systems fail to offer personalized property recommendations that align with individual preferences, such as location, price, and amenities. This lack of customization makes the search process tedious and often leads to subpar experiences for users.

5. **Accessibility Challenges and Scalability Issues:**
   For property owners in remote areas or those with limited digital knowledge, reaching potential buyers and renters is difficult. Additionally, traditional methods do not scale well, making it hard to grow a real estate business efficiently in a competitive market.

Objective:
EstateVista aims to address these issues by replacing traditional methods with a robust digital platform built on the MERN stack and Prisma. The platform focuses on streamlining property listings, enabling real-time communication, offering dynamic search filters, and integrating features like map-based interactions and visit scheduling.

1.4 Expected Outcome

The implementation of the EstateVista platform is expected to transform property management and transactions by addressing the inefficiencies of traditional real estate practices. By integrating advanced digital solutions and user-centric design, EstateVista aims to deliver a seamless, efficient, and personalized real estate experience. The expected outcomes include the following key benefits:

1. Efficient Property Listing and Discovery

- Users will be able to list properties effortlessly, with options to upload high-quality images and detailed descriptions.
- A dynamic, real-time property database will ensure that listings are always up-to-date, eliminating discrepancies and missed opportunities.
- Advanced search filters and personalized recommendations will enable buyers and renters to find properties that match their preferences in terms of location, price, property type, and amenities.

2. Seamless Communication and Scheduling

- The platform will facilitate direct communication between buyers, sellers, and verified real estate agents, ensuring transparency and trust.
- A built-in scheduling feature will allow users to book property visits effortlessly, streamlining coordination and minimizing scheduling conflicts.
- Efficient document sharing and inquiries management will reduce paperwork hassle and increase transactional efficiency.

3. Enhanced User Engagement and Personalization

- Users will benefit from personalized property recommendations and interactive tools that cater to their individual preferences, budgets, and schedules.
- The “Favourites” functionality will enable users to save properties for quick reference and comparison, enhancing engagement and decision-making.
- An intuitive interface and responsive design will make navigation simple and user-friendly across all devices, ensuring a seamless experience on desktop, tablet, and mobile.

4. Scalability and Integration

- The scalable architecture of the MERN stack and Prisma ensures that EstateVista can grow
- alongside the client's real estate business.
- Integration with digital tools and booking services will streamline the property search, communication, and transaction processes, consolidating services into a single cohesive platform.
- The platform will enable better business scalability by attracting more clients and maintaining efficient operations across multiple locations.

CHAPTER 2

LITERATURE SURVEY

This study explores the development and optimization of **EstateVista**, a real estate platform designed to revolutionize property search and transaction experiences. Through an analysis of existing digital tools and technologies, this research aims to identify best practices, evaluate user engagement strategies, and address key challenges within the real estate landscape. The insights from this literature survey focus on interactive features, personalized search tools, integrated booking functionalities, and inclusivity within real estate platforms, ultimately guiding the design and implementation of a robust and scalable solution.

1. Interactive User Engagement in Real Estate Platforms

Several studies emphasize the importance of **interactive features** in enhancing user engagement within real estate websites. Platforms like **Zillow, Realtor.com, Magicbricks**, and **Rightmove** include functionalities such as dynamic property filters, interactive maps, and virtual tours. Research shows that interactive elements significantly increase user participation and satisfaction by providing a more immersive experience, enabling quick decision-making, and fostering trust through transparency.

- **Dynamic Search Filters:** Customizable search filters allow users to tailor their property exploration experience according to their preferences in location, price, amenities, and property type.
- **Virtual Tours and Multimedia Integration:** High-quality videos and photos facilitate remote viewing experiences, allowing users to evaluate properties without the need for physical visits.
- **Map-Based Interfaces:** Interactive maps display critical information about neighbourhoods, commute times, property values, and nearby amenities, helping users make informed choices.

Key Insight: Incorporating dynamic, multimedia-rich, and location-based search tools enhances user engagement by offering a more interactive, transparent, and information-rich experience.

2. Personalized Real Estate Search Tools

Personalization in real estate search platforms is crucial to meet the unique needs of property buyers, sellers, and renters. Studies on platforms like **Compass** and **Trulia** reveal the following insights:

- **User Data Analytics:** Machine learning algorithms analyse user behaviour and preferences to recommend properties tailored to specific tastes, budgets, and schedules.
- **Preference-Based Recommendations:** Tools that allow users to save searches and bookmark properties ensure long-term engagement and a more personalized search experience.

- **Real-Time Updates:** Personalized search tools often include real-time updates about property availability, market trends, and local developments, which enhance decision-making.

Key Insight: A robust personalized search system using machine learning and real-time data provides users with highly relevant property recommendations and ensures an optimized experience.

3. Integration of Communication and Scheduling Tools

Research highlights the role of **integrating communication and scheduling tools** within real estate platforms to streamline interactions between buyers, sellers, and agents. Tools like **Calendly, DocuSign**, and CRM systems (e.g., Salesforce) are often incorporated to facilitate scheduling visits, managing inquiries, and processing documentation.

- **Appointment Scheduling Integration:** Embedded scheduling tools reduce back-and-forth communications and enable quick, hassle-free property visits.
- **Digital Documentation Handling:** Document submission and verification features improve the efficiency and security of property transactions.
- **Agent Communication Interfaces:** Platforms that connect buyers directly with verified agents offer transparency, trust, and quick responses.

Key Insight: A unified communication system embedded within the platform enhances collaboration and streamlines interactions across all stakeholders, ensuring a smooth transaction process.

4. Accessibility and Inclusivity in Real Estate Platforms

Accessibility is a growing area of interest in research related to digital real estate platforms. Many studies address the digital divide and highlight the need to ensure inclusivity for all potential users, including those with disabilities and limited digital literacy.

- **Adherence to Digital Accessibility Standards:** Platforms must comply with standards like **WCAG (Web Content Accessibility Guidelines)** to ensure inclusiveness.
- **Assistive Technologies Integration:** Voice interfaces, screen readers, and intuitive UI/UX design enable users with disabilities to navigate the platform seamlessly.
- **Multilingual Interfaces:** Providing content in multiple languages ensures that real estate tools are accessible to a broader and more diverse audience.

Key Insight: Prioritizing accessibility ensures that all users, regardless of digital expertise, income, or location, can actively engage with property search tools, making real estate opportunities more inclusive and equitable.

5. Booking Services and Transaction Management

Integrated booking and transaction tools are essential in simplifying property-related operations. Studies on platforms like **DocuSign, Booking.com**, and **CRM systems** illustrate their importance:

- **All-in-One Booking Interfaces:** Seamlessly booking property visits, arranging inspections, and signing contracts in a single interface minimizes logistical challenges.
- **Secure Transaction Management:** End-to-end encryption and robust document verification protocols ensure transactional security and trust.
- **Efficient Agent-Buyer Interaction:** Tools that enable real-time interaction and scheduling streamline communication and foster transparency.

Chapter 3

Feasibility Study

3.1. Technical Feasibility

Technical feasibility focuses on the technical resources available to implement the project and evaluates whether the proposed technology stack is suitable for its development.

- **Technology Stack**: EstateVista is being developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), which is robust and widely used for web applications.
- **Frontend**: ReactJS and Chakra UI offer a user-friendly interface and ensure responsive design, making it accessible on various devices.
- **Backend**: Node.js with Express.js provides efficient server-side operations and seamless API integration.
- **Database**: MongoDB, a NoSQL database, supports scalability and flexibility for storing real estate listings and user data.
- **Infrastructure**: Modern hosting platforms like AWS, Google Cloud, or Heroku can be used for deployment.
- **Scalability**: The MERN stack is scalable, which means EstateVista can handle future growth in users and listings efficiently.
- **Development Tools**: The project uses Git for version control and project management tools like Trello or Jira for collaboration.

3.2. Operational Feasibility

Operational feasibility determines if the proposed project aligns with the organization's goals and whether users will adopt it.

- **Ease of Use**: The platform is designed to be intuitive and user-friendly with features like filtering properties, searching for nearby listings, and a clean UI that eliminates clutter.
- **Target Audience**: Users looking for properties online will find this platform efficient and reliable. It bridges the gap between traditional paper-based property dealings and digital solutions.
- **Business Alignment**: The platform aligns with the goal of simplifying property dealings and providing transparency in real estate transactions.

- **User Support**: FAQs, chatbots, and customer service will ensure user satisfaction and seamless operations.
- **Stakeholder Acceptance**: Real estate agents, property owners, and buyers will find the platform convenient for their needs, encouraging adoption.

3.3. Economic Feasibility

Economic feasibility evaluates the cost-benefit analysis of the project to determine its financial viability.

- **Development Costs**:
  - Salaries of developers (MERN stack expertise)
  - Hosting and deployment costs
  - Licenses for premium services (if any)
- **Maintenance Costs**: Regular updates, debugging, and adding new features will incur ongoing costs.
- **Revenue Model**:
  - Commission from property transactions
  - Subscription fees for premium features (e.g., advanced filtering, ad-free browsing)
  - Advertisement revenue from third-party property-related businesses
- **ROI (Return on Investment)**:
  - The platform is expected to recover development costs within 1-2 years through a combination of ad revenue and subscription fees.
  - Increased user base will lead to a higher revenue stream over time.
- **Market Demand**: With the increasing demand for online property listings, the platform has significant revenue potential.

CHAPTER 4

Design

4.1.1 Level 0 Data Flow Diagram

Level 0 Data Flow Diagram will explain the basic flow of data in a system which showshow the new or old user will interact with the system.

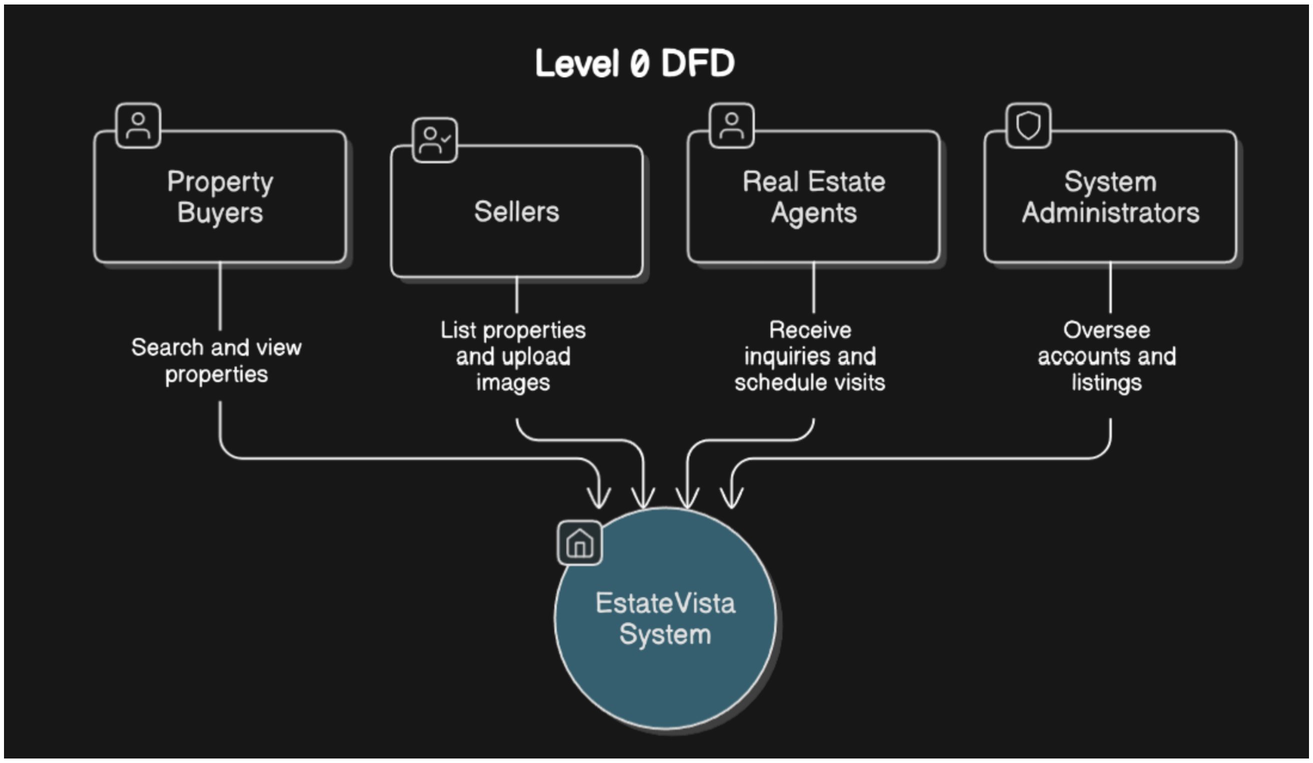
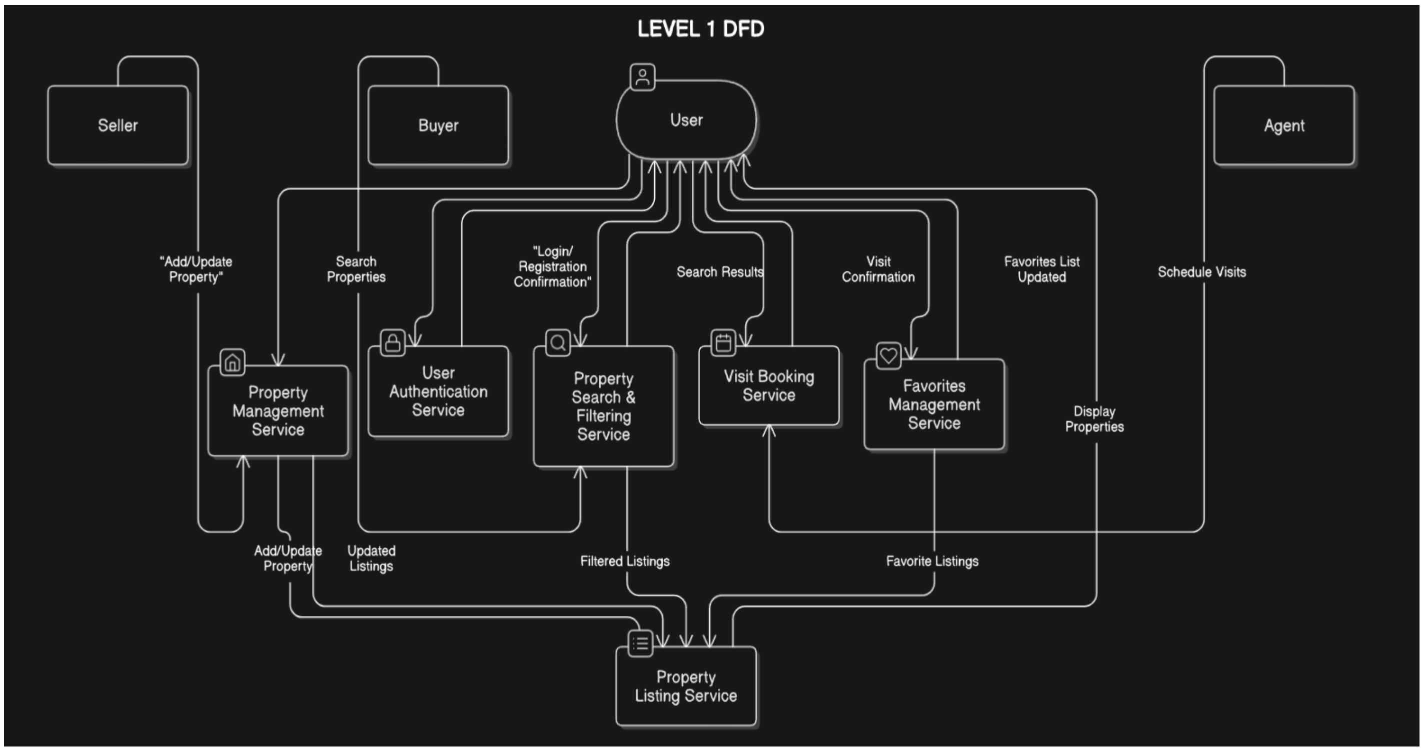

Level 0 DFD
Property Buyers
Sellers
Real Estate Agents
System Administrators
Search and view properties
List properties and upload images
Receive inquiries and schedule visits
Oversee accounts and listings
EstateVista System

Fig. 3.1 Level 0 DFD of EstateVista

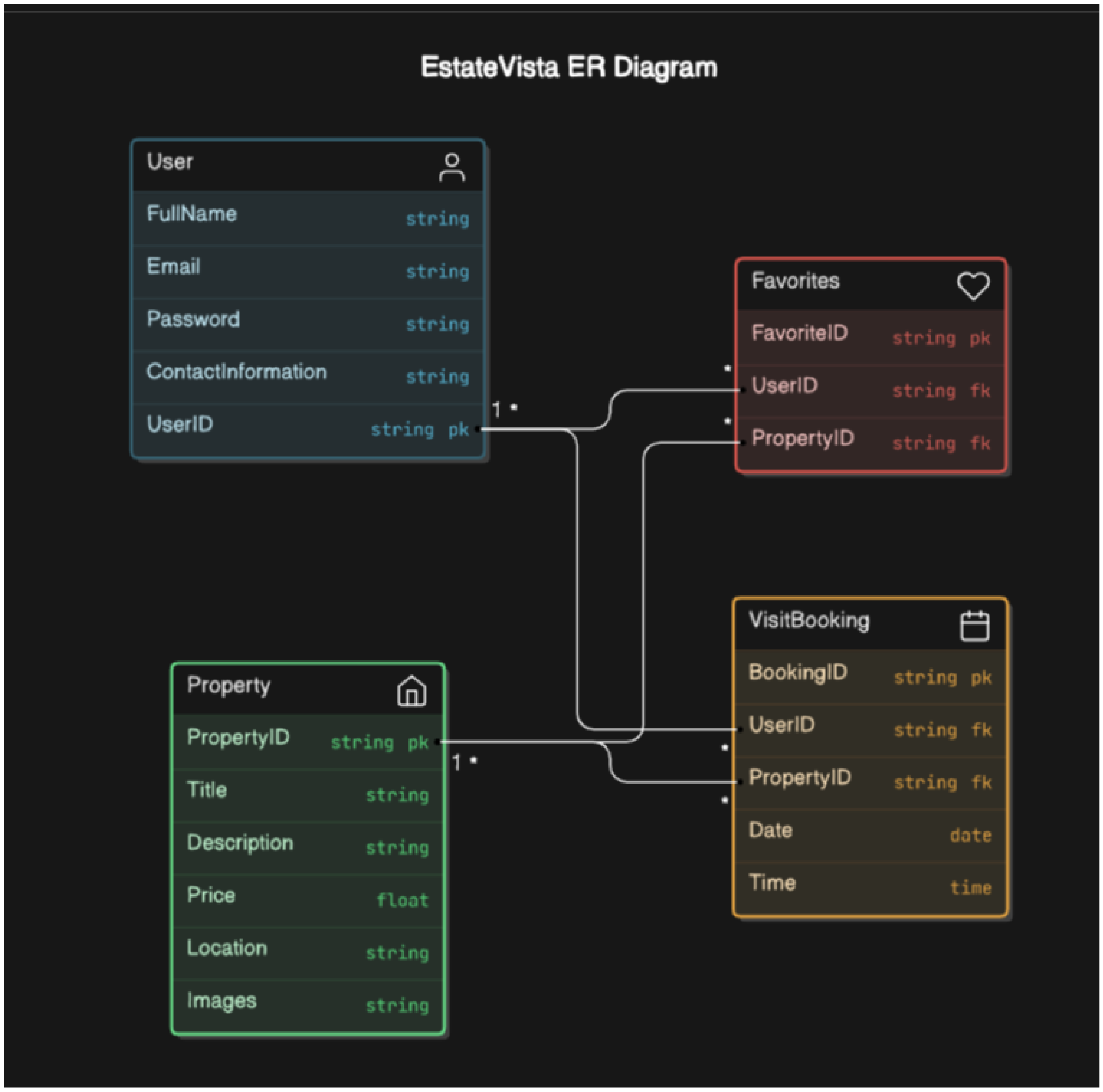
4.1.2 Level 1 Data Flow Diagram

Level 1 Data Flow Diagram will explain the basic flow of data in a system which showshow the new or old user will interact with the system with different processes.



4.2 ER Diagram

An Entity Relationship Diagram is a diagram that represents relationships among entities in a database.



4.3 Use Case Diagram

In Use Case Diagram we elaborate about the purpose, actor, pre-condition, post- condition, basic flow, and alternate flow of all the use cases.

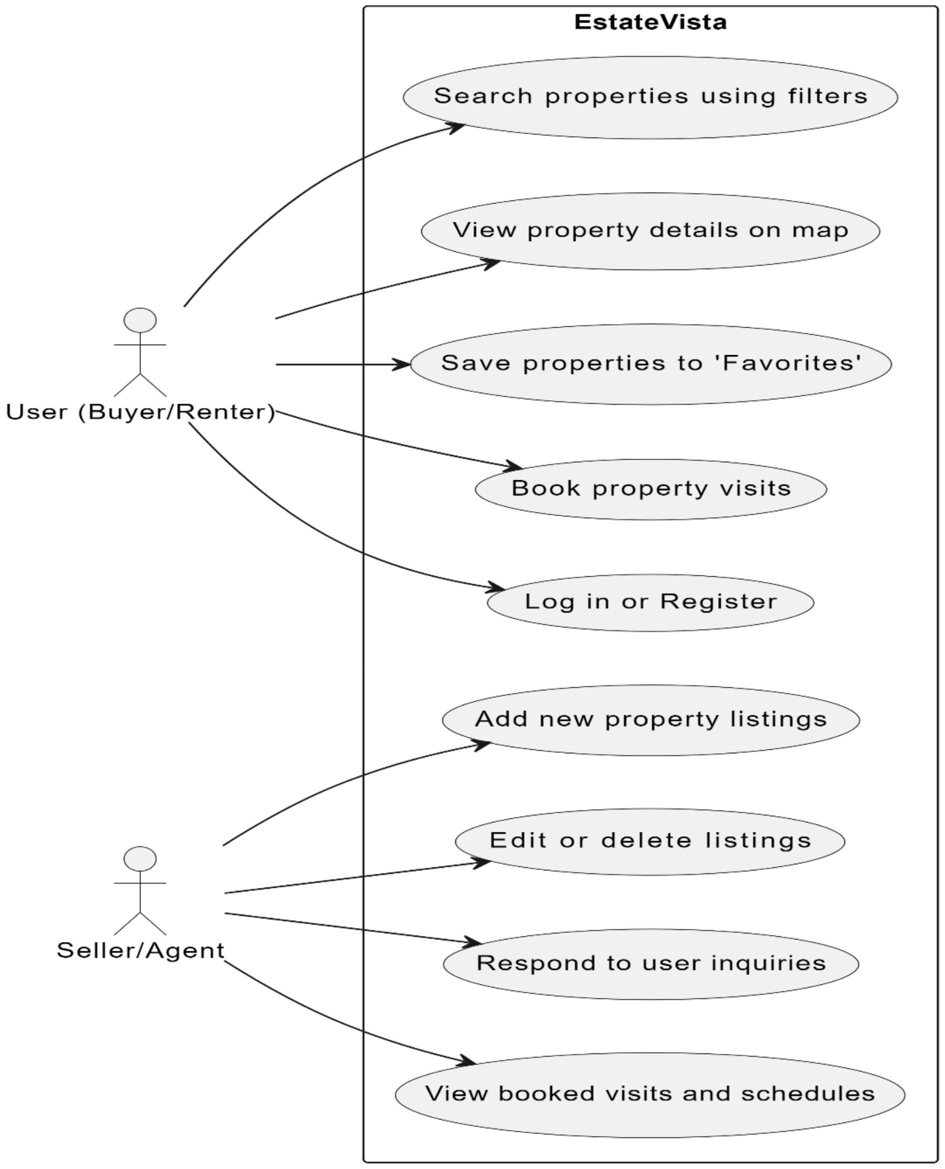

EstateVista
Search properties using filters
View property details on map
Save properties to 'Favorites'
Book property visits
Log in or Register
Add new property listings
Edit or delete listings
Respond to user inquiries
View booked visits and schedules
User (Buyer/Renter)
Seller/Agent

Fig. 3.7 Use Case Diagram of   EstateVista

CHAPTER 5

PROPOSED WORK

5.1 Technology Description

- **Selection of Operating System:** Our website is platform independent, so it does not depend on the operating system.
- **Selection of IDE:** Visual Studio is used to create our software.
- **Languages Used:** React JS, Mongo Db, Prisma, Express JS, Node JS.

5.1Approach Used

EstateVista is a real estate platform designed to provide seamless property management, listing, and booking experiences for buyers, sellers, and agents. The platform offers property search, authentication, map integration, and visit booking functionalities. It is built using **React.js and JavaScript** for the frontend, with **Node JS, Express JS, Mongo Db, Prisma** serving as the backend infrastructure.

5.2.1 Objective

1. **User Authentication and Security**

   - To provide a seamless **user authentication experience** using **Auth0**, ensuring robust security and user privacy.

2. **Effortless Property Management**

   - To enable sellers to **add and update property listings**, upload property images, and showcase these properties on a map based on location.

3. **Efficient Property Search & Filtering**

   - To implement an advanced **property search and filtering system**, ensuring buyers can find properties quickly based on criteria such as **location, price, size, and amenities**.

4. **Personalized Favourites Management**

   - To offer a feature where users can **add properties to their favorites list**, making it convenient to access and revisit properties of interest.

5. **Smooth Visit Booking Service**

   - To integrate a system for **booking visits**, where users can easily schedule viewings with sellers and agents.

5.2.2 Technologies Used

5.2.2.1 **Frontend:** React.js.

5.2.2.2 **Backend:** Node JS, Mongo Db, Prisma, Express JS

5.2.3 Features

User Authentication

- Users can log in or register using Auth0, ensuring a fast and secure authentication process.

Property Management

- Sellers can add properties, including property details and images.
- Properties are displayed on the map with location integration, enhancing visibility and user interaction.

Property Listing & Search

- A comprehensive property listing page displays properties listed by different users (sellers, agents).
- The search filter allows users to specify criteria like location, price, property type, and amenities.

Favourites Management

- Users can save properties to their favourites list, allowing quick and easy access to preferred properties.

Visit Booking Service

- An integrated system enables users to schedule and confirm visits, facilitating real-time communication between buyers and sellers.

5.3 Implementation Details

1. Frontend Development: Utilized React.js to create a dynamic and responsive user interface.

2. Backend Services: Used Node JS, Express JS for the backend.

3. User Authentication: Integrated Auth0 for secure authentication, registration, and password management.

4. Data Management: Stored the user details, property details, favorite data in no SQL database called mongo db.

5.4 Challenges Faced

Scalability

- Managing a growing number of property listings and user interactions required a scalable architecture to ensure smooth performance and quick retrieval of search results.

Security Concerns

- Ensuring a robust authentication system and safeguarding user information required integrating Auth0 and adopting best practices in secure coding and database management.

Performance Optimization

- Ensuring fast page load times and quick property search responses involved optimizing both frontend and backend code, as well as database queries.

Real-Time Map Integration

- Displaying property locations on the map required integrating the Google Maps API, ensuring real-time updates and accurate positioning.

5.5 Future Enhancements

Enhanced Interactive UI

- Integrating more interactive elements, such as 360-degree property tours and multimedia content, to provide a richer user experience.

Social Interaction Features

- Adding social features like user reviews, discussion forums, and community interactions, fostering engagement among buyers, sellers, and agents.

Advanced Analytics Dashboard

- Implementing a comprehensive analytics system to track user interactions, search patterns, favourites preferences, and visit bookings.

CHAPTER 6

CODING

Index.html

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<link rel="icon" type="image/svg+xml" href="/vite.svg" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Estate Vista</title>
<script src="https://upload-widget.cloudinary.com/global/all.js" type="text/javascript"></script>
</head>
<body>
<div id="root"></div>
<script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

Components

About.jsx

```
import React, { useEffect, useState } from 'react';
import CountUp from 'react-countup';
import aboutImg from '../assets/about.jpg';
import { RiDoubleQuotesL } from "react-icons/ri";

const About = () => {
   // Define the statistics
   const statistics = [
      { label: 'Happy clients', value: 12 },
      { label: 'Different cities', value: 3 },
      { label: 'Project completed', value: 45 }
   ];

   const [isVisible, setIsVisible] = useState(false);

   useEffect(() => {
      const handleScroll = () => {
         const aboutSection = document.getElementById('about');
         if (aboutSection) {
            const top = aboutSection.getBoundingClientRect().top;
            const isVisible = top < window.innerHeight - 100;
            setIsVisible(isVisible);
         }
      };

      window.addEventListener('scroll', handleScroll);

      // Cleanup function to remove the event listener
      return () => {
         window.removeEventListener('scroll', handleScroll);
      };
```

```
  }, []);

  // Function to format numbers with commas
  const formatNumberWithCommas = (number) => {
    return number.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ',');
  };

  return (
    <section id='about' className='max-padd-container py-16 xl:py-28'>
      {/* Container */}
      <div className='flex flex-col xl:flex-row gap-10'>
        {/* Left side */}
        <div className='flex-1 relative'>
          <img src={aboutImg} alt="" className='rounded-3xl rounded-tr-[155px] w-[488px]' />
          <div className='bg-white absolute bottom-16 left-16 max-w-xs p-4 rounded-lg
flexCenter flex-col'>
            <span className='relative bottom-8 p-3 shadow-md bg-white h-12 w-12 flex items-
center rounded-full'><RiDoubleQuotesL className='text-2xl'/></span>
            <p className='text-center relative bottom-3'>Lorem ipsum dolor sit amet consectetur
adipisicing elit. Optio, maiores illum!</p>
          </div>
        </div>
        {/* Right side */}
        <div className='flex-1 flex justify-center flex-col'>
          <span className='medium-18'>Unveiling Our Journey</span>
          <h2 className='h2'>Our Commitment Crafting Extraordinary Real Estate
Experiences</h2>
          <p className='py-5'>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nobis
dolore expedita delectus in a eligendi explicabo laborum eveniet? Ratione modi et earum assumenda
est vitae neque laborum fugiat unde expedita perferendis amet rem illum quis facere voluptatum
culpa repudiandae natus provident porro, nihil fuga.</p>
          {/* Statistics Container */}
          <div className="flex flex-wrap gap-4">
            {statistics.map((statistic, index) => (
              <div key={index} className="bg-primary p-4 rounded-lg">
                <div className='flex items-center gap-1'>
                  <CountUp start={isVisible ? 0 : null} end={statistic.value} duration={10}
delay={3}>
                    {({ countUpRef }) => (
                      <h3 ref={countUpRef} className="text-2xl font-semibold "></h3>
                    )}
                  </CountUp>
                  <h4 className='bold-22'>k+</h4>
                </div>
                <p className="text-gray-600">{statistic.label}</p>
              </div>
            ))}
          </div>
        </div>
      </div>
    </section>
  );
};

export default About;
```

AddPropertyModal.jsx

```
import React, { useState } from "react";
import { Container, Modal, Stepper } from "@mantine/core";
import AddLocation from "./AddLocation";
import { useAuth0 } from "@auth0/auth0-react";
import UploadImage from "./UploadImage";
import BasicDetails from "./BasicDetails";
import Facilities from "./Facilities";

const AddPropertyModal = ({ opened, setOpened }) => {
  const [active, setActive] = useState(0);
  const { user } = useAuth0();
  const [propertyDetails, setPropertyDetails] = useState({
    title: "",
    description: "",
    price: 0,
    country: "",
    city: "",
    address: "",
    image: null,
    facilities: {
      bedrooms: 0,
      parkings: 0,
      bathrooms: 0,
    },
    userEmail: user?.email,
  });

  const nextStep = () => {
    setActive((current) => (current < 4 ? current + 1 : current));
  };
  const prevStep = () => {
    setActive((current) => (current > 0 ? current - 1 : current));
  };

  return (
    <Modal
      opened={opened}
      onClose={() => setOpened(false)}
      closeOnClickOutside
      size={"90rem"}
    >
      <Container h={"34rem"} w={"100%"}>
        <Stepper
          active={active}
          onStepClick={setActive}
          breakpoint="sm"
          allowNextStepsSelect={false}
        >
          <Stepper.Step label="Location" description="Address">
            <AddLocation
              nextStep={nextStep}
              propertyDetails={propertyDetails}
              setPropertyDetails={setPropertyDetails}
            />
          </Stepper.Step>
          <Stepper.Step label="Images" description="Upload">
            <UploadImage
            prevStep={prevStep}
            nextStep={nextStep}
            propertyDetails={propertyDetails}
            setPropertyDetails={setPropertyDetails}
            />
          </Stepper.Step>
          <Stepper.Step label="Basics" description="Details">
```

```
            <BasicDetails
            prevStep={prevStep}
            nextStep={nextStep}
            propertyDetails={propertyDetails}
            setPropertyDetails={setPropertyDetails}
            />
          </Stepper.Step>
          <Stepper.Step>
            <Facilities
            prevStep={prevStep}
            propertyDetails={propertyDetails}
            setPropertyDetails={setPropertyDetails}
            setOpened={setOpened}
            setActiveStep={setActive}
            />
          </Stepper.Step>
          <Stepper.Completed>
            Completed, click back button to get to previous step
          </Stepper.Completed>
        </Stepper>
      </Container>
    </Modal>
  );
};

export default AddPropertyModal;
```

Hero.jsx

```
import React from 'react'
import {Link} from "react-router-dom"

const Hero = () => {
return (
<section className='max-padd-container pt-[99px]'>
<div className='max-padd-container bg-hero bg-center bg-no-repeat bg-cover h-[655px] w-full
rounded-3xl'>
<div className='relative top-32 xs:top-52 '>
<span className='medium-18'>Welcome to CasaCentral</span>
<h1 className='h1 capitalize max-w-[40rem] '>Discover Exceptional Homes with Casacentral</h1>
<p className='my-10 max-w-[33rem]'>Lorem, ipsum dolor sit amet consectetur adipisicing elit. Non rerum dolorum cumque magni rem illo quia, autem porro excepturi, quidem commodi ad perferendis culpa.</p>
{/* button */}
<div className='inline-flex items-center justify-center gap-4 p-2 bg-white rounded-xl'>
<div className='text-center regular-14 leading-tight pl-5'>
<h5 className='uppercase font-bold'>10% Off</h5>
<p className='regular-14'>On All Properties</p>
</div>
<Link to={'/listing'} className={"btn-secondary rounded-xl flexCenter !py-5"}>Shop now</Link>
</div>
</div>
</div>
</section>
)
}

export default Hero;
```

Map.jsx

```
import React from 'react'
import { MapContainer, TileLayer } from 'react-leaflet'
import GeoCoderMarker from './GeoCoderMarker'

const Map = ({ address, city, country }) => {
return (
<MapContainer
center={[53.35, 18.8]}
zoom={1}
scrollWheelZoom={false}
className='h-[24rem] w-full mt-5 z-0'
>
<TileLayer url='https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'/>
<GeoCoderMarker address={`${address} ${city} ${country}`} />
</MapContainer>
)
}
```

```
export default Map;
```

BookingModal.jsx

```
import React, { useContext, useState } from "react";
import { Modal, Button } from "@mantine/core";
import { DatePicker } from "@mantine/dates";
import { useMutation } from "react-query";
import UserDetailContext from "../context/UserDetailContext";
import { bookVisit } from "../utils/api";
import { toast } from "react-toastify";
import dayjs from "dayjs";

const BookingModal = ({ opened, setOpened, email, propertyId }) => {
const [value, setValue] = useState(null);
const {
userDetails: { token },
setUserDetails,
} = useContext(UserDetailContext);

const handleBookingSuccess = () => {
toast.success("You have successfully booked visit", {
position: "bottom-right",
});
setUserDetails((prev) => ({
...prev,
bookings: [
...prev.bookings,
{
id: propertyId,
date: dayjs(value).format("DD/MM/YYYY"),
},
],
}));
};
const { mutate, isLoading } = useMutation({
mutationFn: () => bookVisit(value, propertyId, email, token),
onSuccess: () => handleBookingSuccess(),
onError: ({ response }) => toast.error(response.data.message),
onSettled: () => setOpened(false),
});
// console.log(token)
return (
<Modal
opened={opened}
onClose={() => setOpened(false)}
title="Select your date of visit"
centered
>
<div className="flexCenter flex-col gap-4">
<DatePicker value={value} onChange={setValue} minDate={new Date()} />
<Button disabled={!value || isLoading} onClick={() => mutate()}>
Book visit
</Button>
</div>
</Modal>
);
};

export default BookingModal;
```

Item.jsx

```
import React from 'react';
import { MdOutlineBed, MdOutlineBathtub, MdOutlineGarage } from "react-icons/md";
import { CgRuler } from "react-icons/cg";
import { Link, useNavigate } from 'react-router-dom';
import HeartBtn from './HeartBtn';

const Item = ({ property }) => {
  const navigate = useNavigate();
  return (
    <div className='rounded-2xl p-5 bg-white' onClick={()=>navigate(`../listing/${property.id}`)}>
      <div className='pb-2 relative'>
        <img src={property.image} alt={property.title} className='rounded-xl' />
        {/* like btn */}
        <div className="absolute top-4 right-6">
          <HeartBtn id={property?.id}/>
        </div>
      </div>
      <h5 className='bold-16 my-1 text-secondary'>{property.city}</h5>
      <h4 className='medium-18 line-clamp-1'>{property.title}</h4>
      {/* info */}
      <div className='flex gap-x-2 py-2'>
        <div className='flexCenter gap-x-2 border-r border-slate-900/50 pr-4 font-
[500]'><MdOutlineBed /> {property.facilities.bedrooms}</div>
        <div className='flexCenter gap-x-2 border-r border-slate-900/50 pr-4 font-
[500]'><MdOutlineBathtub /> {property.facilities.bathrooms}</div>
        <div className='flexCenter gap-x-2 border-r border-slate-900/50 pr-4 font-
[500]'><MdOutlineGarage /> {property.facilities.parkings}</div>
        <div className='flexCenter gap-x-2 border-r border-slate-900/50 pr-4 font-[500]'><CgRuler />
400</div>
      </div>
      <p className='pt-2 mb-4 line-clamp-2'>{property.description}</p>
      <div className='flexBetween'>
        <div className='bold-20'>${property.price}.00</div>
        <Link to={`/`}>
          <button className='btn-secondary rounded-xl !px-5 !py-[7px] shadow-sm'>View
details</button>
```

```
        </Link>
      </div>
    </div>
  );
}

export default Item;
```

Searchbar.jsx

```
import React from 'react'
import { FaLocationDot } from "react-icons/fa6";

const Searchbar = ({filter, setFilter}) => {
    return (
        <div className='flexBetween pl-6 h-[3.3rem] bg-white w-full max-w-[366px] rounded-full
ring-1 ring-slate-500/5'>
            <input type="text" value={filter} onChange={(e)=> setFilter(e.target.value)}
placeholder='Enter an address or city' className='bg-transparent border-none outline-none' />
            <FaLocationDot className='relative right-4 text-xl hover:text-secondary' />
        </div>
    )
}

export default Searchbar;
```

Hooks

useAuthCheck.jsx

```
import {useAuth0} from "@auth0/auth0-react"
import { toast } from "react-toastify"

const useAuthCheck = () => {

    const { isAuthenticated } = useAuth0()
    const validateLogin = () => {
        if(!isAuthenticated){
            toast.error("Please Login first", {position: "bottom-right"})
            return false
        } else return true
    }
  return (
    {validateLogin}
  )

export default useAuthCheck
```

useBookings.jsx

```
import React, { useContext, useEffect, useRef } from "react";
import UserDetailContext from "../context/UserDetailContext";
import { useQuery } from "react-query";
import { useAuth0 } from "@auth0/auth0-react";
import { getAllBookings } from "../utils/api";

const useBookings = () => {
  const { userDetails, setUserDetails } = useContext(UserDetailContext);
  const queryRef = useRef();
  const { user } = useAuth0();

  const { data, isLoading, isError, refetch } = useQuery({
    queryKey: "allBookings",
    queryFn: () => getAllBookings(user?.email, userDetails?.token),
    onSuccess: (data) =>
      setUserDetails((prev) => ({ ...prev, bookings: data })),
    enabled: user !== undefined,
    staleTime: 30000,
  });

  queryRef.current = refetch;

  useEffect(() => {
    queryRef.current && queryRef.current();
  }, [userDetails?.token]);
  return { data, isError, isLoading, refetch };
};

export default useBookings;
```

useFavourites.jsx

```
import React, { useContext, useEffect, useRef } from 'react'
import UserDetailContext from '../context/UserDetailContext'
import { useQuery } from 'react-query'
import { useAuth0 } from '@auth0/auth0-react'
import { getAllFav } from '../utils/api'

const useFavourites = () => {

    const { userDetails, setUserDetails } = useContext(UserDetailContext)
    const queryRef = useRef()
    const {user} = useAuth0()

    const { data, isLoading, isError, refetch } = useQuery({
        queryKey: "allFavourites",
        queryFn: ()=> getAllFav(user?.email, userDetails?.token),
        onSuccess: (data)=> setUserDetails((prev)=> ({...prev, favourites: data})),
        enabled: user!==undefined,
        staleTime: 30000
    })

    queryRef.current = refetch;

    useEffect(()=> {

        queryRef.current && queryRef.current()
    }, [userDetails?.token])
  return { data, isError, isLoading, refetch }
}

export default useFavourites
```

useProperties.jsx

```
import React from "react";
import { useQuery } from "react-query";
import { getAllProperties } from "../utils/api";

const useProperties = () => {
  const { data, isLoading, isError, refetch } = useQuery(
    "allProperties",
    getAllProperties,
    { refetchOnWindowFocus: false }
  );
  return {
    data,
    isLoading,
    isError,
    refetch,
  };
};

export default useProperties;
```

utils

App.jsx

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Home from "./pages/Home";
import Listing from "./pages/Listing";
import AddProperty from "./pages/AddProperty";
import { QueryClient, QueryClientProvider } from "react-query"
import { ToastContainer } from "react-toastify";
import { ReactQueryDevtools } from 'react-query/devtools';
import "react-toastify/dist/ReactToastify.css"
import Property from "./pages/Property";
import { Suspense, useState } from "react";
import UserDetailContext from "./context/UserDetailContext";
import Layout from "./components/Layout";
import Favourites from "./pages/Favourites";
import Bookings from "./pages/Bookings";

export default function App() {

  const queryClient = new QueryClient();
  const [userDetails, setUserDetails] = useState({
    favourites: [],
    bookings: [],
    token: null
  })
  return (
    <UserDetailContext.Provider value={{ userDetails, setUserDetails }} >
      <QueryClientProvider client={queryClient}>
          <BrowserRouter>
            <Suspense fallback={<div>Loading...</div>}>
            <Routes>
              <Route element={<Layout />}>
                <Route path="/" element={<Home />} />
                <Route path="/listing" >
                  <Route index element={<Listing />} />
                  <Route path=":propertyId" element={<Property />} />
                </Route>
                <Route path="/addproperty" element={<AddProperty />} />
                <Route path="/bookings" element={<Bookings />} />
                <Route path="/favourites" element={<Favourites />} />
              </Route>
            </Routes>
          </Suspense>
        </BrowserRouter>
      <ToastContainer />
      <ReactQueryDevtools initialIsOpen={false} />
    </QueryClientProvider>
  </UserDetailContext.Provider>
 )
}
```

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
import "@mantine/core/styles.css";
import "@mantine/dates/styles.css";
import { Auth0Provider } from "@auth0/auth0-react"
import { MantineProvider } from '@mantine/core'; // Import MantineProvider

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
```

```
    <Auth0Provider
      domain='dev-sma7reoctm83q6qn.us.auth0.com'
      clientId='GnsU5LTNzAnnh5DOarln6gzfoyo8Ohpm'
      authorizationParams={{
        redirect_uri: "http://localhost:5173"
      }}
      audience="http://localhost:8000"
      scope="openid profile email"
    >
      <MantineProvider>
        <App />
      </MantineProvider>
    </Auth0Provider>
  </React.StrictMode>
)
```

Controllers

resdCntrl.js

```
import asyncHandler from "express-async-handler";
import { prisma } from "../config/prismaConfig.js";

export const createResidency = asyncHandler(async (req, res) => {
const {
title,
description,
price,
address,
country,
city,
facilities,
image,
userEmail,
} = req.body.data || req.body;

console.log("Request Body:", req.body); // Log the entire request body
console.log("User Email:", userEmail);  // Log the userEmail

if (!userEmail) {
return res.status(400).send({ message: "User email is required" });
}

try {
const residency = await prisma.residency.create({
data: {
title,
description,
price,
address,
country,
city,
facilities,
image,
owner: { connect: { email: userEmail } },
},
});

res.send({ message: "Residency created", residency });
} catch (err) {
console.error("Error:", err); // Log the error for debugging
if (err.code === "P2002") {
return res.status(400).send({ message: "A residency with the provided address already exists" });
}
res.status(500).send({ message: err.message });
}
});

// Other route handlers...

// getting all documents/residencies

export const getAllResidencies = asyncHandler(async (req, res) => {
```

```
const residencies = await prisma.residency.findMany({
orderBy: {
createdAt: "desc",
},
});
res.send(residencies);
});

// get document/residency by id
export const getResidency = asyncHandler(async (req, res) => {
const { id } = req.params;

try {
const residency = await prisma.residency.findUnique({
where: { id },
});
res.send(residency);
} catch (err) {
throw new Error(err.message);
}
});
```

userCntrl.js

```
import asyncHandler from "express-async-handler";

import { prisma } from "../config/prismaConfig.js";

// function for creating user
export const createUser = asyncHandler(async (req, res) => {
console.log("creating a user");

let { email } = req.body;
const userExists = await prisma.user.findUnique({ where: { email: email } });
if (!userExists) {
const user = await prisma.user.create({ data: req.body });
res.send({
message: "User registered seccessfully",
user: user,
});
} else res.status(201).send({ message: "User already registered" });
});

// function for book visit
export const bookVisit = asyncHandler(async (req, res) => {
const { email, date } = req.body;
const { id } = req.params;

try {
const alreadyBooked = await prisma.user.findUnique({
where: { email },
select: { bookedVisits: true },
});

if (alreadyBooked.bookedVisits.some((visit) => visit.id === id)) {
res
.status(400)
.json({ message: "selected residency already booked by you" });
} else {
await prisma.user.update({
where: { email: email },
data: {
bookedVisits: { push: { id, date } },
},
});
res.send("visit booked successfully");
}
} catch (err) {
throw new Error(err.message);
}
});

// function for allBookings
export const getAllBookings = asyncHandler(async (req, res) => {
const { email } = req.body;

try {
const bookings = await prisma.user.findUnique({
where: { email },
select: { bookedVisits: true },
});
res.status(200).send(bookings);
} catch (err) {
throw new Error(err.message);
}
});

// function for cancel a booking
export const cancelBooking = asyncHandler(async (req, res) => {
const { email } = req.body;
const { id } = req.params;

try {
const user = await prisma.user.findUnique({
where: { email: email },
select: { bookedVisits: true },
});
const index = user.bookedVisits.findIndex((visit) => visit.id === id);

if (index === -1) {
res.status(404).json({ message: "No Booking found" });
} else {
user.bookedVisits.splice(index, 1);
await prisma.user.update({
where: { email },
data: {
bookedVisits: user.bookedVisits,
},
});
res.send("Booking canceled");
}
} catch (err) {
throw new Error(err.message);
}
});

// function for addtofavourite list
export const toFav = asyncHandler(async (req, res) => {
console.log("Adding to favourites"); // Log to ensure function is called
console.log("Request body:", req.body); // Log to debug request body
```

```
const { email } = req.body;
const { rid } = req.params;

if (!email) {
res.status(400).json({ message: "Email is required" });
return;
}

try {
const user = await prisma.user.findUnique({
where: { email },
});

if (user.favResidenciesID.includes(rid)) {
const updateUser = await prisma.user.update({
where: { email },
data: {
favResidenciesID: {
set: user.favResidenciesID.filter((id) => id !== rid),
},
},
});
res.send({ message: "Removed from favourite", user: updateUser });
} else {
const updateUser = await prisma.user.update({
where: { email },
data: {
favResidenciesID: {
push: rid,
},
},
});
res.send({ message: "Updated favorites", user: updateUser });
}
} catch (err) {
console.error('Error in toFav function:', err.message); // Log error
res.status(500).send({ message: 'Internal server error' });
}
});
```

```
// function to getallfav
export const getAllFav = asyncHandler(async (req, res) => {
const { email } = req.body;
try {
const favResd = await prisma.user.findUnique({
where: { email },
select: { favResidenciesID: true },
});
res.status(200).send(favResd);
} catch (err) {
throw new Error(err.message);
}
});
```

routes

residencyRoute.js

```
import express from 'express';
import { createResidency, getAllResidencies, getResidency } from '../controllers/resdCntrl.js';
import jwtCheck from '../config/auth0Config.js';
const router = express.Router();

router.post("/create", jwtCheck, createResidency)
router.get("/allresd", getAllResidencies)
router.get("/:id", getResidency)

export {router as residencyRoute}
```

userRoute.js

```
import express from "express";
import {
  bookVisit,
  cancelBooking,
  createUser,
  getAllBookings,
  getAllFav,
  toFav,
} from "../controllers/userCntrl.js";
import jwtCheck from "../config/auth0Config.js";
const router = express.Router();

router.post("/register", jwtCheck, createUser);
router.post("/bookVisit/:id", jwtCheck, bookVisit);
router.post("/allBookings", getAllBookings);
router.post("/removeBooking/:id", jwtCheck, cancelBooking);
router.post("/toFav/:rid", jwtCheck, toFav);
router.post("/allFav", jwtCheck, getAllFav);

export { router as userRoute };
```

Index.js

```
import cookieParser from 'cookie-parser';
import cors from 'cors'
import dotenv from 'dotenv';
import express from 'express';
import { userRoute } from './routes/userRoute.js';
import { residencyRoute } from './routes/residencyRoute.js';
dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000 ;

app.use(express.json());
app.use(cookieParser());
app.use(cors());

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

app.use('/api/user', userRoute)
app.use('/api/residency', residencyRoute)
```

config

auth0config.js

```
import { auth } from 'express-oauth2-jwt-bearer';

const jwtCheck = auth({
   audience: "http://localhost:8000",
   issuerBaseURL: "https://dev-sma7reoctm83q6qn.us.auth0.com",
   tokenSigningAlg: "RS256"
})

export default jwtCheck;
```

prismaConfig.js

```
import { PrismaClient } from '@prisma/client'
const prisma = new PrismaClient()

export {prisma}
```

CHAPTER 7

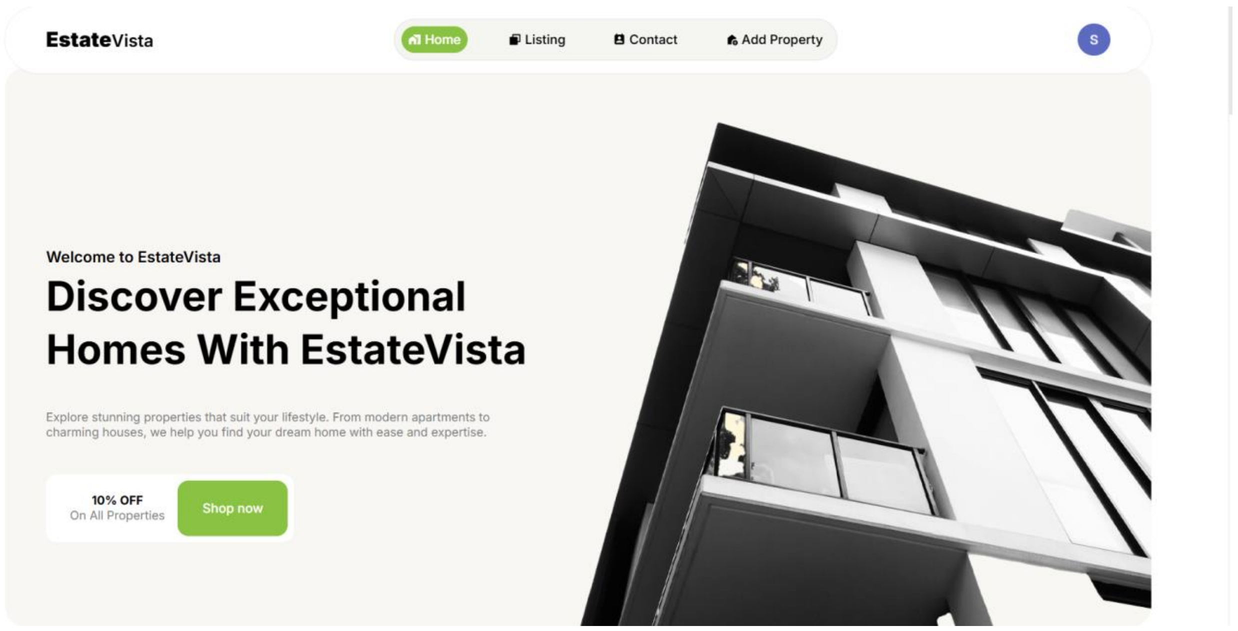
RESULTS

7.1 Screens and Explanations

This chapter will include all the screens available in the project such as home page, registration page, login page, course, quizzes and certification along with detailed explanation of each screen and its functionality. Screens available in the system are as follows:

Screen 1: Home Page

Screen 1 is the home page of the website which displays the hero section of the website.



Welcome to EstateVista

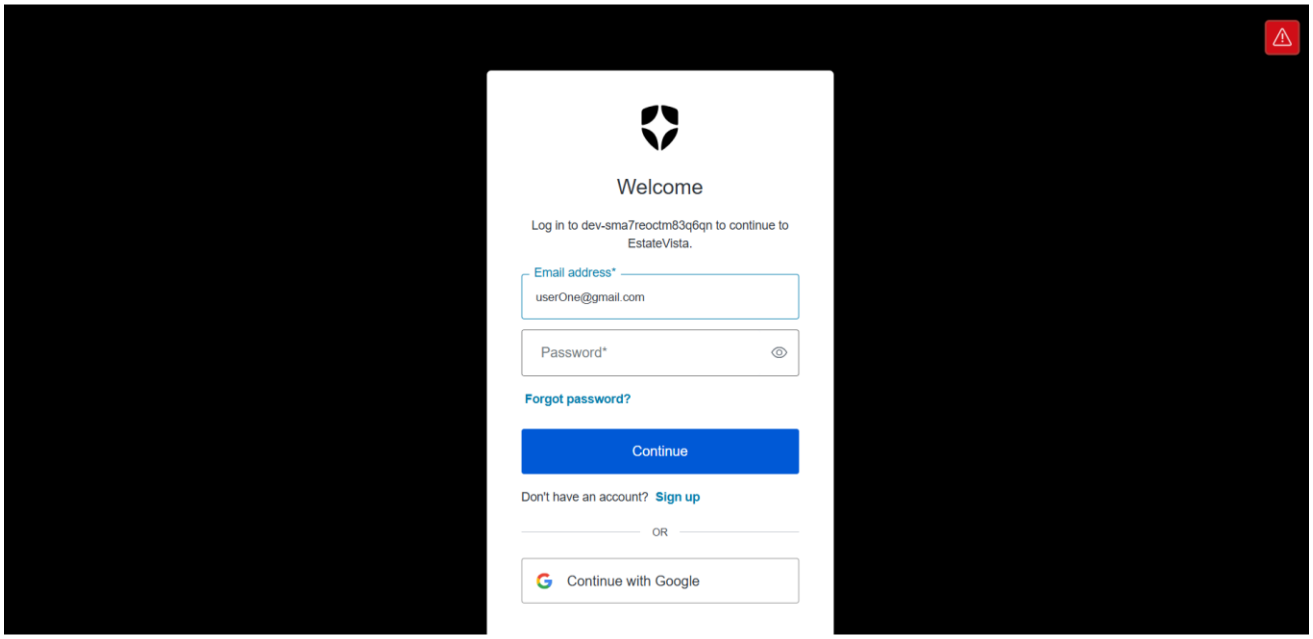
Discover Exceptional Homes With EstateVista

Explore stunning properties that suit your lifestyle. From modern apartments to charming houses, we help you find your dream home with ease and expertise.

Fig. 7.1 Home Page

Screen 2: Login and Registration Screen

Screen 2 is the log in and the registration page. Where if the user is new to the system, then he or she can register themselves to the system by providing the email and password. Password validation is also done at the time of registration. If the user is not new or already registered to the system, then he or she can directly log in to the system by proving some credentials such as email and password. The user can toggle between the login and the registration page we have used the Auth0 to implement the secure authentication.


Welcome

Log in to dev-sma7reoctm83q6qn to continue to EstateVista.

Email address*: userOne@gmail.com

Password*

Fig. 7.2 Login and Registration Screen

99Screen 3: Listing

Screen 3 showcases the properties listed in the website. It is designed to present all relevant property information in an organized, visually appealing, and user-friendly manner.

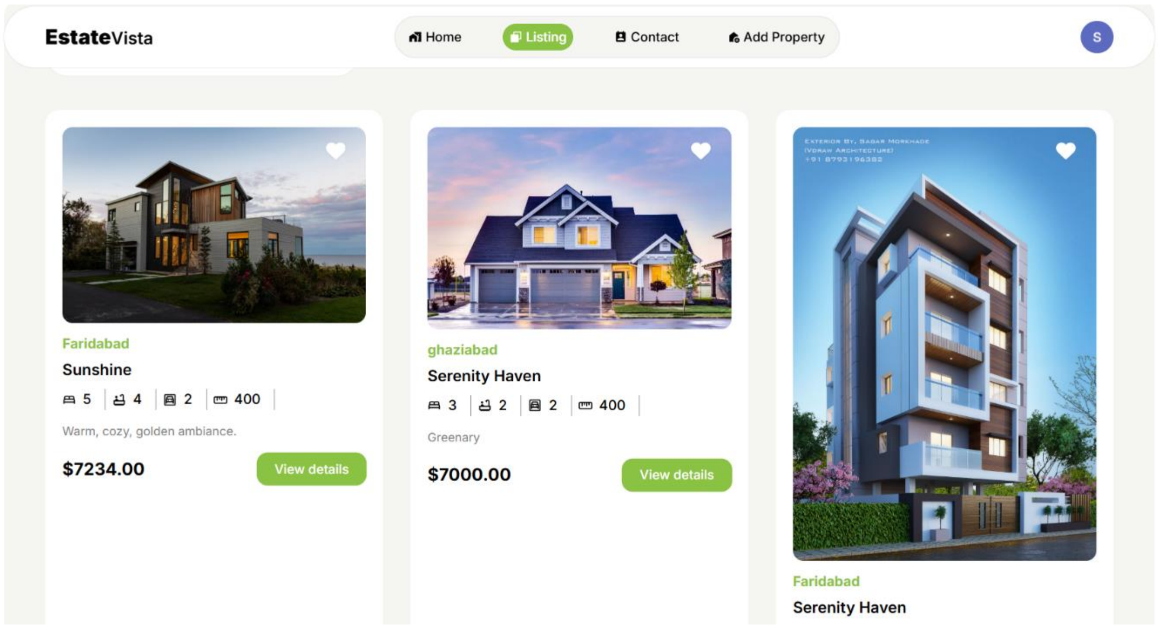


Fig. 7.3 Listing

Screen 4: Add property

Screen 4 Add **Property** feature allows users to effortlessly list their properties on the website. This intuitive, user-friendly tool ensures that sellers and agents can showcase their properties with ease.

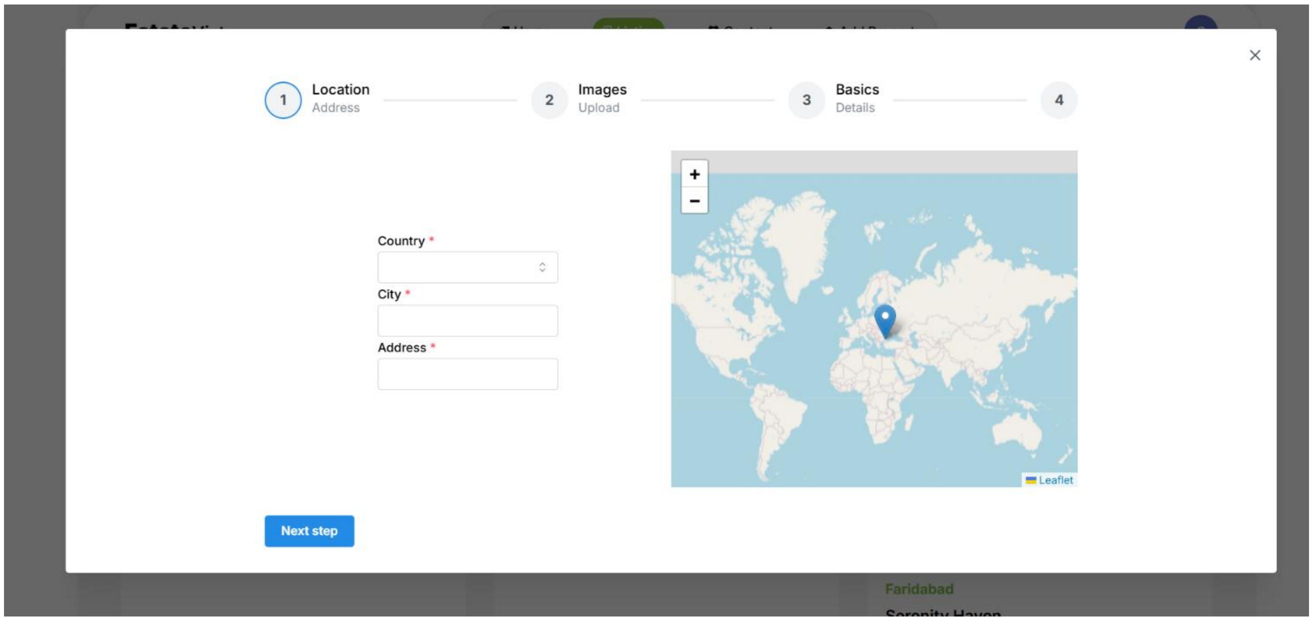
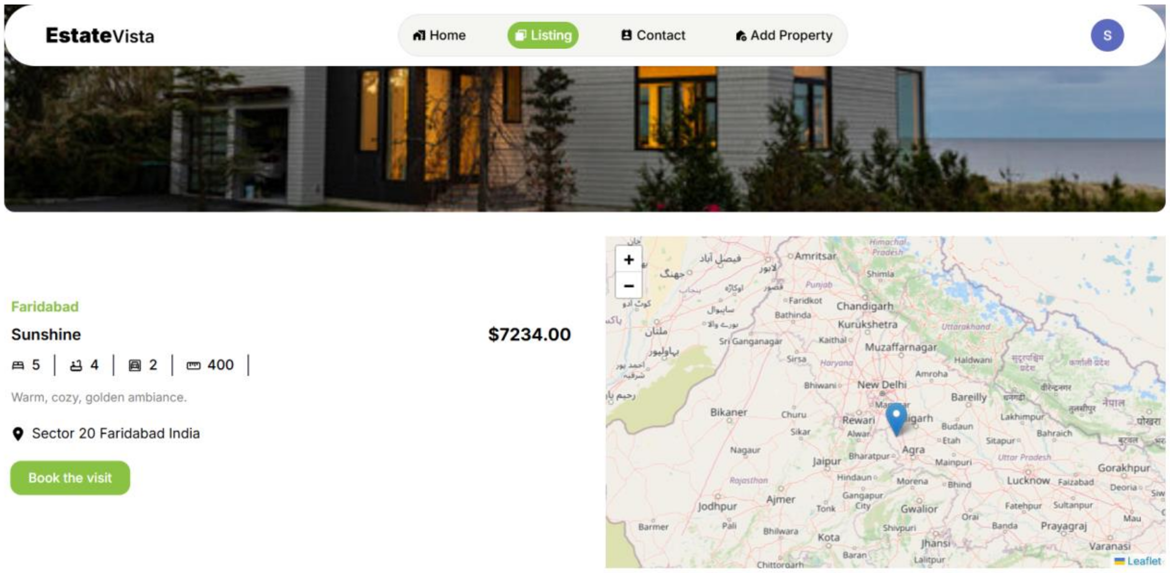


Fig. 7.4 Add property

Screen 5: Details

Screen 5 Detail View feature provides a deep dive into every essential aspect of a property, ensuring that potential buyers and visitors have all the necessary information to make well-informed decisions



Faridabad

Sunshine $7234.00

5 | 4 | 2 | 400 |

Warm, cozy, golden ambiance.

Sector 20 Faridabad India

Book the visit

CHAPTER 8

DISCUSSIONS

The EstateVista platform represents a significant step forward in simplifying property management, discovery, and transactions in the real estate domain. By employing modern web technologies such as the MERN stack, the platform ensures efficiency, scalability, and a user-friendly experience. This discussion highlights the performance of EstateVista, its contributions to the real estate market, and areas for potential enhancements.

8.1 Performance

EstateVista has proven to be a reliable and efficient platform due to the following factors:

- **Scalability**
  The MERN stack ensures that the platform can handle increasing property listings and user interactions without compromising performance. The use of MongoDB allows for flexible and efficient storage of user and property data.
- **User-Friendly Interface**
  React.js ensures a responsive and visually appealing frontend. Users can seamlessly navigate through features like property listings, search filters, and booking systems across devices, enhancing accessibility and engagement.
- **Real-Time Features**
  The integration of map-based property visualization and dynamic search filtering ensures real-time updates, enabling users to make decisions based on the latest property data.
- **Security**
  Auth0-powered authentication ensures secure access for all users, safeguarding personal data and creating a trustworthy environment for property transactions.
- **Quick Search and Filtering**
  Advanced search functionalities allow users to refine property searches based on multiple parameters, delivering fast and accurate results tailored to individual preferences.
- **Robust Backend**
  The use of Node.js and Express.js ensures smooth backend operations, handling data requests efficiently and reducing system lag.

8.2 Research Directions

1. **User Feedback System**
Add a feedback feature where users can rate their experience with the platform or leave suggestions. This will help identify areas for improvement.

2. **Multilingual Support**
Enable the platform to support multiple languages to cater to users from diverse regions, making it more inclusive.

3. **Mobile App Development**
Create a lightweight mobile app for EstateVista to ensure better accessibility and reach more users**.**

4. **Enhanced Search Filters**
Introduce additional filters like "nearby schools," "pet-friendly properties," or "commute time" to make property searches more specific.

5. **Offline Access**
Implement a feature to save property details for offline viewing, making it convenient for users with limited internet access.

6. **Simple Chat Feature**
Add a basic chat system for direct communication between buyers, sellers, and agents to facilitate easier transactions**.**

7. **Local Area Insights**
Provide basic information about neighbourhoods, like safety ratings, nearby amenities, and average property prices, to help users make informed decisions.

CHAPTER 7

CONCLUSION

The EstateVista project successfully addresses the challenges of traditional real estate practices by offering a digital platform that is efficient, user-friendly, and feature-rich. By integrating tools like advanced search filters, interactive mapping, and visit scheduling, the platform simplifies property discovery and management for buyers, sellers, and renters.

The project demonstrates the power of leveraging modern technologies like the MERN stack and Prisma to create a scalable and responsive system accessible on multiple devices. Its secure user authentication, real-time updates, and tailored property recommendations enhance user experience and foster trust among stakeholders.

Looking ahead, EstateVista has immense potential for growth. Simple enhancements such as multilingual support, a mobile app, and user feedback systems can make the platform even more accessible and engaging. Features like local area insights, improved search filters, and a basic chat system can further enrich the user experience, ensuring EstateVista stays ahead in the competitive real estate market.

In conclusion, EstateVista is a comprehensive solution that bridges the gap between technology and real estate, empowering users with tools to navigate the complexities of property transactions effortlessly. It paves the way for a smarter, more streamlined real estate experience while setting the stage for future advancements.

CHAPTER 8

REFERENCES

1 **MongoDB Documentation**: https://www.mongodb.com/docs/

2 **React.js Documentation**: https://reactjs.org/docs/

3 **Express.js Guide:** https://expressjs.com/

4 **Node.js Documentation:** https://nodejs.org/en/docs/

5 **Prisma Documentation:** https://www.prisma.io/docs/

6 **Auth0 Developer Resources:** https://auth0.com/docs

7 **Google Maps API Documentation**: https://developers.google.com/maps/documentation

8 **Bootstrap Documentation:** https://getbootstrap.com/docs/

9 **W3Schools Tutorials on Full-Stack Development:** https://www.w3schools.com/

10 **Real Estate Industry Insights – Zillow Research**: https://www.zillow.com/research/

11 **Digital Trends in Real Estate**: https://www.nar.realtor/reports

CHAPTER 9

BIBLIOGRAPHY

1. **Books**

- Baker, T. (2018). *The Future of Real Estate Technology: Innovations and Applications*. Springer.
- Liu, Z., & Chen, Y. (2021). *Building Scalable Applications with MERN Stack*. Wiley-Blackwell.
- Williams, M., & Donovan, C. (2021). *User Experience in Web Development: Principles and Practices*. Pearson.

2. **Articles and Journals**

- Garrison, D. R., & Akyol, Z. (2011). *Digital Tools for Enhanced User Engagement in Web Applications*. International Journal of Web Technologies.
- Hawkins, L., & Meyer, J. (2017). *The Impact of Modern Technologies in Real Estate Transactions*. Journal of Technology in Business.
- Smith, J., & Lee, K. (2019). *Interactive Map-Based Platforms for Property Listings: A Comparative Study*. Routledge.

3. **Websites**

- MongoDB Documentation: https://www.mongodb.com/docs/
- React.js Documentation: https://reactjs.org/docs/
- Node.js Resources: https://nodejs.org/en/docs/
- Auth0 Authentication Guide: https://auth0.com/docs/
- Google Maps API Documentation: https://developers.google.com/maps/documentation

 Reports and Industry Insights

- National Association of Realtors (2024). *Emerging Trends in Real Estate Technology*.
- Deloitte Insights (2023). *The Future of Real Estate: Digitization and Market Trends*.

4.**Online Tutorials**

- W3Schools. (2024). *Full-Stack Development with MERN Stack*: https://www.w3schools.com/
- Codecademy. (2024). *Developing User-Friendly Interfaces with React*: https://www.codecademy.com/