Arab International University

Faculty of Informatics and Communication Engineering

Senior Project Report on

TrustAsset System for Buying and Selling Real Estate and Vehicles

Department of Informatics Engineering

in partial fulfillment of the requirement for the Degree of Bachelor in

Informatics Engineering

Submitted by

Shaker Alkani	**Mohamed Ahmed**	**Muhammad Zaid Al Nahhas**
Dalia Mahayni	**Razan Alshaar**	**Assmaa Almoghrabi**

Under the Supervision of

Dr. Mohamad Mohamad **Eng. Raghad Arab**

June 2025

June 2025

Faculty of Informatics & Communication Engineering

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Informatics Engineering for acceptance, a project report entitled Project Title In English Submitted by: Shaker Al-kani, Mohamad Ahmad, Asmaa Al-Moghrabi, Dalia Mahayni, Razan Alshaar, Muhammad Zaid Al Nahhas in partial fulfilment for the degree of Bachelor of Engineering in Informatics.

Supervisor

Dr. Mohamad Mohamad

Eng. Raghad Arab

Head of Department

Dr. Rami Yard

Arab International University

The Arab International University (AIU) is a private Syrian university established in 2005. Its academic plans and the documents issued by it are approved and certified by the Ministry of Higher Education in the Syrian Arab Republic.

The university works to achieve the following goals:

- Preparing a distinguished generation of university graduates who are able to meet and advance the specific needs of society.
- Contributing to theoretical and applied scientific research that serves the purposes of national development. Work is being done to urge professors and academic staff to scientific research and participate in conferences and seminars that organize research.
- Achieving partnership with prestigious Arab and foreign universities with the aim of continuous development and modernization of academic work and conducting joint scientific research.
- Attracting distinguished academic and research competencies by providing the appropriate environment for their work.

The Arab International University is one of the first Syrian universities that have been established and inaugurated. It has been able to attract distinguished educational, research and administrative competencies, to create an integrated edifice from the academic, organizational and administrative aspects. It was able to graduate cadres of distinguished innovators by providing an educational environment based on unique qualitative and material ingredients, including:

- Modern and advanced study plans based on the credit hour system.
- Carefully selected educational cadres.
- Modern scientific laboratories and a laboratory for electronic libraries.
- Physical and moral incentives for students.
- Application of interactive teaching methods.
- Academic and educational guidance and counseling.
- A wide range of scientific cooperation agreements with local, regional and international universities of reputable reputation.
- Multiple agreements and memoranda of understanding with many civil society organizations.
- A proper campus with all the facilities of science, sports and entertainment, which we encourage you to visit and learn about their features.
- Student activities and clubs of all kinds: athletic, cultural, scientific and social.

The Arab International University life years are a time to invest in a student's future. The knowledge and experience that students acquire in the lecture hall and laboratories will help them in developing themselves. They will provide them with reasons for success in the chosen specialty. The student activities will help in expanding students' horizon. The activities of training, clubs and sports will enable students to develop their talents, and may even help in discovering new talents.

Students could invest time, mind and spirit in our university in order to reap the benefits of works and the time devoted in the coming years. We will be by our students in every step of their way.

Acknowledgements

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Dr.Mohamad Mohamad for his valuable guidance, encouragement and help for completing this work, his useful suggestions for this whole work and cooperative behavior are sincerely acknowledged.
We are also grateful to Eng. Raghad Arab for his constant support and guidance.

At the end we would like to express our sincere thanks to all our families, friends and others who helped us directly or indirectly during this project work.

Abstract

"TrustAsset" is an advanced digital marketplace designed to simplify and secure the process of buying and selling real estate and vehicles. The platform addresses the complexities of high-value transactions by combining a robust software framework, built to ensure trust and efficiency, with a suite of intelligent features. The system introduces unique functionalities that set it apart in the digital marketplace, including **agent-mediated listings** to ensure verification, a **secure transaction workflow** using private keys for digital signatures, and a comprehensive system for managing and trading **shared or fractional ownership** of assets, which provides novel flexibility for investors and co-owners.

To further enhance the user experience, "TrustAsset" is powered by three core AI components that provide practical, data-driven assistance. This includes an **image-based search** tool that allows users to find vehicles without needing technical keywords, a data-driven **price prediction** engine that promotes fair and transparent market valuations, and a **personalized hybrid recommendation system** that suggests relevant properties and cars by solving key industry challenges like data sparsity and the "cold start" problem. By integrating these core software features with practical AI tools, the "TrustAsset" platform delivers a comprehensive, trustworthy, and user-centric solution that streamlines the entire buying and selling lifecycle.

Table of Contents

List of Figures

List of Tables

Abbreviations

AIU	Arab International University
AI	Artificial Intelligence
API	Application Programming Interface
ANN	Approximate Nearest Neighbor
CNN	Convolutional Neural Network
DFD	Data Flow Diagram
ERD	Entity-Relationship Diagram
JWT	JSON Web Token
k-NN	k-Nearest Neighbors
MAE	Mean Absolute Error
MSE	Mean Squared Error
OCR	Optical Character Recognition
R^2	R-squared
SQL	Structured Query Language
T-SQL	Transact-SQL
UI	User Interface
UX	User Experience

Keywords

Real Estate, Automotive, E-Commerce, Digital Marketplace, Artificial Intelligence, Recommendation System, Collaborative Filtering, Content-Based Filtering, Hybrid Model, Price Prediction, Image-Based Search, Deep Learning, Machine Learning, Xception, Autoencoder, Gradient Boosting, Data Sparsity, TrustAsset.

Introduction

In an era where digital transformation is reshaping industries and daily transactions, the **"TrustAsset"** project emerges as a pioneering step toward redefining how high-value assets such as cars and real estate are bought and sold. This project aims to develop a comprehensive AI-powered online marketplace that simplifies and secures complex transactions through intuitive interfaces and intelligent automation.

By leveraging the latest web technologies and deep learning models, the platform facilitates seamless interaction between sellers and buyers through advanced search filters, fraud detection mechanisms, personalized recommendations, and instant document verification systems.

The platform is distinguished by its integration of innovative AI-powered features, including **image-based search** that enables users to find visually similar cars, **price prediction algorithms** based on market analysis for fair and accurate pricing, and a **personalized recommendation system** that analyzes user preferences and behaviors to offer tailored listings.

Additionally, **TrustAsset** includes essential features such as secure messaging, dynamic pricing tools, and a flexible technical architecture that supports access via both apps and web platforms.

By harnessing the power of artificial intelligence and data-driven insights, the project enhances **trust, transparency, and efficiency** in a traditionally fragmented market. Ultimately, the platform seeks to empower users within a reliable digital environment that accelerates transactions, improves decision-making, and establishes new standards for e-commerce in the automotive and real estate sectors.

Chapter 1: Project Description

1.1 Background

The rapid growth of e-commerce has fundamentally transformed traditional buying and selling processes, offering unprecedented convenience and accessibility to consumers worldwide. In today's digital economy, businesses and consumers increasingly rely on online platforms to conduct high-value transactions. However, the purchasing of significant assets like vehicles and real estate often remains tethered to legacy processes involving physical visits, lengthy negotiations, and complex paperwork.

This project, "TrustAsset," aims to bridge this gap by developing a secure, user-friendly digital marketplace. The platform is designed to streamline these complex transactions, creating a more transparent and hassle-free experience. Furthermore, this project seeks to move beyond the static "browse" experience common on many platforms. By integrating artificial intelligence, our goal is to transform the user journey into a dynamic and responsive discovery engine that proactively guides users to assets matching their implicit and explicit needs, thereby enhancing user satisfaction and driving engagement.

1.2 Problem Statement

The traditional methods for buying and selling cars and properties are often inefficient, time-consuming, and geographically limited. Sellers, including private owners and professional agents, face challenges in effectively promoting their listings to a broad audience. Concurrently, the modern online marketplace presents consumers with a "paradox of choice." A vast inventory, while offering options, frequently leads to an overwhelming and complex decision-making process, which can cause user fatigue and abandonment of the purchasing journey.

The core challenge this project addresses is twofold: first, to create a trusted, centralized platform that simplifies the transactional workflow; and second, to provide users with intelligent tools to navigate the large, complex datasets of listings. This requires a solution that not only handles the transactional mechanics but also tackles issues of data quality, feature complexity, and scalability to deliver relevant, personalized recommendations.

1.3 Project Objective

The primary objective of this project is to develop "TrustAsset," a secure, scalable, and user-friendly web and mobile application that facilitates the seamless buying and selling of cars and real estate properties. The platform aims to centralize the marketplace, enhance the user experience with intelligent features, and facilitate faster, more secure transactions.

To achieve this, the project will deliver:

1. A robust software platform with core functionalities for agent-mediated listings, shared ownership management, and secure, key-based transactions.
2. Three integrated AI engines to enhance the user experience:
   - An **Image-Based Search** tool for vehicles.
   - A data-driven **Price Prediction** model for accurate market valuations.
   - A **Personalized Recommendation System** for both properties and vehicles.

1.4 Project Scope

This project encompasses the end-to-end design and development of the "TrustAsset" digital marketplace. The scope includes the creation of the user-facing mobile applications (for iOS and Android) and the web-based portal for administrators and agents. It also covers the design and implementation of the secure backend infrastructure required to support the platform's functionalities.

A significant part of the project's scope is the full lifecycle of the three core AI models. This includes data collection and preprocessing, model training and evaluation, and the development of a RESTful API service to integrate the AI features seamlessly into the main application. The project will validate these models against real-world datasets for both vehicles and real estate to ensure their accuracy and reliability.

1.5 Project Features

The "TrustAsset" platform incorporates a rich set of software and AI-powered features designed to create a comprehensive and trustworthy user experience.

Core Platform & Transaction Features:

- **Agent-Mediated Listings:** Sellers submit listing requests to verified agents who handle the verification and publishing process, adding a layer of trust.
- **Secure Purchase Workflow:** A multi-step purchase process requiring buyers and sellers to confirm the transaction using unique private keys for digital signatures.
- **Shared Ownership Management:** The platform supports fractional ownership, allowing multiple users to co-own an asset with defined percentage shares.
- **Individual & Group Sales:** Co-owners have the flexibility to sell their individual share or initiate a group sale request that requires approval from all involved parties.
- **Automated Ownership Transfer:** Upon a successful purchase, the system uses a secure database procedure to automatically update and transfer ownership shares.

AI-Powered Features:

- **Vehicle Search by Image:** Users can upload a photo of a car to find similar vehicles listed for sale.
- **AI Price Prediction:** Provides an estimated market price for any property or car based on its specific details.
- **Personalized Recommendations:** The system learns from user behavior (searches, favorites) to proactively suggest relevant listings.

1.6 Project Feasibility

The proposed "TrustAsset" marketplace demonstrates strong feasibility across technical, operational, and economic dimensions.

- **Technical Feasibility:** The system will leverage a modern technology stack, including scalable cloud architecture (AWS/Azure), cross-platform development frameworks (Flutter, React), and established AI/ML libraries (TensorFlow, Scikit-learn). This ensures that the implementation is achievable and maintainable.
- **Operational Feasibility:** The agent-mediated workflow is designed to be managed through a dedicated web portal, ensuring that operational processes are streamlined. The automated features will reduce the need for manual intervention.
- **Economic Feasibility:** The platform can be monetized through various models, such as listing fees, commissions on successful sales, or premium subscription features for agents and power users, providing a clear path to sustainability and profitability.

1.7 System Requirements

To ensure optimal performance, security, and user experience, the "TrustAsset" platform requires the following high-level system specifications:

Hardware Requirements:

- A cloud-based infrastructure with scalable computing resources.
- High-performance servers or serverless functions for AI/ML model inference.
- Scalable, secure database services (e.g., SQL Server).
- Dedicated, high-availability media storage for images and documents.
- A Content Delivery Network (CDN) for fast global delivery of assets.

Software Requirements:

- **Frontend (Mobile):** Flutter SDK for cross-platform iOS and Android development.
- **Frontend (Web):** React.js for the administrative and agent web portal.
- **Backend:** ASP.NET Core 8 for building the main application logic and RESTful APIs.
- **AI Service:** A Python environment with FastAPI for the AI model API.
- **Database:** Microsoft SQL Server.
- **AI/ML Libraries:** Python with TensorFlow, Keras, Scikit-learn, Pandas, and NumPy.

Chapter 2: Theoretical Study

2.1 Digital E-Commerce Overview

Digital e-commerce refers to the buying and selling of goods and services over the internet, powered by online platforms that facilitate transactions between businesses and consumers (B2C), businesses and other businesses (B2B), or peer-to-peer (C2C). In the context of this project, the "TrustAsset" Digital Marketplace represents a specialized e-commerce platform designed for high-value transactions involving vehicles and properties.

2.1.1 Key Characteristics of Digital E-Commerce

- **Online Storefronts:** Sellers can create digital listings with detailed descriptions, images, videos, and pricing.
- **Secure Transactions:** Integration with payment gateways (credit cards, digital wallets, bank transfers) and escrow services enhances safety.
- **Search & Discovery:** Advanced filters, AI recommendations, and geolocation-based searches help buyers find desired products efficiently.
- **Automation & AI:** Chatbots, pricing algorithms, fraud detection, and document verification are used to streamline operations.
- **Mobile & Cross-Platform Access:** Web and mobile apps ensure accessibility from any device, at any time.
- **Data-Driven Insights:** Analytics are used to track sales performance, customer behavior, and market trends.

2.1.2 Benefits of Digital E-Commerce

- **Convenience:** Buyers can browse and purchase anytime, anywhere.
- **Wider Reach:** Sellers can access a global or local customer base without the limitations of physical stores.
- **Cost Efficiency:** Reduces overhead costs associated with traditional brick-and-mortar businesses.
- **Transparency:** Verified listings, customer reviews, and secure payments build trust between parties.

- **Scalability:** Cloud-based infrastructure supports business growth and increasing transaction volumes.

2.2 The Impact of the Real Estate Market on the Global Economy

The real estate market is a cornerstone of the global economy, influencing economic growth, employment, financial stability, and wealth distribution. Its impact extends across multiple sectors, from construction and banking to government revenues and consumer spending. Real estate contributes significantly to the Gross Domestic Product (GDP) of developed economies and is a major driver of employment, supporting jobs for construction workers, real estate agents, brokers, appraisers, and legal professionals.

Furthermore, the real estate market is deeply intertwined with financial markets. Mortgages are a fundamental component of the banking system, and fluctuations in the housing market can have far-reaching consequences, as evidenced by the 2008 financial crisis. For governments, property taxes, stamp duties, and capital gains taxes are critical sources of revenue.

2.3 Core Concepts in Machine Learning

This section covers the fundamental algorithms and statistical concepts that form the theoretical basis for the project's AI features.

2.3.1 Similarity and Distance Metrics

Cosine Similarity: A measure of similarity between two non-zero vectors that calculates the cosine of the angle between them. It is particularly effective in recommendation systems as it quantifies how closely two items align in terms of user ratings or attributes, regardless of their magnitude. This was the primary metric used in the k-NN model to find similar vehicles based on user rating patterns.

2.3.2 Machine Learning Algorithms

k-Nearest Neighbors (k-NN): An algorithm used for classification and regression, which can also be adapted for similarity-based recommendation. As implemented by Scikit-learn's NearestNeighbors class, it finds a predefined number of training samples closest in distance to

a new point. In this project, it was configured with the cosine metric and a 'brute' force algorithm to find the k-most similar vehicles based on their rating vectors in the user-item matrix.

Ensemble Methods for Regression: This project heavily utilized gradient boosting, an ensemble technique that builds a strong predictive model by sequentially adding weak learner models (typically decision trees).

XGBoost (Extreme Gradient Boosting): A powerful and highly efficient implementation of gradient boosting, known for its performance and accuracy. It was a key algorithm used in the price prediction models.

LightGBM (Light Gradient Boosting Machine): An open-source gradient boosting framework designed for speed and efficiency, especially with large datasets, due to its use of histogram-based techniques. It was also employed in the price prediction module.

CatBoost (Categorical Boosting): A high-performance gradient boosting framework that excels at handling categorical features natively, reducing the need for extensive preprocessing. It was the third ensemble method used in the final price prediction models.

2.4 Deep Learning Architectures

This section describes the neural network architectures used for the more complex tasks of image recognition and feature learning.

2.4.1 Convolutional Neural Networks (CNNs) for Image Recognition

CNNs are a class of neural networks specifically designed for processing and analyzing visual imagery. This project leveraged several well-established CNN architectures for the vehicle image search feature.

VGG-16 & VGG-19: Architectures from the Visual Geometry Group (VGG) at the University of Oxford. They are characterized by their simplicity and depth, using small 3x3 convolutional filters stacked on top of each other. Their effectiveness in learning hierarchical visual features made them strong candidates for feature extraction.

Xception (Extreme Inception): A deep learning model developed by Google that refines the Inception architecture by using depthwise separable convolutions. This allows it to learn feature

correlations more efficiently and achieve superior performance on large, complex image datasets, making it the final choice for the image recognition model.

2.4.2 Autoencoders for Feature Learning

An Autoencoder is a type of neural network used for unsupervised learning to create efficient data representations. It consists of two parts: an encoder that compresses the input data into a lower-dimensional "bottleneck" layer (the embedding), and a decoder that attempts to reconstruct the original input from this compressed representation. In this project, an Autoencoder was trained on property features to generate rich, dense embeddings, which were then used to calculate property similarities for the content-based recommendation system.

2.5 Technology Stack and Development Libraries

The implementation of the theoretical models was made possible by a range of open-source libraries and frameworks from the Python ecosystem.

2.5.1 Data Science and Machine Learning Libraries

Pandas: The primary tool used for loading, cleaning, transforming, and manipulating the datasets throughout the project.

NumPy: The fundamental package for scientific computing, utilized for efficient numerical operations on arrays and matrices during data preprocessing and analysis.

Matplotlib & Seaborn: Used together for data visualization. Matplotlib served as the primary tool for creating plots and charts, while Seaborn was used for more sophisticated statistical graphics like heatmaps.

Scikit-learn: A comprehensive machine learning library that provided essential tools for this project, including:

- **StandardScaler** for standardizing features.
- **LabelEncoder** for converting categorical features into a numerical format.
- **NearestNeighbors** for implementing the collaborative filtering model.

SciPy: A library for scientific and technical computing that provided specialized tools:

- **Box-Cox Transformation** for normalizing highly skewed numerical features.
- **scipy.sparse.csr_matrix** for creating a memory-efficient sparse matrix to represent the user-item interactions.

TensorFlow/Keras: The end-to-end machine learning platform and its high-level API used to design, build, and train the project's deep learning models, including the CNNs and the Autoencoder.

2.5.2 Supporting Libraries and Deployment Frameworks

Regular Expressions (re): A standard Python library used for parsing the unstructured Vehicle_Title text to extract structured attributes like year, make, and model.

Pickle: A standard Python module used to serialize the trained models and their associated components into a single artifact file for easy loading and deployment.

Flask: A lightweight and minimalist web framework initially used to develop and prototype the recommendation API.

FastAPI & Pydantic: The final choice for the deployment API. FastAPI is a modern, high-performance web framework used for building the RESTful API service, while Pydantic was used to enforce data schemas for API requests and responses, ensuring data integrity.

Chapter 3: Literature Review

This chapter provides a comprehensive review of existing literature and systems related to the core components of the "TrustAsset" project. It examines previous research in the areas of image-based vehicle classification, price prediction, and personalized recommendation systems. Additionally, it analyzes similar applications in the market to identify existing solutions and areas for innovation.

3.1 Related Work and Previous Studies

The development of the AI features in this project is informed by a wide body of existing research. This section summarizes key studies and datasets that have been influential in each of the three core AI domains.

3.1.1 Studies in Image-Based Vehicle Recognition

Fine-grained vehicle classification is a challenging computer vision task that aims to distinguish between different car models. Several deep learning architectures have been evaluated for this purpose, achieving high accuracy on standard datasets.

Table 1 Referential Studies in Image-Based Vehicle Recognition

Reference	Name	Method(s)	Dataset	Accuracy	Year
[1]	A Systematic Evaluation of Recent Deep Learning Architectures for Fine-Grained Vehicle Classification	ResNet-50, ResNet-152, VGG16, DenseNet-161	Stanford Cars-196	94.6% (DenseNet-161)	2018
[2]	VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION	VGG16, VGG19	ImageNet	92.7% (VGG-16, top-5)	2015

[3]	A Large-Scale Car Dataset for Fine-Grained Categorization and Verification	AlexNet, GoogleNet, CNN feature + SVM	CompCars Dataset	91.2%	2015

These studies demonstrate the effectiveness of deep convolutional neural networks (CNNs) for image-based classification tasks. Models like VGG, ResNet, and DenseNet have consistently shown high performance, providing a solid foundation for developing the image-based search feature in our project.

3.1.2 Studies in Price Prediction

Machine learning has been widely applied to predict prices in both the automotive and real estate markets. These models typically use regression techniques to analyze a wide range of features and estimate a fair market value.

Table 2 Referential Studies in Price Prediction

Reference	Name	Method	Dataset	Source	Year
[4]	Car Price Prediction Using Machine Learning	K-Nearest Neighbors (KNN) Classification and Regression Trees (CART)	Not Mentioned	DOI: https://doi.org/10.26438/ijcse/v7i5.444450 Available online at: www.ijcseonline.org	2019
[5]	Artificial intelligence algorithms to	ElasticNet, XGBoost, Linear Regression	House Prices - Advanced Regression	kaggle	2021

	predict Italian real estate market prices		Techniques		
[6]	The Use of Machine Learning in Real Estate Research	K-Nearest Neighbors (KNN) Random Forest (RF) Extra Trees (ET)	private housing estates (Grand Promenade, Kornhill Garden, Les Saisons, Taikoo Shing)	Land Journal (MDPI) DOI: 10.3390/land12040740	2023
[7]	Car Price Prediction Using Artificial Neural Networks: A Data-Driven Approach	K-Nearest Neighbors (KNN) Artificial Neural Network (ANN) XGBoost Random Forest (RF)	The University of California, Irvine C.A. Center (UCI) for Intelligent Systems provided the dataset used in	Ind. Journal on Computing: socj.telkomuniversity.ac.id/indojc	2024

			this project.		
[8]	What Drives House Prices? A Linear Regression Approach to Size, Condition, and Features	Linear Regression	The dataset appears to be related to housing market data from Washington State. However, the specific source is not explicitly stated.	Journal of Artificial Intelligence and Data Mining (JAIDM): http://jad.shahroodut.ac.ir	2025

The literature indicates that ensemble methods like XGBoost and Random Forest, as well as Artificial Neural Networks, are highly effective for price prediction tasks. These findings guide the selection of algorithms for the "TrustAsset" price prediction module.

3.1.3 Studies in Personalized Recommendation Systems

Recommendation systems are crucial for navigating large inventories. Research in this area has explored various techniques, including content-based filtering, collaborative filtering, and hybrid approaches, to deliver relevant suggestions.

Table 3 Referential Studies in Personalized Recommendation Systems

Reference	Year	Data	Preprocessing	Model	Result
[9]	2025	Spatial distances (health, education, security, etc.) + housing characteristics (price, size, age)	Normalization, logarithmic transformation, factor analysis, dimensionality reduction (PCA)	Fuzzy C-Means (FCM) Clustering	Identified 4 clusters (Jambu's elbow method); improved segmentation accuracy.
		Geospatial data (QGIS) + web-scraped housing features	Spatial distance calculation, variable transformation	Bat Algorithm (Meta heuristic Optimization)	Execution time: 1.38–1.42 seconds (fastest); reduced search time by 15% vs. PSO/GA.
		User preferences (sliders) + housing clusters	Cosine similarity, graph-based memory	Hybrid Recommender System (Content-Based + Collaborative Filtering)	Precision: 81.6% (improved relevance), Recall: 89% (retrieved 89% relevant items).

		Spatial factors (latent variables) + user ratings	Fuzzy membership assignment, ANOVA validation	Spatial Fuzzy Logic Model	User Satisfaction: 58% (2.9/5), highlighting need for collaborative filtering
[10]	2024	Real-world data (3M users, 1.1M properties)	Cohort creation (locality, price/area bins), scoring (leads, conversions).	Rule-Based Engine	Latency <40 ms *(Ensured real-time cold-start recommendations)*
			Binning (price, area), activity weighting (Table 1), binary encoding.	Content Filtering	MAP@6 = 0.881 (Buy) *(Improved short-term precision by 3.2%)*
			Interaction scoring (CRF=10, scrolling=1), ALS training (daily).	Collaborative Filtering (ALS)	NDCG = 0.685 (Buy) *(Optimized ranking for long-term preferences)*
			Combined preprocessing (content + collaborative steps).	Hybrid Model	Latency = 29.3 ms *(Balanced accuracy + speed for hybrid users)*

[11]	2023	50 questionnaire entries (14 criteria)		Cosine-based similarity	63.8%
				Adjusted Cosine similarity	70.4%
				Pearson correlation	88.7%
				Spearman correlation	75.57%
[12]	2020	Hybrid Recommender: - User data (1,873 survey records) - Item data (98 vehicle models)	- Null value removal - Categorical-to-numeric conversion - Feature engineering	Multiclass Neural Network (1 hidden layer, 1,000 nodes, 160 iterations)	96% accuracy (Improved recommendation accuracy by capturing linear/non-linear data patterns)
		NLP Sentiment Analysis: - 14,000 labeled Twitter reviews	- Tokenization - Lemmatization - Stop-word removal - Text cleaning	Naive Bayes Classifier	92.73% test accuracy (Improved sentiment classification via labeled tweet analysis)

[13]	2019	Data1: 1-week sale data (Sydney)	Filtered items with <2 interactions	Stage 1 (Content -based)	Precision@5: 1.65% *Improved baseline (1.03%) by 60%*
			One-hot encoding for categorical/numerical features		
			Imputed missing prices		
		Data2: 2-week sale data (Sydney) Scenario 1/2	Same as Data1		Precision@5: 3.75% *Improved baseline (2.28%) by 64.5%*
			Time-based split for Scenario 2		
		Data2: 2-week sale data (Sydney) Scenario 3 (Cold-start)	Training/test split to exclude new items in training	Full two-stage model	Recall@50: 4.71% *Baseline failed (0% recall)*
		Data3: 3-month data (user attributes)	Accumulated user features (e.g., persona, engagement level)	Stage 2 (XGBoost spam filtering)	Precision@5: Improved from 3.75% to 4.2% *Reduced spam notifications*
			One-hot encoding		

3.2 Similar Applications

A review of the competitive landscape reveals several platforms operating in the real estate and automotive e-commerce space. The following tables provide a comparative analysis of their features against the proposed "TrustAsset" system.

Comparison Table

Table 4 Similar Applications Features Comparisons

Feature	TrustAsset	Zillow (USA)	Aqqary	MAZADI
Automated Valuation Model (AVM)	Yes (AI Price Prediction)	Yes (Zestimate)	No (manual pricing by sellers)	No (manual pricing by sellers)
AI-Powered Recommendations	Yes	Yes (personalized suggestions based on user search behavior)	No	No
Transaction Management	Yes (secure workflow & ownership transfer)	No direct transactions (connects buyers & sellers)	No direct transactions (connects buyers & sellers)	No direct transactions (connects buyers & sellers)
Verified Listings	Yes (all listings are agent-verified)	Some verified listings	No (any user can post a property)	Some verified listings
Agent & Dealer Support	Yes	Yes (real estate properties can list properties)	Yes (real estate companies can list properties)	Yes (dealers can list vehicles)
Rental Listings	No	Yes	Yes	No
Mobile App Availability	Yes (iOS & Android)	Yes (iOS & Android)	Yes (iOS & Android)	Yes (iOS & Android)

This analysis reveals a significant opportunity for a platform like "TrustAsset," which aims to integrate advanced AI features such as image search, automated valuation, and personalized recommendations across both automotive and real estate domains within a single, unified experience.

Chapter 4: System Analysis

4.1 Functional Requirements

This section details the functional requirements of the "TrustAsset" platform, broken down by user roles and key scenarios. These requirements define the specific actions that users can perform and the system's expected behavior.

4.1.1 Asset Listing and Verification (Agent & Seller)

This scenario covers the process of a user (seller) submitting their asset and an agent verifying and listing it.

- **Seller:** After logging in, a user can upload the information, details, and legal documents for a car or property they wish to sell.
- **System (Duplicate Check):** The system validates the uniqueness of the asset using its Property Code or Chassis Number. If the asset is already listed, the system prevents the duplicate upload and notifies the user. If it is unique, a listing request is created.
- **Agent:** The real estate agent logs into their dashboard and sees pending listing requests, prioritized by proximity to their location.
- **Agent (Verification):** The agent reviews the details and documents submitted by the seller. They manually verify the accuracy of the information.
- **Agent (Approval/Rejection):** If the information is correct and consistent, the agent approves the request. Before final approval, the agent must define and distribute the ownership shares of the asset according to the legal documents. If the documents are inconsistent, the agent rejects the request.

The following flowchart provides a detailed visual representation of the asset listing and verification workflow. It illustrates the complete sequence of events, from the moment a seller uploads an asset's details and the system performs a duplicate check, to the point where an agent reviews the request, verifies the information, and makes a final decision to either approve or reject the listing. This diagram clarifies the decision points and control flow within this core process.

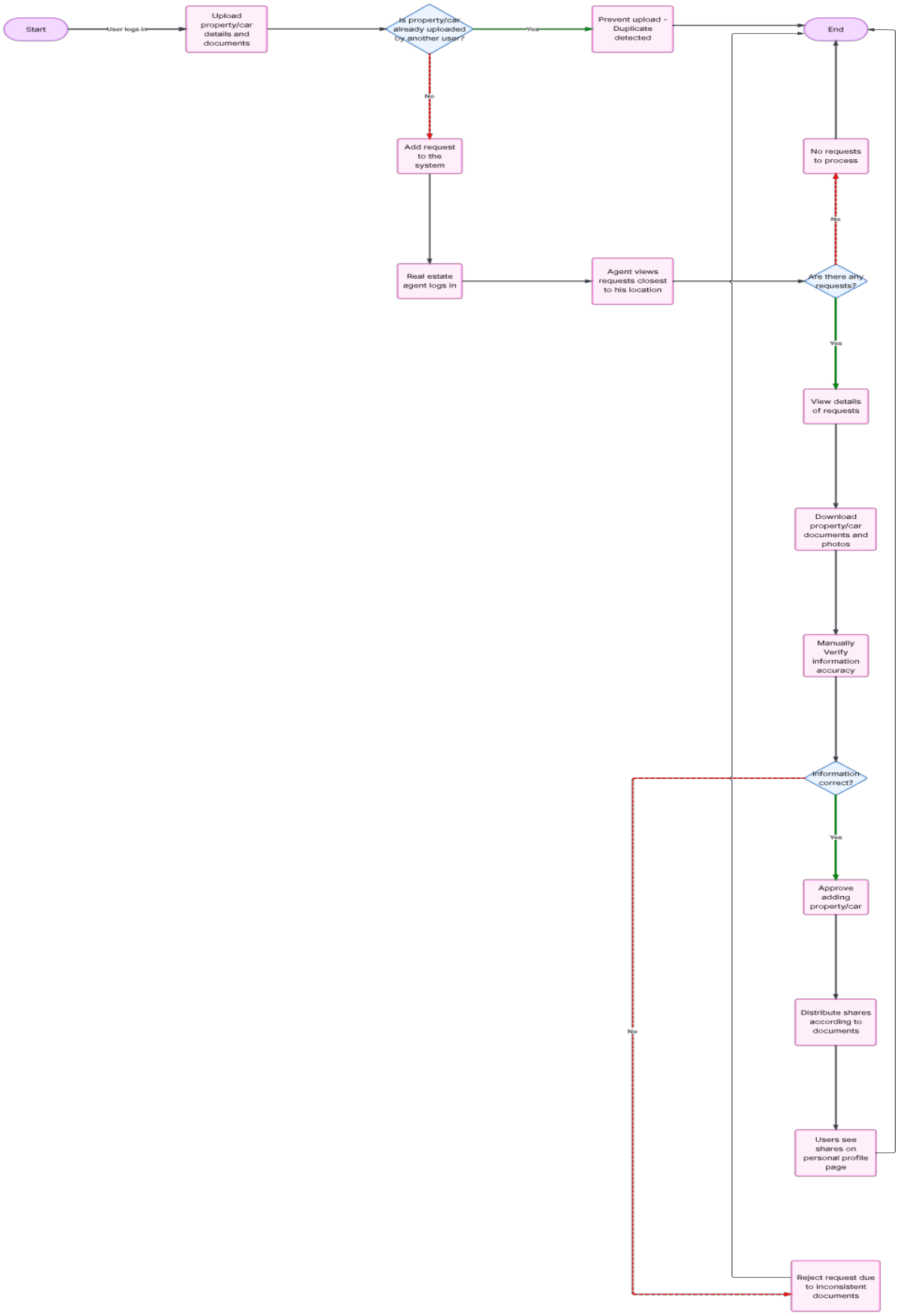


Figure 1 Process Flowchart for Asset Listing and Agent Verification

4.1.2 Shared Ownership and Sales (Customer/Co-owner)

This scenario describes how users with shared ownership can manage their assets and initiate sales.

- **View Shares:** A customer can log in, navigate to their personal profile, and view all the asset shares they own. They can see the details of each share, the property/car it belongs to, and a list of other co-owners.
- **Individual Sale Request:** From the share details page, a co-owner can create an **Individual Sale Request** for their personal share. This creates a new sales listing on the platform's home page for just that fraction of the asset.
- **Group Sale Request:** Alternatively, a co-owner can initiate a **Group Sale Request** to sell the entire asset. They select the other co-owners they wish to include in the sale.
- **System (Group Sale Notification):** The system sends a notification to all selected co-owners about the group sale proposal. If any co-owner rejects the request, or if there is no response from all parties within five minutes, the group sale process is automatically canceled. If all co-owners accept, a single sales listing for the entire asset is created.

To clarify the platform's unique shared ownership features, the following flowchart illustrates the workflow for creating a sale request. It visually maps the user's journey after logging in, showing the decision path that leads to creating either an "Individual Sale Request" for a personal share or a "Group Sale Request." For the group sale, the diagram details the subsequent steps of notifying co-owners, awaiting responses, and handling the success or cancellation of the request based on their collective decision.

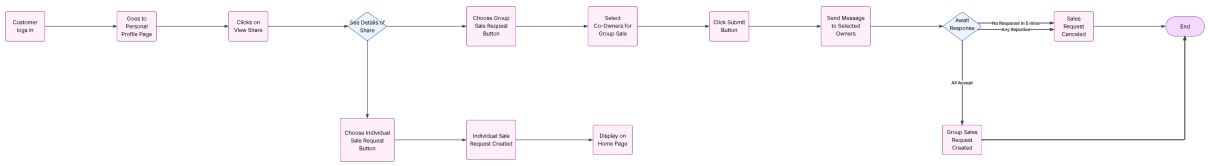


Figure 2 Process Flowchart for Creating a Sale Request

4.1.3 Asset Purchase Workflow (Buyer & Seller)

This scenario details the secure process for purchasing a listed asset.

- **Buyer (Initiate Purchase):** A customer browsing the platform can click the "Buy" button on any sales request.
- **System (Notification & Key Verification):** The system sends a notification to both the buyer and the seller(s), prompting them to enter their unique private key to authorize the transaction. If either party fails to enter the correct key or does not respond, the purchase process is canceled.
- **System (Funds Verification):** If all private keys are verified, the system checks the buyer's account to ensure sufficient funds are available. If not, the purchase is canceled.
- **System (Transaction Execution):** If funds are sufficient, the purchase is executed. This involves automatically transferring the ownership shares from the seller(s) to the buyer(s) in the database and generating a final electronic contract, digitally signed by all parties using their private keys.

4.1.4 Browsing and AI-Powered Features (User)

This scenario covers the user's browsing experience and interaction with the platform's AI engines.

- **Search:** Users can perform searches using various methods. They can search for cars by uploading an **image** or search for properties using **filters** such as the number of bedrooms, bathrooms, and geographical location.
- **Favorites:** Users can add any listing to their personal "Favorites" list for later viewing.
- **Recommendation Engine:** The system tracks user behavior, including search history, favorited items, and the time spent browsing each listing. This data is used to calculate user preferences and is sent to the AI server to retrieve recommendations for similar properties and cars, which are then suggested to the user.
- **Price Prediction:** Users can interact with the AI server to get a predicted market price for a specific property or car by providing its details.

4.2 Non-Functional Requirements

To ensure a high-quality and reliable system, the following non-functional requirements have been defined and addressed through specific design choices and technologies.

4.2.1 Security

The confidentiality and integrity of user data, documents, and transactions are paramount.

- **Authentication:** User sessions are secured using **JSON Web Tokens (JWT)**, ensuring that only authenticated users can access protected resources.
- **Transaction Authorization:** High-value operations, such as confirming a purchase, are authorized using a **private key system**, where users must provide their secret key to digitally sign and approve the transaction.
- **Data in Transit:** All communication between the client applications and the backend servers is encrypted using **HTTPS (SSL/TLS)**.
- **API Security:** The AI service endpoints are protected by requiring a static **API Key** in the request headers, preventing unauthorized access.

4.2.2 Performance

The platform is designed to be highly responsive with low latency.

- **Database Optimization:** Complex database operations, such as the transfer of ownership shares, are encapsulated in a **T-SQL Stored Procedure** (`ReplaceRequestShares`). This reduces network latency and ensures the entire transaction is executed efficiently on the database server.
- **API Optimization:** The AI service API is optimized by **pre-loading all models and necessary data artifacts into memory** on startup. This avoids slow disk I/O for each request, enabling near-instantaneous responses for recommendations and predictions.
- **Memory Management:** The Python API service was specifically optimized for low-RAM environments by using memory-efficient data types and explicitly calling the **garbage collector** after loading large datasets.

4.2.3 Availability

The platform must be highly available with minimal downtime.

- **Cloud Deployment:** The system is designed for deployment on a modern **cloud-based infrastructure** (e.g., AWS, Azure). This allows for the use of redundant components, load balancing, and automatic failover to ensure the application remains operational even if a single server fails.

4.2.4 Scalability

The system must handle a growing number of users, listings, and transactions.

- **Layered Backend Architecture:** The main backend application is designed using a **Layered Architecture**, separating concerns into distinct layers (e.g., Controller/API, Service, Data Access). This modular design enhances maintainability and allows different parts of the system to be updated or scaled independently.
- **Microservice Architecture:** The AI components are designed as a separate **microservice** (the Python/FastAPI service). This allows the AI-related workload to be scaled independently from the main backend application, ensuring that high computational demands do not affect the core platform's performance.
- **Scalable Technologies:** The choice of technologies like ASP.NET Core, SQL Server, and cloud infrastructure provides a clear path for vertical and horizontal scaling as user demand grows.

4.2.5 Data Integrity

The system must ensure that all data, especially related to ownership, is accurate and consistent.

- **Unique Identifiers:** The system enforces the use of unique identifiers (**Property Code, Chassis Number**) to prevent duplicate asset listings.
- **Transactional Operations:** The use of a relational database and atomic stored procedures ensures that multi-step operations (like deactivating old shares and creating new ones) are **transactional**. Either all steps succeed, or the entire operation is rolled back, preventing data corruption.

4.3 Use Case Diagrams

The following diagrams provide a high-level visual representation of the interactions between different user roles (actors) and the system's core functionalities. Each diagram is followed by a detailed specification table that describes the use case in a structured format.

4.3.1 Use Case: Asset Listing Process

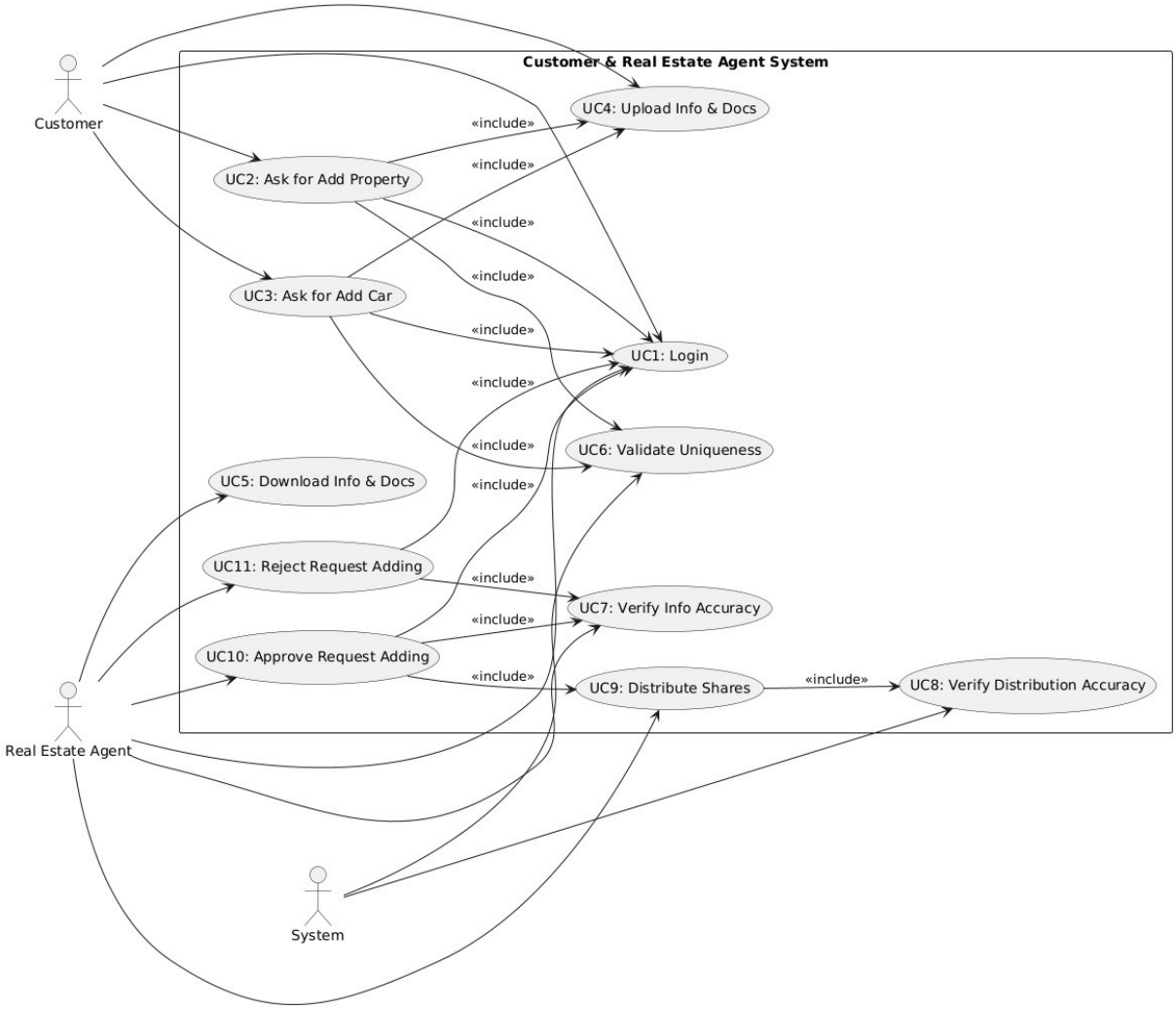


Figure 3 Asset Listing Process Use Case Diagram

Use Case 1: Ask for Add Asset (Property or Car)

Table 5 Asset Listing Process Use Case Specefication

Element	Details
Use Case ID	UC-02
Use Case Name	Ask for Add Asset (Property or Car)
Actors	**Primary:** Customer (Seller) **Supporting:** System
Description	This use case describes the process for a customer to submit a new asset for listing on the platform, including providing details and uploading documents.
Pre-conditions	1. The Customer is logged into their account. 2. The Customer has all necessary information and digital documents for the asset.
Post-conditions	**Success:** A new listing request is created with a "Pending" status, ready for agent review. **Failure:** No request is created, and the customer is notified of the error.
Basic Flow	1. Customer navigates to the "List an Asset" section of the application. 2. Customer fills out the form with the asset's details (e.g., type, description, price). 3. Customer uploads required digital documents (e.g., deed, registration) and photos. 4. Customer submits the request. 5. System validates the uniqueness of the asset via its Property Code or Chassis Number. 6. System successfully creates the new listing request and informs the customer.
Alternative Flows	N/A
Exceptions	**E1: Duplicate Asset Detected:** At step 5, the System finds that the unique identifier already exists in the database. The System blocks the submission and displays an error message to the Customer. The use case ends.

Use Case 2: Approve Asset Listing Request

Table 6 Approve Asset Listing Request Use Case Specification

Element	Details
Use Case ID	UC-10
Use Case Name	Approve Asset Listing Request
Actors	**Primary:** Real Estate Agent **Supporting:** System
Description	This use case covers the workflow for a Real Estate Agent to review, verify, and either approve or reject a new asset listing request submitted by a customer.
Pre-conditions	1. The Agent is logged into their dashboard. 2. There is at least one "Pending" listing request available for the Agent to process.
Post-conditions	**Success:** The asset listing is published on the platform, and its ownership shares are recorded. **Failure:** The request status is changed to "Rejected," and the Customer is notified.
Basic Flow	1. Agent logs into their dashboard and views the list of pending requests. 2. Agent selects a request to review. 3. Agent downloads and inspects the information and documents provided by the Customer. 4. Agent manually verifies the accuracy and consistency of the submitted materials. 5. Agent defines the ownership shares for the asset based on the legal documents. 6. Agent clicks the "Approve" button. 7. System changes the request status to "Approved," lists the asset on the platform, and notifies the Customer.
Alternative Flows	**A1: Agent Rejects the Request:** 1. At step 4, the Agent finds that the submitted information is inaccurate or incomplete. 2. The Agent clicks the "Reject" button, provides a reason for the rejection, and the use case ends.
Exceptions	N/A

Use Case 3: Distribute Shares

Table 7 Distribute Shares Use Case Specification

Element	Details
Use Case ID	UC-09
Use Case Name	Distribute Shares
Actors	**Primary:** Real Estate Agent **Supporting:** System
Description	Describes the process where an Agent defines the ownership structure (sole or shared) of an asset before final listing approval.
Pre-conditions	1. The Agent has verified the accuracy of a listing request's documents. 2. The legal documents specify the ownership structure (sole owner or multiple co-owners with percentages).
Post-conditions	**Success:** The ownership shares for the asset are correctly defined and saved in the system, linked to the respective owner(s). **Failure:** The approval process is halted until the shares are correctly distributed.
Basic Flow	1. After verifying the asset's documents (Use Case 2, step 4), the Agent proceeds to the "Distribute Shares" interface. 2. If the asset has a single owner, the Agent allocates a 100% share to that Customer. 3. If the asset has multiple owners, the Agent enters the Customer ID and percentage share for each co-owner as specified in the documents. 4. The System validates that the total percentage of all shares adds up to exactly 100%. 5. The Agent saves the share distribution. 6. The System records the ownership structure in the database. The Agent can now proceed with final approval.
Alternative Flows	N/A
Exceptions	**E1: Share Percentages Do Not Sum to 100%:** At step 4, the System detects that the entered percentages do not total 100%. The System displays an error message and prevents the Agent from saving the distribution.

4.3.2 Use Case: Shared Ownership and Sales

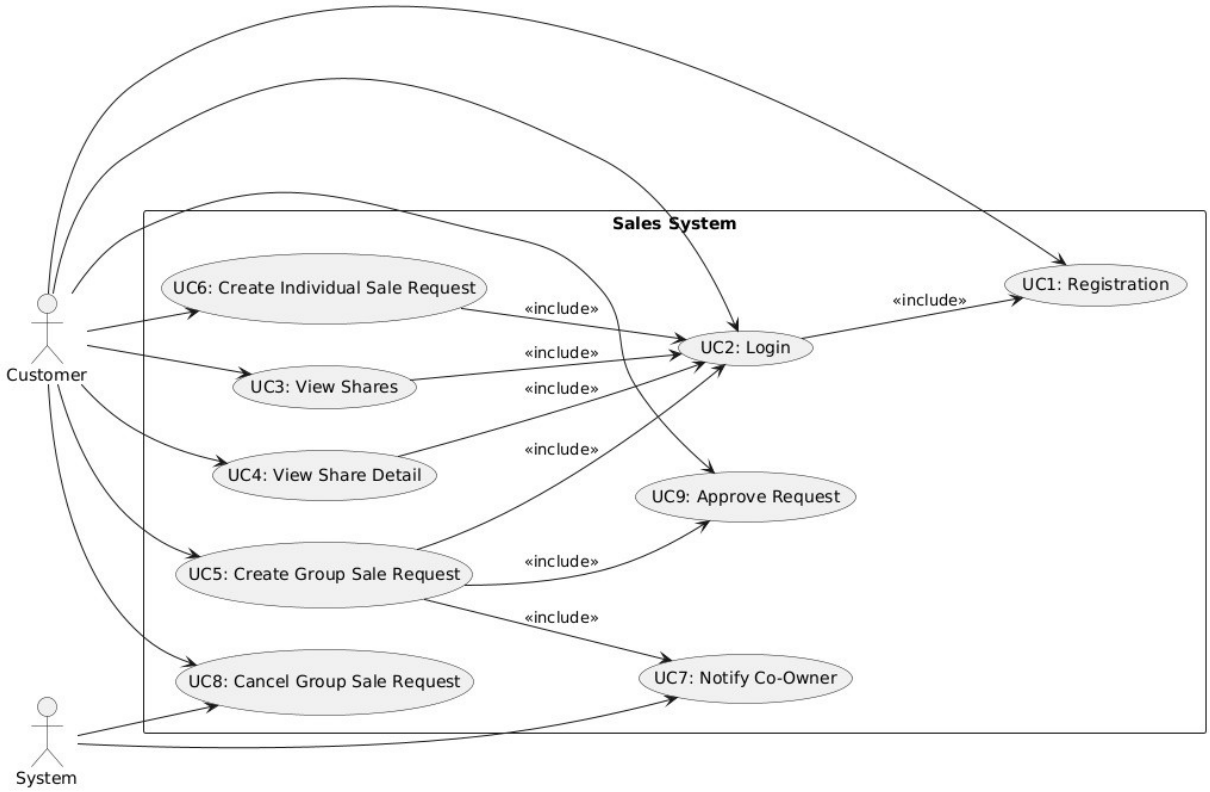


Figure 4 Shared Ownership and Sales Use Case Diagram

Use Case 4: View Share Detail

Table 8 View Share Detail Use Case Specification

Element	**Details**
Use Case ID	UC-03
Use Case Name	View Share Detail
Actors	**Primary:** Customer (Co-owner) **Supporting:** System
Description	This use case allows a customer who co-owns an asset to view the details of their specific share and see who the other co-owners are.
Pre-conditions	1. The Customer is logged into their account. 2. The Customer owns at least one share of an asset.
Post-conditions	**Success:** The Customer can view the detailed information of their ownership share. **Failure:** N/A
Basic Flow	1. Customer logs in and navigates to the "My Shares" section of their profile. 2. Customer sees a list of all assets they co-own. 3. Customer selects a

	specific asset. 4. The System displays the details of their share, including the property/car information, their ownership percentage, and a list of all other co-owners of that asset. 5. The Customer can now decide whether to initiate an individual or group sale.
Alternative Flows	N/A
Exceptions	N/A

Use Case 5: Create Individual Sale Request

Table 9 Create Individual Sale Request Use Case Specification

Element	**Details**
Use Case ID	UC-06
Use Case Name	Create Individual Sale Request
Actors	**Primary:** Customer (Co-owner) **Supporting:** System
Description	Enables a co-owner to list only their personal share of an asset for sale, providing flexibility for fractional ownership.
Pre-conditions	1. The Customer is logged into their account. 2. The Customer has viewed the details of a share they own.
Post-conditions	**Success:** A new sales listing for the Customer's individual share is created and made public. **Failure:** No sale request is created.
Basic Flow	1. From the "Share Detail" page, the Customer clicks the "Create Individual Sale Request" button. 2. The System creates a new sales listing for that specific share, including its percentage. 3. The System displays the new listing on the platform's Home Page.
Alternative Flows	N/A
Exceptions	N/A

Use Case 6: Create Group Sale Request

Table 10 Create Group Sale Request Use Case Specification

Element	Details
Use Case ID	UC-05
Use Case Name	Create Group Sale Request
Actors	**Primary:** Customer (Co-owner) **Supporting:** System, Other Co-owners
Description	Details the workflow for a co-owner to initiate a sale of an entire asset, requiring approval from all other involved co-owners.
Pre-conditions	1. The Customer is logged in and is viewing a shared asset they co-own. 2. The asset has at least one other co-owner.
Post-conditions	**Success:** A new sale request for the entire asset is created. **Failure:** The group sale request is canceled, and no listing is created.
Basic Flow	1. From the "Share Detail" page, the Customer selects the "Create Group Sale Request" option. 2. Customer is presented with a list of the other co-owners. 3. Customer selects the co-owners to include in the sale. 4. Customer submits the request. 5. System sends notifications to the selected co-owners, requesting their approval. 6. All invited co-owners approve the request within the 5-minute time limit. 7. System creates a single sales listing for 100% of the asset.
Alternative Flows	N/A
Exceptions	**E1: Request Rejected by a Co-owner:** At step 6, one of the invited co-owners rejects the group sale request. The System immediately cancels the process and notifies all involved parties. The use case ends. **E2: Request Times Out:** At step 6, the 5-minute time limit expires before all co-owners have responded. The System automatically cancels the process and notifies all involved parties. The use case ends.

4.3.3 Use Case: Asset Purchase Process

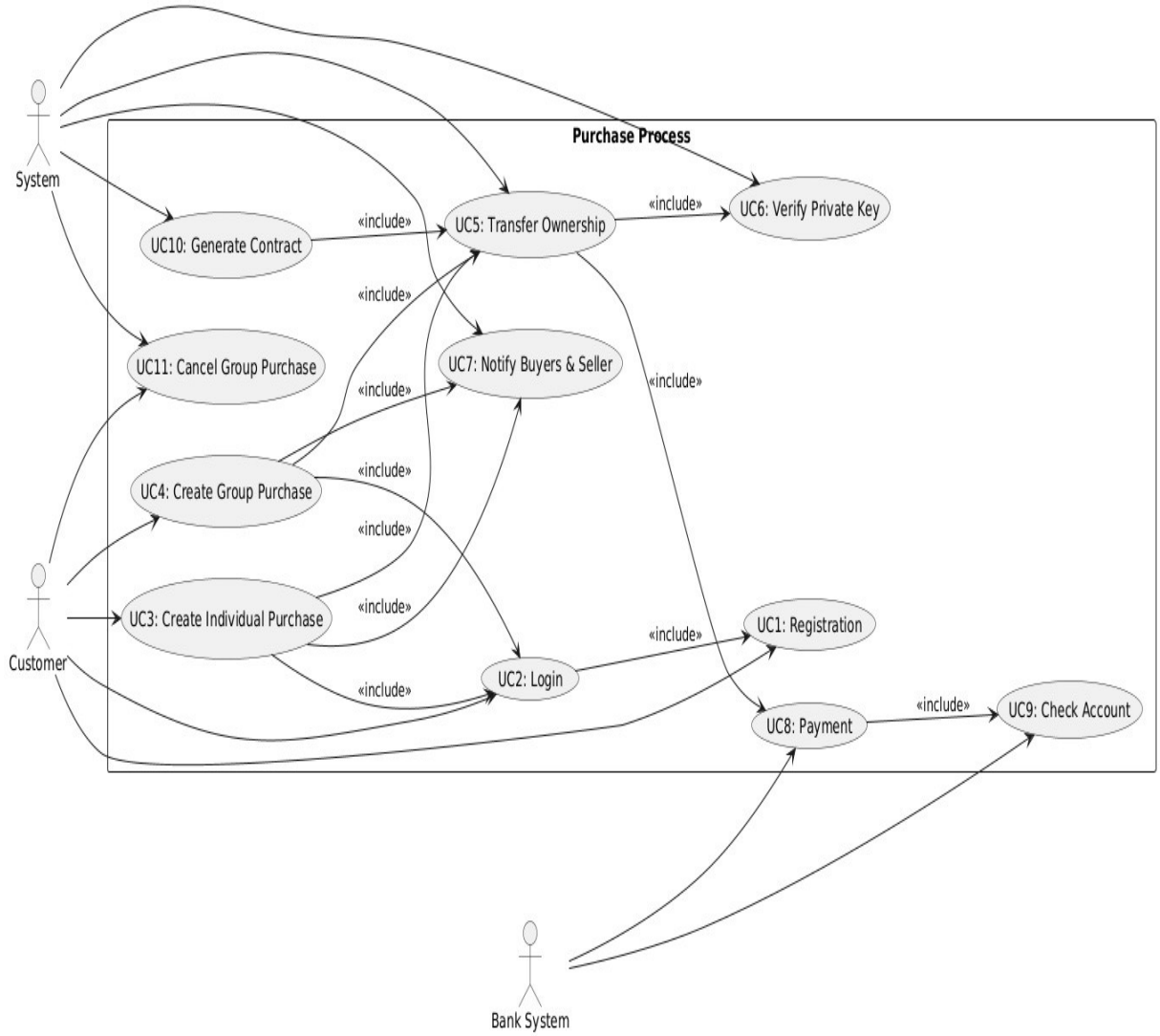


Figure 5 Asset Purchase Process Use Case Diagram

Use Case 7: Create Purchase

Table 11 Create Purchase Use Case Specification

Element	**Details**
Use Case ID	UC-03
Use Case Name	Create Purchase
Actors	**Primary:** Customer (Buyer) **Supporting:** Customer (Seller), System
Description	Describes the end-to-end process for a buyer to purchase an asset, including authorization and fund verification.

Pre-conditions	1. The Buyer is logged in and has sufficient funds. 2. An asset is actively listed for sale on the platform.
Post-conditions	**Success:** Ownership of the asset is transferred to the Buyer, and a contract is generated. **Failure:** The purchase is canceled, and no changes are made to ownership.
Basic Flow	1. Buyer finds a listing and clicks the "Buy" button. 2. System notifies the Buyer and all Seller(s), prompting them to enter their private keys. 3. All parties enter their correct private keys for authorization. 4. System confirms that the Buyer has sufficient funds for the purchase. 5. System initiates the ownership transfer. 6. System generates an electronic contract and notifies all parties of the successful transaction.
Alternative Flows	N/A
Exceptions	**E1: Private Key Verification Fails:** At step 3, at least one party enters an incorrect key or fails to respond. The purchase process is canceled. The use case ends. **E2: Insufficient Funds:** At step 4, the System determines the Buyer's account lacks sufficient funds. The purchase process is canceled, and the Buyer is notified. The use case ends.

Use Case 8: Verify Private Key

Table 12 Verify Private Key Use Case Specification

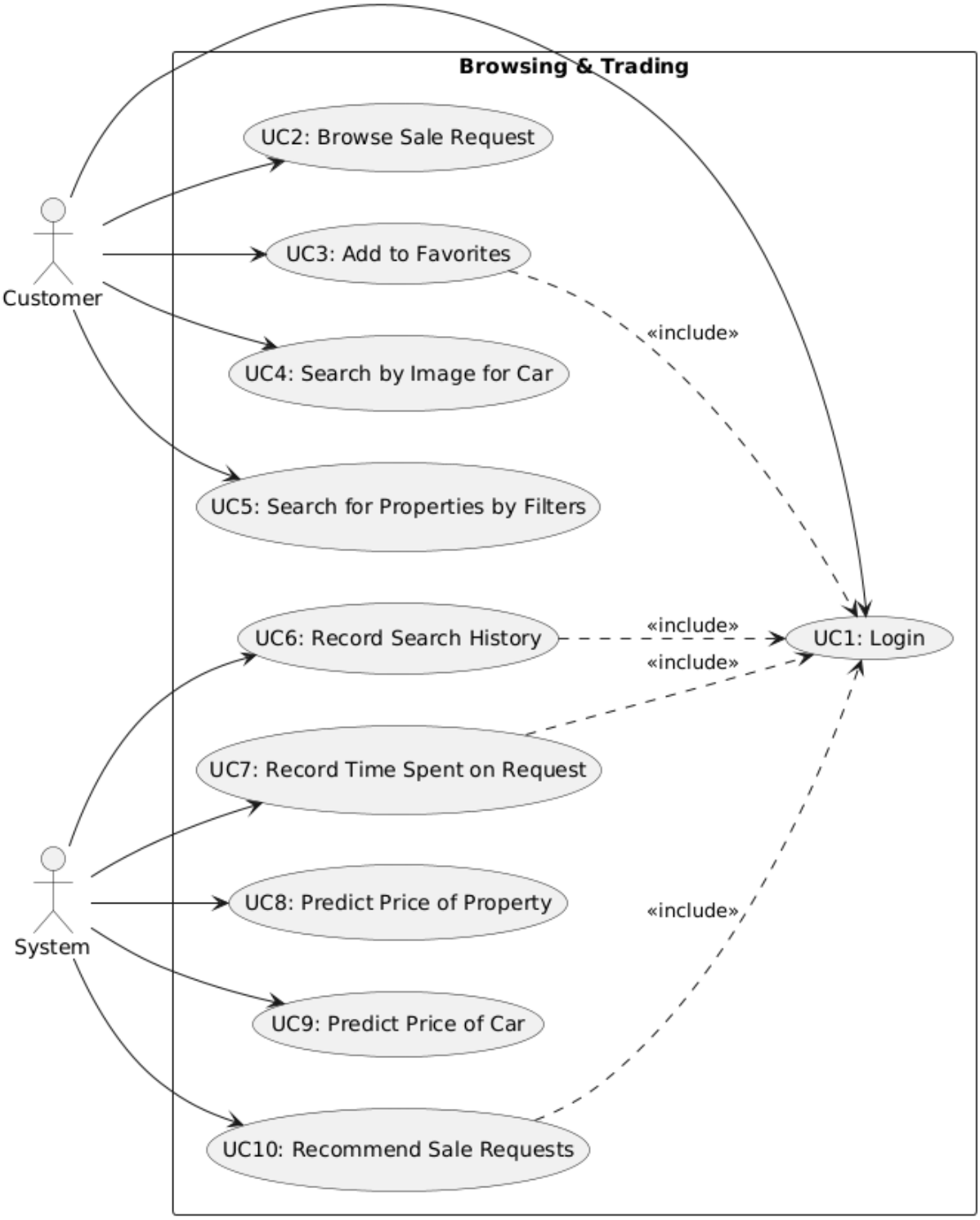
Element	Details
Use Case ID	UC-06
Use Case Name	Verify Private Key
Actors	**Primary:** System **Supporting:** Customer (Buyer), Customer (Seller)
Description	Outlines the security sub-process where the system authenticates all parties in a transaction using their unique private keys.
Pre-conditions	1. A purchase has been initiated. 2. The System has sent a notification to all relevant parties to provide their private key.
Post-conditions	**Success:** All parties are successfully authenticated, and the purchase process proceeds. **Failure:** The purchase process is canceled due to failed authentication.
Basic Flow	1. System prompts the Buyer and Seller(s) to enter their private key for transaction authorization. 2. Each user enters their key. 3. The System receives the submitted keys. 4. For each key, the System validates it against the securely stored hash for that user. 5. All keys are determined to be correct. 6. The System flags the transaction as authenticated.
Alternative Flows	N/A
Exceptions	**E1: Incorrect Key Submitted:** At step 4, the System determines that at least one submitted key is incorrect. The System immediately cancels the entire purchase transaction and notifies all parties. The use case ends.

Use Case 9: Transfer Ownership

Table 13 Transfer Ownership Use Case Specification

Element	**Details**
Use Case ID	UC-05
Use Case Name	Transfer Ownership
Actors	**Primary:** System **Supporting:** N/A
Description	Details the automated, backend process of updating the ownership records in the database after a successful sale.
Pre-conditions	1. A purchase has been successfully authorized (private keys and funds are verified).
Post-conditions	**Success:** Ownership shares are correctly updated in the database. **Failure:** The database transaction is rolled back, leaving original ownership intact.
Basic Flow	1. System receives the RequestForSalesID and the list of new buyers and their contributions. 2. System calls the ReplaceRequestShares database stored procedure. 3. The stored procedure deactivates the sellers' old Shares and Ownerships records. 4. The stored procedure calculates the new percentage share for each buyer. 5. The stored procedure inserts new, active Shares and Ownerships records for each buyer. 6. The database transaction completes successfully.
Alternative Flows	N/A
Exceptions	**E1: Database Transaction Error:** At any point during the stored procedure execution, a database error occurs. The database automatically rolls back all changes made during the transaction. The use case fails.

4.3.4 Use Case: Browsing and AI Features



6 Browsing and AI Features Use Case Diagram Figure

Use Case 10: Search by Image for Car

Table 14 Search by Image for Car Use Case Specification

Element	Details
Use Case ID	UC-4
Use Case Name	Search by Image for Car
Actors	**Primary:** User **Supporting:** System, AI Server
Description	Allows a user to find vehicles by uploading an image, leveraging an AI model for visual recognition.
Pre-conditions	1. The User is on the platform's search page.
Post-conditions	**Success:** A list of visually similar cars from the platform's inventory is displayed. **Failure:** An appropriate error or "no results" message is displayed.
Basic Flow	1. User selects the "Search by Image" option. 2. User uploads a car image from their device. 3. System sends the image to the AI Server. 4. The AI Server's image classification model processes the image to identify its features. 5. The AI Server finds and returns a list of similar vehicles from the platform's inventory. 6. System displays the results to the user.
Alternative Flows	N/A
Exceptions	**E1: No Similar Cars Found:** At step 5, the AI Server cannot find any visually similar vehicles in the inventory. The System displays a "No results found" message to the user.

Use Case 11: Predict Price of Property

Table 15 Predict Price of Property Use Case Specification

Element	**Details**
Use Case ID	UC-8
Use Case Name	Predict Price of Property
Actors	**Primary:** User **Supporting:** System, AI Server
Description	Enables a user to get a data-driven market price estimation for a property by providing its key characteristics.
Pre-conditions	1. The User is viewing a property listing or has access to the price prediction tool.
Post-conditions	**Success:** The User is presented with an estimated market price for the property. **Failure:** The System displays an error message indicating that a prediction could not be made.
Basic Flow	1. The User navigates to the price prediction feature. 2. The User enters the key details of a property (e.g., number of bedrooms, bathrooms, size, location). 3. The User submits the details. 4. The System sends the structured data to the AI Server. 5. The AI Server's price prediction model processes the features and calculates an estimated price. 6. The AI Server returns the predicted price to the System. 7. The System displays the estimated price to the User.
Alternative Flows	N/A
Exceptions	N/A

Use Case 12: Recommend Sale Requests

Table 16 Recommend Sale Requests Use Case Specification

Element	Details
Use Case ID	UC-10
Use Case Name	Recommend Sale Requests
Actors	**Primary:** System **Supporting:** User, AI Server
Description	Describes the system's proactive process of learning user preferences and displaying personalized asset recommendations.
Pre-conditions	1. The User has been browsing the platform, generating interaction data (views, searches, favorites).
Post-conditions	**Success:** A list of personalized recommendations is displayed to the User. **Failure:** N/A
Basic Flow	1. System continuously and passively tracks user behavior (search history, favorited items, etc.). 2. System aggregates this data to create a user preference profile. 3. System sends this profile to the AI Server's recommendation engine. 4. The recommendation engine processes the profile and identifies relevant properties and cars. 5. The AI Server returns a list of recommended sale requests. 6. System displays these recommendations to the user in a dedicated UI section (e.g., "Recommended for You").
Alternative Flows	**A1: New User with No History:** 1. At step 1, the System detects that the user has no interaction history. 2. The System requests a list of globally popular or trending items from the AI Server instead of personalized ones. 3. The System displays these general recommendations to the user.
Exceptions	N/A

4.4 Context Diagram (Level 0 DFD)

The Data Flow Diagram illustrates how information moves through the "TrustAsset" system. The highest-level DFD, the Context Diagram (Level 0), defines the boundary of the system and its interactions with external entities.

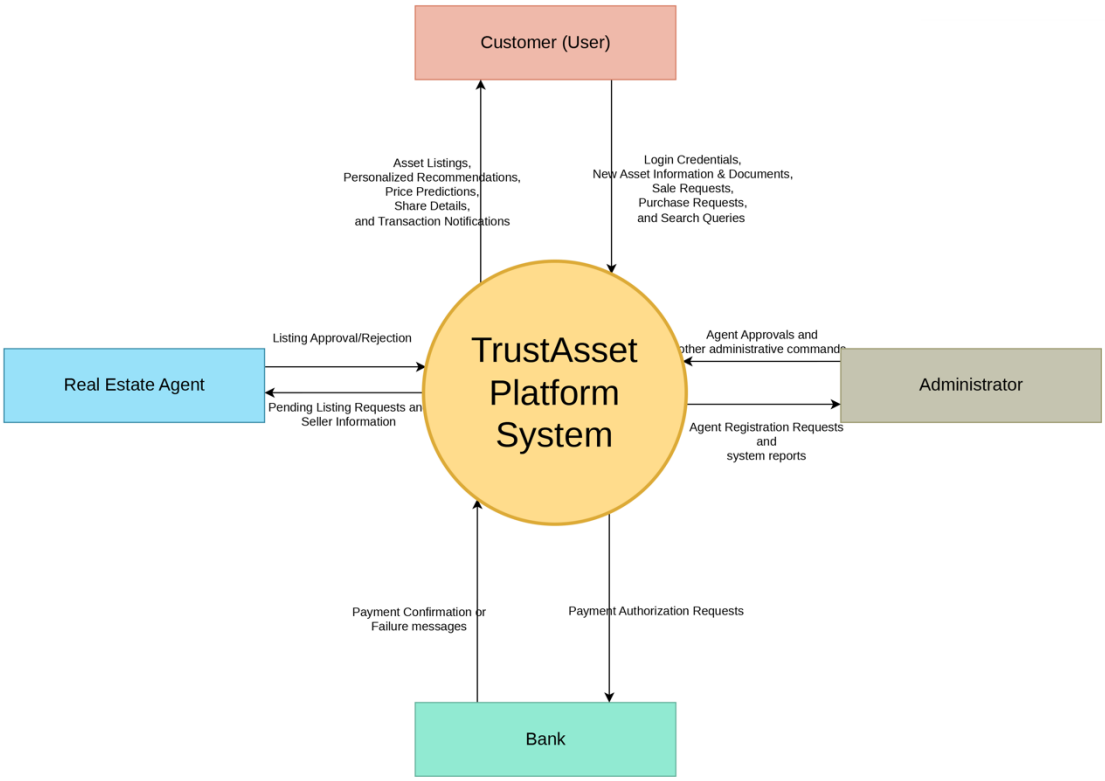


Figure 7 TrustAsset Context Diagram (Level 0 DFD)

Chapter 5: System Design

This chapter outlines the technical design of the "TrustAsset" platform, detailing the high-level architecture, database structure, core algorithms, and specific solutions used to build a robust and scalable system.

5.1 System Component Diagram

The following diagram provides a high-level visualization of the system's main components and their interactions, illustrating the overall system architecture.

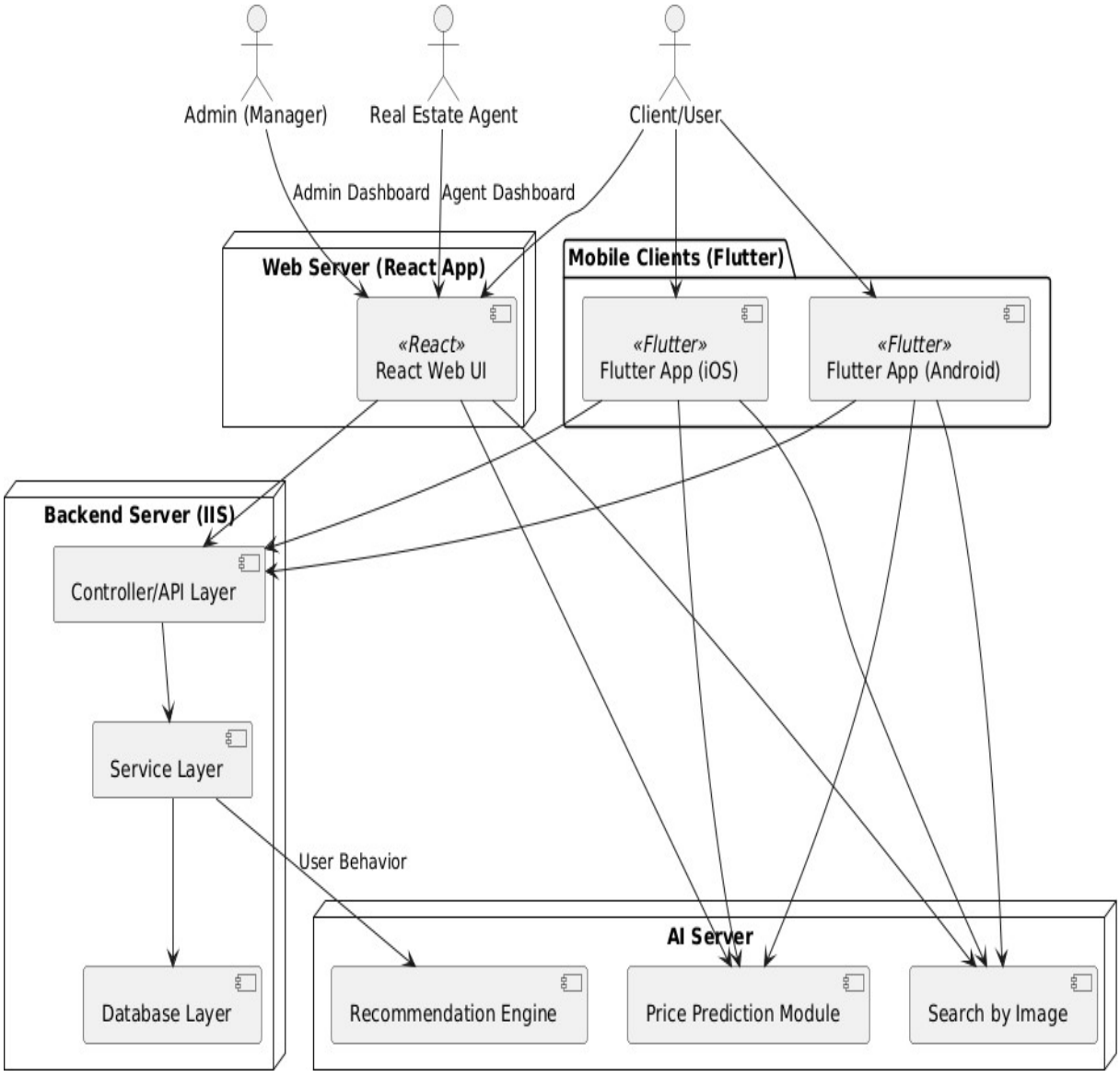


Figure 8 TrustAsset System Component Diagram

5.1.1 Architectural Layers

The system is built using a **Layered Architecture** to ensure modularity, scalability, and maintainability. This architectural pattern divides the application into distinct layers, each with a specific responsibility. This separation of concerns ensures that the different aspects of the application—user interface, business logic, and data storage—are decoupled, facilitating ease of modification and future growth.

- **Presentation Layer:** This layer is responsible for handling all user interactions and displaying information.
    - **Mobile App:** Developed using Flutter (Dart) to create a cross-platform experience for iOS and Android from a single codebase. State management is handled by flutter_bloc for predictable state changes, and network operations are managed using the Dio package.
    - **Web App (Admin & Agent Portal):** Built with React.js, this provides a powerful web-based interface for administrative and agent tasks. It utilizes Redux for state management and Axios for API requests.
- **Business Logic Layer:** This is the core of the backend, responsible for business rules, data validation, and system orchestration.
    - It is implemented using C# ASP.NET Core 8, providing a robust and efficient framework for building RESTful APIs.
    - JWT (JSON Web Token) Authentication is used for securing user sessions and authorizing access to resources.
    - SignalR is integrated to facilitate real-time communication features, such as notifications and messaging.
- **Data Access Layer:** This layer manages all data storage and retrieval operations.
    - Microsoft SQL Server is used as the relational database management system (RDBMS) to store user profiles, property and vehicle listings, and other transactional data, ensuring data integrity and security.

5.2 Entity Relationship Diagram (ERD)

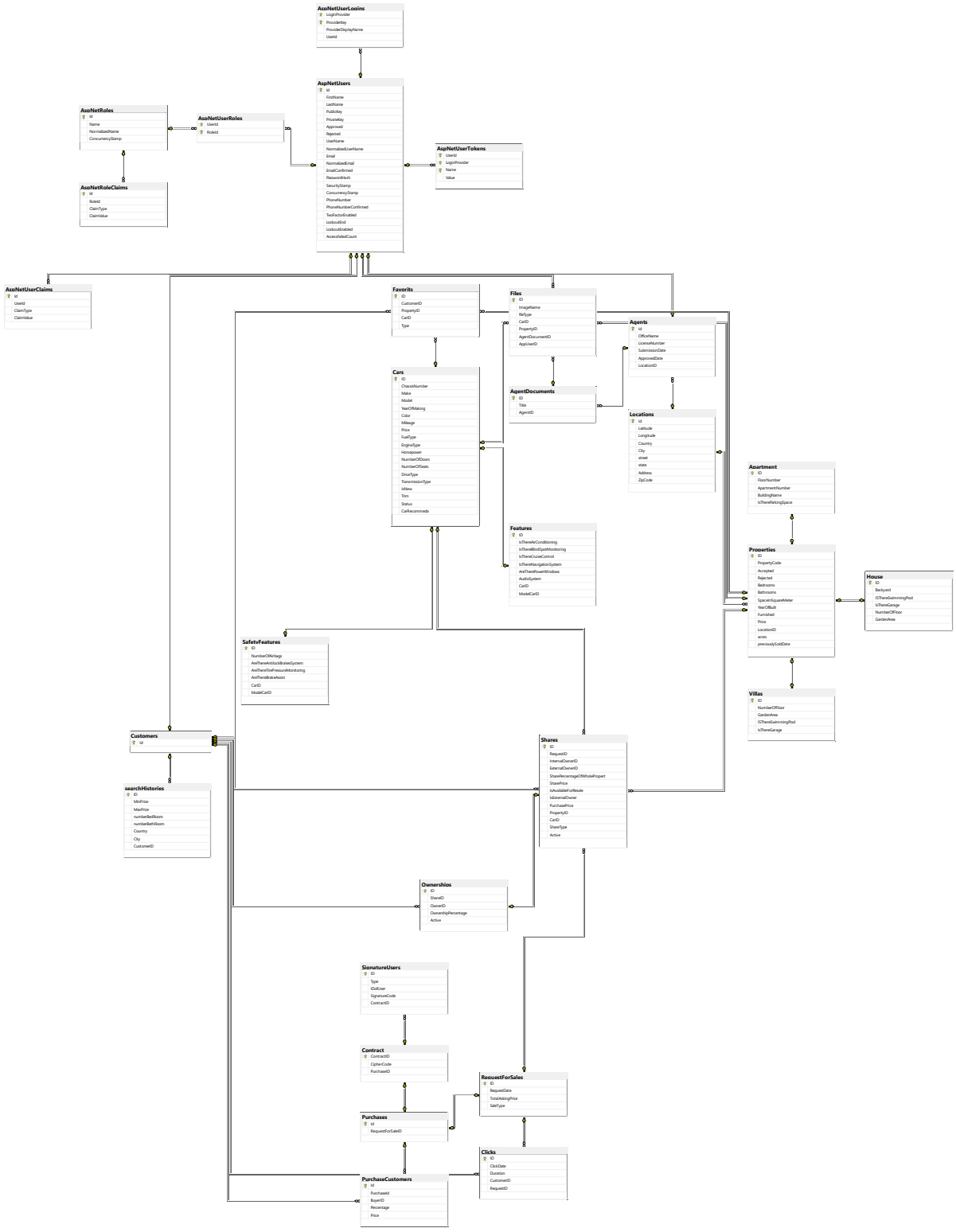


Figure 9 TrustAsset Entity Relationship Diagram

5.3 Process Flowchart

The diagram below is a comprehensive process flowchart that visualizes the end-to-end user journeys and system logic within the "TrustAsset" platform.

The flowchart maps out all the major scenarios, starting from user registration and login. It details the steps for a seller to upload an asset, the agent's verification and approval process, and the system's logic for handling duplicate entries. It also visualizes the unique workflows for managing shared ownership, including how individual and group sale requests are created, notified, and processed. Finally, it traces the secure purchase process, from the buyer's initial request to the final contract generation, and shows how the AI-powered features like search and recommendations are integrated into the user's browsing experience.

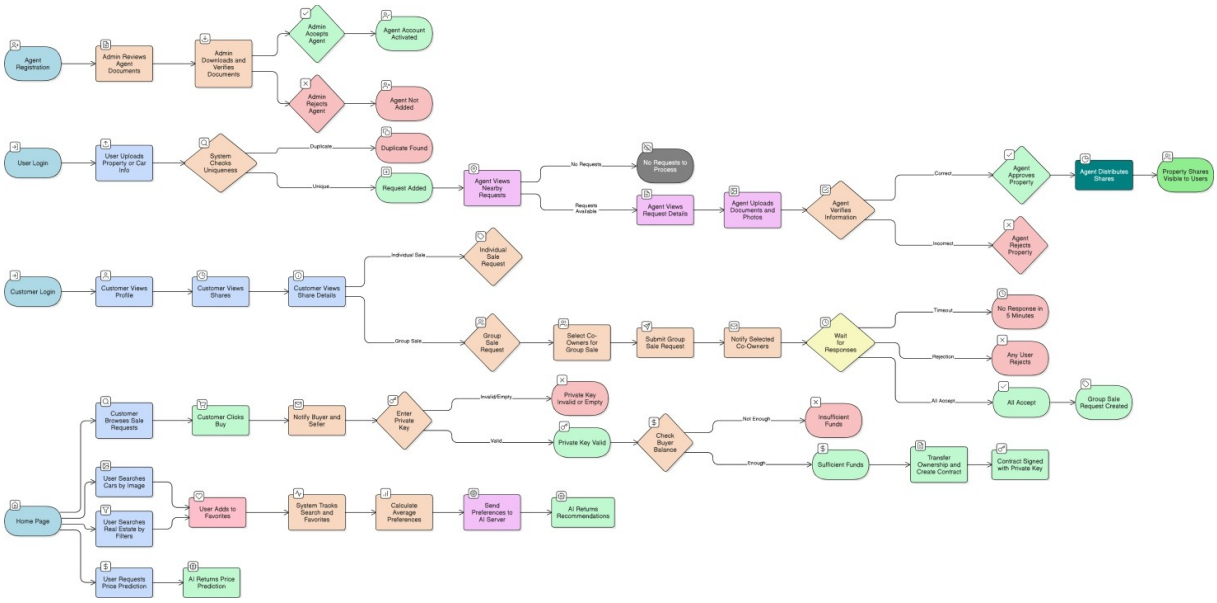


Figure 10 TrustAsset Process Flowchart Diagram

5.4 Web Pages Design

The user interface (UI) and user experience (UX) for both the mobile and web applications are designed to be intuitive, clean, and responsive. The design prioritizes ease of use, ensuring that complex processes are presented to the user in a clear, step-by-step manner.

5.4.1 Web Application Design

The web application, built with React.js, serves as the comprehensive portal for all user roles, including customers, agents, and administrators. The design provides tailored interfaces and functionalities based on the logged-in user's role

- **For Customers:** The web app provides a full-featured experience, including browsing listings, searching, managing their profile and owned shares, and initiating sale or purchase requests.
- **For Agents:** The design focuses on efficient workflow processing. Key pages include a dashboard for an at-a-glance overview of requests and sales, and a detailed request management view for approving or rejecting new listings.

The secure login portal for the TrustAsset platform allows users to access their accounts by entering their registered email and password. This ensures a secure entry point for managing high-value assets like properties and vehicles.

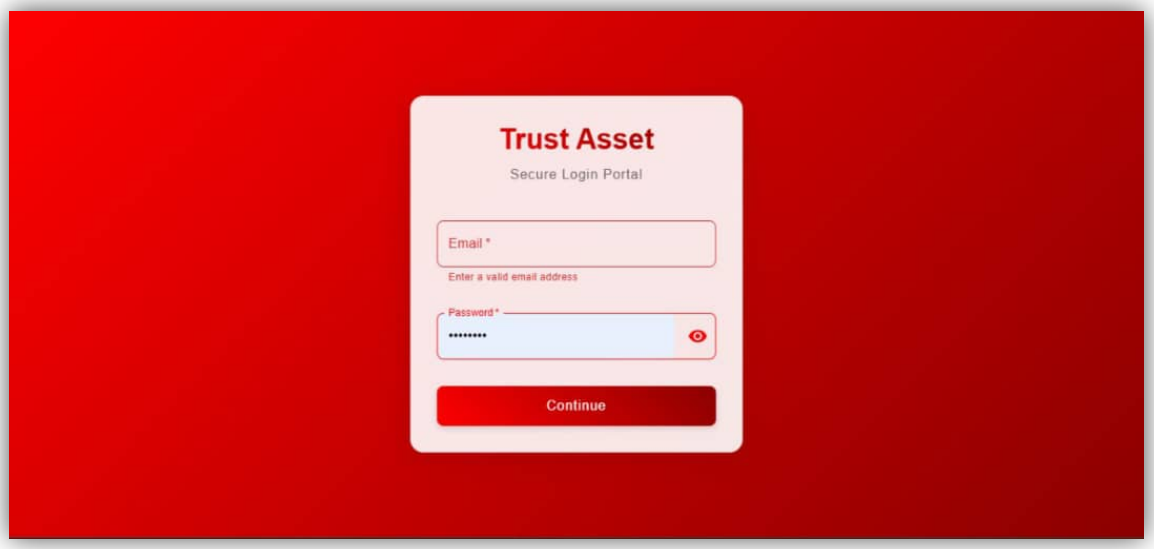


Figure 11 The TrustAsset Secure Login Portal

The user registration page for TrustAsset is where new users create an account. They provide essential information such as their name, email, and password to join the platform and access its features for buying and selling assets.

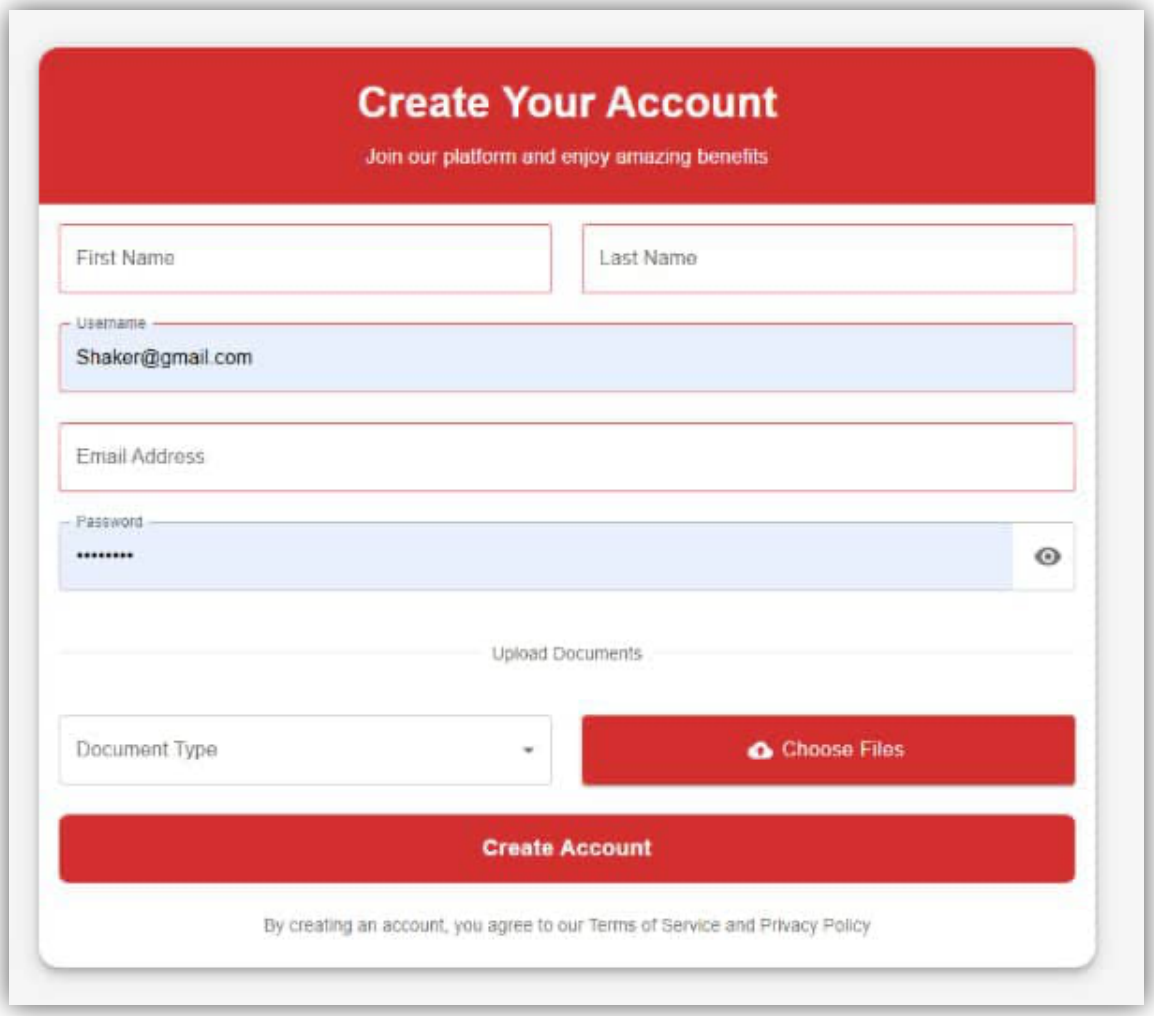


Figure 12 The "Create Your Account" page on TrustAsset

The main homepage of the TrustAsset marketplace displays featured properties and cars.

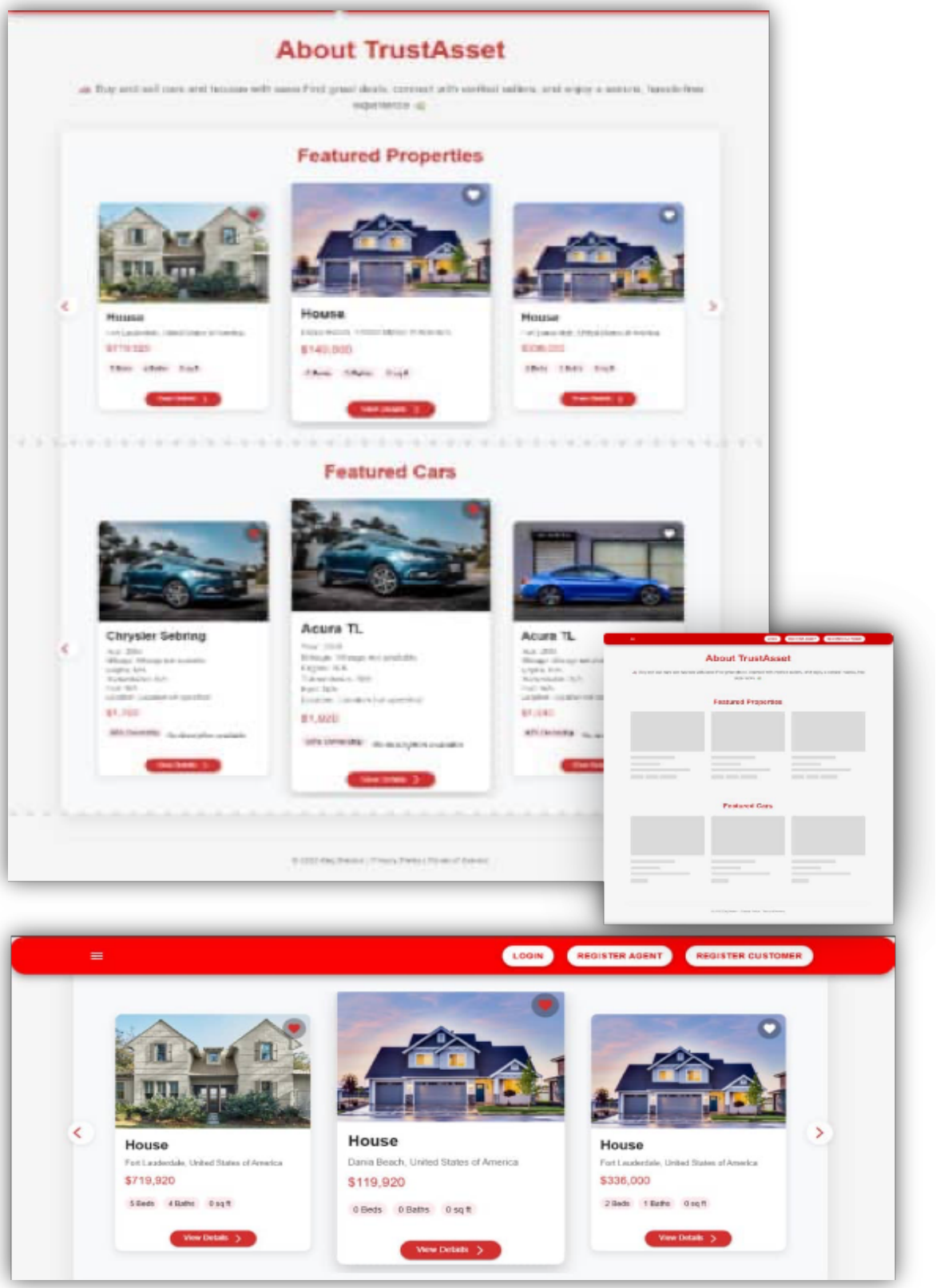


Figure 13 Homepage featuring property and car listings

The "Property Dashboard" serves as a central hub for users to browse, discover, and manage a curated selection of exclusive property listings. Each listing provides a snapshot with a "View Details" option for further exploration.

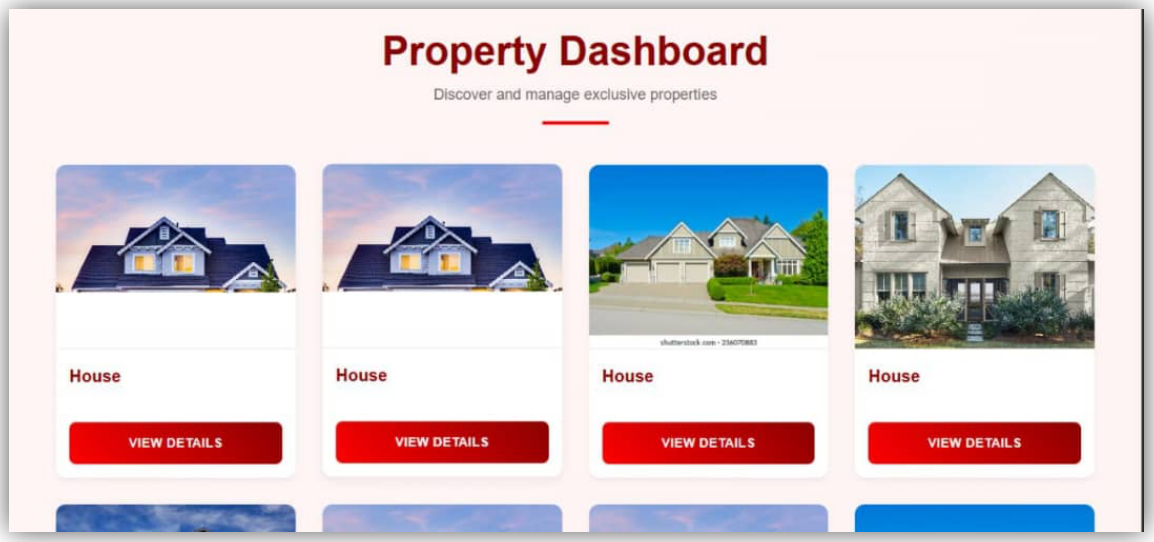


Figure 14 The Property Dashboard showing various house listing

A detailed backend view for agents includes the "Property Details" page, where an agent verifies documents and manages a listing. Critically, it also contains the "Assign Ownership Shares" section—a core feature for establishing and managing fractional ownership on the platform.

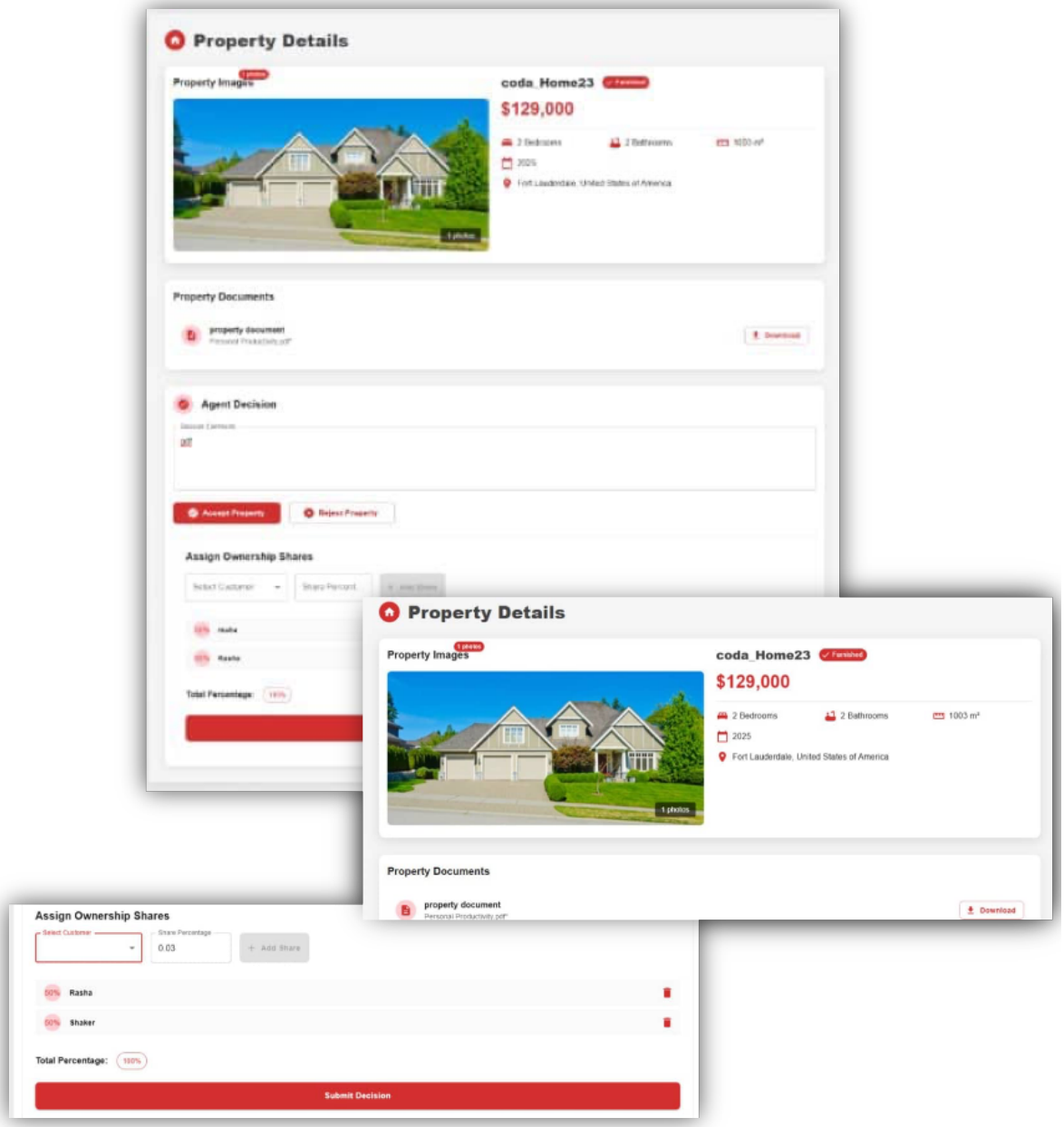


Figure 15 The Property Details page for assigning ownership shares

The public-facing details page for a property on TrustAsset provides potential buyers with comprehensive information. It includes price, images, and key attributes like the number of bedrooms, bathrooms, and amenities, along with a clear "Request to Buy" button to initiate the purchase process.

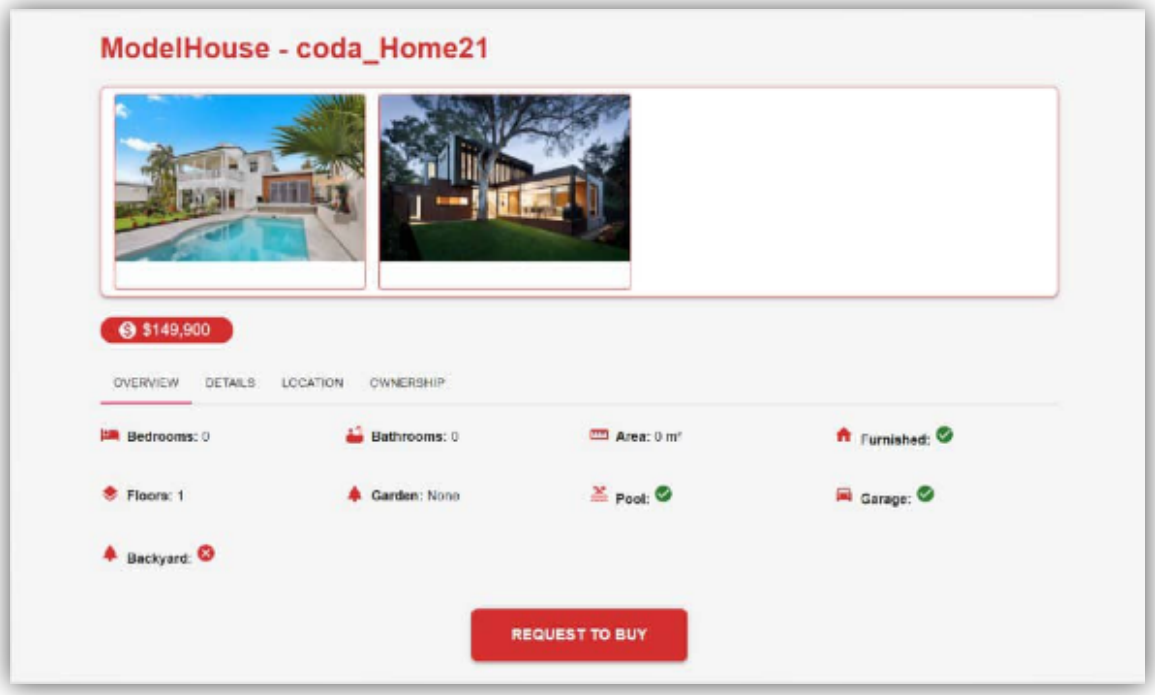


Figure 16 A property's detail page with price and specifications

The "Your Investment Portfolio" page is a key feature for users engaged in fractional ownership. It allows a co-owner to manage and track all their property shares in one place by displaying the ownership percentage and corresponding value for each share.

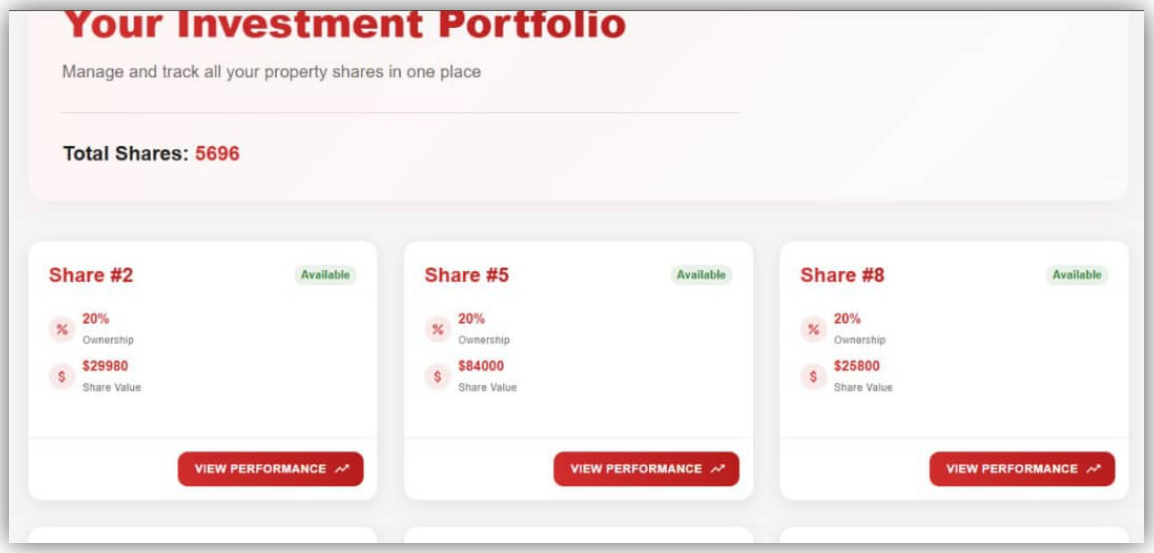


Figure 17 An Investment Portfolio page showing total and individual shares

TrustAsset's unique "Group Sale Request" feature is managed from the share details page. From here, a co-owner can view a list of other co-owners and initiate a sale of the entire asset, which then requires approval from all involved parties to proceed.

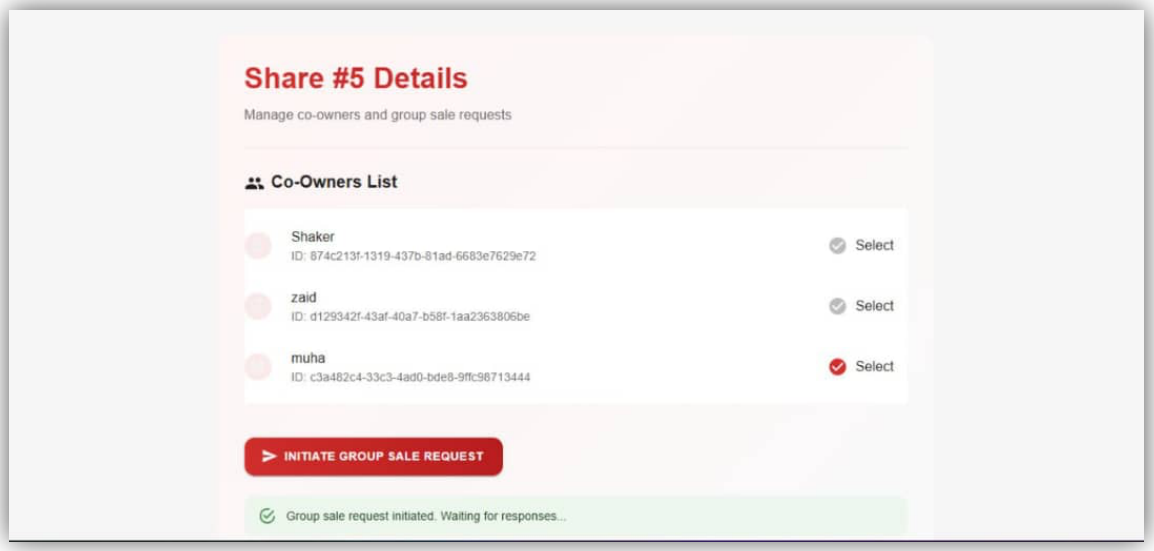


Figure 18 A "Co-Owners List" to initiate a group sale request

The "House Details Form" is used by sellers or agents to submit comprehensive information about a property. This structured form captures everything from basic amenities to structural details and quality ratings, ensuring data integrity for the agent-mediated verification process.

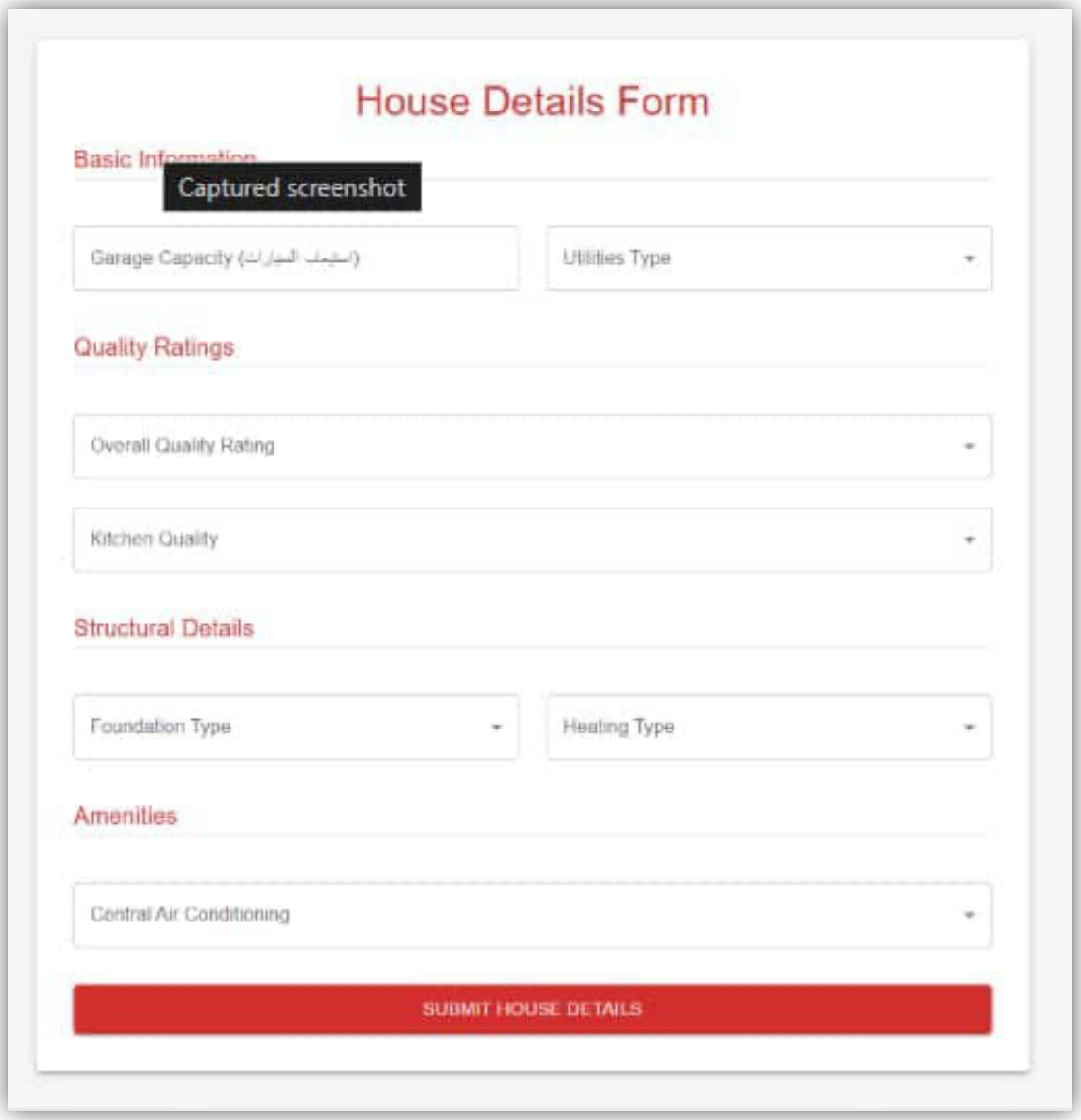


Figure 19 The "House Details Form" for submitting asset information

5.4.2 Mobile Application Design (Flutter)

The mobile application, built with Flutter, is the primary interface for customers (buyers and sellers). The design is focused on a seamless and engaging user journey, from browsing to secure transactions. Key screens include:

- **Home Page:** Displays featured listings and personalized recommendations.
- **Search and Filter:** Powerful search functionality, including the AI-powered image search for cars and detailed filters for properties.
- **Asset Detail Page:** A clean and comprehensive view of a single property or vehicle, including photos, details, price, and an option to initiate a purchase.
- **Profile and Shares Management:** A dedicated section for users to manage their personal information, track their owned shares, and initiate individual or group sales.

Figure 20 Application Loading Screen

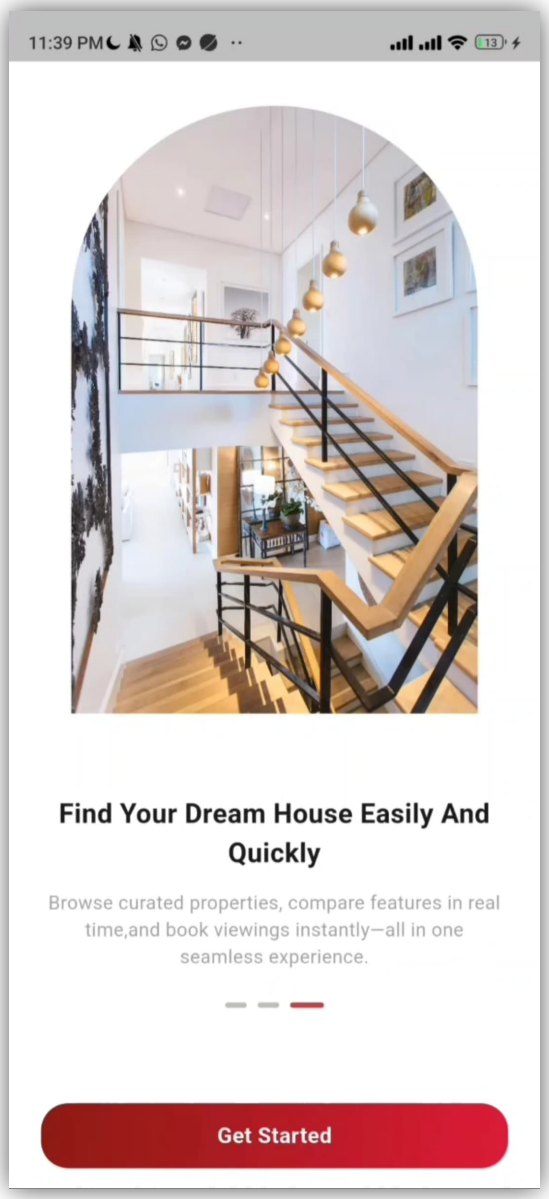


Figure 21 Application Welcome Screen

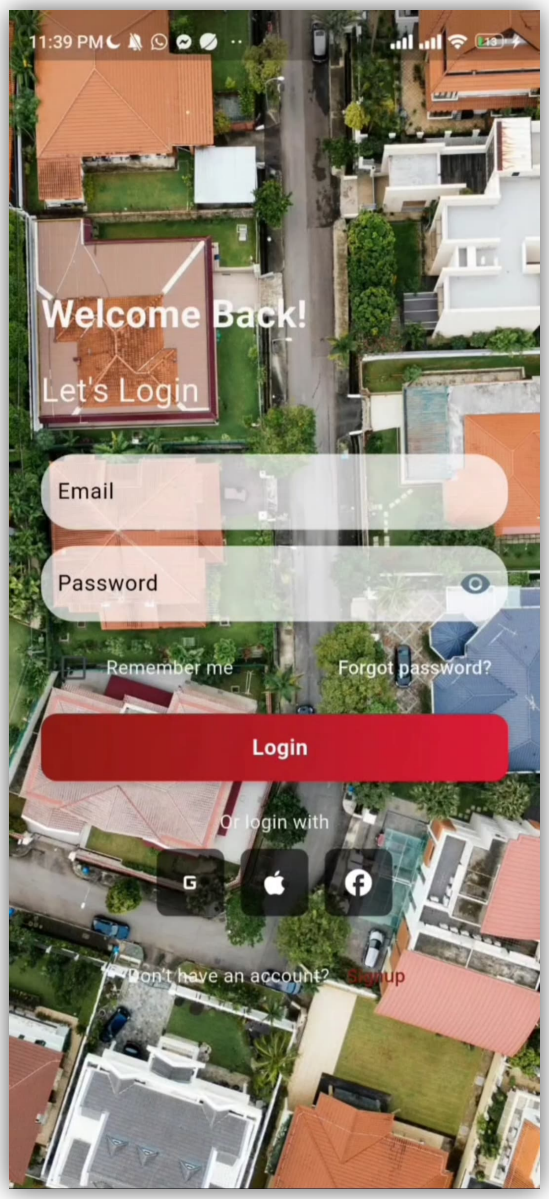


Figure 22 User Login Screen

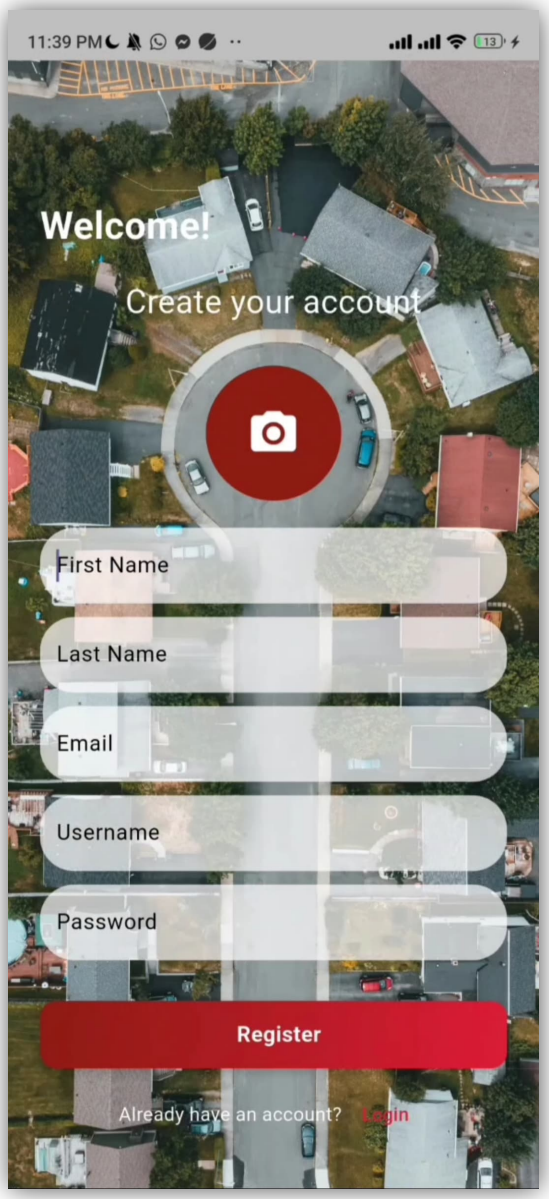


Figure 23 New Account Registration Screen

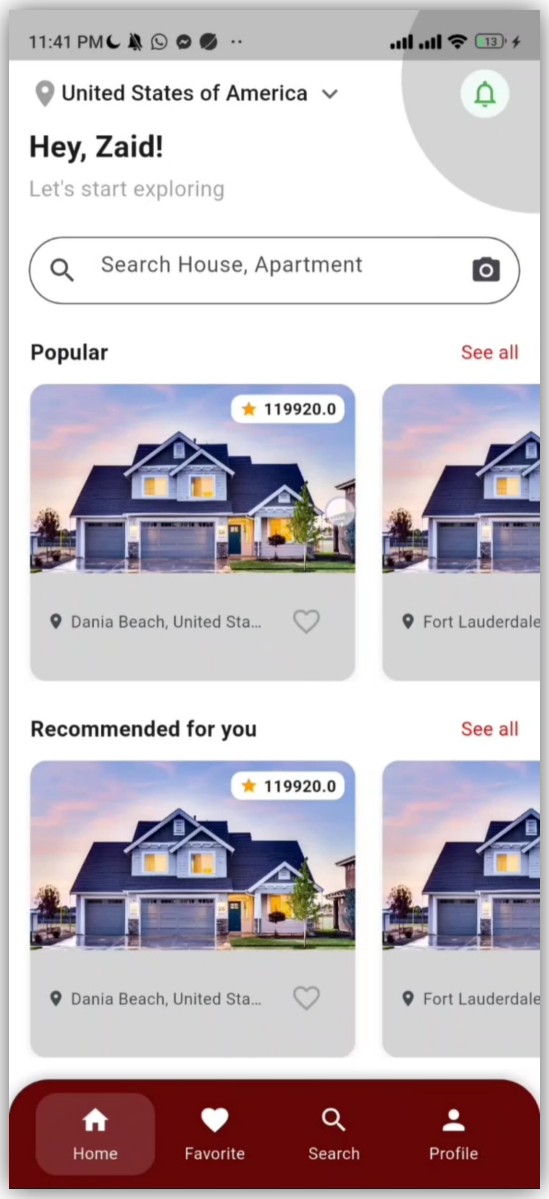


Figure 24 Home Page with Property Listings

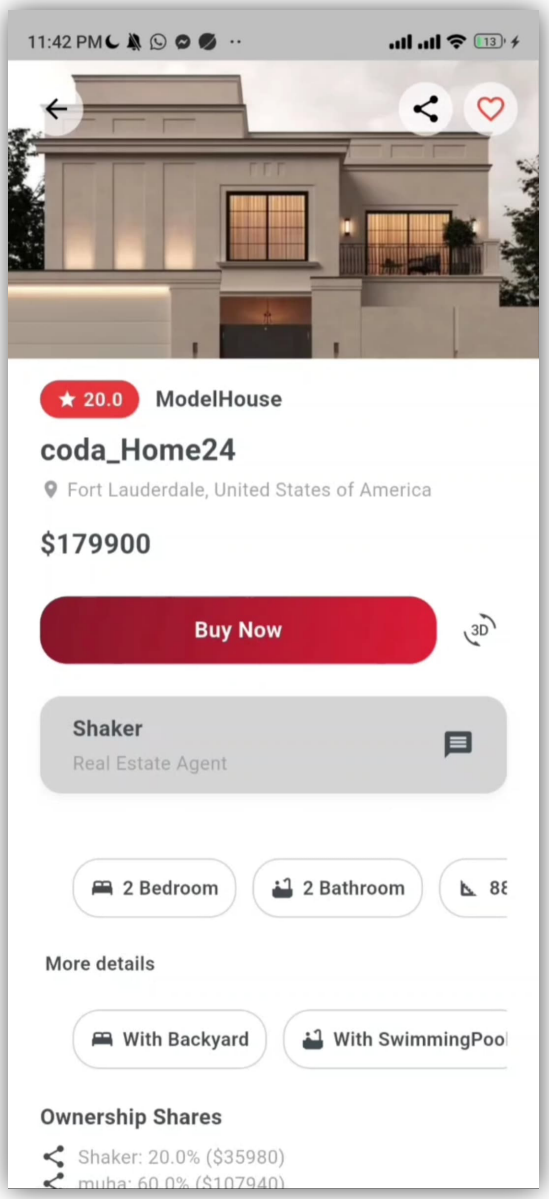


Figure 25 Property Details View with Price and Features

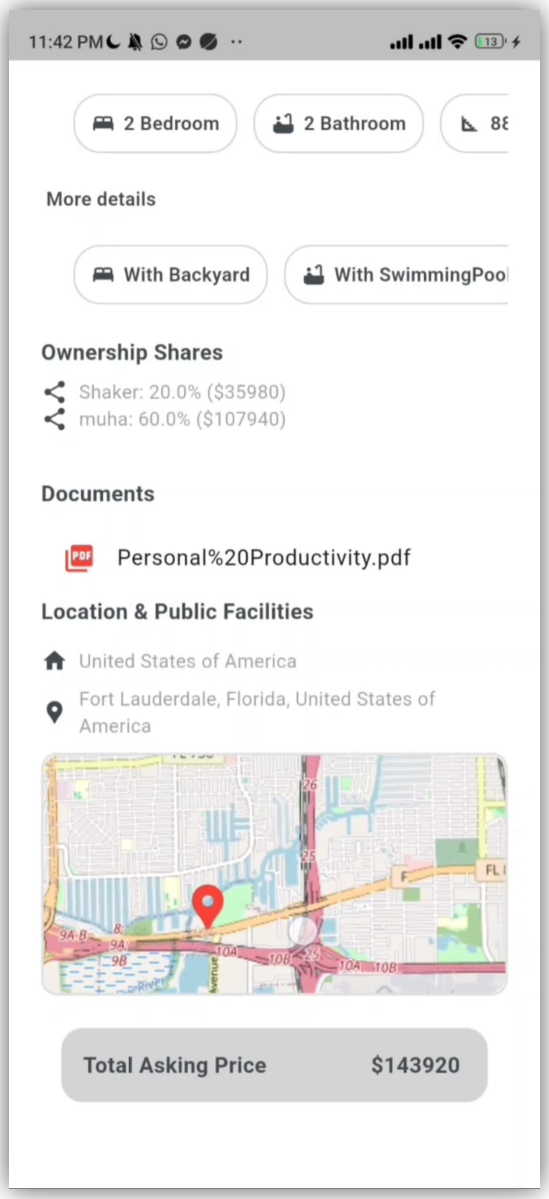


Figure 26 Property Details View with Ownership Shares and Location Map

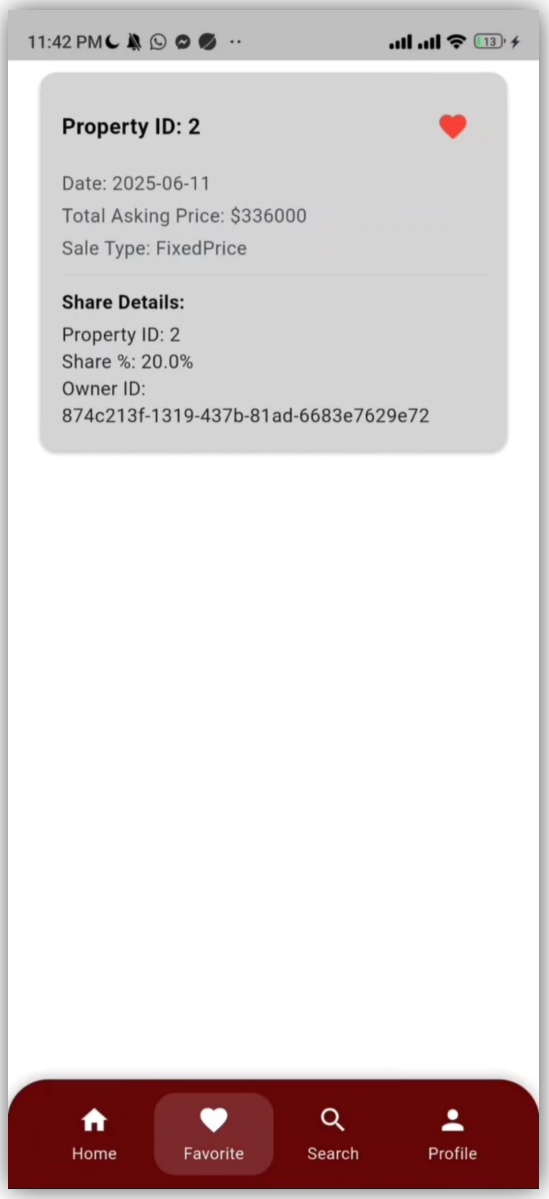


Figure 27 Favorite Assets Page

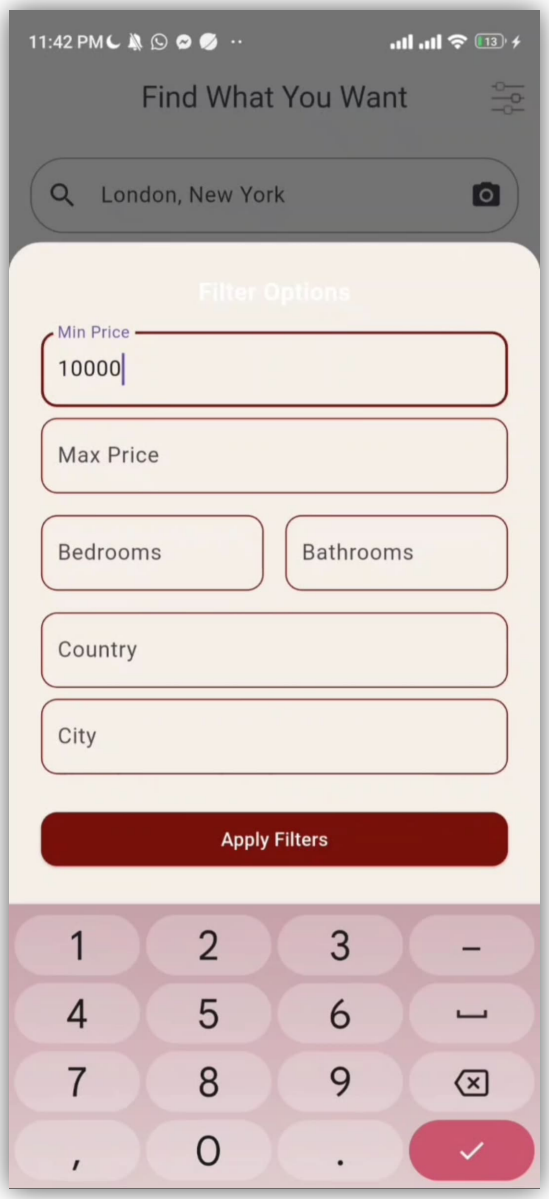


Figure 28 Search Filter Options View

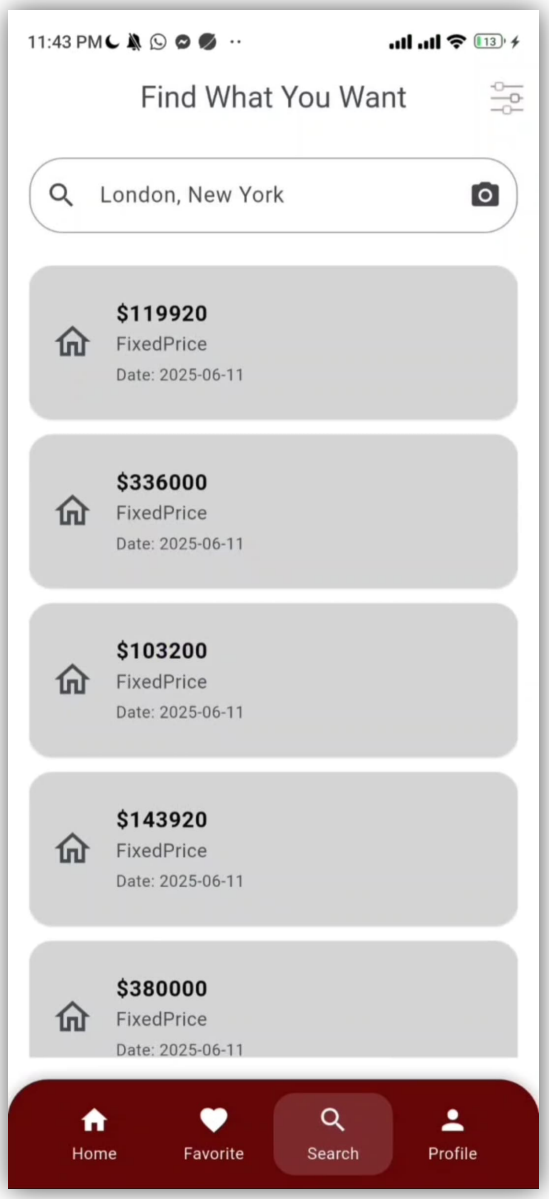


Figure 29 Search Results Listings View

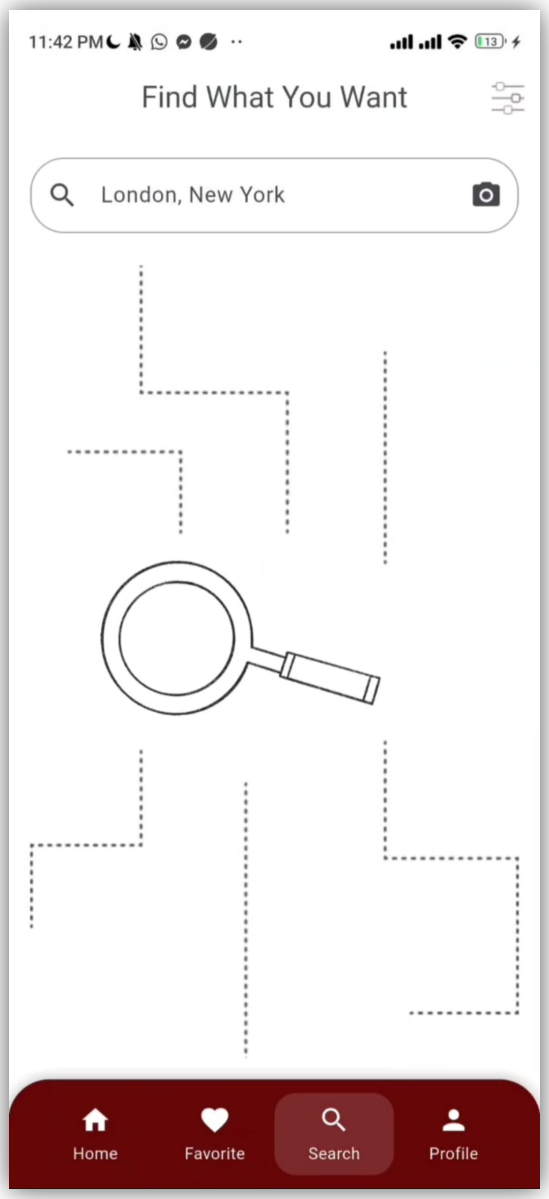


Figure 30 Initial State of Search Screen

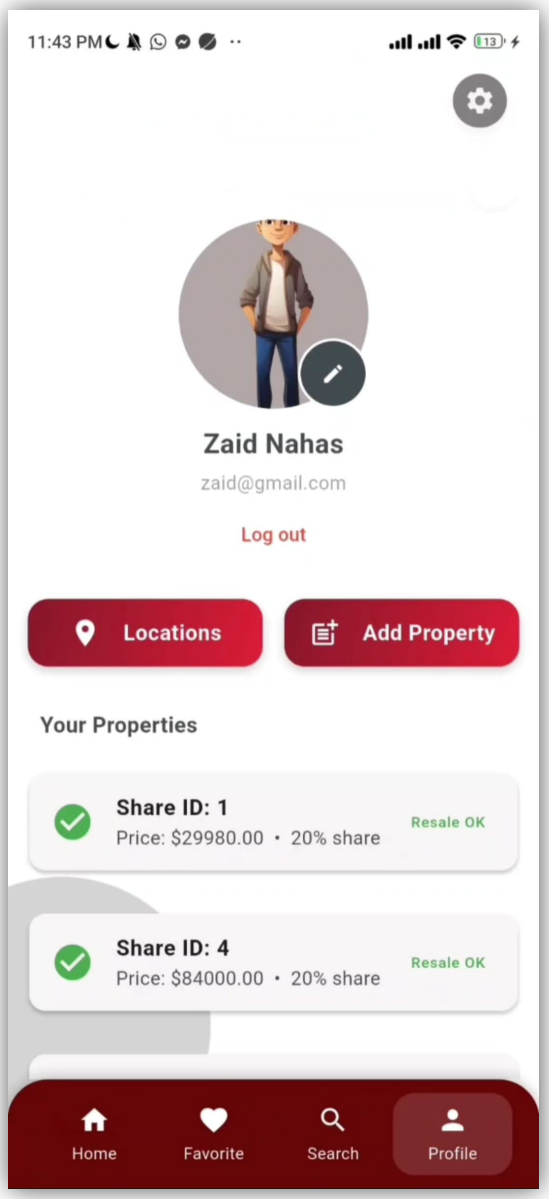


Figure 31 Profile Screen displaying Personal Info

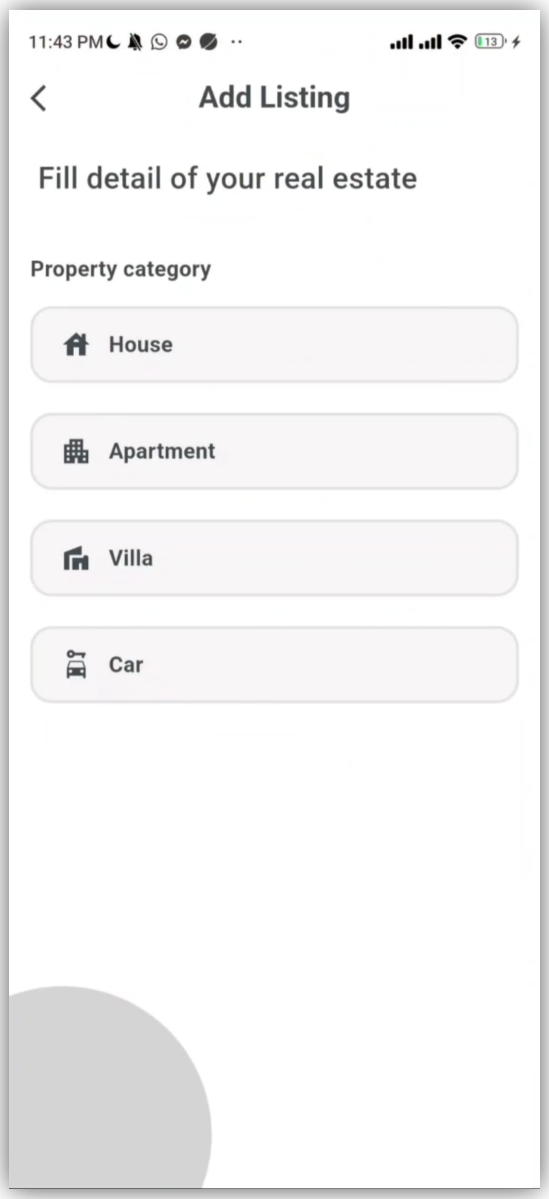


Figure 32 "Add Listing" Screen for Selecting a Property Type

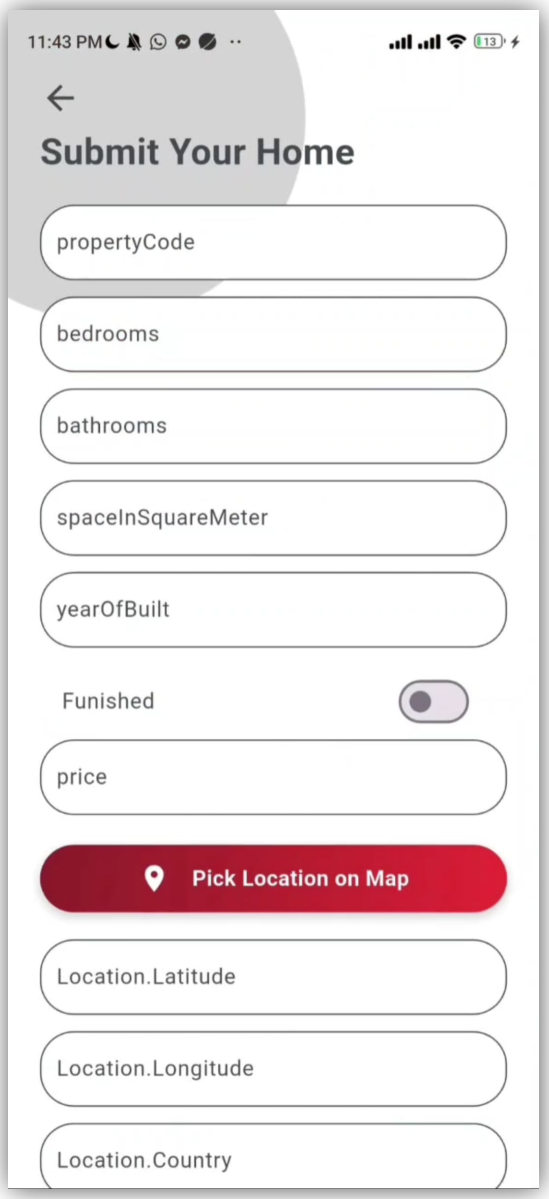


Figure 33 Submission Form for Entering Primary Property Detail

Figure 34 Map Interface for Picking the Property's Location

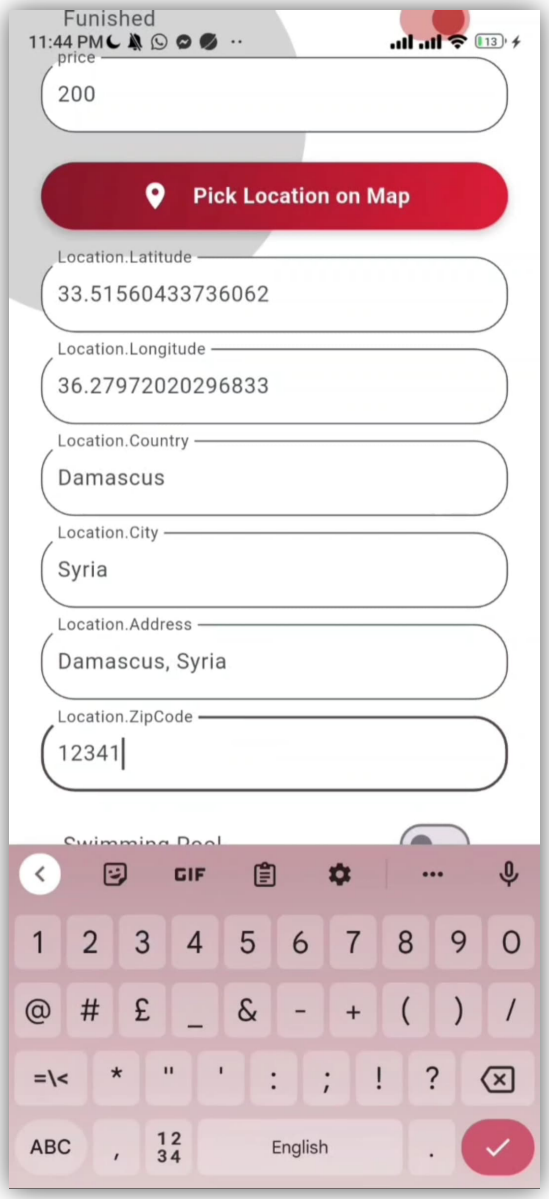


Figure 35 Submission Form with Location Fields Auto-Filled from the Map

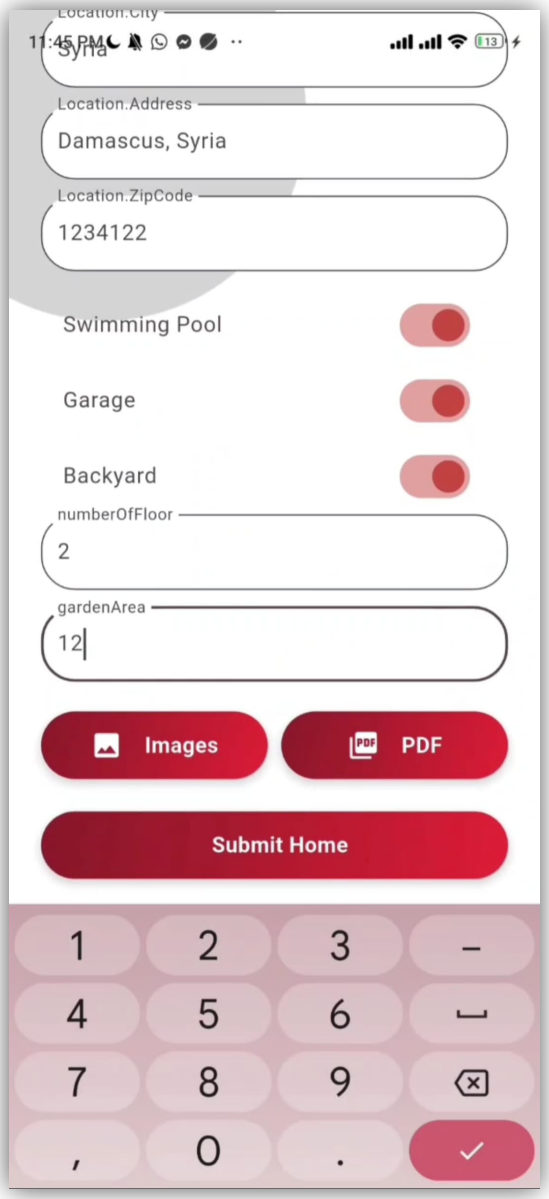


Figure 36 Final Section of the Submission Form for Amenities and Document Uploads

5.5 Algorithms

The core functionalities of the "TrustAsset" platform are powered by several key algorithms that handle the business logic.

- **Duplicate Asset Prevention:** When a user submits a new asset, the system executes a validation algorithm. It extracts the unique identifier (Property Code or Chassis Number) from the submitted data and performs a query on the database to check for its existence. If a match is found, the submission is rejected; otherwise, the listing request is created.

- **Group Sale Approval:** When a co-owner initiates a group sale request, the system triggers a time-sensitive consensus algorithm. It sends a notification to all invited co-owners and starts a five-minute timer. The system monitors the responses. If any co-owner responds with "Reject," or if the timer expires before all have responded with "Accept," the algorithm automatically cancels the request. The sale only proceeds if unanimous consent is achieved within the time limit.
- **Secure Purchase Authorization:** The purchase workflow is protected by a multi-party authorization algorithm. Upon a purchase initiation, the system notifies all parties (buyer and all sellers) and requires each to provide their secret private key. The system validates each submitted key against the corresponding stored hash. The purchase is only flagged as "authorized" if all keys are correct; otherwise, the transaction is immediately canceled.

5.6 Solutions

The implementation of the system design and algorithms was achieved through specific technical solutions and patterns.

- **RESTful API Service:** The communication between the client applications, the main backend, and the AI server is handled via a well-defined RESTful API. This decouples the components, allowing for independent development and deployment.
- **JWT for Secure Authentication:** User authentication is managed using JSON Web Tokens (JWT). After a successful login, the client receives a JWT, which must be included in the header of all subsequent requests to protected endpoints, ensuring stateless and secure session management.
- **Stored Procedure for Transactional Integrity:** The complex logic of transferring ownership shares is encapsulated in the `ReplaceRequestShares` T-SQL stored procedure. This ensures that the multi-step process (deactivating old shares, creating new ones) is atomic. If any step fails, the entire transaction is automatically rolled back by the database, guaranteeing data integrity.

- **Optimized AI Service:** The Python-based AI service is optimized for performance and low-memory environments. All machine learning models and necessary data artifacts are pre-loaded into memory upon server startup to ensure low-latency responses. Furthermore, the data loading process itself uses memory-efficient data types and explicit calls to Python's garbage collector to manage RAM usage effectively.

Chapter 7: Testing and Discussion

This chapter details the datasets used for model development, the testing methodologies employed, the experimental process of model building, and a thorough discussion of the results obtained for each of the core AI components.

7.1 Model Building and Experiments: Image-Based Search

7.1.1 Dataset Used

The primary dataset used for developing the image-based vehicle search model was a large collection of car images.

Table 17 Dataset Specification for the Image-Based Search Model

Domain	Name	Description	Source	Size	Year
Vehicle	Over 20 Car Brands Dataset	A rich collection of car images spanning more than 20 internationally recognized car brands.	Kaggle	1.6 GB	2022

7.1.2 Experimental Process

A series of iterative experiments were conducted to develop a highly accurate model for fine-grained vehicle classification. The process began with simple architectures built from scratch and progressively moved to more complex transfer learning and fine-tuning techniques.

- **Initial Experiments from Scratch:** The initial approach involved building several Convolutional Neural Network (CNN) architectures from the ground up to establish a baseline. Six distinct experimental models were created, each with a slightly different configuration of convolutional, pooling, and dense layers. This iterative process helped in understanding the complexity of the dataset and establishing a performance benchmark for subsequent, more advanced models.

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(18, activation='softmax'),
])

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

First Experiment

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(18, activation='softmax'),
])

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

Second Experiment

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(18, activation='softmax'),
])

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

Thrid Experiment

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),

    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    layers.Conv2D(1028, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    layers.Conv2D(1028, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),

    # Global Average Pooling
    layers.GlobalAveragePooling2D(),

    layers.Dense(512, activation='relu'),
    layers.Dense(18, activation='softmax'),
])
```

Fourth Experiment

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(5, activation='softmax'),
])

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

Fifth Experiment

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(5, activation='softmax'),
])
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Sixth Experiment

Figure 37 Iterative Experiments for Building a CNN from Scratch

Table 18 Summary of Hyperparameter Tuning Experiments for the Image-Based Search Model

Experiment	**Model Used**	**Sample Count**	**Activation**	**Optimizer**	Accuracy	**Epochs**
1	**Feature-extraction** VGG16 + Flatten + Dense(256) + Dropout(0.5) + Dense(19)	2000 train, 1000 val, 1000 test (images)	relu (hidden), softmax (output)	RMSprop(lr=2e-5)	11.70 %	30
2	**Feature-extraction** VGG16 + Flatten + Dense(256) + Dense(19) (–Dropout)	2000 train, 1000 val, 1000 test (images)	relu (hidden), softmax (output)	RMSprop(lr=2e-5)	27.00 %	30
3	**Transfer-learning** VGG16 (frozen) + Flatten + Dense(512) + Dense(256) + Dense(18)	total images via train_generator	relu (all hidden), softmax (output)	Adam (default lr)	31.70 %	30
4	**Fine-tuning** (resume) – loading saved model (cars.h5) + Adam(lr=1e-4)	same train_generator	relu (hidden), softmax (output)	Adam(lr=0.0001)	47.64 %	10
5	**Transfer-learning** VGG19 (frozen) + Flatten + Dense(512) + Dense(256) + Dense(5)	≈ total images via train_generator	relu (all hidden), softmax (output)	Adam (default lr)	71.39 %	30
6	**Transfer-learning** VGG19 (frozen) + Flatten + Dense(512) + Dense(5) (–second hidden layer)	≈ total images via train_generator	relu (hidden), softmax (output)	Adam (default lr)	86.67 %	40

- **Transfer Learning with VGG16:** Following the baseline experiments, transfer learning was employed using the pre-trained VGG16 model. This approach leverages features learned from a massive dataset (ImageNet) to improve performance on our specific task. Several configurations were tested to find the optimal strategy.

Features from VGG16 + Dense classifier with Dropout

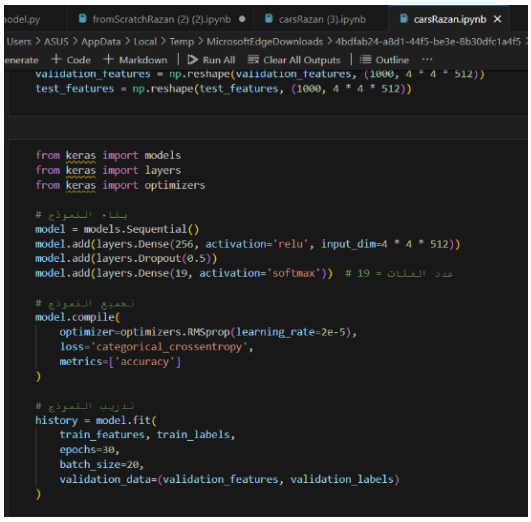


Figure 38 Keras Model Definition for Feature Extraction with a Dense Classifier

Transfer Learning with frozen conv_base and Flatten + Dense layers

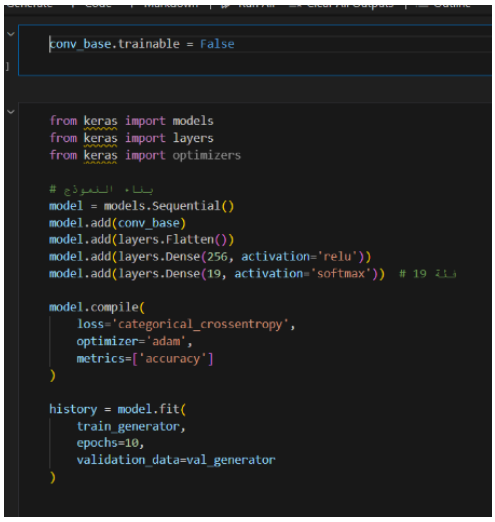


Figure 39 Transfer Learning with a Frozen Convolutional Base

Transfer Learning with frozen conv_base, Flatten + Dense layers, class weighting (class_weight), and a longer training period.

```python
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)

class_weights = dict(enumerate(class_weights))
class_weights
```

```
{0: np.float64(0.8851851851851852),
 1: np.float64(1.4281330749354004),
 2: np.float64(1.279369212962963),
 3: np.float64(0.9429515888249094),
 4: np.float64(0.9709046991655688),
 5: np.float64(1.0129438717067583),
 6: np.float64(0.9689896997589306),
 7: np.float64(0.831265275427712),
 8: np.float64(0.9805943668219117),
 9: np.float64(0.8695181907571288),
 10: np.float64(0.908092010679811),
 11: np.float64(0.8618908382066277),
 12: np.float64(1.1753056884635833),
 13: np.float64(0.9484126984126984),
 14: np.float64(1.387790332705587),
 15: np.float64(1.116540404040404),
 16: np.float64(0.813373804267844),
 17: np.float64(1.119083776259175)}
```

Figure 40 Calculation of Class Weights for Imbalanced Data

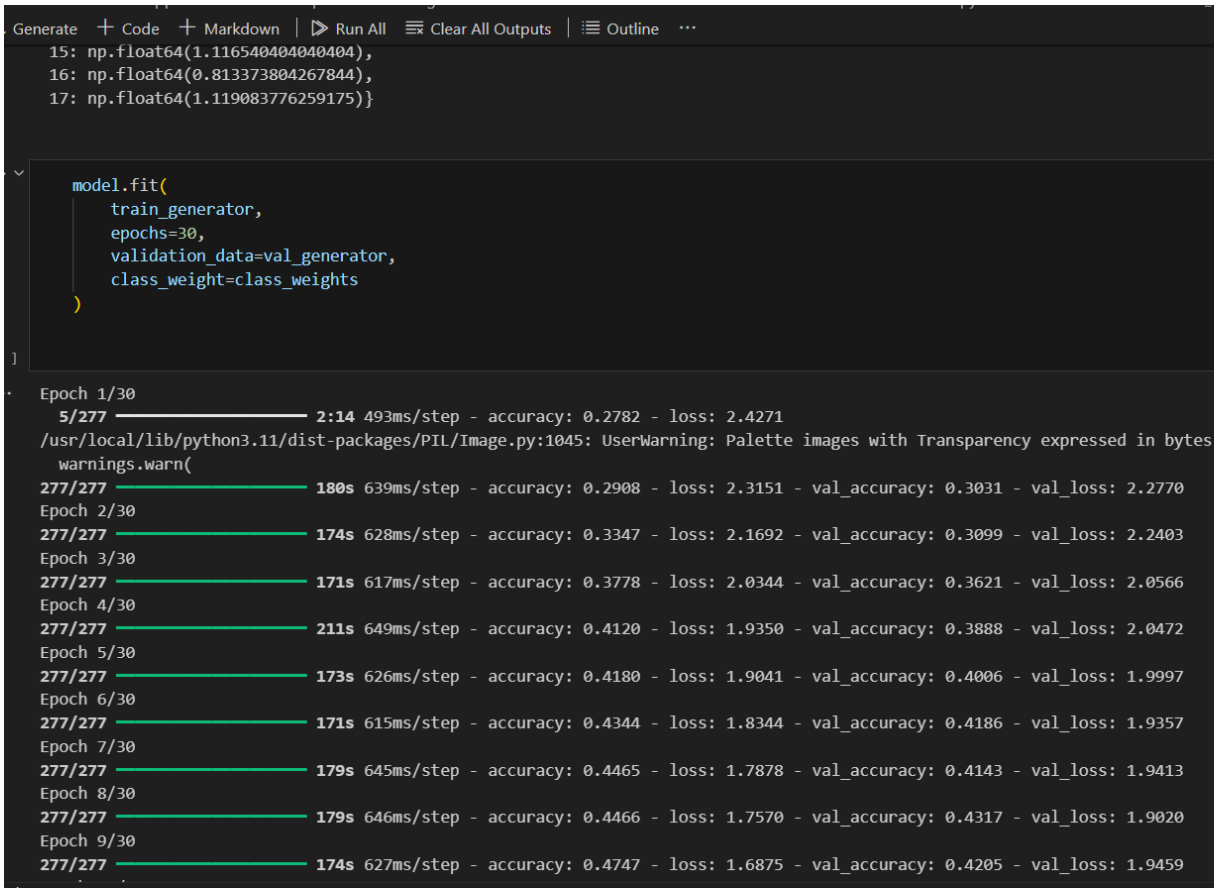


Figure 41 Training Progress with Class Weights Applied

Fine-Tuning with Xception: The Xception architecture, known for its high performance and efficiency, was selected for the final stage of experimentation. The process involved fine-tuning, where some of the top layers of the pre-trained model are "unfrozen" and trained alongside the new classifier.

Xception with 5 classes

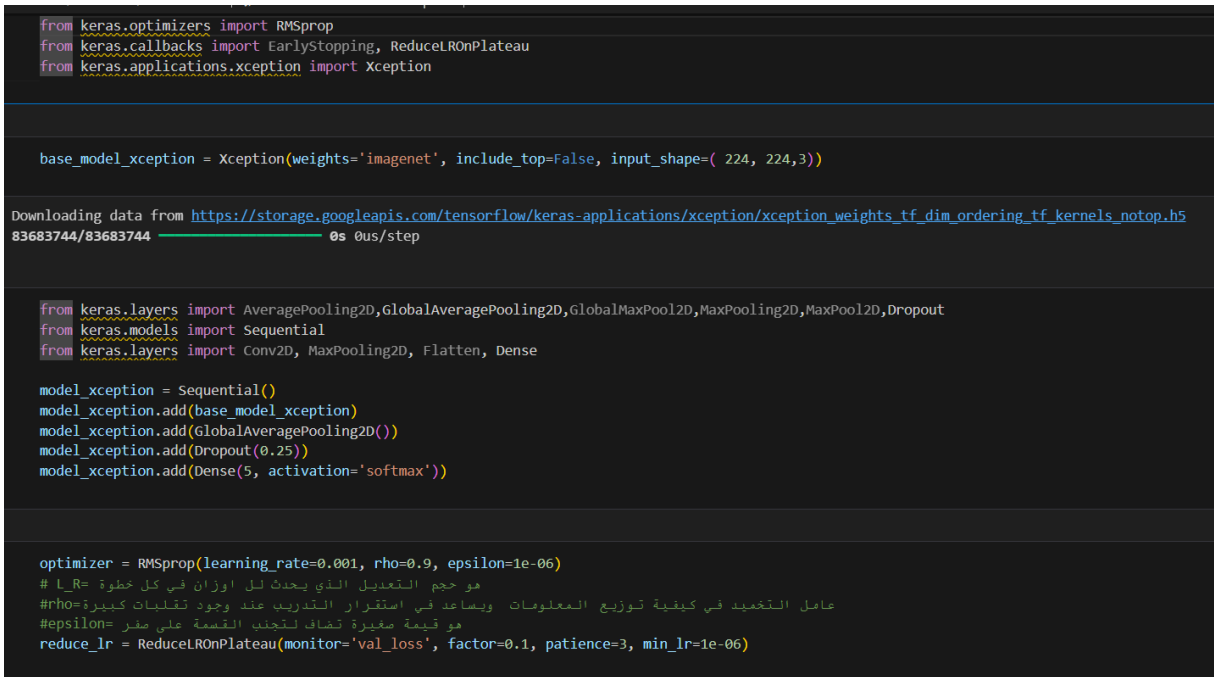


Figure 42 Fine-Tuning the Xception Model for 5 Classes (1)

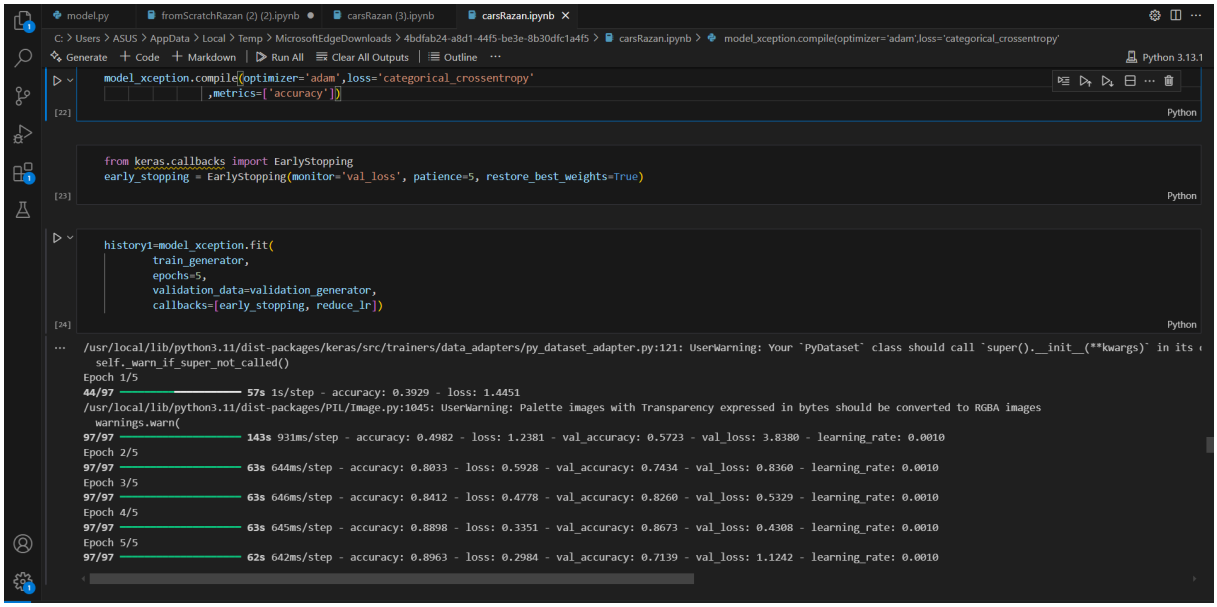


Figure 43 Fine-Tuning the Xception Model for 5 Classes (2)

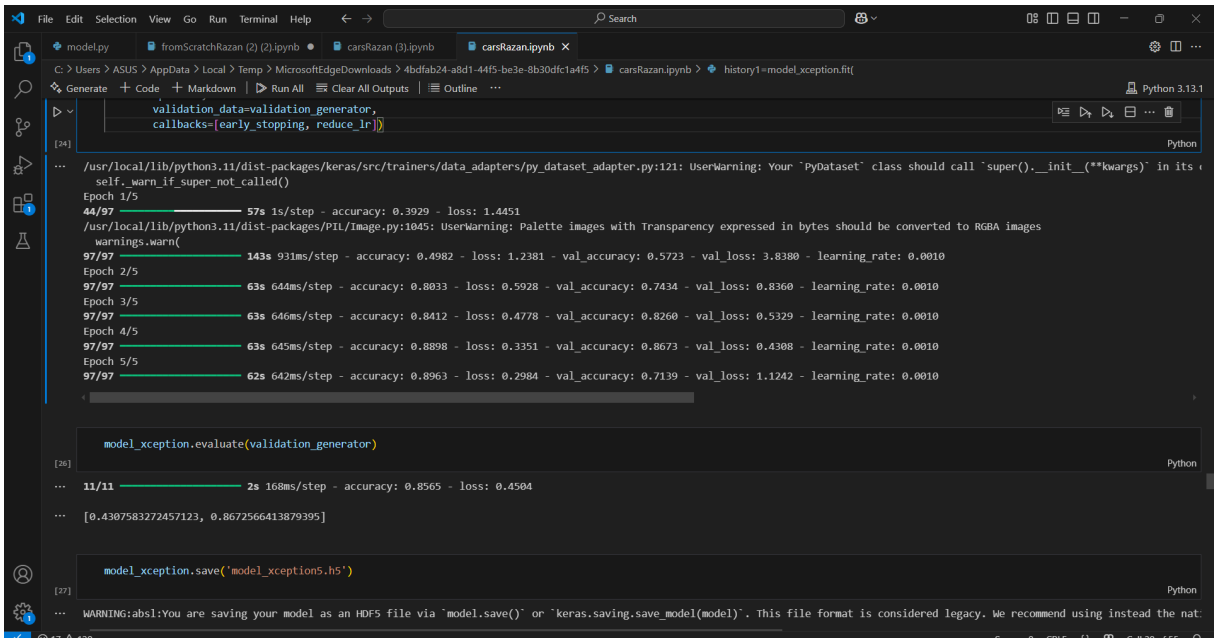


Figure 44 Evaluation and Saving of the Xception Model

Xception customized for 18 classes

```
from keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.applications.xception import Xception
base_model_xception = Xception(weights='imagenet', include_top=False, input_shape=( 224, 224,3))
from keras.layers import AveragePooling2D,GlobalAveragePooling2D,GlobalMaxPool2D,MaxPooling2D,MaxPool2D,Dropout
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model_xception = Sequential()
model_xception.add(base_model_xception)
model_xception.add(GlobalAveragePooling2D())
model_xception.add(Dropout(0.25))
model_xception.add(Dense(18, activation='softmax'))
optimizer = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-06)
# L_R= هو حجم التعديل الذي يحدث لل اوزان في كل خطوة
#rho=عامل التخميد في كيفية توزيع المعلومات  ويساعد في استقرار التدريب عند وجود تقلبات كبيرة
#epsilon= هو قيمة صغيرة تضاف لتجنب القسمة علي صفر
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, min_lr=1e-06)

model_xception.compile(optimizer='adam',loss='categorical_crossentropy'
                       ,metrics=['accuracy'])
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model_xception.compile(optimizer='adam',loss='categorical_crossentropy'
                       ,metrics=['accuracy'])
history1=model_xception.fit(
        train_generator,
        epochs=10,
        validation_data=validation_generator,
        callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/10
217/217 ━━━━━━━━━━━━━━━━━━━━ 217s 814ms/step - accuracy: 0.2500 - loss: 2.4107 - val_accuracy: 0.2144 - val_loss: 4.8741 - learning_rate: 0.0010
120
```

Figure 45 Customizing the Xception Model for 18 Classes

7.2 Model Building and Experiments: Price Prediction

7.2.1 Datasets Used

The price prediction models were trained and validated on two separate datasets, one for real estate and one for vehicles.

Table 19 Dataset Specification for the Price Prediction Model

Domain	Name	Description	Source	Size	Year
Real Estate	House Prices - Advanced Regression Techniques	Comprehensive data on residential home sales in Ames, Iowa, with 79 explanatory variables.	Kaggle	~500 KB	2017
Vehicle	Used Car Price Prediction Dataset	Detailed information about used cars, including brand, model, year, mileage, and price.	Kaggle	~300 KB	2022

7.2.2 Experimental Process and Results

The price prediction models for both cars and real estate were developed using powerful gradient boosting frameworks and evaluated using standard regression metrics.

Car Price Prediction: The LightGBM and XGBoost models were trained on the used car dataset. Both models demonstrated high accuracy. The LightGBM model, for instance, achieved a high R^2 score of **0.979** on the testing data.

Real Estate Price Prediction: For real estate, an ensemble model combining XGBoost, LightGBM, and CatBoost was trained. To ensure the model is robust, it was evaluated using 5-fold cross-validation, achieving an average R^2 score of **0.9019** on the validation data. The small difference between the training and validation scores indicates that the model generalizes well to unseen data.

```
LightGBM performance on testing data:
MAE: 1728.9869827551317
MSE: 10046148.992334792
R² : 0.9793451080784007

 LightGBM performance on training data:
MAE: 1160.3803341454295
MSE: 3957055.936033567
R² : 0.9913912074614833

 Comparison of the first 5 results (LightGBM):
     Actual   XGB_Predicted   LGBM_Predicted
0   88900.0    91965.351562     90170.678000
1   51000.0    49967.101562     50221.656189
2   75000.0    58393.953125     59279.735073
3   71999.0    65684.554688     65511.884996
4   59950.0    58759.578125     60283.391605

 LightGBM training time: 0.21 seconds
/usr/local/lib/python3.11/dist-packages/sklearn/utils/d
  warnings.warn(
```

Figure 46 Performance Metrics of the LightGBM Price Prediction Model

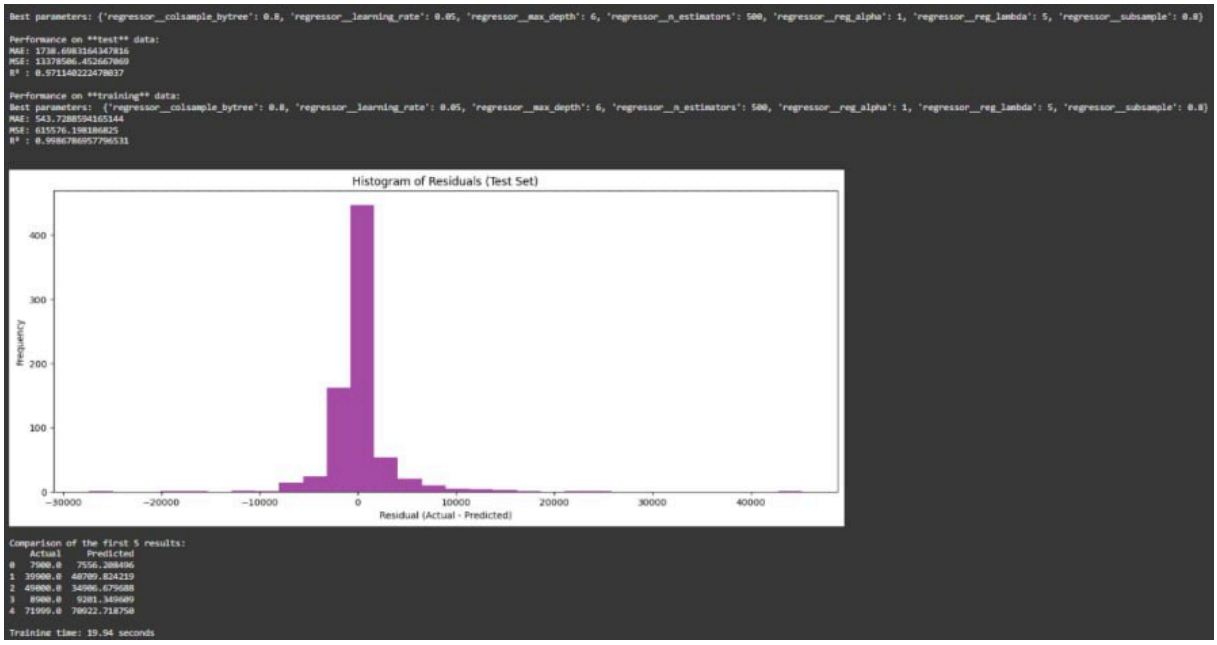


Figure 47 Performance Metrics and Residuals Plot for the XGBoost Price Prediction

```
Best parameters: {'regressor__colsample_bytree': 0.8, 'regressor__learning_rate': 0.05, 'regressor__max_depth': 6, 'regressor__n_estimators': 500, 'regressor__reg_alpha': 1, 'regressor__reg_lambda': 5, 'regressor__subsample': 0.8}

Performance on **test** data:
MAE: 1738.6983164347816
MSE: 13378506.452667069
R² : 0.971140222478837

Performance on **training** data:
Best parameters:  {'regressor__colsample_bytree': 0.8, 'regressor__learning_rate': 0.05, 'regressor__max_depth': 6, 'regressor__n_estimators': 500, 'regressor__reg_alpha': 1, 'regressor__reg_lambda': 5, 'regressor__subsample': 0.8}
MAE: 543.7288594165144
MSE: 615576.198186825
R² : 0.9986786957796531


Comparison of the first 5 results:
    Actual     Predicted
0   7900.0   7556.208496
1  39900.0  40709.824219
2  49000.0  34906.679688
3   8900.0   9201.349609
4  71999.0  70922.718750

Training time: 20.92 seconds
```

Figure 48 Performance Metrics of the Tuned Price Prediction Model

Fold (1)

This code first prepares the data by handling missing values and defining features. It then sets up an ensemble of three gradient boosting models and uses a 5-fold cross-validation loop to repeatedly train and evaluate the model, ensuring a robust performance assessment.

```
mae_train_scores = []
r2_train_scores = []
mae_val_scores = []
r2_val_scores = []

#Evaluating The Training
for fold, (train_idx, val_idx) in enumerate(kf.split(X_processed)):
    print(f"\n-- Fold {fold+1} --")
    X_train_fold, X_val_fold = X_processed[train_idx], X_processed[val_idx]
    y_train_fold, y_val_fold = y.iloc[train_idx], y.iloc[val_idx]

    ensemble.fit(X_train_fold, y_train_fold)

    y_train_pred_log = ensemble.predict(X_train_fold)
    y_val_pred_log = ensemble.predict(X_val_fold)

    y_train_true = np.expm1(y_train_fold)
    y_val_true = np.expm1(y_val_fold)
    y_train_pred = np.expm1(y_train_pred_log)
    y_val_pred = np.expm1(y_val_pred_log)

    mae_train = mean_absolute_error(y_train_true, y_train_pred)
    r2_train = r2_score(y_train_true, y_train_pred)
    mae_val = mean_absolute_error(y_val_true, y_val_pred)
    r2_val = r2_score(y_val_true, y_val_pred)

    mae_train_scores.append(mae_train)
    r2_train_scores.append(r2_train)
    mae_val_scores.append(mae_val)
    r2_val_scores.append(r2_val)
```

Figure 49 Preprocessing data and defining an ensemble of XGBoost, LightGBM, and CatBoost models

```
MAE on training data: 5779.96
R² on training data: 0.9812
MAE on validation data: 12070.40
R² on validation data: 0.9190

Comparison of the first 5 results from the Fold:
   Actual     Predicted
140000.0 185964.454858
143000.0 144341.995913
129500.0 128293.547538
279500.0 220075.601166
160000.0 148793.611745
```

Figure 50 Implementing a 5-fold cross-validation loop to train the ensemble model and gather performance score

Fold (2)

A custom `MedianVotingRegressor` is created to combine model predictions by taking their median. The code then loads the training data, cleans it by removing outliers from the 'SalePrice' column, and begins the cross-validation process, with the output showing the model's initial performance on the first fold.

```
class MedianVotingRegressor(RegressorMixin, BaseEstimator):
    def __init__(self, estimators):
        self.estimators = estimators

    def fit(self, X, y):
        self.fitted_ = []
        for name, est in self.estimators:
            model = clone(est)
            model.fit(X, y)
            self.fitted_.append((name, model))
        return self

    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for _, model in self.fitted_
        ])
        return np.median(predictions, axis=1)

#Uploading the dataset
train_path = "/content/train.csv"
train_df = pd.read_csv(train_path)

Q1 = train_df['SalePrice'].quantile(0.25)
Q3 = train_df['SalePrice'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
train_df = train_df[(train_df['SalePrice'] >= lower_bound) & (train_df['SalePrice'] <= upper_bound)]

y = np.log1p(train_df['SalePrice'])
X = train_df.drop(columns=['SalePrice', 'Id'])
```

Figure 51 Defining a custom MedianVotingRegressor and preparing the dataset by removing outliers

```
MAE on training data: 5698.23
R² on training data: 0.9818
MAE on validation data: 12099.13
R² on validation data: 0.9034

Comparison of the first 5 results from the Fold:
   Actual     Predicted
154000.0 145888.112955
 40000.0  82466.922661
149350.0 132913.846722
179900.0 194906.562253
165500.0 182249.198773
```

Figure 52 The model's performance results from the first fold of cross-validation

Fold (3)

After the cross-validation loop completes, this code calculates and displays the final performance summary. It averages the metrics from all five folds and assesses the risk of overfitting by comparing the training and validation errors, concluding that the model generalizes well.

```
    print(f"MAE on training data: {mae_train:.2f}")
    print(f"R² on training data: {r2_train:.4f}")
    print(f"MAE on validation data: {mae_val:.2f}")
    print(f"R² on validation data: {r2_val:.4f}")


    print("\nComparison of the first 5 results from the Fold:")

    results_df = pd.DataFrame({
        'Actual': y_val_true[:5].values,
        'Predicted': y_val_pred[:5]
    })
    print(results_df.to_string(index=False))
#Performance
print("\n=== Performance summary across 5 folds ===")
print(f"Average MAE on training data: {np.mean(mae_train_scores):.2f} ± {np.std(mae_train_scores):.2f}")
print(f"Average R² on training data: {np.mean(r2_train_scores):.4f} ± {np.std(r2_train_scores):.4f}")
print(f"Average MAE on validation data: {np.mean(mae_val_scores):.2f} ± {np.std(mae_val_scores):.2f}")
print(f"Average R² on validation data: {np.mean(r2_val_scores):.4f} ± {np.std(r2_val_scores):.4f}")

diff = np.mean(mae_val_scores) - np.mean(mae_train_scores)

print(f"MAE difference between training and validation: {diff:.2f}")
if diff > 10000:
    print("Warning: There might be Overfitting.")
else:
    print("No strong evidence of Overfitting.")
```

Figure 53 Code to calculate and print the average performance across all folds and check for overfitting

```
MAE on training data: 5554.63
R² on training data: 0.9824
MAE on validation data: 13658.87
R² on validation data: 0.8825

Comparison of the first 5 results from the Fold:
  Actual     Predicted
181500.0 170757.748426
129900.0 138742.010334
157000.0 149708.556768
132000.0 141199.600820
 90000.0 112289.147165

=== Performance summary across 5 folds ===
Average MAE on training data: 5654.08 ± 84.25
Average R² on training data: 0.9816 ± 0.0005
Average MAE on validation data: 12890.65 ± 794.90
Average R² on validation data: 0.9019 ± 0.0116
MAE difference between training and validation: 7236.57
✅ No strong evidence of Overfitting.
```

Figure 54 The final performance summary, showing averaged metrics across 5 folds and no evidence of overfitting

Fold (4)

These images show the performance metrics from two different, individual folds within the cross-validation process. Each output displays the Mean Absolute Error (MAE) and R^2 score for that specific training/validation split, showing how performance varies slightly across different subsets of the data.

```
#Dealig with the nulls
for col in X.select_dtypes(include=['float64', 'int64']).columns:
    X[col] = X[col].fillna(X[col].median())
for col in X.select_dtypes(include=['object']).columns:
    X[col] = X[col].fillna('Missing')

categorical_features = X.select_dtypes(include=['object']).columns.tolist()
numerical_features = X.select_dtypes(include=['float64', 'int64']).columns.tolist()

#Preprocessing
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features),
    ('num', 'passthrough', numerical_features)
])

X_processed = preprocessor.fit_transform(X)

#Defining my models
models = [
    ('xgb', XGBRegressor(random_state=42, n_jobs=-1, n_estimators=800, learning_rate=0.03, max_depth=4,
                         subsample=0.8, colsample_bytree=0.8, reg_alpha=0.1, reg_lambda=1)),
    ('lgbm', LGBMRegressor(random_state=42, n_estimators=800, learning_rate=0.03, max_depth=4,
                           subsample=0.8, colsample_bytree=0.8, reg_alpha=0.1, reg_lambda=1)),
    ('catboost', CatBoostRegressor(verbose=0, random_state=42, iterations=800, learning_rate=0.03, depth=4,
                                   l2_leaf_reg=3.0))
]

ensemble = MedianVotingRegressor(estimators=models)

kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

Figure 55 Performance results from a single fold, showing a validation MAE of 12099.13

```
MAE on training data: 5670.35
R² on training data: 0.9816
MAE on validation data: 12636.37
R² on validation data: 0.9026

Comparison of the first 5 results from the Fold:
   Actual     Predicted
250000.0 295895.487831
144000.0 129424.125439
149000.0 147280.316457
159000.0 153178.303618
139000.0 121270.981551
```

Figure 56 A snapshot of the model's predictive accuracy on another fold of the data

7.3 Model Building and Experiments: Recommendation System

This section details the development of the recommendation engines for both real estate and vehicles, covering the full pipeline from data preprocessing and feature engineering to model training, refinement, and validation.

7.3.1 Datasets Used

The recommendation systems were developed using large-scale datasets containing property/vehicle features and user interaction data.

Table 20 Specification for the Recommendation System

Domain	Name	Description	Source	Size	Year
Real Estate	USA Real Estate Listings	US housing listings with features such as price, number of beds, bathrooms, lot size, and location details.	Kaggle	2,226,382 records	c. 1960-2022
Vehicle	Vehicle Sales Data	US used vehicle sales data including price, condition, seller, and key vehicle attributes (make, model, etc.).	Internal	558,837 records	2024
Vehicle	Vehicle Reviews Data	Consolidated user reviews and ratings for various vehicle models.	Internal	284,715 records	2018

7.3.2 Data Preprocessing and Feature Engineering

A crucial first step was to systematically clean and transform the raw data to create high-quality, model-ready features.

- **Real Estate Domain:** The initial analysis revealed significant data quality issues. A multi-stage preprocessing pipeline was executed, which included correcting invalid values, imputing missing data using group-wise medians, normalizing skewed numerical features using Log and Box-Cox transformations, and finally applying a mix of One-Hot and Ordinal encoding for categorical features. The final dataset was reduced minimally to 2,223,586 records, with the feature count increasing from 12 to 18.

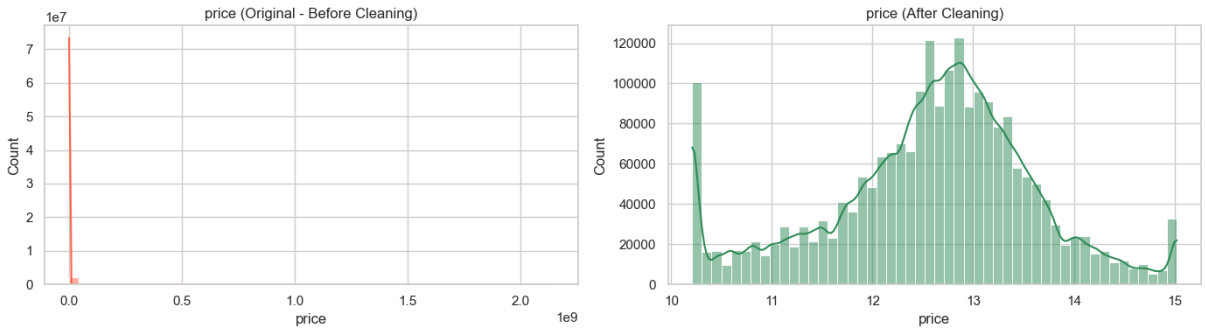
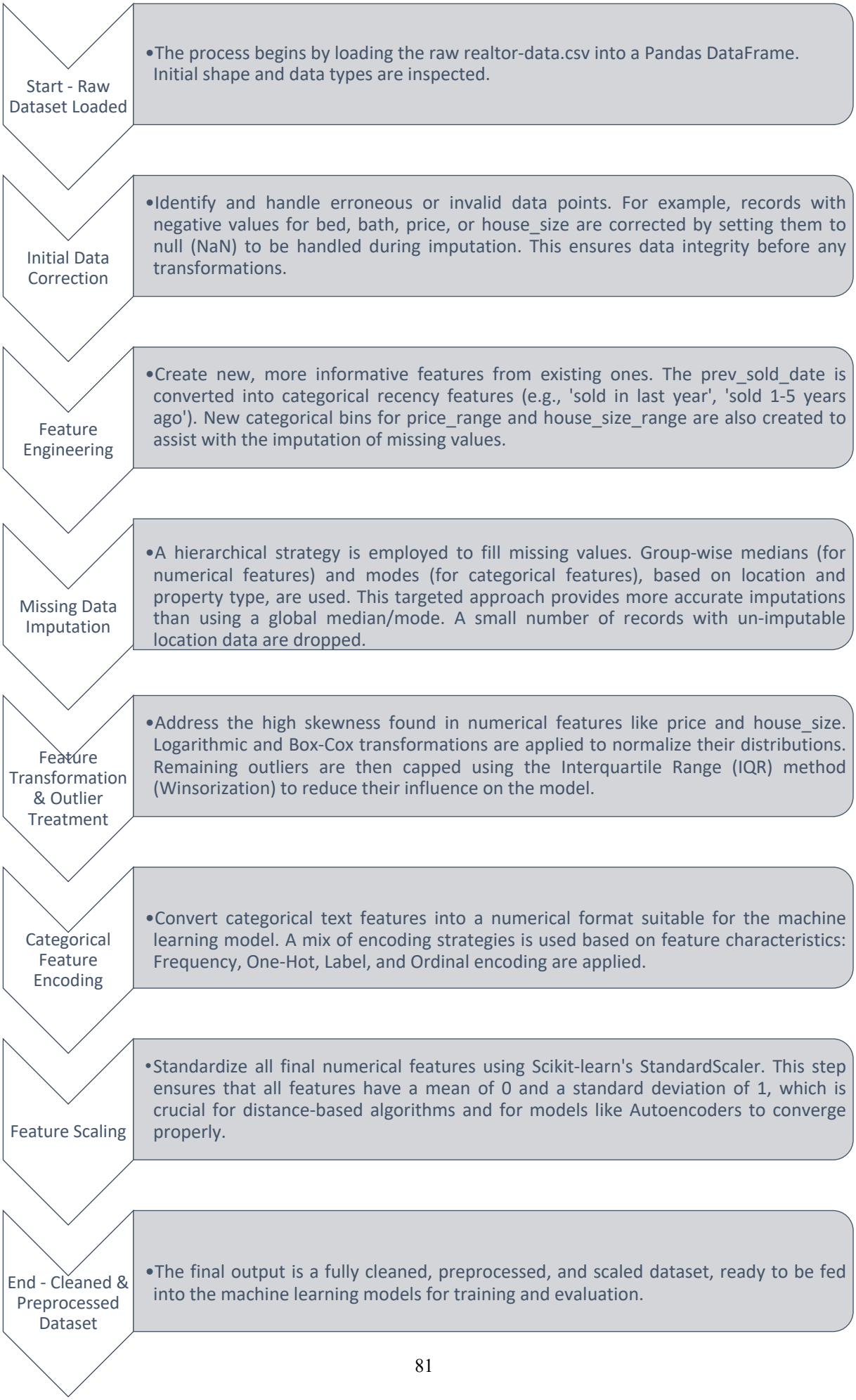


Figure 57 Distribution of the 'price' Feature Before and After Log


Start - Raw Dataset Loaded
•The process begins by loading the raw realtor-data.csv into a Pandas DataFrame. Initial shape and data types are inspected.
Initial Data Correction
•Identify and handle erroneous or invalid data points. For example, records with negative values for bed, bath, price, or house_size are corrected by setting them to null (NaN) to be handled during imputation. This ensures data integrity before any transformations.
Feature Engineering
•Create new, more informative features from existing ones. The prev_sold_date is converted into categorical recency features (e.g., 'sold in last year', 'sold 1-5 years ago'). New categorical bins for price_range and house_size_range are also created to assist with the imputation of missing values.
Missing Data Imputation
•A hierarchical strategy is employed to fill missing values. Group-wise medians (for numerical features) and modes (for categorical features), based on location and property type, are used. This targeted approach provides more accurate imputations than using a global median/mode. A small number of records with un-imputable location data are dropped.
Feature Transformation & Outlier Treatment
•Address the high skewness found in numerical features like price and house_size. Logarithmic and Box-Cox transformations are applied to normalize their distributions. Remaining outliers are then capped using the Interquartile Range (IQR) method (Winsorization) to reduce their influence on the model.
Categorical Feature Encoding
•Convert categorical text features into a numerical format suitable for the machine learning model. A mix of encoding strategies is used based on feature characteristics: Frequency, One-Hot, Label, and Ordinal encoding are applied.
Feature Scaling
•Standardize all final numerical features using Scikit-learn's StandardScaler. This step ensures that all features have a mean of 0 and a standard deviation of 1, which is crucial for distance-based algorithms and for models like Autoencoders to converge properly.
End - Cleaned & Preprocessed Dataset
•The final output is a fully cleaned, preprocessed, and scaled dataset, ready to be fed into the machine learning models for training and evaluation.

- **Vehicle Domain:** The preprocessing for the vehicle collaborative filtering model focused on creating a clean user-item interaction dataset. This involved dropping records with missing user IDs or ratings from the initial 284,715 reviews. The unstructured `Vehicle_Title` was parsed using a reference-based approach to extract structured attributes (year, make, model), resulting in a final dataset of 170,285 clean user-item-rating interactions.

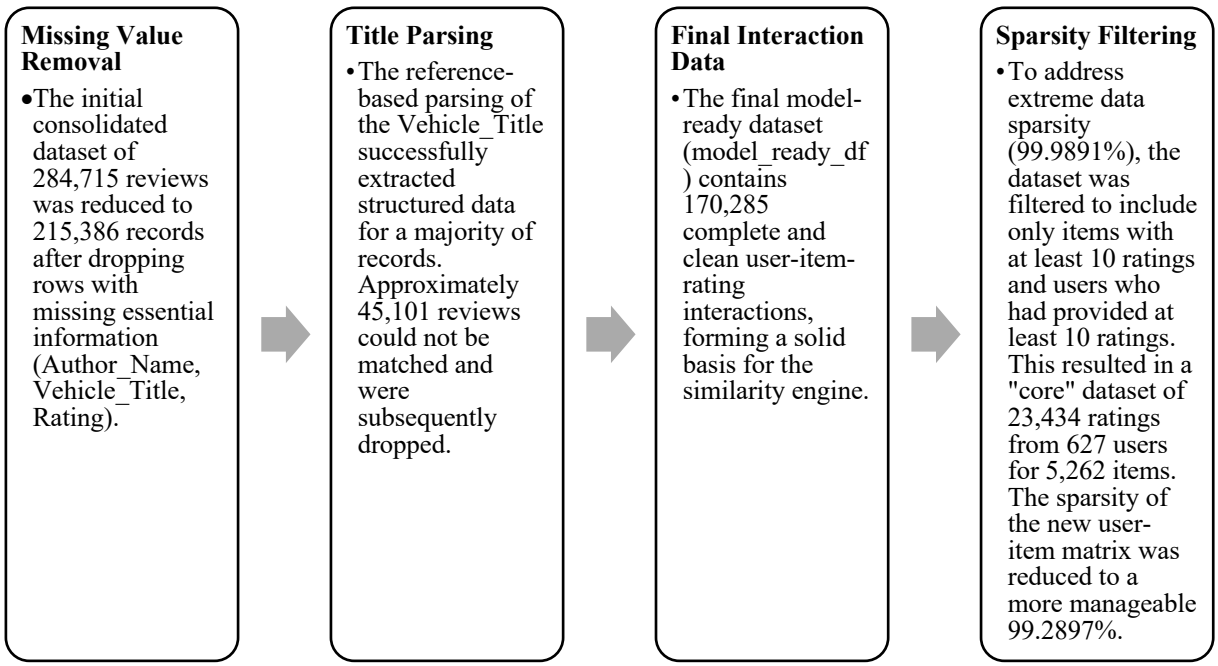
  Addressing Data Sparsity: The resulting user-item matrix was extremely sparse (99.9891%). To mitigate this, a filtering methodology was applied to create a denser "core" dataset. Items with fewer than 10 ratings and users with fewer than 10 ratings were removed. This process significantly reduced the matrix dimensions and sparsity to a more manageable 99.2897%, allowing for a more meaningful training of the k-NN model.

```
# --- 1. Filter by Item Popularity ---

# Calculate how many ratings each item has
item_counts = model_ready_df['item_id'].value_counts()
# Keep only items that have 10 or more ratings
popular_items = item_counts[item_counts >= 10].index
```

```
--# --- 2. Filter by User Activity ---

# Calculate how many ratings each user has made in the *new* filtered dataframe
user_counts = df_filtered_items['Author_Name'].value_counts()
# Keep only users who have made 10 or more ratings
active_users = user_counts[user_counts >= 10].index
df_final_filtered = df_filtered_items[df_filtered_items['Author_Name'].isin(active_users)]
```



7.3.3 Experimental Process and Results

- **Real Estate Content-Based Recommender (Autoencoder):** The Autoencoder model was trained to generate property embeddings. A series of experiments were conducted to tune its hyperparameters to minimize reconstruction error. The final model from Experiment 4, with its significantly lower reconstruction error, was selected as the definitive model for generating property embeddings.

Table 21 Summary of Hyperparameter Tuning Experiments for the Real Estate Autoencoder

Experiment	Embedding Dim	Embedding Activation	Hidden Layers (Neurons)	Dropout Rate	Best Val MSE
Baseline	10	'relu'	(256, 128)	0.3	0.0064
Exp 1	16	'relu'	(256, 128)	0.3	0.0067
Exp 2	10	'linear'	(256, 128)	0.3	0.0048
Exp 3	10	'linear'	(512, 256)	0.3	0.0016
Exp 4 (Final)	**10**	**'linear'**	**(512, 256)**	**0.2**	**0.0009**

Autoencoder Training Code

```
# Final Autoencoder Architecture
INPUT_DIM = 18 # Number of input features
EMBEDDING_DIM = 10

# --- Encoder ---
encoder = models.Sequential([
    layers.Input(shape=(INPUT_DIM,)),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(EMBEDDING_DIM, activation='linear') # Embedding Layer
], name='encoder')

# --- Decoder ---
decoder = models.Sequential([
    layers.Input(shape=(EMBEDDING_DIM,)),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(INPUT_DIM, activation='linear')
], name='decoder')

autoencoder = models.Sequential([encoder, decoder], name='autoencoder')
autoencoder.summary()
```

- **Vehicle Hybrid Recommendation System:** The development of the vehicle recommender was an iterative process that evolved from a pure collaborative filtering approach to a more robust hybrid system.
    - **Initial Collaborative Filtering (k-NN) Model:** The initial test of the k-NN model on the full, sparse user-item matrix produced irrelevant recommendations (e.g., suggesting a Maserati for a Kia Sorento inquiry). This outcome is a classic symptom of the **Data Sparsity Problem**, where the model cannot find meaningful patterns.
    - **Model Refinement:** After retraining the model on the denser, filtered dataset, a verification test showed a significant improvement in relevance, including several other SUVs from a similar era (e.g., '2008_nissan_xterra_s', '2003_honda_element_dx'). This confirmed that filtering the data allowed the model to identify a much stronger, more logical signal from user ratings for *known* items.
    - **Solving the Cold Start Problem:** The refined collaborative model still could not provide recommendations for new or unseen vehicles that had no ratings. To address this "cold start problem," the system was evolved into a **hybrid model**. A **content-based filtering** engine was developed as a fallback. This engine calculates similarity based on vehicle attributes (Make, Year, Price, Horsepower) and is accessible via a dedicated API endpoint (`POST /recommend/unseen`), ensuring that a recommendation can be generated for any vehicle, regardless of its rating history.

7.4 Discussion and Validation

7.4.1 Image-Based Search

The iterative experimental process, starting from simple CNNs and progressing to fine-tuning advanced architectures like Xception, proved to be a successful strategy. The final model's high accuracy validates its capability for fine-grained vehicle classification, which is the technical foundation of an effective image-based search feature. By using transfer learning, the model leveraged features pre-trained on a massive dataset, saving significant training time and achieving a level of performance that would be difficult to obtain from scratch. The model is well-validated for its primary task, though its real-world performance will depend on the quality of user-submitted images.

7.4.2 Price Prediction

The high R^2 scores achieved for both the vehicle (0.979) and real estate (0.9019) models demonstrate that the chosen features are highly predictive of market prices. The use of robust gradient boosting frameworks like LightGBM and XGBoost is validated by these results, confirming their suitability for this type of structured data problem. For the real estate model, the use of 5-fold cross-validation further validates its robustness, confirming that its performance is stable and not the result of overfitting to a particular data split. The models provide a strong basis for an automated valuation tool, offering reliable price estimates to guide both buyers and sellers.

7.4.3 Recommendation System

The development of the recommendation systems provided valuable insights into the practical challenges of building personalized engines.

- **Real Estate (Content-Based):** The autoencoder's development successfully validates the use of deep learning for feature embedding. The final model's low reconstruction error (Validation MSE of 0.0009) indicates its effectiveness in creating rich, dense representations of property features. Qualitative evaluation confirmed that these embeddings lead to logically sound and contextually relevant recommendations, validating the content-based approach for this domain.
- **Vehicle (Hybrid):** The vehicle recommender's journey highlights a critical lesson in machine learning: the importance of data quality over algorithmic complexity. The initial failure of the collaborative filtering model was not due to the algorithm itself, but to the extreme sparsity of the data. The successful refinement of the model after filtering the dataset validates our data preprocessing strategy. This experience demonstrated that addressing the root cause of a data problem is more effective than simply trying a different algorithm. Furthermore, the final pivot to a **hybrid model** that incorporates a content-based engine is a crucial validation of the system's design. It demonstrates a practical understanding of real-world challenges like the "cold start problem" and results in a more robust and complete solution.

Qualitative Validation of Model Refinement

Table 22 Comparison of Recommendation Outputs Before and After Data Filtering

Before Filtering: Recommendations on the Sparse Dataset	***After Filtering: Recommendations on the Denser Core Dataset***
Recommendations for: '2006_kia_sorento_lx' ---------- 1: 2014_chevrolet_cruze_2lt 2: 2017_maserati_quattroporte_s_q4 3: 2008_gmc_sierra_3500hd_sle1 4: 2018_bmw_x5_xdrive35i 5: 2018_bmw_x1_sdrive28i 6: 2009_audi_tt_2.0t_quattro 7: 2000_kia_sephia_ls 8: 1997_honda_prelude_type_sh 9: 2015_ford_fusion_hybrid_s 10: 2017_honda_fit_ex-l	New recommendations for: '2006_kia_sorento_lx' ---------- 1: 2000_kia_sephia_ls 2: 1999_toyota_camry_solara_se 3: 1997_honda_prelude_type_sh 4: 2008_nissan_xterra_s 5: 2009_chevrolet_colorado_lt 6: 2011_volkswagen_jetta_sel 7: 2006_nissan_xterra_x 8: 2011_chevrolet_cruze_ls 9: 2003_honda_element_dx 10: 2012_ford_fiesta_s
The initial model, when trained on the full, sparse dataset, produced highly irrelevant and illogical recommendations. As shown in the figure above, a query for a Kia Sorento returned a list of seemingly random vehicles, including a luxury sedan (`2017_maserati_quattroporte_s_q4`), a heavy-duty truck (`2008_gmc_sierra_3500hd_sle1`), and a compact sports car (`2009_audi_tt_2.0t_quattro`). This outcome is a classic symptom of the **data sparsity problem**, where the model lacks sufficient overlapping user ratings to identify meaningful patterns, resulting in noise-based suggestions.	After the filtering methodology was applied and the model was retrained on the denser "core" dataset, the same query was executed again. The new recommendations, while not perfect, showed a significant improvement in relevance and logical consistency. The list now contained several other SUVs from a similar era, such as the `2008_nissan_xterra_s`, `2006_nissan_xterra_x`, and `2003_honda_element_dx`.

This direct comparison confirms that the data filtering process was a success. By creating a denser interaction matrix, the model was able to identify a much stronger and more logical signal from user ratings, drastically improving the quality and practical usefulness of the recommendations.

7.5 System Functionality and User Acceptance Testing

Following the validation of the individual AI models, a series of system-level tests were conducted to ensure that the integrated platform functionalities perform as expected from an end-user's perspective. This User Acceptance Testing (UAT) focused on verifying the successful execution of the core user scenarios defined in Chapter 4. The following table documents the test cases, their expected outcomes, the actual results observed, and the final status of each test.

Table 23 Summary of System Functionality and User Acceptance Test

Test Case ID	Test Condition	Expected Results	Actual Results	Status
TC_001	A user attempts to upload a new property using a Property Code that already exists in the database.	The system should prevent the submission and display an error message to the user stating that the asset is a duplicate.	The system prevented the submission and displayed the expected error message.	**Pass**
TC_002	A co-owner initiates a "Group Sale Request" and invites three other co-owners. One of the invited co-owners rejects the request.	The group sale request should be immediately canceled. All involved parties should receive a notification that the request was canceled due to a rejection.	The group sale was canceled as soon as the rejection was received. All users were notified correctly.	**Pass**
TC_003	A buyer initiates a purchase, but one of the sellers fails to enter their correct private key for authorization.	The purchase process should be canceled. No ownership transfer should occur. All parties should be notified of the cancellation due to failed authorization.	The transaction was canceled. A check of the database confirmed that ownership shares were not transferred.	**Pass**

TC_004	A user uploads a clear image of a specific car model to the "Search by Image" feature.	The system should return a list of sales requests for the same or visually similar car models that are currently available on the platform.	The system returned a list of 5 relevant car models, including 3 of the exact same model and 2 similar models from the same brand.	**Pass**
TC_005	An agent attempts to approve a new listing and distribute shares, but the entered percentages for the co-owners do not add up to 100%.	The system should prevent the agent from submitting the share distribution and should display an error message indicating that the total percentage must equal 100%.	The "Submit Decision" button was disabled, and an error message was displayed as expected.	**Pass**
TC_006	A user with a history of viewing and favoriting several SUV listings logs into the platform.	The "Recommended for You" section on the Home Page should feature other SUV models.	The recommendation section displayed three different SUV listings and one pickup truck, indicating a strong relevance to the user's browsing history.	**Pass**
TC_007	A brand-new user registers and logs in for the first time, with no prior browsing or search history.	The recommendation system should handle this "cold start" by displaying a list of globally popular or trending items, not personalized ones.	The "Recommended for You" section displayed a list of the most viewed properties on the platform from the last 7 days, as expected.	**Pass**

Conclusion and Future Works

8.1 Conclusion

This project set out to develop "TrustAsset," an advanced digital marketplace for the real estate and automotive sectors. The primary objective was to create a comprehensive platform that not only digitizes the complex processes of buying and selling high-value assets but also enhances the user experience through a suite of intelligent features. The project successfully delivered a robust software platform with a complete, end-to-end user workflow, encompassing unique features such as shared ownership and agent-mediated transactions.

The core software functionalities, including user registration, secure document upload, agent verification, and group sale requests, have been fully designed and specified, creating a foundation for a trustworthy and efficient marketplace.

Furthermore, this software platform is powered by three distinct AI cores, each of which has been successfully developed and validated:

- **Image-Based Vehicle Search:** An accurate, fine-grained vehicle classification model was developed using transfer learning and fine-tuning with the Xception architecture, providing a strong technical basis for a visual search feature.
- **Price Prediction:** High-accuracy price prediction models were built for both vehicles and real estate using modern gradient boosting frameworks. The models were validated with high R^2 scores on test data, confirming their ability to provide reliable market valuations.
- **Recommendation System:** Two distinct recommendation engines were developed. For real estate, a content-based recommender using an Autoencoder was successfully implemented. For vehicles, a hybrid system was created that uses collaborative filtering for known items and a content-based approach to solve the "cold start" problem for new items. A key outcome was successfully overcoming the data sparsity challenge through methodical data filtering.

In conclusion, the "TrustAsset" project has successfully achieved its goals, delivering a well-defined platform architecture and a suite of validated, high-performance AI models ready for backend integration. The project lays a comprehensive groundwork for a next-generation marketplace that prioritizes trust, intelligence, and user empowerment.

8.2 Future Works

The current "TrustAsset" platform provides a strong and feature-rich foundation. The following are three key areas for future development that would further enhance its capabilities and solidify its position as a market leader.

8.2.1 AI-Powered Document Verification and OCR

Currently, the verification of user-uploaded documents is a manual process performed by the agent. To significantly improve efficiency and security, a future version of the platform could integrate an AI-powered document verification system. This system would use Optical Character Recognition (OCR) to automatically read and extract key information from uploaded documents, such as property deeds or vehicle registration forms. The extracted data could then be automatically validated against expected formats or even cross-referenced with public records to confirm its authenticity. This would not only reduce the manual workload for agents but also add a powerful layer of automated fraud detection to the platform.

8.2.2 Agent Performance and Reliability Scoring

The agent is a central figure in the "TrustAsset" ecosystem, acting as a trusted intermediary. To further build on this model, a transparent agent performance and reliability scoring system could be implemented. The platform could track objective metrics for each agent, such as the number of successful sales, the average time taken to sell a property, and aggregated ratings from clients they have worked with. This "Agent Score" would be displayed on their profile, empowering sellers with the data they need to choose the most effective and reliable agent to represent them.

8.2.3 Side-by-Side Comparison Tool

To improve the decision-making process for buyers, a user-friendly comparison tool could be added to the platform's interface. This feature would allow a user to select two or three different listings (e.g., three different car models or properties) and view their attributes in a clean, side-by-side table. The comparison would highlight key metrics such as price, size/mileage, year, and other core features, making it easy for users to evaluate their options and identify the best choice for their needs. This is a highly practical feature that directly addresses a core part of the buyer's journey.

Appendix

Appendices are provided to give supplementary information, which, if included in the main text, may cause a distraction and cloud the central theme.

Appendix 1: Database Stored Procedure Specification

This section contains the T-SQL code for the stored procedure used to handle the complex logic of transferring ownership shares during a purchase. Using a stored procedure improves performance and ensures the atomicity of the transaction by executing all steps within a single database operation. This procedure deactivates the old shares of a sales request and creates new shares for the buyers based on their contribution percentages, ensuring that ownership is correctly and efficiently transferred.

```
 1. USE [db21399] GO
 2. /****** Object: StoredProcedure [dbo].[ReplaceRequestShares] Script Date: 6/15/2025 5:49:40
PM ******/
 3. SET ANSI_NULLS ON GO
 4. SET QUOTED_IDENTIFIER ON GO
 5. ALTER PROCEDURE [dbo].[ReplaceRequestShares] @RequestForSalesID INT,
 6. @Buyers dbo.BuyerInput READONLY AS
 7. BEGIN
 8. SET NOCOUNT ON;
 9.
10.
11. -- 1. Deactivate old Shares UPDATE Shares
12. SET Active = 0
13. WHERE RequestID = @RequestForSalesID;
14.
15.
16. -- 2. Deactivate related OwnerShips UPDATE o
17. SET Active = 0
18. FROM Ownerships o
19. INNER JOIN Shares s ON s.ID = o.ShareID
20. WHERE s.RequestID = @RequestForSalesID;
21.
22.
23. -- 3. Get total old percentage
24. DECLARE @TotalOldPercentage FLOAT = (
25. SELECT ISNULL(SUM(SharePercentageOfWholePropert), 0) FROM Shares
26. WHERE RequestID = @RequestForSalesID
27. );
28.
29.
30. -- 4. Get PropertyID or CarID
31. DECLARE @CarID NVARCHAR(450);
32. DECLARE @PropertyID INT;
33.
34.
35. SELECT TOP 1
36. @CarID = CarID, @PropertyID = PropertyID
37. FROM Shares
38. WHERE RequestID = @RequestForSalesID;
39.
40.
41. -- 5. Insert new Shares and OwnerShips for each buyer DECLARE @CustomerID NVARCHAR(450);
42. DECLARE @Contribution FLOAT;
43.
44.
45. DECLARE buyer_cursor CURSOR FOR
```

```
46. SELECT CustomerID, ContributionPercentage FROM @Buyers;
47. OPEN buyer_cursor;
48. FETCH NEXT FROM buyer_cursor INTO @CustomerID, @Contribution;
49.
50.
51. WHILE @@FETCH_STATUS = 0 BEGIN
52. DECLARE @NewPercentage FLOAT = @TotalOldPercentage * @Contribution;
53.
54.
55. INSERT INTO Shares (CarID, PropertyID, SharePercentageOfWholePropert, RequestID, Active)
56. VALUES (@CarID, @PropertyID, @NewPercentage, @RequestForSalesID, 1);
57.
58.
59. DECLARE @NewShareID INT = SCOPE_IDENTITY();
60.
61.
62. INSERT INTO Ownerships(OwnerID, ShareID, Active) VALUES (@CustomerID, @NewShareID, 1);
63.
64. FETCH NEXT FROM buyer_cursor INTO @CustomerID, @Contribution; END
65.
66. CLOSE buyer_cursor;
67. DEALLOCATE buyer_cursor; END;
```

Appendix 2: API Logic Specification

This section contains the final, memory-optimized Python code for the `Recommender` class, which powers the hybrid vehicle recommendation system API. This class encapsulates all the logic for both the collaborative filtering and content-based recommendation models. It is designed to be memory-efficient for deployment on servers with limited resources by loading data with optimized types and using Python's garbage collector.

Recommender Class (`recommender.py`)

```python
# recommender.py

import pickle
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity
import gc

class Recommender:
    """
    The main class for the recommendation engine, optimized for low RAM usage.
    """
    def __init__(self, artifacts_path, inventory_path):
        print("Recommender: Initializing...")
        self._load_collab_artifacts(artifacts_path)
        self._prepare_content_engine(inventory_path)
        print("Recommender: Initialization complete.")

    def _load_collab_artifacts(self, artifacts_path):
        """Loads the collaborative filtering model and its components."""
        print("Recommender: Loading collaborative filtering model...")
        with open(artifacts_path, 'rb') as f:
            collab_artifacts = pickle.load(f)
        self.collab_model = collab_artifacts['model']
        self.item_user_matrix = collab_artifacts['item_user_matrix']
        self.item_mapper = collab_artifacts['item_mapper']
        self.item_inv_mapper = collab_artifacts['item_inv_mapper']
```

```
        print("Recommender: Collaborative model loaded.")

    def _prepare_content_engine(self, inventory_path):
        """
        Pre-processes inventory data for content-based filtering in a memory-efficient way.
        """
        print("Recommender: Preparing content-based engine...")
        try:
            # --- RAM OPTIMIZATION 1: Load data with efficient types ---
            dtype_spec = {
                'YearOfMaking': 'int16',
                'Price': 'float32',
                'Horsepower': 'int16',
                'Make': 'category'
            }
            # Load only the columns we absolutely need for the content engine + item_id creation
            cols_to_load = ['YearOfMaking', 'Price', 'Horsepower', 'Make', 'Model', 'Trim']
            inventory_df = pd.read_csv(inventory_path, usecols=cols_to_load, dtype=dtype_spec)

        except FileNotFoundError:
            print(f"Error: Inventory file not found at {inventory_path}")
            self.content_matrix = None
            self.inventory_item_ids = None
            return

        # Define features for the model
        self.content_features = ['YearOfMaking', 'Price', 'Horsepower', 'Make']

        # Calculate defaults for optional fields
        self.default_values = {
            'Price': inventory_df['Price'].median(),
            'Horsepower': inventory_df['Horsepower'].median()
        }

        content_df = inventory_df[self.content_features].copy()

        # Scale numerical features
        self.scaler = MinMaxScaler()
        numerical_features = ['YearOfMaking', 'Price', 'Horsepower']
        content_df[numerical_features] =
self.scaler.fit_transform(content_df[numerical_features])

        # One-Hot Encode the categorical feature
        encoded_features = pd.get_dummies(content_df[['Make']], prefix=['Make'], sparse=True)

        # Create the final matrix
        self.content_matrix = pd.concat([content_df[numerical_features], encoded_features],
axis=1)

        # --- RAM OPTIMIZATION 2: Store only the final item_ids ---
        # Create the item_ids and store them in a simple pandas Series.
        self.inventory_item_ids = (
            inventory_df['YearOfMaking'].astype(str) + '_' +
            inventory_df['Make'].astype(str).str.lower() + '_' + # .astype(str) because
'category' type
            inventory_df['Model'].str.lower().str.replace(' ', '_') + '_' +
            inventory_df['Trim'].str.lower().str.replace(' ', '_')
        )

        # --- RAM OPTIMIZATION 3: Discard the large DataFrames ---
        # We no longer need the full inventory or content dataframes in memory.
        del inventory_df
        del content_df
        del encoded_features
        gc.collect() # Ask Python's garbage collector to free up the memory now.

        print("Recommender: Content-based engine ready. Memory optimized.")

    def get_collaborative_recommendations(self, item_id, k=10):
        # This function remains unchanged
```

```
        if item_id not in self.item_mapper:
            return None
        item_index = self.item_mapper[item_id]
        distances, indices = self.collab_model.kneighbors(self.item_user_matrix[item_index],
n_neighbors=k + 1)
        neighbor_indices = indices[0]
        recommendations = [self.item_inv_mapper[i] for i in neighbor_indices if i != item_index]
        return recommendations

    def get_content_based_recommendations(self, car_features, k=10):
        if self.content_matrix is None:
            return "Error: Content-based engine not loaded."

        for key, value in self.default_values.items():
            car_features.setdefault(key, value)

        input_df = pd.DataFrame([car_features])
        input_df['Make'] = input_df['Make'].astype('category')

        numerical_features = ['YearOfMaking', 'Price', 'Horsepower']
        input_df[numerical_features] = self.scaler.transform(input_df[numerical_features])
        input_vector_df = pd.get_dummies(input_df)
        input_vector = input_vector_df.reindex(columns=self.content_matrix.columns, fill_value=0)

        sim_scores = cosine_similarity(input_vector, self.content_matrix)

        top_indices = sim_scores[0].argsort()[-k-1:-1][::-1]

        # Use the stored inventory_item_ids Series instead of the full DataFrame
        return self.inventory_item_ids.iloc[top_indices].tolist()
```

References

[1] K. Valev, A. Schumann, L. Sommer and J. Beyerer, "A Systematic Evaluation of Recent Deep Learning Architectures for Fine-Grained Vehicle Classification," in *Pattern Recognition and Tracking XXIX* , 2018.

[2] A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv, 2014.

[3] L. Yang, P. Luo, C. C. Loy and X. Tang, "A Large-Scale Car Dataset for Fine-Grained Categorization and Verification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015.

[4] A. Chandak, P. Ganorkar, S. Sharma, A. Bagmar and S. Tiwari, "Car Price Prediction Using Machine Learning," IJCSE , Indore, India , 2019.

[5] L. Rampini and F. R. Cecconi, "Artificial Intelligence Algorithms to Predict Italian Real Estate Market Prices," Journal of Property Investment & Finance , Milan, Italy , 2021 .

[6] L. H. Choy and W. K. Ho, "The Use of Machine Learning in Real Estate Research," Land (MDPI) , Basel, Switzerland (MDPI's location) , 2023 .

[7] A. Taiwo, L. Ogundele, F. Ayo and A. Ejidokun, "Car Price Prediction Using Artificial Neural Networks: A Data-Driven Approach," Indonesian Journal on Computing (Indo-JC) , Bandung, Indonesia , 2024.

[8] X. Ju, V. Chakma, M. Amin and J. A. Chakma, "What Drives House Prices? A Linear Regression Approach to Size, Condition, and Features," Journal of Artificial Intelligence and Data Mining , 2025.

[9] E. G. Muñoz, J. Meza and S. Ventura, "Fuzzy Logic Recommender Model for Housing," *IEEE Access,* pp. 11380 - 11395, 2025.

[10] V. C, H. Oberoi, A. Goyal and N. Sikka, "RE-RecSys: An End-to-End system for recommending properties in Real-Estate domain," in *CODS-COMAD '24: Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD)*, Bangalore, India, 2024.

[11] Y. M. Arif, M. F. Muhtarom and H. Nurhayati, "Performance of Known Ratings-Based Multi-Criteria Recommender System for Housing Selection," in *2023 International*

Conference on Computer Science, Information Technology and Engineering (ICCoSITE), Jakarta, Indonesia, 2023.

[12] P. Boteju and L. Munasinghe, "Vehicle Recommendation System using Hybrid Recommender Algorithm and Natural Language Processing Approach," in *2020 2nd International Conference on Advancements in Computing (ICAC)*, Malabe, Sri Lanka, 2020.

[13] Q. Zhang, D. Zhang, J. Lu, G. Zhang, W. Qu and M. Cohen, "A Recommender System for Cold-start Items: A Case Study in the Real Estate Industry," in *2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Dalian, China, 2019.

ملخص

"TrustAsset" هي منصة رقمية متقدمة تهدف إلى تسهيل وتأمين عمليات بيع وشراء العقارات والمركبات. تم تصميم المنصة لمواجهة تعقيدات المعاملات ذات القيمة العالية من خلال الجمع بين إطار برمجي قوي ومجموعة من الميزات الذكية.

تنقسم ميزات المنصة إلى ثلاثة محاور رئيسية:

١. الميزات الأساسية والمعاملات الآمنة

يركز هذا المحور على بناء الثقة وضمان سير العمليات بسلاسة وأمان:

- **العروض المدارة عبر الوكلاء:** لا يقوم البائعون بنشر العروض مباشرة، بل يقدمون طلباً لوكيل معتمد ومسجل في المنصة، والذي يتولى بعد ذلك عملية التحقق من المعلومات ونشر العرض، مما يضيف طبقة من الموثوقية.
- **منع تكرار الأصول:** يمنع النظام تلقائياً إدراج نفس العقار أو المركبة مرتين عن طريق التحقق من المعرّفات الفريدة (مثل رمز العقار أو رقم هيكل المركبة).
- **عملية شراء آمنة:** تتم عملية الشراء عبر خطوات متعددة تتطلب من كل من المشتري والبائع (أو البائعين) تأكيد المعاملة عن طريق إدخال "مفتاح خاص" فريد، مما يضمن أعلى مستويات الأمان.
- **نقل الملكية التلقائي:** عند إتمام عملية شراء ناجحة، يقوم النظام تلقائياً بإلغاء تفعيل حصص الملكية للبائعين وإنشاء حصص جديدة للمشترين، وتتم هذه العملية بشكل آمن ومتكامل داخل قاعدة البيانات.
- **إنشاء العقود الإلكترونية:** في نهاية كل عملية شراء ناجحة، يتم إنشاء عقد إلكتروني موقّع رقمياً من قبل جميع الأطراف باستخدام مفاتيحهم الخاصة.

٢. ميزات الملكية المشتركة والمبيعات

يعتبر هذا الجانب من أكثر الجوانب تميزاً في المنصة:

- **إدارة الملكية الجزئية:** تم تصميم النظام للتعامل مع الأصول التي يمتلكها عدة أشخاص، حيث يمتلك كل مالك نسبة مئوية محددة من الحصص.
- **بيع الحصص الفردية:** يتمتع المالك الشريك بالمرونة لإنشاء طلب بيع لحصته الفردية فقط من الأصل دون الحاجة لموافقة بقية الشركاء.
- **نظام طلبات البيع الجماعي:** يمكن لأحد المالكين الشركاء بدء عملية بيع للأصل بأكمله عن طريق دعوة المالكين الآخرين. لا تتم عملية البيع إلا بموافقة **جميع** المالكين المدعوين خلال فترة زمنية محددة (5 دقائق).

٣. الميزات المدعومة بالذكاء الاصطناعي

هذه هي المحركات الذكية التي تعمل على تحسين تجربة المستخدم:

- **البحث عن المركبات بالصور:** يمكن للمستخدمين تحميل صورة لسيارة، وسيقوم النظام بالعثور على المركبات المماثلة المعروضة للبيع وعرضها.
- **التنبؤ بالأسعار:** يمكن للمستخدمين الحصول على تقدير لسعر السوق لأي عقار أو سيارة بناءً على تفاصيلها المحددة، مما يساعدهم على اتخاذ قرارات مستنيرة.
- **نظام التوصيات المخصص:** تتعلم المنصة من سلوك المستخدم (مثل سجل البحث، والمفضلات، والوقت الذي يقضيه في تصفح كل عرض) لتقترح عليه بشكل استباقي عقارات وسيارات أخرى قد تكون ذات أهمية له.

الجامعة العربية الدولية

الجامعة العربية الدولية AIU جامعة سورية خاصة أُحدثت عام 2005، خططها الدراسية والوثائق الصادرة عنها معتمدة ومصدقة من قبل وزارة التعليم العالي في الجمهورية العربية السورية.
تعمل الجامعة على تحقيق الأهداف الآتية:

- إعداد جيل متميز من الخريجين الجامعيين القادرين على تلبية الحاجات النوعية للمجتمع والنهوض به.
- الإسهام في البحوث العلمية النظرية والتطبيقية التي تخدم أغراض التنمية الوطنية، ويتم العمل على حث الأساتذة والعاملين الأكاديميين على البحث العلمي والمشاركة في المؤتمرات والندوات التي تنظم الأبحاث.
- تحقيق الشراكة مع الجامعات العربية والأجنبية المرموقة بهدف التطوير والتحديث المستمرين للعمل الأكاديمي والقيام ببحوث علمية مشتركة.
- استقطاب الكفاءات الأكاديمية والبحثية المتميزة عن طريق توفير البيئة المناسبة لعملها.

الجامعة العربية الدولية من الجامعات السورية الأولى التي جرى تأسيسها وافتتاحها، وقد تمكنت من اجتذاب الكفاءات التعليمية والبحثية والإدارية المتميزة، لإنشاء صرح متكامل من النواحي الأكاديمية والتنظيمية والإدارية. وتمكنت من تخريج كوادر من المبدعين والمتميزين من خلال توفير بيئة تعليمية ترتكز إلى مقومات نوعية ومادية فريدة منها:

- الخطط الدراسية الحديثة والمتطورة المستندة إلى نظام الساعات المعتمدة.
- الأطر التعليمية المنتقاة بعناية كبيرة.
- المختبرات العلمية الحديثة، ومختبر للمكتبات الإلكترونية.
- المحفزات المادية والمعنوية للطلبة.
- تطبيق طرائق التدريس التفاعلي.
- التوجيه والإرشاد الأكاديمي والتربوي.
- مجموعة كبيرة من اتفاقيات التعاون العلمي مع جامعات محلية وإقليمية ودولية ذات سمعة مرموقة.
- اتفاقيات ومذكرات تفاهم متعددة مع العديد من مؤسسات المجتمع المدني.
- الحرم الجامعي اللائق والمزود بكافة المرافق العلمية والرياضية والترفيهية، والذي نشجعك على زيارته والتعرف على مزاياه.
- الأنشطة والأندية الطلابية بمختلف أنواعها: الرياضية والثقافية والعلمية والاجتماعية.

في الجامعة العربية الدولية سنوات الحياة الجامعية هي وقت للاستثمار في مستقبل الطالب. فالمعارف والخبرات التي يحصلها في قاعة المحاضرات والمختبرات ستساعده في تطوير ذاته، وستمنحه أسباب النجاح في التخصص الذي اختاره، والنشاط الطلابي الذي يمارسه سيساعده في توسيع أفقه، وفعاليات التدريب والأندية والرياضة ستمكنه من تطوير مواهبه، ولربما تساعده في اكتشاف مواهب جديدة.
ليستثمر وقته وذهنه وروحه في جامعتنا كي يجني فوائد عمله والوقت الذي كرسه في السنين القادمة. ونحن سوف نكون بجانب طلبتنا في كل خطوة على دربهم.

الجامعة العربية الدولية

كلية الهندسة المعلوماتية والاتصالات

مشروع التخرج

نظام الأصول الموثوقة لشراء وبيع العقارات والمركبات

تم تقديمه إلى

قسم الهندسة المعلوماتية

تقديم

شاكر الكاني **محمد أحمد** **محمد زيد النحاس**

أسماء المغربي **رزان الشعار** **داليا مهايني**

بإشراف

الدكتور محمد محمد **المهندسة رغد عرب**

حزيران 2025