Palestine Polytechnic University

College of Information Technology and Computer Engineering

Project Title:

Machine-Learning-Based Land-Price Prediction System

Team Members:

Mohammad Alqadi | Mohammad Alamlah

Supervisor: Dr. Hashem Altamimi

إهداء

إلى والدينا

بفصاحة القلب وبكل احترام وتقدير، نتوجه إليكم برسالة ممتلئة بعمق المشاعر وارتفاع الجلال. إن ما نحمله في قلوبنا من امتنان ومودة لا يمكن وصفه بكلمات بسيطة، فأنتما الركيزة الثابتة التي بنينا عليها حياتنا، والشمعة الساطعة التي أضاءت دربنا في ظلمة الليالي..

...

إلى أصدقائنا

بكل احترام وتقدير، نرفع لكم تحية الود والاعتزاز، فأنتم أصدقاؤنا الأوفياء والرفاق المخلصون. لقد كنتم دائماً العون والسند في السراء والضراء، والصخرة الصلبة التي نستند إليها في عبور مياه الحياة العميقة. فشكرًا لكم على كل لحظة قضيناها معًا، وعلى كل دعمكم اللامحدود وتضحياتكم الجليلة.

شكر وتقدير

إلى أساتذتنا الكرام، نتقدم بأخلص الشكر والتقدير على الجهود الجبارة التي بذلتموها خلال سنواتنا في الدراسة. لقد كنتم قدوةً ومصدرَ إلهامٍ لنا، وساهمتم بشكل كبير في تشكيل مستقبلنا الأكاديمي والمهني.

نود أن نخصّ بالشكر **الدكتور هاشم هشام التميمي** على تفانيه وإرشاده القيم، وعلى كل العلم والمعرفة التي شاركنا بها. لقد كنتم داعمًا لنا في كل خطوة نخطوها في طريقنا التعليمي.

نشكركم على صبركم الذي لا يُضاهى واحتوائكم لنا في كل الظروف، وعلى توفير بيئة تعليمية محفِّزة ومليئة بالتشجيع. إن مساهماتكم **لن تُنسى**، وستظل خالدة في ذاكرتنا.

ونخصّ بالشكر **المخمّن العقاري قيس ادعيس** على تزويدنا ببيانات ميدانية واقعية وإرشادات مهنية أسهمت مباشرةً في بناء قاعدة البيانات واختبار النموذج.

ونتوجه بالشكر أيضًا إلى **جامعة بوليتكنك فلسطين** على توفير المرافق التعليمية المتميزة والخدمات التي ساعدتنا على تحقيق أهدافنا الأكاديمية بنجاح.

ندعو الله أن يجزيكم خير الجزاء وأن يوفقكم في كل ما تسعون إليه من خير وتطور في خدمة العلم والتعليم.

Abstract

In the era of artificial intelligence and technological advancements, the process of predicting (or estimating) land prices is still implemented using traditional methods that rely on human estimation, which makes it prone to bias and inconsistency in results. In response to these challenges, this project aims to develop an intelligent system that depends on machine learning and real-world data to estimate land prices more objectively and more accurately, and in less time compared to traditional methods. The town of Bani Na'im, located in the Hebron Governorate in Palestine, was chosen as an experimental area to apply the system because there is enough available data about its lands, and local land appraisers cooperated by providing us with this data.

The system is designed with an interactive user interface that provides a form with fields to enter land features such as area, location, political classification, and other influencing factors, to provide an immediate estimated price for the user. The regression tree algorithm was chosen for the project in its early stage due to its simplicity and efficiency in dealing with a limited amount of data, which is the case with the data currently available. The used data included both numerical and categorical features, the model was trained on this data to estimate the price based on the entered factors. The data was collected from various sources, the most important being the land appraisers from Bani Na'im as well as referring to official maps and structural plans to extract important information about the lands, such as their location, classification, shape, and price, so they can be manually entered into the system. The diverse sources helped build a realistic database, and reinforced the authenticity of the model and its relevance to the practical field. Although the available data is limited, the model was optimized to achieve a balance between precision and speed, which makes it an effective helping tool for the decisions of real estate appraisers, since they are the main users who benefit from it. This project represents the first step in automating the process of real estate valuation, and it is planned to develop it in the future using more advanced algorithms such as CatBoost and Random Forest, to keep up with the increasing volume and variety of available data.

الخلاصة

في عصر نهضة الذكاء الاصطناعي والتطور الملحوظ لا تزال عملية تخمين (أو تثمين) أسعار الأراضي تُنفّذ بأساليب تقليدية تعتمد على التقدير البشري؛ فهذا يجعلها عرضة للتحيّز والتفاوت في النتائج. استجابةً لهذه التحديات، يهدف هذا المشروع إلى تطوير نظام ذكي يعتمد على تعلم الآلة لتقدير أسعار الأراضي بموضوعية ودقة أعلى، وفي زمن أقل مقارنةً بالأساليب التقليدية، وذلك بالاعتماد على بيانات واقعية تم جمعها. وقد تم اختيار بلدة بني نعيم الواقعة في محافظة الخليل، فلسطين، كنموذج أولي لتطبيق النظام. نظراً لتوفر بيانات كافية حول أراضيها، وتعاون مخمني الأراضي من خلال تزويدنا بها. تم تصميم النظام بواجهة مستخدم تفاعلية تتيح إدخال خصائص الأرض مثل المساحة، والموقع، والتصنيف السياسي، وغيرها من العوامل المؤثرة، ليحصل المستخدم على سعر تقديري فوري. اعتمد المشروع في مرحلته الأولى خوارزمية شجرة الانحدار، نظرًا لبساطتها وقدرتها على التعامل بكفاءة مع أحجام بيانات محدودة، كما هو الحال مع البيانات المتوفرة حاليًا. تضمنت البيانات المستخدمة خصائص عددية وأخرى تصنيفية وتم تدريب النموذج عليها لتقدير السعر بناءً على العوامل المدخلة. وقد تم تحصيل هذه البيانات من مصادر متنوعة أهمها مخمنو الأراضي في بلدة بني نعيم، بجانب الرجوع إلى خرائط رسمية ومخططات هيكلية لاستخلاص معلومات تنظيمية عن الأراضي مثل موقعها وتصنيفها وشكلها، وسعرها، وذلك لإدخالها يدويًا إلى النظام. ساعد هذا التنوع في بناء قاعدة بيانات واقعية، وعزّز من موثوقية النموذج وارتباطه بالميدان العملي. ورغم محدودية البيانات المتوفرة، تم ضبط النموذج لتحقيق توازن فعّال بين الدقة وسرعة التنفيذ، مما يجعله أداة مساعدة فعالة لقرارات المخمّنين العقاريين، كونهم الفئة المستفيدة منه بشكل رئيسي. يمثّل هذا المشروع الخطوة الأولى في أتمتة عملية التثمين العقاري، ويُخطط لتطويره لاحقًا باستخدام خوارزميات أكثر تقدمًا مثل CatBoost وRandom Forest، بما يتماشى مع الازدياد في حجم وتنوع البيانات المتوفرة.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

This chapter introduces the main elements of the project. It begins with the idea of the project, then goes to its importance, followed by the goals of the project, the scope and limitations, the theoretical background, and finally the chosen algorithm and alternatives.

1.2 Idea of the Project

The main idea of the project is building a web application for an intelligent system that is capable of accurately predicting the land prices in the town of Bani Na'im. In order for the system to predict accurately, it will depend on the techniques of artificial intelligence (AI) and machine learning (ML). The machine learning model will be trained by providing for it all the major factors affecting land prices, these factors include: area, distance to main roads and markets, availability of water and electricity supplies, among others. By feeding the machine learning algorithm with this data the project aims to provide a faster, more accurate, and more transparent alternative to traditional land valuation methods. The project intends to benefit the appraisers and help them make objective and data driven decisions.

1.3 Importance

Due to the frequent transactions in the area, the need for a faster and more efficient pricing method is growing, and one of the main advantages of this project's AI-powered approach is speed, while traditional/manual methods can take several hours to evaluate the price, the trained machine learning model can do it in seconds. Another need is reducing subjectivity, it is crucial to avoid the human bias in the field of land price evaluation because human bias in land valuation can shift prices by thousands of shekels. The project eliminates such bias by relying on data and algorithms alone, ensuring objective, data-driven, and transparent predictions.

1.4 Goals of the Project

The main goal of the project is to develop a machine learning model capable of accurately estimating land prices in Bani Na'im and to achieve this goal, the project has the following objectives:

1- Data collection: Gather all relevant land data like the area, location, suitability for agriculture and more.

2- Data cleaning: After collecting the raw data, data cleaning is performed, where the data's quality will be enhanced by removing duplicate and irrelevant data entries, and correcting inconsistencies.

3- Model development: Train machine learning algorithms with the cleaned data and compare them to select the most suitable algorithm in predicting the prices.

4- Model evaluation: Test the model and evaluate it by comparing the results of the model with the actual results of traditional pricing methods.

5- Tool implementation: Design a user-friendly website as a tool for the land appraisers.

The project aims to increase the efficiency and transparency of the land price estimations as well as making the process of price estimation easier for the appraisers.

1.5 Scope and Limitations

This project aims to predict land prices in the town of Bani Na'im using machine learning techniques depending on available real Bani Na'im land data. The scope of the project includes developing a predictive machine learning model that predicts land prices depending on features like:

- Location
- Political classification of the land (area A, B, or C)
- Intended land use (e.g., residential, commercial)
- Land area
- Availability of infrastructure
- Proximity to essential public services (e.g., hospitals)

The project also involves designing a user-friendly interface that allows authorized users such as land appraisers, system admins, and data scientists to use the system — everyone as allowed to.

On the other hand, the project faces many challenges that may affect the accuracy of the prediction. The accuracy depends directly on data quality and completeness in addition to the used model. The most important limitations are:

- **Limited data availability:** The collected data may be outdated or incomplete, and some land prices may not be documented.

- **Assumption of data representativeness:** This project assumes that the available data reflects typical land characteristics in Bani Na'im.
- **Geographic limitation:** The model is specifically designed for lands just in Bani Na'im.
- **Not considering all external factors:** The model does not account for sudden market shifts, and in Palestine, Palestinians are vulnerable to forced displacement at any moment, which could cause a sudden gap in land prices.
- **Limited time:** Because of the limited time that we have, our team was not able to try many machine learning algorithms to choose the best one that validated our project.

1.6 Background

1.6.1 Artificial Intelligence (AI)

Artificial Intelligence (AI) is one of the most significant fields in modern computer science. It aims to develop intelligent systems capable of performing tasks that traditionally require human intelligence, such as reasoning, decision-making, pattern recognition, and prediction. AI systems rely on processing large datasets, extracting meaningful relationships, and generating insights that enable faster, more accurate, and more objective decisions compared to traditional manual approaches. In recent years, AI applications have expanded across numerous domains, including the real estate sector, where AI contributes to producing reliable, data-driven land and property price estimations.

1.6.2 Machine Learning (ML)

Machine Learning (ML) is a core discipline within AI that focuses on constructing models capable of learning automatically from data rather than being explicitly programmed for every possible case. ML models analyze historical data, discover patterns and relationships between input features and target variables, and utilize this knowledge to make predictions on new, unseen data.

Machine learning approaches are commonly categorized into three main types:

- **Supervised Learning:** The model is trained using labeled data that includes both input features and their corresponding correct outputs. This enables the model to learn the mapping between inputs and outputs.
- **Unsupervised Learning:** The model learns from unlabeled data, aiming to uncover hidden structures, clusters, or patterns within the dataset.

- **Reinforcement Learning:** The model learns by interacting with an environment, receiving feedback in the form of rewards or penalties, and improving its performance over time.

Since the objective of this project is to predict a continuous numerical value representing land price, the most suitable approach is supervised regression learning.

1.6.3 Regression in Machine Learning

Regression techniques are used when the target variable is continuous rather than categorical. In this project, regression is employed to estimate land prices based on multiple influential features, including:

- Geographic location
- Land area
- Administrative and political classification
- Availability of services and infrastructure
- Proximity to main roads and essential facilities

Using regression enables objective, consistent, and data-driven valuation while reducing reliance on subjective human estimation, which may vary among assessors.

1.6.4 Regression Tree

A Regression Tree is one of the widely used supervised learning algorithms for predicting continuous values such as real estate and land prices. A regression tree consists of internal nodes, branches, and terminal leaf nodes.

The dataset is recursively divided into increasingly homogeneous subsets by selecting the most informative feature and an appropriate splitting threshold at each node. This process continues until specific stopping criteria are satisfied, such as:

- Reaching a maximum tree depth
- Reaching a minimum number of samples in a node
- Achieving an acceptable prediction error

Ultimately, each leaf node represents a group of lands sharing similar characteristics, and a corresponding price estimation is assigned to that group.

1.6.5 Overfitting and Pruning

Regression trees may suffer from overfitting when the tree becomes too complex, learns noise from the training data, and performs poorly on new data. To address this limitation, pruning techniques are applied.

Pre-pruning restricts tree growth through constraints such as limiting maximum depth or requiring a minimum number of samples per node. Post-pruning builds a full tree and then removes branches that do not contribute significantly to prediction performance. These methods enhance the model's generalization capability and improve the stability of predictions.

1.6.6 Reasons for Choosing Decision Tree Regression

The Regression Tree algorithm was selected for this project because it provides interpretable decision-making, supports both numerical and categorical data, performs effectively with small to medium-sized datasets, captures non-linear relationships, and matches the multi-factor nature of land price estimation. Therefore, it represents a scientifically justified and practically efficient choice for predicting land prices in Bani Na'im. In addition, and from a computational perspective, regression trees are efficient to train and evaluate, making them suitable for deployment within a web-based system. Their relatively low computational cost ensures fast response times during user interactions, which enhances system usability and scalability.

1.7 Mathematical Background

The Regression Tree model implemented in this project is mathematically supported by several fundamental principles governing node splitting, prediction generation, and model complexity control.

1.7.1 Sum of Squared Residuals (SSR)

At each node, the quality of the grouping is evaluated using the Sum of Squared Residuals (SSR), which measures how close the values are to their mean:

$$SSR = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where y_i is the actual price of sample i, $\bar{y}$ is the mean price of samples in the node, and n is the number of samples in the node. A lower SSR indicates better homogeneity and therefore a better-quality node.

1.7.2 Best Split Criterion

For each potential split, SSR is computed for the left and right subsets. The total resulting error is:

$$SSR_{total} = SSR_{left} + SSR_{right}$$

The optimal split is the one that minimizes SSR_{total}, ensuring that the resulting subsets are more homogeneous and stable.

1.7.3 Leaf Node Prediction

After the splitting process terminates, each leaf node represents a set of similar samples. The predicted value assigned to a leaf node is the mean value of all samples within it:

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Any new instance that reaches this leaf will be assigned this value as its predicted land price.

1.7.4 Model Complexity Control (Cost Complexity Pruning)

To avoid overfitting, a penalty term is introduced to balance accuracy and structural complexity:

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $R(T)$ is the prediction error of the tree, $|T|$ is the number of terminal nodes, and α is a regularization parameter controlling complexity. Increasing α reduces tree size and enhances generalization capability.

1.8 Alternatives

There are several alternative methods for evaluating the land prices, but the most common method used in Bani Na'im is **comparative market analysis (CMA)**, which is comparing the land to be evaluated to similar lands that were recently sold. These lands have shared attributes to be compared.

Although this method is commonly used, it is less accurate and less effective than the machine learning method, and also more complicated to justify the result because the CMA method relies heavily on the subjective judgment of experts rather than objective land statistics, which can introduce bias and inconsistency.

In contrast, this project leverages machine learning models, which can automatically learn from data and adapt to changing market and political conditions to provide faster, more accurate predictions and justifiable, transparent results.

Chapter 2: Requirement Specifications

2.1 Overview

This chapter identifies the main users of the land pricing system and describes the role of each. It also outlines the functional and non-functional requirements that define how the system should behave, and shows how the components of the system interact with each other. It also presents visual representations such as a use-case diagram and a context diagram as well as functional requirements tables.

2.2 Actors

The system has three main actors, each with distinct responsibilities:

1- **Land Appraiser** — Enters target-land characteristics and receives an automated price prediction. Uses the result to validate their own estimate or as a data-backed estimation.

2- **Admin** — Manages user accounts and system configuration (view roles/emails, activate/deactivate, update or remove users). Maintains a safe, secure, and smooth operation of the platform.

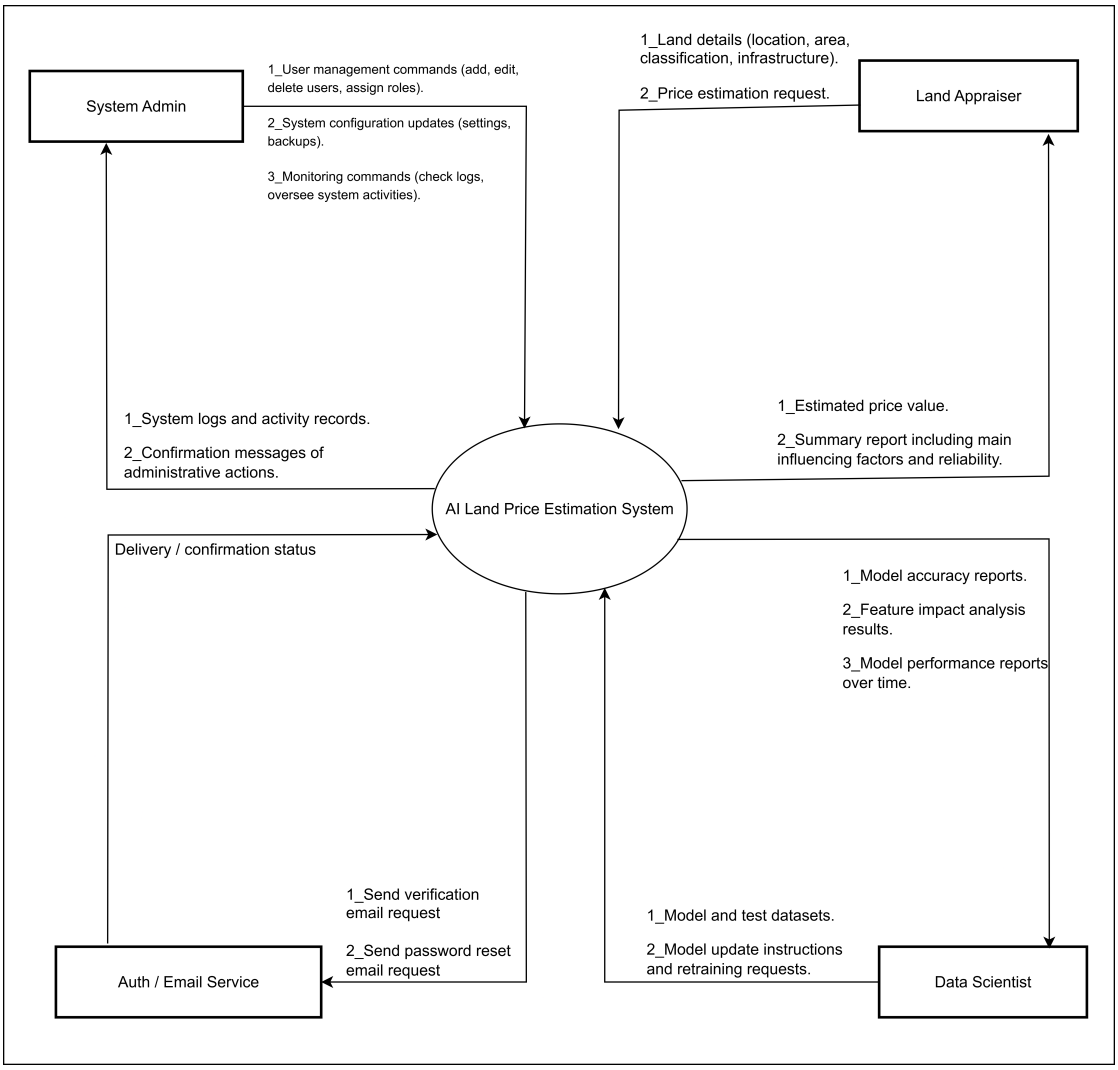
3- **Data Scientist** — Ensures model and platform quality. Prepares/curates datasets, tests and validates the ML model with real or synthetic data, monitors accuracy and performance, and suggests improvements.

Together, these actors keep the system reliable and continuously improving.

2.3 Context Diagram

Figure 2.3.1 illustrates the context diagram of the AI Land Price Estimation System, showing the main external entities and their interactions with the system. The key entities are the System Admin, Land Appraiser, Data Scientist, and Authentication/Email Service. Each entity communicates with the system through specific commands, data inputs, or reports, ensuring the overall functionality of user management, model development, account security, and land price estimation.

Figure 2.3.1 Context Diagram



2.4 Functional Requirements

2.4.1 Land Appraiser's Side

1. **User Registration and Login**
    - Appraisers must be able to register using a valid email address and password.
    - An activation code provided by the administrator is required to complete registration.
    - Once registered, appraisers can log in securely using their email and password.
    - A password reset option must be available in case appraisers forget their password.
2. **Profile Management**
    - View and edit personal information (e.g., name, email).
    - Change password from profile settings.
3. **Add a New Project**
    - Create a new project.
    - Input land details for estimation.
4. **Selecting an Old Project**
    - Select a previously created project.
    - Edit the input data and re-estimate the price.
5. **Price Estimation**
    - The system processes the entered data and displays the estimated land price.
    - The appraiser receives a summary of the estimation and the influencing factors.
6. **Project History**
    - The system saves each submitted land estimation as a separate project.
    - The appraiser can view a list of all past projects.
    - Each project shows input details, results, and the date of submission.
7. **Edit or Delete Land Inputs (Before Submission)**
    - Edit or clear the form data before submitting for estimation.

8. **Input Validation**
    - The system checks for missing or invalid entries and shows helpful error messages.
9. **Rating the Estimation Result**
    - The appraiser can rate the estimation result after it is displayed.

2.4.2 Admin's Side

1. **Login** — The admin can securely log in to the system using their credentials.
2. **Manage Users**
    - View all registered users.
    - Remove user accounts.
    - Edit user roles.
    - Activate / Deactivate accounts.
3. **Creating Admin Accounts** — Create new admin accounts when necessary to expand system management.
4. **Manage Form Data** — Manage selectable regions and update system data fields relevant to land evaluation to keep the platform consistent with current geographic and regulatory information.
5. **View System Logs** — See records of user activity and system events to monitor and diagnose issues.
6. **Manage Backups** — Save backups of system data and restore them in case of data loss or system problems.

2.4.3 Data Scientist's Side

1. **User Registration and Login**
    - Register using a valid email address and password.
    - Provide an activation code issued by the administrator to complete registration.
    - Log in securely using credentials once registered.
    - Reset password when needed.
2. **Test Model Accuracy** — Run tests using known or sample land data to evaluate model accuracy.

3. **Review Feature Impact** — View which land features (e.g., area, location) most influence the predicted price based on the model's analysis.
4. **Monitor Model Performance Over Time** — Track model performance across time and compare older versions with newer ones.
5. **Select Any Project to Analyze** — Select any existing project in the system (including those created by any land appraiser) for analysis.

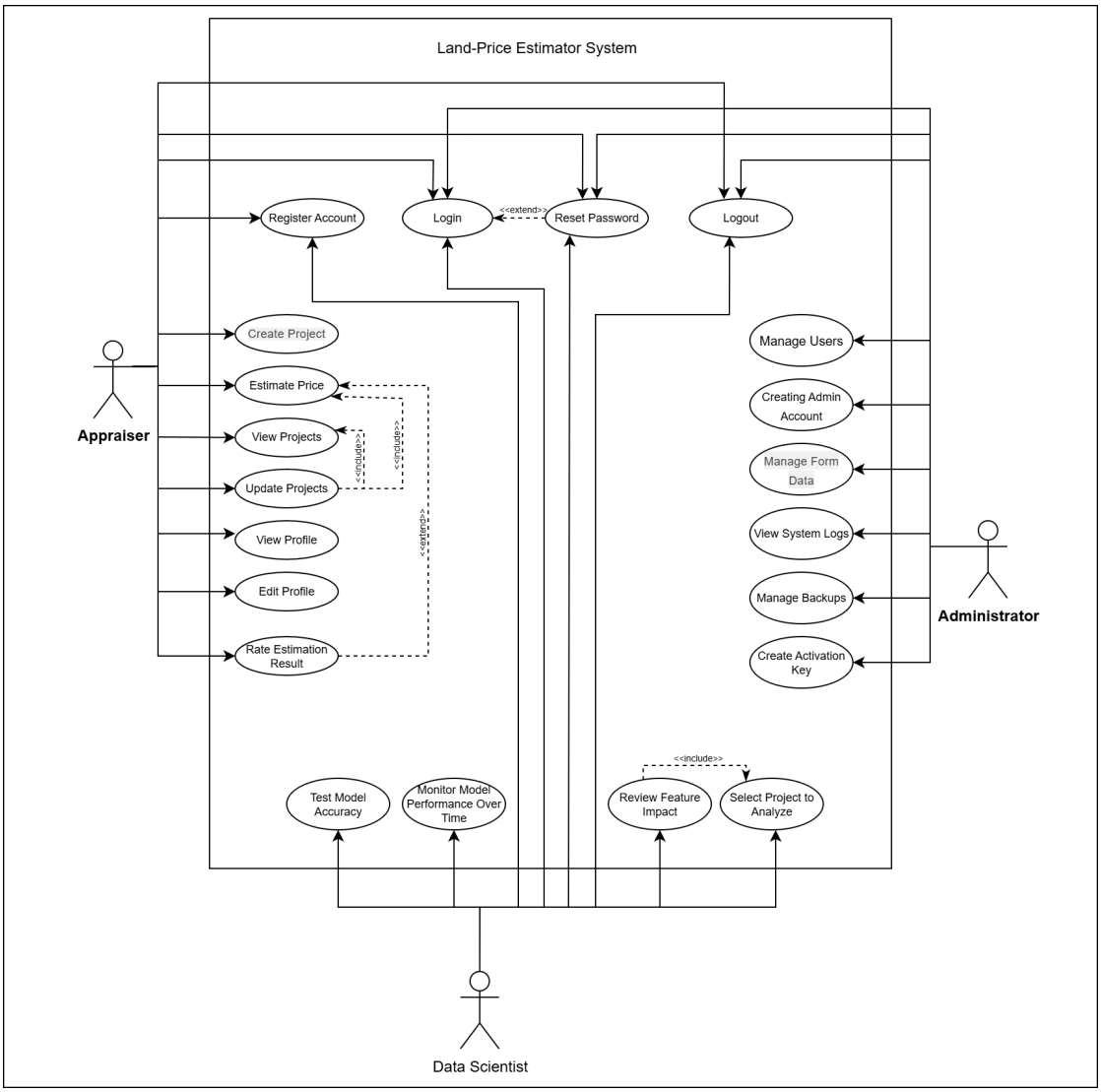
2.5 Nonfunctional Requirements

The nonfunctional requirements describe how the system should behave to provide the best user experience.

1. **Usability**
   - The system should provide a simple and user-friendly interface.
   - The interface should support both desktop and mobile browsers.
2. **Performance**
   - The system should return land price estimation results in less than 5 seconds after submission.
   - Login and registration should complete in less than 3 seconds under normal load.
3. **Availability**
   - The system should be available at least 99% of the time.
4. **Security**
   - The system must protect user information by applying strong encryption methods.
   - Passwords should be securely hashed.
   - Only authorized users can access their personal projects and information.
5. **Data Backup and Recovery**
   - All user accounts and project details should be backed up regularly.
   - When a system failure occurs, users should be able to recover their information without data loss.

2.6 Use-Case Diagram

Figure 2.5.1 Use Case Diagram



2.7 Appraiser's Functional Requirements Tables

Table 2.7.1 Login

Field	**Content**
Requirement	Login
Actor	Land Appraiser
Objective	Access the appraiser's account
Precondition	The appraiser must be registered.
Scenario	1. The appraiser enters email and password. 2. The appraiser clicks 'Submit'. 3. The system verifies credentials and grants access.
Exceptions	1. Incorrect credentials — the system displays an error message. 2. Account locked due to failed attempts. 3. Account not activated. 4. No internet connection. 5. Server or network error — system prompts the user to try again later.

Table 2.7.2 Register Account

Field	**Content**
Requirement	Register Account
Actor	Land Appraiser
Objective	Create a new appraiser account.
Precondition	The appraiser must have the activation key from the administrator.
Scenario	1. The appraiser selects ‘Register‘. 2. The appraiser enters the activation code. 3. The appraiser fills in required details (name, email, phone, password). 4. Verification code is sent to the entered email. 5. The appraiser clicks the link in the email to confirm the email. 6. The appraiser submits the form. 7. The system creates the account and confirms registration.
Exceptions	1. Email already in use. 2. Weak or invalid password. 3. Required fields missing. 4. Activation code expired or incorrect. 5. Server or network error.

Table 2.7.3 Logout

Field	**Content**
Requirement	Logout
Actor	Land Appraiser
Objective	Securely end the current session and prevent unauthorized access to the account.
Precondition	The appraiser is logged into the system.
Scenario	1. The appraiser clicks the Logout button from the system interface. 2. The system ends the current session. 3. The appraiser is sent to the login page.
Exceptions	1. Server or network error.

Table 2.7.4 Create Project

Field	**Content**
Requirement	Add Project
Actor	Land Appraiser
Objective	Create a new project and enter information needed to estimate the land price.
Precondition	Appraiser must be logged in.
Scenario	1. Appraiser selects ‘New Project’. 2. Names the Project. 3. Fills in land details. 4. The system checks and validates the input data.
Exceptions	1. Missing or invalid fields — display helpful error messages. 2. Network failure.

Table 2.7.5 Estimate Price

Field	**Content**
Requirement	Estimate Price
Actor	Land Appraiser
Objective	Predict and view the price of the land.
Precondition	Land data has been successfully submitted.
Scenario	1. Appraiser clicks “estimate price”. 2. System runs the model on the input. 3. Displays the estimated price and summary.
Exceptions	1. System error in model execution. 2. Timeout or delay in result. 3. Server or network error.

2.8 Admin's Functional Requirements Tables

Table 2.8.1 Manage Users

Field	Content
Requirement	View Users
Actor	Admin
Objective	View a list of all registered users with their details, and the ability to select any user to edit their account.
Precondition	Admin is logged in.
Scenario	1. Admin opens the user management panel. 2. System displays a list of all registered users with basic details (e.g., name, email, registration date, role). 3. Admin can sort or filter the list. 4. Admin can select any user to make actions. 5. The actions are: Delete User, Deactivate Account (if activated), Activate Account (if deactivated), and Change Role.
Exceptions	1. No users found in the system. 2. Server error when retrieving user data. 3. Database connection failure. 4. Failure of action.

Table 2.8.2 Manage Backups

Field	Content
Requirement	Manage Backups
Actor	Admin
Objective	Ensure system and user data is regularly backed up.
Precondition	Backup system is active.
Scenario	1. Admin opens 'Backup Settings'. 2. Triggers manual backup or sets automatic schedule. 3. Confirms successful completion.
Exceptions	1. Backup failed due to storage limit. 2. Scheduled backup skipped due to server downtime.

Table 2.8.3 Create Activation Key

Field	**Content**
Requirement	Create Activation Key
Actor	Admin
Objective	Generate a unique activation key for a data scientist or appraiser to use when registering their account.
Precondition	Admin is logged in.
Scenario	1. Admin opens the ”Activation Keys” panel. 2. Selects the account type (Data Scientist or Appraiser). 3. Clicks ”Generate Key”. 4. System generates a unique activation key.Clicks 5. Admin copies or sends the key Clicksto the intended recipient.
Exceptions	1. Server error during key generation. 2. Database access failure when saving the new key.

2.9 Data Scientist’s Functional Requirements Tables

Table 2.9.1 Test Model Accuracy

Field	**Content**
Requirement	Test Model Accuracy
Actor	Data Scientist
Objective	Evaluate the prediction accuracy of the machine learning model.
Precondition	The system must have a trained model and a dataset available for testing.
Scenario	1. The data scientist selects the ”Model Testing” section. 2. Uploads or selects a dataset for testing. 3. Runs the model to predict prices. 4. Compares predicted results with actual prices.
Exceptions	1. Incomplete or invalid test dataset. 2. Model not available or not trained.

Table 2.9.2 Review Feature Impact

Field	**Content**
Requirement	Review Feature Impact
Actor	Data Scientist
Objective	Analyze which features had the most influence on the land price prediction for a specific project.
Precondition	1. The model must be trained and support feature importance analysis. 2. The project must have completed the estimation.
Scenario	1. Access the “Projects” section. 2. Select a specific project from the list. 3. Open the “Feature Importance” view for that project. 4. View ranked list of features by their impact on the project’s prediction. 5. Export or download the report if needed.
Exceptions	1. Feature analysis tool is unavailable or unsupported for that project. 2. Insufficient project data to generate meaningful insights. 3. Selected project not found or inaccessible.

Table 2.9.3 Monitor Model Performance Over Time

Field	**Content**
Requirement	Monitor Model Performance Over Time
Actor	Data Scientist
Objective	Track how the model performs across different versions and datasets, and test any selected version on demand.
Precondition	1. System must store model versions, related datasets, and performance logs. 2. At least one model version must exist.
Scenario	1. Go to the “Model History” tab. 2. System displays a list of stored model versions with their details. 3. Select a version to see its past results. 4. Optionally, choose a dataset to re-test the selected model version. 5. System runs the test and shows the new results.
Exceptions	1. No stored model versions available. 2. Past performance data is missing or incomplete. 3. Testing fails due to corrupted data or unsupported dataset format. 4. Network or server error during testing.

2.10 Other less-related use cases

In addition to the functional requirements presented above, the system includes several additional use cases that are common across multiple user roles. These use cases—such as user login, logout, profile viewing, and profile editing—were omitted from the Admin and Data Scientist sections because they are already fully covered within the Normal User functional requirements and exhibit identical behavior across roles.

Furthermore, certain administrative functions, including user management and system log viewing, were excluded from the detailed tables as they are not central to the core objectives of the Land Price Estimator system.

Chapter 3: Architecture and Design

3.1 Overview

This chapter explains how the Land Price Estimator system is organized and how its parts work together. It covers the system's design, the chosen architecture and its possible alternatives, the database structure, and the main interfaces for the user, administrator, and data scientist.

3.2 Chosen Architecture Design

We studied multiple architecture design options, and concluded that the MVT (Model–View–Template) architecture is the best fit for our project.

Table 3.2.1 MVT Components

Component	**Role**
Model	Manages data, database structure, and rules.
View	Handles user actions; retrieves data from the model and selects the appropriate template to display results.
Template	Presentation layer controlling how data is rendered to the user (HTML).

The separation makes it easy for developers to work on different components of the application at the same time without affecting each other's work, and makes future scalability as well as maintaining, debugging, and testing the application easier.
Why MVT? The MVT architecture is provided by Django, which is the framework we are using to develop the web application for the Land-Price Prediction system.

3.3 Architecture Implementation

In our Land-Price Prediction system, the MVT architecture is implemented as follows:

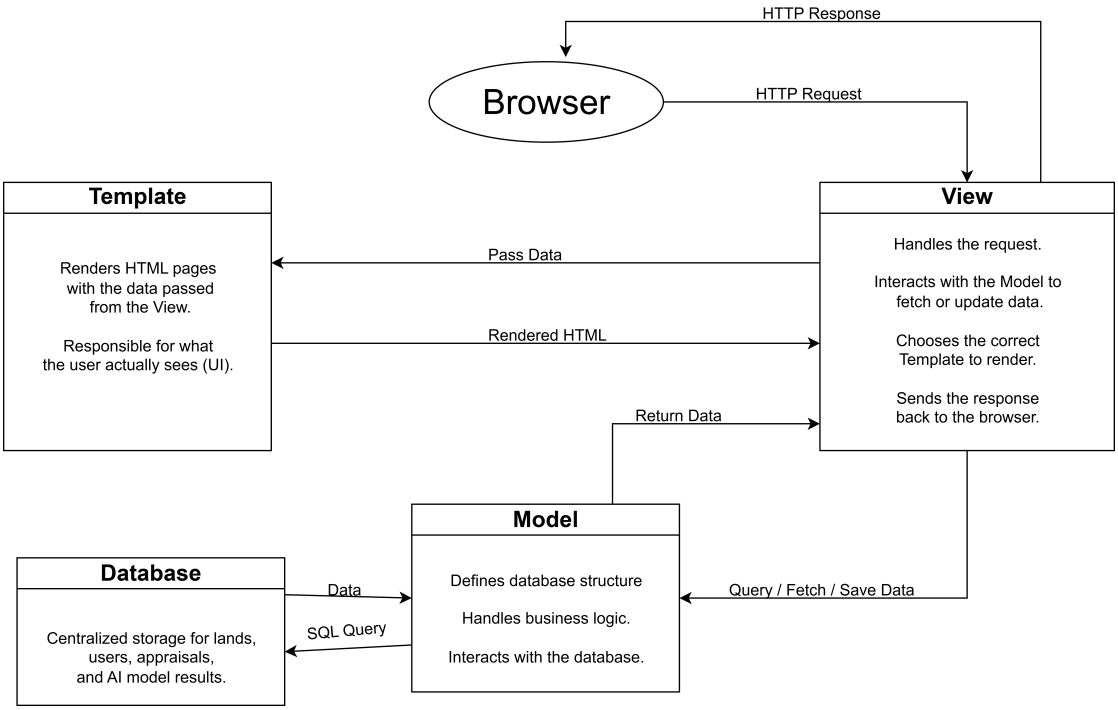
Model: Stores all the data related to each entity of the system, such as the lands and users attributes, and how they are stored in the database and how to retrieve them.

View: The view stores the business logic and connects the models with the templates; it processes user requests, retrieves data from the model, and gives it to the template.

Template: It's the interface that the user sees and interacts with. Through it, the user sees the results of the predictions and other information. The template has no business logic to ensure a clean separation from backend processing.

This structured method ensures that each layer is independent but still connected.

Figure 3.3.1 MVT Architecture



3.3.1 Example Models in the System

The project will have multiple models to manage the data effectively within the MVT architecture. Each model represents an entity of the Land-Price Prediction system and has its own attributes. Example models include:

1. **User:** Represents a system user. Attributes include `id`, `full_name`, `email`, `password`, `role`, and `created_at`.

2. **Project:** Represents a land estimation project. Attributes include `id`, `name`, `description`, `created_by`, `created_at`, and `status`.

3. **Plot (Land Parcel):** Represents a land parcel. Attributes include `id`, `plot_code`, `governorate_id`, `town_id`, `neighborhood_id`, `area_m2`, `slope`, `soil_type_id`, `rock_type_id`, `current_land_use`, `planned_land_use`, `far`, `coverage_ratio`, `is_current`, `version_no`, `created_by`, and `created_at`.

4. **Plot Document:** Represents documents related to a plot. Attributes include `id`,

plot_id, doc_type_id, issuing_authority_id, doc_number, issue_date, share_type, and share_ratio.

5. **Model:** Represents a machine learning model. Attributes include id, name, version, description, created_by, created_at, and is_active.

6. **Valuation:** Represents the output of a model prediction for a plot in a project. Attributes include id, project_id, plot_id, model_id, predicted_price, created_at, and created_by.

7. **Project Plot:** Represents the association between a project and its plots. Attributes include id, project_id, plot_id, valuation_id, and note.

8. **Plot Feedback:** Stores user feedback on a plot's estimated price. Attributes include id, plot_id, user_id, is_price_accepted, suggested_price, and created_at.

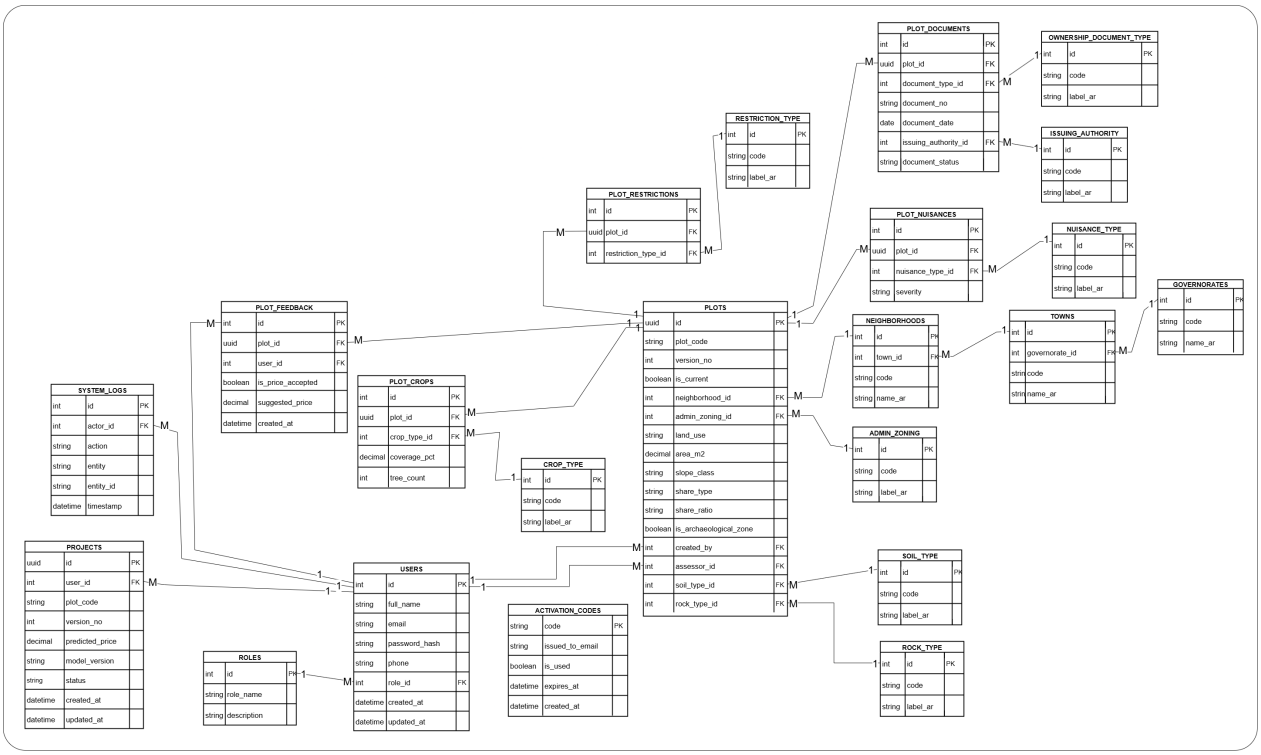
9. **Governorate, Town, Neighborhood:** Represent geographical hierarchy. Attributes include id, code, name_ar, and relevant foreign keys.

10. **Supporting Lookup Tables:** Includes Admin Zoning, Ownership Document Type, Issuing Authority, Soil Type, Rock Type, Crop Type, Nuisance Type, and Restriction Type with their respective codes and labels.

11. **Plot Crops, Plot Nuisances, Plot Restrictions:** Represent details of crops, nuisances, and restrictions on a plot. Attributes include plot foreign key, type foreign key, and specific measurements such as coverage_pct, tree_count, or severity.

3.4 ER Diagram

Figure 3.4.1 ER Diagram



3.5 Database Description

3.5.1 Users

- `id`: integer; PK; auto-increment.
- `full_name`: string; not-null.
- `email`: string; unique; not-null.
- `password`: string; not-null; ≥ 8 chars.
- `role`: enum {ADMIN, ASSESSOR, DATA_SCIENTIST}; not-null.
- `created_at`: datetime; not-null.

3.5.2 Projects

- `id`: UUID; PK.
- `name`: string; not-null.
- `description`: string; not-null.
- `created_by`: integer; FK $\rightarrow$ `users.id`; not-null.
- `created_at`: datetime; not-null.
- `status`: enum {ACTIVE, ARCHIVED}; not-null.

3.5.3 Plots (Land Parcels)

- `id`: UUID; PK.
- `plot_code`: string; unique; not-null.
- `governorate_id`: integer; FK $\rightarrow$ `governorates.id`; not-null.
- `town_id`: integer; FK $\rightarrow$ `towns.id`; not-null.
- `neighborhood_id`: integer; FK $\rightarrow$ `neighborhoods.id`; not-null.
- `area_m2`: `decimal(12,2)`; ≥ 0; not-null.
- `slope`: enum {FLAT, SLIGHT, MODERATE, STEEP}; not-null.
- `soil_type_id`: integer; FK $\rightarrow$ `soil_type.id`; nullable.
- `rock_type_id`: integer; FK $\rightarrow$ `rock_type.id`; nullable.
- `current_land_use`: enum {RES, COM, AGR, IND, MIX, VACANT}; not-null.

- `planned_land_use`: enum {RES, COM, AGR, IND, MIX, VACANT}; nullable.
- `far`: `decimal(6,3)`; ≥ 0; nullable.
- `coverage_ratio`: `decimal(4,2)`; 0–1; nullable.
- `is_current`: boolean; not-null.
- `version_no`: integer; not-null.
- `created_by`: integer; FK $\rightarrow$ `users.id`; not-null.
- `created_at`: datetime; not-null.

3.5.4 Plot_Documents

- `id`: integer; PK; auto-increment.
- `plot_id`: UUID; FK $\rightarrow$ `plots.id`; not-null.
- `doc_type_id`: integer; FK $\rightarrow$ `ownership_document_type.id`; not-null.
- `issuing_authority_id`: integer; FK $\rightarrow$ `issuing_authority.id`; not-null.
- `doc_number`: string; not-null.
- `issue_date`: date; not-null.
- `share_type`: enum {INDIVIDUAL, MUSHA}; not-null.
- `share_ratio`: string (e.g., “3/8”); nullable if `INDIVIDUAL`.

3.5.5 Models

- `id`: UUID; PK.
- `name`: string; not-null.
- `version`: string; not-null.
- `description`: string; not-null.
- `created_by`: integer; FK $\rightarrow$ `users.id`; not-null.
- `created_at`: datetime; not-null.
- `is_active`: boolean; not-null.

3.5.6 Valuations

- `id`: UUID; PK.
- `project_id`: UUID; FK → `projects.id`; not-null.
- `plot_id`: UUID; FK → `plots.id`; not-null.
- `model_id`: UUID; FK → `models.id`; not-null.
- `predicted_price`: `decimal(12,2)`; not-null.
- `created_at`: datetime; not-null.
- `created_by`: integer; FK → `users.id`; not-null.

3.5.7 Project_Plots

- `id`: integer; PK; auto-increment.
- `project_id`: UUID; FK → `projects.id`; not-null.
- `plot_id`: UUID; FK → `plots.id`; not-null.
- `valuation_id`: UUID; FK → `valuations.id`; nullable.
- `note`: string; optional short label.

3.5.8 Plot_Feedback

- `id`: integer; PK; auto-increment.
- `plot_id`: UUID; FK → `plots.id`; not-null.
- `user_id`: integer; FK → `users.id`; not-null.
- `is_price_accepted`: boolean; not-null.
- `suggested_price`: `decimal(12,2)`; required if not accepted.
- `created_at`: datetime; not-null.

3.5.9 Governorates

- `id`: integer; PK; auto-increment.
- `code`: string; unique; not-null.
- `name_ar`: string; not-null.

3.5.10 Towns

- `id`: integer; PK; auto-increment.
- `governorate_id`: integer; FK → `governorates.id`; not-null.
- `code`: string; unique; not-null.
- `name_ar`: string; not-null.

3.5.11 Neighborhoods

- `id`: integer; PK; auto-increment.
- `town_id`: integer; FK → `towns.id`; not-null.
- `code`: string; unique; not-null.
- `name_ar`: string; not-null.

3.5.12 Admin_Zoning

- `id`: integer; PK; auto-increment.
- `code`: string; unique; not-null.
- `label_ar`: string; not-null.

3.5.13 Ownership_Document_Type

- `id`: integer; PK; auto-increment.
- `code`: string; unique; not-null.
- `label_ar`: string; not-null.

3.5.14 Issuing_Authority

- `id`: integer; PK; auto-increment.
- `code`: string; unique; not-null.
- `label_ar`: string; not-null.

3.5.15 Restriction_Type

- `id`: integer; PK; auto-increment.
- `code`: string; unique; not-null.
- `label_ar`: string; not-null.

3.5.16 Plot_Restrictions

- `id`: integer; PK; auto-increment.
- `plot_id`: UUID; FK → `plots.id`; not-null.
- `restriction_type_id`: integer; FK → `restriction_type.id`; not-null.

3.6 Interfaces

Figure 3.6.1 Account Registration

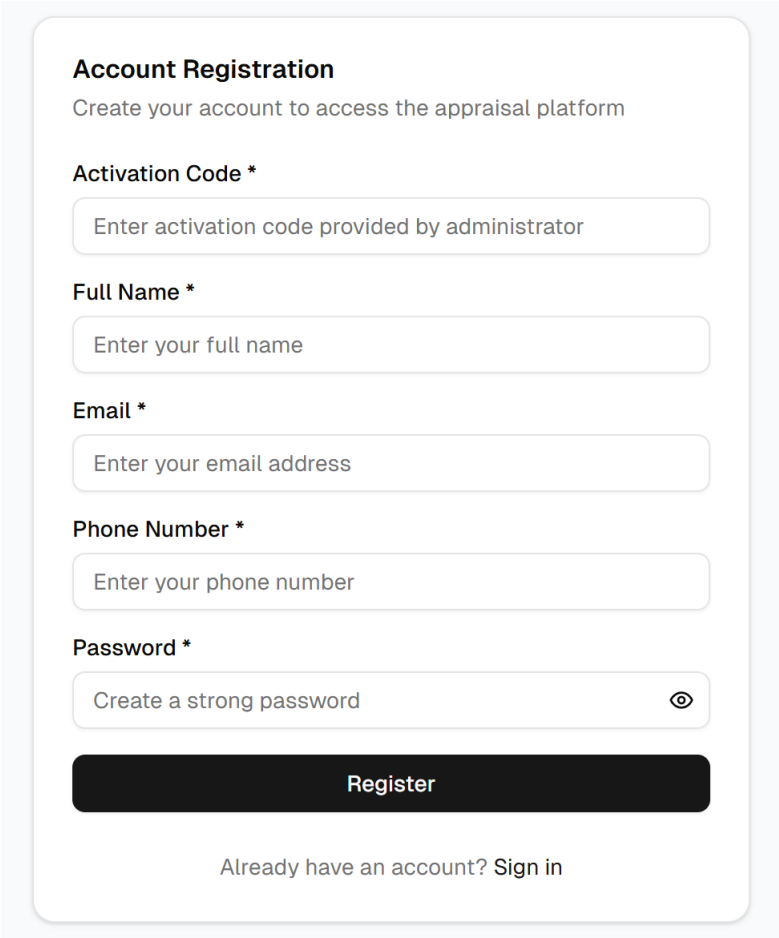


Figure 3.6.2 Create New Project

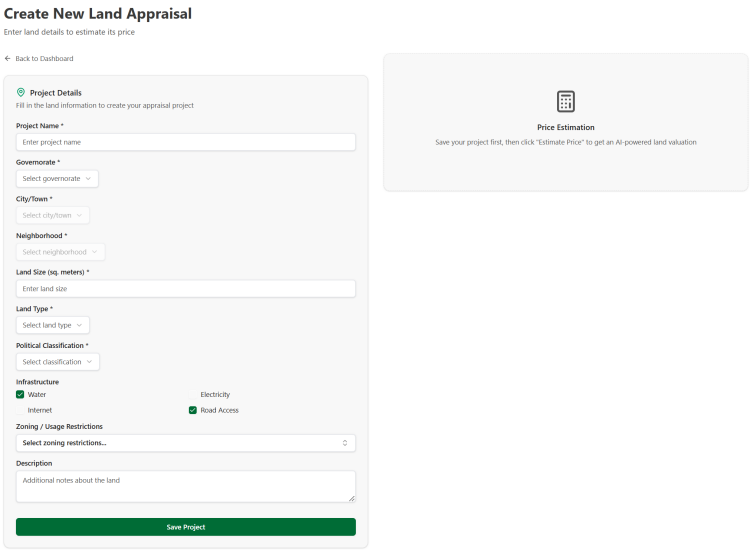


Figure 3.6.3 Test Model Accuracy

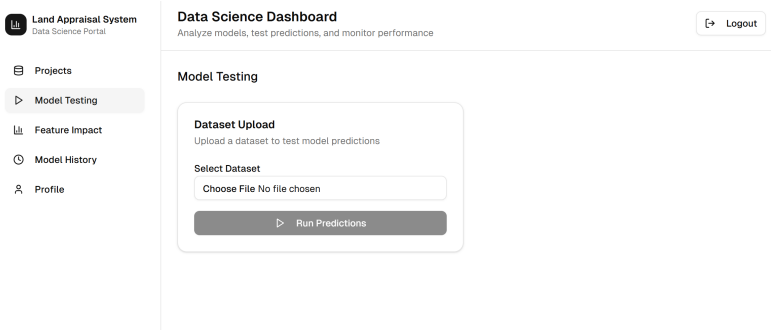


Figure 3.6.4 Review Feature Impact

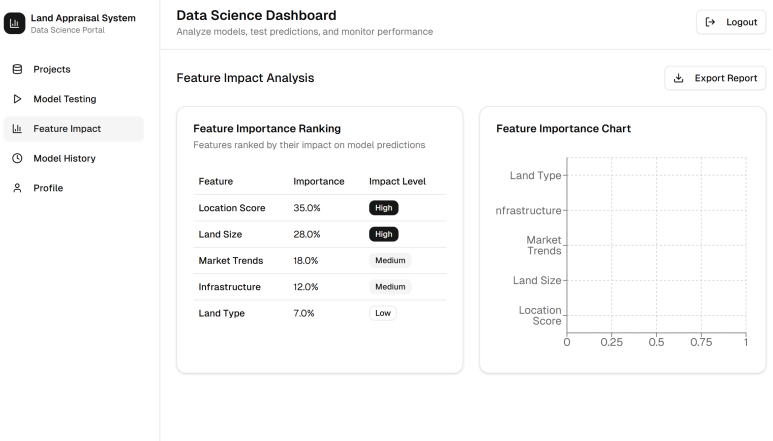


Figure 3.6.5 Monitor Model Performance Over Time

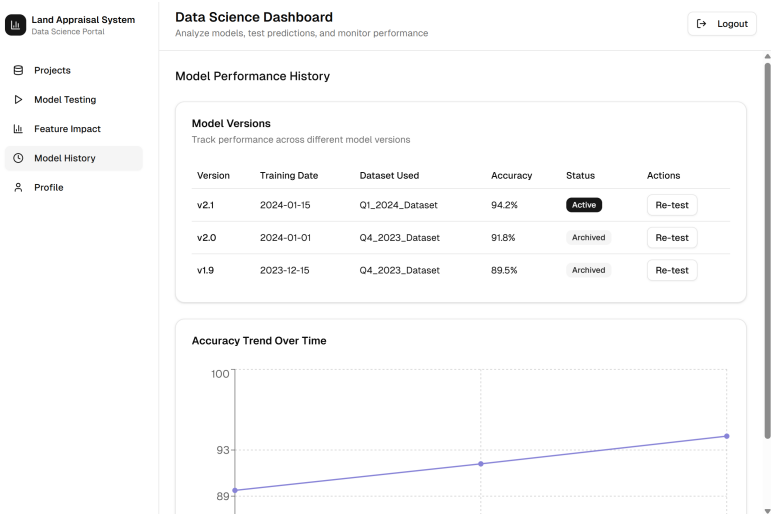


Figure 3.6.6 View And Manage Users

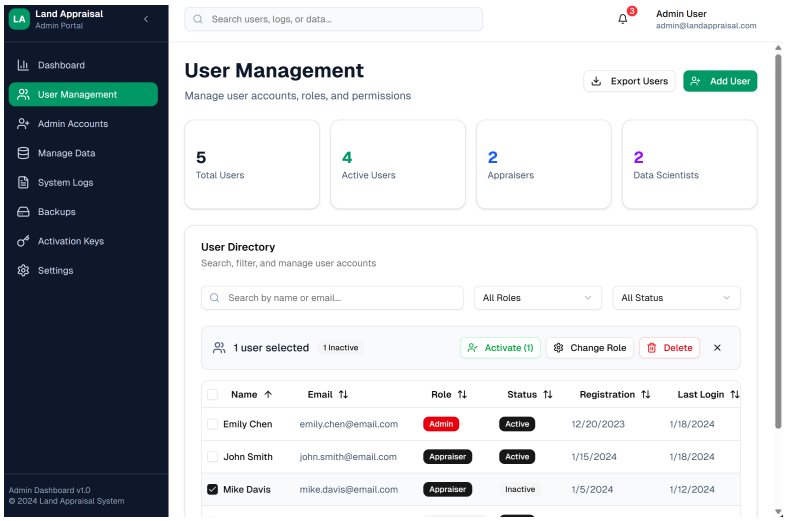
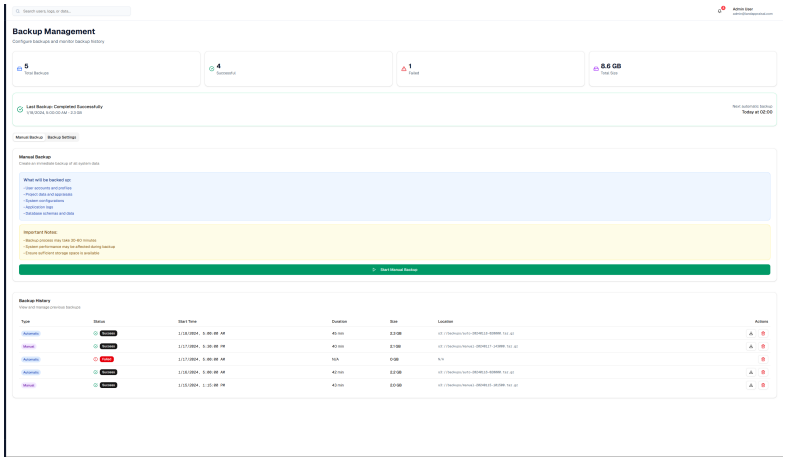


Figure 3.6.7 Manage Backups



Chapter 4: System Implementation

4.1 Overview

This chapter presents the implementation details of the Land Price Estimator system. It describes the software environment, frontend and backend development, database technology, system integration, and security mechanisms. Each section focuses on how the theoretical design presented in previous chapters was translated into a working and functional system.

4.2 Software Environment

4.2.1 Programming Languages

The Land Price Estimator system is built using a combination of programming and markup languages to efficiently handle both backend and frontend functionalities. The selected languages and their roles are described below:

- **Python:**
  Python is the main programming language used for the backend of the system. It is primarily utilized through the Django framework to implement the Model–View–Template (MVT) architecture, handle business logic, manage the database using the Object-Relational Mapper (ORM), and process user requests.

- **HTML (HyperText Markup Language):**
  HTML is used to structure the content of the web pages. It defines elements such as forms, links, buttons, and headings, providing the foundation for all frontend interfaces presented to users.

- **CSS (Cascading Style Sheets):**
  CSS is responsible for styling HTML elements, including layout, colors, fonts, spacing, and responsiveness. It ensures a consistent and visually appealing interface across different devices and browsers.

- **JavaScript:**
  JavaScript is used to implement interactive features on the frontend, such as dynamic form validation, responsive components, and user interface enhancements that improve overall usability.

4.2.2 Web Framework

The Land Price Estimator system is developed using the **Django** web framework. Django is a high-level Python framework that follows the **Model–View–Template (MVT)** ar-

chitectural pattern and is designed to support rapid development, scalability, and secure web applications.

Django serves as the core backend framework of the system. It handles HTTP requests, implements business logic, manages database interactions, and renders dynamic web pages. The MVT architecture is clearly reflected in the project structure, where models define the data schema, views contain application logic, and templates generate the user interface.

The system adopts a **server-side rendering** approach using Django's template engine. Dynamic content is generated on the server and delivered to users as fully rendered HTML pages, ensuring better security, simpler logic flow, and broad browser compatibility.

Django's built-in **Object-Relational Mapping (ORM)** is used to interact with the database using Python objects instead of raw SQL queries. This abstraction improves code readability and maintainability while reducing the risk of SQL injection attacks.

User authentication and authorization are implemented using Django's authentication framework with a **custom user model**. This allows the system to support multiple user roles, including normal users, data scientists, and administrators. Role-based access control is enforced using decorators such as `@login_required`, and users are redirected to the appropriate system section after successful login based on their role.

Django Forms and ModelForms are used to handle user input, perform server-side validation, manage file uploads, and securely process sensitive data such as passwords. Password handling relies on Django's built-in hashing mechanisms to ensure secure credential storage.

The project follows a **modular multi-application structure**, where each user role is implemented as a separate Django application. This design improves scalability, code organization, and long-term maintainability.

Overall, Django was selected due to its strong security features, clear architectural structure, built-in authentication system, and excellent support for data-driven web applications.

4.2.3 Development Tools

- **Integrated Development Environment (IDE):**
  Visual Studio Code (VS Code) was used as the primary IDE. It provides robust support for Python and Django development, including syntax highlighting, debugging tools, code formatting, and extensibility through plugins. Its lightweight and flexible design made it suitable for managing both frontend and backend code.

- **Version Control System:**

Git was used for version control to track code changes and manage development history. The project repository was hosted on GitHub, enabling centralized code management, backups, and collaboration.

- **Database Management:**
  SQLite was used during development as the default database system provided by Django. It is lightweight and easy to configure. Database interactions were handled using Django's ORM without writing raw SQL queries.

- **Web Browser and Testing Tools:**
  Modern web browsers such as Brave and Comet were used for frontend testing and debugging. Built-in developer tools were used to inspect HTML elements, debug JavaScript, and test responsiveness across different screen sizes.

- **Machine Learning Development Environment:**
  Google Colab was used to develop, train, and validate the machine learning model for land price prediction. Using Python libraries such as scikit-learn, a Decision Tree Regression model was trained on historical land data. The trained model was then exported and integrated into the Django backend to provide real-time price predictions based on user input.

4.2.4 Supporting Libraries

In addition to the main frameworks and tools, several supporting libraries were used to facilitate backend development, form handling, authentication, and machine learning:

- **Django Built-in Libraries:**
  Django's built-in libraries were used for authentication, authorization, request handling, and form processing. These utilities ensure that only authorized users can access restricted system features.

- **Django Forms:**
  Django's ModelForm system was used to automatically generate forms based on database models. This simplifies form creation, enforces server-side validation, and ensures consistency between user input and stored data.

- **Machine Learning Libraries:**
  Scikit-learn was used to implement the Decision Tree Regression algorithm. The `train_test_split` function was used to divide the dataset into training and testing sets to ensure proper model evaluation.

- **Data Processing Library:**
  Pandas was used for data manipulation and preprocessing during model development. It was utilized to load datasets, handle missing values, and prepare structured data suitable for training the regression model.

4.3 Frontend Implementation

4.3.1 General Frontend Design

The frontend of the Land Price Estimator system is a web-based interface accessed through a standard web browser, requiring no additional software. It is designed to be simple, clear, and accessible across different devices.

The system uses Django server-side rendering, where pages are generated as complete HTML documents. This approach ensures reliable performance and seamless integration with backend features such as authentication and form processing.

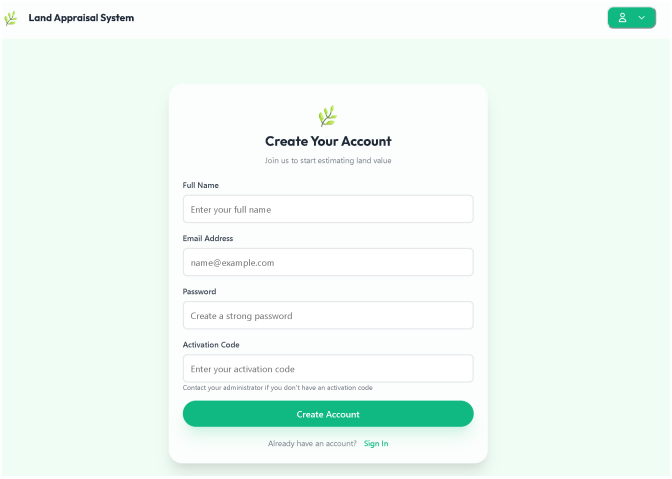
A base template is used to provide a consistent layout and shared components across all pages, while individual templates extend it for specific functionality. Styling is implemented using CSS to achieve a responsive and uniform design. Minimal JavaScript is used to support basic interactivity and enhance usability.

4.3.2 Authentication Interfaces

The system provides authentication interfaces that control access to the platform and ensure that only authorized users can register and log in. These interfaces include the registration page and the login page.

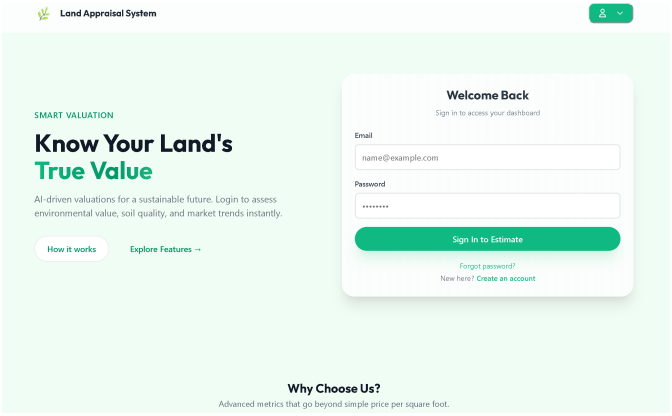
During registration, users are required to enter an activation code provided by the system administrator. This activation code determines the user role (such as appraiser, administrator, or data scientist) and controls which part of the system the user can access after registration. This approach ensures controlled user onboarding and role-based access from the moment the account is created.

Figure 4.3.1 Registration Page



The login interface allows registered users to authenticate using their email and password. After successful login, users are automatically redirected to their respective dashboard based on their assigned role. If authentication fails, the system displays clear error messages to guide the user.

Figure 4.3.2 Login Page



Both authentication interfaces are designed to be simple and user-friendly, with clear input fields, validation feedback, and consistent styling. This design helps users complete authentication tasks easily while maintaining security and proper access control.

4.3.3 User Dashboard (Home Page)

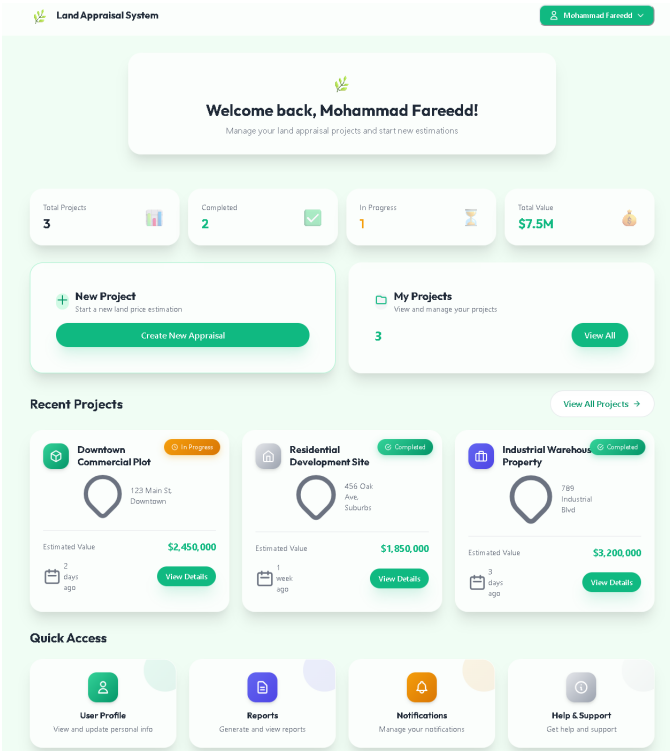
After successful authentication, users are redirected to their dashboard, which serves as the main entry point to the system. The dashboard provides an overview of the user's activity and quick access to the system's core features.

The interface includes clear navigation options that allow the user to:

- View existing land price estimation projects.
- Create a new project (land price estimation).
- Access their profile and account settings.

The dashboard layout is designed to be simple and role-aware, meaning each user sees options relevant to their assigned role (e.g., appraiser or data scientist). This improves usability and prevents access to unauthorized features.

Figure 4.3.3 Home Page



Overall, the dashboard acts as a centralized control panel, enabling users to efficiently navigate the system and manage their land price estimation projects.

4.3.4 Project Management Interface (View Projects)

The Project Management Interface allows users to view and manage all previously created land price estimation projects in an organized manner. This page displays a list of the user's projects, including key information such as project name, creation date, and current status.

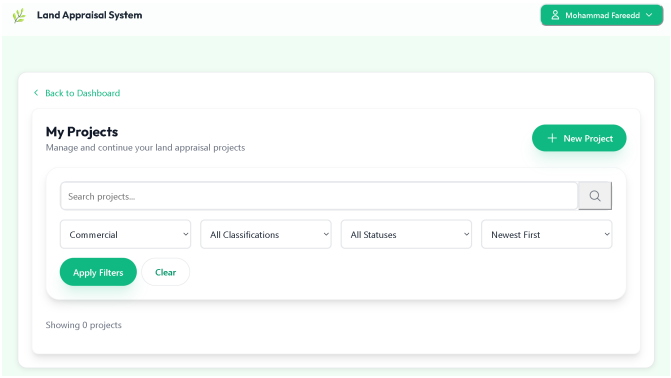
To improve usability and efficiency, the interface provides several tools that help users quickly locate specific projects:

- Search functionality to find projects by name or keyword.

- Filtering options to narrow results based on project status or other attributes.
- Sorting options to order projects by date.

Each project entry includes actions that allow the user to open the project, review its details, or continue working on it. This interface helps users manage multiple estimations efficiently without confusion.

Figure 4.3.4 Project Management Interface



The project management page plays a crucial role in organizing user data and ensuring easy access to land price estimation records.

4.3.5 New Project / Land Price Estimation Interface

The New Project / Land Price Estimation Interface allows users to create a new land evaluation project by entering detailed information about a land parcel. This page is centered around a structured form designed to collect all required attributes needed for price estimation.

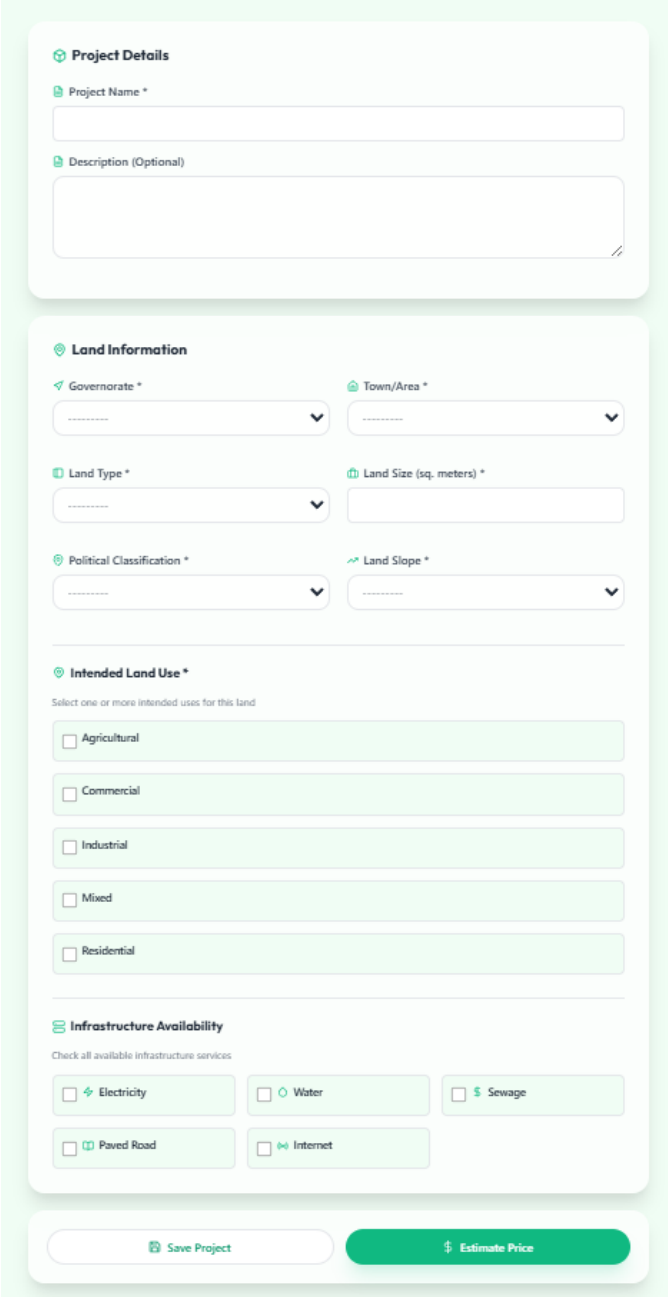
The form includes fields related to land characteristics such as location, area, zoning type, and other relevant parameters. Input validation is applied to ensure data accuracy before submission.

Users are provided with two main actions:

- **Save Project:** Stores the entered data without performing price estimation, allowing users to complete or modify the project later.
- **Estimate Price:** Submits the land data to the system, triggers the price estimation process, and automatically saves the project along with the estimated land price.

The interface is designed to be simple, guiding users step-by-step through the data entry process while minimizing errors. Clear labels and structured layout help ensure a smooth user experience.

Figure 4.3.5 New Project Form



This interface represents the core functionality of the system, connecting user input with the land price estimation process.

4.4 Backend Implementation

4.4.1 Overview of Backend Architecture

The backend of the Land Price Estimator system is implemented using the Django web framework and follows the Model–View–Template (MVT) architectural pattern. All

core application logic, including authentication, project management, role-based access control, and machine learning inference, is handled on the server side.

The backend is responsible for:

- Managing user authentication and authorization
- Handling project creation, storage, and retrieval
- Enforcing user ownership over projects
- Integrating a trained machine learning model for land price estimation
- Serving validated data to frontend templates

The system does not rely on external API servers; all logic is processed internally within Django.

4.4.2 Application Structure and Separation of Concerns

The backend is organized into multiple Django applications, each responsible for a specific domain:

- **Users_Handling_App**
  Handles authentication, registration, activation codes, and user redirection based on roles.
- **normal_user**
  Represents the main application used by land appraisers. It includes project creation, project listing, filtering, and machine learning–based price estimation.
- **data_scientist & admin apps**
  Planned for future work. Currently, administrative tasks are handled via Django's built-in admin panel.

This modular structure improves maintainability, scalability, and clarity of responsibilities.

Figure 4.4.1 Project structure

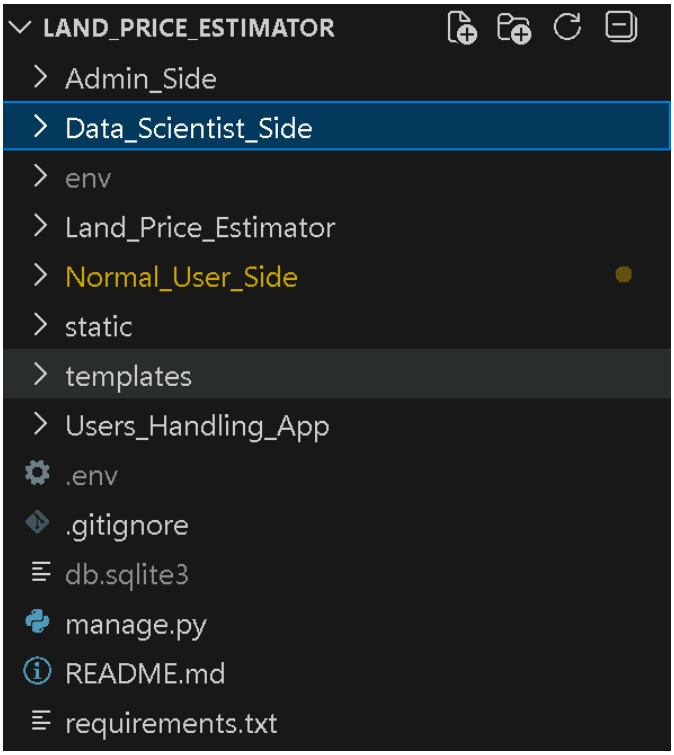
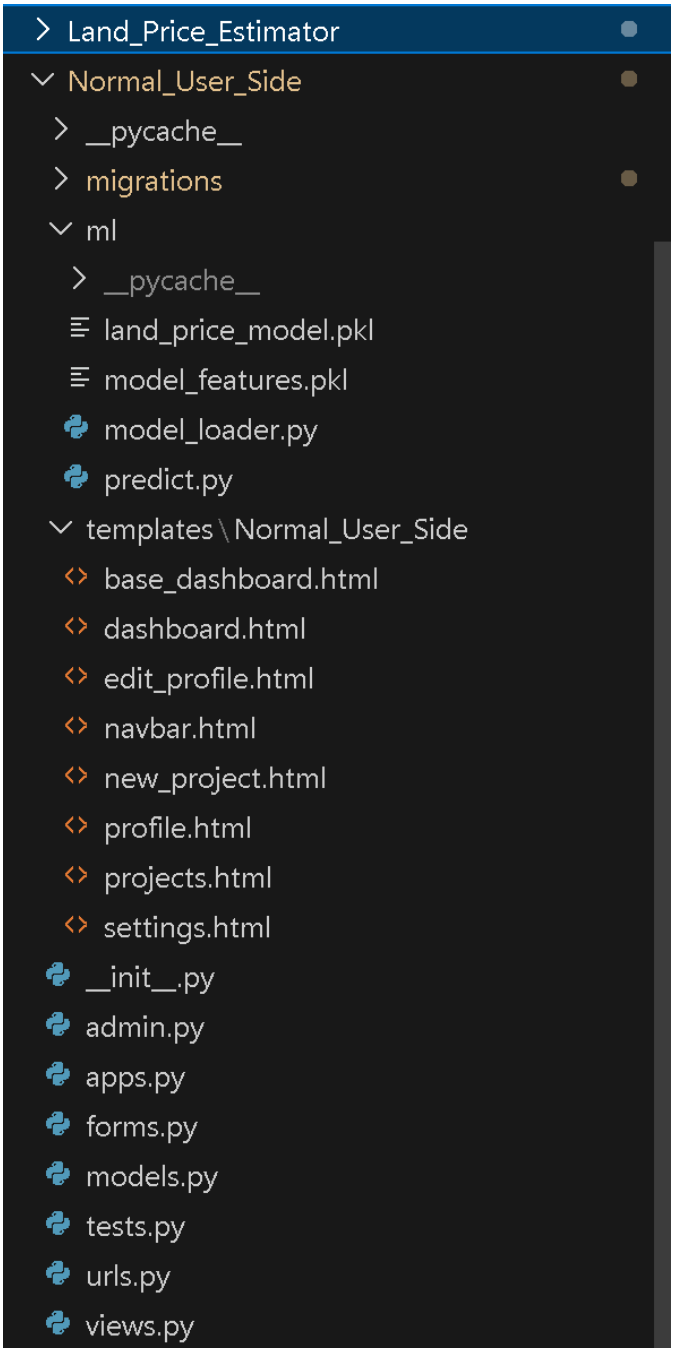


Figure 4.4.2 Normal user side app structure



4.4.3 Data Models and Database Design

Custom User Model

A custom user model is implemented by extending Django's `AbstractUser`, replacing the username with email-based authentication. Each user is assigned a role that determines system access.

Key features:

- Email-based login
- Role-based user types (Land Appraiser, Data Scientist, Admin)
- Custom user manager for controlled user creation

Figure 4.4.3 Custom user model

```python
class User(AbstractUser):
    USER_TYPES = [
        ('normal', 'Land Appraiser'),
        ('scientist', 'Data Scientist'),
        ('admin', 'Admin'),
    ]

    username = None
    name = models.CharField(max_length=200, blank=True)
    email = models.EmailField(unique=True)
    type = models.CharField(max_length=20, choices=USER_TYPES, default='normal')
    phone = models.CharField(max_length=20, blank=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = CustomUserManager()

    def __str__(self):
        return self.email
```

Project Model

The Project model represents a land valuation request created by a land appraiser. Each project is associated with exactly one user, while each user may have multiple projects.

Key attributes include:

- Governorate
- Land size
- Land type
- Political classification
- Project status (draft or completed)
- Creation date

This design enforces ownership and allows secure filtering of projects per user.

Figure 4.4.4 Project model

```python
class Project(models.Model):
    LAND_TYPES = [
        ("agricultural", "Agricultural"),
        ("residential", "Residential"),
        ("commercial", "Commercial"),
        ("industrial", "Industrial"),
        ("mixed", "Mixed Use"),
    ]

    STATUS_CHOICES = [
        ("draft", "Draft"),
        ("completed", "Completed"),
    ]

    POLITICAL = [
        ("a", "Area A"),
        ("b", "Area B"),
        ("c", "Area C"),
    ]
    GOVERNORATES = [
        ("ramallah", "Ramallah"),
        ("nablus", "Nablus"),
        ("bethlehem", "Bethlehem"),
        ("jenin", "Jenin"),
        ("hebron", "Hebron"),

    ]

    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='projects'
    )

    name = models.CharField(max_length=255)
    governorate = models.CharField(max_length=50, choices=GOVERNORATES)
    land_size = models.PositiveIntegerField()
    land_type = models.CharField(max_length=50, choices=LAND_TYPES)
    political_classification = models.CharField(max_length=50, choices=POLITICAL)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES)
    description = models.TextField(blank=True)
    date_created = models.DateField(auto_now_add=True)
    estimated_price = models.FloatField(null=True, blank=True)
```

4.4.4 Forms and Server-Side Validation

Django ModelForms are used to handle structured user input and enforce validation rules.

- **UserForm**
  Used for profile updates, including secure password change with validation.
- **ProjectForm**
  Used to create new land valuation projects. The form maps directly to the Project model and ensures data consistency.

Server-side validation ensures:

- Required fields are enforced
- Invalid data is rejected
- Security is maintained regardless of frontend behavior

Figure 4.4.5 Project form

```python
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = [
            'name',
            'governorate',
            'land_type',
            'political_classification',
            'status',
            'description',
            'land_size'
        ]
```

Figure 4.4.6 User form

```python
class UserForm(forms.ModelForm):
    current_password = forms.CharField(widget=forms.PasswordInput, required=False)
    new_password = forms.CharField(widget=forms.PasswordInput, required=False)
    confirm_password = forms.CharField(widget=forms.PasswordInput, required=False)

    class Meta:
        model = User
        fields = ['name', 'email', 'phone']

    def clean(self):
        cleaned_data = super().clean()
        current_password = cleaned_data.get("current_password")
        new_password = cleaned_data.get("new_password")
        confirm_password = cleaned_data.get("confirm_password")

        if new_password or confirm_password:
            if not current_password:
                raise ValidationError("Current password is required.")

            if not check_password(current_password, self.instance.password):
                raise ValidationError("Current password is incorrect.")

            if new_password != confirm_password:
                raise ValidationError("Passwords do not match.")

            validate_password(new_password, user=self.instance)

        return cleaned_data

    def clean_phone(self):
        phone = self.cleaned_data.get("phone")
        if phone and not re.match(r'^\+?[0-9]{7,15}$', phone):
            raise ValidationError("Enter a valid phone number.")
        return phone
```

4.4.5 Project Creation and State Management

When a user submits the project creation form, the backend distinguishes between two actions:

- **Save as Draft**
  Stores the project without invoking the machine learning model.

- **Estimate Price**
  Triggers the ML prediction process and marks the project as completed.

This logic is handled within the Django view by detecting the submitted button name.

Figure 4.4.7 newProject view

```
def newProject(request):
    if request.method == 'POST':
        form = ProjectForm(request.POST)
        if form.is_valid():
            project = form.save(commit=False)
            project.user = request.user

            if request.POST.get('action') == 'estimate':
                input_data = {
                    "governorate": project.governorate,
                    "land_size": project.land_size,
                    "land_type": project.land_type,
                    "political_classification": project.political_classification,

                }
                try:
                    project.estimated_price = predict_land_price(input_data)
                    project.status = 'completed'
                except Exception as e:
                    messages.error(request, "Price estimation failed. Please try again later.")
                    project.status = 'draft'
            else:
                project.status = 'draft'

            project.save()
            return redirect('normal_user:projects')

    else:
        form = ProjectForm()

    return render(request, 'Normal_User_Side/new_project.html', {'form': form})
```

4.4.6 Machine Learning Model Integration

A Decision Tree Regression model was trained externally using Google Colab and exported using `joblib`. The trained model is loaded into Django and used for real-time inference.

ML Integration Workflow:

1. Project data is validated and saved
2. Relevant features are extracted
3. Data is preprocessed to match training format
4. The trained model predicts land price
5. Prediction is stored in the database
6. Project status is updated to completed

The ML logic is isolated in a dedicated module to maintain separation between business logic and machine learning inference.

Figure 4.4.8 Import the ML model

```
import joblib
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent
MODEL_PATH = BASE_DIR / "land_price_model.joblib"

model = None
model_features = None

def load_model():
    global model, model_features
    if model is None:
        model = joblib.load(MODEL_PATH)
        model_features = model.feature_names_in_
    return model, model_features
```

Figure 4.4.9 Prediction function

```
import pandas as pd
from Normal_User_Side.ml.model_loader import load_model

def predict_land_price(input_data):
    model, model_features = load_model()
    X = pd.DataFrame([input_data])[model_features]
    return model.predict(X)[0]
```

4.4.7 Access Control and Data Security

The backend enforces strict access control rules:

- Users can only view and manage their own projects
- Authentication is required for all protected views
- Role-based redirection ensures users are sent to the correct interface
- Django's built-in authentication and session management mechanisms are used

Filtering and sorting operations are processed on the server side to prevent unauthorized data exposure.

Figure 4.4.10 Filtering logic in the view

```python
def viewProjects(request):
    projects = Project.objects.filter(user=request.user).order_by('-date_created')

    land_type = request.GET.get('land_type')
    political_type = request.GET.get('political_type')
    status = request.GET.get('status')
    search_query = request.GET.get('search')


    if land_type:
        projects = projects.filter(land_type=land_type)
    if political_type:
        projects = projects.filter(political_classification=political_type)
    if status:
        projects = projects.filter(status=status)
    if search_query:
        projects = projects.filter(name__icontains=search_query)
    context = {
        'projects': projects,
        'land_types': Project.LAND_TYPES,
        'political_types': Project.POLITICAL,
        'statuses': Project.STATUS_CHOICES,
    }
    return render(request, 'Normal_User_Side/projects.html',context)
```

4.5 Database Technology Implementation

This section explains the database technology used in the Land Price Estimator system. It describes the selected database engine, the data access methodology, core entities, relationships, validation mechanisms, and schema management techniques. The database layer plays a critical role in maintaining data consistency, integrity, and reliability throughout the system.

4.5.1 Database Engine Selection

The system uses **SQLite** as its database management system. SQLite was selected due to its lightweight architecture, ease of deployment, and seamless integration with the Django framework. As a file-based database engine, SQLite does not require a separate database server, which makes it suitable for academic projects and prototype systems.

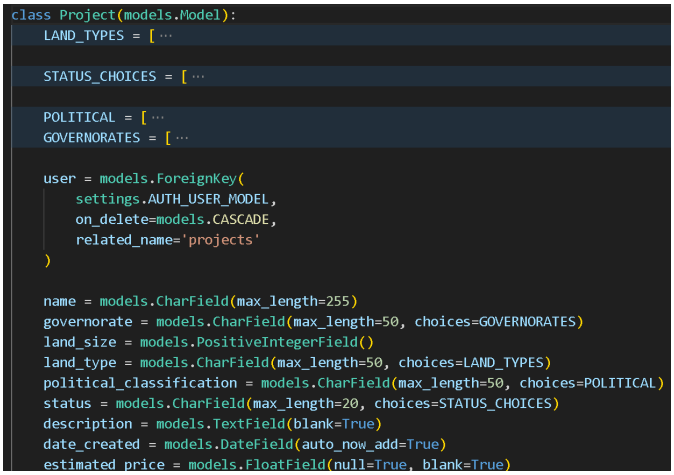
The use of SQLite allows rapid development, straightforward testing, and efficient storage of structured relational data during the project lifecycle.

4.5.2 Database Access Using Django ORM

All interactions with the database are handled through the **Django Object-Relational Mapping (ORM)** layer. The ORM abstracts direct SQL queries and enables developers to manipulate database records using Python objects and classes.

Each database table is represented as a Django model, where fields correspond to table columns, and relationships are expressed using foreign key definitions. This approach improves code readability, simplifies maintenance, and reduces the risk of security vulnerabilities such as SQL injection.

Figure 4.5.1 Django Models Definition



```
class Project(models.Model):
    LAND_TYPES = [...

    STATUS_CHOICES = [...

    POLITICAL = [...
    GOVERNORATES = [...

    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='projects'
    )

    name = models.CharField(max_length=255)
    governorate = models.CharField(max_length=50, choices=GOVERNORATES)
    land_size = models.PositiveIntegerField()
    land_type = models.CharField(max_length=50, choices=LAND_TYPES)
    political_classification = models.CharField(max_length=50, choices=POLITICAL)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES)
    description = models.TextField(blank=True)
    date_created = models.DateField(auto_now_add=True)
    estimated_price = models.FloatField(null=True, blank=True)
```

4.5.3 Core Database Entities

The database schema is centered around two main entities:

Users store authentication and role-related information for system participants, including administrators, land appraisers, and data scientists.

Projects represent land valuation tasks created by users. Each project is associated with its creator, ensuring ownership tracking and controlled access to project data.

This structure supports clear separation of responsibilities and organized storage of estimation records.

4.5.4 Primary Keys and Relationships

All tables use **BigAutoField** as their primary key, which provides an auto-incremented integer identifier for each record. This ensures uniqueness and efficient indexing.

Foreign keys are used to establish relationships between tables, such as linking projects to users. These relationships enforce **referential integrity**, ensuring that dependent records always reference valid parent records.

UUID values are not used as primary keys. Instead, UUIDs are generated only for activation codes, providing secure and unpredictable identifiers for account activation purposes.

Figure 4.5.2 Primary and Foreign Key Relationships

```
user = models.ForeignKey(
    settings.AUTH_USER_MODEL,
    on_delete=models.CASCADE,
    related_name='projects'
)
```

4.5.5 Data Validation and Constraints

Data integrity is ensured through multiple validation mechanisms at the model level. These include enforcing correct data types, restricting null values for essential fields, and applying unique constraints on sensitive attributes such as email addresses.

Primary and foreign key constraints further guarantee the consistency of relational data. This validation strategy minimizes invalid entries and preserves database reliability throughout system operation.

4.5.6 Database Migrations

The database schema is managed using Django's built-in migration framework. Migrations allow controlled and incremental updates to the database structure while preserving existing data.

Commands such as `makemigrations` and `migrate` are used to synchronize model definitions with the SQLite database schema.

4.5.7 Dataset Usage

The current implementation relies on **synthetic data** generated for development, testing, and model evaluation. This approach enables controlled experimentation without dependency on sensitive or unavailable real-world datasets.

Future work includes integrating real land data through an official land registry API, which would allow continuous updates and enhance prediction accuracy and system realism.

4.5.8 Design Considerations

The combination of SQLite and Django ORM provides a balanced solution between simplicity, maintainability, and performance. The database design allows smooth migration

to more advanced database engines in future deployments without major architectural changes.

4.6 System Integration

System integration ensures that all components of the Land Price Estimator system work together seamlessly. This includes the interaction between the frontend, backend, database, machine learning model, and email service. Proper integration allows users to submit data, receive estimations, and manage projects efficiently while maintaining security and role-based access.

4.6.1 Integration of Frontend and Backend

The frontend interface communicates with the backend through Django views using server-side rendering. When a user interacts with the interface, such as submitting a new project form, the following occurs:

1. The form data is sent via a POST request to the corresponding Django view (newProject).
2. The view validates the data using Django forms (ProjectForm) and performs the requested action, either saving as a draft or estimating the land price.
3. After processing, the user is redirected to the appropriate page (e.g., project list or dashboard).

4.6.2 Integration with the Machine Learning Model

The system integrates the Decision Tree Regression model for estimating land prices:

1. Upon selecting the “Estimate Price” action, the backend view prepares the input data from the form.
2. The input data is sent to the predict_land_price() function, which uses the model_loader to load the exported ML model.
3. The model predicts the price and the result is stored in the Project model in the database.

Error handling is implemented to manage cases where the ML model is missing or input data is invalid, ensuring users receive a user-friendly message instead of a system error.

4.6.3 Role-Based Access Control Integration

User roles are integrated to ensure security and appropriate access:

- Normal Users (Appraisers) can create projects, view their projects, and estimate land prices.
- Data Scientists (planned for future work) will access projects for analysis and model improvement.
- Administrators manage users, activation codes, and project oversight via the Django admin panel.

The backend enforces these restrictions using Django authentication, the User model, and role checks in views.

4.6.4 Integration of Email Service

The email service is integrated for password resets:

1. Users can request a password reset through the frontend.
2. Django's built-in password reset views handle the request and send an email using the configured SMTP backend.
3. Users follow the link in the email to reset their password securely.

Figure 4.6.1 Email service integration in settings.py

```
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp-relay.brevo.com"
EMAIL_PORT = 587
EMAIL_USE_TLS = True

EMAIL_HOST_USER = os.environ.get("BREVO_SMTP_USER")
EMAIL_HOST_PASSWORD = os.environ.get("BREVO_SMTP_PASS")

DEFAULT_FROM_EMAIL = "LandPriceEstimator <msamalq306@gmail.com>"
```

4.6.5 Data Flow and Database Integration

All user actions and generated data are stored in the Project model, while user credentials and roles are managed in the Users_Handling_App models. The integration ensures:

- Data from forms is validated before saving.
- ML predictions are stored alongside the project data.

- Only authorized users can view or modify their own projects.

Data flow:

User input → View (processing & validation) → Database → Dashboard / Project List update

4.7 Security Implementation

This section describes the security mechanisms applied in the system to protect user data, prevent unauthorized access, and ensure safe interaction between system components. The implementation relies on Django's built-in security features combined with application-level controls.

4.7.1 Authentication and Access Control

User authentication is handled using Django's built-in authentication system. Only authenticated users can access protected views such as project creation, project listing, and land price estimation. Access to system functionality is restricted based on user roles (e.g., normal user, administrator, data scientist). Each user can only access views and data relevant to their role, preventing unauthorized actions or data exposure.

Figure 4.7.1 Authentication form

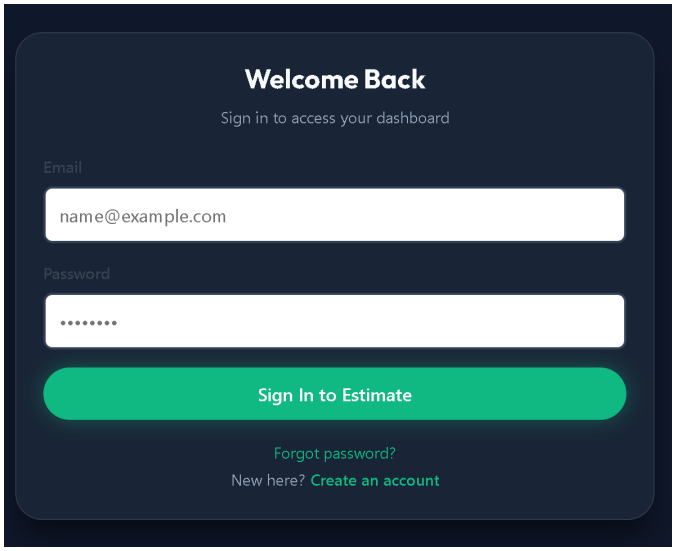


Figure 4.7.2 Authentication in loginPage view

```
user = authenticate(request, email=email, password=password)

if user is not None:
    login(request, user)
```

4.7.2 Authorization and Data Isolation

To ensure data privacy, users are restricted to viewing and managing only their own projects. Database queries are filtered at the view level to return records associated exclusively with the currently authenticated user. This approach prevents the user from attempting to access or manipulate another user's data.

Figure 4.7.3 User-based query filtering in viewProjects view

```
projects = Project.objects.filter(user=request.user).order_by('-date_created')
```

4.7.3 Password Management and Account Recovery

Passwords are securely stored using Django's hashing framework, which applies industry-standard hashing algorithms. The system also supports password recovery through email-based password reset functionality, implemented using Django's built-in password reset views. This ensures that users can safely recover access to their accounts without exposing sensitive credentials. In addition to password recovery via email, authenticated users are allowed to change their passwords from the profile editing interface. To enhance security, users are required to provide their current password before setting a new one. The system validates the old password server-side before applying any changes, preventing unauthorized password updates.

Figure 4.7.4 Change Password section in edit_profile page

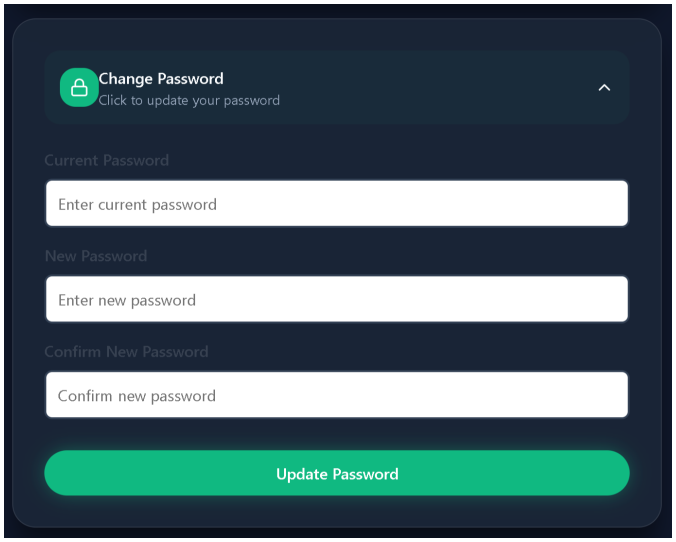
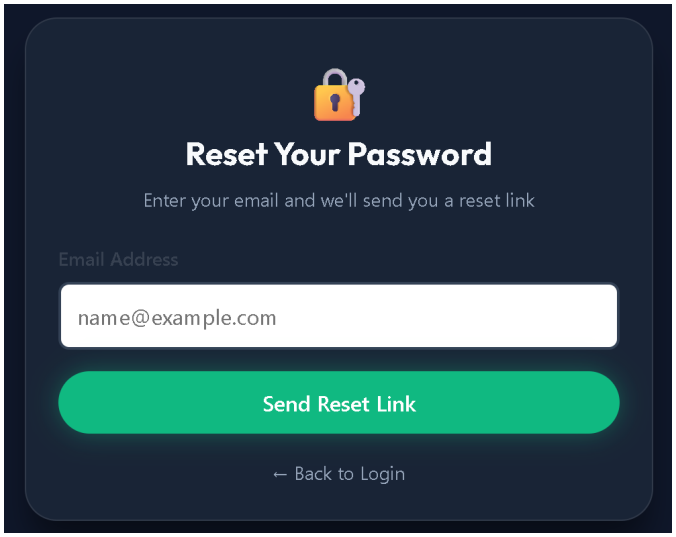


Figure 4.7.5 Reset password via email



4.7.4 Form Validation and Input Protection

All user inputs are validated on the server side using Django forms and model validation. This includes checking required fields, validating data formats, and enforcing business rules before any data is stored in the database. This reduces the risk of invalid data entry and helps protect against common attacks such as malformed input and unintended data manipulation.

Figure 4.7.6 ModelForm validation

```
form = ProjectForm(request.POST)
if form.is_valid():
    project = form.save(commit=False)
    project.user = request.user
```

4.7.5 Cross-Site Request Forgery (CSRF) Protection

All POST requests in the system are protected using Django's CSRF middleware. CSRF tokens are embedded in all forms and verified on submission, ensuring that requests originate from legitimate users and not malicious third-party sources. This protection is enabled by default and requires no additional configuration beyond proper template usage.

Figure 4.7.7 CSRF token usage example

```
<form method="POST" action="">
    {% csrf_token %}
```

4.7.6 Error Handling and Safe Failure

The system includes structured error handling to prevent application crashes and avoid exposing sensitive internal information to users. For example, failures in machine learning prediction or missing model files are caught and handled gracefully, displaying user-friendly messages instead of system errors. This approach improves both system reliability and security by preventing information leakage.

Figure 4.7.8 Exception handling during land price estimation

```
try:
    project.estimated_price = predict_land_price(input_data)
    project.status = 'completed'
except Exception as e:
    messages.error(request, "Price estimation failed. Please try again later.")
    project.status = 'draft'
```

4.7.7 Deployment and Future Security Enhancements

The current system is developed in a local environment; however, it is designed to be production-ready. Future deployment plans include enabling HTTPS, secure email credentials, environment-based configuration, and stricter logging and monitoring. These measures will further strengthen system security when deployed in a production environment.

Chapter 5: Testing

5.1 Unit Testing

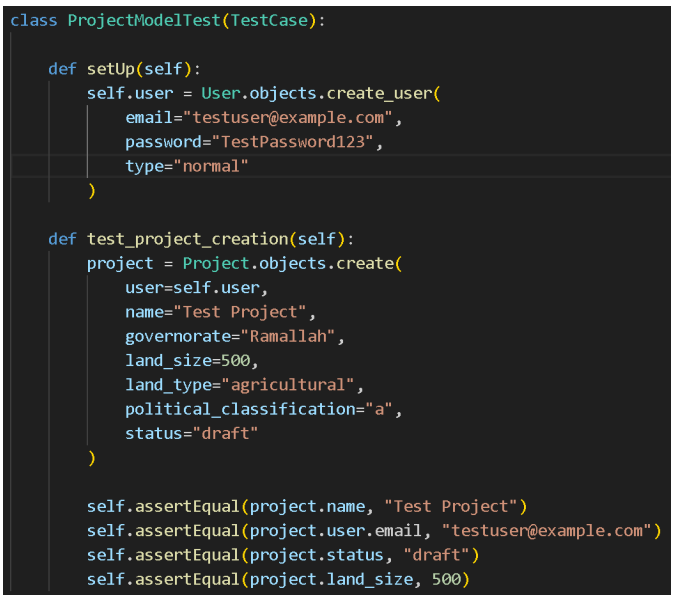
Unit testing was conducted to verify the correctness of individual system components in isolation, particularly the data model responsible for storing land project information. Django's built-in testing framework was used to ensure reliability, repeatability, and isolation from production data.

5.1.1 Project Model Unit Test

A unit test was implemented to validate the correct creation of a Project instance. This test focuses on confirming that essential attributes—such as the associated user, land size, governorate, and project status—are correctly stored in the database.

Figure 5.1.1 shows the unit test implementation within the tests.py file of the Normal_User_Side application. The test uses Django's TestCase class, which automatically provides a temporary test database and a clean environment for each test run.

Figure 5.1.1 Unit test implementation for the Project model



```
class ProjectModelTest(TestCase):

    def setUp(self):
        self.user = User.objects.create_user(
            email="testuser@example.com",
            password="TestPassword123",
            type="normal"
        )

    def test_project_creation(self):
        project = Project.objects.create(
            user=self.user,
            name="Test Project",
            governorate="Ramallah",
            land_size=500,
            land_type="agricultural",
            political_classification="a",
            status="draft"
        )

        self.assertEqual(project.name, "Test Project")
        self.assertEqual(project.user.email, "testuser@example.com")
        self.assertEqual(project.status, "draft")
        self.assertEqual(project.land_size, 500)
```

The test setup creates a sample user and a project instance, then applies assertions to verify that:

- The project is correctly linked to the authenticated user
- The project fields contain the expected values
- The default project status is assigned correctly

This approach ensures that the core data model behaves as intended before being integrated with views, forms, or machine learning components.

5.1.2 Test Execution and Results

The unit test was executed using Django's test runner. Figure 5.1.2 illustrates the successful execution output, where Django automatically creates a temporary test database, runs the test, and then destroys the database after completion.

Figure 5.1.2 Unit test result

```
(env) C:\Users\msama\OneDrive\Desktop\Land_Price_Estimator>python manage.py test Normal_User_Side
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.293s

OK
Destroying test database for alias 'default'...
```

The successful test result confirms that the Project model functions correctly in isolation and that the system's foundational data layer is stable.

5.2 Integration Testing

Integration testing ensures that different parts of the application work together correctly. For this project, the main focus was on the interaction between the project creation view, the ML model, and the database.

We created a simplified integration test in Normal_User_Side/tests.py that covers the following scenarios:

1. Creating a project as a draft.
2. Creating a project and performing ML price estimation.
3. Handling ML prediction failures gracefully.
4. Ensuring that users can only see their own projects in the project list.

5.2.1 Test Code

Figure 5.2.1 Project Integration Test

```python
class ProjectIntegrationTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(
            email="testuser@example.com",
            password="testpass123",
            type="normal_user"
        )
        self.client = Client()
        self.client.login(email="testuser@example.com", password="testpass123")

        self.project_data = {
            "name": "Test Project",
            "governorate": "ramallah",
            "land_type": "agricultural",
            "political_classification": "a",
            "description": "Test description",
            "land_size": 100
        }

    def test_create_project_draft(self):
        response = self.client.post(reverse("normal_user:new-project"), {**self.project_data, "action": "save"})
        self.assertEqual(response.status_code, 302, "Draft creation should redirect")

        project = Project.objects.get(name="Test Project")
        self.assertEqual(project.status, "draft", "Project status should be 'draft'")
        self.assertEqual(project.user, self.user, "Project should be linked to the logged-in user")

    @patch("Normal_User_Side.views.predict_land_price")
    def test_create_project_with_estimation(self, mock_predict):
        mock_predict.return_value = 50000
        response = self.client.post(reverse("normal_user:new-project"), {**self.project_data, "action": "estimate"})
        self.assertEqual(response.status_code, 302, "Estimation creation should redirect")

        project = Project.objects.get(name="Test Project")
        self.assertEqual(project.status, "completed", "Project status should be 'completed'")
        self.assertEqual(project.estimated_price, 50000, "Estimated price should match mocked ML value")

    @patch("Normal_User_Side.views.predict_land_price")
    def test_estimation_failure_handled(self, mock_predict):
        mock_predict.side_effect = Exception("ML error")
        response = self.client.post(reverse("normal_user:new-project"), {**self.project_data, "action": "estimate"})
        self.assertEqual(response.status_code, 302, "Redirect should occur even if ML fails")

        project = Project.objects.get(name="Test Project")
        self.assertEqual(project.status, "draft", "Project should fallback to 'draft' on ML failure")
```

5.2.2 Explanation

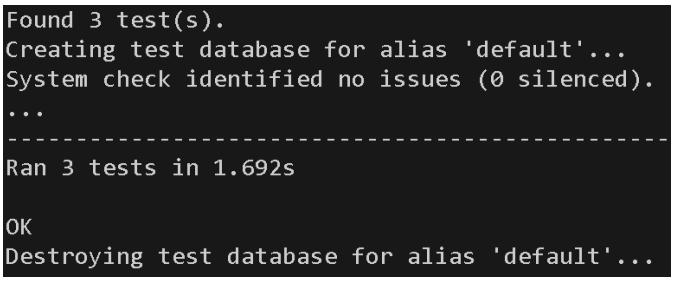
The test class ProjectIntegrationTest logs in a test user, submits project data to the new-project view, and checks the following:

- Whether the project is correctly saved as draft or completed.
- If the ML prediction is performed and the estimated price is saved.
- Whether errors in ML prediction are handled properly and do not break the flow.
- Whether the project list view only shows projects belonging to the logged-in user.

This ensures that the end-to-end workflow of project creation and listing works as intended.

5.2.3 Test Results

Figure 5.2.2 Integration Test Result



5.2.4 Explanation

The results confirm that all integration scenarios are working:

- Draft projects are saved correctly.
- ML estimation is integrated successfully.
- Error handling works as expected.
- User-based project filtering is functional.

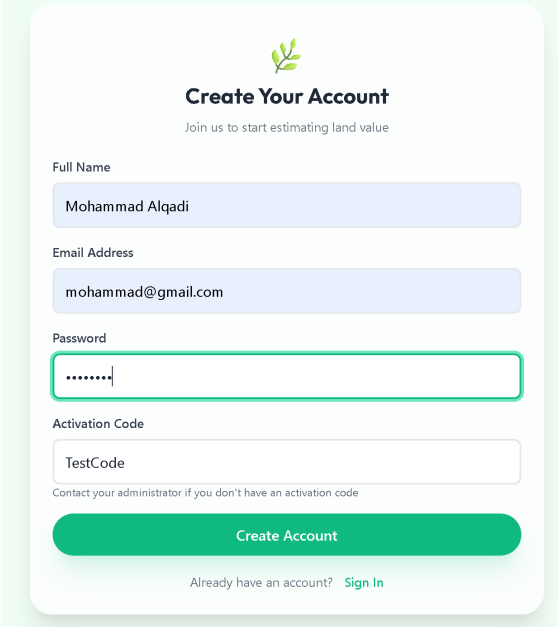
5.3 End-to-End (E2E) Testing

End-to-end testing evaluates the system as a complete unit by simulating real user interactions from initial access to final output. This testing approach ensures that all system layers—user interface, backend logic, authentication, database operations, and machine learning integration—work together seamlessly. In this project, end-to-end testing was conducted manually, reflecting realistic usage scenarios performed by a normal user (land appraiser).

5.3.1 User Registration and Authentication

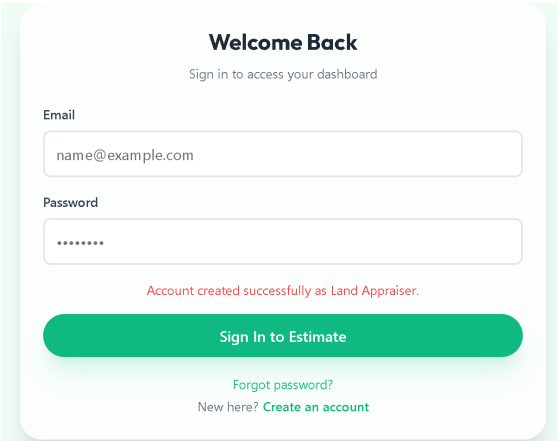
The testing process begins with the registration of a new normal user through the system's registration interface.

Figure 5.3.1 Registration



Explanation: The user creates an account by providing required information and upon successful registration, the user will be redirected to the login page with a success message and will be able to authenticate using the login interface.

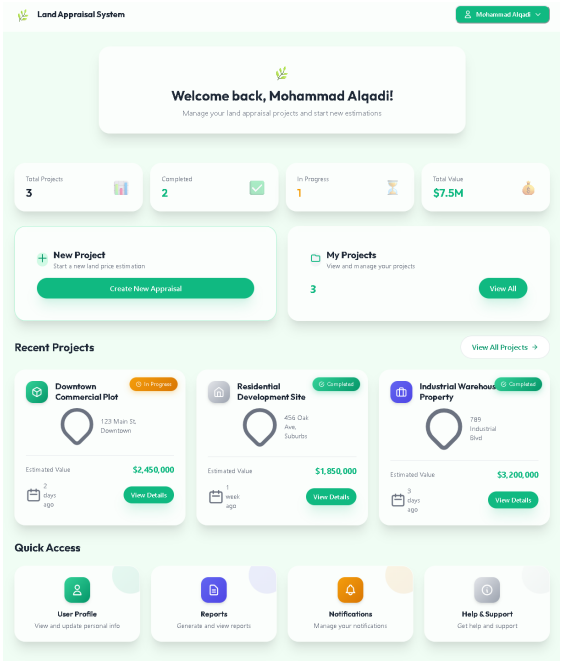
Figure 5.3.2 Registration successful



5.3.2 Dashboard Access and Navigation

Once authenticated, the user gains access to the main dashboard.

Figure 5.3.3 Dashboard

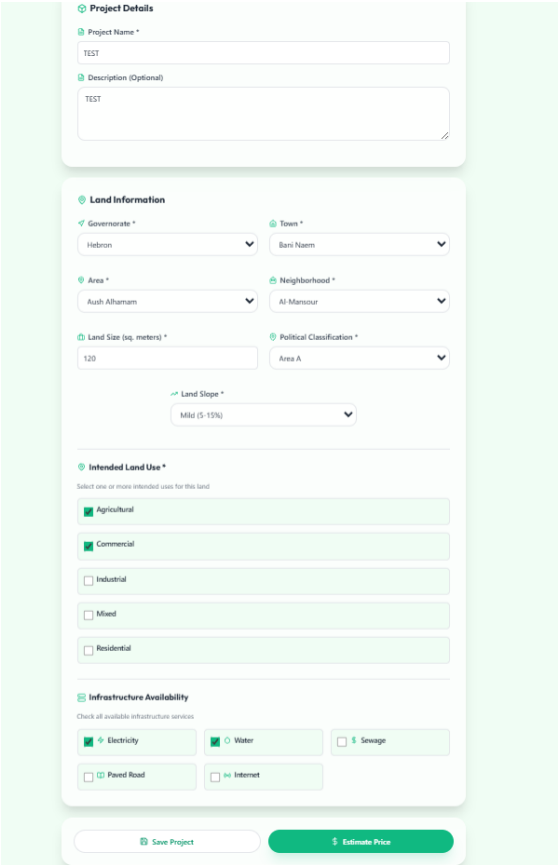


Explanation: The dashboard serves as the central interface for project management, allowing the user to create new projects or view previously created ones. This confirms correct session handling and access control.

5.3.3 New Project Creation

The user initiates the creation of a new land price estimation project by navigating to the “New Project” interface.

Figure 5.3.4 New Project form filled with data



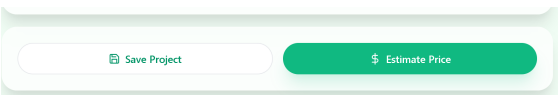
Explanation: The user fills in land-related attributes such as governorate, land size, land type, and political classification. These inputs represent the real-world data required for land price estimation.

5.3.4 Project Submission (Draft vs. Estimation)

At submission time, the user can choose between two actions:

- Save as Draft: Stores the project without running the machine learning model.
- Estimate Price: Triggers the machine learning prediction process.

Figure 5.3.5 Save and Estimate buttons

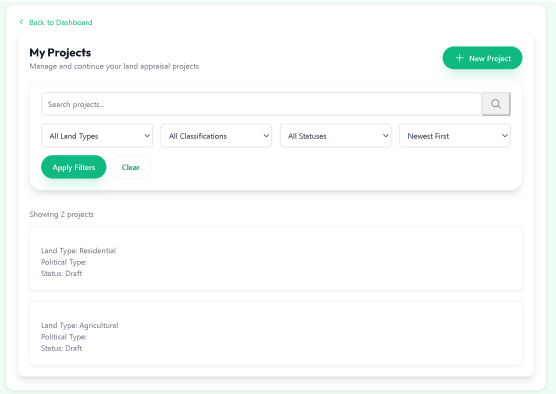


Explanation: This step validates conditional backend logic based on user action. Draft projects are stored with a “draft” status, while estimated projects are marked as “completed” after prediction.

5.3.5 Project Viewing and Verification

After submission, the user is redirected to the projects list page.

Figure 5.3.6 View the created project in the list of projects



Explanation: The projects page displays all projects created by the authenticated user. This confirms correct database persistence, user-based data filtering, and real-time dashboard updates.

Summary: The successful completion of these end-to-end scenarios confirms that the system satisfies its functional and user requirements. All critical user actions—from account creation and authentication to land price estimation and project review—behaved as expected, thereby validating the system from an acceptance testing perspective.

References

[1] J. Starmer, "Regression Trees, Clearly Explained!!!," *StatQuest with Josh Starmer*, YouTube, 2020. [Online]. Available: https://youtu.be/g9c66TUylZ4?si=0V35RgYavNRFB-yx. Accessed: Jul. 3, 2025.

[2] J. Starmer, "CatBoost Part 1: Ordered Target Encoding," *StatQuest with Josh Starmer*, YouTube, 2023. [Online]. Available: https://youtu.be/KXOTSkPL2X4?si=Iq890z0lxjhSlImH. Accessed: Jul. 3, 2025.

[3] J. Starmer, "CatBoost Part 2: Building and Using Trees," *StatQuest with Josh Starmer*, YouTube, 2023. [Online]. Available: https://youtu.be/3Bg2XRFOTzg?si=SUU2vVzNxbofFelA. Accessed: Jul. 3, 2025.