QuantEstate

Eddy Boris, Anish Guntreddi, Diego Herrera

The Problem

- Of the important things needed in modern life, a home is one of the biggest financial decisions you can make.
- The issue is that with the current economy and available houses, it may be hard to find a proper house within your budget that gives you the best return on investment.
- This, along with the fact that there are a wide variety of houses at various places, it can be hard to find the best area to buy a house as well

Concept

- QuantEstate is a web-based software application that uses machine learning and potentially deep learning to conduct quantitative analysis on the housing market and uses these results to determine what the best possible options for buying houses are within a certain area given your budget and other factors such as credit score, interest rate, etc.
- This software application will use Python for the machine learning/deep learning side and the MERN stack (Mongo Express React Node) for the front end side.
- This will particularly help the general public with making good financial investments in the properties they want to buy and could potentially save them a lot of time

Functional Requirements

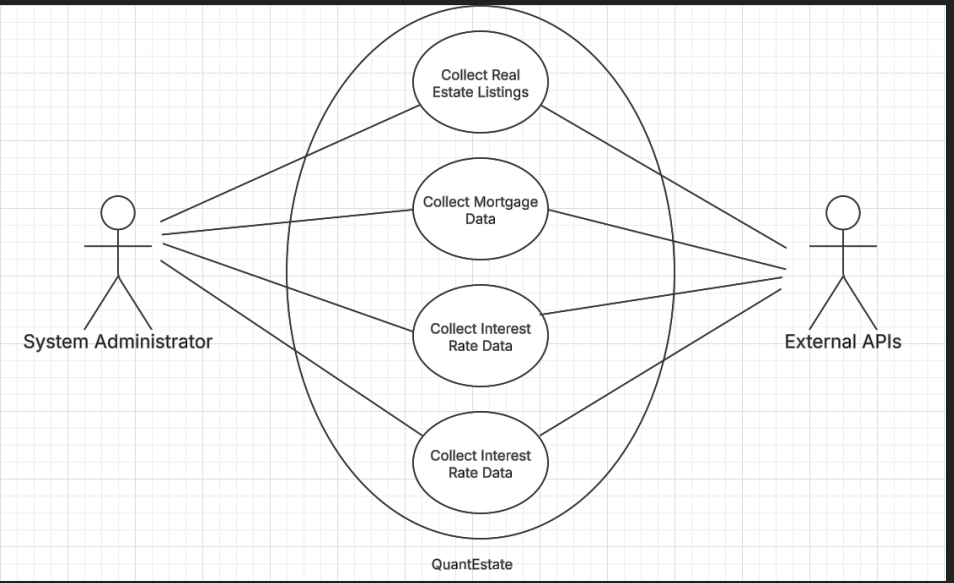
- User Interface with ReactJS
- Allows for user to sign in with auth provided by jwt and uses MongoDB as the database (will later move to the cloud for larger scaling).
- Allows for users to input where they live and what their expected budget is to purchase a house so our application can return a list of houses that they feel are the best for those given conditions
- Allows for users to create their own account and set their own preferences for the recommendation system.

Non-Functional requirements

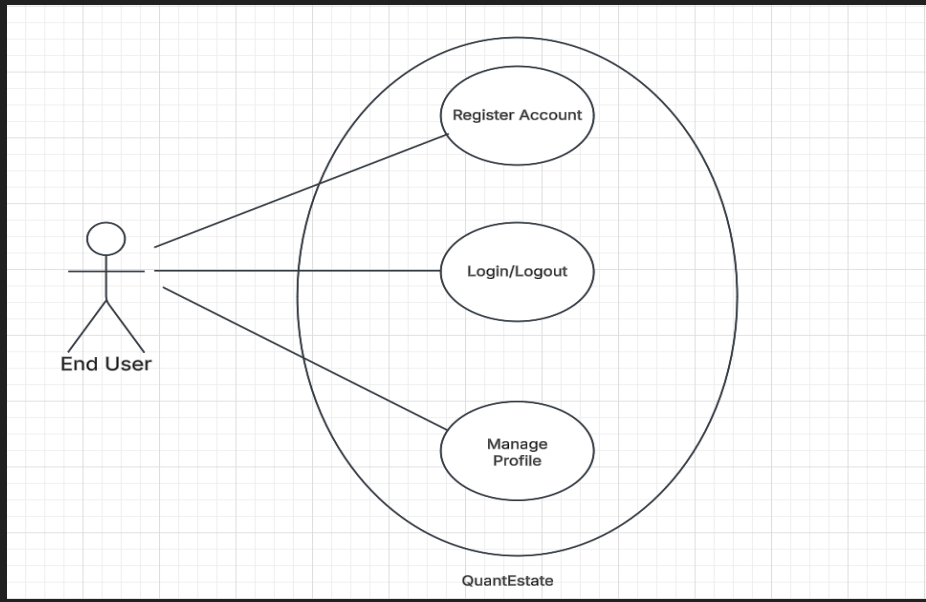
- The system needs to be able to perform reliably by being able to handle the data collection, training, front-end operations, etc. It needs to be consistent when generating reports and results along with user authentication.
- Ensuring Security and Privacy with user credentials and data by using methods such as 2 Factor authentication and using hashing algorithms such as bcrpyt and jwt (json web token) for added security
- Usability and accessibility of the front-end. The front end needs to be interactive and intuitive for user
- Making the app secure for users in the future (as mentioned earlier, we can transition from using MongoDB to cloud computing in the future and potentially getting better cybersecurity)

Use Case Diagram Analysis

Data Collection and Integration

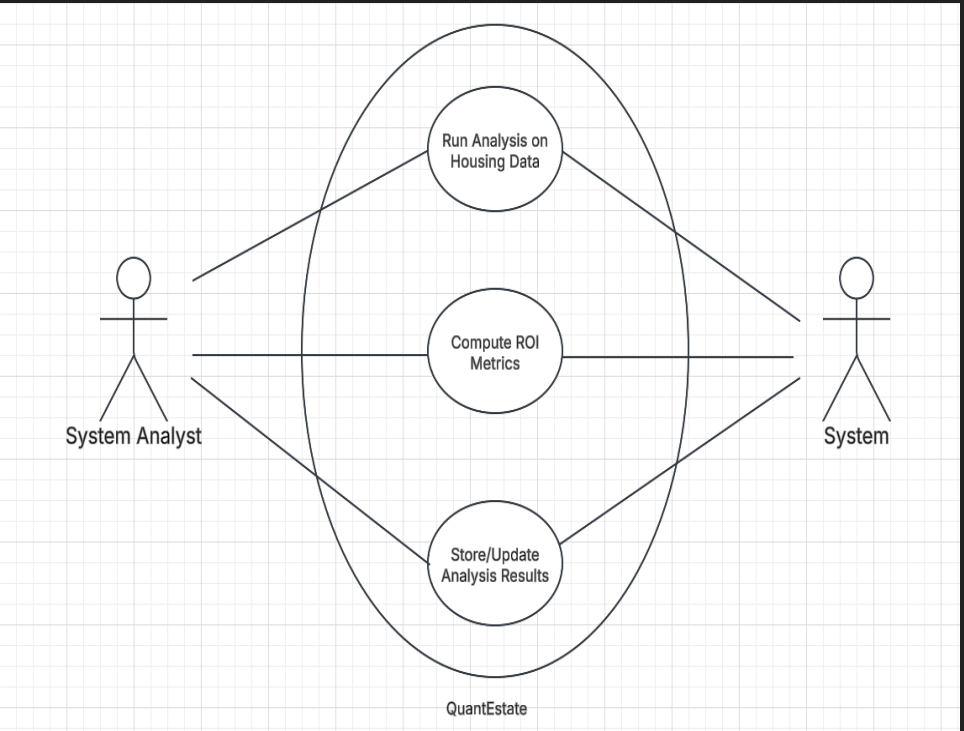


User Account Management

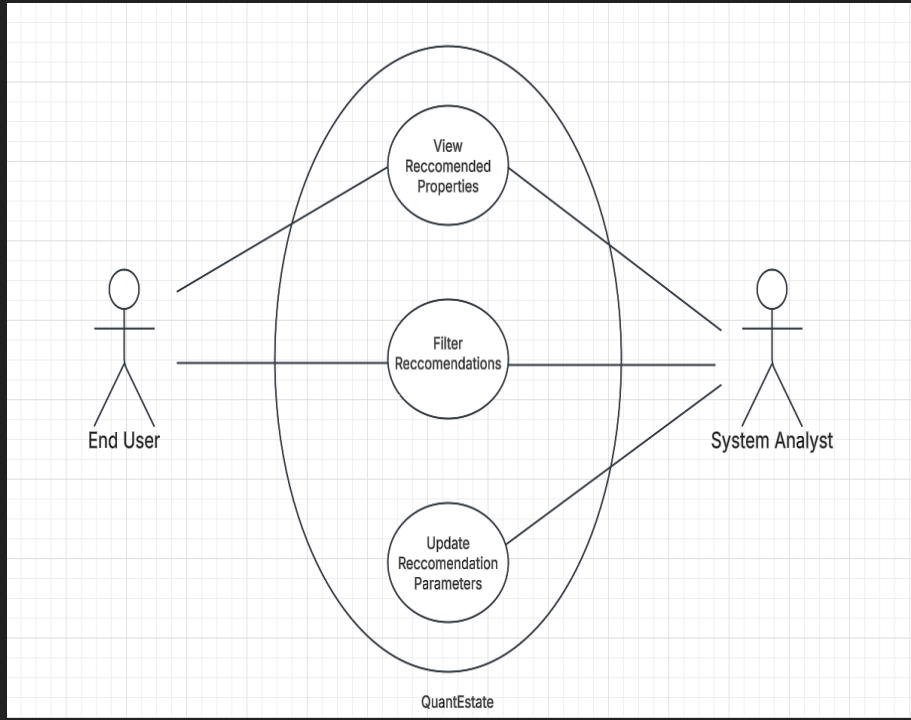


Use Case Diagram Analysis Part 2

Data Analysis and ROI



Recommendation Engine



System Constraints

- Python Virtual Environment for development of ML or deep learning model.
- We need to use Javascript as our language for the development of front-end applications.
- The application must run on web browsers since it is a web application
- For CPU and GPU constraints, a minimum of 8 GB of RAM, a CPU with 4 cores, and sufficient hard drive for storage is recommended. High-end GPU can also be used for accelerated development of ML model
- Application requires high-speed, reliable internet connection to be able to load between pages on the front end.
- The application should also be ready to deploy to a cloud environment
- The project requires good documentation and version control.
- Application must comply with legal requirements

Evolutionary Functional Requirements

- Incorporating Advanced Predictive Analytics
  - Using More advanced forms of ML and incorporating deep learning models for more accurate ROI predictions and potentially incorporating a larger historical data set
  - This can result in more robust predictions and accurate predictions for housing prices.
  - This must also require more computing power and hardware in order to be able to train larger models more efficiently
- Incorporating an additional feature of this for other assets as well such as purchasing land, cars, etc.
  - Can become a multipurpose-based tool with different kinds of assets to assist people in large financial decisions/investments.

Evolutionary Non-Functional Requirements

- Using more secure services for encryption and protection of data
  - Potentially moving over to cloud computing for more storage and faster operations while being more reliable in handling user traffic and also fetching data.
- Incorporating support for multiple languages for larger demographics (more of a minor component)