QUANTESTATE

AI Investing Advisor

TABLE OF CONTENT

- System Overview
- System Diagram
- Description of Actors
- Architecture Style
- Design Patterns
- Frameworks
- Functional Design
- Structural Design

MEET THE TEAM

Anish Gutreddi: BS Comptuer Science, Data Science and AI, Mathematics

Eddy Boris: B.S. Computer Science - Comprehensive Track

Diego Herrera: BS Comptuer Science - Comprehensive Track

SYSTEM OVERVIEW

QuantEstate is a web-based platform that helps investors and homebuyers find high-ROI real estate opportunities. It pulls data from property listings, mortgage providers, and interest rates, then uses machine learning to forecast ROI and recommend properties based on user preferences.

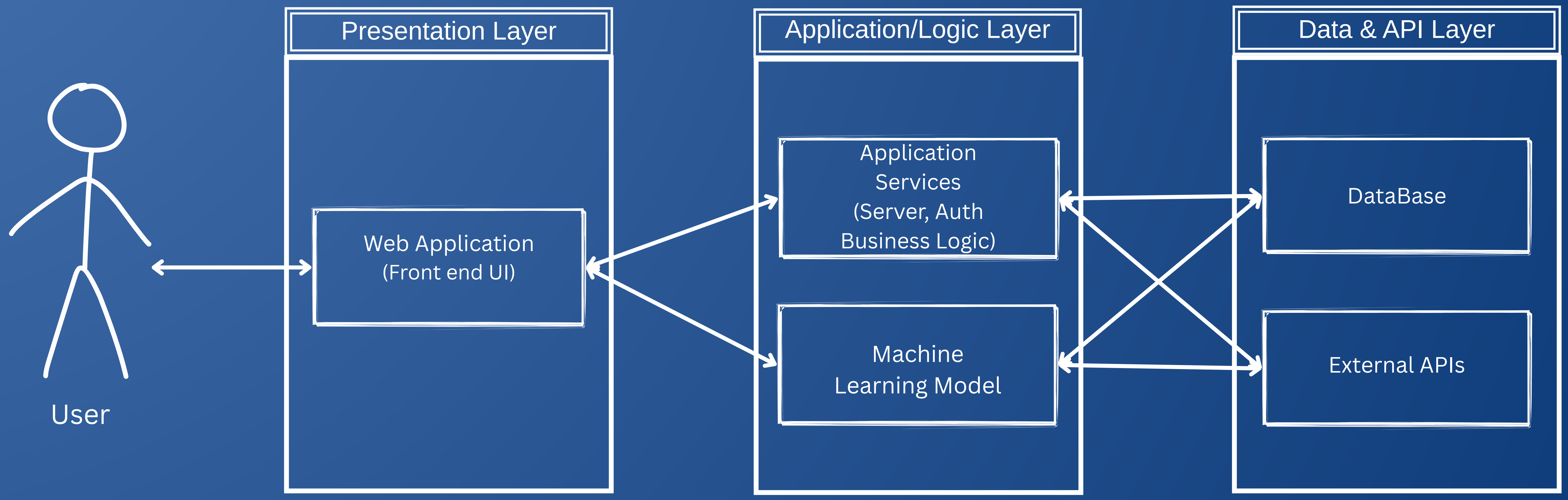
Core Features:

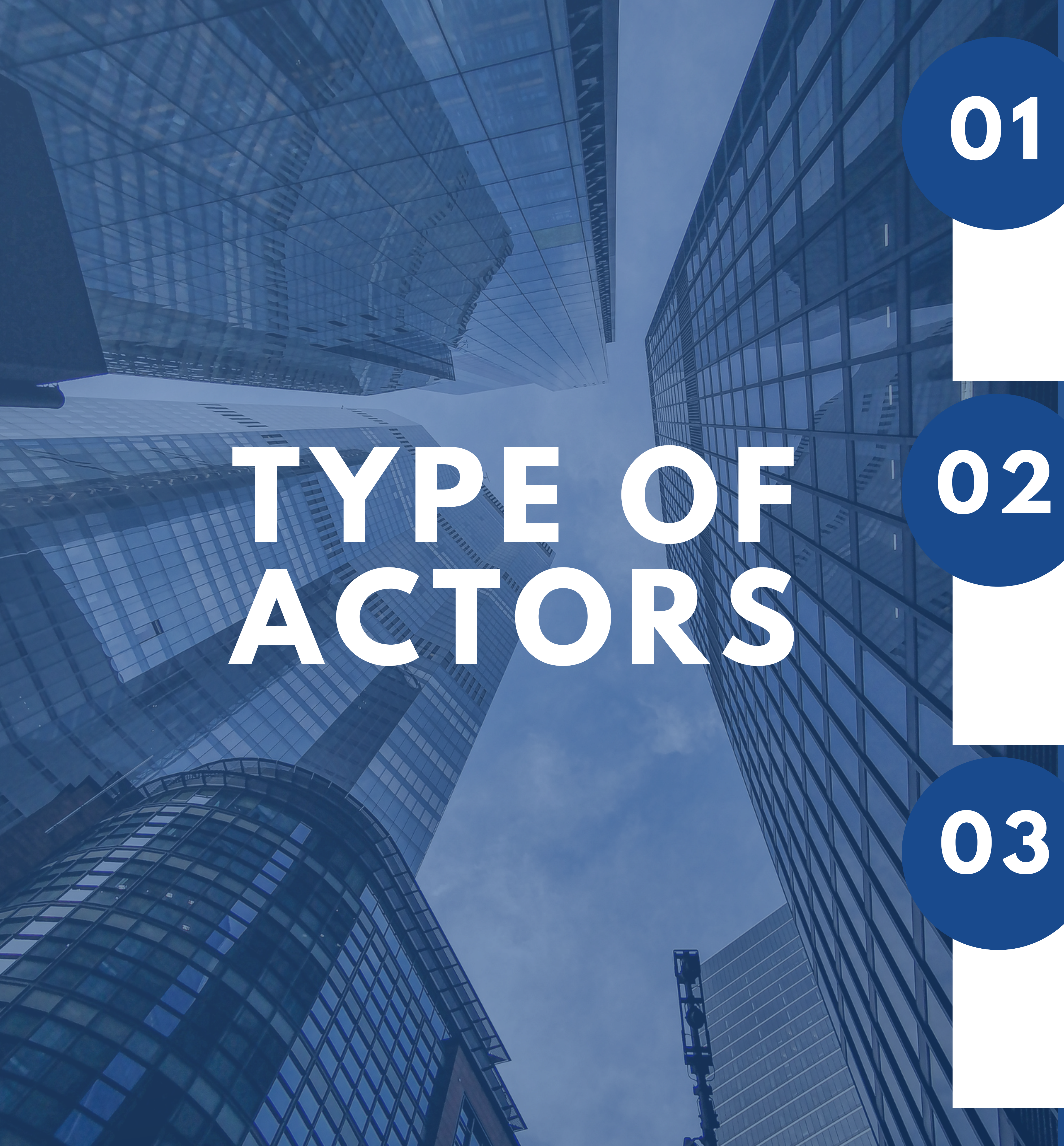
- User account management (sign-up, login, profile updates)
- Integration with external APIs (e.g., Zillow, mortgage data)
- Data cleaning and transformation (Python-based)
- ROI prediction using ML/statistical models
- Personalized property recommendations

System Architecture: **QuantEstate** uses a three-tier architecture for scalability and maintainability:

1. Front-End: User interface
2. Logic Layer: Business logic (Node.js/Python)
3. Data Layer: Databases and third-party APIs

SYSTEM DIAGRAM





TYPE OF ACTORS

01 NON-REGISTERED USERS

These visitors explore the QuantEstate platform without creating an account. Their experience is limited to viewing public information—basic property listings and general platform features. While they can begin the registration process, they don't have access to personalized recommendations or ROI metrics.

02 REGISTERED USERS

Once signed in, registered users unlock the full capabilities of QuantEstate. They can browse property listings, apply filters like budget and location, and view detailed ROI predictions. With the ability to save properties, receive market alerts, and manage their profiles, these users are equipped to make data-driven real estate decisions.

03 SYSTEM ADMINISTRATOR

The System Administrator manages the platform's inner workings. They ensure machine learning models are kept up-to-date, handle database migrations, and oversee technical maintenance. Their role is essential for keeping the system stable, accurate, and continuously evolving.

ARCHITECTURAL STYLE

Modularity

QuantEstate's three-tier architecture separates concerns into display, logic, and data layers.
Each tier handles its own job, making it easier to maintain, debug, and scale parts of the system independently.

Flexibility

Changes in the front end won't mess with the back end—and vice versa.
You can upgrade APIs, swap models, or redesign the UI without disrupting the whole flow.

Scalability

As usage grows, you can spin up more upscale database servers to handle incoming traffic.
Databases can scale horizontally or vertically to meet performance needs.

DESIGN PATTERNS

Model View Controller	At the back-end level, the Node.js application loosely follows the MVC pattern. We have models to represent data entities such as Users or Properties, Controllers to handle incoming HTTP requests, translating them into service calls and returning appropriate responses and views that are not strictly defined in the back-end since the presentation is handled by React/Next.js on the front end. However, the concept remains valuable for structuring controllers and model logic in separate modules.
Factory Pattern	As data acquisitions can involve creating new property objects or user objects systematically, a Factory Pattern may be used to simplify the logic for constructing these objects based on external data. This pattern provides a consistent interface for generating domain-specific objects that represent the information from Zillow or other APIs, simplifying the code needed to handle diverse incoming data formats.
Singleton Pattern	Database connections or application-level configuration can often follow the Singleton Pattern, ensuring that only a single instance of a connection pool or configuration manager is in use throughout the system. This helps manage resources efficiently and avoid synchronization issues that might arise if multiple connections attempt to modify the same shared data concurrently.

FRAMEWORKS

NextJS

This framework is used in the front end to create a dynamic, responsive web interface. Next.js enables server-side rendering to improve performance and SEO (search enginer optimization) while the react frameowrk delivers a modular component model that facilitates maintainable UI development.

NodeJS

1. This environment provides a lightweight, flexible setup for developing RESTful APIs or GraphQL endpoints. Express routing enables clear definitions of endpoints for user management, property listing retrieval, and ROI computations.

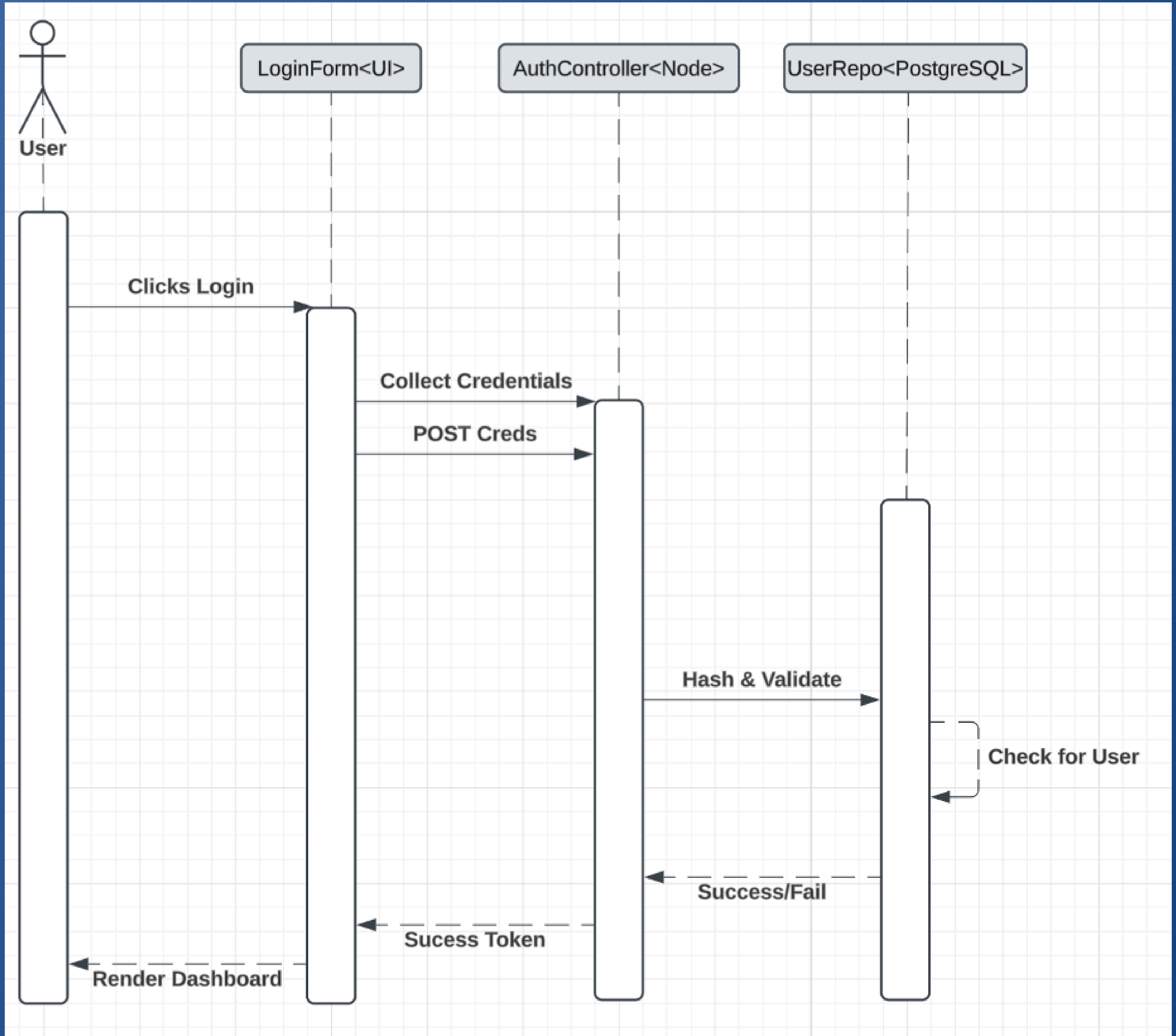
Python (with pandas, scikit-learn, or TensorFlow)

Chosen for data ingestion, cleaning, and machine learning tasks. Python's data science foundation ensures fast development for ROI computation and advanced analytics, which can be exposed to Node.js through a microservice architecture or simple script calls.

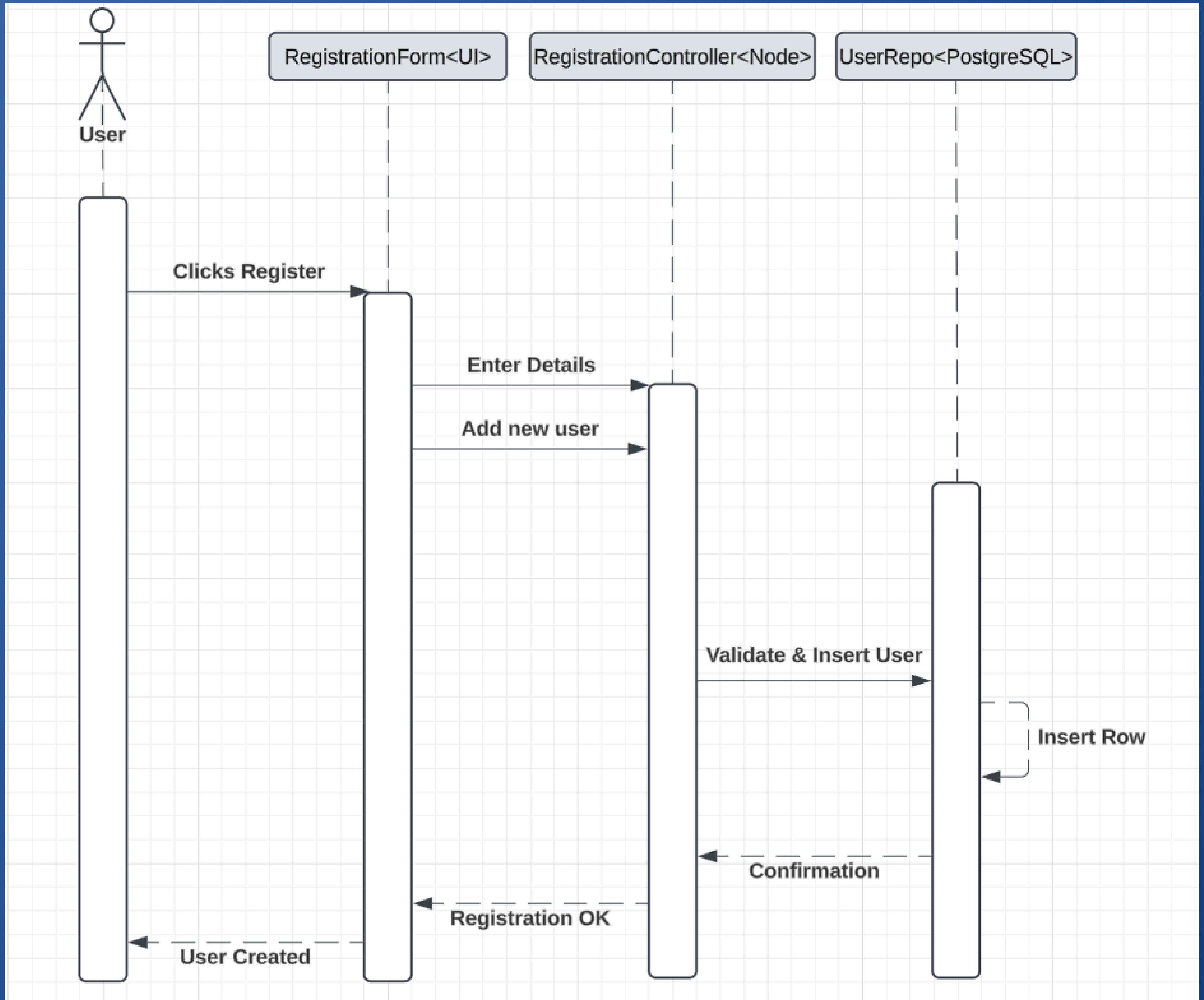
PostgreSQL (MongoDB)

An open-source relational database, PostgreSQL is both robust and flexible, making it suitable for storing user information, property records, and results of ROI calculations. SQL's maturity and reliability make it an ideal candidate in a real-world setting.

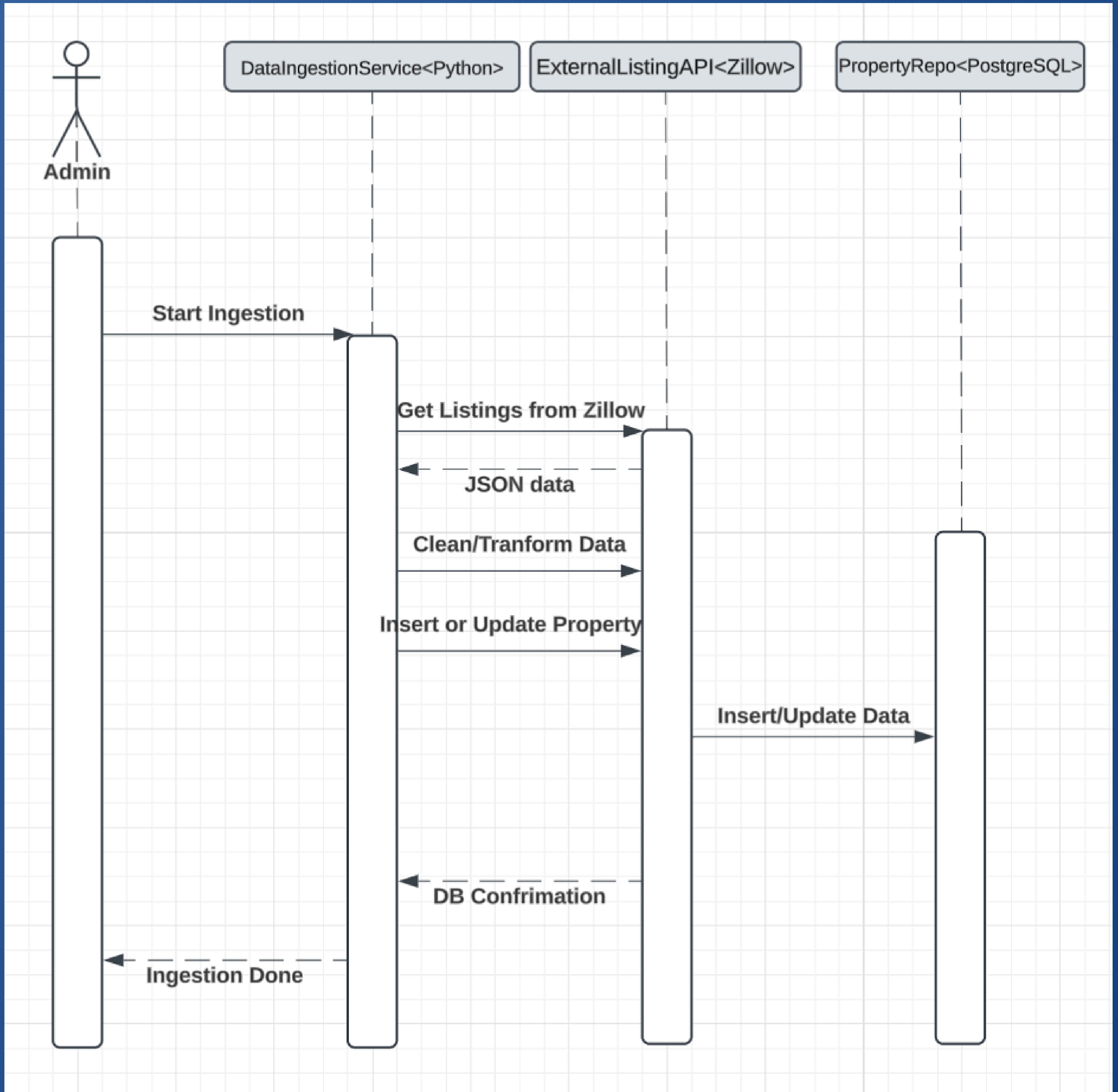
QUANTESTATE LOGIN



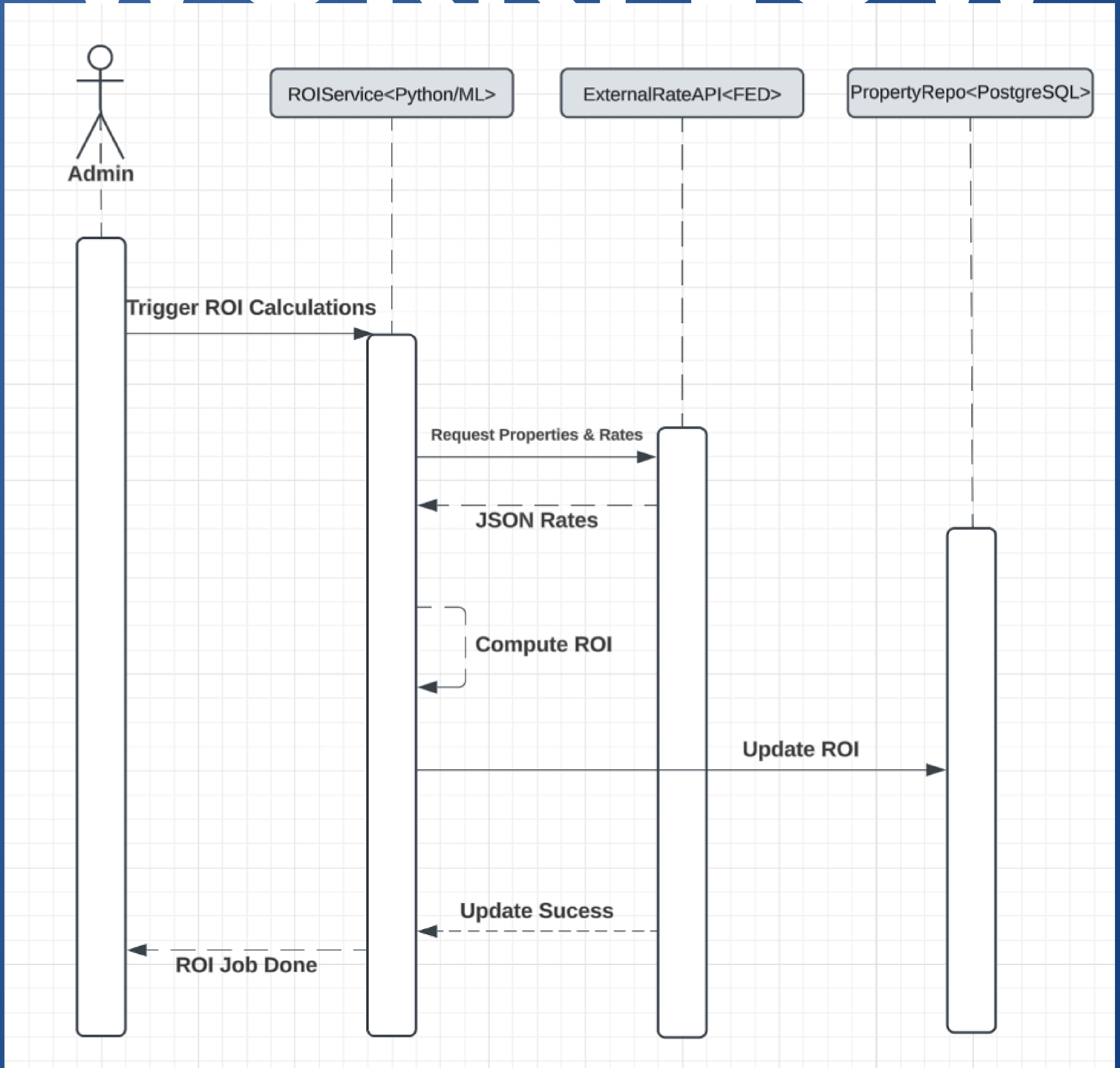
QUANTESTATE REGISTER



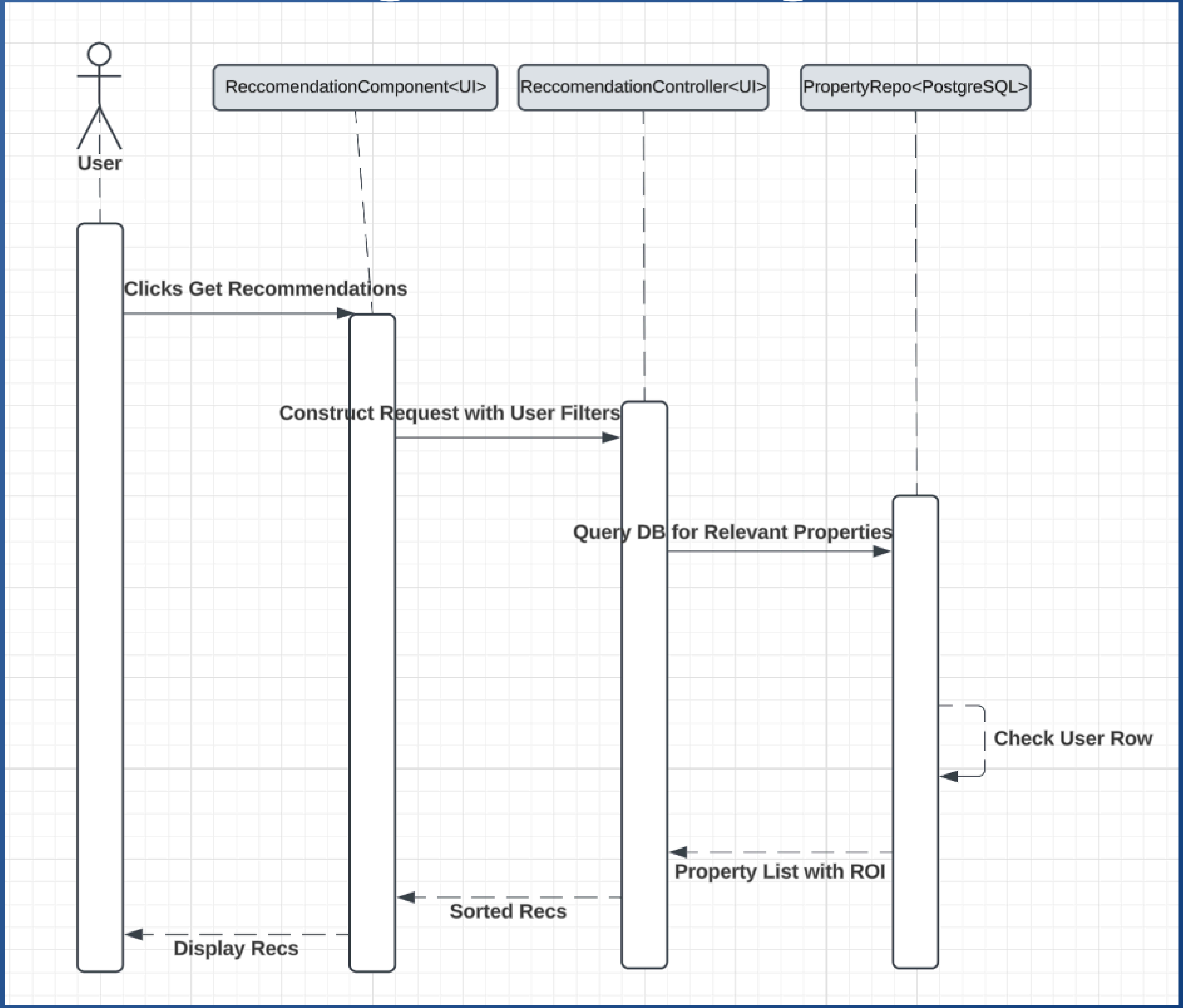
DATA COLLECTION WORKFLOW



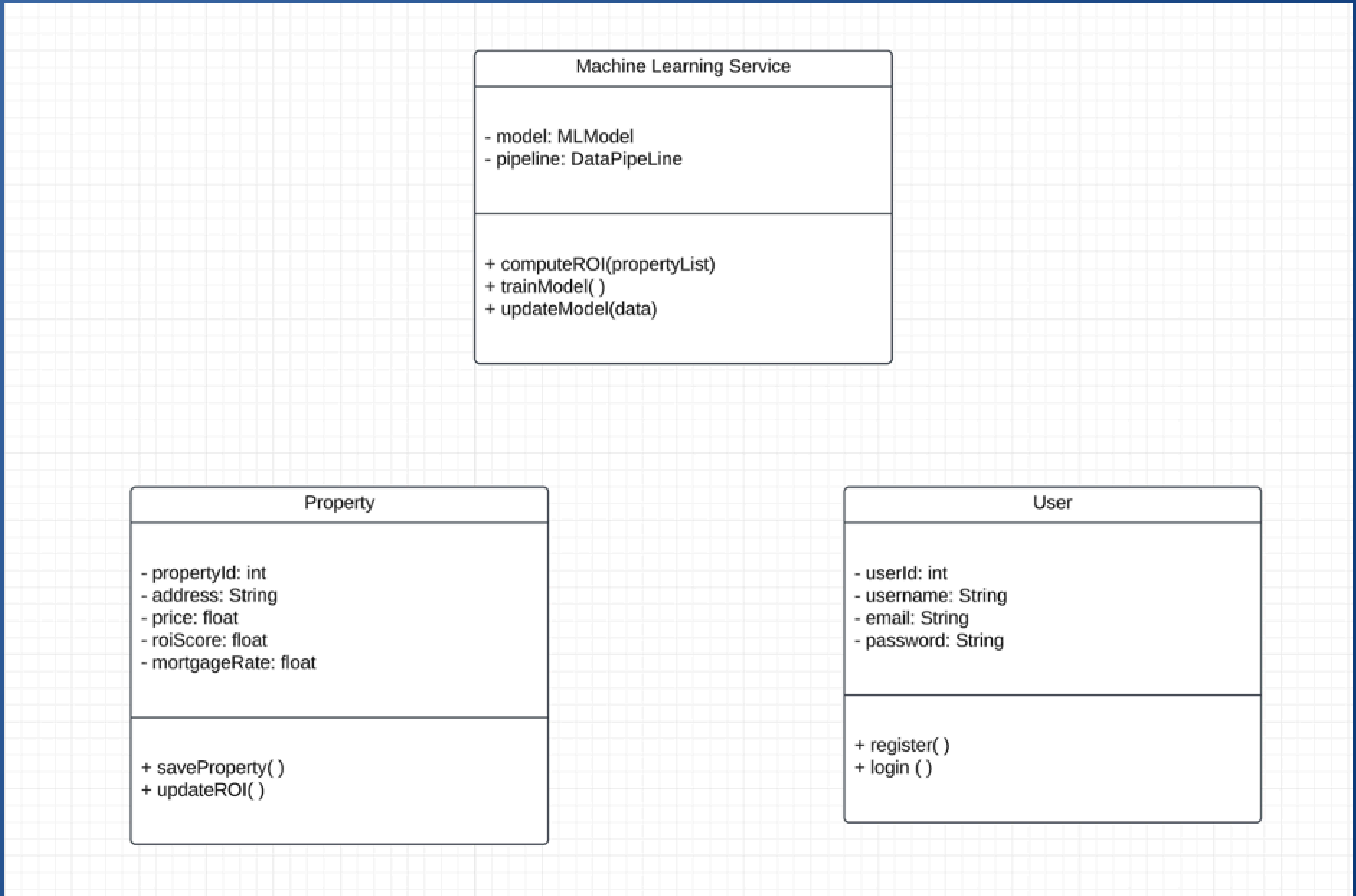
ROI COMPUTATION WORKFLOW



PROPERTY RECOMMENDATION WORKFLOW


User
ReccomendationComponent<UI>
ReccomendationController<UI>
PropertyRepo<PostgreSQL>
Clicks Get Recommendations
Construct Request with User Filters
Query DB for Relevant Properties
Check User Row
Property List with ROI
Sorted Recs
Display Recs

STRUCTURAL DESIGN



THANK YOU