**Asset Manager: Vincenzo’s final project**

This project is a lightweight asset management platform designed with privacy and ease-of-use in mind. It allows users to quickly register using a randomly generated identifier, ensuring anonymity while they track their investments in various asset classes—be it stocks, cryptocurrencies, or fiat currencies. Users can log transactions such as buying stocks at a specified price, and later, the system will dynamically compare these against live market values via an external API. Initially built as a small, focused project to validate core functionalities, the design intentionally minimizes registration friction and data collection. This approach not only enhances user privacy but also accelerates adoption, while laying a scalable foundation for future expansion and feature enhancements.

**Part A:**

**Database creation**

Why We Use Filess.io for Our Database Hosting

Simplified Management:

Filess.io provides a user-friendly interface and manages many of the routine database administration tasks (such as backups, scaling, and security) for you. This allows our team to focus on building the application and core functionalities rather than being bogged down by infrastructure management.

Scalability:

While our project is small right now, Filess.io’s cloud-based platform is built to scale. As we grow from a simple asset tracker to a larger system with more users and features, it’s easy to upgrade our resources without major disruptions.

High Availability & Security:

Filess.io takes care of high availability and data security. This means our data is reliably accessible and well-protected, which is crucial even for a project focused on minimal registration and transaction logging.

Speed of Deployment:

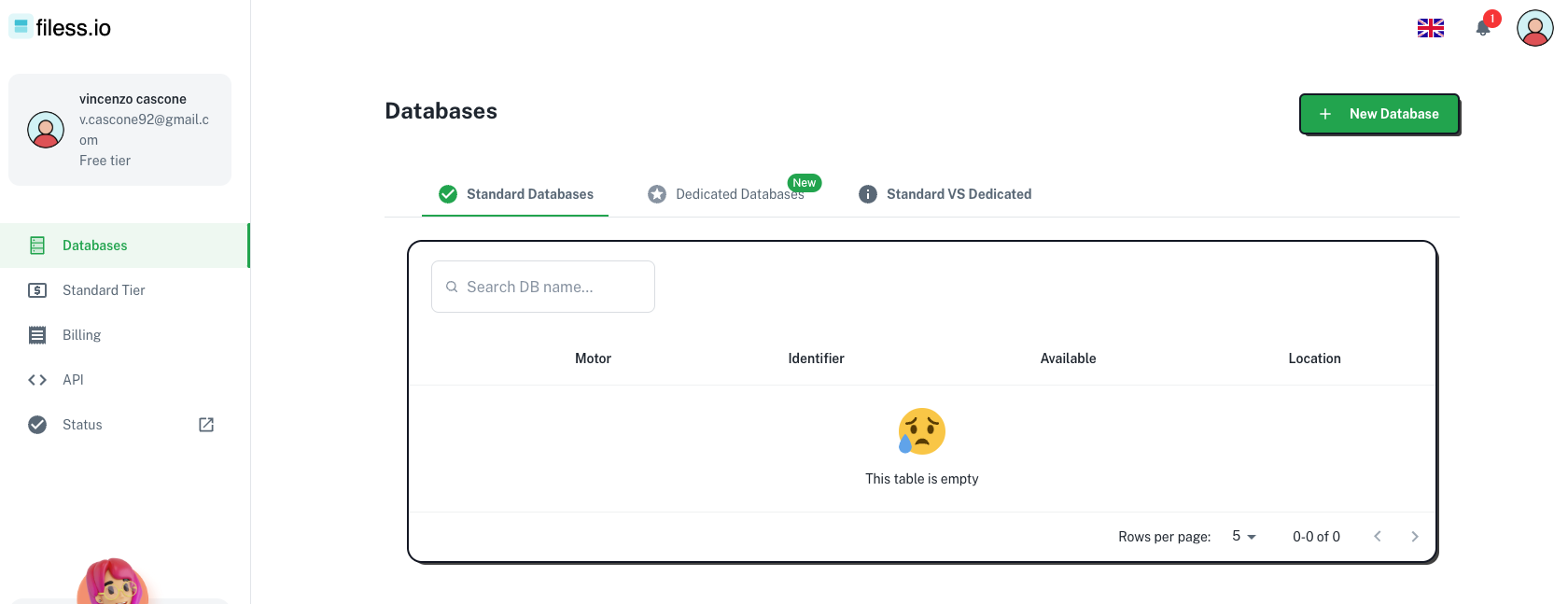
The platform allows for quick deployments. Since our registration system is minimal (using just a randomly generated user ID for anonymity), we don’t need a complex infrastructure. Filess.io perfectly fits the lightweight nature of our project and helps us iterate rapidly.

Cost-Efficiency:

For small projects or early prototypes, it’s essential to keep costs low while still having a robust system. Filess.io offers competitive pricing plans that suit projects in the early stages while providing room to grow.

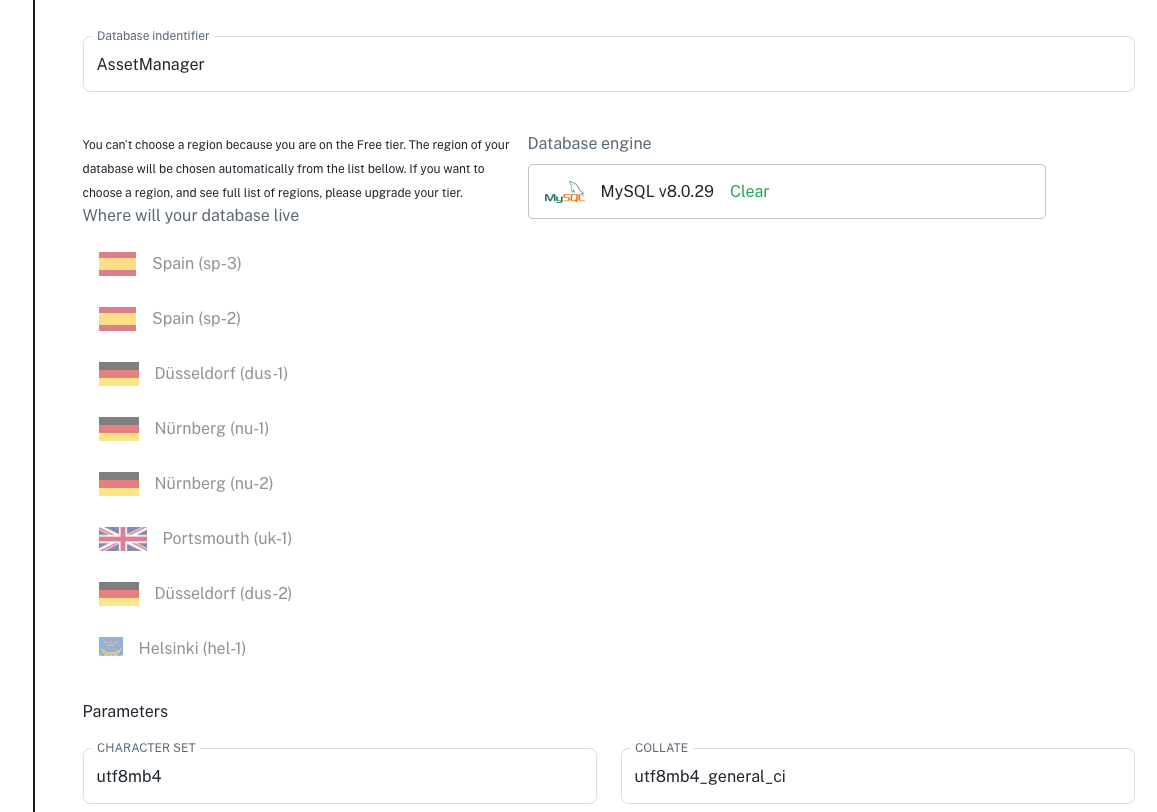
Step1:

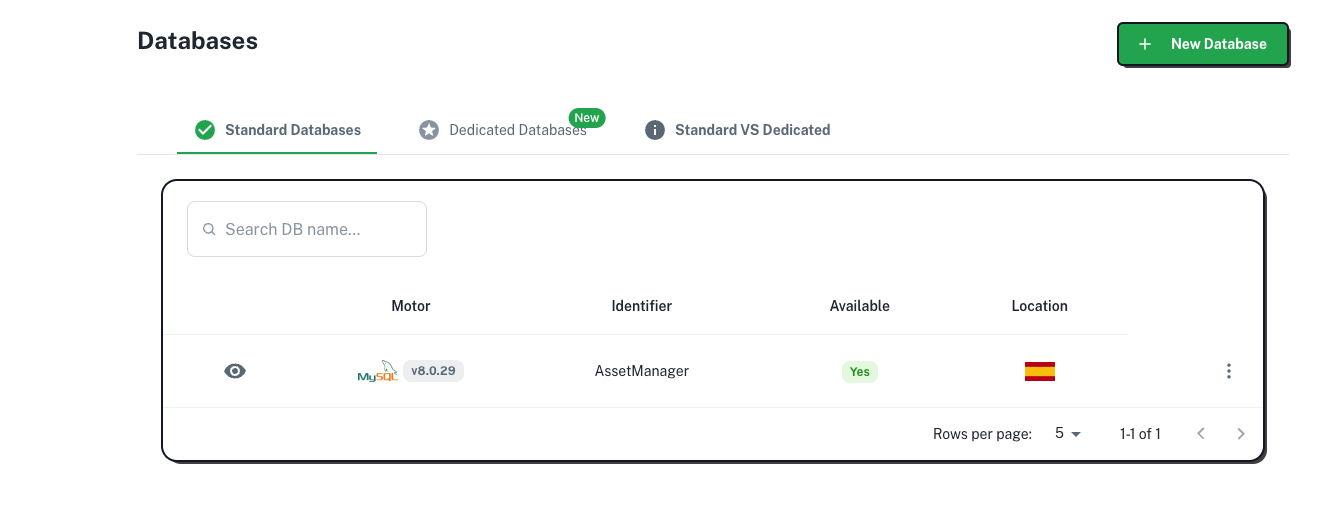
1.Locate the Database Section:

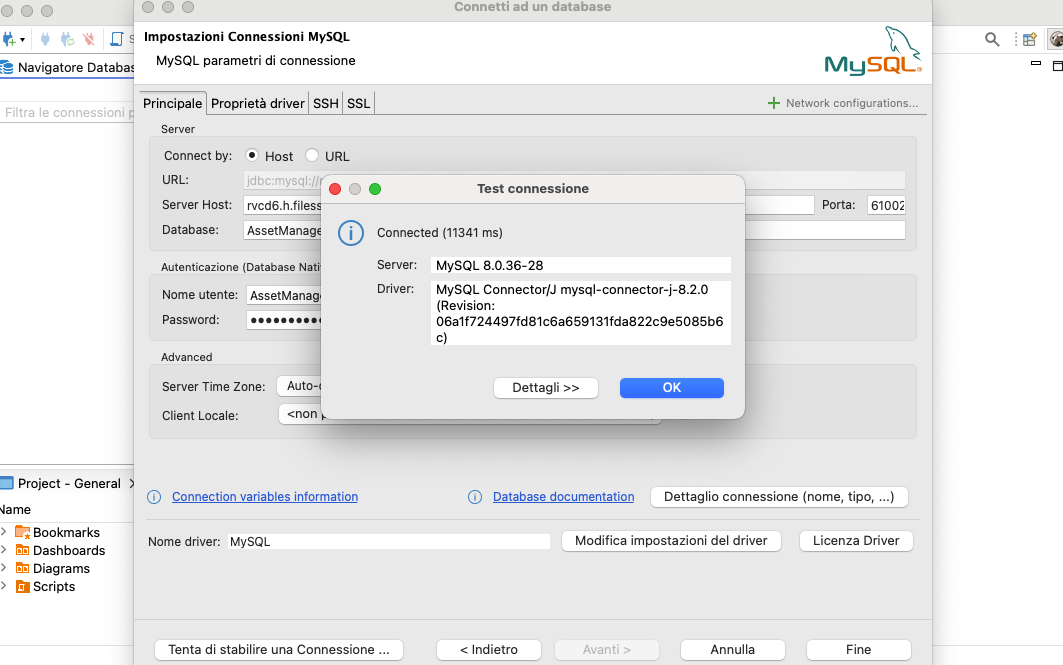


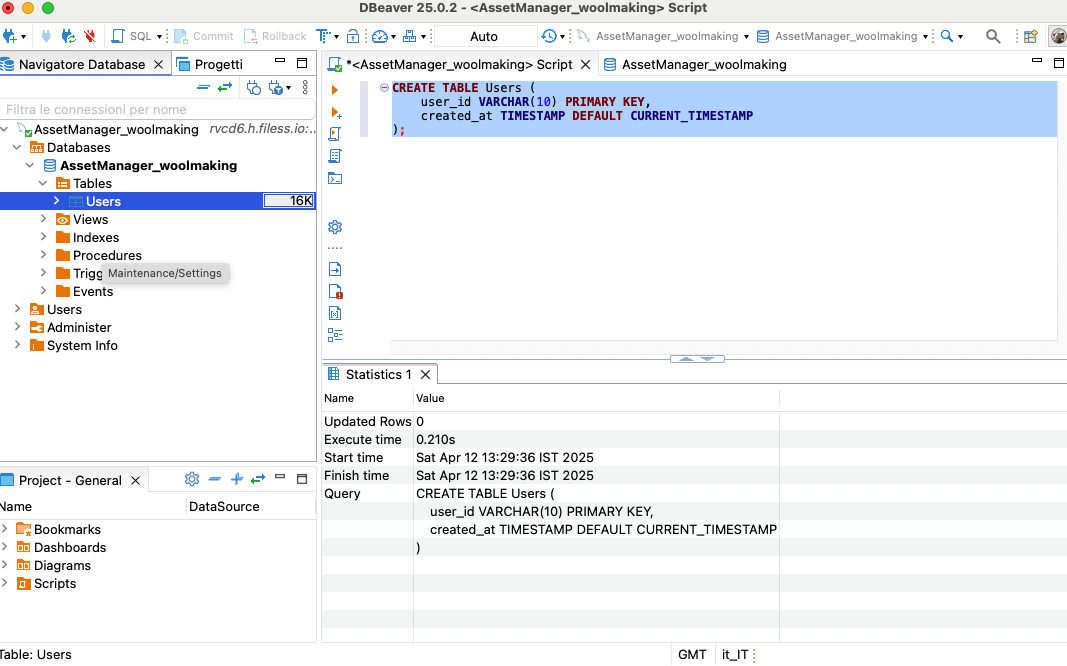
**2. Type of Database used: SQL**

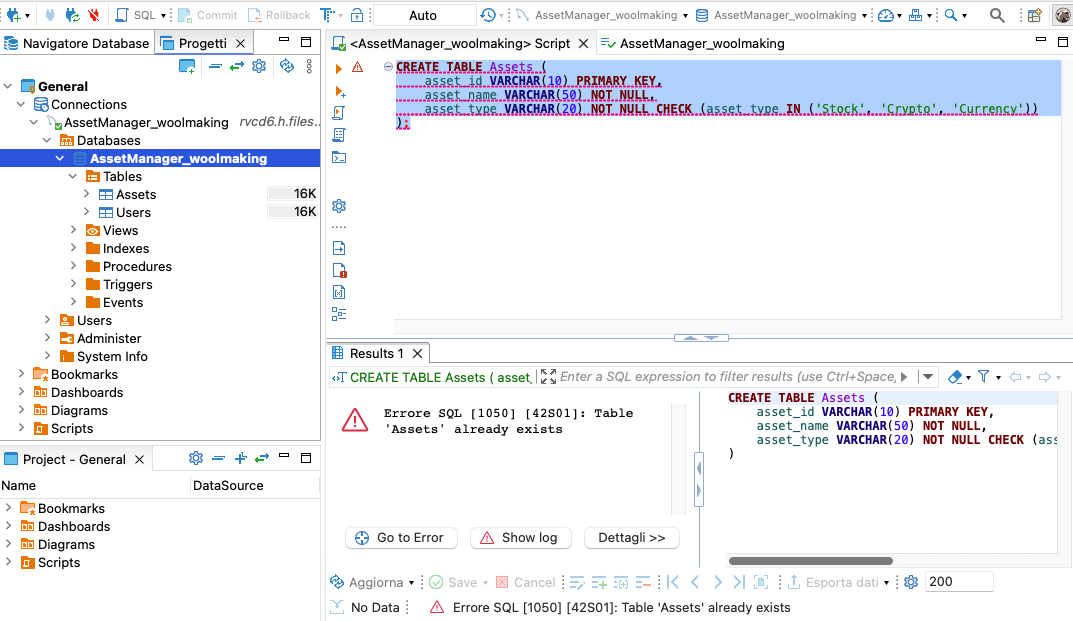
**3.Database Identifier: AssetManager**

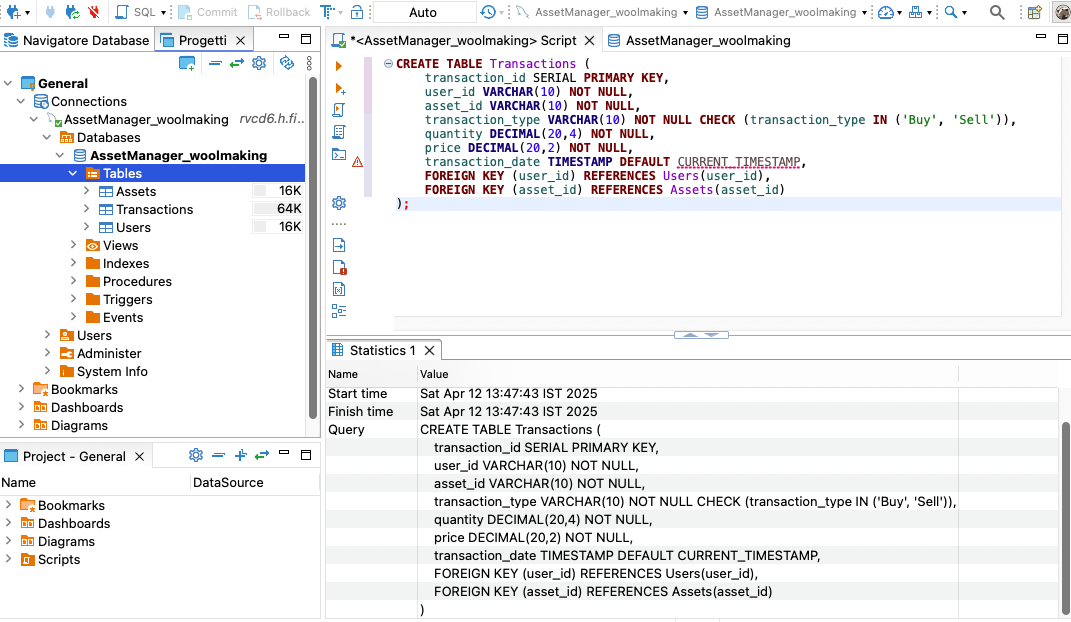


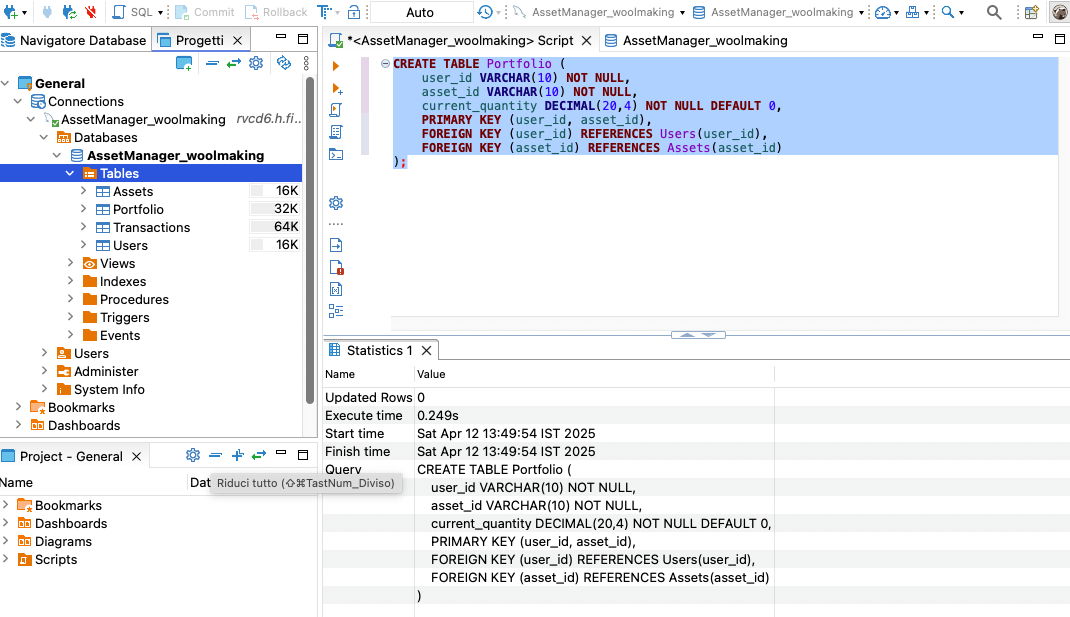




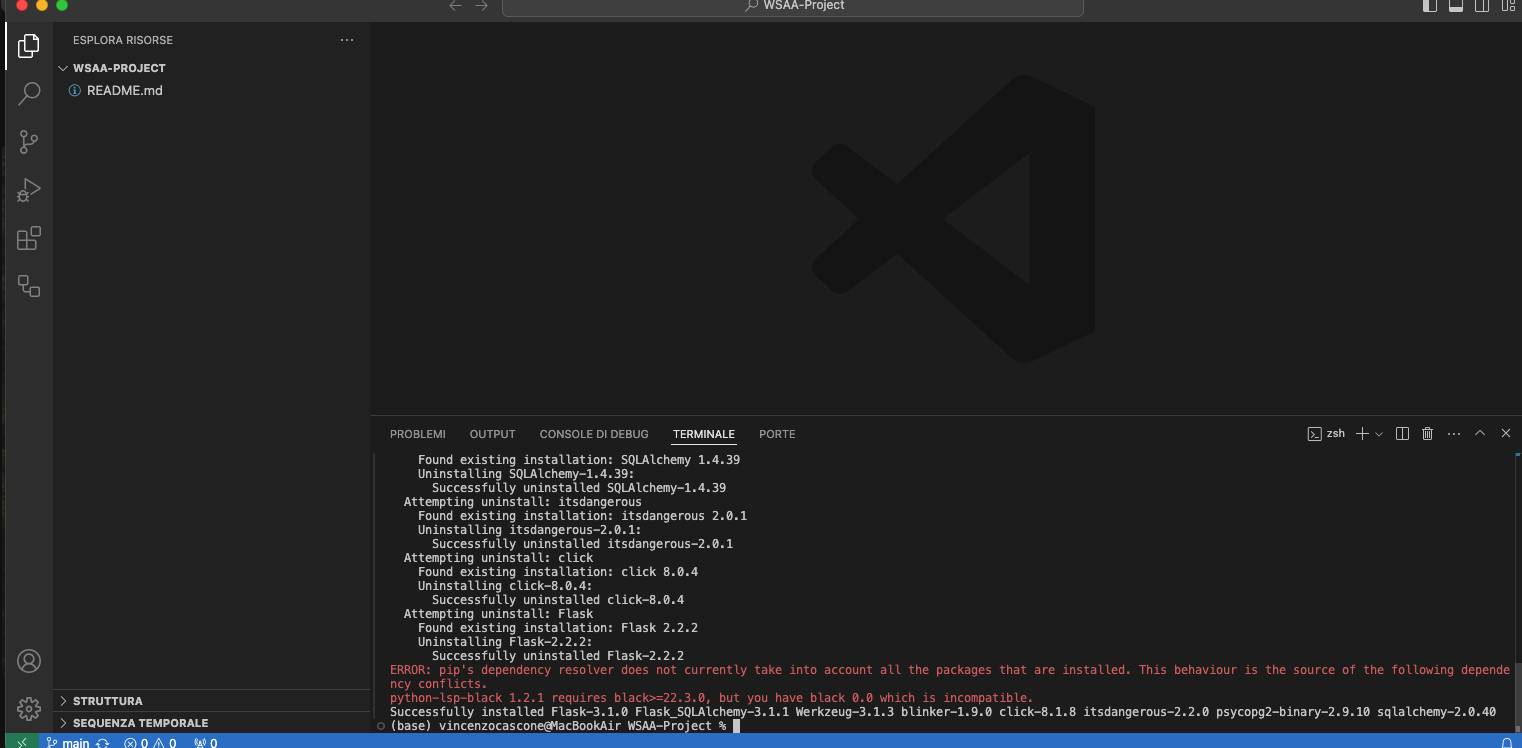








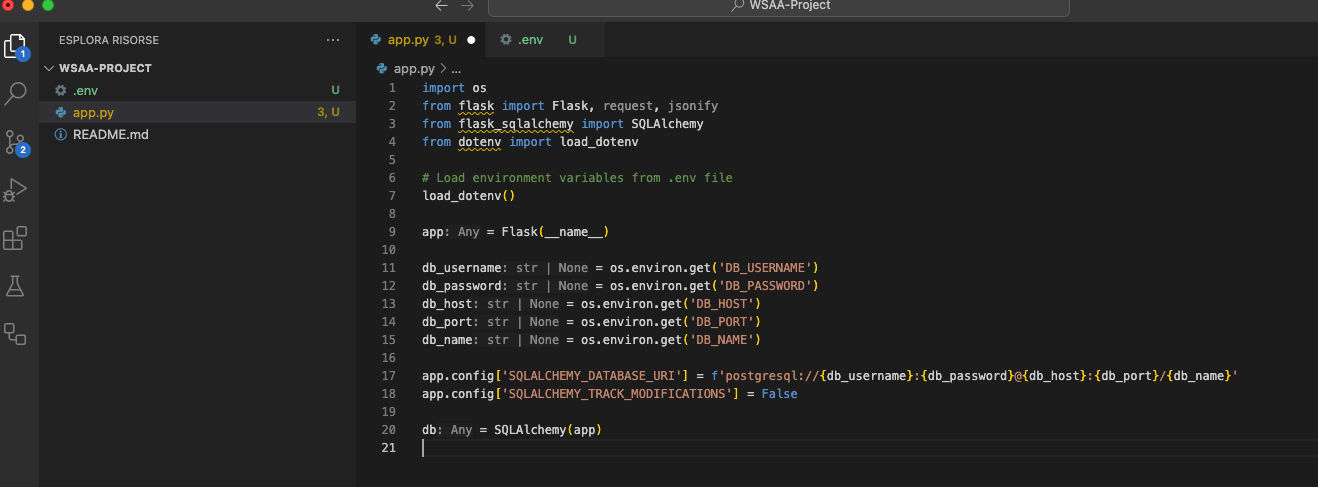
**Part two: creation of Python API**



pip install Flask Flask\_SQLAlchemy psycopg2-binary

Created a .env file to securely store the keys of my database.

Then I have imported the libraries to connect to the SQL database.

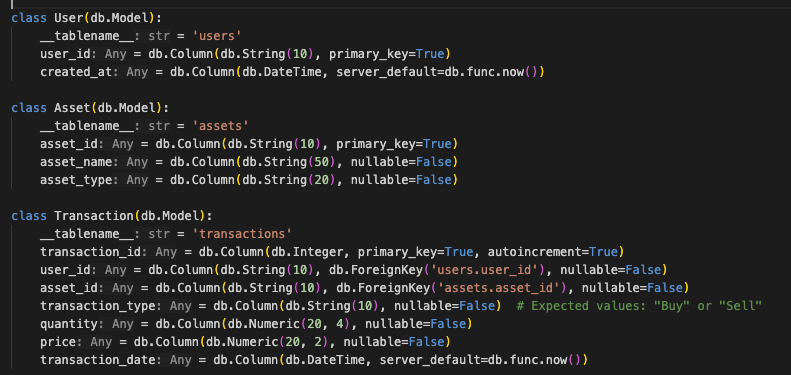


User Model: Represents users by their unique IDs and the time they registered.

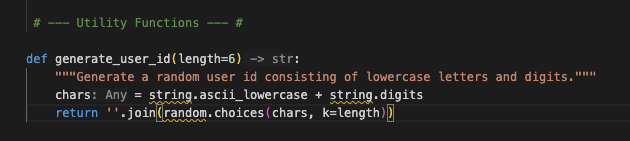
Asset Model: Represents the assets (stocks, currencies, etc.) available for transactions, each with a unique ID, name, and type.

Transaction Model: Represents each buy or sell operation, storing details such as which user performed the transaction, which asset was transacted, the type of transaction, how many units were transacted, the price, and the date of the transaction.

These models collectively form the backbone of the application's database, allowing you to manage user registrations, asset listings, and transaction logging efficiently while enforcing data integrity through primary and foreign key constraints.

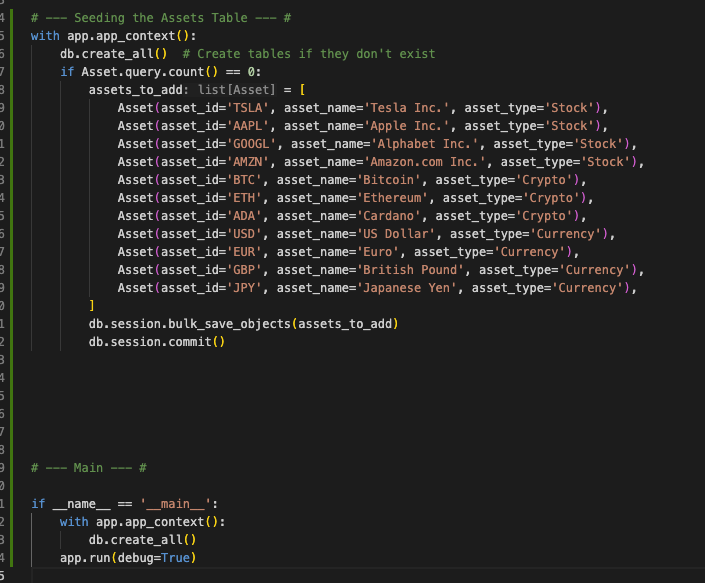


Added the utility functions:

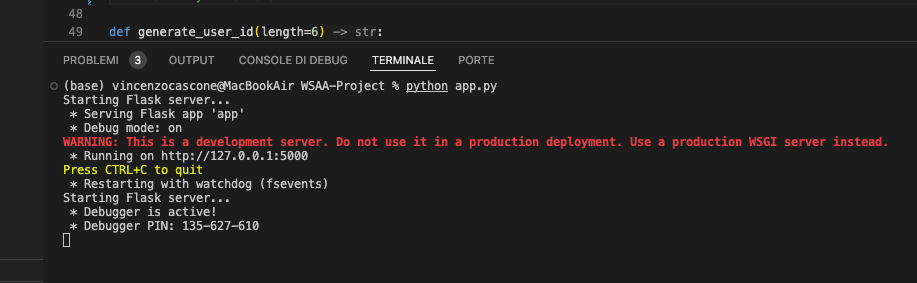


This function is a concise way to generate a random and anonymous user ID from a set of 36 characters, offering over 2 billion unique combinations (36⁶ possibilities). Although collisions (duplicate IDs) are extremely unlikely, the surrounding registration logic ensures that if a collision occurs, a new ID will be generated.

Running API and creating sample assets:

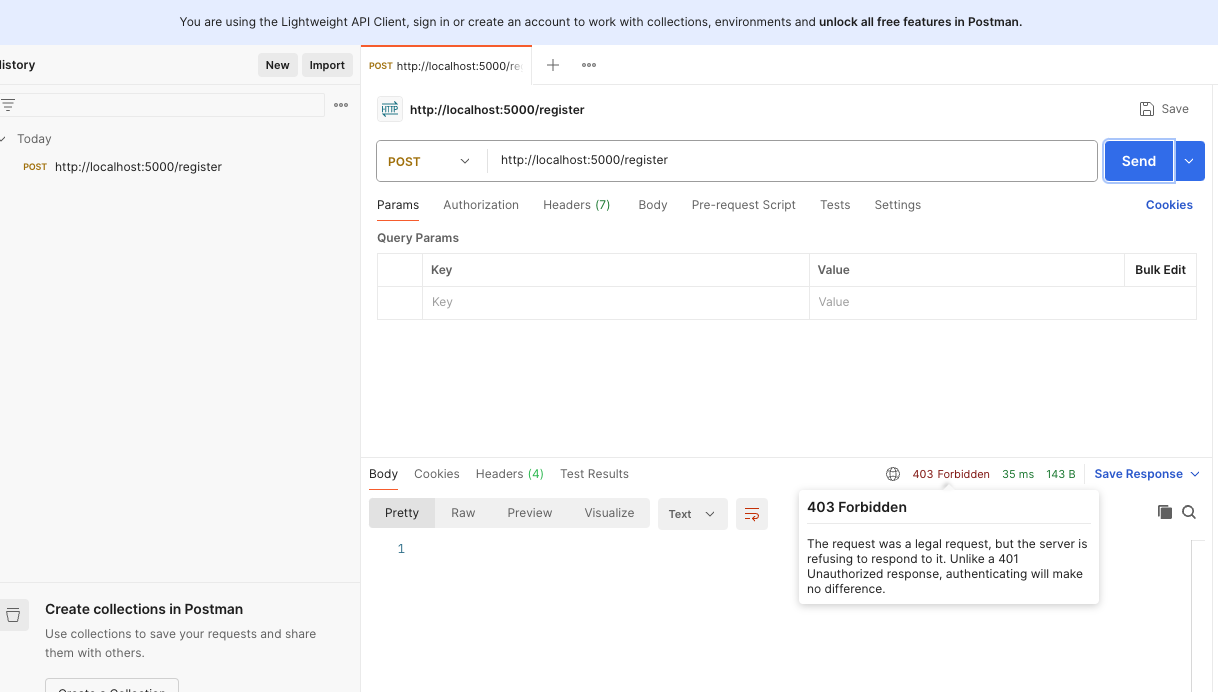


API running correctly after errors debugging:

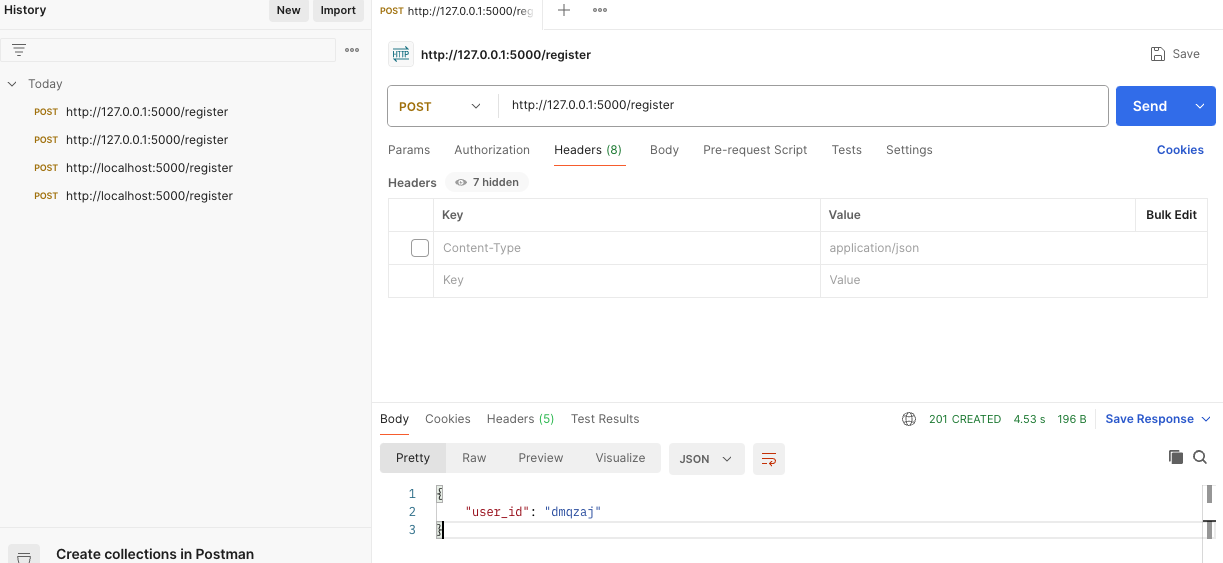


Testing the API

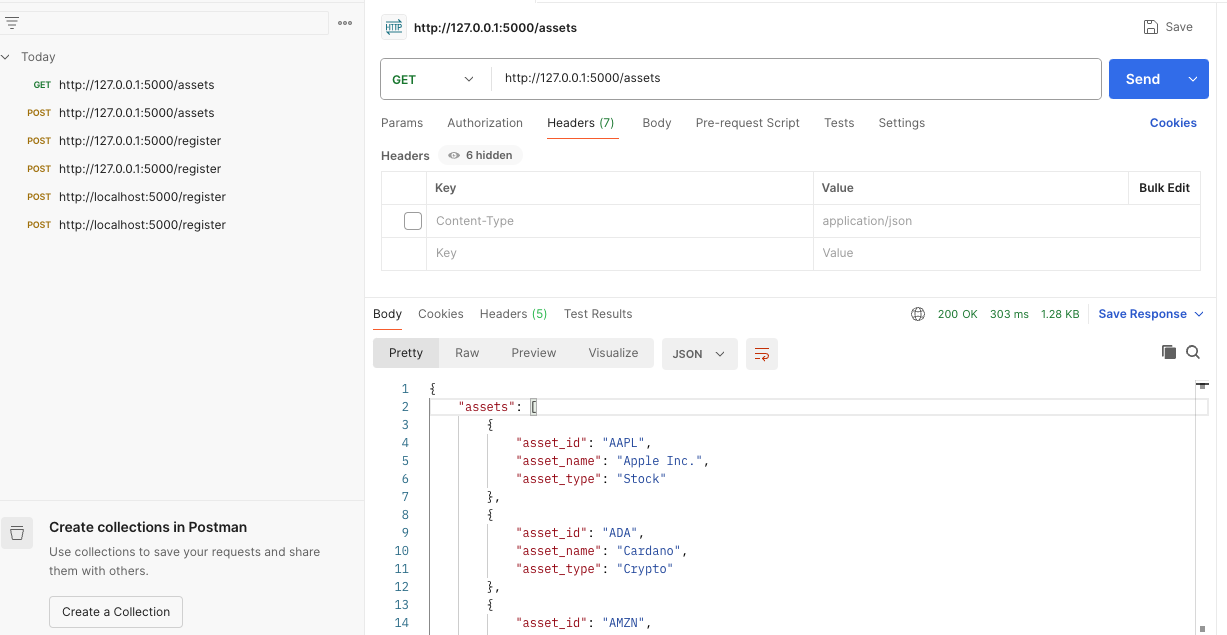
Downloaded Postman to test the API, first attempt failed:



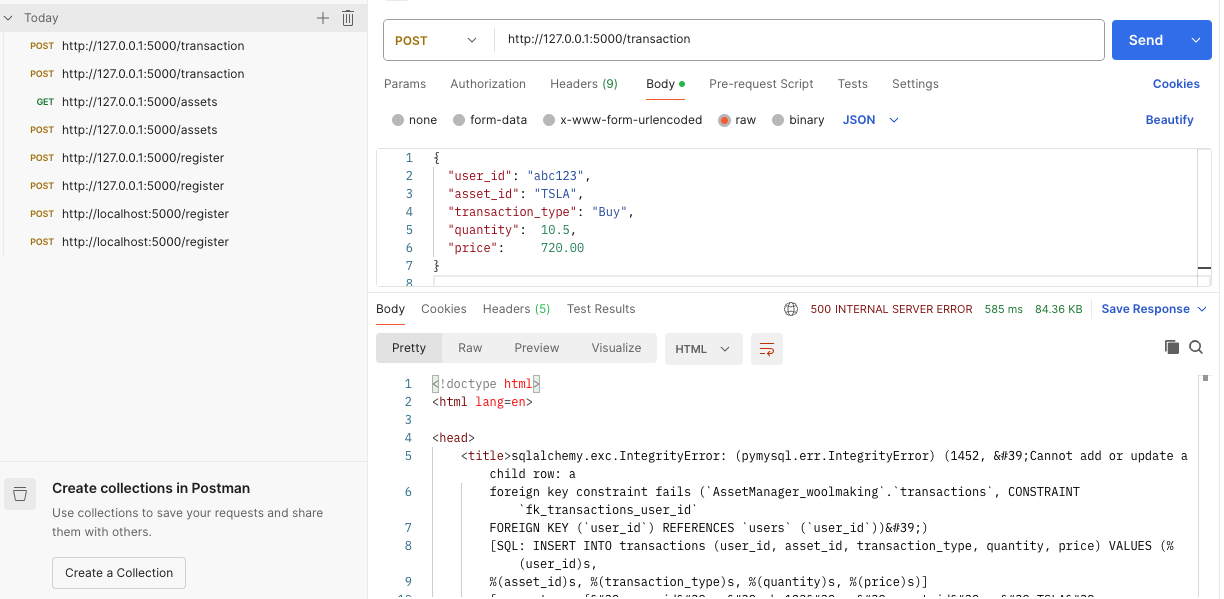
The second attempt was successful:



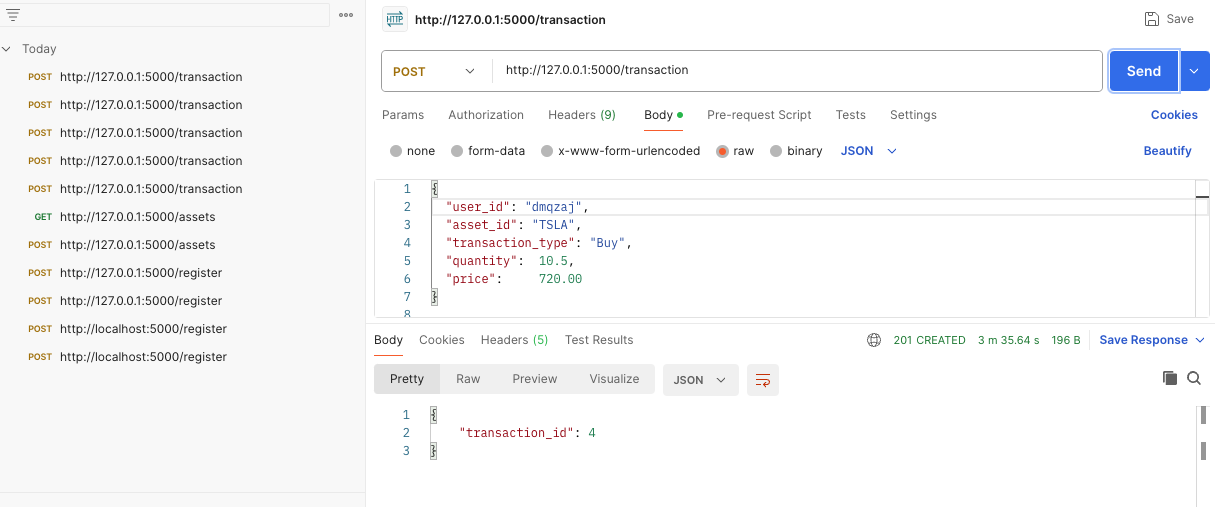
Get request to get the valid assets:



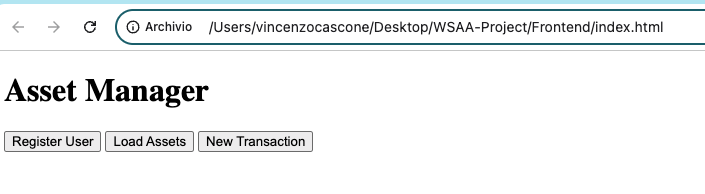
The attempt to retrieve the transaction for a user it doesn’t exist failed:



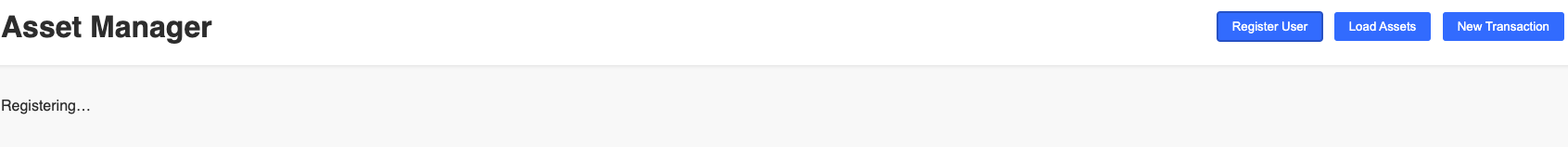
Created the transaction for the user:



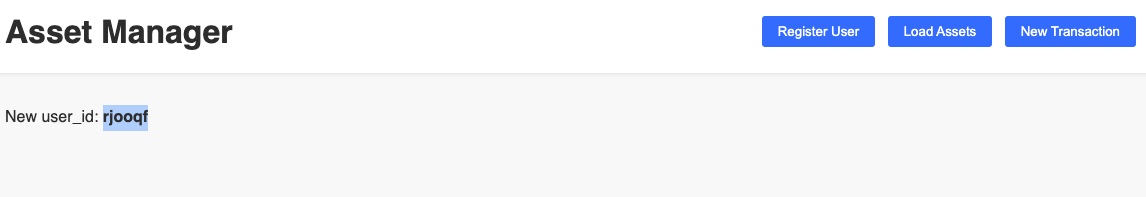
Created the initial interface:



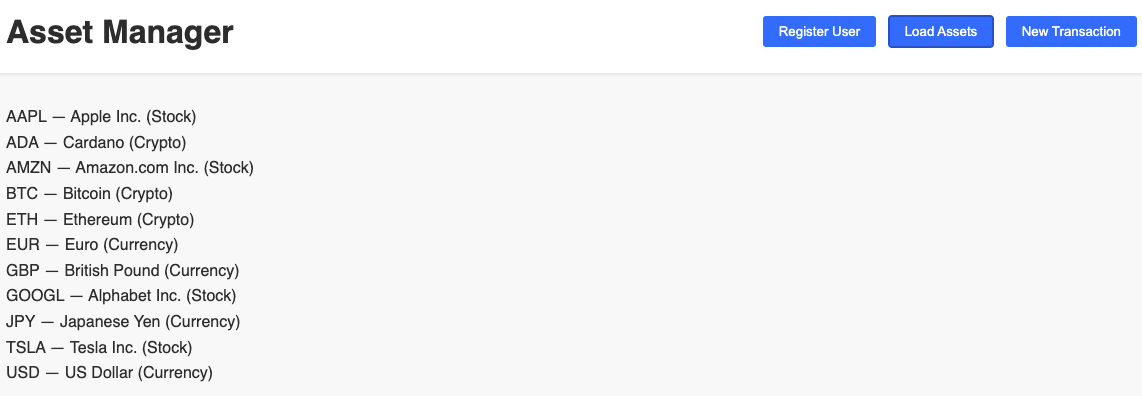
Initial HTML skeleton for the Project Frontend with css styling AJAX script:



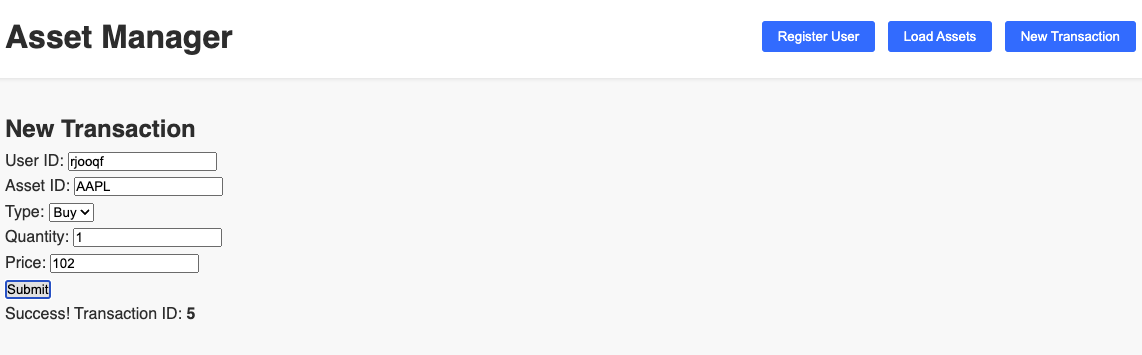
User created:



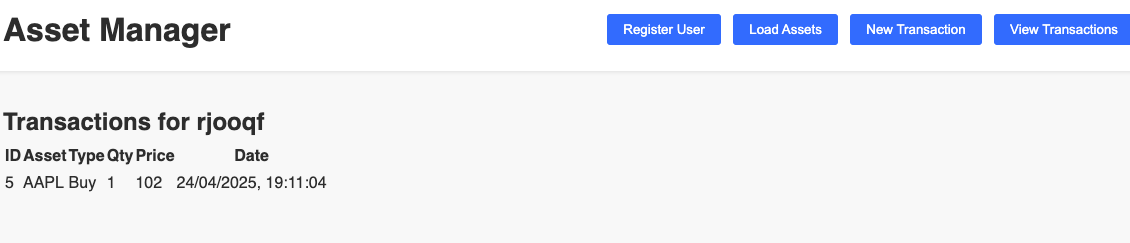
Assets loaded:



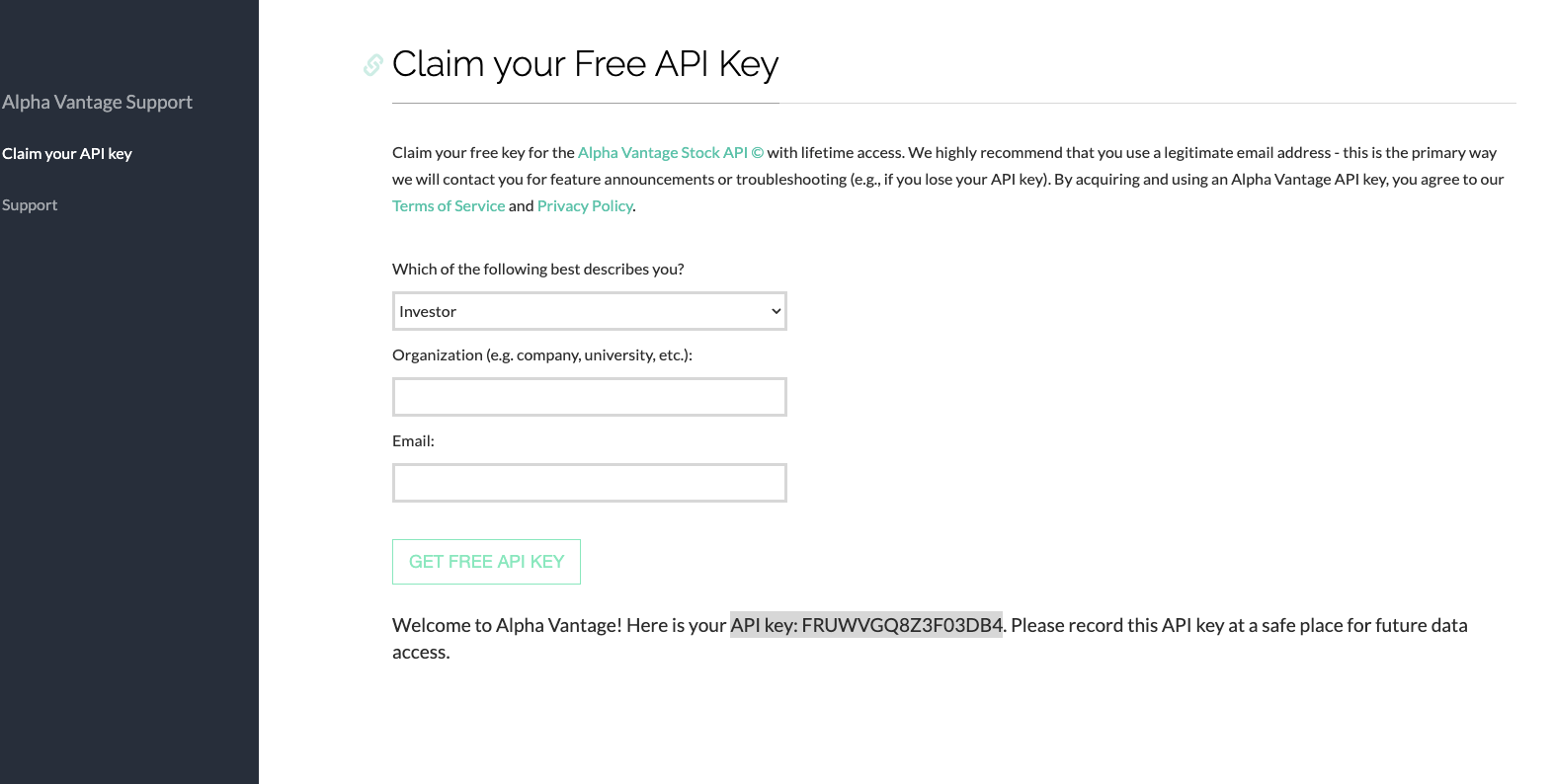
Generated the first transaction:



loading the transaction:



Got Alpha Vantage Key to get real time stock data:



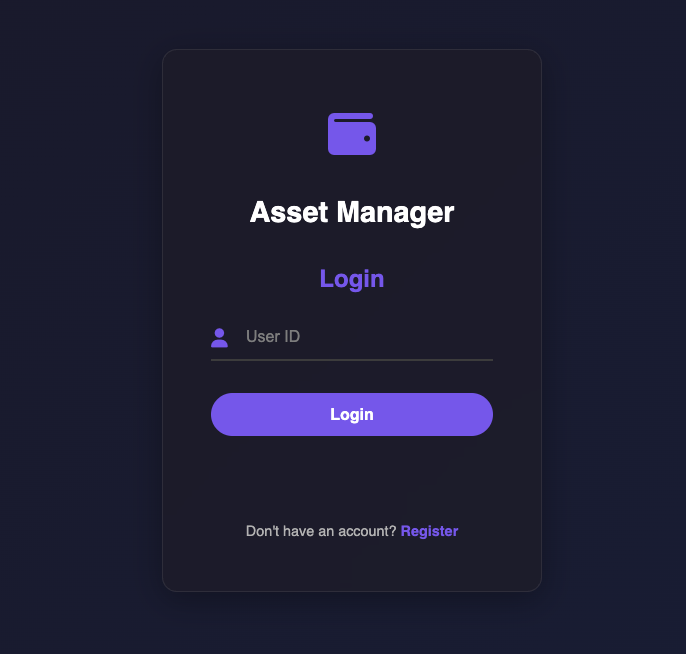
3rd party API to get real time stock information

Separated the pages into login, register and dashboard

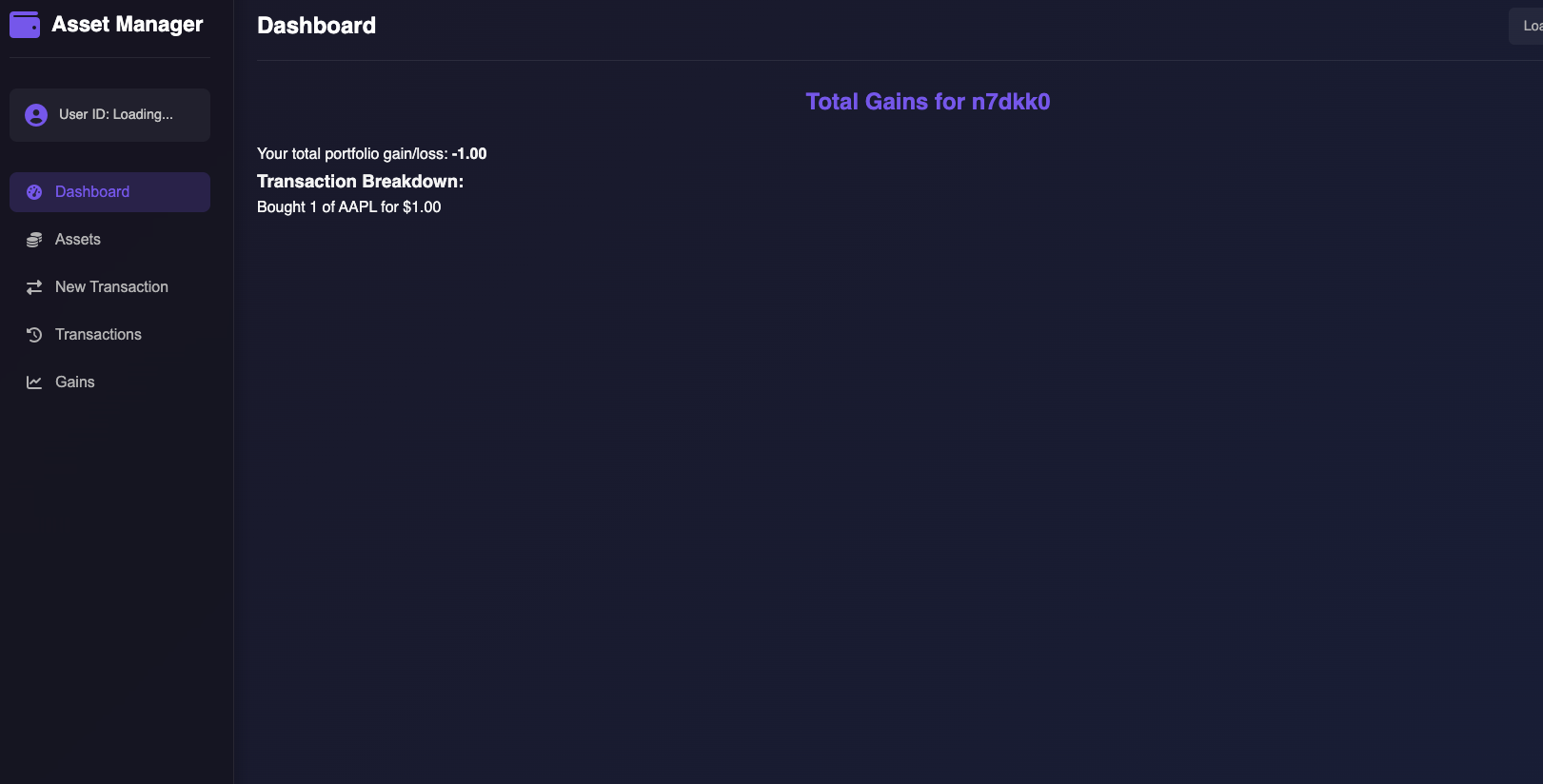
Designing dashboard for user information

Separated javascript files

Added css to improve the interface:



Changed design of the dashboard, added navigation tool and sidebar:



n7dkk0

After debugging the dashboard showed live price data:

