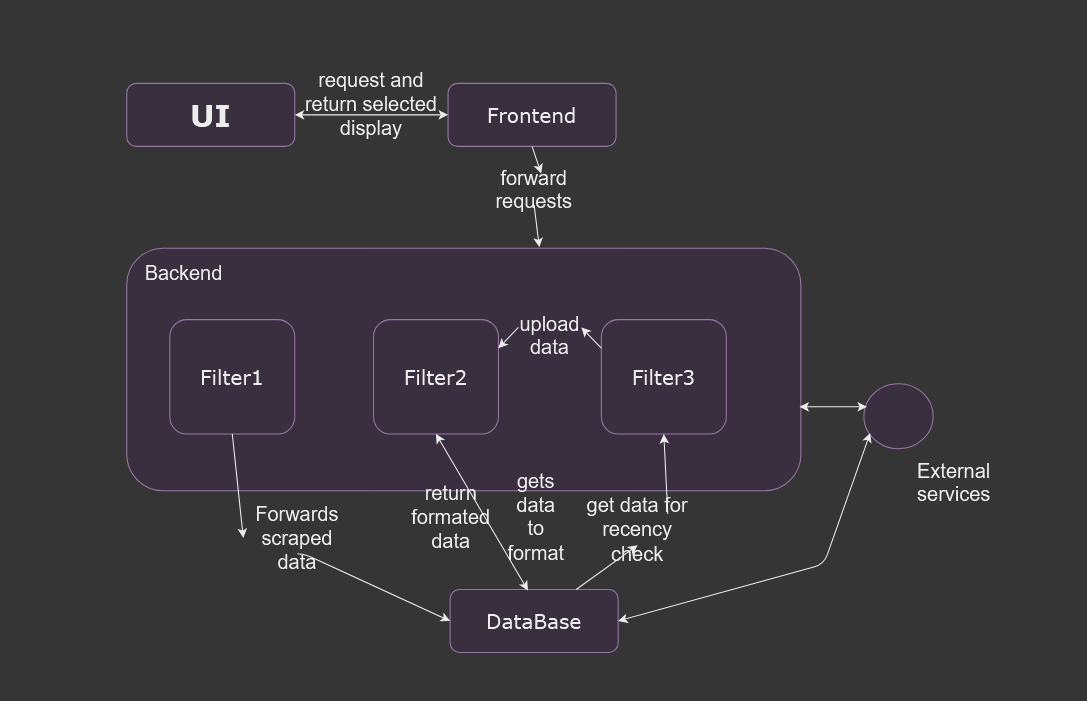
# **Macedonian Stocks Analyzer – Architectural Design**

**Conceptual Architecture Model**



**User Interface / Frontend:**

* Displays data,
* Search functionality,
* Issuers data,
* Data visualizations,
* Personalization,
* Report Generation,

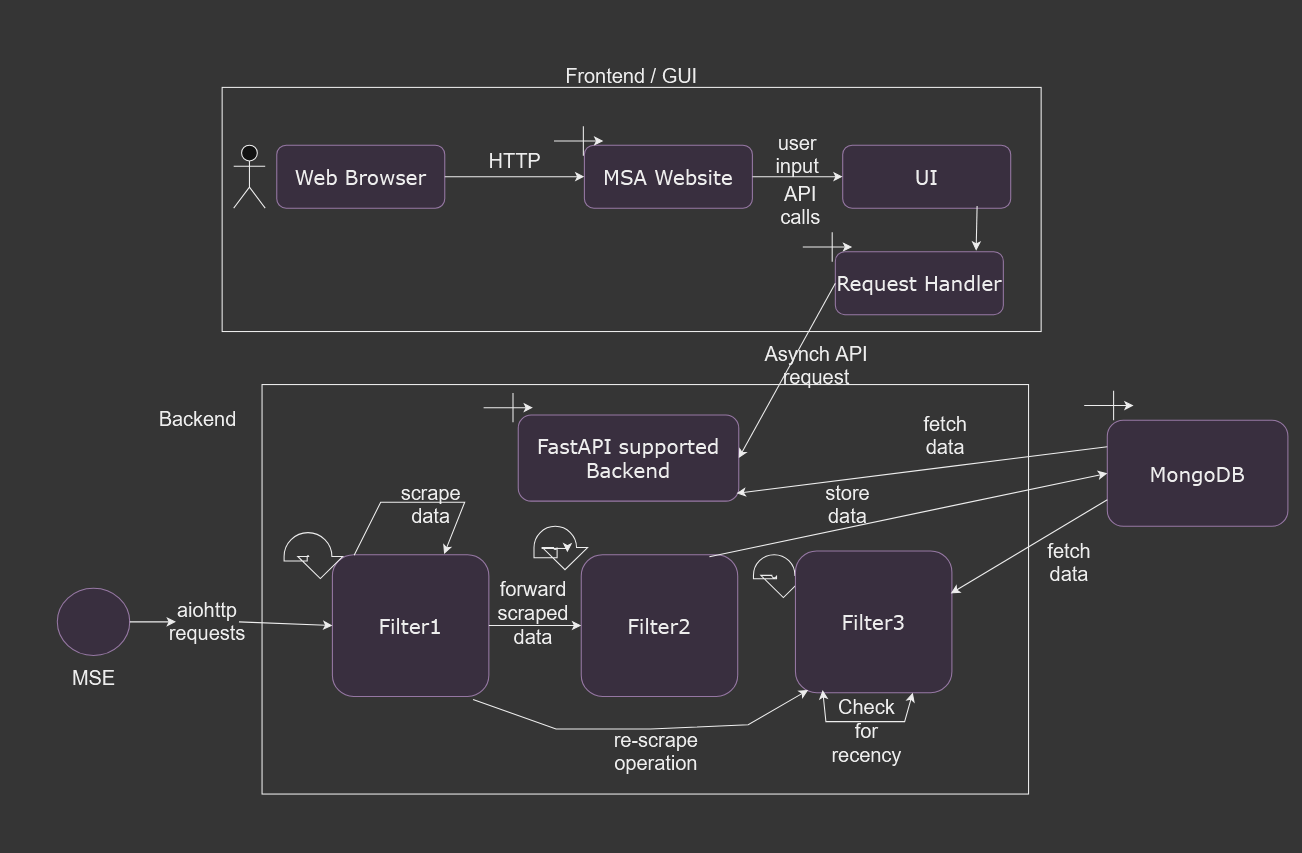
**Backend:**

* Process requests sent from the **Frontend** and return results through the API
  + Filter 1: Scrape issuers from the MSE website, validate them and insert them in the database,
  + Filter 2: Scrape data for valid issuers, format the price values, and store them in the database,
  + Filter 3: Get last scraping date for valid issuers, compare to the user's system date, and scrape new data if found within the date range.

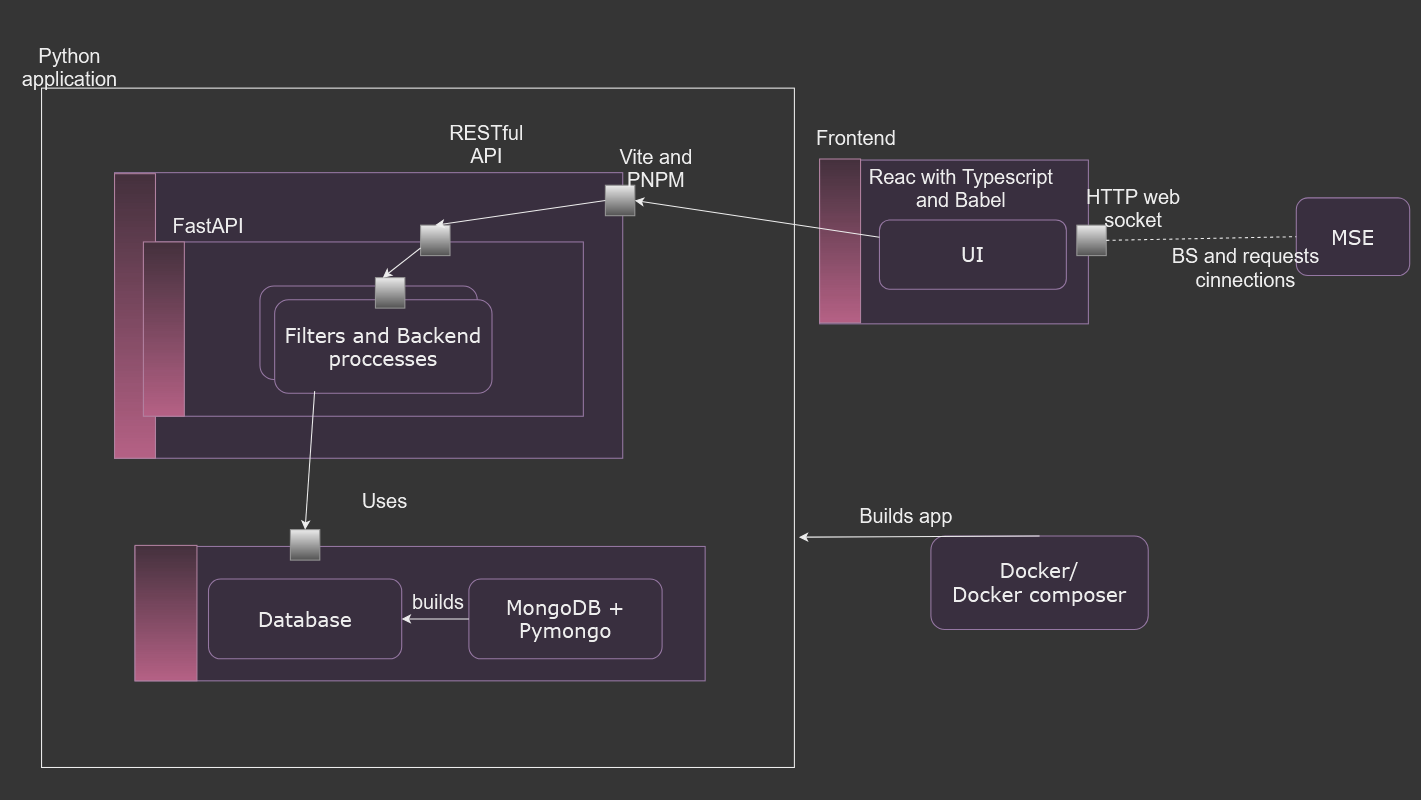
**Database:**

* Store data,
* Forward requested data.

**Execution Architecture Model**



**Implementation Architecture Model**



**Frontend (React + TypeScript + Babel)**:

* Interacts with the backend API (FastAPI) to fetch and display stock data.
* Sends requests to FastAPI for filtering and refreshing the stock data.

**Backend (FastAPI + Uvicorn)**:

* Exposes REST APIs to interact with the frontend.
* Manages the execution of **Filter 1, Filter 2, and Filter 3.**
* Handles interactions with **MongoDB** to store/retrieve data.
* Manages asynchronous execution.

**MongoDB**:

* A NoSQL database that stores processed stock data.

**Docker and Docker Composer:**

* Used to build the application before running it.

# **Technologies and Tools**

**1. Frontend**:

- ReactJS with TypeScript and Babel,

- Interactive UI,

- API integration.

**2. Backend**:

- FastAPI (Python-based framework for RESTful APIs),  
 - Uvicorn (ASGI server for concurrency),  
 - RESTful API endpoints for frontend interaction.

**3. Containerization:**

- Docker (to containerize each component),

- Docker Compose (to orchestrate multiple containers).

**4. Deployment:**

- Build Docker images for all components,

- Deploy and manage containers using Docker Compose.

# **Architecture styles and patterns**

**Layered Architecture**

* **Pipes and Filters**

Processing data decouples and made into a independent system to

Scrape data, format, validate and update.

* **Distributed architecture**
* Filters 1-3 all are microservice threads

All components are contained separately in order to improve maintainability, scalability and to ensure we can easily improve upon the application in the future.

**Execution data flow:**

* UI send an API call based on user request and input on the UI,
* Backend fetches requested data.
* The data is checked if it is outdated, if so, backend will initiate an update.
* Processed data is then sent to the frontend for displaying.

**Implementation aspect:**

Model View Controller

* Model: MongoDB – storage
* View: React with TypeScript and Babel – UI and frontend technologies,
* Controller: FastAPI – backend request handler.

Database acts as a repository, abstracting data access and operation.

FastAPI and Uvicorn use asynchronous request messages.

**Hybrid architecture overview:**

The application uses:

* Layered architecture
* Pipes and filters
* Threads and microservices
* Distributed architecture
* MCV and repository systems
* Asynchronous invocations
* Containerization