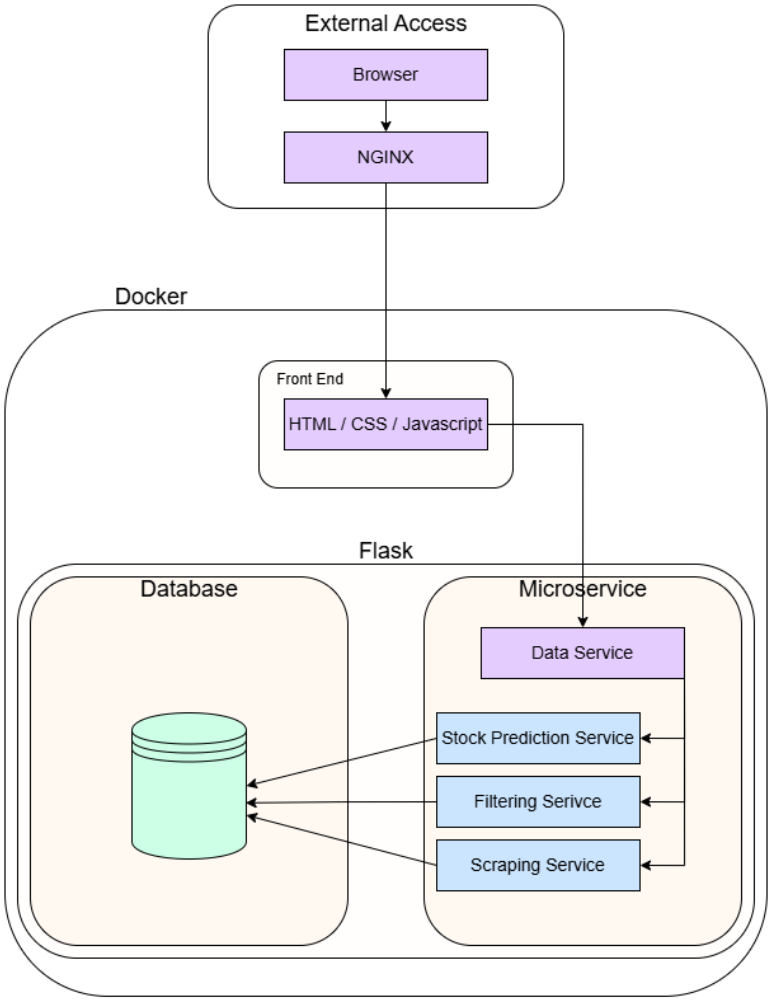
# Implementation Architecture

## *Components, Data Flow and Communication*

The provided implementation architecture for the stock market analysis application is a hybrid design that integrates various components to ensure modularity, scalability, and efficiency. The architecture leverages modern software development techniques such as containerization, microservices, and layered design.




## *Components and Their Roles*

**1. External Access Layer:**

- Browser: Acts as the user interface for external clients, allowing users to interact with the system via a graphical or web-based interface.

- NGINX: Serves as a reverse proxy, routing requests from the browser to the appropriate backend components. It handles load balancing and improves security by isolating the frontend from direct backend access.

**2. Docker Environment:**

- All components are encapsulated within Docker containers to ensure consistent deployment across environments. Docker isolates the application from the host system, providing reliability and portability.

**3. Frontend:**

- Built using HTML, CSS, and JavaScript, the frontend layer provides an interactive user interface for accessing and visualizing processed stock market data.
- It communicates with the backend through REST APIs, making it loosely coupled and easily replaceable or upgradable.

**4. Backend - Flask Framework:**

- The backend is implemented using Flask, a Python web framework, and is responsible for orchestrating the core functionalities of the system. It is divided into two main components:
    - Database:
        - A centralized data storage solution that holds historical and real-time stock data.
        - The database interacts with various services to provide efficient read and write operations.
    - Microservices:
        - A set of independent, modular services, each with a specific responsibility:
            - Data Service: Acts as an intermediary between the frontend and other backend services. It handles user requests, fetches data, and processes responses.
            - Stock Prediction Service: Analyzes historical stock data to provide predictions and trends.
            - Filtering Service: Cleanses, formats, and transforms raw data into a consistent structure for storage and analysis.

- Scraping Service: Automates the retrieval of raw stock market data from external sources, such as the Macedonian Stock Exchange website.

## *Data Flow and Communication*

- The architecture uses a pipe-and-filter pattern for data processing:
    1. The Scraping Service retrieves raw data and passes it through the Filtering Service, which formats and cleanses the data.
    2. The filtered data is then stored in the Database.
    3. The Stock Prediction Service retrieves this stored data for analysis and prediction.
    4. All components communicate with the Data Service, which provides a unified interface for the frontend to interact with the backend.
- Arrows in the Diagram:
    - Represent the flow of data and interactions between components.
    - Solid arrows between frontend, backend, and microservices show direct communication.
    - Arrows between microservices and the database indicate data access operations.