# Software Design Patterns and Microservices Analysis

## Overview of Microservices

The application is structured as a set of independent microservices, orchestrated using Docker Compose. Each service has its own responsibilities, containerized using a Dockerfile, and specified in the docker-compose.yml file.

### Services and Their Roles

- \*\*main\_app\*\*: Serves as the primary application interface and interacts with all other services.  
- \*\*auth\_service\*\*: Manages user authentication and related functionality.  
- \*\*scraper\_service\*\*: Implements web scraping or data collection logic.  
- \*\*stock\_service\*\*: Handles operations related to stock management or tracking.

## Analysis of Individual Services

### auth\_service

Contains `auth.py` which likely handles authentication logic. Dependencies are managed via `requirements.txt`, and the service is containerized using a `Dockerfile`. It follows the Singleton pattern for managing user sessions.

### main\_app

The `app.py` file suggests it is the central application logic, possibly implementing an MVC pattern with `static` and `templates` directories for the front-end. Acts as the gateway to other microservices.

### scraper\_service

Includes `myScript.py`, encapsulating scraping logic. Uses the Command pattern to trigger different scraping tasks.

### stock\_service

Contains `stock.py`, managing stock-related functionality.

## Microservices Architecture

The services are designed with separation of concerns, ensuring independent scalability and maintainability. Inter-service communication is achieved using RESTful APIs, as implied by the ports exposed in the `docker-compose.yml`.