# Architectural Design Document for Macedonian Stock Exchange Scraper

## 1. Introduction

This document outlines the architectural design for an application that scrapes data from the Macedonian Stock Exchange (MSE) website (www.mse.mk). The app collects historical data for each company from January 2014 to January 2024 and stores it in a NoSQL database. The data is showcased in a table format. This document provides conceptual, execution, and implementation views of the architecture.

## 2. Conceptual Architecture

The conceptual architecture presents the high-level vision of the application's structure, focusing on the main components and their responsibilities.

Key components include:  
- \*\*Web Scraper\*\*: Fetches data from www.mse.mk for the specified timeframe.  
- \*\*Data Processor\*\*: Cleans and organizes the scraped data.  
- \*\*NoSQL Database\*\*: Stores the processed data for retrieval.  
- \*\*Data Viewer\*\*: Provides a tabular representation of the stored data.

\*\*Diagram Needed:\*\* A block diagram showing these components and their interactions.

## 3. Execution Architecture

The execution architecture focuses on the runtime behavior of the application, including the flow of data and communication between components.

Key elements include:  
- \*\*Web Scraper\*\*: Uses HTTP requests to fetch data. Implements retry logic for reliability.  
- \*\*Data Flow\*\*: The scraper sends raw data to the data processor, which then stores it in the NoSQL database.  
- \*\*Data Viewer\*\*: Queries the database and renders data in a tabular format.  
- \*\*Deployment\*\*: The app will be containerized using Docker, with each component running in its own container.

\*\*Diagram Needed:\*\* A sequence diagram to illustrate data flow during execution.

## 4. Implementation Architecture

The implementation architecture provides details on the technologies and frameworks used to build the application.

Key decisions include:  
- \*\*Programming Language\*\*: Python, for its robust web scraping libraries (e.g., BeautifulSoup, Requests).  
- \*\*Database\*\*: MongoDB, chosen for its schema-less design and flexibility.  
- \*\*Frontend : Uses React.  
- \*\*Containerization\*\*: Docker, to simplify deployment and ensure consistency across environments.

\*\*Diagram Needed:\*\* A class or module diagram showing the code structure.

## 5. Subarchitectures

### 5.1 Pipes and Filters

The application uses the Pipes and Filters architecture to process data. Data flows through a sequence of filters:  
- \*\*Scraper (Pipe)\*\*: Fetches raw data.  
- \*\*Processor (Filter)\*\*: Cleans and formats the data.  
- \*\*Storage (Pipe)\*\*: Saves the data into the database.

### 5.2 Layered Web Architecture

The application is designed with separation of concerns:  
- \*\*Presentation Layer\*\*: (Future) User interface to view data.  
- \*\*Business Logic Layer\*\*: Processes data before storage.  
- \*\*Data Access Layer\*\*: Interacts with the NoSQL database.

### 5.3 Distributed Architecture with Microservices

The application will be modularized into the following microservices:  
- \*\*Scraping Service\*\*: Responsible for fetching data.  
- \*\*Processing Service\*\*: Cleans and organizes data.  
- \*\*Data Viewer Service\*\*: Queries the database and displays data.

### 5.4 Containerization

The app will be containerized using Docker. Each component (e.g., scraper, processor, database) will run in its own container, ensuring modularity and simplifying deployment.

## 6. Design Documents

The following diagrams and descriptions are essential for understanding the application's design:  
- \*\*Conceptual Block Diagram\*\*: Shows major components (Web Scraper, Data Processor, Database, Viewer).  
- \*\*Execution Sequence Diagram\*\*: Illustrates the data flow between components during runtime.  
- \*\*Implementation Class Diagram\*\*: Details the organization of code into classes and modules.

These diagrams should be created using tools like draw.io or Microsoft Visio and included in the final submission.

## 7. Future Considerations

While the current design focuses on basic functionality, future considerations include:  
- Expanding the scraper to handle dynamic pages if required.  
- Adding visualization features like graphs or charts.  
- Enhancing scalability to handle larger datasets.