stoX.mk

Stanko Dzeparoski 221562
Marija Blazheska 221503
Natalija Trajkoska 211514

Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the functional and non-functional requirements for **stoX.mk**, a cloud-hosted web application designed for analyzing stock market data from the Macedonian Stock Exchange. This document provides a detailed overview of the application's objectives, features, and operational framework, ensuring alignment among stakeholders, developers, and end users throughout the development process.

1.2 Document Conventions

This section outlines the document conventions used throughout this SRS to ensure consistency and clarity.

1.2.1 Notational Conventions

- Bold Text: Used for section headers and key terms that are being defined.
- Italic Text: Used to emphasize important information or terms within a sentence.

1.2.2 Requirement Labeling

- FR: Functional Requirements are labeled as FR followed by a number (e.g., FR1, FR2).
- NFR: Non-Functional Requirements are labeled as NFR followed by a number (e.g., NFR1, NFR2).

1.2.3 Use Case Labeling

- UC: Use Cases are labeled as UC followed by a number (e.g., UC1, UC2).

1.2.4 Table Conventions

- Table Headers: Bold and centered.
- Table Data: Regular font, left-aligned for text, and centered for numerical data.

1.2.5 Cross-References

- Requirements: Cross-references to requirements will be indicated in parentheses (e.g., FR1).
- Figures and Tables: Cross-references to figures and tables will be indicated with their label and number (e.g., Figure 1, Table 2).

1.3 Scope

stoX.mk is a database-driven web application that automates the collection, processing, and analysis of historical daily stock data for companies listed on the Macedonian Stock Exchange. Covering data from the past decade, the application facilitates informed decision-making by providing users with a simple, login-free interface to view stock data and perform advanced analysis such as technical and fundamental evaluations.

The application is built using the **Model-View-Controller** *(MVC)* design pattern and leverages a microservices architecture where each service communicates through **RESTful APIs.** This modular approach ensures scalability and maintainability, while the cloud-hosted infrastructure provides high availability and seamless accessibility. The application's deployment strategy incorporates containerization technologies like **Docker**, enabling portability and efficient cloud deployment.

1.4 Intended Audience
This document is intended for the project stakeholders, including developers, testers, project managers, and potential users such as investors and analysts. It serves as a blueprint for the application's development and deployment and as a reference for ensuring all requirements are met.

1.5 Goals and Objectives
The primary goal of **stoX.mk** is to provide investors, financial analysts, and other stakeholders with accurate, comprehensive, and up-to-date stock market data in a user-friendly platform. Specific objectives include:

- Automating data collection and processing from the Macedonian Stock Exchange.
- Ensuring data accuracy and consistency through standardization and validation.
- Enabling users to perform advanced technical and fundamental analysis.
- Deploying the application to the cloud for scalability and accessibility.
- Providing a streamlined and intuitive interface requiring no login credentials.

2. Overall Description

2.1 Product Perspective
stoX.mk is a standalone, cloud-hosted web application designed to automate and simplify the process of collecting, processing, and analyzing stock market data from the Macedonian Stock Exchange. The system integrates a backend database, micro-services for data processing, hosted on a cloud infrastructure to ensure high availability and reliability and a user-friendly web interface.

2.2 Product Features
- **Automated Data Collection:** Retrieve historical daily stock data for all issuers on the Macedonian Stock Exchange.
- **Data Standardization:** Clean and format raw stock data into a standardized structure suitable for analysis.
- **Advanced Analysis Tools:** Enable users to perform technical and fundamental analysis directly through the web interface.
- **Login-Free Access:** Simplified interface allowing users to view data without requiring account creation or login credentials.
- **Real-Time Updates:** Keep stock data up-to-date by automatically fetching new information as it becomes available.
- **Cloud Deployment:** High availability and scalability through cloud hosting, ensuring consistent performance.
- **Performance Metrics:** Monitor and optimize the efficiency of data population and processing pipelines (available to stakeholders only).

2.3 User Classes and Characteristics
The primary user classes for **stoX.mk** include:

- **Investors:** Casual or professional investors seeking detailed stock performance data to make informed investment decisions.
- **Financial Analysts:** Professionals analyzing market trends, stock movements, and company performance.
- **Data Scientists:** Users interested in accessing clean, structured datasets for further modeling and analysis.
- **Academic Researchers:** Researchers studying stock market dynamics and trends.

Characteristics of all user classes:

- No technical expertise required for basic data viewing and analysis.

- Users rely on accurate and up-to-date data to support their decisions.
- Advanced users may leverage analysis tools for deeper insights into stock performance.

2.4 Operational Environment

The **stoX.mk** application will operate in the following environments:

- **Client-Side Requirements:**
  - Modern web browsers (e.g., Chrome, Firefox, Edge).
  - Internet access for accessing the cloud-hosted platform.
- **Server-Side Requirements:**
  - Cloud-based infrastructure with container orchestration (e.g., Docker Swarm).
  - RESTful API endpoints for data processing and retrieval.
  - A relational database (e.g., SQLite 3) for storing processed data.

2.5 Design and Implementation Constraints

- **Technological Constraints:**
  - The application must use **Python** for data processing and transformation tasks, utilizing its extensive libraries for handling large datasets efficiently.
  - **Java** and **Python** are used for backend services and API development, ensuring robust and scalable server-side functionality.
  - The database system will use **SQLite 3** to store stock market data, chosen for its lightweight and serverless architecture, making it suitable for the project's initial scope.
  - The application must utilize **RESTful APIs** to enable communication between the microservices, ensuring a standardized and modular approach to system integration.
- **Performance Constraints:**
  - The SQLite 3 database may face limitations when handling extensive concurrent operations or very large datasets. Scaling to a more robust database solution may be necessary in future iterations.
  - RESTful APIs must be designed and optimized to minimize latency and handle multiple requests efficiently.
  - Seamless interaction between Python, Java, and the database is critical to ensure system responsiveness and reliability.
- **Deployment Constraints:**
  - The application will be containerized using **Docker** to encapsulate its Python, Java, and SQLite components, along with all dependencies and configurations.
  - Docker containers must be compatible with the chosen cloud hosting environment, ensuring smooth deployment and scalability.
  - The deployment process must include orchestrating multiple containers, if necessary, to manage the microservices architecture effectively.
- **Integration Constraints:**
  - RESTful APIs must be implemented to allow seamless communication between microservices, supporting data exchange between Python and Java components.
  - The SQLite 3 database must be accessible to all relevant microservices, with appropriate mechanisms to prevent conflicts during read/write operations.
- **Regulatory and Security Constraints:**
  - The application must ensure secure data transmission over APIs, employing encryption protocols like HTTPS for all communications.
  - Sensitive financial data must be stored securely in the SQLite 3 database, following best practices for database encryption and access control.

2.6 Assumptions and Dependencies

- **Assumptions:**
  - Users have access to the internet and a modern web browser.
  - The Macedonian Stock Exchange website provides accessible and reliable stock data.
  - Cloud hosting services are available and support containerized applications.
- **Dependencies:**
  - Availability of Python and Java libraries for data processing and analysis.
  - Integration with a cloud provider that supports Docker-based deployments.
  - Reliable third-party tools for monitoring performance and uptime of the cloud-hosted platform.

3. System Features

3.1 Requirements Priority

Priority Level	Description
Priority 1	Essential and required functionality
Priority 2	Desirable functionality
Priority 3	Extra features, features planned to be further implemented

3.2 Functional Requirements

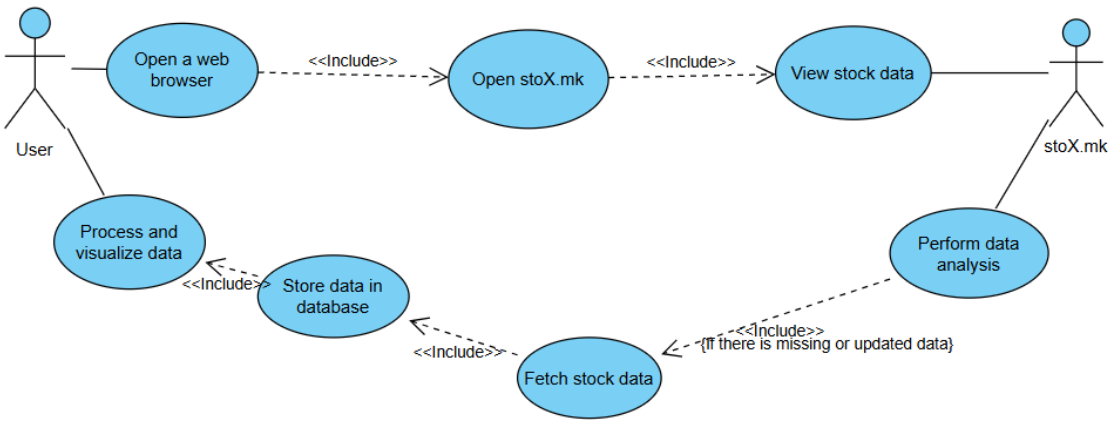
FR1: The system shall automatically retrieve daily stock data for all issuers listed on the Macedonian Stock Exchange (MSE), using an automated process to extract the list of valid issuers from the MSE webpage.
FR2: The system shall process raw data to ensure that only relevant stock information is retained and that all data entries are correctly formatted.
FR3: The system shall support exporting the stock data to a SQLite 3 database for further analysis, allowing users to save data for offline access.
FR4: The system shall update the database with newly fetched data, combining it with existing records, ensuring there are no duplicates and maintaining data integrity.
FR5: The system shall provide basic data analysis features, such as calculating daily price changes, volume trends, and other stock metrics to support decision-making.
FR6: The system shall detect and log errors in data retrieval or processing, providing meaningful error messages for troubleshooting and recovery.
FR7: The system shall use a RESTful API to enable seamless communication between micro-services, ensuring modularity and extensibility.
FR8: The system shall implement the Model-View-Controller (MVC) design pattern to maintain a clear separation of concerns between the data processing logic, user interface, and backend components.
FR9: The system shall allow the Python-based data processing service to interact with Java-based backend services via RESTful API endpoints, ensuring interoperability between components.
FR10: The system shall support deployment through Docker containers, enabling seamless packaging, portability, and consistent runtime environments across different platforms.
FR11: The system shall provide an API endpoint for exporting analysis results, allowing integration with third-party tools or platforms.
FR12: The system shall generate visualizations of stock performance trends, including line charts, bar graphs, and candlestick charts, directly within the web interface.
FR13: The system shall include an automated scheduler to periodically fetch and update stock data without manual intervention.
FR14: The system shall allow users to filter and sort stock data based on metrics such as issuer name, price, volume, and date.

3.3 Non-functional Requirements

NFR1: The system shall ensure that stock data retrieval, processing, and updates happen with minimal delays in the user interface.
NFR2: The system shall be designed to handle large amounts of data, supporting the addition of new issuers and the growth of historical data.
NFR3: The system shall be compatible with multiple operating systems, such as Linux, macOS, and Windows, ensuring accessibility.
NFR4: The system shall be highly available and capable of recovering gracefully from errors, ensuring no data is lost and no processing is corrupted in case of failure.
NFR5: The system shall ensure that all stock data is accurately recorded and formatted consistently, with no missing or corrupted data entries.
NFR6: The system shall follow established software development practices and standards, ensuring that the codebase is clean, well-documented, and easy to maintain.
NFR7: The system shall optimize the use of RESTful APIs for efficient data exchange between microservices, ensuring low latency and high reliability.
NFR8: The system shall utilize Docker to containerize individual components, enabling smooth deployment, scalability, and compatibility across cloud environments.
NFR9: The system shall ensure secure API communication by employing encryption protocols such as HTTPS and secure authentication mechanisms where applicable.
NFR10: The system shall maintain consistent performance and scalability when deployed on cloud infrastructure, supporting dynamic workloads and traffic fluctuations.
NFR11: The system shall support multi-threaded or asynchronous operations to improve performance for data-intensive tasks.
NFR12: The system shall provide a monitoring and logging framework to track system health, API usage, and data processing errors.
NFR13: The system shall provide scalability to support a growing number of users and increased data volume without degradation in performance.
NFR14: The system shall adhere to container orchestration standards (e.g., Kubernetes) for efficient management of Docker containers in production environments.

4. External Interface Requirements

4.1 UC



Actor: User
Description: User accesses the stoX.mk website
Goal: View stock data

Requirements covered: FR1, FR3, FR5, FR12, NFR1, NFR2, NFR5, NFR6, NFR10

5. References

- stoX.mk project this document is based on - https://github.com/StankoDzeparoski/DAS-MSE-project
- Macedonian Stock Exchange website - https://www.mse.mk/mk