Stock Market Dashboard Documentation for WEBLAB Project

Introduction and Goals

This chapter describes the basic goals of the Stock Market Dashboard project and the main requirements.

Task Description

The goal of the Stock Market Dashboard is to display stocks in real-time. This should be realized through a dashboard that can be created and modified as desired.

Primary functions of the dashboard are: * User registration and login. * Creation of a personal dashboard where the display is saved separately for each user. * The user can decide which stocks they want to see. * Graphical representation of stock prices in the dashboard. * Live updates of stock prices.

Quality Goals

- High performance and fast loading times.
- Scalability to support many concurrent users.
- High security to protect user data.
- User-friendliness and intuitive operation.

Constraints

- Use of TypeScript and JavaScript.
- Use of npm for package management.
- Use of the Angular framework for the frontend.
- MongoDB as the database.
- Docker for containerization of the application.

Context and Boundaries

The project must meet the following external and internal requirements: * Compliance with data protection regulations. * Use of open-source technologies. * Integration into existing IT infrastructure.

Solution Strategy

The solution consists of a backend written in TypeScript using a MongoDB database, and a frontend developed with Angular. Docker is used for containerization and easy deployment of the application.

Building Block View

The system consists of the following main components: * Backend server (TypeScript, Node.js) * Frontend application (Angular) * MongoDB database * Docker containers for easy deployment

Runtime View

The backend server provides a REST API that is used by the frontend to load and save data. The MongoDB database stores stock information and user data. Docker is used to run the application in containers, making deployment and scaling easier.

Deployment View

The application runs in several Docker containers: * One container for the backend server. * One container for the frontend application. * One container for the MongoDB database.

Cross-cutting Concepts

- **Security:** Use of JWT for authentication and authorization.
- **Data Persistence:** MongoDB for storing user data and stock information.
- **Containerization:** Docker for deployment and scaling of the application.

Architectural Decisions

- Use of TypeScript for the backend for better type safety.
- Use of Angular for the frontend due to its modularity and strong community support.
- Use of MongoDB for its flexibility and scalability.
- Use of Docker for containerization for easy deployment and scaling.

Quality Requirements

- **Performance:** Fast loading times and smooth user experience.
- **Scalability:** Support for many concurrent users.
- **Maintainability:** Well-structured and documented code.

- **Security:** Protection of user data and secure authentication.

Risks and Technical Debt

- **Risks:** Potential security vulnerabilities, data loss, performance issues under high load.
- **Technical Debt:** Need for regular updates and maintenance of used libraries and frameworks.

Glossary

- **JWT:** JSON Web Token, an open standard for securely transmitting information between parties as a JSON object.
- **REST API:** Representational State Transfer, an architectural style for designing network APIs.
- **Docker:** A platform for containerizing applications.

Appendix

Contains additional information, references, or supporting materials.