Stock Market Dashboard Dokumentation WEBLAB

Hrvat Leo

Table of Contents

Chapter 1. Einführung & Ziele

Das Ziel des Stock Market Dashboards ist es, Aktien in Echtzeit anzuzeigen. Dies soll durch ein Dashboard realisiert werden, das nach Belieben erstellt und modifiziert werden kann.

1.1. Das Projekt hat folgende Ziele:

- Anzeige von Aktienkursen in Echtzeit.
- Benutzerdefinierte Dashboards erstellen und bearbeiten.
- Benutzerregistrierung und -authentifizierung.
- Benutzerfreundliche Oberfläche.

1.2. Qualitätsziele

Das Projekt soll, ausgehend von den grundlegenden Anforderungen, folgende Qualitätsziele erfüllen:

Priorität	**Qualitätsziel**	**Beschreibung**
1	Performance	Schnelle Ladezeiten und reibungslose Benutzererfahrung.
2	Benutzerfreundlichkeit	Intuitive Benutzeroberfläche und einfache Bedienung.
3	Zugangskontrolle	Sichere Benutzerregistrierung und -authentifizierung.
4	Persistenz	Zuverlässige Speicherung von Benutzerdaten und Aktieninformationen.
5	Personalisierung	Anpassbare Dashboards und Benutzeroberfläche.

1.3. Stakeholder

Die Hauptinteressengruppen des Projekts sind:

Rolle	**Erwartungen**
Benutzer	Einfache Bedienung und schnelle Anzeige von Aktienkursen.
Entwickler	Gut strukturierter und dokumentierter Code.
Dozent	Erwarten eine vollständige und pünktliche Lieferung des Projekts.

Chapter 2. Randbedingungen

Die Enwticklung des Stock Market Dashboards unterliegt verschiedenen technischen und organisatorischen Randbedingungen, die im Folgenden erläutert werden.

2.1. Technische Randbedingungen

Randbedingung	Beschreibung
Betriebssystem	Das Stock Market Dashboard wird auf einem Linux-Server betrieben.
Programmiersprache	Die Anwendung wird Front- wie auch Backend mit Javascript/Typescript entwickelt.
Framework	Das Frontend wird mit Angular und das Backend mit Expressjs realisiert.
Datenbank	Die Datenbank wird mit MongoDB realisiert.
CI/CD	Die Anwendung wird mit GitHub Actions automatisiert getestet und deployed.

2.2. Organisatorische Randbedingungen

Randbedingung	Beschreibung
WEBLAB	Das Projekt wird im Rahmen der Blockwoche WEBLAB an der Hochschule Luzern - Departement Informatik durchgeführt
Hosting	Die Anwendung wird aufgrund von keinem Server nur lokal gehosted.

2.3. Konventionen

Konvention	Beschreibung
Git	Der Code wird mittels Git verwaltet und auf GitHub gehosted.
Dokumentation	Die Dokumentation wird in Asciidoc geschrieben und auf GitHub gehosted.
Codestruktur	Der Code wird nach dem MVC-Pattern strukturiert.

Chapter 3. Kontext & Abgrenzung

3.1. Fachlicher Kontext

3.1.1. Überblick

Das Stock Market Dashboard ist eine webbasierte Anwendung zur Echtzeitüberwachung und -analyse von Aktienkursen. Die Plattform richtet sich an Privatinvestoren, Finanzanalysten und Interessierte, die Aktienmärkte beobachten möchten, ohne professionelle Trading-Software nutzen zu müssen.

3.1.2. Fachanwendungsbereiche

Bereich	Beschreibung
Aktienüberwachung	Benutzer können ausgewählte Aktien in Echtzeit überwachen und Kursbewegungen verfolgen.
Portfolioanalyse	Zusammenstellung und Visualisierung eigener Portfolios zur Performanceanalyse.
Markttrends	Identifizierung und Analyse von Markttrends durch grafische Darstellungen.

3.1.3. Benutzergruppen und Stakeholder

Stakeholder	Interessen und Anforderungen
Privatinvestoren	Einfache und übersichtliche Darstellung relevanter Aktieninformationen zur Entscheidungsunterstützung.
Finanzanalysten	Detaillierte Datenanalyse und Vergleichsmöglichkeiten zwischen verschiedenen Aktien.
Gelegenheitsnutzer	Niedrige Einstiegshürde, intuitive Bedienung und grundlegende Marktübersicht.
Systemadministratoren	Einfache Wartung, Skalierbarkeit und Robustheit des Systems.

3.1.4. Externe Schnittstellen

System/Schnittstelle	Beschreibung
Aktien-APIs	Anbindung an externe Dienste zur Beschaffung von Echtzeit-Aktiendaten (mock-Daten für Entwicklung).
Authentifizierungssysteme	Integration von sicheren Anmeldeverfahren (JWT).
Web-Browser	Kompatibilität mit gängigen Browsern für den Zugriff auf die Anwendung.

3.1.5. Geschäftsprozesse

Die Anwendung unterstützt folgende primäre Geschäftsprozesse:

1. **Benutzerregistrierung und -verwaltung**: Sichere Erstellung und Verwaltung von Benutzerkonten.
2. **Dashboard-Konfiguration**: Personalisierung der Anzeigeelemente nach individuellen Bedürfnissen.
3. **Aktienüberwachung**: Kontinuierliche Aktualisierung und Visualisierung von Aktienkursen.
4. **Datenanalyse**: Bereitstellung von Tools zur Analyse historischer und aktueller Kursdaten.

Chapter 4. Lösungsstrategie

4.1. Überblick

Die Lösungsstrategie für das Stock Market Dashboard basiert auf einer modernen Webanwendungsarchitektur mit klarer Trennung zwischen Frontend und Backend. Die Lösung nutzt etablierte Technologien und folgt bewährten Entwurfsmustern, um eine hohe Codequalität und Wartbarkeit zu gewährleisten.

4.2. Technologieentscheidungen

Technologie	Begründung
TypeScript	Bietet Typsicherheit und verbesserte Entwicklungserfahrung gegenüber reinem JavaScript, reduziert Laufzeitfehler durch statische Typprüfung.
Angular	Robustes Framework für komplexe Single-Page-Anwendungen mit umfangreicher Community-Unterstützung und ausgereiftem Komponentenmodell.
Express.js	Schlankes und flexibles Node.js-Framework für die Backend-Entwicklung mit guter Performance und einfacher API-Implementierung.
MongoDB	Dokumentenorientierte NoSQL-Datenbank, ideal für die Speicherung heterogener Daten und flexible Schemaanpassungen.
Docker	Containerisierung für konsistente Entwicklungs-, Test- und Produktionsumgebungen.
JWT	Sicherer, zustandsloser Token-basierter Authentifizierungsmechanismus.

4.3. Architekturprinzipien

Prinzip	Umsetzung
Trennung der Verantwortlichkeiten	Klare Trennung zwischen Frontend, Backend und Datenbank nach dem MVC-Pattern.
Modularität	Aufteilung der Anwendung in wiederverwendbare, unabhängige Module und Komponenten.
RESTful API	Implementierung einer REST-Schnittstelle für die Kommunikation zwischen Frontend und Backend.
Reactive Programming	Nutzung von RxJS für reaktive Datenströme und asynchrone Operationen im Frontend.
Dependency Injection	Verwendung des Angular DI-Systems für lose Kopplung und bessere Testbarkeit.

4.4. Vorgehensweise

Die Entwicklung erfolgt iterativ mit folgenden Schwerpunkten:

1. **Basisinfrastruktur**: Aufsetzen der Entwicklungsumgebung mit Docker, Konfiguration der CI/CD-Pipeline.
2. **Backend-Grundfunktionen**: Implementierung der API-Endpunkte und Datenbankanbindung.
3. **Authentifizierung**: Entwicklung des Benutzerregistrierungs- und Anmeldesystems mit JWT.
4. **Frontend-Basiskomponenten**: Erstellung der UI-Komponenten für Navigation und Layout.
5. **Dashboard-Funktionalität**: Implementierung der Aktienüberwachung und personalisierten Dashboard-Ansichten.
6. **Datenvisualisierung**: Integration von Charts und Graphen für die Darstellung von Aktienkursen.
7. **Benutzerinteraktion**: Entwicklung der interaktiven Elemente zur Dashboard-Konfiguration.

4.5. Qualitätssicherung

Maßnahme	Beschreibung
Unit-Tests	Automatisierte Tests für einzelne Komponenten und Services mit Jasmine und Karma.
End-to-End-Tests	Browserbasierte Tests des Gesamtsystems mit Protractor.
Continuous Integration	Automatisierte Builds und Tests bei jedem Code-Commit mit GitHub Actions.
Code Reviews	Systematische Überprüfung von Code-Änderungen durch andere Teammitglieder.

4.6. Technische Risiken und Lösungsansätze

Risiko	Auswirkung	Lösungsansatz
Leistungsprobleme bei großen Datenmengen	Langsame Reaktionszeit der Anwendung	Implementierung von Pagination, Lazy Loading und Daten-Caching
Echtzeit-Aktualisierungen	Hohe Serverlast durch ständige API-Abfragen	Verwendung von WebSockets für effiziente Echtzeit-Kommunikation
Sicherheitslücken	Unbefugter Zugriff auf Benutzerdaten	Konsequente Validierung von Eingabedaten, Schutz gegen XSS und CSRF-Angriffe
Browserkompatibilität	Inkonsistentes Erscheinungsbild	Cross-Browser-Testing und responsive Design-Ansätze

Chapter 5. Bausteinsicht

5.1. Übersicht System

Die Bausteinsicht zeigt die statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme) sowie deren Beziehungen zueinander.

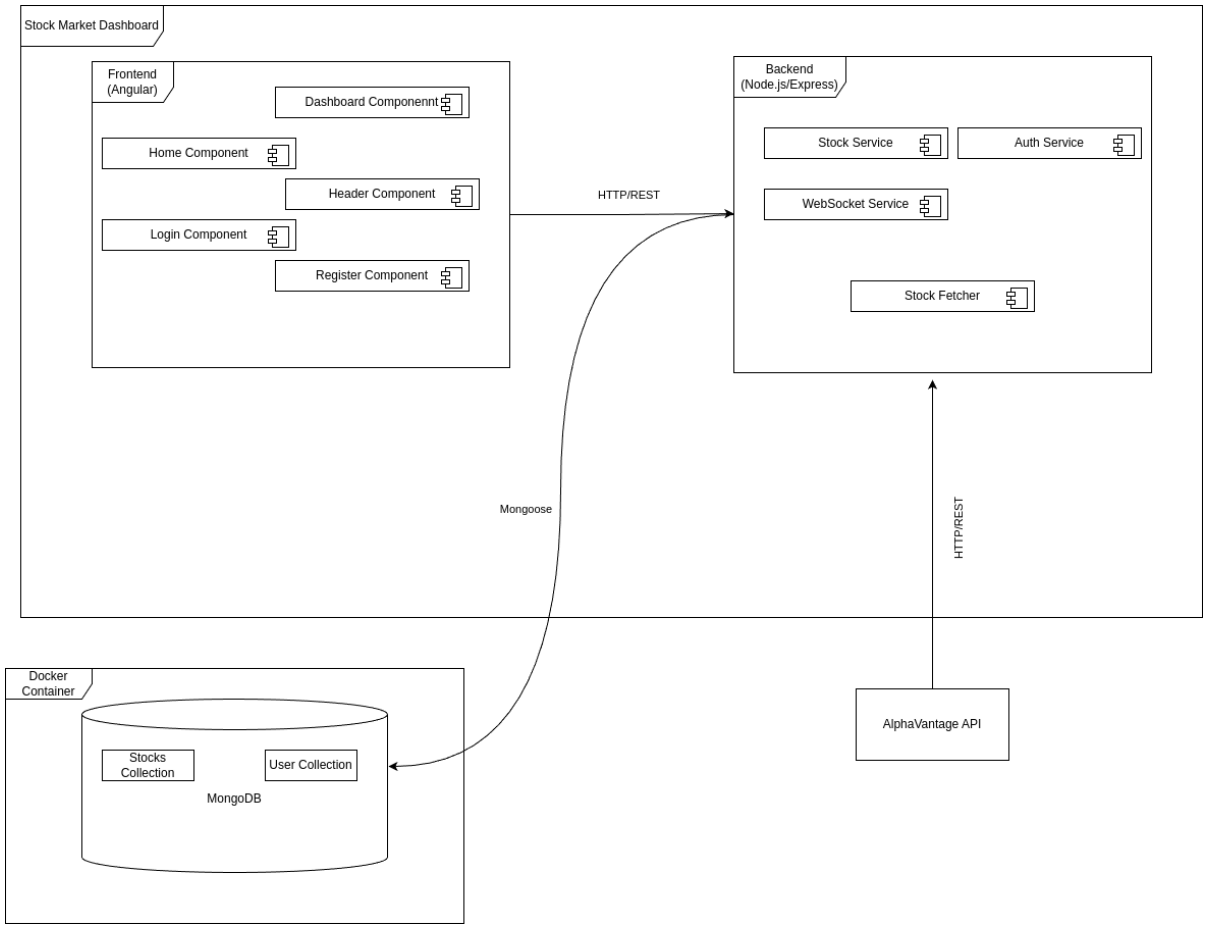

Figure 1. Systemübersicht

5.2. Ebene 1: Hauptkomponenten

Baustein	Verantwortlichkeit
Frontend (Angular)	Präsentationslogik, Benutzerinteraktion, reaktive Darstellung der Aktieninformationen
Backend (Node.js/Express)	Geschäftslogik, API-Bereitstellung, Datenbankzugriff, Integration externer Dienste
MongoDB	Persistierung von Benutzer-, Dashboard- und Aktiendaten
External Stock APIs	Bereitstellung von Echtzeit- und historischen Aktiendaten

5.3. Ebene 2: Frontend-Komponenten

Baustein	Verantwortlichkeit
Core Module	Kernfunktionalitäten wie Authentifizierung, HTTP-Interceptoren für Tokens und Routing-Guards
Shared Module	Wiederverwendbare UI-Komponenten, Pipes und Direktiven für die gesamte Anwendung
Authentication Module	Verwaltung der Benutzeranmeldung, -registrierung und Profilbearbeitung
Dashboard Module	Darstellung und Konfiguration des personalisierten Dashboards des Benutzers
Stock Visualization Module	Darstellung von Aktienkursen in verschiedenen Formaten (Charts, Listen, Details)
User Settings Module	Verwaltung der Benutzereinstellungen und Präferenzen

5.4. Ebene 2: Backend-Komponenten

Baustein	Verantwortlichkeit
API Routes	Definition der REST-Endpunkte und Weiterleitung an Controller
Controllers	Verarbeitung von HTTP-Anfragen und -Antworten, Eingabevalidierung
Services	Implementierung der Geschäftslogik und Interaktion mit der Datenbank
Models	Definition der Datenstruktur und Validierungsschemata
Middleware	Querschnittliche Funktionalitäten wie Authentifizierung, Fehlerbehandlung, Ratenbegrenzung

5.5. Ebene 3: Wichtige Schnittstellen

Schnittstelle	Beschreibung	Technologie
Frontend zu Backend	REST-API für alle Datenzugriffe und -manipulationen	HTTP/JSON, JWT für Auth
Backend zu Datenbank	Datenzugriff für Persistierung und Abfrage	Mongoose/MongoDB Driver
Backend zu externen APIs	Abrufen von Aktieninformationen	HTTP/REST, API Keys
WebSocket-Verbindung	Echtzeit-Updates für Aktienkurse	Socket.io

Chapter 6. Laufzeitschicht

6.1. Registrierung

Die Registrierung eines neuen Benutzers erfolgt über die Benutzeroberfläche des Frontends. Die Benutzerdaten werden an den Backend-Server gesendet, der die Anfrage validiert und die Daten in der MongoDB-Datenbank speichert. Nach erfolgreicher Registrierung wird der Benutzer auf den Login Screen weitergeleitet um sich anzumelden.

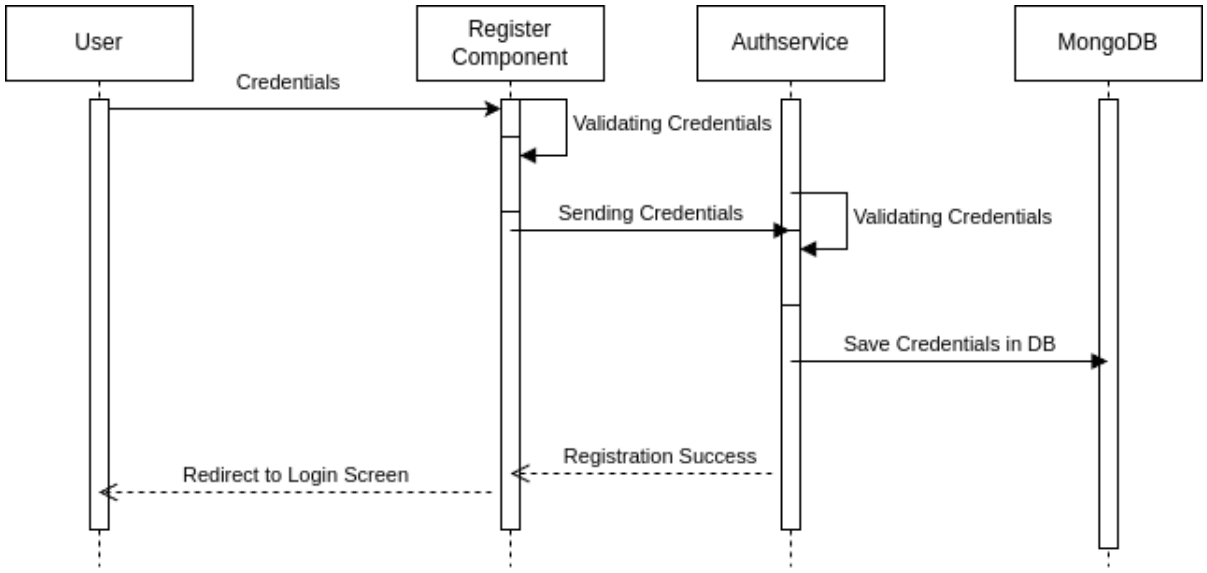

Figure 2. Registrierung

6.2. Anmeldung

Die Anmeldung eines Benutzers erfolgt über die Benutzeroberfläche des Frontends. Die Benutzerdaten werden an den Backend-Server gesendet, der die Anfrage validiert und die Daten mit der Datenbank abgleicht. Bei erfolgreicher Anmeldung wird ein JWT-Token generiert und an den Client zurückgesendet. Dieser Token wird für alle weiteren Anfragen benötigt um den Benutzer zu authentifizieren.

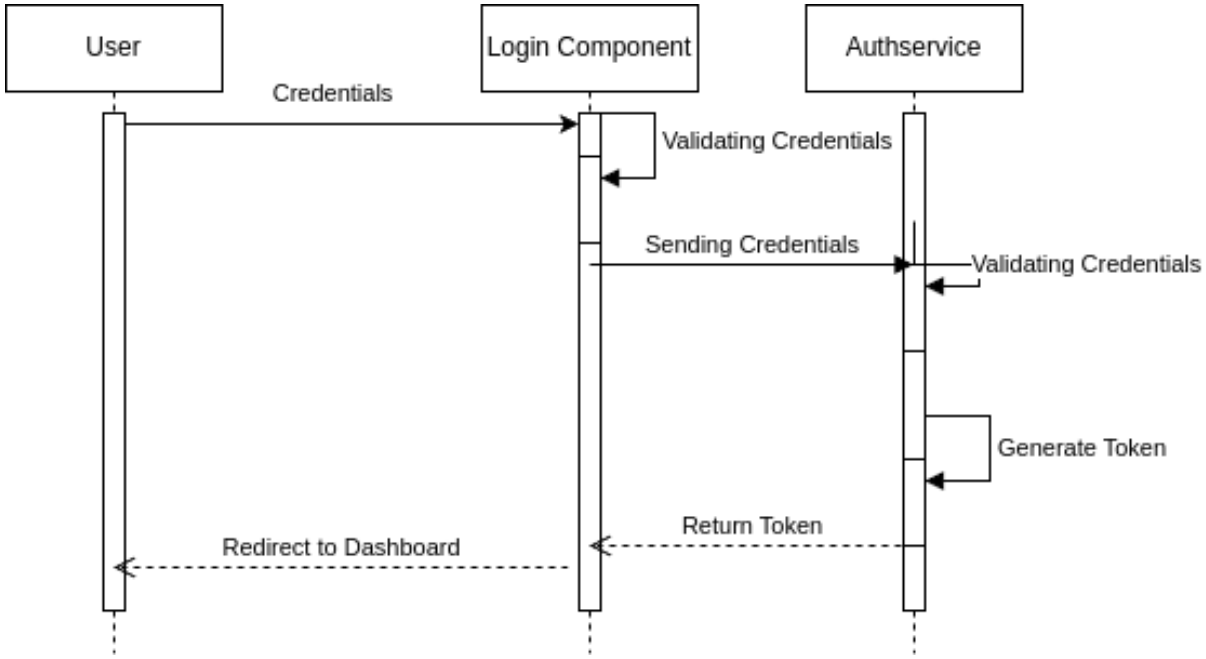

Figure 3. Anmeldung

6.3. Dashboard

Das Dashboard ist die zentrale Ansicht der Anwendung, auf der der Benutzer seine personalisierten Aktieninformationen und -visualisierungen sieht. Die Daten werden über die REST-API vom Backend abgerufen und im Frontend dynamisch dargestellt. Der Benutzer kann Aktien hinzufügen, entfernen und konfigurieren, um sein Dashboard individuell anzupassen.

HINWEIS: Falls die Aktie nicht gefunden wird wird nichts zurückgegeben.

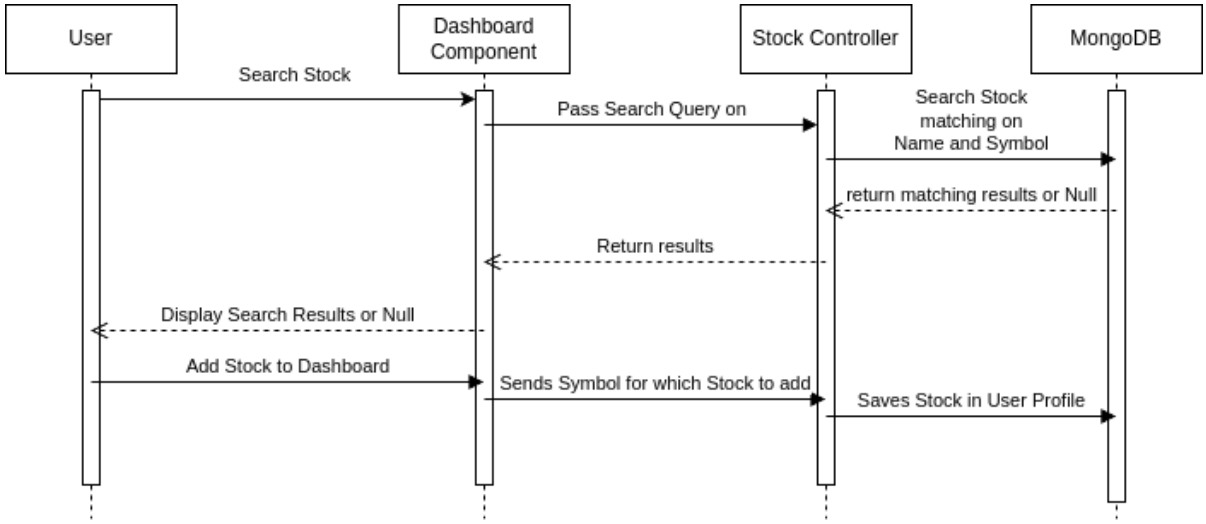

Figure 4. Dashboard

6.3.1. Live-Aktualisierung

Die Aktienkurse werden in Echtzeit aktualisiert, um dem Benutzer stets die aktuellen Informationen zu liefern. Hierfür wird eine WebSocket-Verbindung zwischen Frontend und Backend aufgebaut, über die die Kursdaten in regelmäßigen Abständen übertragen werden. Die Daten werden im Frontend dynamisch aktualisiert, ohne dass der Benutzer die Seite neu laden muss.

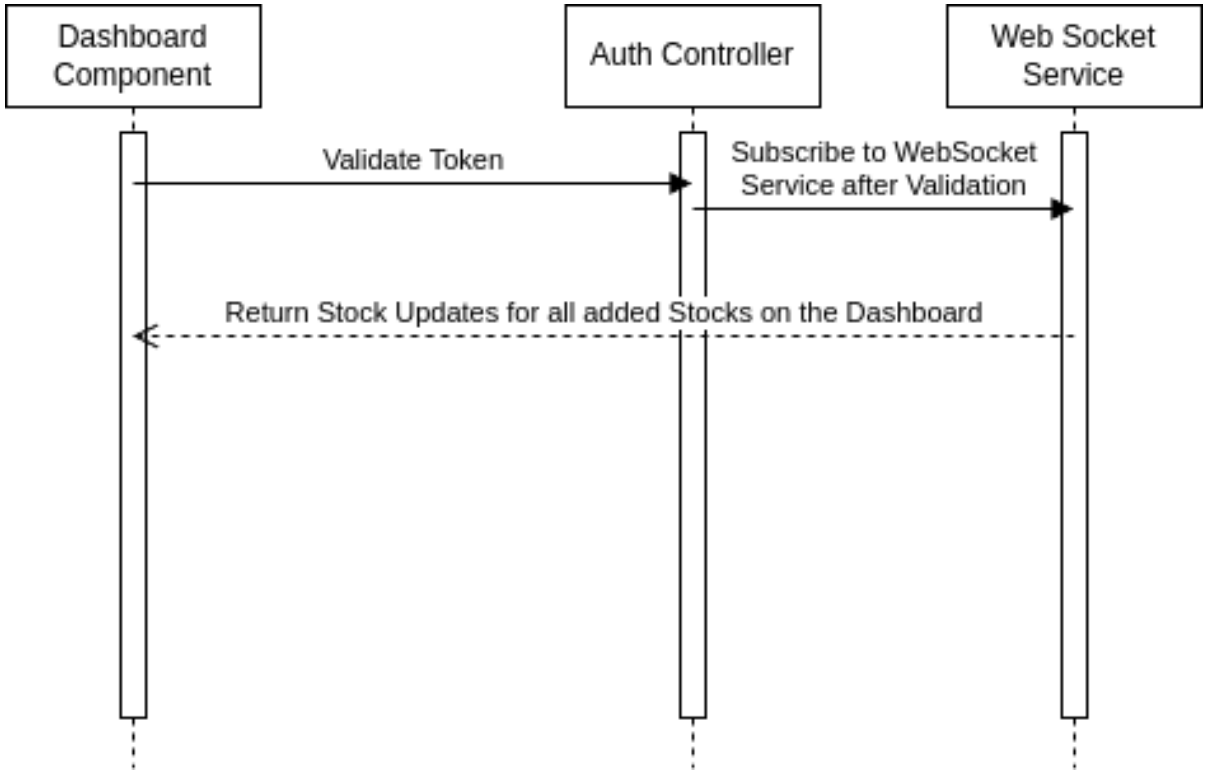

Figure 5. Live-Aktualisierung

Chapter 7. Verteilungsschicht

Die Software wird in drei Schichten aufgeteilt: Frontend, Backend und Datenbank. Die Frontend-Schicht ist für die Darstellung der Benutzeroberfläche zuständig, das Backend für die Geschäftslogik und die Datenbank für die Speicherung der Daten.

7.1. Deployment

Der Build wird auf Github durch Github Actions realisiert. Jedoch wird die Anwendung nicht auf einem Server gehostet, sondern nur lokal betrieben. Grund dafür war das keine Serverressourcen zur Verfügung standen.

7.2. Domänenmodell

Das Domänenmodell ist in drei Schichten aufgeteilt: Frontend, Backend und Datenbank. Die Frontend-Schicht ist für die Darstellung der Benutzeroberfläche zuständig, das Backend für die Geschäftslogik und die Datenbank für die Speicherung der Daten.

7.3. Schnittstellen

Die Kommunikation zwischen den Schichten erfolgt über REST-APIs. Das Frontend sendet HTTP-Anfragen an das Backend, das die Anfragen verarbeitet und die Daten aus der Datenbank abruft. Die Daten werden im JSON-Format übertragen.

API	Beschreibung
GET /api/stocks	Abrufen einer Liste von Aktien
GET /api/stocks/:symbol	Abrufen von Details zu einer bestimmten Aktie
POST /api/user-stocks	Hinzufügen einer neuen Aktie zum Dashboard
GET /api/user-stocks	Abrufen der Aktien im Dashboard
PUT /api/stocks/:symbol	Aktualisieren der Daten einer Aktie
POST /api/auth/register	Registrierung eines neuen Benutzers
POST /api/auth/login	Anmeldung eines Benutzers

Chapter 8. Querschnittliche Konzepte

Querschnittliche Konzepte sind Aspekte, die mehrere Bausteine oder Schichten eines Systems betreffen und nicht in einem einzelnen Baustein oder einer einzelnen Schicht lokalisiert sind. Sie schneiden durch die Architektur und beeinflussen die Implementierung und das Verhalten des Systems. Die folgenden querschnittlichen Konzepte sind in der Architektur des Aktienverwaltungssystems relevant:

8.1. Authentifizierung und Autorisierung

Die Authentifizierung und Autorisierung von Benutzern ist ein querschnittliches Konzept, das die gesamte Anwendung betrifft. Benutzer müssen sich anmelden, um auf das System zuzugreifen, und dürfen nur auf bestimmte Ressourcen zugreifen, wenn sie die erforderlichen Berechtigungen haben. Dieses Konzept umfasst die Verwaltung von Benutzerkonten, die Überprüfung von Anmeldeinformationen, die Ausstellung und Überprüfung von Zugriffstoken sowie die Definition von Rollen und Berechtigungen.

8.2. Logging und Monitoring

Das Logging und Monitoring ist ein querschnittliches Konzept, das die Nachverfolgung von Aktivitäten und das Überwachen des Systemverhaltens umfasst. Durch das Protokollieren von Ereignissen und Metriken können Entwickler und Administratoren das Systemverhalten analysieren, Fehler diagnostizieren, Leistungsprobleme identifizieren und Sicherheitsbedrohungen erkennen. Dieses Konzept beinhaltet die Definition von Protokollierungsrichtlinien, die Implementierung von Protokollierungsmechanismen und das Einrichten von Überwachungssystemen.

Chapter 9. Architekturentscheidungen

In diesem Kapitel werden die wichtigsten Architekturenscheidungen begründet und beschrieben.

9.1. Frontend

Für das Frontend wird Angular verwendet. Dies eignet sich besonders für Single Page Applications. Angular bietet viele Features out-of-the-box, wie z.B. Dependency Injection, Routing, Forms, HTTP Client und Observables. Zudem ist Angular sehr gut dokumentiert und hat eine grosse Community. Dies erleichtert die Einarbeitung und die Fehlersuche.

9.2. Backend

Für das Backend wird Express.js verwendet. Express.js ist ein minimalistisches Webframework für Node.js. Es bietet eine einfache und flexible API für das Erstellen von Webanwendungen und APIs. Express.js ist sehr performant und eignet sich gut für kleine bis mittlere Projekte. Zudem gibt es viele Middleware und Plugins, die die Entwicklung erleichtern.

9.3. Datenbank

Für die Datenbank wird MongoDB verwendet. MongoDB ist eine NoSQL-Datenbank, die sich besonders für unstrukturierte Daten eignet. MongoDB speichert Daten im BSON-Format (Binary JSON) und bietet eine flexible Datenmodellierung. Zudem ist MongoDB sehr performant und skalierbar. Dies eignet sich gut für die Speicherung von Aktiendaten.

9.4. Authentifizierung

Für die Authentifizierung wird JWT (JSON Web Token) verwendet. JWT ist ein sicheres und zustandsloses Authentifizierungsverfahren. Der Benutzer erhält nach der Anmeldung ein Token, das er für alle weiteren Anfragen verwenden muss. Das Token enthält alle nötigen Informationen zur Authentifizierung und ist verschlüsselt. Dies ist sicherer als herkömmliche Session-Cookies.

9.5. Aktien Darstellung

Für die Aktiendarstellung wird D3js verwendet. D3js ist eine JavaScript-Bibliothek zur Erstellung von interaktiven Datenvisualisierungen im Web. D3js bietet viele Funktionen zur Manipulation von DOM-Elementen und zur Erstellung von SVG-Grafiken. Dies eignet sich gut für die Darstellung von Aktiendaten in Form von Charts und Graphen.

Chapter 10. Qualitätsanforderungen

Das Stock Market Dashboard soll folgende Qualitätsanforderungen erfüllen:

Priorität	User Story	Akzeptanzkriterien	Implementiert
Must	Als Benutzer möchte ich mich registrieren/anmelden können auf der Seite.	Der Benutzer kann sich mit einer E-Mail-Adresse, Name und einem Passwort registrieren und anmelden.	Wurde vollständig implementiert.
Must	Als Benutzer möchte ich Aktien meinem Dashboard hinzufügen können welche bei erneuter Anmeldung oder laden der Seite persitent bleiben.	Der Benutzer kann Aktien zu seinem Dashboard hinzufügen und diese bleiben auch nach einem Neuladen der Seite erhalten.	Wurde vollständig implementiert.
Could	Als Benutzer kann ich mein Dashboard beliebig anpassen.	Der Benutzer kann die Anordnung der Aktien auf dem Dashboard anpassen.	Wurde vollständig implementiert.
Should	Als Benutzer erhalte ich live-updates der Aktien ohne das ein neuladen der Webseite benötigt wird	Der Benutzer erhält in Echtzeit die neusten Aktienkurse.	Wurde vollständig implementiert.
Could	Das Dashboard zeigt Aktien als Graphen an	Der Benutzer kann die Aktien als Graphen anzeigen lassen.	Wurde vollständig implementiert.
Should	Als Benutzer kann ich mein User Portfolio und historische Aktien Daten abspeichern können	Der Benutzer kann sein Portfolio abspeichern und historische Aktienkurse einsehen.	Wurde nicht implementiert aufgrund von Zeitmangel.
Should	Als Benutzer kann ich Aktien favorisieren	Der Benutzer kann Aktien favorisieren und diese werden auf dem Dashboard hervorgehoben.	Wurde nicht implementiert daher das Dashboard bereits modifizierbar ist und man Aktien so anordnen kann wie man möchte.

Chapter 11. Risiken und Technische Schulden

11.1. Risiken

- Es wurde keine Teststrategie implementiert sowie auch automatisierte Tests. Dies führt dazu, dass die Software nicht getestet ist und somit Fehler in der Software vorhanden sein können. Grund dafür ist Zeitdruck und der Mangel an Erfarhung mit Testen von Webprojekten.
- Es wurden keine minimalen Passwort Anforderungen implementiert. Dies führt dazu, dass Benutzer unsichere Passwörter verwenden können.
- Es wurde keine Rate Limitierung implementiert. Dies führt dazu, dass ein Benutzer unendlich viele Anfragen an die API senden kann.

11.2. Massnahmen zur Risikoverminderung

- Implementierung von automatisierten Tests
- Implementierung von minimalen Passwort Anforderungen
- Implementierung von Rate Limitierung

11.3. Technische Schulden

- Fehlende automatisierte Test für die Angular Komponenten
- Fehlende automatisierte Test für die Express.js API
- Fehlende Rate Limitierung
- Hardcodierte API Endpoints
- Fehlende Dokumentation
- Fehlende Logging und Monitoring
- MongoDB-Schema ohne ausreichende Validierung

Chapter 12. Glossar

Begriff	Definition	Quelle
Node.js	Node.js ist eine serverseitige Plattform, die auf der JavaScript-Laufzeitumgebung von Chrome basiert.	https://nodejs.org/en
Angular	Angular ist ein TypeScript-basiertes Open-Source-Framework zur Entwicklung von Single-Page-Webanwendungen.	https://angular.dev/
Express.js	Express.js ist ein minimalistisches Webframework für Node.js, das eine einfache und flexible API für die Entwicklung von Webanwendungen und APIs bietet.	https://expressjs.com/
MongoDB	MongoDB ist eine dokumentenorientierte NoSQL-Datenbank, die sich besonders für die Speicherung unstrukturierter Daten eignet.	https://www.mongodb.com/
Docker	Docker ist eine Open-Source-Plattform zur Automatisierung der Bereitstellung von Anwendungen in Containern.	https://www.docker.com/
JWT	JSON Web Token (JWT) ist ein offener Standard (RFC 7519) zur sicheren Übertragung von Informationen zwischen Parteien als JSON-Objekt.	https://jwt.io/

Begriff	Definition	Quelle
RxJS	RxJS ist eine JavaScript-Bibliothek zur Programmierung mit asynchronen Datenströmen und Ereignissen.	https://rxjs.dev/
Jasmine	Jasmine ist ein JavaScript-Testframework für die Entwicklung von Unit- und Integrationstests.	https://jasmine.github.io/
D3js	D3.js ist eine JavaScript-Bibliothek zur Erstellung von interaktiven Datenvisualisierungen im Web.	https://d3js.org/