Projektdarstellung: StockBroker

Informatik Q12
Laurens Hackenjos

1. Einleitung

StockBroker ist eine webbasierte Börsensimulation.

Benutzer können sich registrieren, mit einem virtuellen Kapital an der Börse handeln, ihr Portfolio verwalten und sich auf einer Rangliste mit anderen messen. Die Anwendung integriert externe APIs zur Abfrage von Echtzeit-Börsendaten und zur Generierung von Kurs-Charts. Sie wird lokal gehostet.

2. Kern Features

2.1 User Stories zur Definition der Anforderungen. Definieren des Projektziels

Vorlage:
Ich als [WER = ROLLE/PERSONA],
möchte [WAS = TUN/ZIEL],
um [WARUM=GRUND/NUTZEN]

1. Als Aktionär will ich nach Wertpapieren **suchen**, um mir deren Kursverlauf, möglichst in Echtzeit, anzeigen zu lassen.

2. Als Aktionär will ich ein **bestimmtes Wertpapier auswählen, um es zu kaufen**.

3. Als Aktionär will ich **Wertpapiere** in mein Musterdepot **kaufen**, um unter “Simulationsbedingungen” und damit ohne negative finanzielle Auswirkungen den Wertpapierhandel zu üben.

4. Als Aktionär will ich Wertpapiere aus meinem Musterdepot **verkaufen**, um Gewinne zu realisieren bzw. Verluste zu stoppen und wieder “Geld” für weiteren Handel freizusetzen.

5. Als Aktionär will ich **in einem eigenen passwortgeschützten Bereich arbeiten** können, um meine finanzielle Privatsphäre zu schützen.

6. Als Aktionär will ich mein **Musterdepot** mit den von mir gekauften Wertpapieren und Informationen zur bisherigen Entwicklung der einzelnen Werte und der Performance des Depots insgesamt **einsehen**, um in Abhängigkeit davon weitere

Kauf-/Verkaufsentscheidungen zu treffen.

7. Als Aktionär will ich meine **Profildaten** nachträglich **ändern** können, um sie an neue Gegebenheiten wie zb. eine geänderte Emailadresse anzupassen.

8. Als Aktionär wünsche ich mir ein geordnetes Verfahren bei der **Registrierung**, um zu vermeiden, dass sich Fremde mit meiner Emailadresse auf der Webseite registrieren.

9. Als Musterdepot-Aktionär will ich mich mit anderen “Spielern” in einer **Rangliste** der besten Broker messen, um mein Bedürfnis nach Wettbewerb zu befriedigen.

10. Als Aktionär will ich beim Handeln **zwischen verschiedenen gängigen Handelsoptionen wählen (Market Buy, Market Sell, Limit Buy, Limit Sale, Stop Loss)** können, um mein Risiko zu minimieren bzw. meinen Gewinn zu maximieren.

11. Als verspielter Schüler-Aktionär mit einem Musterdepot will ich beim Benutzen des Programms **Spass haben**, damit das Thema nicht zu ernst wird.

12. Als vergesslicher Schüler-Aktionär will ich die Möglichkeit haben, mein **Passwort zurückzusetzen**, um wieder Zugriff auf mein Depot zu haben.

13. Als Benutzer mit Geltungsbedürfnis will ich mein **Instagram Profil verlinken** können, so dass andere Benutzer sehen, wie krass cool ich bin.

2.2 Benutzerauthentifizierung

Das System verfügt über ein vollständiges Authentifizierungsmodul, das die grundlegenden Sicherheitsanforderungen erfüllt. Es ist an machen Stellen etwas overengineered.

- **Registrierung & Verifizierung:** Neue Benutzer können ein Konto mit einem eindeutigen Benutzernamen und einer E-Mail-Adresse erstellen. Passwörter werden nicht im Klartext, sondern **sicher gehasht mit einem individuellen Salt** gespeichert, um die Sicherheit zu erhöhen. Die Sicherheit der Passwörter ist somit nicht mehr abhängig von der Länge der Passwörter. Nach der Registrierung wird eine **Bestätigungs-E-Mail mit einem zeitlich begrenzten Token** versendet, um die E-Mail-Adresse zu verifizieren und den Account zu aktivieren.
- **Login:** Angemeldete Benutzer werden über eine **Flask-Session** verwaltet. Der Login kann sowohl mit dem Benutzernamen als auch mit der E-Mail-Adresse erfolgen.
- **Passwort-Reset:** Benutzer, die ihr Passwort vergessen haben, können einen Reset-Prozess anstoßen. Sie erhalten per E-Mail einen Code, mit dem sie ein neues Passwort festlegen können.

- **Logout:** Beendet die aktuelle Session des Benutzers.

2.3 Aktiensuche und -darstellung

Benutzer können nach Aktien suchen und detaillierte Informationen sowie historische Kursverläufe einsehen.

- **Aktiensuche:** Die Suche wird über die **Alpha Vantage API** realisiert, um Aktien anhand von Stichwörtern oder Tickersymbolen zu finden. Die Suchergebnisse zeigen an, ob eine Aktie über die yfinance-Bibliothek handelbar ist.
- **Detailseite:** Für jede Aktie kann eine Detailseite aufgerufen werden. Hier werden fundamentale Unternehmensdaten, ein Kurzprofil, Kennzahlen und Analystenempfehlungen über die **yfinance-Bibliothek** geladen und angezeigt.
- **Dynamische Kurs-Charts:** Das Herzstück der Detailseite ist ein interaktiver Candlestick-Chart, der mit der **Plotly-Bibliothek** generiert wird. Benutzer können den angezeigten Zeitraum (z.B. 5 Tage, 1 Jahr, Maximal) und die Datenqualität (Intervall der Datenpunkte) flexibel anpassen.

2.4 Handelssystem (Trading)

Das Handelssystem ermöglicht den Kauf und Verkauf von Aktien durch verschiedene Order-Typen.

- **Order-Typen:** Das System unterstützt:
  - **Market-Orders (Buy/Sell):** Werden sofort zum bestmöglichen aktuellen Marktpreis ausgeführt.
  - **Limit-Orders (Buy/Sell):** Werden nur ausgeführt, wenn der Marktpreis ein vordefiniertes Limit erreicht oder über-/unterschreitet. Dies gibt dem Benutzer Kontrolle über den Ausführungspreis.
  - **Stop-Loss-Orders (Sell):** Eine Verkaufsorder, die automatisch ausgelöst wird, wenn der Kurs auf einen bestimmten Stop-Preis fällt, um Verluste zu begrenzen.
  - Optionsscheine aufgrund der Komplexität der Generierung nur geplant
- **Orderverarbeitung:** Offene Orders (Limit/Stop) werden nicht sofort ausgeführt, sondern in der orders-Tabelle in der Datenbank gespeichert. Ein **periodisch minütlicher Hintergrundprozess (Scheduler)** überprüft regelmäßig alle offenen Orders und führt sie aus, sobald deren Bedingungen erfüllt sind. Dies ist der beste Kompromiss zwischen Realitätsnähe und Performence/Umsetzbarkeit.

2.5 Depot- und Vermögensverwaltung

Jeder Benutzer hat ein eigenes Depot, das sein aktuelles Vermögen und seine Aktienpositionen darstellt.

- **Depot-Ansicht:** Die Depot-Seite (/dashboard) zeigt eine Übersicht über den Barbestand, den Wert des Aktienportfolios und das Gesamtvermögen. Für jede gehaltene Aktie werden die Menge, der durchschnittliche Kaufpreis und der aktuelle Gewinn oder Verlust (absolut und prozentual) berechnet und angezeigt.
- **Vermögens-Chart:** Ein Liniendiagramm visualisiert die Entwicklung des Gesamtvermögens über die Zeit. Die Daten hierfür stammen aus der leaderboard-Tabelle, in der historische Vermögensstände gespeichert werden. Um die Tabelle nicht zu überfüllen, werden die Einträge pro User auf einen festgelegten Wert, aktuell 1000, dezimiert. Dazu werden zuerst gleiche Einträge gelöscht, danach immer der Punkt, der den geringsten zeitlichen Abstand zum nächsten hat.
- **Caching für Performance:** Die auf der Startseite angezeigten Charts der beliebtesten Aktien werden im Hintergrund vom **Daily-Scheduler** generiert und in der cached_charts-Tabelle als HTML zwischengespeichert, um die Ladezeiten für alle stark Benutzer zu verkürzen. Denn so kann auf Anfragen an yfinance beim laden der Seite verzichtet werden.
- **Instagram verlinken:** damit die User sehen können, wer hinter einem Benutzernamen im Leadderboard steckt, können sie ihr Instagram-Profil in den Einstellungen verlinken. Die ersetzt ein aufwendig zu implementierendes internes Soziales Netzwerk oder Profilsystem.

2.6 Leaderboard

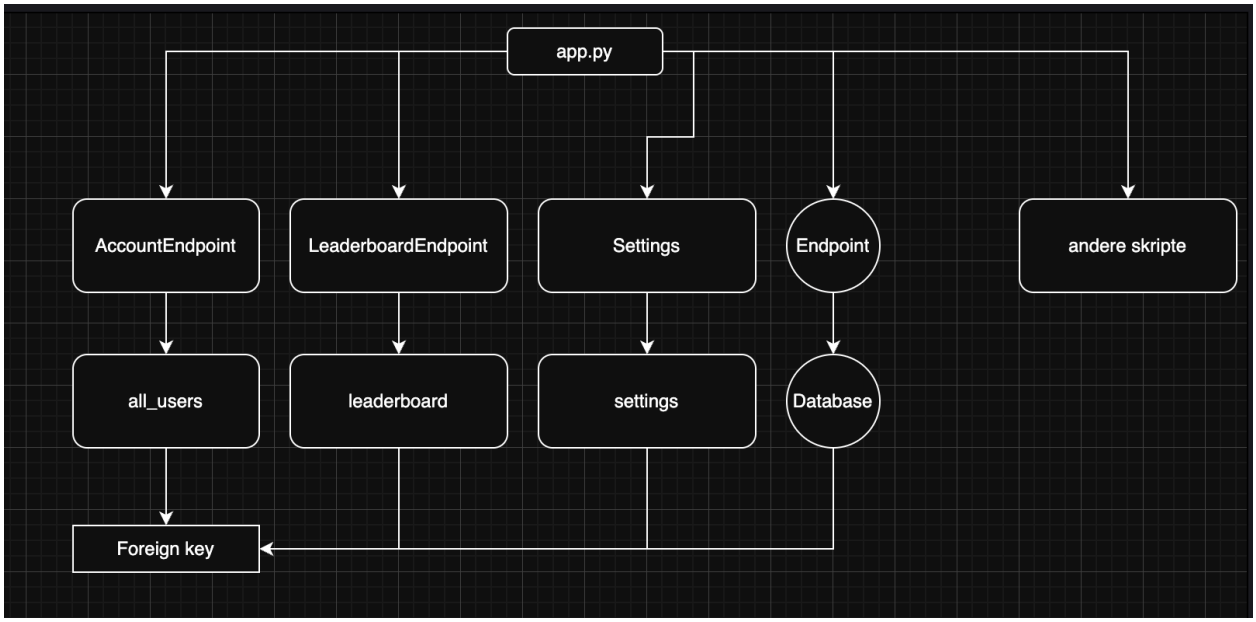
Ein globales Leaderboard schafft das kompetitives Element, indem es alle Benutzer nach ihrem Gesamtvermögen ordnet.

- **Ranking:** Die Rangliste ist paginiert und zeigt die Benutzer in absteigender Reihenfolge ihres Vermögens an.
- **Automatische Aktualisierung:** Das Gesamtvermögen jedes Benutzers wird durch den **10-Minutes-Scheduler** in regelmäßigen Abständen neu berechnet und in der leaderboard-Tabelle gespeichert, um die Rangliste aktuell zu halten. Die Historie dieser Werte werden für den Depot-History-Chart verwendet.

3. Architektur und Design Patterns

- **Model-View-Controller (ähnlich):**
  - **Controller:** Die app.py nimmt HTTP-Anfragen entgegen, verarbeitet die Eingaben und ruft die entsprechenden Funktionen auf. Sie entscheidet, welche Antwort an den Client zurückgesendet wird.
  - **View:** Die HTML-Templates und der Browser sind für die Darstellung der Daten zuständig.
  - **Model:** Die Logik und die Datenmodelle sind in den Python-Dateien im backend-Ordner gekapselt (z.B. accounts_to_database.py, trading.py). Sie enthalten die Abläufe und interagieren mit der Datenbank.
- **Facade Pattern:** Dieses Muster wird in den Endpoint-Klassen (AccountEndpoint, TradingEndpoint, LeaderboardEndpoint, DepotEndpoint) verwendet. Diese Klassen bieten eine vereinfachte, einheitliche Schnittstelle zu komplexeren Subsystemen. Beispielsweise kapselt AccountEndpoint.create_account die Logik für die Validierung, das Hashen von Passwörtern, die Datenbankinteraktion und das Senden von E-Mails.
- **Decorator Pattern:** Das @login_required Decorator-Muster wird in app.py verwendet, um Routen zu schützen. Es fügt Funktionen eine zusätzliche Verantwortlichkeit (Überprüfung der Benutzer-Session) hinzu, ohne deren Code direkt zu verändern und erhöht die lesbarkeit extrem
- **Observer / Scheduler Pattern:** Die gunicorn.conf.py in Verbindung mit der apscheduler-Bibliothek implementiert dieses Muster. Ein Scheduler "beobachtet" die Zeit und benachrichtigt registrierte "Observer" (die Jobs in jobs.py), wenn bestimmte Zeitpunkte erreicht sind (z.B. jede Minute, jede Nacht um 5:00 Uhr).
- Caching Pattern: Die Funktion get_or_generate_widget_chart in app.py implementiert eine Caching-Strategie. Bevor ein ressourcenintensiver Chart neu generiert wird, prüft die Funktion, ob bereits eine aktuelle Version in der cached_charts-Tabelle existiert. Dies reduziert die Serverlast und beschleunigt die Antwortzeiten.
- Template Method Pattern: Das base.html dient als Skelett für alle anderen Seiten. Es definiert die allgemeine Seitenstruktur (Header, Footer, Navigation), während Unter-Templates wie login.html oder depot.html nur die spezifischen Inhaltsblöcke ({% block content %}) füllen.

- **Data Access Object (DAO) / Repository Pattern (Abstraktion):** Die Endpoint-Klassen agieren auch als eine Form von DAOs. Sie entkoppeln die Anwendungslogik in app.py von der Logik, den direkten SQL-Abfragen und der Datenbankstruktur. Änderungen am Datenbankschema müssen primär nur in den Endpoint-Klassen vorgenommen werden.



Jeder Zugriff auf eine Tabelle läuft über den entsprechenden Endpunkt. Jeder Endpunkt greift nur auf seine Tabelle zu. Manche Endpunkte greifen auf keine Tabelle zu.
Diese klare Struktur hat die Entwicklung extrem vereinfacht, da Fehler und Zuständigkeiten direkt auf einen Ort zurückgeführt werden können.

4. Datenbankstruktur

Die Struktur ist in database_setup.py definiert und folgt dem relationalen **Parent-Child-Modell**. Die Tabelle all_users dient als Parent-Tabelle. Die anderen Tabellen sind **Child-Tabellen**, die über den **Fremdschlüssel** user_id_fk auf den **Primärschlüssel** user_id der all_users-Tabelle verweisen. Dieses stellt die Datenintegrität sicher. Besonders stolz sind wir auf die Verwendung von ON DELETE CASCADE bei der Definition der Fremdschlüssel. Dies sorgt dafür, dass beim Löschen eines Benutzers automatisch alle zugehörigen Daten (seine Einstellungen, Orders, Depotbestände etc.) ebenfalls aus der Datenbank entfernt werden, was die Datenkonsistenz wahrt. Allerdings gibt es für den Nutzer direkt noch keine Methode, sich zu löschen. Dies ist bisher nur per SQL-Befehl möglich

5. Deployment-Struktur auf dem Raspberry Pi

- Die Anwendung ist für den Einsatz auf einem Raspberry Pi konzipiert. Die Architektur verwendet eine Kombination aus Gunicorn, NGINX und Certbot, um eine stabile, sichere und über das Internet erreichbare Webanwendung bereitzustellen.

- Gunicorn (WSGI Application Server): Anstatt den eingebauten Flask-Entwicklungsserver zu verwenden, wird die Applikation über Gunicorn ausgeführt. Gunicorn ist ein robuster WSGI-Server, der die Python-Anwendung effizient ausführt und mehrere Worker-Prozesse verwalten kann. Die Konfigurationsdatei gunicorn.conf.py wird außerdem genutzt, um die periodischen Hintergrundjobs mittels apscheduler zu starten, die für die Verarbeitung von Orders und die Aktualisierung des Leaderboards zuständig

sind.

- NGINX (Reverse Proxy): Vor Gunicorn wird NGINX als Reverse Proxy geschaltet. NGINX nimmt die Anfragen aus dem öffentlichen Internet auf den Standard-Ports 80 (HTTP) und 443 (HTTPS) entgegen und leitet sie an den intern laufenden Gunicorn-Server weiter.

- SSL-Terminierung: NGINX übernimmt die Ver- und Entschlüsselung des HTTPS-Traffics.

- No-IP (Dynamic DNS): Auf dem Raspberry Pi ist der No-Ip-.service installiert, der jede Minute die öffentliche IP-Adresse überprüft und an den No-IP-Server weitergibt, um die Weiterleitung von der Domain zu ermöglichen.

- Certbot (Let's Encrypt): Das SSL/TLS-Zertifikat für die HTTPS-Verschlüsselung wird mit Certbot verwaltet, einem Tool, das kostenlose Zertifikate von Let's Encrypt automatisch anfordert, installiert und erneuert.

- UFW und Port Forwarding:

    - Zur Absicherung des Raspberry Pi wird eine Firewall wie UFW (Uncomplicated Firewall) eingesetzt. Diese ist so konfiguriert, dass sie nur den notwendigen eingehenden Verkehr für NGINX erlaubt.

    - Damit die Anfragen aus dem Internet den Raspberry Pi im lokalen Netzwerk erreichen, musste im Router Port Forwarding eingerichtet werden.

6. Danke an die hilfreichen Websites

6.1 SQL

- foreign Key https://www.w3schools.com/sql/sql_foreignkey.asp
- On Delete cascade https://help.hcl-software.com/onedb/1.0.1/sqs/ids_sqs_0304.html

…