ETF-APP

DIPLOMARBEIT

Höhere Abteilung für Informatik

07/03/2024 – 03/04/2025

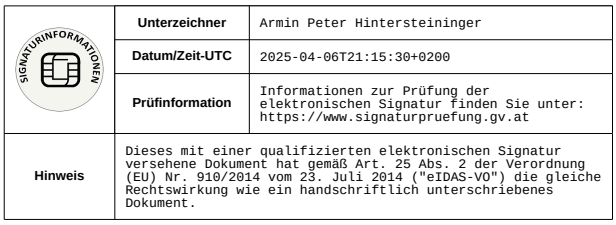
Projektmitglieder: Armin Hintersteininger
Jan Oppitz
Gabriele Praher
Adrian Tagwerker

Betreuer:in: Ing. Patrick Praher, MSc

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung der Arbeit haben wir das generative KI-Tool ChatGPT zur grammatikalischen Korrektur und Rechtschreibkontrolle verwendet.



Danksagung

Wir bedanken uns bei sämtlichen Personen der HTL Perg, die diese Diplomarbeit ermöglicht haben.

Besonderer Dank gilt unserem Diplomarbeitsbetreuer Herrn Professor Ing. Patrick Praher, MSc, der uns sowohl im theoretischen als auch im praktischen Teil dieser Arbeit mit großem Engagement unterstützt hat.

Wir möchten uns außerdem herzlich bei Uni Software Plus für die Möglichkeit bedanken, diese Diplomarbeit in Kooperation mit einem externen Partner durchzuführen. Ein besonderer Dank gilt dabei Herrn Simon Primetzhofer, der uns während der gesamten Projektlaufzeit als kompetenter Ansprechpartner zur Seite stand und uns mit wertvollen fachlichen Inputs unterstützt hat.

Abstract

The aim of this diploma thesis was to develop a modern, cross-platform application for the analysis and visualization of Exchange Traded Funds (ETFs). The application enables users to import structured ETF data, filter it based on various criteria, and display it in an intuitive and interactive way. The focus was placed on usability, technical scalability, and efficient handling of large datasets.

The backend was developed using Java Spring Boot and provides all core functionalities through a REST API. This includes managing, filtering, and analyzing ETF data, as well as importing structured XML files. PostgreSQL was used as the database system, supported by extensive unit tests written with JUnit.

The frontend was implemented as a Progressive Web App (PWA) using React Native, allowing for a seamless user experience across both desktop and mobile devices. Interactive data visualizations were integrated using Recharts, enabling detailed insights into individual ETF values.

As part of the project, multiple frontend frameworks (React Native, Angular, Flutter, Ionic) were evaluated, with React Native ultimately chosen based on performance and development efficiency.

The collaboration with Uni Software Plus and the supervision of Simon Primetzhofer provided practical insights and real-world relevance throughout the development process.

The result is a fully functional and extensible application that can serve as a foundation for future data-driven financial tools.

Zusammenfassung

Ziel dieser Diplomarbeit war die Entwicklung einer modernen, plattformübergreifenden Applikation zur Analyse und Darstellung von ETFs (Exchange Traded Funds). Die Anwendung soll Nutzer:innen ermöglichen, strukturierte ETF-Daten zu importieren, zu filtern und grafisch auszuwerten. Ein besonderer Fokus lag auf Benutzerfreundlichkeit, technischer Skalierbarkeit und der effizienten Verarbeitung großer Datenmengen.

Das System besteht aus mehreren zentralen Komponenten: Die Backend-Logik wurde mittels Java Spring Boot realisiert und stellt über eine REST-API sämtliche Funktionen zur Verfügung. Diese umfasst unter anderem das Verwalten, Filtern und Analysieren der ETF-Daten sowie den Import strukturierter XML-Dateien. Zur Datenpersistenz wurde PostgreSQL eingesetzt. Ergänzt wird das Backend durch umfangreiche Unit Tests mittels JUnit.

Das Frontend wurde als plattformunabhängige Progressive Web App (PWA) mit React Native umgesetzt. Dieses ermöglicht eine ansprechende und intuitive Benutzeroberfläche, die sowohl auf mobilen Geräten als auch am Desktop verwendet werden kann. Für die Visualisierung wurden interaktive Diagramme mit Recharts integriert, die eine detaillierte Analyse einzelner ETF-Werte erlauben.

Ein zentraler Bestandteil war außerdem der Vergleich verschiedener Frontend-Frameworks (React Native, Angular, Flutter, Ionic), auf dessen Basis eine fundierte Entscheidung für React Native getroffen wurde.

Die Zusammenarbeit mit Uni Software Plus und die Betreuung durch Simon Primetzhofer ermöglichte eine praxisnahe Umsetzung und wertvolle Einblicke in reale Softwareentwicklungsprozesse.

Das Ergebnis ist eine voll funktionsfähige Applikation, die flexibel erweiterbar ist und als Grundlage für weitere datenbasierte Finanzanwendungen dienen kann.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das Projekt ËTF-Appïst ins Leben gerufen worden, um Daten über ETFs in einer Datenbank speichern zu können und auf diese über eine API zugreifen zu können. Es handelt sich dabei nicht um eine fertige Anwendung, sondern nur um einen Teil eines größeren Projekts des Unternehmens. Das vollständige Projekt wurde von einer Schweizer Bank in Auftrag gegeben.

1.2 Zielsetzung

Das fertige Projekt soll aus vier Teilbereichen bestehen, welche im Folgenden näher beschrieben werden:

- Zunächst soll aus den derzeit am häufigsten verwendeten Frontend-Frameworks ein geeignetes Framework im Rahmen einer Evaluierungsphase ausgewählt werden. Auf Basis dessen wird ein Prototyp entwickelt, der eine Übersicht aller ETFs anzeigt und zudem detaillierte Informationen zu einzelnen ETFs bereitstellt.
- Eine Datenbank, die die Datenstruktur der ETFs abbildet und sämtliche Informationen speichert. Dabei sollen alle in der XML-Datei enthaltenen Daten übernommen werden, ohne dass Informationen verloren gehen.
- Ein Programm, das alle relevanten Daten aus umfangreichen XML-Files, die jeweils ein ETF darstellen, ausliest und in Java-Objekten speichert. Danach sollen diese Daten in der zuvor erstellten Datenbank gespeichert werden.
- Die ursprüngliche Aufgabenstellung der Firma für das Backend lautete: „Ein Java-Spring-REST-Backend, das Daten im JSON-Format zur Verfügung stellt, sowie die Implementierung mehrerer Unit-Tests.“ Im Laufe mehrerer Besprechungen kristallisierte sich heraus, dass eine API entwickelt werden soll, welche häufig veränderte Daten des Risk Services mit eher statischen Daten aus einer Datenbank kombiniert.

Abbildung 1: Logo der Uni Software plus GmbH

1.3 Projektumfeld

1.3.1 Projektteam

Das Projektteam besteht aus den Mitgliedern Adrian Tagwerker, Armin Hintersteininger, Gabrielle Praher und Jan Oppitz – alle vier sind Schüler der HTL Perg. Aufgrund des gemeinsamen Interesses an der Finanzwelt wurde dieses Thema als Grundlage für unsere Diplomarbeit gewählt.

1.3.2 Betreuung

Die Diplomarbeit wurde im technischen und theoretischen Teil durch Ing. Patrick Praher, MSc., mit seinem Wissen über Algorithmen und API-Programmierung unterstützt.

1.3.3 Auftraggeber

Unser Auftraggeber ist die Uni Software Plus GmbH mit Sitz in Perg. Dieses Unternehmen hat es sich zur Aufgabe gemacht, Softwarelösungen für komplexe Probleme zu entwickeln. Es spezialisiert sich auf die Bereiche Analytik, Simulation und Modellierung. Mithilfe eines großen, engagierten Teams werden für den Kunden maßgeschneiderte Anwendungen in der Finanz- und Industriebranche entwickelt. Die Betreuung im Unternehmen wurde maßgeblich von Simon Primetzhofer übernommen, der als Software Engineer tätig ist.

2 Theoretische und fachpraktische Grundlagen und Methoden

2.1 Grundlegende Fachbegriffe

2.1.1 Microservices

2.1.2 Rest API

REST wurde im Jahr 2000 von Roy Fielding entwickelt und ist jetzt einer der wichtigsten Systeme für das moderne Internet. REST (Representational State Transfer) ist ein Architekturstil für verteilte Systeme, der auf der Idee der Ressourcenmodellierung basiert. Es geht um die Verteilung von Ressourcen, die durch eindeutige URIs identifiziert und über standardisierte Operationen manipuliert werden.REST basiert nicht grundlegend auf HTTP doch in der Regel wird es mit HTTP benützt. Die Bekanntesten Operationen, welche von REST und HTTP hervorkommen sind. Sie sind auch standardisiert und jedes Computer kann sie:

1. **GET** um Ressourcen abzufragen oder lesen
2. **POST** um Ressourcen hinzuzufügen
3. **UPDATE** um schon existierende Ressourcen zu verändern
4. **DELETE** um Ressourcen wieder zu löschen

RESTful bedeutet in diesem Kontext, dass der Server keine persistente Verbindung zum Client aufbaut und keinen Sitzungsstatus speichert. Jede Anfrage enthält alle notwendigen Informationen, sodass der Server sie unabhängig von vorherigen Anfragen verarbeiten kann. Der Client erhält den Status seiner Anfrage ausschließlich über die standardisierten HTTP-Statuscodes. REST kommuniziert in JSON (Java Script Object Notation). [1]

2.1.3 Design Patterns

Design Patterns ist eine Problemlösung für ein oft aufkommendes Problem. Ein Design Pattern beschreibt wie dieses Problem in einer bestimmten Software löst. [2]

2.1.4 Cross Plattform Frontend Framework

Ein Cross-Platform Frontend Framework ist ein Entwicklungsansatz, der es ermöglicht, mit einer einzigen Codebasis Anwendungen für mehrere Plattformen wie Webbrowser, Android, iOS, Windows oder macOS zu erstellen. Anstatt separate Anwendungen für jede Plattform zu entwickeln, beispielsweise eine native iOS-App in Swift und eine Android-App in Kotlin, nutzen Entwickler ein Framework, das den Code übersetzt oder abstrahiert, sodass er auf unterschiedlichen Systemen funktioniert. Ziel ist es, Entwicklungsaufwand, Zeit und Kosten zu reduzieren, ohne dabei auf Benutzerfreundlichkeit oder Performance zu verzichten. [3]

Diese Frameworks bieten meist eigene UI-Komponenten, State-Management-Systeme und Schnittstellen zu nativen Funktionen wie Kamera, GPS oder Push-Benachrichtigungen. Sie übersetzen die App-Logik in plattformspezifischen Code oder stellen sogenannte "Bridges"bereit, die den Zugriff auf native APIs ermöglichen. Dabei entstehen hybride oder sogar nahezu native Benutzererfahrungen. [4]

Ein Vorteil dieser Frameworks ist die Wiederverwendbarkeit von Code, was nicht nur Entwicklungsressourcen spart, sondern auch die Wartung erleichtert. Änderungen oder Bugfixes müssen nur an einer Stelle vorgenommen werden. Gleichzeitig wird durch Cross-Platform-Entwicklung die Time-to-Market verkürzt, da parallele Entwicklungen für verschiedene Plattformen wegfallen. [5]

Jedoch gibt es auch Herausforderungen: Manche Plattform-spezifische Funktionen lassen sich nur schwer integrieren oder benötigen separate native Module. Auch die Performance kann in bestimmten Szenarien, vor allem bei grafikintensiven Anwendungen, hinter nativen Lösungen zurückbleiben. Daher ist bei der Auswahl des passenden Frameworks stets eine Abwägung zwischen Flexibilität, Performance, Community-Support und Komplexität nötig. [6][7]

2.1.5 PWA

Eine Progressive Web App (PWA) ist eine moderne Art von Webanwendung, die darauf ausgelegt ist, die Vorteile klassischer Websites mit denen nativer mobiler Apps zu vereinen. Sie wird mit Standard-Webtechnologien wie HTML, CSS und JavaScript entwickelt, kann jedoch, dank spezieller Funktionen – wie eine native App auf dem Smartphone oder Desktop erscheinen und funktionieren. [8]

PWAs zeichnen sich vor allem durch drei Kernmerkmale aus: Zuverlässigkeit, Geschwindigkeit und Benutzerfreundlichkeit. Sie funktionieren auch bei schlechter oder fehlender Internetverbindung, sind schnell in der Nutzung und bieten ein app-ähnliches Erlebnis, inklusive Funktionen

wie Push-Benachrichtigungen und Installation auf dem Startbildschirm. Durch sogenannte Service Worker, Skripte, die im Hintergrund laufen, kann die App Inhalte cachen, Synchronisation im Hintergrund durchführen und auf verschiedene Ereignisse reagieren, selbst wenn die App gerade nicht aktiv ist. [8] [9]

Ein weiterer Vorteil ist die Plattformunabhängigkeit: Eine einzige Codebasis kann auf verschiedenen Geräten und Betriebssystemen (Android, iOS, Windows, macOS etc.) laufen, ohne dass mehrere native Apps entwickelt werden müssen. [10] Die Installation erfolgt direkt über den Browser, was die Verbreitung erleichtert. PWAs sind sicher, da sie über HTTPS bereitgestellt werden müssen, und sie können regelmäßig und automatisch aktualisiert werden. [11]

2.1.6 Expo

Expo ist ein Open-Source-Framework und eine Plattform, die die Entwicklung mit React Native vereinfacht und beschleunigt. Es bietet eine Sammlung von Tools, Bibliotheken und Services, mit denen sich mobile Anwendungen für iOS und Android schreiben, testen und veröffentlichen lassen und das alles mit TypeScript, ohne dass native Codekenntnisse erforderlich sind. [12][13]

Mit Expo kann man Apps direkt im Browser oder in der Expo Go App auf dem Smartphone testen, ohne Xcode oder Android Studio installieren zu müssen. Es liefert außerdem APIs für häufig genutzte Funktionen wie Kamera, Push-Benachrichtigungen, Standortdienste und mehr. [12][13]

Expo eignet sich besonders für Einsteiger und Rapid Prototyping, lässt sich aber auch in bestehende React Native-Projekte integrieren, wenn mehr Kontrolle über native Module benötigt wird.

[12][13]

2.1.7 Objekt relationales Modell

Das objekt-relationale Modell ist eine Kombination aus folgenden zwei Datenbank-Modellen:

Relationales Modell: Dieses Modell basiert auf einer tabellenorientierten Struktur und organisiert Daten in Relationen (Tabellen), Tupeln (Zeilen) und Attributen (Spalten). Die Verwaltung der Daten erfolgt über ein Datenbankmanagementsystem (DBMS), das häufig SQL (Structured Query Language) zur Datenmanipulation und Abfrage verwendet. Das relationale Modell bietet eine hohe Datenintegrität und ermöglicht effiziente Abfragen durch relationale Verknüpfungen.

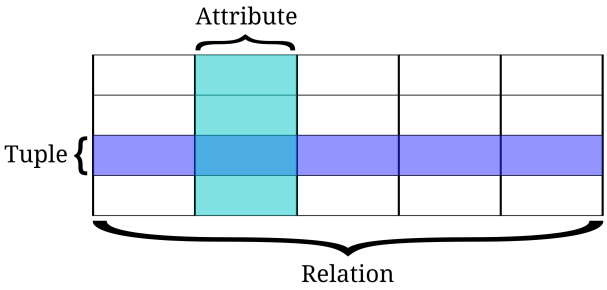


Abbildung 2: relational datamodel

Objektorientiertes Modell: Dieses Modell basiert auf Konzepten der objektorientierten Programmierung, wie sie beispielsweise in Programmiersprachen wie Java oder C++ verwendet werden. Es ermöglicht die Definition von Klassen, die Eigenschaften und Methoden enthalten, sowie die Unterstützung von Vererbung und Kapselung. Dadurch wird eine engere Integration zwischen Datenbankstrukturen und objektorientierten Anwendungen ermöglicht.

[?] [14] [15]

2.1.8 ACID Kriterien

Die ACID-Kriterien garantieren, dass Transaktionen die Datenbankkonsistenz nicht gefährden. Wenn eine Transaktion ausgeführt wird, dann gibt es zwei Möglichkeiten. Sie kann durchlaufen und alles erfüllen, somit wird die Datenkonsistenz nicht beschädigt und die zweite Option ist, dass sie nicht durchläuft, in dem Fall wird die ganze Aktion abgebrochen und die Schritte zurückgeführt (Rollback). ACID steht für atomicity, consistency, isolation und durability. Die Begriffe stehen jeweils für einen Faktor der erfüllt werden muss.

- atomicity (Atomarität): Jeder Befehl in einer Transaktion wird einzeln bearbeitet. Sobald ein Befehl der Transaktion nicht erfolgreich durchgeführt wird, bricht die ganze Transaktion ab und alle Schritte werden Rückgängig gemacht (Rollback).
- consistency (Konsistenz): Transaktionen werden nach bestimmten Regeln ausgeführt, so dass keine falsche Daten ins System kommen. Also man bringt die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand.
- isolation (Isolation): Wenn mehrer Benutzer auf die gleichen Datensätze anfragen, dann kommen sich diese Anfragen nicht in die Quere. Und die Zwischenschritte sind für andere nicht sichtbar. Man überweist 100€ von Konto A auf Konto B. Während dieser Transaktion steht der neue Kontostand von A (100€ weniger) zwar schon im System, aber der Betrag

wurde noch nicht auf Konto B gutgeschrieben. Man sieht entweder den Betrag der am Anfang auf dem Konto ist oder den Betrag nach erfolgreichem Abschluss.

- durability (Haltbarkeit): Erfolgreich ausgeführte Transaktionen, werden auch bei Systemabsturz gespeichert.

[16]

2.1.9 Concurrency Methoden

Um mehrere Transaktionen zeitgleich handhaben zu können, verwendet man Concurrency Control. Daten sollen konsistent bleiben, das schafft man, indem man die ACID-Kriterien erfüllt. Die drei Methoden probieren sicherzustellen, dass diese Kriterien erfüllt sind, besonders Atomarität und Isolation. Die drei Arten sind:

- Pessimistic Locking: Dieser Ansatz geht davon aus, dass es Wahrscheinlich ist, dass Konflikte beim zeitgleichen Zugriff auf Daten Konflikte auftreten. Verhindert wird das, indem man beim Zugriff auf Daten ganze Zeilen oder Tabellen sperrt. Das hat den Vorteile, dass garantiert wird, dass wirklich keine Konflikte auftreten. Jedoch wird insgesamt mehr gesperrt und darunter leidet die Geschwindigkeit der ganzen Prozesse. Wenn eine Transaktion aufgerufen wird, baut diese erst mal auf alle benötigten Datensätze sperren auf (read oder write), danach werden alle Schritte durchgeführt. Bei erfolgreicher Durchführung der Transaktion geht man in die Freigabephase, in der alle gesperrten Datensätze wieder freigegeben werden.
- Optimistic Locking: Hier wird mit den Sperren viel lockerer umgegangen. Es wird davon ausgegangen, dass Konflikte bei Zugriffen nicht so häufig auftreten. Es werden nicht bei allen Transaktionen sperren gesetzt und Konflikte werden erst dann gelöst, wenn sie auftreten. Grundsätzlich ist die Geschwindigkeit wesentlich höher als bei dem pessimistischen Sperransatz, jedoch ist die Abbruchrate der Transaktionen um einiges erhöht, weil eben mehr Konflikte auftreten.
- Multiversion Concurrency Control: Am Anfang der Transaktion wird ein Snapshot erstellt, so dass die Daten mit denen man arbeitet nicht verändert werden können. Ein Snapshot ist eine Speicherung des aktuellen Standes, dieses Prinzip findet in der Informatik häufig Anwendung. Zusätzlich werden auch noch Sperren gesetzt, um wirklich sicher zu gehen, dass sich keine inkonsistenten Daten in die Datenbank einschleichen.

[17]

2.1.10 Opencore und Opensource

Open-Source ist eine Art der Veröffentlichung, bei der der Quellcode öffentlich einsehbar ist. Damit hat jeder Zugriff auf das Programm und kann es prinzipiell frei zu seinen eigenen Zwecken verändern. Jedoch kann der Publisher der Software Limits setzen, in der Art, wie die Software verändert und genutzt werden darf. Das wird in den Nutzungsbedingungen festgelegt. Dabei dürfen keine Menschengruppen ausgeschlossen oder diskriminiert werden. Open-Source-Software ist kostenfrei verfügbar.

Bei einer Open-Core-Veröffentlichung ist nur ein Teil des Quellcodes öffentlich zugänglich. Für den Zugriff auf erweiterte Funktionen oder zusätzliche Module muss jedoch bezahlt werden. Der Code der restlichen Funktionalitäten ist nicht einsehbar, sondern ist ausschließlich zur Nutzung verfügbar. Die Services kommunizieren untereinander, das machen sie über eine API. Auf diese Art und Weise entsteht ein vollständiges Programm.

[18]

2.1.11 Micro Services

Bei einem Microservice-Modell wird das Programm so aufgebaut, dass die Komponenten unabhängig voneinander arbeiten können. Das bietet den Vorteil, dass ein Entwicklungsteam arbeiten kann, ohne die anderen Teams zu behindern.

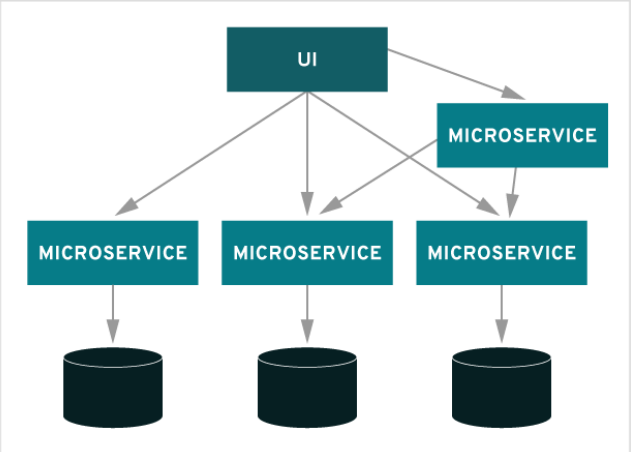


Abbildung 3: Microservice-Architekturk

Die Kernpunkte, die diese Struktur erfüllt, sind:

- Komponenten: Teams entwickeln Komponente die unabhängig voneinander sind.

- gute Wartbarkeit und Testfähigkeit: Änderungen und Spielereien an einer Komponente behindern den Fortschritt der anderen Komponenten nicht. Das heißt es kann rumexperimentiert werden, ohne dass sich Sorgen um andere Teams gemacht werden muss.
- Kleine Teams: Die Entwicklung der einzelnen Komponenten benötigt keine enorm großen Teams. Das führt zu einfacherer Kommunikation und daraus folgender effizienter Entwicklung, weil Entscheidungsprozesse schneller verlaufen.
- Kompetenzverteilung: Teams können ihre Stärken besser nutzen, da der Fokus nur auf der Komponente liegt.

Die API ist die Grundlage der Kommunikation der Microservices, sie bestimmt, wer wie miteinander kommuniziert. Weiters garantiert sie auch die Unabhängigkeit der Komponenten. Die Unabhängigkeit wird damit garantiert, dass keine direkte Kommunikation der Komponenten benötigt wird, sondern nur die Kommunikation mit der API.

[19] [20] [21]

2.2 Verwendete Technologien

2.2.1 Java

Java ist eine objektorientierte Programmiersprache. Laut dem PYPL-Index (Stand März 2025) [22] ist sie die zweitbeliebteste Programmiersprache der Welt und wird nur von Python auf Platz zwei verdrängt. Die Programmiersprache besteht aus mehreren Bestandteilen:

1. **Java Development Kit (JDK)**: Das JDK ist ein Entwicklungspaket für Java. Es besteht aus dem Java-Compiler, Debugging-Tools und dem JRE. Es ermöglicht das Ausführen und Entwickeln von Java-Code.
2. **Java Runtime Environment (JRE)**: Das Java Runtime Environment ist die Laufzeitumgebung für Java-Programme. Es besteht aus der JVM und weiteren Standardbibliotheken, um Java-Programme auszuführen. Es wird benötigt, um Programme auszuführen, aber nicht, um sie zu entwickeln.
3. **Java Virtual Machine (JVM)**: Die JVM kümmert sich um das eigentliche Kompilieren, indem sie Java-Bytecode in Maschinencode übersetzt.
4. **Java-Bibliotheken (Java API)**: Die Bibliotheken sind der letzte wichtige Teil von Java. Sie enthalten vordefinierte Klassen und Methoden, die bei der Entwicklung helfen.

[23] [24]

2.2.2 Artifactory

Artefakt-Repositories sind Plattformen oder Speicherorte, an denen Softwareartefakte bzw. Binärdateien abgelegt und verwaltet werden. Beispiele für solche Binärdateien sind Installationsprogramme für Anwendungen, Container-Images, Bibliotheken, Konfigurationsdateien und vieles mehr.

Ein Beispiel für ein Artefakt-Repository ist *Artifactory*, eine von JFrog entwickelte Lösung zur Speicherung und Verwaltung von Softwareartefakten. Artifactory bietet eine zentrale Plattform zum Hosten, Verwalten und Verteilen von Binärdateien. Um sich dies bildlich vorzustellen, kann es mit einer Lagerhalle verglichen werden: Diese Halle verfügt über zahlreiche Bereiche, in denen alle für die Softwareentwicklung benötigten Komponenten systematisch abgelegt werden.

Der Name *Artifactory* setzt sich aus dem englischen Wort „Factory“ und dem zuvor genannten Begriff „Artefakt“ zusammen.

Die konkreten Vorteile der Nutzung von Artifactory sind folgende:

1. Durch eine zentrale Verwaltung aller Bestandteile der Softwareentwicklung erhalten Unternehmen eine einheitliche Sicht auf ihre Binärdateien. Anstatt diese im gesamten Unternehmen verstreut abzulegen, werden sie in einem System gespeichert, das als sogenannte *Single Source of Truth* fungiert.
2. Es können öffentliche Repositories bereitgestellt werden, um Kunden oder Partnern Zugriff auf Anleitungen oder bestimmte Schnittstellen zu ermöglichen.
3. Versionierung und Abhängigkeiten lassen sich in Artifactory einfach darstellen und verwalten.
4. Auch innerhalb des Unternehmens können Überprüfungen und Freigaben von Binärdateien durchgeführt werden, um beispielsweise einem bestimmten Projektteam Zugriff auf ausgewählte Dateien zu gewähren.
5. Artifactory unterstützt nativ alle gängigen Paketformate und Build-Tools (z. B. Maven, Gradle, npm oder Docker). *Nativ* bedeutet in diesem Zusammenhang, dass die Tools direkt mit Artifactory kommunizieren können, ohne dass komplexe Zwischenlösungen erforderlich sind.
6. Für eine möglichst hohe Verfügbarkeit kann Artifactory auf mehreren Knoten parallel betrieben werden.
7. Darüber hinaus lassen sich zahlreiche Abläufe in Artifactory automatisieren. So kann zum Beispiel eine Sicherheitsprüfung (*Security Check*) erfolgen, bevor neue Binärdateien hochgeladen werden.

Abbildung 4: Gitlab Logo

Allerdings ist Artifactory nicht als alleinige Lösung ausreichend, da es nicht für die Verwaltung von Quellcode konzipiert ist. Es dient weder dem Hosting noch der Versionierung oder Verwaltung von Source Code. Für diese Aufgaben sollten vielmehr spezialisierte Tools wie Git oder CVS verwendet werden. Stattdessen liegt der Fokus von Artifactory auf der Verwaltung der aus dem Quellcode generierten Artefakte. Diese Artefakte enthalten häufig zusätzliche Metadaten, die auf der Ebene des Quellcodes nicht vorhanden sind. [25]

2.2.3 Git/Gitlab

Gitlab ist eine Plattform, die es ermöglicht, Git-Repositories anzulegen. Git, die dahinterliegende Technologie, ist eine Open-Source-Software, die es erlaubt, Projekte zu speichern und die Änderungen im Blick zu behalten. Dabei werden Snapshots erstellt, auf die zurückgewechselt werden kann, falls Fehler auftreten. Zusätzlich ist es möglich, mehrere Äste (Branches) zu erstellen, damit die Projekte der Entwickler nicht kollidieren. Im Grunde ist die Aufgabe von Git, Entwicklerteams dabei zu unterstützen, ihre Projekte zu teilen und das gemeinsame Arbeiten zu ermöglichen.

Git entstand durch das Linux-Kernel-Projekt, das in den frühen 2000er Jahren von Linus Torvalds entwickelt wurde. Damals war der Plan, BitKeeper als "distributed version control systemßu verwenden, jedoch gab es Probleme beim Hersteller von BitKeeper, so dass sie sich dazu entschieden, sich von ihrer gratis Version auf eine bezahlte Version umzustellen. Daraufhin

hat Linus das Problem in seine eigenen Hände genommen und hat innerhalb von wenigen Wochen die Lösung GIT programmiert.

Die wichtigsten Funktionalitäten von Git sind:

- Verteiltes System: Jeder Entwickler hat eine vollständige Kopie des Repositories inklusive aller vergangen Versionen lokal auf seinem PC gespeichert. Dadurch kann das Projekt auch bei einem Ausfall des zentralen Servers über andere lokale Kopien wiederhergestellt werden.
- Kompatibilität: Git läuft auf allen gängigen Betriebssystemen und kann sogar mit Repositories anderer ähnlicher Systeme zusammenarbeiten. Ein Umstieg auf Git ist daher absolut kein Problem.
- Nicht-lineare Entwicklung: Git unterstützt das Arbeiten mit Branches, so dass mehrere Entwickler unabhängig voneinander und zeitlgeich entwickeln können. Man kann über die Versionshistorie die alten Versionen jederzeit wieder überprüfen.
- Branching: Mit Git können einfach neue Zweige erstellt werden, um an Funktionen zu arbeiten ohne den Mainbranch zu beeinflussen. Später können diese Änderungen wieder in den Hauptbranch integriert (merged) werden.
- Leichtgewichtigigkeit: Git speichert Daten sehr effizient. Beim Klonen des Repositories auf den Rechner werden die Daten komprimiert, um Speicherplatz zu sparen. Selbst große Projekte benötigen daher nur wenig Speicherplatz.
- Geschwindigkeit: Durch die lokale Speicherung ist Git extrem schnell, zum Beispiel beim Durchsuchen der ganzen Versionen, beim Vergleichen von Dateien oder beim Erstellen von Commits. Zusätzlich ist Git in einer der performantesten Programmiersprachen geschrieben nämlich C.
- Open Source: Git ist eine Open-Source-Software. Es werden viele Funktionen kostenlos angeboten, die bei Konkurrenten etwas Kosten.
- Zuverlässigkeit: Da jeder Entwickler eine lokale Kopie des Repositories hat, ist Git sehr ausfallsicher. Auch bei Datenverlust auf dem Server oder beim Entwickler kann der Code wiederhergestellt werden.
- Sicherheit: Git verwendet SHA1-Hashes, um jede Änderung eindeutig zu identifizieren. Damit ist es möglich jede Änderung zurück zu verfolgen.

Gitlab wurde 2011 als Open-Source-Software von zwei ukrainischen Software-Entwicklern veröffentlicht und 2014 ist aus dem Projekt eine ganze Firma geworden.

Gitlab und Github sind beides Systeme zur Speicherung und zum Managen von Code und es werden im Kern die gleichen Funktionalitäten geboten. Geboten wird, dass online über die jeweiligen Plattformen, oder eben auch lokal im Terminal gearbeitet werden kann. Auch werden private und öffentliche Repositories, des Weiteren bieten sie unzählige Funktionen, die das Arbeiten im Team vereinfachen. Die Unterschiede der beiden Plattformen sind in erster Linie, wer sie besitzt und die daraus folgende Größe, im Falle von Github ist das Microsoft und im Falle von Gitlab ist das die Gitlab Incorporation. Microsoft bietet die größere Infrastruktur, um das zu kompensieren, legt Gitlab mit manchen Features vor. Zum Beispiel bietet GitLab viele Funktionen wie Rollenverwaltung, CI/CD-Pipelines und DevOps-Integration bereits im kostenlosen Plan an, während diese bei GitHub teilweise nur mit einem kostenpflichtigen Abo verfügbar sind. Beide Plattformen erlauben inzwischen unbegrenzte private Repositories mit beliebig vielen Mitarbeitern im kostenlosen Plan.

Eine Funktion, die bei Gitlab standardmäßig mitimplementiert ist, ist das "Continuous Integration"beziehungsweise das "Continuous Deployment". Continuous Integration beschreibt einen Prozess beim Hochladen in das Repository, dabei wird der Code automatisch getestet und validiert. Beim Continuous Deployment werden, wenn alle Tests erfolgreich sind, die Änderungen automatisch in die Produktivumgebung ausgerollt, also live geschaltet, ohne manuelles Eingreifen.

[26] [27] [28] [29] [30]

2.2.4 Docker

Docker bietet die Möglichkeit zur Virtualisierung mithilfe von Containern, das geschieht in der Docker-Engine. Container sind eine Standard-Einheit, um Programme und Code in einer virtuellen Umgebung zu verpacken. Dabei haben die Container Zugriff auf den Kernel des Hostsystems, jedoch werden die Ressourcen beschränkt und je nach Bedürfnis des Containers verteilt. Der Zugriff auf die Ressourcen ist limitiert.

Containervirtualisierung gibt es vom Prinzip schon seit 1979, damals hat Linux ein Kommando eingeführt, das es erlaubt hat, Teile eines Betriebssystems zu entgliedern. Die Entwicklung weiterer, darauf basierender Anwendungen ist aus Performance-Gründen ausgeblieben. Das Konzept wurde in den 2000er Jahren wieder aufgegriffen, als es ermöglicht wurde, dass mehrere

Webseiten auf einem Server gehostet werden. Heutzutage ist es eine wichtige Technologie, die nicht mehr wegzudenken ist.

Abbildung 5: Dockerlogo

Das heißt, Container sind kompakt verpackte Softwarepakete, in denen alle Abhängigkeiten zur Ausführung des Programms enthalten sind. Diese Abhängigkeiten können externe Codepakete, Libraries und weitere diverse Elemente sein. Die Container laufen eine Ebene über dem Betriebssystem. Virtuelle Computer sind wesentlich umfangreichere Softwarepakete, die auch noch die Hardwarekomponenten und ein Betriebssystem simulieren. Des Weiteren laufen virtuelle Maschinen auf einem sogenannten Hypervisor, dieser siedelt sich auf der gleichen Ebene wie das Betriebssystem. Der Hypervisor ist dafür zuständig, die Ressourcen auf die verschiedenen VMs zu verteilen. Bei beiden Virtualisierungen ist der Zugriff auf die Systemressourcen begrenzt und sie sind auch beide isoliert. Container ähneln der herkömmlichen Virtualisierung sehr stark. Der Hauptunterschied besteht darin, dass bei der herkömmlichen Virtualisierung auch die Hardware simuliert wird und wie die Ressourcen verteilt werden, einmal über den Hypervisor und einmal über das Betriebssystem direkt. Die Gründe, warum man Container benutzt, sind:

- Flexibilität: Der Entwickler hat die Entscheidung, wo er den Code laufen lässt, dadurch dass die Container so einfach sind. Alle Abhängigkeiten sind gespeichert und dann besteht die Möglichkeit sich zu entscheiden ob es in einer Virtuellen-Maschine laufen lässt oder auf dem Hostsystem, der Hypervisor ist nicht unbedingt von Nöten. Auch das Umstellen geht super schnell, die Container sind einfach austauschbar, benötigte Änderungen im Image sind auch einfach durchzuführen. Das alles ohne dass auf andere Programme die im Hostsytem laufen zugegriffen wird.
- einfaches Managment: Die Flexibilität führt natürlich auch zu einer Vereinfachung des Managements, weil einige Dinge wie zum Beispiel das Tauschen einfach geht. Weiters gibt es noch Programme die jemanden bei dem Managment und der Automatiesierung von mehreren Containern unterstützt.
- erhöhte Sicherheit: Durch die Isolation der Container
- Beweglichkeit: Mit Beweglichkeit ist die Einfachheit sie zu verschieben und zu benutzen gemeint. Es geht schnell die Container ein und auszuschalten oder auch neue zu erstellen.

Weiters gibt es auch Automatisierungsmethoden, so dass die Container sich von alleine aktivieren und deaktivieren je nach dem wie sie benötigt werden.

- Effizienz: Ressourcen werden so eingeteilt, dass nicht zu viele Ressourcen verwendet werden, also dass nur auf das gerade Benötigte zugegriffen wrid. Es muss auch kein eigenes Betriebssystem simuliert werden, sondern es wird direkt über das Hostsystem auf die Ressourcen zugegriffen. Durch die Isolation werden Operationen von anderen Programmen nicht gestört.
- Portabilität: Container können samt ihrer Abhängigkeiten aufgrund der Isolation und klaren Definition über was sie benötigen, einfach auf anderen Geräten verwendet werden. Der Container muss also nur einmal erstellt werden und kann danach weitergereicht werden und er muss immer noch funktionieren.

Docker Desktop ist ein Programm für Entwickler, um Container zu erstellen, zu warten und zu teilen. Die Applikation hat viele Funktionen, die diese Prozesse vereinfachen, dazu gehören Kubernetes, Docker Engine, Docker CLI Client und noch einige weitere.

Die Docker Engine ist das Kernstück für die Containerisierung. Sie fungiert quasi als Client-Server-Applikation, die sämtliche Arbeitsprozesse übernimmt, von der Erstellung bis zur Ausführung. Es ist eine Open-Source-Software, bei der mithilfe eines Daemon-Prozesses die Docker-Objekte verwaltet werden. Ein Daemon-Prozess ist eine Anwendung, die im Hintergrund ausgeführt wird und kein Terminal benötigt. Die Docker CLI ist eine eingebaute Kommandozeile, mit der man Befehle an den Daemon schicken kann und die REST-API dient zur Kommunikation zwischen CLI und Daemon, aber sie ist auch eine Schnittstelle zu anderen Anwendungen.

Ein Dockerfile ist eine Textdatei, die Befehle, Konfigurationen und Prozeduren beinhaltet. Man kann es als den Sourcecode für ein Dockerimage bezeichnen, nach diesem baut sich das Image auf. Das erlaubt das automatische Bauen von den Images und sie sind damit immer auf dem neuesten Stand. Das Image up to date zu halten, steigert die Sicherheit enorm. Die Images werden dabei nach dem Layersystem gebaut. Ein Layer ist eine Schicht, beim Layersystem wird also ein Image Schicht für Schicht generiert. Jede dieser Schichten repräsentiert eine Änderung oder einen Befehl aus dem Dockerfile. Man kann zwischen drei Schichtarten unterscheiden, dem Baselayer (Basisschicht), dem Toplayer (Oberschicht) und dem Rest. Die Basisschicht ist der Layer, auf dem das Betriebssystem angegeben wird. Der Toplayer wird bei der Ausführung des Images automatisch von Docker erstellt, dabei ist sie die einzige Schicht, in der man etwas schreiben kann beziehungsweise in der man etwas verändern kann. Zum Beispiel, wenn man Logfiles schreiben will, funktioniert das nur über den Toplayer, man könnte es als Arbeitsfläche

bezeichnen. Beim Rest werden die ganzen Befehle ausgeführt, die im Dockerfile angegeben sind.

Es gibt zwei Möglichkeiten, ein Docker-Image zu generieren:

- aus bereits bestehenden Containerimages: Es wird sich also ein bereits bestehendes Dockerimage genommen und es werden die nötigen Konfigurationen vorgenommen, so dass es die eigenen Ansprüche erfüllt. Dies ist die einfachste und schnellste Methode ein Image für seine Zwecke zu kreieren.
- aus einem Dockerfile: Bei diesem Ansatz baut wird sich sein eigenes Dockerfile von Grund auf neu gebaut, mitsamt aller Abhängigkeiten und Kommandos. Solange dieser Ansatz sauber und gut geplant durchgeführt wird, garantiert es eine nahezu perfekte Individuelle Lösung, die die gewünschte Performance und Funktionalität liefert. Der Prozess zur Erstellung ist aber wesentlich zeitintensiver und auch um einiges komplizierter.

Es wurde die Methode erwähnt, bei der bereits fertige Containerimages genommen werden und sie überarbeitet, diese Images sind im DockerHub zu finden. Dies ist ein Opensource-Cloudservice, auf dem Dockerimages veröffentlicht werden. Andere Personen können diese hochgeladenen Images dann herunterladen und verwenden. Beim Veröffentlichen hat man die Möglichkeit, das Image auf public oder private zu setzen, bei private haben dann nur bestimmte Personen Zugriff. Dieses Feature ist zum Beispiel für Firmen und Projektgruppen nützlich.

Netzwerke in Docker oder im Englischen Networking erlauben es, Container untereinander kommunizieren zu lassen, es erlaubt auch die Kommunikation mit anderer Software. Ein Container weiß nicht, in welcher Art Netzwerk er sich befindet und es weiß auch nicht, mit welcher Software er kommuniziert. Er kennt dabei nur, wohin er sich verbinden muss. In Docker Desktop gibt es die Möglichkeit, ein Netzwerk mithilfe von Befehlen zu erstellen, diesem Netzwerk können die Container zugewiesen werden. Es ist auch möglich, dass ein einzelner Container mit mehreren Netzwerken verbunden ist, dabei müssen die Netzwerke nicht vom gleichen Typ sein. Es gibt fünf Netzwerkarten:

- bridge: Es gibt eine Brücke zwischen Container und Host, diese erlaubt die Kommunikation zwischen den beiden. Jedoch sind sie von Software außerhalb des Netzwerkes isoliert. Jedem Container wird eine eigene IP-Adresse zugewiesen, damit sie klar voneinander zu unterscheiden sind.
- host: Der Networkstack wird mit dem Host geteilt, das heißt unter anderem auch, dass es die gleiche IP-Adresse wie der Host hat. Die Container werden dann mithilfe von Ports adressiert.

- overlay: Beim Overlaynetworking wird ein ein virtuelles Netzwerk aus mehreren Docker Hosts aufgebaut, diese bilden, eine Art Straße für bestimmte Geräte und Software. Mit dieser Option ist es möglich ein Netzwerk so erweitern, dass Container auch über mehrere Geräte miteinander verbunden sein können. Denn normalerweise kommunizieren die Container nur innerhalb eines Netzwerks oder einer virtuellen Maschine miteinander. Das vereteilen von den Containern, wird Docker Swarm Cluster genannt.
- IPvLAN: Das ist ein Linux Netzwerk-Driver, der einem Kontrolle über alle Arten von IP-Adressen von den Containern ermöglicht. Das ganze arbeitet auf Layer 3 also auf der IP-Ebene, kann aber auch auf Layer 2 dem Mac Layer arbeiten. Mit diesen zwei Optionen besteht die Möglichkeit seine Container mit seinem Physischen Netzwerk zu verbinden, ohne dass eine virtuelle Brücke benötigt wird. Das ganze erlaubt die direkte Kommunikation der Container mit anderen Geräten in dem Netzwerk ganz ohne Ports. IPvLAN hat gegenüber der bridge einen Performance Vorteil.
- macvlan: Das beschreibt den Vorgang einen Container als ein physisches Gerät im Netzwerk anzuzeigen. Einem Container kann eine MacAdresse gegeben werden und auch eine IP-Adresse aus dem Subnet des Hosts.

Meistens ist ein bridged network ausreichend, es ist einfach und schnell zum Aufsetzen und erfüllt die meisten Kriterien. Weiters sind alle nötigen Funktionen dabei und die Isolation sorgt für erhöhte Sicherheit. Damit sind sie eine geeignete Lösung für allgemeine Container-Anwendungen. Falls spezifischere Anforderungen nötig sind, muss zu den anderen Methoden gegriffen werden. Ein Hostnetzwerk hilft zum Beispiel sehr bei der Performance, da kein großer Overhead entsteht. Der Overhead ist kleiner, weil die IP-Adresse des Hosts verwendet wird und die Container wie reguläre Apps auf dem Rechner über Ports aufgerufen werden. Wenn Container auf verschiedenen Rechnern verteilt sind, dann muss eigentlich das Overlaynetworking benützt werden. IPvLAN wird verwendet, wenn die Container und ihre IP-Adressen genau konfiguriert werden müssen. Zu guter Letzt, Macvlan ist unbedingt zu verwenden, wenn die Anwendung als physisches Gerät erscheinen muss, zum Beispiel wenn Netzwerkverkehr überwacht werden soll.

Alle Netzwerkmodi in Docker basieren grundsätzlich auf dem Netzwerkstack des Hosts. Dieser Stack ist für die Verwaltung des gesamten Netzwerkverkehrs im Hostsystem verantwortlich. Docker greift auf diesen Stack zu und nutzt ihn, um ein eigenes virtuelles Netzwerkmodell für Container bereitzustellen und zu verwalten. Dabei konfiguriert Docker automatisch die notwendigen Netzwerkkomponenten, zum Beispiel virtuelle Netzwerkschnittstellen, IP-Adressen und Routingregeln. Zudem setzt Docker iptables-Regeln ein, um den Datenverkehr zu steuern. Beispielsweise wird bei einer Portweiterleitung festgelegt, dass Anfragen an den entsprechenden

Container weitergeleitet werden. Diese Portweiterleitungen ermöglichen die Kommunikation zwischen dem Host und den Containern sowie zwischen Containern untereinander – je nach gewähltem Netzwerkmodus. Gleichzeitig tragen sie zur Isolation der Container bei. Durch die Nutzung von iptables lassen sich außerdem zusätzliche Filterregeln definieren, um den Netzwerkzugriff weiter abzusichern. Diese Konfigurationen übernimmt Docker in der Regel automatisch. Zusätzliche Isolation wird durch Namespaces erreicht. Docker verwendet sie, um sicherzustellen, dass jeder Container nur auf seine eigene Netzwerkumgebung zugreifen kann. So wird gehandhabt, welche Container miteinander kommunizieren dürfen.

Ein Tool zum Managen von Docker-Containern wird von Docker selbst bereitgestellt und zwar ist das Docker Compose. Es vereinfacht den Prozess von der Erstellung und macht es einfacher, mehrere Container zeitgleich auszuführen, indem es den Prozess des Managements mehrerer Container automatisiert. Standardmäßig gibt es die "docker-compose.yaml"Datei, in dieser gibt man alle seine Container an und man kann sie dann zeitgleich starten. Bei der Ausführung der YAML-Dateien wird auch automatisch ein Dockernetwork erstellt, damit die Container dann die Möglichkeit haben, miteinander zu kommunizieren. Es automatisiert auch Teile der Speicherverteilung und gibt welchen frei wenn man gewisse Container abstellt oder reserviert ihn wieder, wenn man ihn wieder startet. Es ist also praktisch, wenn man eine App mit mehreren Komponenten hat. Als Beispiel nehme ich eine Applikation, die Wetterdaten sammelt, diese müssen dann also in eine Datenbank gespeichert werden und zusätzlich haben wir dann noch ein Frontend, in dem wir alle Daten anzeigen, damit die Information einfach zugänglich ist. Dann können wir drei Container erstellen, einmal für die Datenbank, in der man die Daten speichert, dann für das Backend, das die Analyse übernimmt und noch einen letzten für das Frontend. Diese jetzt jedes Mal einzeln managen zu müssen, ist sehr aufwendig und daher ist Docker Compose ein praktisches Tool.

[31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44]

2.2.5 PostgreSQL

Abbildung 6: PostgreSQL Logo

PostgreSQL ist ein leistungsfähiges objekt-relationales Datenbankmanagementsystem, im Gegensatz zu ähnlichen Systemen unterstützt PostgreSQL relationale und nicht-relationale Datentypen. Damit ist es eines der vielseitig einsetzbarsten Datenbankmanagementsysteme. Zusätzlich verspricht PostgreSQL auch noch extrem leistungsfähig und stabil zu sein. Verschiedene eingebaute Systeme, die eben die hohe Leistung und Stabilität versprechen, eine riesige Community, die bei zahllosen Problemen hilft und eine große Menge an Dokumentation machen PostgreSQL zu einer der verlässlichsten und Ëasy-to-learn"DBMS. Und obendrein ist sie noch Open-Source, das bedeutet, dass der Quellcode der Öffentlichkeit zur Verfügung steht und jeder sie benutzen kann. Natürlich gibt es eine offizielle Version, die von einer Open-Source-Gruppe weiterentwickelt wird, diese Gruppe heißt PostgreSQL Global Development Group.

Ihren Ursprung hat das Datenbankmanagementsystem schon in den 80er Jahren, dort wurde nämlich ein vom Militär gesponsertes Projekt entwickelt. Der ursprüngliche Name war "Berkeley Postgres Project", geleitet wurde das von Professor Michael Stonebreaker an der University of California. Die erste Demoversion wurde 1987 in einer Expo gezeigt, die erste offizielle Version wurde 1989 an bestimmte Nutzer gegeben. Mit den weiteren Versionen wurde das Berkeley Postgres Project dann bereits an Universitäten zu Forschungs- und Lernzwecken und in manchen Unternehmen zu Datenanalysezwecken verwendet.

1994 wurde das Berkeley Postgres Project zu Postgres95, diese Änderung kam mit der Implementierung des SSQL language Interpreteründ damit war es möglich, SQL zu nutzen. Vor der Einführung des Interpreters hat man eine eigene Sprache namens PostQUEL verwendet, diese hat sich aber nicht durchgesetzt.

Und nach nur 2 Jahren wurde der Name Postgres95 bereits wieder abgelöst und zwar mit dem Namen, mit dem es heute bekannt ist, PostgreSQL. Die Änderung entstand hauptsächlich wegen der Zahl 95, weil diese offensichtlich an das Jahr angelehnt war.

PostgreSQL hat als Ziel, dem Entwickler den Aufbau und die Entwicklung der Datenbank möglichst einfach zu gestalten. Konkreter soll es dabei helfen, die Datenintegrität zu erhalten und das Management der Daten einfach zu halten, egal ob große oder kleine Datensätze. Kern des ganzen ist eine nahezu vollständige Implementierung der nötigen SQL Funktionen. 170 von 177 wurden bereits implementiert, auch wenn manche Features etwas verändert wurden um im Kontext der PostgreSQL-Philosophie zu bleiben. Funktionen die die Ziele umsetzen und mit denen PostgreSQL wirbt, sind:

- Datentypen: Es gibt eine weitreichende Variation von Datentypen.
  - Nummerische Typen: Es nutzt diese um Zahlen zu speichern.

  - Geld Typ: Es gibt einen Typen mit dem es möglich ist, Währungen zu speichern. Dieser heißt einfach Money und es ist möglich ihn auf verschieden Währungen einzustellen.
  - Character Typen: In diesen speichert es Zeichen oder längere Texte.
  - Binary Typ: Dieser wird einfach zur Speicherung für pure binäre Daten verwendet.
  - Datum und Zeit Typen: Um eine klare zeitlich Einordung zu haben werden diese Typen genutzt, es kann das Format natürlich anpassen (zum Beispiel: MM/DD/YYYY oder DD/MM/YYYY).
  - Boolean Typ: Ein ganz normaler Boolean der true, false oder NULL speichert.
  - Enumeration Typ: Dieser Typ wird vom Benutzer definiert, wenn ich also eine ganz spezielle Art habe ein Objekt zu speicher. (Zum Beispiel Datatype Noten hat fünf Möglichkeiten: 'sehr gut', 'gut', 'befriedigend', 'Genügend' und 'nicht genügend')
  - Geometrische Typen: Diese Datentypen speichern zweidimensionale Werte, es kann Kreise, Polygone, Linien und noch einige mehr speichern.

- Datenintegrität: Datenintegrität versichert, dass die Daten konsistent sind. Mit konsistent ist gemeint, dass in den Daten keine Fehler sind (zum Beispiel in einer Nummer darf kein negativer Wert sein, damit der Datensatz Sinn ergibt). In PostgreSQL wird das mithilfe von Datentypen, Triggern und Constraints gewährleistet. Datentypen sichern, dass zum Beispiel eine Nummer auch wirklich eine Nummer ist. Zusätzlich ist es möglich, die eingegebenen Werte weiter zu limitieren. Ein Text darf nicht länger als 30 Zeichen sein, zum Beispiel. Trigger sind eine Funktion von Datenbanken, mit der gewisse Situationen automatisch überprüft werden. Bei Inserts, Deletes oder Updates wird eine Kontrolle durchgeführt. Trigger können selbst programmiert werden, um individuelle Probleme zu lösen und Absicherungen vorzunehmen. Mit Constraints wird gesichert, dass bei der Erstellung und Änderung von Daten allen Anforderungen entsprochen wird (Ein Preis für ein Produkt darf keine negative Zahl sein).
- Parallelität: Parallelität oder im Englischen Concurrency kümmert sich um das Verarbeiten von mehreren Transaktionen, so dass die Daten konsistent bleiben. Bei PostgreSQL ist das Verfahren, mit dem das gewährleistet wird, das "Multiversion Version Concurrency Control"(MVCC). Um zu verstehen, wie PostgreSQL Concurrency Control handhabt, muss verstanden werden, wie die 6 "Building blocks"funktionieren.

  - Transaction IDs: Jede Transaktion bekommt eine eigene ID über die sie einmalig identifizierbar ist. Bei jeder Transaktion wird diese ID automatisch generiert und wird so lange beibehalten bis die Ausführung erfolgreich beendet wurde oder ein Rollback durchgeführt worden ist.
  - Versioning of Tuple: Jede Tupel besitzt zwei Felder xmin und xmax. Xmin speichert die Transkations-ID die diese Version, der Zeile erstellt hat und xmax welche Transaktions-ID die Version der Zeile zerstört hat. Damit ist es möglich in Tupeln zu arbeiten, so wird eine Reihenfolge aufgebaut, in der die Transaktionen durchgeführt werden und man kann auch ältere Versionen überprüfen falls man das möchte.
  - Visibility Check Rules: Mithilfe dieser Regeln wird festgelegt, welche Version einer Zeile für eine bestimmte Transaktion sichtbar ist. So wird sichergestellt, dass jede Transaktion eine konsistente Sicht auf die Daten hat und nur gültige Versionen lesen oder bearbeiten kann.
  - Read consistency: Die Read Consistency garantiert, dass zum Beginn der Transaktion ein konsistenter Snapshot zur Verfügung steht.
  - Write Operations: Insert Operationen werden so wie in den meisten anderen Datenbanken gehandhabt, bei Update und Delete wird MVCC verwendet. Ein Datensatz wird bei einem Update nicht wirklich gelöscht, sondern als tot markiert und eine neue Version wird als "lebendig"markiert. So hat man immer eine Konsistente Version auf die zugegriffen werden kann. Weiters ist es möglich auf ältere Versionen zurück zu wechseln oder sie einfach nur zu kontrollieren. Dieses Prinzip ist äußerst Speicher intensiv. Beim Löschen ist es das gleiche Prinzip, es wird eine neue Snapshot erstellt, in der der gelöschte Datensatz nicht vorhanden ist.
  - Garbage Collection: Es ist möglich alte Versionen mithilfe des VACUUM-Kommandos löschen und so wieder Speicherplatz freizugeben.

- Leistung: Query-Performance wird durch automatische Planung gewährleistet. Sobald eine Transaktion ausgeführt wird, plant PostgreSQL den effizientesten Weg, diesen auszuführen. Bei der Planung werden verschiedene Faktoren wie verfügbare Ressourcen berücksichtigt.
- Verlässlichkeit: Der Begriff Verlässlichkeit impliziert einige Funktionen. Wenn eine Transaktion committet wurde, darf sie auch bei Systemabstürzen nicht mehr verloren gehen. Jeder Schritt wird vor seiner Ausführung in eine Logdatei geschrieben (Write-Ahead-Log). Bei einem Systemabsturz ist sofort erkennbar, ob die Transaktion beendet wurde. Wenn

die Transaktion nicht fertiggestellt wurde, können die einzelnen Schritte wiederhergestellt werden und die Transaktion fortgeführt werden. Ein Rollback ist natürlich auch möglich.

- Sicherheit: PostgreSQL garantiert die Sicherheit der Daten durch mehrere Schutzmechanismen. Alle Datenbankdateien können standardmäßig nur vom PostgreSQL-Superuser gelesen und verändert werden. Die Benutzerauthentifizierung sorgt dafür, dass sich Nutzer, um mit der Datenbank zu arbeiten, erst einmal anmelden müssen. Durch die Rollenverteilung bekommen Benutzer bestimmte Rechte. Ein Personalmanager kann zum Beispiel Zugriff auf Mitarbeiterdaten erhalten, während ein Entwickler nur auf technische Daten zugreifen darf. Zusätzlich lassen sich nur bestimmte Verbindungsmethoden und -quellen bestimmen. Dort kann man IP-Whitelists definieren und festlegen, dass nur Verbindungen von bestimmten IP-Adressen erlaubt sind.
- Erweiterbarkeit und Skalierbarkeit: Skalierbarkeit ist die Funktion, sich den unterschiedlichen Leistungsbedürfnissen anzupassen. Nehmen wir als Beispiel Netflix in der DACH-Region. Zu den herkömmlichen Arbeitszeiten 8-17 Uhr sind die Server wesentlich weniger ausgelastet als am Abend. Also sollen von 8-17 Uhr Ressourcen freigegeben werden, um Energie und Kosten zu sparen. Am Abend hingegen müssen die Server dem erhöhten Andrang an Benutzern standhalten. Man unterscheidet zwischen zwei Arten von Skalierung:
  - Vertikale Skalierung: Hierbei werden bestehende Server durch leistungsfähigere Hardwarekomponenten erweitert, zum Beispiel durch mehr Arbeitsspeicher oder eine schnellere CPU.
  - Horizontale Skalierung: Hier wird die Last auf mehrere Server verteilt. Ein Beispiel: Wenn in der Region Schweiz aktuell viele Ressourcen ungenutzt sind, während in Österreich eine hohe Auslastung herrscht, könnten die Schweizer Server zur Entlastung herangezogen werden.

  PostgreSQL bietet verschiedene Funktionen, die eine einfache Skalierung unterstützen.

[45] [46] [47] [48] [49] [50] [51] [17] [52] [53] [54] [55]

2.2.6 JUnit

Bei JUnit handelt es sich um ein Framework von Java, das dazu verwendet wird, Unit-Tests zu erstellen. Mit normalen Tests ist der Vorgang des Überprüfens, ob eine Applikation alle Anforderungen erfüllt, gemeint. Unit-Testing hingegen beschreibt das Testen einzelner Entitäten

(Klassen oder Methoden). Unittesting ist wichtig, um wartbarere Produkte auf den Markt zu bringen. Grundsätzlich kann Unit-Testen auf zwei Weisen unterschieden werden:

Manuelles Testen: Davon ist die Sprache, wenn Tests manuell ohne Tools ausführt werden. Das kann sehr langsam, und teuer sein, um die dafür eingestellten Tester zu bezahlen. Außerdem könnten diese Fehler machen.

Automatisches Testen: Im Gegensatz zum manuellen Testen erfolgt dabei das Testen mittels eines Tools automatisch. Es ist dadurch günstiger, schneller und meistens zuverlässiger, da keine/weniger menschliche Tester mehr benötigt werden.

Wie bereits erwähnt, ist JUnit ein Unittesting-Framework für Java. JUnit glaubt an den Spruch ”first testing, then coding”, womit gemeint ist, dass zuerst Tests zu einem Stück Code erstellt werden und erst danach dieser Code implementiert wird. Dieser Zugang kann die Motivation des Programmierers erhöhen, da das Projekt Stück für Stück programmiert wird, anstatt sich gleich mit dem ganzen Projekt auf einmal auseinanderzusetzen. Dadurch kann nicht nur der Stress des Programmierers gesenkt werden, sondern auch die Produktivität gesteigert werden. Außerdem stellt es sicher, dass der geschriebene Code so funktioniert, wie erwartet.
Eine Test-Methode kann sehr einfach verfasst werden. Zuerst ist eine Methode nötig, die getestet werden soll. Beispielsweise eine Methode, die den String "Hello Worldßurückgeben soll. Die Methode befindet sich in der Klasse "MessagePrinter". Danach wird eine Testklasse, die JUnit verwendet, erstellt, die diese Methode testen soll:

Listing 1: example JUnit Test

```
1  public class TestJunit {
2
3     MessagePrinter messagePrinter = new messagePrinter();
4
5     @Test
6     public void testPrintMessage() {
7        String message = "Hello World";
8        assertEquals(message,messagePrinter.printMessage());
9     }
10 }
```

Mithilfe der Annotation "@Test"wird gekennzeichnet, dass es sich um eine Test-Methode handelt. Der String message ist der erwartete Wert, den die Methode zurückgeben soll. Die ässert...-Methoden sind dazu da, festzustellen, ob ein Test gelungen ist oder nicht. In diesem Fall wird ässertEquals"verwendet, um sicherzustellen, dass der Rückgabewert der Methode mit dem des message-Strings übereinstimmt. Wenn nicht, scheitert der UnitTest. In der Praxis gibt es, besonders bei größeren Projekten, normalerweise mehr UnitTests. Wenn dort später im Test etwas verändert wurde, das an einer anderen Stelle einen Fehler auslöst, wird das bemerkt, sobald das nächste Mal getestet wird. [56]

2.2.7 XML

XML steht für Extensible Markup Language und ist eine der beliebtesten Technologien, wenn es um Datenaustausch geht. Um zu verstehen, was XML ist, muss erst das Konzept der Markup-Language verstanden werden:

Menschen erstellen schon seit langer Zeit Markierungen/Anmerkungen in ihren Dokumenten. Mithilfe der Markierungen werden Anweisungen gegeben, wie der Text eigentlich aussehen sollte. Wie ein Lehrer / eine Lehrerin, der / die einen Text korrigiert. Die Struktur, die Erscheinung, die Bedeutung des Dokuments, all das kann mithilfe von Markierungen definiert werden. Diese Markierungen haben sich in der Datenverarbeitung zu Markup weiterentwickelt. Hier werden eben diese Dinge mittels Code definiert. Im Falle von XML wird die Bedeutung der Daten definiert. Auch HTML erstellt Markierungen mithilfe der HTML-Tags (...). HTML definiert beispielsweise den Anfang und das Ende eines Absatzes, ob es sich um eine Überschrift handelt, etc.

Die Syntax von XML ist sehr ähnlich zu HTML. Beide verwenden Tags, zwischen denen sich Daten befinden, jedoch ist der Gebrauch vollkommen unterschiedlich. HTML-Markierungen definieren das Aussehen und das Verhalten eines Dokuments. XML definiert allerdings die Struktur der Daten und was diese bedeuten. Dadurch können die definierten Daten auf andere Weise wiederverwendet werden. Es können dadurch beispielsweise Daten erzeugt, im XML-Format gespeichert werden und danach dann in verschiedenen Systemen wiederverwendet werden. Außerdem ist es vollkommen hardware- und betriebssystemunabhängig. Das bedeutet, dass jedes Programm, egal welche Hardware und welches Betriebssystem, das dazu entwickelt wurde, XML-Daten zu lesen und damit zu arbeiten, auch dazu in der Lage ist. XML erlaubt es, jede Markierung, die zur Beschreibung der Daten benötigt wird, zu erstellen. Die XML-Tags ermöglichen es, dass es leicht ersichtlich ist, was die Daten bedeuten und in welcher Struktur sie sich befinden. Beispielsweise ist relativ schnell ersichtlich, dass folgende Daten ein Buch beschreiben und dass von diesem Buch der Autor und der Titel bekannt sind:

Listing 2: example XML

```
1 <book>
2     <author>J.R.R. Tolkien</author>
3     <title>Der Herr der Ringe</title>
4 </book>
```

Eine XML-Datei muss strikten Regeln folgen, um verwendet werden zu können, ansonsten wird ein Fehler auftreten. Ein Beispiel für eine dieser Regeln wäre, dass jede öffnende Markierung eine dazugehörige schließende Markierung aufweisen muss. Wenn alle Regeln befolgt werden, handelt es sich um ein sogenanntes ”wohlgeformtes” XML.

In der Regel gibt es in XML-Systemen noch weitere Komponenten:

Schema: Bei einem Schema handelt es sich um eine XML-Datei, welche Regeln festlegt, welche bestimmen, was eine XML-Datendatei speichert und was nicht. Die Endung dieser Dateien ist .xsd. Die Datendateien hingegen enden auf .xml. Schemas dienen dazu, dass ein Programm überprüfen kann, ob die Daten gültig und sinnvoll sind. Sie verhindern außerdem falsche Eingaben, indem es den Benutzer bei einer ungültigen Eingabe auffordert, diese zu korrigieren. Das Schema kann von Programmen genutzt werden um Daten zu lesen, zu interpretieren, zu nutzen und zu bearbeiten.

Transformationen: Durch sogenannte XSLT(Extensible Stylesheet Language Transformation)-Transformationen wird es leichter die XML-Daten wiederzuverwenden. Sie ermöglichen es Daten aus 2 unterschiedlichen System auszutauschen, und sie in passende Strukturen zu überführen.

All diese Komponenten zusammen ergeben ein XML-System. In der Regel wird zuerst die Datendatei vom Schema überprüft und danach wird von der Transformation, verschiedene Weisen der Verwendung dieser Daten dargestellt. [57]

2.2.8 Unterschied POJO, Java-Beans und DAO

`POJO` steht für ”Plain Old Java Object” und beschreibt Klassen, welche keinen speziellen Namenskonventionen folgen. Hauptsächlich dient es als Datencontainer mit den erforderlichen Zugriffsmethoden. Da es kein spezifisches Framework braucht, kann es von jedem Java-Programm verwendet werden. Es fördert die Lesbarkeit und die Wiederverwendbarkeit. Diese Struktur bringt allerdings auch 2 Probleme mit sich:

1. Neuen Entwicklern könnte es schwerer fallen, die Klasse zu verstehen.
2. Viele Frameworks erwarten eine bestimmte Struktur oder Methoden, um zu verstehen, wie die Klasse verwendet wird.

Hier ist ein Beispiel für eine POJO-Klasse:

Listing 3: example POJO Class

```
1  public class Book {
2      private String title;
3      private String author;
4
5      public Book(String title, String author) {
6          this.title = title;
7          this.author = author;
8      }
9
10     public String getTitle() {
11         return title;
12     }
```

```
13
14      public String getAuthor() {
15          return author;
16      }
17  }
```

Diese Klasse enthält weder einen Standardkonstruktor noch Setter-Methoden.

Java-Beans sind auch POJOs, jedoch müssen diese Klassen bestimmten Regeln Folge leisten:

1. **Zugriffsebenen**

   Alle Eigenschaften sind privat, doch es werden public Getter- und Settermethoden zur Verfügung gestellt, um darauf zuzugreifen.

2. **Methodennamen**

   Manche Methodennamen haben einem Namensschema zu folgen. Für Getter und Setter zum Beispiel getTitle und setTitle. Sollte es sich allerdings um einen boolean handeln, darf die Gettermethode auch isX heißen.

3. **Standardkonstruktor**

   Jede Klasse muss einen Konstruktor ohne Parameter haben. Dadurch kann eine Instanz dieser Klasse erstellt werden, ohne einen Parameter zu übergeben. Das ist beispielsweise für die Deserialisierung notwendig.

4. **Serializable**

   Damit die Serialisierbarkeit der Klasse gewährleistet ist, muss das Interface ”Serializable” implementiert werden.

So würde die Book-Klasse aussehen, wenn sie den JavaBean-Regeln folgen würde:

Listing 4: example JavaBean Class

```
1   public class Book implements Serializable {
2       private String title;
3       private String author;
4
5       public Book() {
6       }
7
8       public Book(String title, String author) {
9           this.title = title;
10          this.author = author;
11      }
12
13      public String getTitle() {
14          return title;
15      }
16
17      public void setTitle(String title) {
18          this.title = title;
19      }
20
21      public String getAuthor() {
22          return author;
23      }
24
25      public void setAuthor(String author) {
26          this.author = author;
27      }
```

```
28 }
```

Der Konstruktor mit Parametern ist optional.
[58]
`DAO` steht für "Data Access Objectünd es handelt sich dabei um ein Entwurfsmuster, das die Businesslogic-Schicht von der Datenbankschicht trennt. Also trennt es die Datenbank von der Anwendung, was diese unabhängiger macht. Auf der DAO-Schicht werden außerdem die CRUD-Operationen (Create, Read, Update, Delete) ausgeführt. Ohne dieses Muster könnte datenbankbezogener Code überall in der Anwendung gestreut sein. Es könnte an Konsistenz mangeln, aber auch an der Skalierbarkeit. Mit DAO-Pattern wird allerdings ein zentrales Set an Datenbankmethoden eingeführt, worauf zugegriffen werden kann. Das DAO-Pattern bietet folgende Vorteile:

Organisation All der datenbankbezogene Code wird zentralisiert, was die Codebasis übersichtlicher macht.

Konsistenz Da nicht auf unterschiedliche Arten auf die Datenbank zugegriffen wird, sind Fehler unwahrscheinlicher. Die standardisierte DAO-Schnittstelle greift nur auf eine Art auf die Datenbank zu.

Leichtere Wartung Sollte sich etwas auf Datenbankebene verändern, muss nur die DAO-Schicht angepasst werden.

Einfachere Tests Da die DAO-Funktionen einheitlich sind, brauchen nur diese getestet werden, was die Unittests vereinfacht.

Skalierbarkeit Sollte die Anwendung wachsen, werden nur neue Schnittstellen hinzugefügt.

Das DAO-Muster benötigt mehrere Bestandteile:

BusinessObject Dieses Objekt stellt die Kerndaten der Anwendung dar. Dieses Objekt speichert die Daten aus der Datenquelle in sich, damit die Anwendung damit arbeiten kann.

DataAccessObject (DAO) Dieses Objekt enthält die Methoden für den Datenbankzugriff. Es bildet die standardisierte Schnittstelle mit allen CRUD-Operationen. Es erlaubt der Anwendung mit den Daten arbeiten zu können, ohne sich dabei um die Umsetzung kümmern zu müssen.

DataSource Diese verwaltet die Verbindung zur Datenbank. Es wird von den DAOs verwendet, um Verbindungen herzustellen.

DataTransferObject (DTO) Dieses Transportobjekt dient als Datenträger zwischen den Schichten der Anwendung, oder Client und Server. Es wird also beispielsweise verwendet, um Daten, die die Anwendung liefert, in die Datenbank zu speichern / zu aktualisieren. Es vereinfacht und verbessert den Prozess des Aktualisierens, indem sich mehrere Datenfelder in einem Objekt befinden. Dadurch müssen weniger Methodenaufrufe gemacht werden.

`Zusammenspiel der Objekte`

Das BusinessObject interagiert mit dem DTO, um Daten zu übertragen.
Das DAO verwendet beide Objekte. Es ist dazu da, die Daten aus der Anwendung (Businessobject) in die Form der Datenübertragung (DataTransferObject) zu konvertieren, oder umgekehrt. Das DAO verwendet außerdem die Datenquelle, um eine Verbindung zur Datenbank aufbauen zu können. Das wird benötigt, damit das DAO auf der Datenbank CRUD-Methoden ausführen kann.
[59]

Diese Grafik bildet dieses Zusammenspiel ab:

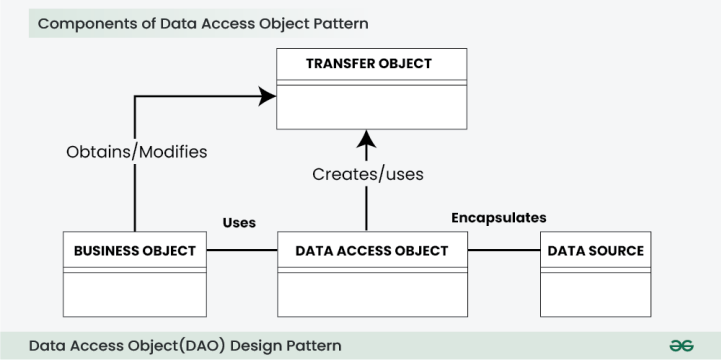


Abbildung 7: Abbildung des Zusammenspiels, der einzelnen DAO-Komponenten [59]

2.3 Verwendete Bibliotheken und Plug-Ins

2.3.1 Spring Framework

Das Spring Framework ist ein leichtgewichtiges Java-Framework, das für die Entwicklung von testbaren, wiederverwendbaren und leistungsstarken Java-Applikationen verwendet wird. Es wurde 2003 von Rod Johnson veröffentlicht und basiert auf einer POJO (Plain Old Java Object)-Programmierung. Das Framework ist modular aufgebaut und besteht aus mehreren Komponenten.

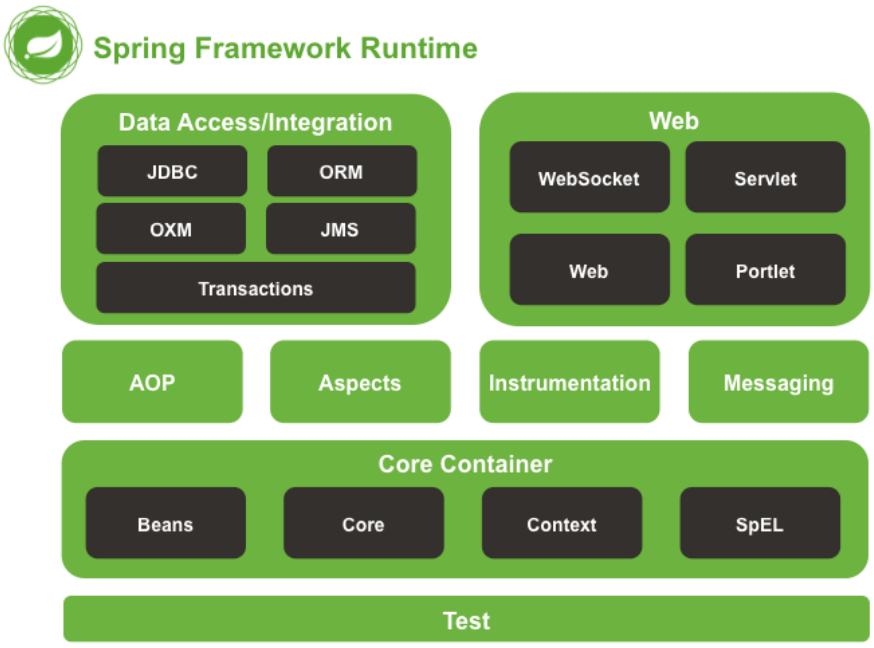


Abbildung 8: Java Spring Framework Runtime

[60] Das **Core-Modul** (siehe **??**) bildet die Basis des Spring Frameworks und ist für grundlegende DI (Dependency Injections) und IoC (Inversion of Control) verantwortlich.

DI (Dependency Injections) ist eine zentrale Komponente des Spring Frameworks, die eine lose Kopplung zwischen Klassen ermöglicht, indem Abhängigkeiten nicht direkt instanziiert, sondern bereitgestellt werden. Dadurch wird die Testbarkeit und Flexibilität des Codes verbessert.

Listing 5: Dependecy Injection Example in Java Spring

```
1
2  public class SimpleMovieLister {
3
4    // the SimpleMovieLister has a dependency on a MovieFinder
5    private final MovieFinder movieFinder;
6
7    // a constructor so that the Spring container can inject a MovieFinder
8    public SimpleMovieLister(MovieFinder movieFinder) {
9      this.movieFinder = movieFinder;
10   }
11
12   // business logic that actually uses the injected MovieFinder is omitted...
13 }
```

[61]

IoC (Inversion of Control) ist ein Design-Pattern 2.1.3, das darauf abzielt, bestimmte Aufgaben des Hauptprogramms an ein Framework oder andere Komponenten zu delegieren. Diese Aufgaben umfassen das Erstellen und Verwalten von abhängigen Objekten sowie die Steuerung des Ablaufs der Applikation.

IoC (Inversion of Control) definiert eine Richtlinie, die festlegt, wie ein Programm in mehrere sogenannte Module unterteilt wird und wie diese Module miteinander verbunden sind. [2]

Das **Beans-Modul** übernimmt die Verwaltung und Erzeugung von Objekten (sogenannte Beans). Beans bilden die Grundlage von Spring, werden mit DI (Dependency Injections) verwendet und vom Spring-IoC (Inversion of Control)-Container erstellt und verwaltet. Das Modul enthält außerdem eine Implementierung der `BeanFactory`.

Das **Kontext-Modul** stellt eine Erweiterung der beiden anderen Module dar. Es implementiert den `ApplicationContext`, der die Funktionalitäten der `BeanFactory` erweitert.

Zusätzlich gibt es das **Expression Language-Modul**, das die Manipulation von Datenobjekten zur Laufzeit ermöglicht.

Das Spring Framework verfügt außerdem über einen Datenzugriffsteil, der im weiteren Verlauf dieses Kapitels 2.3.3 näher erläutert wird.

[62] [63]

Spring Boot

Mit Spring Boot wird es ermöglicht, eine Spring-Anwendung mit minimalem Konfigurationsaufwand zu erstellen. Ziel ist es, mit geringem Aufwand eine lauffähige Spring-Anwendung bereitzustellen, sodass sich Entwickler*innen primär auf die Erstellung der Business-Logik konzentrieren können.

Zusätzlich bietet Spring Boot weitere Funktionen:

1. Verwaltung von Versionen und Konfigurationen für eine Vielzahl unterschiedlicher Frameworks, Server und Spezifikationen.
2. Bereitstellung von Standardoptionen für die Anwendungssicherheit.
3. Standardisierte Anwendungsmetriken mit der Möglichkeit zur Erweiterung.
4. Grundlegende Anwendungsüberwachung durch Gesundheitschecks.
5. Unterstützung verschiedener Optionen für externalisierte Konfigurationen.

[63]

Spring REST API

Mit Spring kann schnell und einfach eine REST-API implementiert werden. Dazu wird eine Klasse mit der Annotation *@RestController* versehen. Dies kann, wie im Codebeispiel 6 gezeigt, werden. Zudem muss ein Pfad mit *@RequestMapping(/api/test")* festgelegt werden.

Um einen *GET*-Endpunkt zu erstellen, wird eine Methode mit der Annotation *@GetMapping* definiert. Innerhalb von *@GetMapping(/1")* kann ein spezifischer Pfad angegeben werden, beispielsweise `"1"`. Zur Steuerung des HTTP-Status wird das *ResponseEntity*-Objekt verwendet. Standardmäßig wird mit *ResponseEntity.ok* geantwortet, jedoch können auch beliebige HTTP-Statuscodes mit *ResponseEntity.status(int)* zurückgegeben werden.

Zusätzlich kann *@PathVariable* genutzt werden, um Daten über einen *GET*-Request an den Server zu übermitteln, beispielsweise zur Abfrage eines spezifischen Objekts. Im folgenden Codebeispiel wird, abhängig von der übergebenen `id`, entweder "test1"(bei `id = 1`) oder "test2"(bei `id = 2`) zurückgegeben.

Listing 6: GET-API Beispiel

```
@RestController
@RequestMapping("/api/test")
public class TestController {
    @GetMapping
    public ResponseEntity<String> getTest() {
        return ResponseEntity.ok("test");
    }
    @GetMapping("/{id}")
    public ResponseEntity<String> getTestByID(@PathVariable int id) {
        return switch (id) {
            case 1 -> ResponseEntity.ok("test1");
            case 2 -> ResponseEntity.ok("test2");
            default -> ResponseEntity.notFound().build();
        };
    }
}
```

Ein *POST*-Request wird mit der Annotation *@PostMapping* implementiert. Um auf den Body des Requests zuzugreifen, wird *@RequestBody* verwendet. Der Body wird automatisch in ein Objekt gemappt, welches nach *@RequestBody TestObjekt testObjekt* definiert ist. In diesem Fall ist es ein Objekt der Klasse `TestObjekt`. Falls das Mapping fehlschlägt, wird eine *Bad Request*-Exception geworfen.

Listing 7: POST-API Beispiel

```
@RestController
@RequestMapping("/api/test")
public class TestController {
    List<TestObjekt> list = new ArrayList<>();
    @PostMapping("/employees")
    public TestObjekt newEmployee(@RequestBody TestObjekt testObjekt) {
        list.add(testObjekt);
        return testObjekt;
    }
}
```

Ein *PUT*-Request ist ähnlich aufgebaut wie ein *POST*-Request. Der einzige Unterschied besteht darin, dass statt *@PostMapping* die Annotation *@PutMapping* verwendet wird.

Schließlich wird ein *DELETE*-Request mit der Annotation *@DeleteMapping* realisiert.

2.3.2 Feign

Feign ist ein deklarativer Webservice-Client 2.3.2, der die Erstellung von Webservice-Clients effizienter und benutzerfreundlicher gestaltet. Um Feign zu verwenden, wird ein Interface erstellt und mit der Annotation **`@FeignClient`** versehen.

Damit Feign in einem Projekt auf Basis des Spring Frameworks genutzt werden kann, muss zusätzlich die Hauptklasse mit der Annotation **`@EnableFeignClients`** versehen werden.

Listing 8: Java Spring main method add Feign

```
1  @SpringBootApplication
2  @EnableFeignClients
3  public class Application {
4      public static void main(String[] args) {
5          SpringApplication.run(Application.class, args);
6      }
7  }
```

Im Code-Beispiel wird die Annotation **`@EnableFeignClients`** hinzugefügt, um Feign im Projekt zu aktivieren. Anschließend muss das Interface für den Client implementiert werden, um die Schnittstelle zur API bereitzustellen.

Listing 9: Java Spring Feign Client

```
1  @FeignClient("stores")
2  public interface StoreClient {
3      @RequestMapping(method = RequestMethod.GET, value = "/stores")
4      List<Store> getStores();
5
6      @RequestMapping(method = RequestMethod.GET, value = "/stores")
7      Page<Store> getStores(Pageable pageable);
8
9      @RequestMapping(method = RequestMethod.POST, value = "/stores/{storeId}", consumes =
           "application/json")
10     Store update(@PathVariable("storeId") Long storeId, Store store);
11
12     @RequestMapping(method = RequestMethod.DELETE, value = "/stores/{storeId:\\d+}")
13     void delete(@PathVariable Long storeId);
14 }
```

Es wird ein Interface erstellt, das den Anforderungen der genutzten API entspricht. Mithilfe der Annotation **`@FeignClient(ßtores")`** wird festgelegt, welche API verwendet werden soll. Anschließend kann durch DI (Dependency Injections) ein Client in einem anderen Programmteil instanziiert werden. Der Aufruf erfolgt dann beispielsweise über **`storeClient.getStores()`**. Die Verwaltung der HTTP-Anfragen und -Antworten übernimmt dabei automatisch Feign. [64]

Deklarativer Webservice-Client

Ein deklarativer Webservice-Client ist ein Ansatz zur Erstellung von Webservice-Clients, bei dem die Definition mithilfe von Annotationen, Konfigurationen oder einer deklarativen Syntax erfolgt. Der Nutzer muss sich dabei nicht mit der Verarbeitung der HTTP-Anfragen und -Antworten beschäftigen.

Ein wesentlicher Vorteil dieses Ansatzes besteht darin, dass deutlich weniger Overhead-Code entsteht, da keine manuellen HTTP-Requests implementiert werden müssen.

2.3.3 Spring JDBC

JDBC

JDBC (Java Database Connectivity) ist eine Schnittstelle, die den Zugriff auf Datenbanken in Java ermöglicht. Ein wesentliches Merkmal von JDBC ist die Unterstützung für Entwickler, indem es häufig wiederkehrende Aufgaben automatisiert. Es handelt sich um eine Low-Level-API, die es erlaubt, einfache SQL-Befehle auf einer relationalen Datenbank auszuführen. Zudem kann JDBC unabhängig vom verwendeten Datenbankverwaltungssystem genutzt werden.

[65]

Low-Level-API

Beim Arbeiten mit Datenbanken gibt es zwei grundsätzliche Ansätze: Low-Level-APIs und High-Level-APIs. Der wichtigste Unterschied zwischen beiden besteht darin, wie direkt mit der Datenbank interagiert wird und wie viel Verantwortung beim Entwickler liegt.

Bei Low-Level-APIs wie JDBC muss die Verbindung zur Datenbank manuell hergestellt und der entsprechende Treiber selbst konfiguriert werden. Die Verbindung zur Datenbank kann mit `DriverManager.getConnection(url, user, password);` aufgebaut werden. Abbildung 10 zeigt ein vollständiges Beispiel dazu. Der notwendige Treiber wird mit `Class.forName("com.mysql.cj.jdb` geladen. Nach diesen beiden Schritten kann mit der Datenbank interagiert werden.

Um eine Abfrage zu erstellen, wird in JDBC zunächst ein `Statement stmt` erzeugt. Anschließend wird ein SQL-Query definiert, der ausgeführt werden soll. Schließlich wird die Abfrage mit dem zuvor erstellten `Statement` ausgeführt. Um das Ergebnis zu speichern, wird die Klasse `ResultSet` verwendet. Diese speichert eine Menge von Zeilen und ermöglicht eine schrittweise Iteration mit `.next()`. Einzelne Werte können entweder über den Index oder den Spaltennamen abgerufen werden.

Um Speicherlecks zu vermeiden und die Datenbank-Performance zu optimieren, müssen die Verbindungen ordnungsgemäß geschlossen werden. Abbildung 10 zeigt, wie dies in JDBC umgesetzt wird.

Listing 10: Low Level JDBC

```
public class JDBCExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");

            Connection conn = DriverManager.getConnection(url, user, password);

            Statement stmt = conn.createStatement();
            String query = "SELECT * FROM employees";
            ResultSet rs = stmt.executeQuery(query);

            while (rs.next()) {
                System.out.println(rs.getInt("id") + " - " + rs.getString("name"));
            }

            rs.close();
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Spring JDBC und High Level-API

Um zu verstehen, wie Spring JDBC funktioniert, ist es wichtig, zunächst High Level-APIs zu verstehen. High Level-APIs sind grundsätzlich ähnlich wie Low Level-APIs, da es bei beiden darum geht, mit einer Datenbank zu kommunizieren. Allerdings wird bei High Level-APIs viel Boilerplate-Code, wie das Erstellen und Schließen von Datenbankverbindungen, automatisch übernommen. Spring JDBC ist eine solche High Level-API. Mit der Klasse `JdbcTemplate` können einfache Datenbankanfragen durchgeführt werden. In Abschnitt 11 wird gezeigt, wie mit `JdbcTemplate` SQL-Anfragen ausgeführt werden können. Zudem wird ein RowMapper definiert, um das ResultSet der Datenbank direkt in eine Liste von `Employee`-Objekten umzuwandeln.

Listing 11: High Level JDBC

```
@Autowired
JdbcTemplate jdbcTemplate;

public List<Employee> getAllEmployees() {
    return jdbcTemplate.query("SELECT * FROM employees",
        (rs, rowNum) -> new Employee(rs.getInt("id"), rs.getString("name")));
}
```

Spring JDBC ermöglicht auch das Einfügen, Verändern und Löschen von Datensätzen. Durch die Nutzung von Spring JDBC kommen folgende Vorteile auf.

1. Weniger Boilerplate Code
2. Automatischen verwwalten von Verbindungen
3. Einheitliche Fehlerbehebung durch eigene Exceptions
4. Spring JDBC unterstützt durch `@transactional` das einfache erstellen von Transaktionen
5. Mehrere Operationen mit einer Funktion durchführen.

Spring JDBC ist vom Konzept her um einiges einfacher als Spring JPA, welches ebenfalls eine High-Level-API für die Datenbankkommunikation ist. Doch im Gegensatz zu JPA nutzt Spring JDBC weder Lazy Loading noch Caching. Wenn ein Objekt aufgerufen wird, wird die gesamte Entität geladen, was bedeutet, dass auch Referenzen mitgeladen werden.

Spring JDBC verwendet zudem keine Sessions oder andere Mechanismen zur Nachverfolgung gespeicherter Objekte. Weiterhin bietet es nur einfache Objektmapper, sodass für komplexere Objekte eigene Objektmapper notwendig sind. All diese Punkte sind zwar Nachteile, führen jedoch zu einem großen Vorteil: Einfachheit. Das bedeutet, dass Spring JDBC deutlich leichter zu verstehen und zu erlernen ist und sich insbesondere für einfache Projekte schnell implementieren lässt. Dennoch ist es wichtig, im Voraus abzuschätzen, wie komplex die Datenbank wird, um zu vermeiden, dass später erheblicher Mehraufwand entsteht, weil benötigte Features fehlen.

Spring JDBC arbeitet mit den Konzepten von *Repositories* und *Aggregate Root* aus dem **Domain-Driven Design (DDD)**. Ein **Aggregate** ist eine Gruppe von Entitäten, die bei Änderungen stets in einem konsistenten Zustand gehalten werden. Ein **Beispiel** hierfür ist eine Bestellung (*Order*) von Lebensmitteln mit einer Anzahl gekaufter Artikel (*numOfItems*). Zusätzlich gibt es die Entität *OrderItem*, welche einzelne Bestellpositionen repräsentiert. Das **Aggregate** stellt sicher, dass die Anzahl der Lebensmittel (*numOfItems*) mit der Anzahl der vorhandenen *OrderItem* übereinstimmt. Diese Konsistenz tritt jedoch nicht sofort bei einer Änderung ein, sondern erst nach einer gewissen Zeit. Wenn ein neues *OrderItem* hinzugefügt wird, dauert es also etwas, bis sich auch der Wert von *numOfItems* entsprechend anpasst.

Zudem besitzt jedes **Aggregate** eine eigene **Aggregate Root**, über die Änderungen vorgenommen werden. Für jede **Aggregate Root** existiert ein eigenes **Repository**, um Änderungen an ihr durchzuführen. In unserem Beispiel ist die Bestellung (*Order*) die **Aggregate Root**. Die *OrderItem* gehören ebenfalls zu diesem Aggregate, da sie ohne die **Aggregate Root** nicht existieren können – es kann keine Bestellpositionen ohne eine Bestellung geben.

Ein **Repository** ist eine Möglichkeit von Spring JDBC, mit einer Sammlung aller Aggregate eines bestimmten Typs zu arbeiten. Dabei gelten jedoch folgende Bedingungen:

1. Jedes **Aggregate Root** hat sein eigenes **Repository**.

2. Jede vom **Aggregate Root** erreichbare Entität ist ebenfalls Teil dieses Aggregats. Wenn wir ein **Aggregate** speichern, werden auch alle untergeordneten Entitäten mitgespeichert.
3. Nur ein **Aggregate Root** soll eine Foreign-Key-Beziehung zu seinen Unterentitäten haben. Das bedeutet, dass *OrderItem* nur eine Beziehung zu *Order*, aber zu keinen anderen Aggregaten haben soll.

Bei der Speicherung von Aggregaten werden – sofern nicht anders konfiguriert – alle Entitäten, die zu dem Aggregat gehören und von ihrem Root referenziert werden, von Spring Data JDBC beim Speichern gelöscht und neu angelegt.

[66]

2.3.4 OAuth

OAuth ist ein Standard, der es einer Anwendung erlaubt, auf Daten zuzugreifen, die bei einem anderen Dienst gehostet werden. OAuth steht für Open Authorization, und die aktuelle Version (Stand März 2025) ist OAuth 2.0. Nach der Zustimmung des Nutzers kann die Anwendung mit OAuth auf die Daten dessen Webservers zugreifen, ohne dabei die Anmeldedaten des Nutzers kennen zu müssen. Es kann außerdem eingeschränkt werden, welche Aktionen durchgeführt werden dürfen. OAuth ist ein Autorisierungsprotokoll (nicht Authentifizierung) und ist somit nur für den Zugriff auf bestimmte Ressourcen verantwortlich, wie auf externen APIs. Dazu verwendet OAuth sogenannte Zugriffstokens. Das sind Daten, die die Autorisierung auf Ressourcen des Clients darstellen. OAuth schreibt dafür kein bestimmtes Format vor, häufig werden allerdings JSON Web Tokens (JWTs) verwendet. In JSON Web Tokens kann der Aussteller des Tokens auch andere Daten schreiben. Aus Sicherheitsgründen sollten diese jedoch ein Ablaufdatum haben, welches möglichst kurz sein sollte. Grund dafür ist, dass die Tokenlöschung bei JSON Web Tokens schwierig ist, und sollte ein Token von Angreifern entdeckt werden, können diese auf die Daten so lange zugreifen, wie der Token gültig ist.

Rollen sind Teil der Autorisierungsarchitektur von OAuth 2.0. Folgende sind die wesentlichen Bestandteile dieses Autorisierungsprotokolls:

1. `Ressourcenbesitzer`

   Der Ressourcenbenutzer ist der Nutzer, der die Daten besitzt, auf die zugegriffen werden soll. Dieser kann den Zugriff darauf erteilen.
2. `Client`

   Der Client möchte auf die geschützten Ressourcen zugreifen. Dazu benötigt er den erforderlichen Zugriffstoken.

3. **`Autorisierungsserver`**
   Wenn sich der Client authentifiziert hat und die Berechtigung des Ressourcenbesitzers hat, stellt dieser Server den Zugriffstoken an den Client aus. Dazu braucht er einerseits den Autorisierungsendpunkt, der für die Authentifizierung und die Zustimmung des Besitzers zuständig ist, andererseits den Tokenendpunkt, der bei einer Transaktion zwischen Maschine und Maschine teilnimmt.
4. **`Ressourcenserver`**
   Dieser Server schützt die Daten des Besitzers. Er empfängt die Anfragen des Clients, validiert und akzeptiert den Zugriffstoken und gibt darauf die angeforderten Ressourcen frei oder nicht.

Es wird allerdings nicht immer gleich ein Zugriffstoken erstellt, sondern oft zuerst ein Autorisierungscode ausgegeben. Dieser kann gegen einen Zugriffstoken ausgetauscht werden. Es können auch Aktualisierungstoken ausgestellt werden. Diese haben lange Ablaufzeiten und sind dazu da, dass sich der Client selbst Zugriffstoken mit kurzen Ablaufzeiten erstellen kann. Das erhöht die Sicherheit im Falle einer Tokenentdeckung.

Um OAuth 2.0 zu nutzen, muss der Client eine ClientID und ein ClientSecret vom Autorisierungsserver erhalten, um sich identifizieren zu können. In OAuth werden Anfragen immer vom Client gestellt. Das läuft folgendermaßen ab:

1. Zuerst fragt der Client beim Autorisierungsserver eine Autorisierung an. Dazu übermittelt er die Client-ID und das Client-Secret. Zudem wird ein Endpunkt-URI angegeben, an dem der Token gesendet werden soll, und die Scopes. Scopes sind dazu da, den Grund zu spezifizieren, weshalb auf die Ressourcen zugegriffen werden soll.
2. Der Autorisierungsserver authentifiziert den Client. Dazu überprüft er, ob die angegebenen Scopes zulässig sind.
3. Der Ressourcenbesitzer kommuniziert mit dem Autorisierungsserver. Er erteilt Zugriff auf die angefragten Ressourcen.
4. Danach wird dem Client ein Autorisierungscode / ein Zugriffstoken übermittelt. Möglicherweise auch ein Aktualisierungstoken.
5. Da nun der Client den Token hat, kann er auf die Ressourcen des Ressourcenservers zugreifen.

OAuth verwendet sogenannte Grants, um einen Ablauf festzulegen, den der Client ausführen muss, bevor er Zugriff auf die Ressource bekommt. Es gibt folgende Arten von Grants:

1. **`Autorisierungscode-Grant`**: Wie oben bereits erklärt, wird hierzu ein einmaliger Autorisierungscode ausgestellt. Diese Lösung empfiehlt sich für Web-Apps, wo der Austausch

auf der Serverseite sicher ablaufen kann. Diese Variante ist allerdings unsicherer als ein PKCE-Grant.

2. **`Implicit-Grant`**: Hierbei wird der Token direkt an den Client übergeben. Meistens als Antwort auf einen Form-Post.
3. **`PKCE-Grant`**: PKCE ist kurz für Äutorisierungscode-Grant mit Proof Key for Code Exchange". Dieser Ansatz ist dem Autorisierungscode ähnlich, allerdings gibt es zusätzliche Sicherheitsmaßnahmen, die den Grant sicherer für bspw. Mobile-Grants machen.
4. **`Grant mit Anmeldedaten des Ressourcenbesitzers`**: Hierbei bekommt der Client die Anmeldedaten des Besitzers. Das basiert zwar darauf, dass der Client vertrauenswürdig ist, aber es erspart die Umleitung über den Autorisierungsserver.
5. **`Grant mit Client-Anmeldedaten`**: Diese Art von Grant wird bei nicht interaktiven Anwendungen verwendet. Hier wird die Anwendung selbst durch die Client-ID und das Client-Secret authentifiziert.
6. **`Geräteautorisierungsablauf`**: Dieser ist für Geräte mit beschränkter Eingabe wie Smart-TVs.
7. **`Aktualisierungstoken-Grant`**: Hier wird ein Aktualisierungstoken ausgestellt, der zum Auffrischen des Zugriffstokens genutzt werden kann.

[67]

2.3.5 Jackson

Zum Datenaustausch werden Daten im JSON-Format gespeichert. Bei Jackson handelt es sich um eine populäre Open-Source-Java-Bibliothek, wenn mit JSON-Dateien gearbeitet wird. Es ist gut darin, Java-Objekte zu JSON zu serialisieren oder umgekehrt und das auf eine effiziente Weise. Bei der Serialisierung ist von der Konvertierung von einem JSON-String in ein Java-Objekt die Rede, wobei Deserialisierung das Gegenteil bedeutet. Wichtige Konzepte zur Verwaltung von JSON-Daten von Jackson sind:

Streaming API Hier werden die Daten als diskrete Ereignisse gelesen/geschrieben. Es stellt JsonParser/JsonGenerator bereit. Diese Methode ist die schnellste und ressourcenschonenste Methode.

Tree Model Hier wird eine Baumdarstellung des JSON-Dokuments erzeugt. Die ObjectMapper-Klasse stellt diesen Baum aus Knoten(JsonNode) her. Die XmlMapper-Klasse macht dasselbe, nur aus einem XML-File (das geht allerdings nur ab Jackson 2.x). Zwar ist diese Methode nicht so schnell wie Streaming API, jedoch bietet sie größere Flexibilität.

Data Binding Es handelt sich dabei um einen Weg Java-Objekte mit Annotationen, oder durch direkten Zugriff auf properties, zu serialisieren/deserialisieren. Die Datenbindung ist in zwei Teile unterteilt:

1. Simple Databinding (JSON in und aus null-Objekten, Map, List, String, Number und Boolean konvertieren)
2. Full Databinding (in und aus jeglichem Java-Datentyp)

Die ObjectMapper-Klasse kann dabei hilfreich sein, JSON-Strings in Java-Objekte zu deserialisieren. Angenommen es wäre folgende Java-Klasse vorhanden:

Listing 12: Example Java-Class

```
1  public class Book {
2      private String title;
3      private String author;
4
5      public Book() {
6      }
7  }
```

Auch Getter und Setter sind noch zusätzlich erforderlich. Sollten die Attribute in der Java-Klasse anders heißen als im JSON-String, kann dies mithilfe der Annotation @JsonProperty("jsonName") definiert werden. Folgender Code könnte nun einen JSON-String (hier jsonString) mithilfe von Jacksons ObjectMapper zu einem Java-Objekt von der Klasse Book deserialisieren:

Listing 13: Example Jackson Reading

```
1      public static void main(String[] args) {
2          String jsonString = "{ \"title\": \"Herr der Ringe\", \"author\": \"J.R.R.
               Tolkien\" }";
3          ObjectMapper objectMapper = new ObjectMapper();
4
5          try {
6              Book book = objectMapper.readValue(jsonString, Book.class);
7          } catch (Exception e) {
8              e.printStackTrace();
9          }
10     }
```

[68]

2.3.6 Apache Xerces

Bei Apache Xerces handelt es sich um ein Softwareprojekt, das von einer großen Community entwickelt wurde und kommerziell frei verwendbare XML-Parser, oder ähnliche Technologien bietet. Es werden viele Plattformen und Programmiersprachen unterstützt. Dieses Projekt wird von einer Vielzahl von kompetenten Individuen rund um den Globus entwickelt, die über das Internet kommunizieren und Daten austauschen.

Apache hat es sich zur Mission gemacht, die Nutzung von XML zu fördern. XML dient dazu, Rohdaten zu strukturieren und sie dadurch in für Menschen lesbare Informationen zu verwandeln. Das kann die Effizienz einiger Informationssysteme steigern. Durch die öffentlich zugänglichen XML-Parser will Apache solche Verbesserungen für jeden ermöglichen. Diese Parser unterstützen standardisierte APIs und sind für hohe Leistung, Zuverlässigkeit und Benutzerfreundlichkeit entworfen worden. Damit Ideen leicht von einer Programmiersprache auf die andere übertragen werden können, wurden die APIs, sofern durch Architektur oder Sprache möglich, ähnlich gestaltet. Außerdem sollten die Parser natürlich effizient mit anderen Apache-Projekten zusammenarbeiten. Das Apache-Team hofft auf enge Zusammenarbeit zwischen Unternehmen und Einzelpersonen, um das Bestmögliche aus diesem Projekt zu machen. Weiters werden, sofern möglich, weit verbreitete Standards für die APIs verwendet, um eine zusammenhängende Architektur zwischen den XML-Parsern und anderen Komponenten zu ermöglichen. Gleichzeitig versucht das Projekt, Innovation zu unterstützen, um Konzepte, die noch nicht standardisiert sind, voranzutreiben. [69]

Xerces2 ist die Java-Variante von Apache Xerces. Diese basiert auf dem Xerces Native Interface (kurz XNI). Dabei handelt es sich um ein vollständiges Framework zum Erstellen von Parser-Komponenten. Es ist modular und einfach zu programmieren.

So funktioniert XNI:

Ein Parser dieses Frameworks ist als Pipeline von Parserkomponenten konfiguriert. Fließen nun Daten durch diese Pipeline, wird als Ausgabe eine Art von Programmierschnittstelle erzeugt. Eine Pipeline besteht aus einer Quelle, einer bestimmten Anzahl an Filtern (nicht notwendig) und einem Ziel. Als Quelle wird häufig ein XML-Scanner verwendet, als Filter XML-Schema-Validatoren und als Ziel ein Parser von den XNI-Ereignissen zur Programmierschnittstelle (bspw. DOM).

Abbildung 9: Abbildung einer XNI-Pipeline von Apache Xerces [70]

Diese Grafik zeigt eine stark simplifizierte Darstellung einer Pipeline. Meistens handelt es sich um 2 Pipelines. Eine für die Dokumenteninformation und eine zweite für die Dokumententypdefinition (DTD). [70]

2.3.7 Mapstruct

Mapstruct ist ein Open-Source-Framework und wird von einer Community aus qualifizierten Java-Entwicklern gepflegt. Bei Mapstruct handelt es sich um einen Annotationsprozessor für Java, welcher effizienten und typsicheren Bean-Mapping-Code generiert. Um Mapstruct zu verwenden, muss nur ein Mapping-Interface definiert werden, welches alle Mappingmethoden deklariert. Der Mapping-Code wird zur Kompilierzeit erzeugt. Der generierte Code verwendet zum Mappen zwischen zwei Objekten einfache Java-Methoden wie Getter und Setter. Ohne Mapstruct würden diese Mappermethoden von Hand geschrieben werden müssen, was nicht nur zeitaufwendig ist, sondern auch fehleranfällig.

Vorteile von Mapstruct:

Schnelligkeit Dadurch, dass Mapstruct einfache Methodenaufrufe verwendet, arbeitet es schneller, als andere Mapping-Frameworks, welche Reflection verwenden.

Compile-time type safety Nur Objekte und Attribute, die zueinander kompatibel sind, können gemappt werden. Das verhindert versehentliches Mapping von zwei ähnlichen Klassen.

Klare Fehlermeldungen zur Build-Zeit Sollten Mappings unvollständig (nicht alle Zieleigenschaften gemappt), oder fehlerhaft (keine Möglichkeit zum Mappen gefunden) sein, wird bereits ein Fehler zur Build-Zeit ausgegeben.

Convention over Configuration Mapstruct verwendet Standardeinstellungen. Sollte allerdings ein spezielles Verhalten gewünscht sein, so geht es bei Konfiguration und Implementierung aus dem Weg.

Hier ein Beispiel zu Mapstruct, wobei ein Datenbankobjekt (BookEntity) zu einem Data-Transfer-Object (BookDTO) gemappt werden soll:

Listing 14: Example Database Object

```
1 public class BookEntity {
2     private Long id;
3     private String bookTitle;
4     private String writerName;
5     private int publishedYear;
6 }
```

Listing 15: Example DTO

```
1 public class BookDTO {
2     private String title;
3     private String author;
4 }
```

So muss das Mapping-Interface definiert werden:

Listing 16: Example Mapstruct

```
@Mapper
public interface BookMapper {
    BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

    @Mapping(source = "bookTitle", target = "title")
    @Mapping(source = "writerName", target = "author")
    BookDTO bookEntityToBookDTO(BookEntity entity);
}
```

Mapper-Interfaces müssen mit der @Mapper-Annotation gekennzeichnet werden. Die @Mapping-Annotation ist dazu da, um zwei Attribute zu mappen, die unterschiedlich heißen. Der Mappingcode wird beim nächsten Kompilieren generiert. [71]

2.3.8 Project Lombok

Project Lombok ist eine Open-Source-Javabibliothek, die es sich zur Aufgabe gemacht hat, Boilerplate-Code in Java zu verringern. Das Ziel dabei ist, dass dadurch die Produktivität von Entwicklern gesteigert werden soll. Es hält den Code übersichtlich und spart Zeit, die sonst für das Schreiben häufig verwendeter Methoden aufgebracht werden würde.

Lombok bietet viele Annotationen, die an der richtigen Stelle im Code platziert werden müssen, anstatt den Code selbst zu schreiben. Hier sind einige interessante Annotationen:

1. **`@ToString`**

   Wird diese Annotation am Beginn einer Klasse platziert, so wird automatisch eine toString()-Methode erzeugt. Standardmäßig sieht die Ausgabe einer Klasse, die diese Annotation verwendet, beispielsweise so aus: ”MyClass(foo=123, bar=234)”. Es können einige Dinge angepasst werden. Beispielsweise können vor Attributen dieser Klasse die Annotationen @ToString.Exclude/@ToString.Include gesetzt werden, um diese in die Methode aufzunehmen, oder sie davon auszuschließen.

2. **`Konstruktoren`**

   Je nachdem, welcher Konstruktor benötigt wird, gibt es mehrere Annotationen. Mithilfe von @NoArgsConstructor, wird ein Konstruktor ohne Parameter erstellt. Sollte das nicht möglich sein, da Konstanten oder andere Variablen initialisiert werden müssen, so wird ein Error zur Kompilierzeit geworfen. Es sei denn, es wird force auf true gesetzt: @NoArgsConstructor(force=true). Das bewirkt, dass jeder notwendige Wert mit 0, false, oder null initialisiert wird.

   Weiters gibt es @RequiredArgsConstructor. Diese erzeugt einen Konstruktor, der nur Parameter für die notwendigen Variablen benötigt.

   Als letztes gibt es die Annotation @AllArgsConstructor. Diese benötigt Werte für jedes

Attribut der Klasse. Die Reihenfolge der Parameter stimmt mit der Reihenfolge der Attribute in der Klasse überein.

3. **`@Getter/@Setter`**

   Diese Annotationen können vor jeder Variable platziert werden, um die standardmäßige Getter-/Setter-Methode dazu zu generieren. Die Methoden heißen "getFoo/ßetFoo". Grundsätzlich sind diese Methoden public, das kann aber mit "AccessLevel" konfiguriert werden.

4. **`@EqualsAndHashCode`**

   Diese Annotation sorgt dafür, dass automatisch eine Equals- und eine HashCode-Methode erstellt wird. Hier können auch wieder einige Variablen mit @EqualsAndHashCode.Include/.Exclude gekennzeichnet werden, um sicherzugehen, dass diese verwendet werden. Sollte es sich um eine Klasse handeln, die von einer anderen erbt, so kann callSuper=true angegeben werden, um deren hashCode-, equals-Methoden in die neu erzeugten Methoden der Kindklasse miteinzubeziehen.

5. **`@Data`**

   @Data kombiniert @ToString, @RequiredArgsConstructor, @Getter, @Setter und @EqualsAndHashCode in einer Annotation. Zusammengefasst, es wird all der für POJOs und Beans typische Boilerplate-Code mithilfe von Lombok auf einmal generiert.

[72]

2.4 Strukturen der Software

2.4.1 Microservices

3 Implementierung

3.1 JavaSpring RestAPI

3.1.1 Programm Aufbau

Der Klassenaufbau des Programms wird in 10 in einem UML-Diagramm dargestellt. Es zeigt die drei wichtigsten Bestandteile der ETF-Applikation und veranschaulicht zudem, welche Datentypen im Fundservice zu einem Gesamtobjekt zusammengeführt werden.

Der External API/Risk Service stammt von Unisoftware Plus, der Firma, mit der wir dieses Projekt umgesetzt haben. Er besteht aus den Komponenten *InstrumentIdentifier* und *spotPriceApiClient*, die im Kapitel **??** näher beschrieben werden.

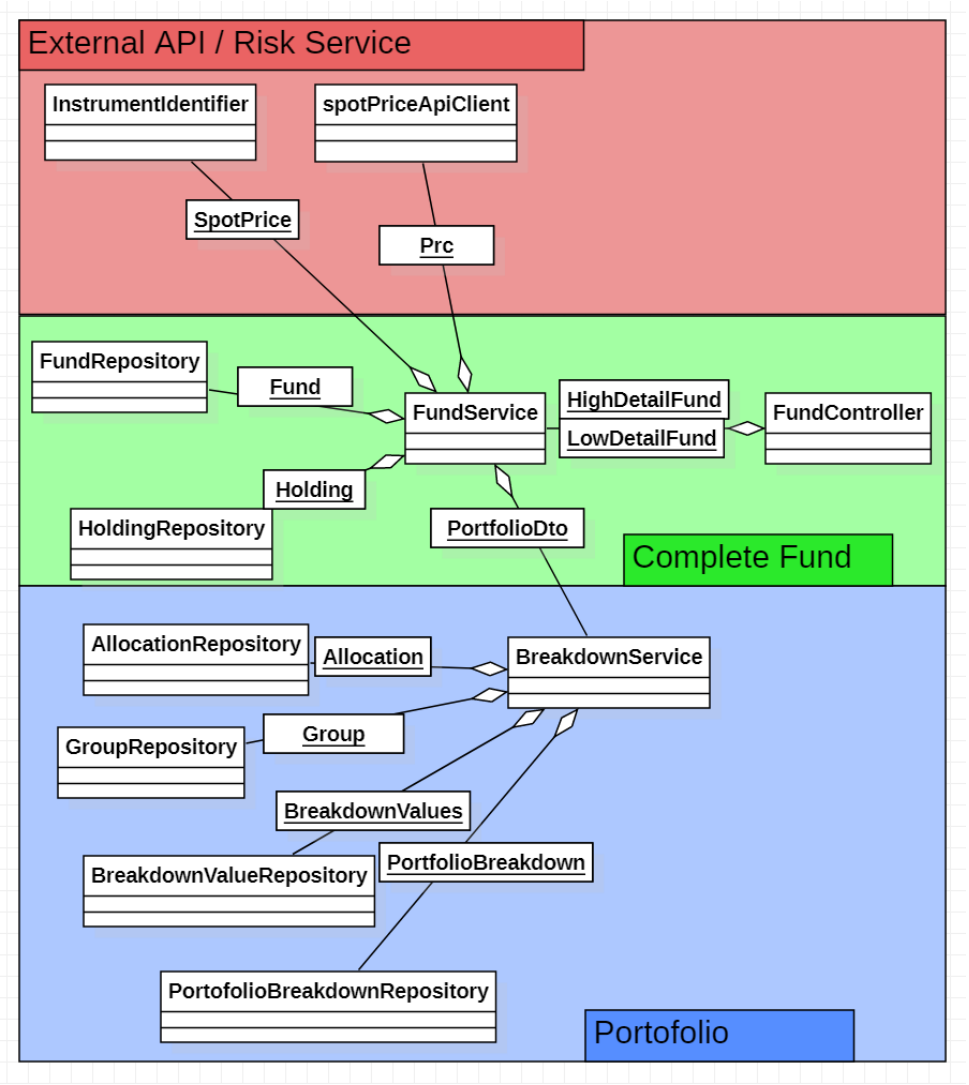


Abbildung 10: Programmaufbau von der ETF Applikation

Der zweite Teil des Systems ist das Portfolio, das aus mehreren Klassen besteht, die sich mit der Verwaltung von *Portfoliobreakdowns* befassen. In der Klasse *Portfoliobreakdown* werden verschiedene Einteilungen für einen ETF (Exchange Traded Fund) gespeichert. Ein Beispiel hierfür ist die geografische Herkunft der enthaltenen Aktien.

Das Portfolio setzt sich aus mehreren Repositories zusammen, wie in 11 dargestellt. Diese Repositories kommunizieren mit der Datenbank, wobei ein konkretes Beispiel im Kapitel zur JDBC-Implementierung 3.1.6 zu finden ist. Alle unterschiedlichen Daten werden im *Breakdown-repository* zusammengeführt. Dazu gehören unter anderem *Allocation*, *Group*, *BreakdownValues* und *PortfolioBreakdown*.

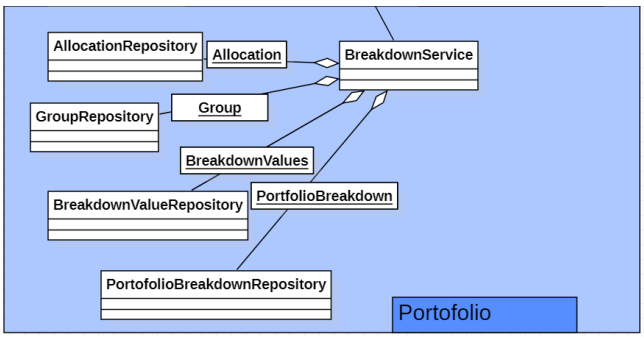


Abbildung 11: Portfoliobreakdown aufbau

Der letzte und wichtigste Bestandteil ist die Klasse *Fundservice*, die alle Objekte zusammenführt und daraus ein *HighDetailFund*-Objekt erstellt. Zusätzlich existiert auch das *LowDetailFund*-Objekt, das jedoch nur verwendet wird, wenn eine Liste aller Fonds abgefragt wird.

Das *LowDetailFund*-Objekt enthält lediglich die Grundinformationen, die vom *FundRepository* aus der Datenbank abgerufen werden. Im vollständigen Fund-Modell gibt es außerdem zwei Repositories sowie einen *FundController*, der die Endpunkte für die API bereitstellt.

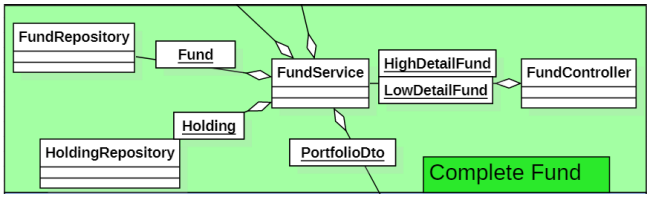


Abbildung 12: Enter Caption

3.1.2 Datenlaufbahn

13 zeigt wie die Klassenaufgebaut sind und miteinander verbunden. Es kommen manchmal "...äm ende vor wenn es noch zusätzliche Attribute gibt die aber für das Diagramm nicht relevant sind.

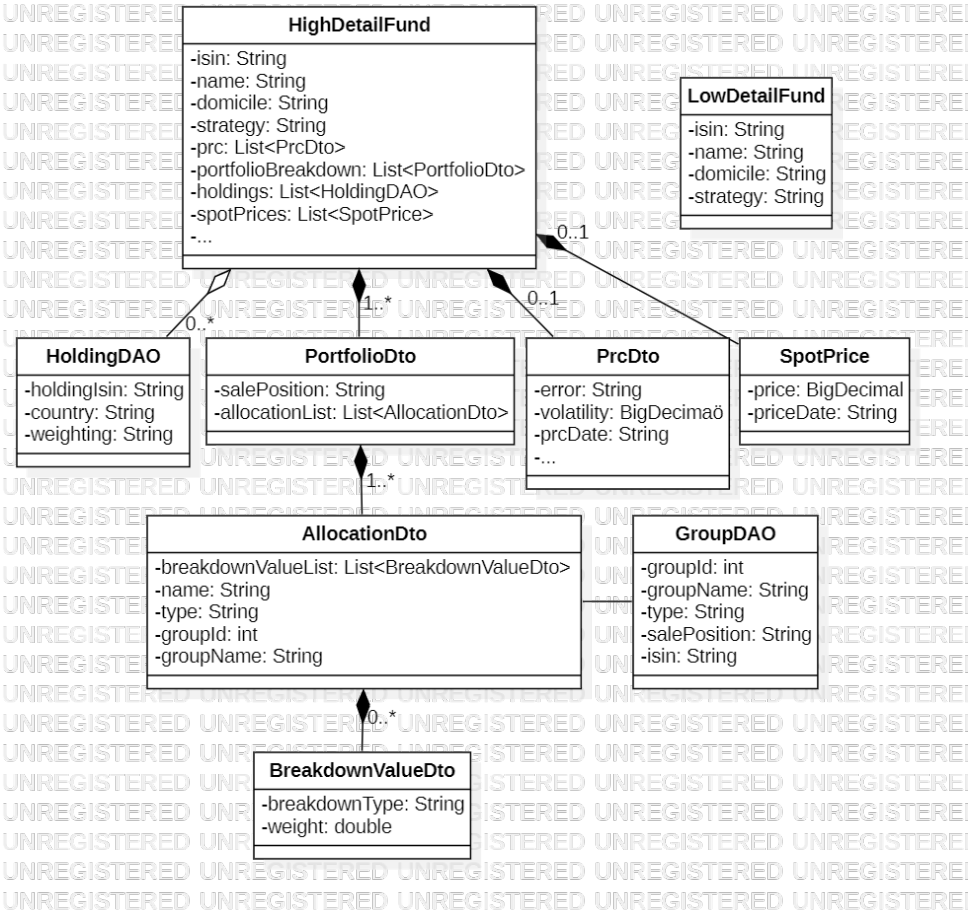


Abbildung 13: Klassendiagramm von der RESTAPI

AllocationDto, BreakdownValueDto und PortfolioDto

Eine Allocation ist eine Klasse zur Speicherung von BreakdownValues. Ein Beispiel für eine Allocation ist:

- **Name**: Verteilung der Aktien auf Kontinente
- **type**: 1
- **groupId**: 1 (die Gruppe wird nur gesetzt wenn die Allocation keine wurzel Allocation ist.)
- **groupId**: Verteilungen
- **List<BreakdownValues>**: "CH",40;"DE",30;ÄU",30

BreakdownValues bestehen wie oben zu sehen sind aus jeweils einem String und einer Gewichtung diese sind immer in % angegeben. Also eine Allocation ist eine Einteilung von dem ETF (Exchange Traded Fund) in verschiedene Unterteilungen.

PortfolioDto ist noch eine Überklasse in welcher eine Liste an Allocations gespeichert werden. PortfolioDto ist wichtig denn in einem ETF gibt es mehrere Salepositions welche oft gleiche Allocations haben.

Low- und HighDetailFund

Die beiden Modellklassen sind Objekte, die später nach außen weitergegeben werden. *LowDetailFund* besteht nur den wichtigsten ETF informationen um eine Liste an ETFs dazustellen. Außerdem ist es unabhängig von dem restlichen Model und braucht nur einen Datenbank aufruf.

HighDetailFund ist wie der Name schon aussagt eine Klasse bei alle möglichen Information zusammengetragen werden. Sie hat eine Liste an allen Aktienpositionen im ETF und auch deren Verteilungen. Weiter noch hollt es sich vom Risk Service aktuelle Daten für dem ETF. Diese bestehen aus ein paar berechnete Kennzahlen und die Kursentwicklung von dem ETF.

PrcDto

PRC (siehe Listing 17) ist ein Datentyp, dessen Parameter vom Risk-Service berechnet werden und sich häufig ändern. Er besteht aus einem String namens *Error*, der standardmäßig *null* ist, es sei denn, es tritt ein Fehler bei der Abfrage auf.

Zusätzlich gibt es das Attribut *prcDate*, das als String das Datum speichert, an dem die Kennzahlen berechnet wurden. Die restlichen sechs Parameter sind Kennzahlen, die sich auf einen ETF (Exchange Traded Fund) beziehen. Beispielsweise gibt die *volatility* an, wie stark ein ETF (Exchange Traded Fund) in einem bestimmten Zeitraum schwankt.

Listing 17: PRC Object

```
1  public class PrcDto {
2      private String error;
3      private BigDecimal volatility;
4      private BigDecimal valueAtRisk;
5      private BigDecimal impliedCreditSpread;
6      private String prcDate;
7      private BigDecimal prcGlobal;
8      private BigDecimal prcCredit;
9      private BigDecimal prcMarket;
10 }
```

3.1.3 Services

BreakdownService

Die Aufgabe des *BreakdownService* besteht darin, die Objekte, die von den Repositories stammen, zusammenzuführen. Es umfasst drei Funktionen: eine Hauptfunktion sowie zwei Unterfunktionen, die von außen nicht zugänglich sind.

Die Klasse implementiert zudem folgende Repositories:

- *PortfolioBreakdownRepository*

- *RepositoryBreakdown*
- *ValueRepository*
- *AllocationRepository*
- *GroupRepository*

Diese werden mit Lombok per DI (Dependency Injections) (Dependency Injection) implementiert.

Die Hauptfunktion heißt *getAllPortfolioBreakdownByIsin*. Sie erhält eine ISIN eines ETFs als Eingabe und gibt eine Liste von *PortfolioDto*-Objekten zurück. Die ISIN ist eine eindeutige Identifikationsnummer für einen ETF (Exchange Traded Fund) und besteht aus einem Länderkürzel (z. B. CH für die Schweiz) sowie einer zehnstelligen Nummer.

Die Funktion hat die Aufgabe, für die übergebene ISIN alle zugehörigen Informationen und Allocations aus der Datenbank abzurufen. Falls die ISIN nicht in der Datenbank vorhanden ist, wird eine *DataNotFoundException* zurückgegeben.

Der erste Teil der Funktion dient der Fehlerbehandlung, um darauf zu reagieren, wenn für die übergebene ISIN kein Wert gefunden wurde. Im Codebeispiel **??** ist zu sehen, dass die Liste auf Leerheit überprüft wird, da das Repository nicht *null* zurückgibt, wenn es keine Daten findet, sondern eine leere Liste. Falls dieser Fehlerfall eintritt, wird eine *DataNotFoundException* an die übergeordnete Klasse weitergegeben.

Listing 18: Fehrlerbehandlung

```
1       List<PortfolioBreakdownDAO> breakdownList =
            portfolioBreakdownRepository.findAllByIsin(isin);
2       if (breakdownList.isEmpty()) {
3           throw new DataNotFoundException("No portfolio breakdowns found for ISIN: " + isin);
4       }
```

Der zweite Teil der Funktion (siehe Listing 19) befasst sich mit der Erstellung der Ergebnisliste. Bisher liegen lediglich die einzelnen *PortfolioBreakdowns* vor, jedoch enthalten diese noch keine Unterdaten. Diese werden nun aus der Datenbank mit dem *allocationRepository* abgefragt, indem die Fund-ISIN sowie die *breakdown.getSalePosition()* der zuvor abgerufenen *PortfolioBreakdownDAO*-Objekte übergeben werden.

Ein entscheidender Schritt in dieser Funktion ist die Umwandlung der Datenbank-Objekte. In der Datenbank werden sogenannte *DAO*-Objekte gespeichert, die nicht objektorientiert aufgebaut sind. Diese müssen mithilfe eines Mappers in objektorientierte Objekte umgewandelt werden. Der Mapper ist eine der beiden Unterfunktionen und wird später genauer erläutert. Er gibt eine Liste von *AllocationDto*-Objekten zurück, die vorerst gespeichert wird.

Der gleiche Vorgang wird anschließend für *Groups* wiederholt. Alle Gruppen, die zur ISIN und *breakdown.getSalePosition()* passen, werden aus der Datenbank geholt. Diese müssen jedoch nicht gemappt werden, da sie einen Sonderfall darstellen: Das *Group*-Objekt wird nicht objektorientiert mit anderen Klassen verbunden, weshalb direkt das *DAO*-Objekt weiterverwendet wird.

Als nächstes werden die *PortfolioBreakdownDAO*-Objekte in *DTO*-Objekte überführt. Für einfache Mapping-Aufgaben gibt es ein Interface namens *DtoMapper*, das mithilfe der Bibliothek *mapstruct* die Daten umwandelt.

Abschließend wird die zweite Unterfunktion aufgerufen, die jeder *AllocationDto* die zugehörige Gruppe zuweist. Danach wird die fertige Liste dem *Portfolio*-Objekt zugewiesen und zurückgegeben.

Listing 19: Ergebnis Liste erstellen

```
return breakdownList.stream().map(breakdown -> {
    List<AllocationDAO> allocations =
        allocationRepository.findAllByIsinAndSalePosition(isin,
        breakdown.getSalePosition());
    List<AllocationDto> allocationDtoList = mapAllocationsToBreakdownValues(allocations);
    List<GroupDAO> groups = groupRepository.findAllForPortfolioBreakdown(isin,
        breakdown.getSalePosition());

    PortfolioDto portfolioDto =
        DtoMapper.INSTANCE.PortfolioBreakdownToPortfolioDto(breakdown);
    mapGroupToAllocationDto(groups, allocationDtoList);
    portfolioDto.setAllocationList(allocationDtoList);

    return portfolioDto;
}).collect(Collectors.toList());
```

Der Code in 20 hat die Aufgabe, für eine bestimmte Gruppen-ID, die in *AllocationDTO* gespeichert ist, die zugehörige Gruppe zu finden und deren Namen in *AllocationDTO* zu setzen.

Listing 20: Funktion um Group Allocations zuzuweisen

```
/**
 * a Method to save the group name into the funds of the portfolio
 * the Method directly changes the AllocationDto list
 *
 * @param groups            all available groups
 * @param allocationDtoList the list of all allocations
 */
private void mapGroupToAllocationDto(List<GroupDAO> groups, List<AllocationDto>
    allocationDtoList) {
    // Create a map for quick lookup of Group by groupId
    Map<Integer, GroupDAO> groupMap =
        groups.stream().collect(Collectors.toMap(GroupDAO::getGroupId, group -> group));
    // Iterate through the allocationDtoList stream and update the Group information
    allocationDtoList.stream().forEach(allocationDto -> {
        Optional<GroupDAO> groupOptional =
            Optional.ofNullable(groupMap.get(allocationDto.getGroupId()));
        groupOptional.ifPresent(group -> {
            allocationDto.setGroupId(group.getGroupId());
            allocationDto.setGroupName(group.getGroupName());
        });
    });
}
```

Die zweite Unterklasse 21 ist ebenfalls ein Mapper, hat jedoch zusätzlich die Aufgabe, den *AllocationDTO*-Objekten die jeweiligen *BreakdownValues* zuzuweisen. Dabei werden *BreakdownValueDAO*-Objekte mit allen Parametern von *AllocationDTO* gesucht, da sie dieselben Parameter speichern. Das eigentliche Mapping erfolgt erneut über den bereits zuvor genannten *DtoMapper*.

Listing 21: Funktion um Group Allocations zuzuweisen

```
1  /**
2   * a Method to take a list of allocation and map them to a list of allocationDtos
3   * but also fill them with BreakdownValues
4   *
5   * @param allocations a list of allocation without breakdown values
6   * @return returns a List of AllocationDto with all breakdown values added
7   */
8  private List<AllocationDto> mapAllocationsToBreakdownValues(List<AllocationDAO>
       allocations) {
9      return allocations.stream().map(allocation -> {
10         List<BreakdownValueDAO> breakdownValues =
               breakdownValueRepository.findAllByIsinAndSalePositionAndNameAndType(allocation.getIsin(),
               allocation.getSalePosition(), allocation.getName(), allocation.getType());
11         List<BreakdownValueDto> breakdownValueDtoList =
               Optional.ofNullable(breakdownValues).map(DtoMapper.INSTANCE::breakdownValueListToBreakdownV
12         AllocationDto allocationDto =
               DtoMapper.INSTANCE.AllocationToAllocationDto(allocation);
13         allocationDto.setBreakdownValueList(breakdownValueDtoList);
14         return allocationDto;
15     }).collect(Collectors.toList());
16 }
```

FundService

Die *FundService*-Klasse ist so aufgebaut, dass ihre Funktionen direkt an den Controller angepasst sind. Der Controller enthält die Methoden *getAllFunds()* und *getFundByIsin()*, und diese Funktionen sind in der *FundService*-Klasse ebenfalls vorhanden. Während der Controller hauptsächlich für die Fehlerbehandlung zuständig ist, werden im *FundService* die eigentlichen Objekte erstellt.

Im Programm wird der Begriff *Fund* anstelle von *ETF* verwendet, beispielsweise auch in *FundService*. Dieser Begriff wird hier als generelle Bezeichnung für ETFs genutzt und kann als Synonym betrachtet werden.

Die Funktion *getAllFunds()* (siehe Listing 22) gibt eine Liste aller aktuell in der Datenbank gespeicherten ETFs zurück. Im Rahmen der Diplomarbeit wurde diese Funktion noch nicht für Techniken wie *Lazy Loading* optimiert, da die Anzahl der ETFs noch so gering war, dass dies nicht erforderlich war.

Die Funktion ruft mithilfe des *fundRepository* alle ETFs ab und versucht, sie direkt auf *LowDetailFundDto* zu mappen. Für die einfache Anzeige von ETFs in einer Liste sind viele Detailinformationen nicht relevant. Außerdem könnte häufiges Abfragen externer APIs (wie des Risk Service) zu unnötiger Netzwerkauslastung führen.

Falls der Mapper einen Fehler wirft, weil keine ETFs gefunden wurden, wird eine *FundNotFoundException* ausgelöst.

Listing 22: Alle ETFs werden abgefragt

```
/**
 * A Method to retrieve all funds available
 *
 * @return a dto that contains only contains name, isin and domicile
 */
public List<LowDetailFundDto> getAllFunds() throws FundNotFoundException {
    try {
        List<FundDAO> funds = fundRepository.findAll();
        return DtoMapper.INSTANCE.fundListToLowDetailFundList(funds);
    } catch (Exception e) {
        throw new FundNotFoundException("Failed to retrieve any funds");
    }
}
```

Die zweite Methode 23, *getFundByIsin()*, ist für das Abrufen eines *HighDetailFundDto* vorgesehen. Hierbei wird eine ISIN eines bestimmten ETFs übergeben, und anschließend werden alle gespeicherten Informationen zu diesem ETF zurückgegeben.

Zunächst wird mithilfe der ISIN versucht, den ETF abzurufen. Falls dies fehlschlägt, wird eine *FundNotFoundException* geworfen. Bei erfolgreicher Abfrage wird das DAO-Objekt in ein DTO-Objekt umgewandelt, das diesmal den Namen *HighDetailFund* trägt (anstatt die Endung *DTO* zu verwenden).

Im Code-Snippet wird zudem eine separate Methode zur Abfrage der PRC-Daten aufgerufen, die in 3.1.3 näher erläutert wird.

Listing 23: Mitgabeparameter und Rückgabewerte für getFundByIsin

```
/**
* A method to retrieve all information about a particular fund per Isin
* This includes: general fund data, prc data, holdings, portfolio breakdown
*
* @param fundIsin the isin of the fund to retrieve
* @return a dto that contains all information
*/
public HighDetailFundDto getFundByIsin(String fundIsin) throws FundNotFoundException,
    PrcNotFoundException, DataNotFoundException {
    FundDAO fund =
        Optional.ofNullable(fundRepository.getFundByIsin(fundIsin)).orElseThrow(() -> new
        FundNotFoundException("Fund not found for ISIN: " + fundIsin));
    HighDetailFundDto highDetailFunds =
        DtoMapper.INSTANCE.fundListToHighDetailFundList(fund);
    List<PrcResponse> prcResponse = getPrcByIsin(fundIsin);
    if (prcResponse.isEmpty()) {
       throw new PrcNotFoundException("Prc data not found for ISIN: " + fundIsin);
    }
    ...
    return highDetailFunds;
}
```

Bei PRC (siehe 3.1.2) handelt es sich um Werte, die von der Kooperationsfirma berechnet werden. Diese Werte werden über eine externe API abgefragt. Für die Abfrage muss lediglich die ISIN eines ETF (Exchange Traded Fund) übermittelt werden.

Dabei wird ein *InstrumentIdentifier* erstellt, der an den *securityApiClient* übergeben wird, um die Informationen über den Body der Anfrage zurückzuerhalten. Der *securityApiClient* ist ein importierter Client, der von der Firma Uni Software Plus über Artifactory bereitgestellt wird.

Listing 24: PRC daten werden von dem Risk Service abgefragt

```
1 public List<PrcResponse> getPrcByIsin(String fundIsin) {
2     InstrumentIdentifier instrumentIdentifier = new InstrumentIdentifier().isin(fundIsin);
3     List<InstrumentIdentifier> instrumentIdentifiers = List.of(instrumentIdentifier);
4     return
          Optional.ofNullable(securityApiClient.getPrcsByInstrumentIdentifiers(instrumentIdentifiers).get
5 }
```

Im Code-Snippet 3.1.3 werden die zum ETF (Exchange Traded Fund) gehörenden Details abgefragt. Zusätzlich werden alle Aktienpositionen, die hier als *Holdings* bezeichnet werden, an das *HighDetailFund*-Objekt angefügt. Abschließend wird die Methode *getSpotPriceHistoryByIsin(...)* (siehe 3.1.3) aufgerufen, um die Spotpreise (siehe **??**) über eine weitere API abzurufen. Zusätzlich gibt es einige Fehlerbehandlungen, bei denen entsprechende Exceptions geworfen werden.

Listing 25: Alle ETF (Exchange Traded Fund)s werden abgefragt

```
1  public HighDetailFundDto getFundByIsin(String fundIsin) throws FundNotFoundException,
       PrcNotFoundException, DataNotFoundException {
2      ...
3      List<PortfolioDto> portfolioBreakdown =
           breakdownService.getAllPortfolioBreakdownByIsin(fundIsin);
4      if (portfolioBreakdown.isEmpty()) {
5          throw new FundNotFoundException("Portfolio breakdown not found for ISIN: " +
               fundIsin);
6      }
7      highDetailFunds.setPortfolioBreakdown(portfolioBreakdown);
8      List<HoldingDAO> holdings = holdingRepository.findAllByISIN(fundIsin);
9      if (holdings.isEmpty()) {
10         throw new FundNotFoundException("Holdings not found for ISIN: " + fundIsin);
11     }
12     highDetailFunds.setHoldings(holdings);
13     highDetailFunds.setSpotPrices(getSpotPriceHistoryByIsin(fundIsin).get(0).getSpotPrices());
14     return highDetailFunds;
15 }
```

Das Abfragen der Kursentwicklung funktioniert vom Prinzip her ähnlich wie die PRC-Abfrage (siehe 3.1.2). Da es sich hierbei jedoch um eine Zeitreihe handelt, wird eine Liste von Wertepaaren zurückgegeben. Diese bestehen jeweils aus einer Zahl (aktueller Kurswert) und dem zugehörigen Datum.

Im Code-Snippet 26 werden zudem die Konstanten *STARTDATE* und *ENDDATE* gesetzt. Dabei wird das *ENDDATE* auf das aktuelle Datum *now()* gesetzt, während das *STARTDATE* ein Jahr davor liegt. Da das Datum in einem spezifischen Format vorliegen muss, wird ein *DateTimeFormatter* verwendet, um das richtige Format sicherzustellen.

Listing 26: SpotPriceHistory wird abgefragt

```
1  private static final DateTimeFormatter dateFormatter =
       DateTimeFormatter.ofPattern("yyyy-MM-dd");
2  public static final String ENDDATE = LocalDateTime.now().format(dateFormatter);
3  public static final String STARTDATE =
       LocalDateTime.now().minusYears(1).format(dateFormatter);
4
5  public List<SpotPriceHistory> getSpotPriceHistoryByIsin(String isin) {
6          List<SpotPriceHistoryRequest> spotPriceHistoryList = new ArrayList<>();
7          SpotPriceHistoryRequest spotPriceHistoryRequest = new SpotPriceHistoryRequest();
8          spotPriceHistoryRequest.instrumentIdentifier(new
               InstrumentIdentifier().isin(isin));
9          spotPriceHistoryRequest.startDate(STARTDATE);
10         spotPriceHistoryRequest.endDate(ENDDATE);
11         spotPriceHistoryList.add(spotPriceHistoryRequest);
12         return spotPriceApiClient.getSpotPrices(spotPriceHistoryList).getBody();
13     }
```

3.1.4 Controller

Der Controller ist die Klasse, die die Endpunkte für die REST-API implementiert. Es gibt zwei GET-Endpunkte, da entschieden wurde, dass die API nur die nötigsten Funktionen bereitstellen soll. Aus diesem Grund wurden nicht alle CRUD-Operationen implementiert. Die API wurde mit dem Spring-Framework entwickelt. Eine genauere Erklärung zu Spring findet sich in Kapitel 2.3.1. Der Controller verwendet den zuvor beschriebenen *FundService* und verfügt über einen speziellen *Logger*, um eine robustere Fehlerausgabe zu ermöglichen.

Die API stellt zwei Endpunkte bereit: Der erste Endpunkt (siehe 27) ist ein GET-Endpunkt, der eine Liste von *LowDetailFundDto*-Objekten zurückgibt. Die Hauptlogik wird bereits im *FundService* ausgeführt, während der Controller lediglich den Pfad definiert, unter dem die API erreichbar ist.

In diesem Fall lautet die URL: **“http://[URL]/api/funds”**.

Zusätzlich übernimmt der Controller die Fehlerbehandlung. Falls ein Fehler auftritt, erhält der Client einen entsprechenden HTTP-Fehlerstatuscode. Wenn die Anfrage erfolgreich war, wird der HTTP-Statuscode *OK* zurückgegeben. Die eigentliche Erstellung des HTTP-Response-Pakets übernimmt das Spring-Framework.

Listing 27: Funktion um alle ETFs in der Datenbank abzufragen

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/funds")
public class FundController {
    /**
     * Endpoint to retrieve all available funds.
     *
     * @return a ResponseEntity containing a list of LowDetailFundDto
     */
    @GetMapping
    public ResponseEntity<List<LowDetailFundDto>> getAllFunds() {
        logger.info("Fetching all funds");
        List<LowDetailFundDto> funds = null;
        try {
            funds = fundService.getAllFunds();
        } catch (FundNotFoundException e) {
            logger.error("Error:", e.fillInStackTrace());
            return ResponseEntity.status(404).body(null);
        }
        return ResponseEntity.ok(funds);
    }
    ...
}
```

Der zweite Endpunkt (siehe 28) ist, ähnlich wie der erste Endpunkt, hauptsächlich für die Fehlerbehandlung zuständig. Beim Aufruf mit einer gültigen ISIN wird ein ETF mit allen zugehörigen Details zurückgegeben. Falls jedoch eine ungültige ISIN übermittelt wird, erhält der Client einen *404 Not Found* HTTP-Statuscode als Antwort.

Listing 28: Controller Funktion um Details über einen Spezifischen ETF

```java
/**
 * Endpoint to retrieve detailed information about a specific fund by its ISIN.
 *
 * @param isin the ISIN of the fund to retrieve
 * @return a ResponseEntity containing the HighDetailFundDto
 */
@GetMapping("/{isin}")
public ResponseEntity<HighDetailFundDto> getFundByIsin(@PathVariable("isin") String isin) {
    logger.info("Fetching fund details for ISIN: {}", isin);
    try {
        HighDetailFundDto fund = fundService.getFundByIsin(isin);
        return ResponseEntity.ok(fund);
    } catch (FundNotFoundException | PrcNotFoundException | DataNotFoundException e) {
        logger.error("Error: {}", isin, e);
        return ResponseEntity.status(404).body(null);
    } catch (Exception e) {
        logger.error("Error fetching fund details for ISIN: {}", isin, e);
        return ResponseEntity.status(500).body(null);
    }
}
```

3.1.5 Exceptions

Das Programm verwendet drei verschiedene Exceptions, um die Fehlerbehandlung zu verbessern:

- **DataNotFoundException** Diese allgemeine Exception wird ausgelöst, wenn eine gesuchte Ressource nicht gefunden wird, sofern sie nicht vom Risk Service stammt oder ein gesamter Fund betroffen ist.

- **FundNotFoundException** Diese spezielle Exception wird geworfen, wenn ein gesamter Fund nicht gefunden wird, da dies eine besondere Relevanz hat.
- **PrcNotFoundException** Obwohl es „PRC“ im Namen enthalten ist, wird diese Exception für alle Fehler verwendet, die beim Abfragen des Risk Service auftreten.

3.1.6 JDBC

Im Projekt wurde Spring JDBC verwendet. In 2.3.3 wird erklärt, wie es funktioniert. Spring JDBC wurde im Projekt eingesetzt, da es vom Auftraggeber der Unisoftware Plus vorgegeben wurde. Im Rahmen dieses Prozesses wurde auch JPA in Betracht gezogen, jedoch entschied man sich für JDBC. Obwohl JDBC weniger Features bietet, wurde beschlossen, dass diese zusätzlichen Funktionen nicht benötigt werden.

Listing 29: Java JDBC

```
1
2 public interface HoldingRepository extends CrudRepository<HoldingDAO, String> {
3     @Query("select * from holding where isin = :isin")
4     List<HoldingDAO> findAllByISIN(@Param("isin") String fundIsin);
5 }
```

Im folgenden Codebeispiel wird eine JDBC-Verbindung dargestellt, wie sie im Projekt verwendet wird. Das selbst erstellte Interface **`HoldingRepository`** wird hierbei mit **`CrudRepository`** erweitert.

Das gezeigte Beispiel dient dazu, Datenbankanfragen zu stellen, um alle Aktien für einen bestimmten ETF abzufragen. Obwohl das Beispiel nur einen Ausschnitt des Data Access Layers zeigt, sind die Repositories im Projekt grundsätzlich ähnlich aufgebaut.

Durch die Verwendung von **`CrudRepository`** erhält das Interface grundlegende JDBC-Funktionalitäten. Zu den bereitgestellten Funktionen gehören beispielsweise **`findAll()`** zum Abrufen aller Datensätze, **`save()`** zum Speichern eines Objekts und **`delete()`** zum Löschen einer Aktienposition.

Darüber hinaus wird definiert, welche Klasse den Rückgabewert darstellt, in diesem Fall **`HoldingDAO`**, sowie der Datentyp des Primärschlüssels. Mithilfe der Annotation **`@Query`** können benutzerdefinierte SQL-Abfragen erstellt werden.

Das Interface kann anschließend in einer anderen Klasse mithilfe von DI (Dependency Injections) eingebunden werden. Für jede Tabelle gibt es ein eigenes Interface.

3.1.7 JUnit Tests

Zum Testen der Endpunkte wurden passende Unittests erstellt. Dazu wurde JUnit (siehe Abschnitt 2.2.6) verwendet. Der Anfang der Testklasse musste so aussehen:

Listing 30: Fund Controller Test Class

```
1
2  @SpringBootTest
3  @AutoConfigureMockMvc
4  public class FundControllerTest {
5      @Autowired
6      private MockMvc mockMvc;
7      @Autowired
8      private ObjectMapper objectMapper;
9  }
```

Die Annotation @SpringBootTest ist eine Alternative für das standardmäßige @ContextConfiguration, die verwendet wird, wenn Spring-Funktionen benötigt werden. Die Annotation @AutoConfigureMockMvc hingegen ist dazu da, die automatische Konfiguration der MockMvc-Klasse vorzunehmen. Die MockMvc-Klasse dient als Haupteinstiegspunkt der serverseitigen Spring MVC-Testunterstützung. In diesem Fall dient sie dazu, MockMvc so zu konfigurieren, dass es HTTP-Anfragen simuliert. [73] ObjectMapper von Jackson (siehe Abschnitt 2.3.5) wird benötigt, um die Daten aus den JSON-Objekten zu extrahieren. @Autowired ist in Spring dazu da, Dependencies automatisch in eine Klasse einzufügen, ohne diese konfigurieren zu müssen.
Die erste Testmethode ist dazu da, den Endpunkt "getAllFunds()äus der FundController-Klasse zu testen. Dabei sollten alle Funds in einer Liste aus LowDetailFundDto-Objekten zurückgegeben werden. Zuerst schickt das MockMvc eine HTTP-Anfrage an den Endpunkt. Es wird erwartet, dass im JSON-Format geantwortet wird und das mit dem HTTP-Status OK. Danach wird die JSON-Antwort mittels des ObjectMappers in eine Liste von LowDetailFund-Objekten umgewandelt. Es wird überprüft, ob eine Antwort zurückgegeben wird (assertThat(lowDetailFunds).isNotNull()). Zusätzlich muss überprüft werden, ob diese Liste auch nicht leer ist (assertThat(lowDetailFunds.size()).isGreaterThan(0)).

Listing 31: test GetFunds

```
1  @Test
2      void testGetFunds() throws Exception {
3          MvcResult result = mockMvc.perform(get("/api/funds")
4                          .contentType(MediaType.APPLICATION_JSON))
5                  .andExpect(status().isOk())
6                  .andReturn();
7          String content = result.getResponse().getContentAsString();
8          List<LowDetailFundDto> lowDetailFunds = objectMapper.readValue(content, new
               TypeReference<List<LowDetailFundDto>>() {
9          });
10         assertThat(lowDetailFunds).isNotNull();
11         assertThat(lowDetailFunds.size()).isGreaterThan(0);
12     }
```

Die zweite Methode testet den "testGetFundByIsin(String isin)Ëndpunkt. Dieser Endpunkt erwartet eine ISIN als Pfadparameter als ID eines Funds, um ein einzelnes HighDetailFundDto-Objekt

zurückzugeben. Im Gegensatz zu LowDetailFundDto enthalten diese Objekte genauere Informationen, wie Werte, die für eine Analyse verwendet werden könnten. Hier simuliert MockMvc zwei unterschiedliche Anfragen. Die eine schickt wieder eine Anfrage an den getAllFunds-Endpunkt. Das ist nur notwendig, um eine ISIN-Nummer zurückzubekommen, die zu einhundert Prozent in der Datenbank gespeichert ist. Ein konstanter Wert könnte nicht immer vorhanden sein und in Zukunft möglicherweise für Verwirrung sorgen. Die zweite Anfrage geht nun an den getFundByIsin-Endpunkt. An den Pfad zur API wird die ISIN-Nummer des ersten Elements des vorherigen Ergebnisses angehängt. Der ObjectMapper wandelt es danach wieder in ein HighDetailFundDto-Objekt um. Danach befinden sich einige assert-Methoden (wie assertFalse(highDetailFund.getPortfolioBreakdown().get(0).getAllocationList().isEmpty())), die sicherstellen, ob tatsächlich alle zusätzlichen Listen zurückgegeben wurden.

Listing 32: Test GetFundsByIsin

```
1   @Test
2       void testGetFundByIsin() throws Exception {
3           MvcResult test = mockMvc.perform(get("/api/funds")
4                           .contentType(MediaType.APPLICATION_JSON))
5                   .andExpect(status().isOk())
6                   .andReturn();
7           String testContent = test.getResponse().getContentAsString();
8           List<LowDetailFundDto> lowDetailFunds = objectMapper.readValue(testContent, new
                TypeReference<List<LowDetailFundDto>>() {
9           });
10
11          MvcResult result = mockMvc.perform(get("/api/funds/" +
                lowDetailFunds.get(0).getIsin())
12                          .contentType(MediaType.APPLICATION_JSON))
13                  .andExpect(status().isOk())
14                  .andReturn();
15          String content = result.getResponse().getContentAsString();
16          HighDetailFundDto highDetailFund = objectMapper.readValue(content,
                HighDetailFundDto.class);
17      }
```

3.2 Datenbanken und Datenmodellierung

3.2.1 Intro

Aufgabe

Zu erfüllen war, ein Datenmodell für das Projekt zu entwickeln. Als Vorlage wurden wenige XML-Dateien gegeben, jede einzelne Datei stellt einen ETF dar.

Exchange Traded Funds

ETF steht für Exchange Traded Funds, diese sind vergleichbar mit einer Ansammlung von Aktien. Die Ansammlung repräsentiert einen Index, das können Regionen oder Branchen sein. ETFs und Aktien werden an der Börse gehandelt.

Technologien

Am Anfang wurden auch Vorgaben zu den zu benutzenden Technologien getroffen. Im Datenbankbereich ist neben den XML-Vorlagen noch zusätzlich vorgegeben worden, dass PostgreSQL verwendet werden muss. Das wurde von der Firma beschlossen, weil nur PostgreSQL-Entwickler eingestellt sind. Der Aufwand und die eingesetzten Ressourcen, einen neuen Mitarbeiter einzustellen oder jemanden weiterzubilden, wären es nicht wert.

Als Gruppe ist sich ohne Vorgaben der Firma darauf geeinigt worden, dass Datagrip für die Entwicklung der Skripte und Docker als Umgebung für die Ausführung der Datenbank verwendet wird. Datagrip war aufgrund von vergangenen Projekten bereits bekannt und auch ausreichend für unsere Zwecke. Bei Docker ging es in erster Linie darum, dass Gebrauch von der Containerisierung und der Möglichkeit, diese einfach zu teilen, gemacht wird.

3.2.2 Analyse und Entwicklung

Bevor mit der Entwicklung einer Datenbank begonnen werden kann, müssen die Daten, mit denen gearbeitet wird, analysiert werden. Die Vorlage waren wenige XML-Dateien und ein erstes Schema von der Firma. 14

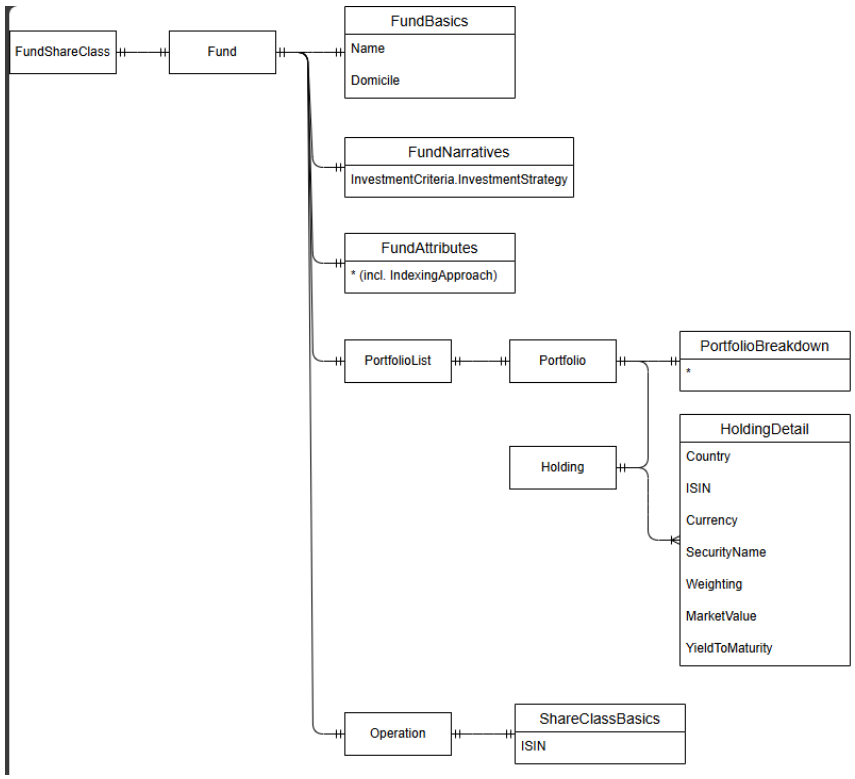


Abbildung 14: Erstes Schema

Abbildung 15: Oberflächliche Struktur

Aus 14 15 diesen beiden Bildern wurde die Erkenntnis getroffen, dass "FundShareClass"nur der Überbegriff ist, der alle großen Gruppen zusammenhält. Die Aufgabe, die aus dem ersten Schema abzulesen ist, dass von den großen Gruppen nur "Fund"benötigt wird. Dementsprechend müssen die anderen Begriffe aus der XML-Abbildung nicht beachtet werden.

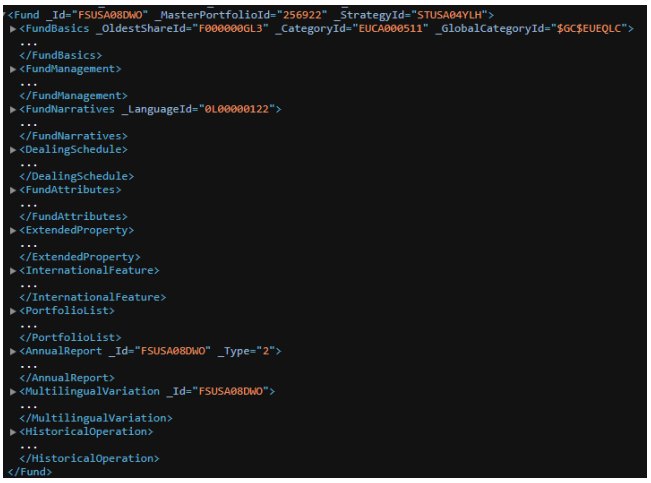


Abbildung 16: XMLFund

In der Abbildung 16 sieht man mehr Werte als im vorgegebenen Schema. Die zusätzlichen Werte führten zu Rücksprachen mit dem Betreuer und das Ergebnis war, dass nur die angegebenen Klassen berücksichtigt werden müssen.

Das nächste Thema in der Rücksprache war die "Portfoliolist", diese speichert grundsätzlich nur ein Portfolio, in dem Portfolio sind dann alle weiteren Daten zu finden. Daher wurde eine Ebene tiefer gegangen und nur Portfolio als Klasse gespeichert. 16

Abbildung 17: XMLPortfolio

Auf diesen Erkenntnissen wurde das erste Relationsschema gebaut:

- Fund (FundID)
  - FundBasics (FundBasicsID, Name, Domicile)
  - FundNarratives (FundNarrativeID, Strategy)
  - FundAttributes (FundAttributesID, EnhancedIndexFund, FundOfFunds, IndexFund, LifeCycleFund, SecurityLending, MasterFeeder, NonDiversifiedFund, TermTrusFund, MoneyMarketFund, UmbrellaStructure, HedgeFund, PrivateLabelFund, OverlayManaged, ManagedDistribution, LeveragedFund, CoveredCall, ContinuouslyOffered, OffshoreVehicle, InsuredMuniFund, CurrencyTracking, PhysicalFull, PhysicleSample, DerivativeBased, SyntheticReplication, StratifiedSampling, Structured, CensusReplication, NotApplicable)
- Portfolio (PortfolioID)
  - PortfolioBreakdown (PortfoliobreakdownID)
  - BreakdownValue (Type, Weight)

- Holding (HoldingID)
  - HoldingDetail (HoldingDetailID, Country, ISIN, Currency, SecurityName, Weighting, MarketValue, YieldtoMaturity)

Dieses Schema ähnelt der Vorgabe noch sehr stark.

Nach jedem Ergebnis, das erreicht wurde, ist dieses mit dem Betreuer besprochen worden. Dabei wurde der Entschluss gefasst, dass alle Daten, die in FundBasics, FundNarratives und FundAttributes sind, in der Oberklasse Fund gespeichert werden sollen, weil diese nur eine ID besitzt. Das zweite Relationsschema sah dann also so aus:

- Fund (FundID, Name, Domicile, Strategy, EnhancedIndexFund, FundOfFunds, IndexFund, LifeCycleFund, SecurityLending, MasterFeeder, NonDiversifiedFund, TermTrusFund, MoneyMarketFund, UmbrellaStructure, HedgeFund, PrivateLabelFund, OverlayManaged, ManagedDistribution, LeveragedFund, CoveredCall, ContinuouslyOffered, OffshoreVehicle, InsuredMuniFund, CurrencyTracking, PhysicalFull, PhysicleSample, DerivativeBased,
- Portfolio (PortfolioID)
  - PortfolioBreakdown (PortfoliobreakdownID)
  - BreakdownValue (BreakdownValueID, Weight)
- Holding (HoldingID)
  - HoldingDetail (HoldingDetailID, Country, ISIN, Currency, SecurityName, Weighting, MarketValue, YieldtoMaturity)

Dass die Überklasse Fund so überfüllt ist, führte zu großen Übersichtsproblemen im ERD. Gelöst wurde die Überfüllung, indem alle Werte der Unterklassen von Funds, in der Klasse FundAttributes zusammengefasst wurden. Weiters ist ein Problem mit Holdings aufgetaucht, da davon ausgegangen wurde, dass sie nicht auf der gleichen Ebene wie die Funds sind. Das wohl größte Problem waren die Allocations. Diese sind die Inhalte der Portfolio Breakdowns, das alleine ist kein Problem, jedoch speichern die Allocations weitere Allocations in sich und um redundante Daten zu vermeiden, musste sich eine Lösung überlegt werden.

Listing 33: Allocationverschachtelung

```
<PortfolioBreakdown _SalePosition="L">
    <AssetAllocation>
        <BreakdownValue Type="1">99.27383</BreakdownValue>
        <BreakdownValue Type="3">0.01674</BreakdownValue>
        <BreakdownValue Type="5">0.11592</BreakdownValue>
        <BreakdownValue Type="6">0</BreakdownValue>
        <BreakdownValue Type="7">0.59351</BreakdownValue>
        <BreakdownValue Type="8">0</BreakdownValue>
    </AssetAllocation>
    <IndiaAssetAllocation>
        <IndiaAssetTypeBreakdown Level="1">
            <BreakdownValue Type="11">99.32162</BreakdownValue>
            <BreakdownValue Type="12">0.01737</BreakdownValue>
            <BreakdownValue Type="14">0.10164</BreakdownValue>
            <BreakdownValue Type="16">0.07488</BreakdownValue>
            <BreakdownValue Type="17">0.41636</BreakdownValue>
            <BreakdownValue Type="21">0.06812</BreakdownValue>
        </IndiaAssetTypeBreakdown>
        <IndiaAssetTypeBreakdown Level="2">
            <BreakdownValue Type="1110">99.20571</BreakdownValue>
            <BreakdownValue Type="1120">0.11592</BreakdownValue>
            <BreakdownValue Type="1210">0.01737</BreakdownValue>
            <BreakdownValue Type="1410">0.02587</BreakdownValue>
            <BreakdownValue Type="1440">0.07577</BreakdownValue>
            <BreakdownValue Type="1610">0.07488</BreakdownValue>
            <BreakdownValue Type="1710">0.41636</BreakdownValue>
            <BreakdownValue Type="2110">0.06812</BreakdownValue>
        </IndiaAssetTypeBreakdown>
    <IndianCustomAssetAllocation Level="1">
        <BreakdownValue Type="2">99.27383</BreakdownValue>
        <BreakdownValue Type="3">0.01737</BreakdownValue>
        <BreakdownValue Type="5">0.59289</BreakdownValue>
        <BreakdownValue Type="6">0.11592</BreakdownValue>
    </IndianCustomAssetAllocation>
        <IndianCustomAssetAllocation Level="2">
            <BreakdownValue Type="21">97.91053</BreakdownValue>
            <BreakdownValue Type="22">1.06107</BreakdownValue>
            <BreakdownValue Type="23">0.30223</BreakdownValue>
            <BreakdownValue Type="33">0.01737</BreakdownValue>
            <BreakdownValue Type="50">0.59289</BreakdownValue>
            <BreakdownValue Type="60">0.11592</BreakdownValue>
        </IndianCustomAssetAllocation>
    </IndiaAssetAllocation>
</PortfolioBreakdown>
```

In diesem Code ist sichtbar, dass AssetAllocation auf Ebene 1 ist und keine weiteren Allocations speichert. IndiaAssetAllocation ist ebenfalls Ebene 1, speichert aber eine Ebene darunter IndiaAssetTypeBreakdown. Es könnte der Entschluss gefasst werden, dass mit "Levelëinfach unterschieden werden kann, wie die Allocations geordnet gehören. Jedoch wiederholen sich Namen und Level innerhalb der XML-Datei und teilweise sogar innerhalb einer Allocation. Hinzu kommt, dass in anderen Dateien die Level öfter nicht vorhanden sind.

Zur Klarstellung, es ist sich auf 'Allocations' als Überbegriff geeinigt worden, für alle Werte, die direkt im 'Portfoliobreakdown' sind. In den anderen XMLs ist die Struktur immer ähnlich, aber nicht alle Klassen unter "Portfoliobreakdown"haben einen Namen mit Ällocation". Als erstes wurde darüber diskutiert, ob nicht einfach die Werte gekürzt werden, also dass eine Schleife durchgegangen wird, bis eine Allocation keine weiteren Allocations speichert. Das wurde aber schnell abgelehnt, weil dadurch die Zusammenhänge und Auflistungen verloren beziehungsweise komplizierter gemacht werden würden. Die Lösung, auf die sich geeinigt worden ist, war die, dass

Allocation auf sich selbst verweist. Dabei speichert die Allocation der überliegenden Allocation, wenn es keine überliegende Allocation gibt, dann ist der Wert einfach NULL.

Die Levelverschiebung wäre möglich, indem Allocation einen komplexeren Primary Key bekommt. Zum Beispiel mit: PBID: PortfoliobreakdownID Name: dem Namen der derzeitigen Allocation Type/Level: der Name alleine wäre nicht unique, jedoch wenn es mehrere mit dem gleichen Namen gibt, speichern diese ein unique Attribut mit. Wenn es die Allocation nur einmal gibt, dann muss ein Standardwert verwendet werden, um diese Lücke des PK zu füllen.

Zur Gruppierung wäre noch der Fremdschlüssel der Top-Level-Allocation benötigt worden, aber dieser kann nicht im Primärschlüssel gespeichert werden, weil er nicht immer gegeben ist.

Des Weiteren ist aufgefallen, dass die Attribute von FundAttributes und Allocation sehr unterschiedlich sind. Da es aber sehr ineffizient ist, für jede Situation eigene Klassen anzulegen, werden alle relevanten Variablen gespeichert. Die meisten der Variablen können aber NULL sein.

Breakdownvalues enthält die Werte, die Allocations beschreiben.

Das hat zu diesem ERD geführt. 18

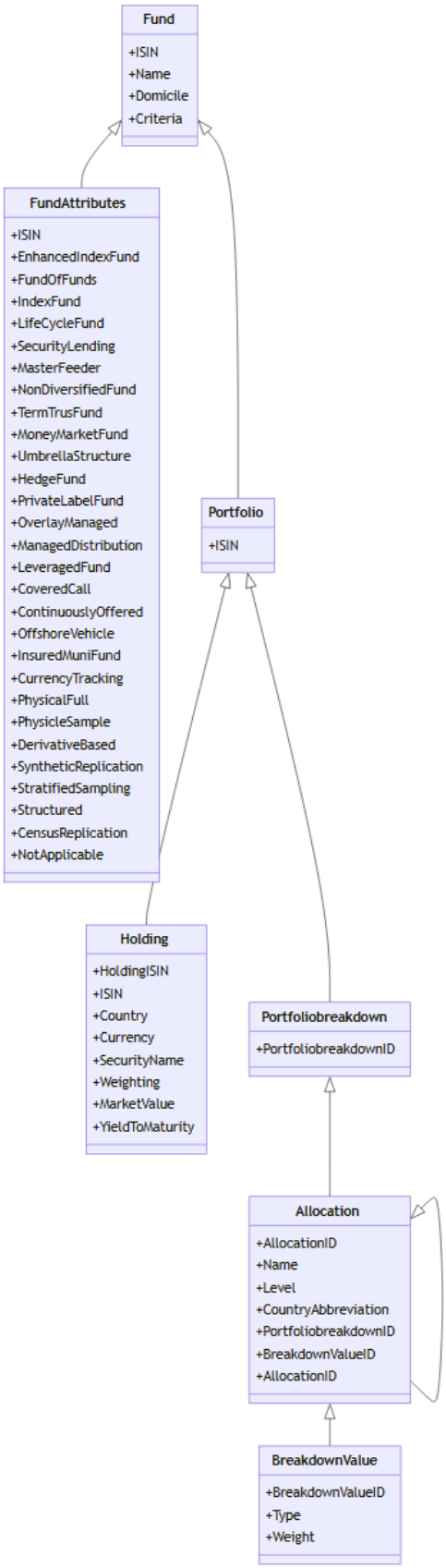


Abbildung 18: Idee 3 grafisch

Im nächsten Schritt sind Fund und Fundattributes aus Performance-Gründen wieder zusammengefügt worden. Die Performance wird dahingehend verbessert, dass nicht weiter auf eine Kindklasse zugegriffen wird und alle Werte direkt aus dem Fund genommen werden. Weiters ist aufgefallen, dass ein Fund immer nur ein Portfolio hat, daher sind Holding und Portfoliobreakdown direkt mit dem Fund verbunden.

Das ERD dazu sieht so aus:

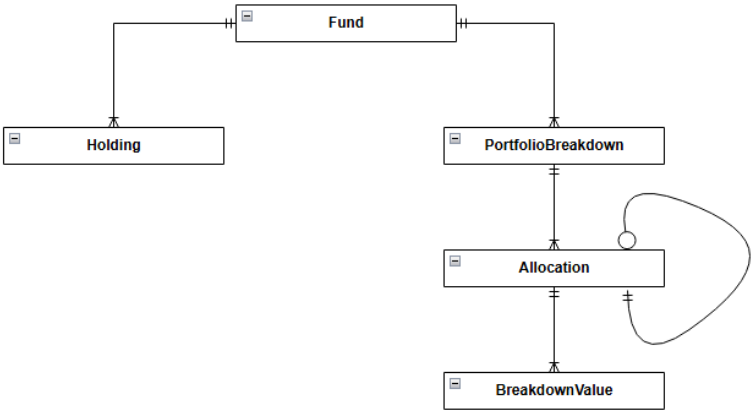


Abbildung 19: Idee 4 grafisch

Nach der Erstellung des ERD, wurde noch eine Änderung in Portfoliobreakdown vorgenommen. Bis jetzt wurde für jeden Portfoliobreakdown eine ID generiert, jedoch ist das nicht nötig, weil sie eindeutig über die Fund.Isin und sale.position identifizierbar sind. Und um Allocations eindeutig zu identifizieren, ist ebenfalls von einer extra generierten ID zu einem zusammengesetzten Primärschlüssel gewechselt worden. Der Schlüssel sieht so aus: Allocation (Name, Type, Portfoliobreakdown.PortfoliobreakdownID, Allocation.AllocationID)

Es gab einige Probleme in der Programmierabteilung bezüglich der Allocations, um genau zu sein mit dem rekursiven Aufruf auf sich selbst. Um das zu lösen, wurde der Verweis auf sich selbst gelöscht und eine weitere Klasse 'AllocationGroup' hinzugefügt.

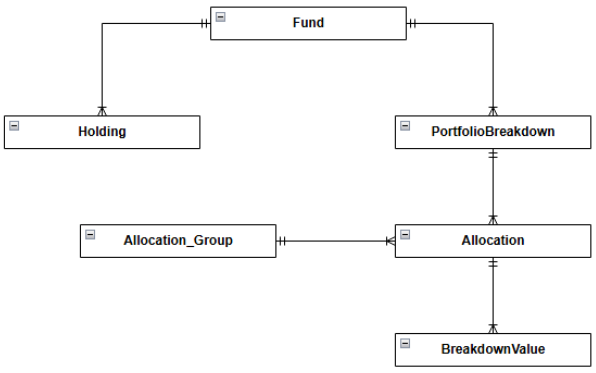


Abbildung 20: Idee 5 grafisch

In Allocationgroup werden nur die Daten zur Identifikation der Allocation gespeichert. AllocationGroup (allocation.groupid, Allocation.name, Allocation.type, Allocation.sale.position, Allocation.isin)

Nun erfüllt Allocation die dritte Normalform.

Das fertige Datenmodell inklusive aller Attribute sieht so aus:

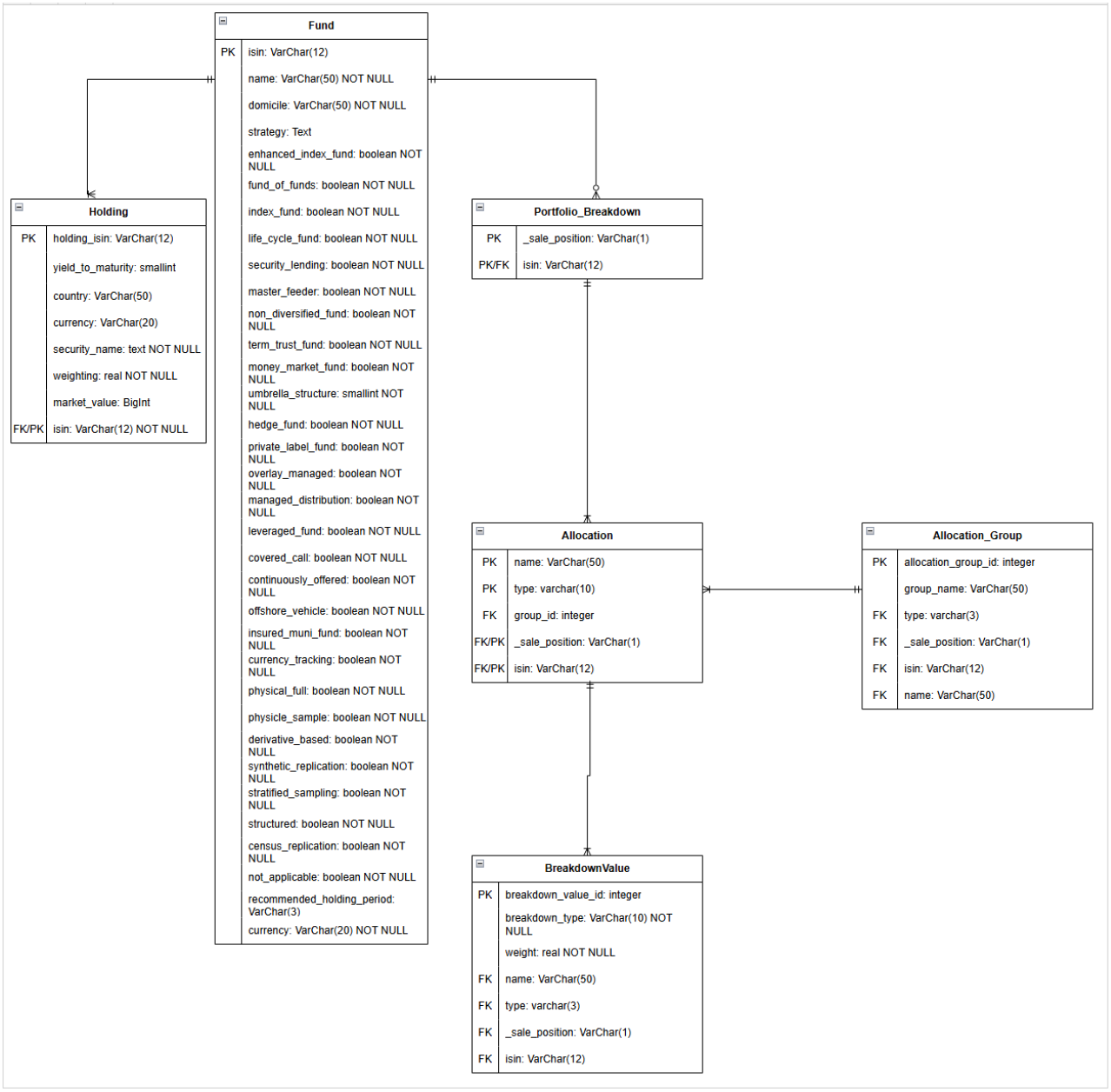


Abbildung 21: Fertiges Datenmodell

Die ISIN ist die ID des Funds und mit dieser kann alle Tabellen miteinander verknüpft werden.

3.2.3 Docker Implementierung

Die Erstellung des Containers ist sehr simpel, es werden einfach einige Kommandos ausgeführt und schon ist er erstellt.

Listing 34: Erstellung Docker DB

```
1          docker pull postgres #Installiert die PostgreSQL version in dem Fall latest
2          docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -d postgres
               #Erstellt die Postgre Instanz
```

Wenn alles erfolgreich durchgelaufen ist, sieht man das im Docker. 22

Abbildung 22: Docker-Image

3.2.4 SQL Skripte

Die Datenbank läuft auf Localhost:5432, jetzt müssen noch Tabellen erstellt und befüllt werden. Dafür muss sich in Datagrip mit der Datenbank verbunden werden. 23 Das grüne Häkchen zeigt, dass die Verbindung erfolgreich getestet wurde.

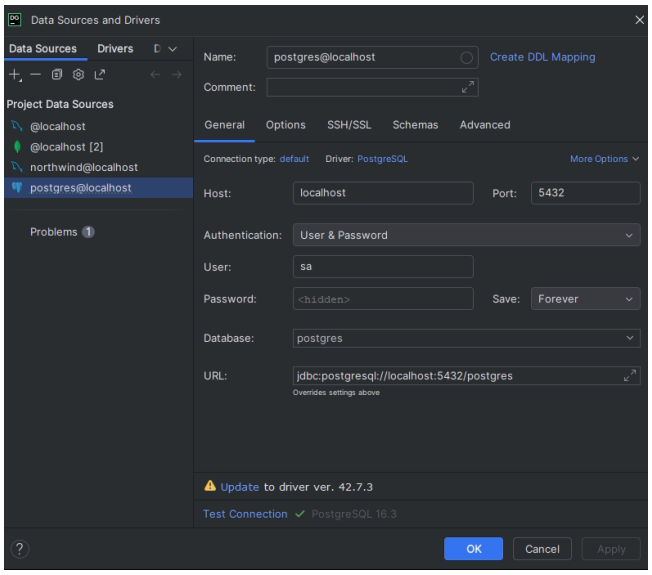


Abbildung 23: Verbindung-erfolgreich

Danach werden die Skripte erstellt und ausgeführt. Diese 4 Skripte wurden zum Testen verwendet:

- Create zur Erstellung der Tabellen.

Listing 35: Erstellung einer Tabelle

```
CREATE TABLE Portfoliobreakdown (
    sale_position VARCHAR(1),
    isin VARCHAR(12),
    PRIMARY KEY (sale_position, isin),
    FOREIGN KEY (isin) REFERENCES Fund (isin)
);
```

- Insert zum befüllen der Tabelle mit Test-Daten.

Listing 36: Befüllung einer Tabelle

```
INSERT INTO Portfoliobreakdown (sale_position, isin)
VALUES
('A', 'US1234567890'),
('B', 'US2234567890'),
('C', 'US3234567890');
```

- Select um die Daten auszugeben und um zu kontrollieren ob die Test-Daten korrekt eingefügt wurden.

Listing 37: Erfassung der Daten aus einer Tabelle

```
SELECT * FROM Allocation;
```

- Delete um die Tabellen wieder zu entfernen, falls das benötigt wird

Listing 38: Löschen einer Tabelle

```
DROP TABLE Portfoliobreakdown;
```

3.3 Evaluierung Frontend Framework

3.3.1 React Native

React Native ist ein Open-Source-Framework zur Entwicklung mobiler Anwendungen, das ursprünglich von Meta entwickelt wurde. Es ermöglicht Entwickler*innen, mit nur einer einzigen Codebasis gleichzeitig Apps für iOS und Android zu erstellen und das in JavaScript oder TypeScript, zwei der meistgenutzten Programmiersprachen der Webentwicklung. [74]

Im Gegensatz zu klassischen Hybrid-Apps, die lediglich in einem WebView innerhalb einer nativen Hülle ausgeführt werden, nutzt React Native echte native UI-Komponenten. Dadurch entsteht ein nativeres Look & Feel, das sich für Nutzerinnen kaum von einer vollständig nativ entwickelten App unterscheidet. Gleichzeitig profitieren Entwicklerinnen von einer besseren Performance, insbesondere bei der Darstellung von Benutzeroberflächen und Übergängen. [75][74] Das Framework basiert auf der bekannten JavaScript-Bibliothek React, was bedeutet, dass viele Webentwickler*innen mit Vorkenntnissen in React sehr leicht in React Native einsteigen können. Statt div oder span verwendet man hier Komponenten wie View, Text oder TouchableOpacity, die für die mobile Darstellung optimiert sind. [76]

In React Native gibt es die Möglichkeit, plattformübergreifend zu entwickeln. Das bedeutet: Ein Großteil des Codes funktioniert sowohl unter Android als auch unter iOS, was die Entwicklung erheblich beschleunigt und vereinfacht. Trotzdem bleibt es flexibel genug, um bei Bedarf plattformabhängige Komponenten zu schreiben, z.B. wenn bestimmte Funktionen auf einem Betriebssystem anders umgesetzt werden müssen. [74]

Zusätzlich unterstützt React Native moderne Entwicklungsfunktionen wie Hot Reloading bzw. Fast Refresh, wodurch Änderungen am Code direkt in der App sichtbar werden – ohne dass die App neu gestartet werden muss. Das spart enorm viel Zeit beim Entwickeln. [74] [77]

React Native eignet sich vor allem für Projekte, bei denen keine extrem rechenintensiven Funktionen notwendig sind, wie zum Beispiel Spiele oder Echtzeit-3D-Apps. Für klassische Business-Apps, E-Commerce, Social Media oder Buchungssysteme ist es jedoch eine äußerst beliebte und zuverlässige Wahl. Die App-Entwicklung mit React Native profitiert außerdem von einer riesigen Community, zahlreichen Open-Source-Bibliotheken und ständiger Weiterentwicklung durch Meta und viele andere Contributor weltweit. [77] [74] [78] [76]

Vorteile

In React Native ist es möglich hybride Apps zu erstellen, deren Aussehen und Benutzererlebnis dem einer nativen App ähnelt. Zur Darstellung werden native Komponente in React Native zur Darstellung der Benutzeroberfläche verwendet, wodurch eine native Optik und Haptik erzielt wird, während gleichzeitig eine plattformübergreifende Portabilität gewährleistet ist. Im Gegensatz zu herkömmlichen Web-Apps, die im Browser laufen und oft in ihrer Funktionalität eingeschränkt sind, können mit React Native entwickelte Anwendungen direkt auf native APIs zugreifen. Dies ermöglicht eine tiefere Integration mit Hardware-Funktionen wie GPS oder Kamera und führt zu einer Performance, die nahe an der von nativen Apps liegt. [74, 79]

Zusammenfassend bietet React Native die Möglichkeit, plattformübergreifende Anwendungen zu entwickeln, die das Aussehen und die Benutzererfahrung nativer Apps bieten, ohne für jede Plattform separat entwickeln zu müssen. [80]

Einer der Hauptvorteile von React Native ist, dass es eine gemeinsame Codebasis für mehrere Plattformen (iOS, Android und teilweise Web) bietet. React Native erreicht dies, indem es JavaScript mit React verwendet und dabei native Komponenten nutzt. Falls nötig, können spezifische Anpassungen für jede Plattform vorgenommen werden. [81].

Ein weiterer Vorteil von React Native ist die umfangreiche Verfügbarkeit von Plugins und Bibliotheken, die die Entwicklung plattformübergreifender Anwendungen erleichtern. Diese Plugins ermöglichen den Zugriff auf native Gerätefunktionen wie Kamera, GPS und Push-Benachrichtigungen, ohne dass umfangreiche plattformspezifische Programmierung erforderlich ist. Beispiele hierfür sind react-native-maps für die Integration von Karten und react-native-firebase für die Nutzung von Firebase-Diensten. [81]

Expo, ein Framework und eine Plattform für universelle React-Anwendungen, erweitert diese Funktionalität, indem es eine Reihe von Tools und Diensten bereitstellt, die die Entwicklung und den Aufbau von React Native-Projekten vereinfachen. Ein herausragendes Merkmal von Expo ist die Unterstützung von Config Plugins. Diese ermöglichen es Entwicklern, native Module und Funktionen zu ihrem Projekt hinzuzufügen, ohne direkt in den nativen Code eingreifen zu müssen. Config Plugins modifizieren die Expo-Konfiguration während des Pre-Build-Prozesses und automatisieren so die Integration nativer Module. [82, 83]

Die Verwendung von Expo und seinen Config Plugins bietet mehrere Vorteile: Automatisierte native Konfiguration: Durch den Einsatz von Config Plugins können native Module automatisch konfiguriert werden, was den Entwicklungsprozess beschleunigt und Fehler reduziert. Aktive Community und regelmäßige Updates: Sowohl React Native als auch Expo verfügen über eine

aktive Entwicklergemeinschaft, die kontinuierlich neue Plugins und Updates bereitstellt, um mit den neuesten Technologien und Best Practices Schritt zu halten. Zusammenfassend ermöglichen React Native und Expo durch ihre umfangreiche Plugin-Unterstützung und die Möglichkeit zur plattformübergreifenden Code-Nutzung eine effiziente und konsistente Entwicklung von Anwendungen für verschiedene Plattformen. [82, 83]

Nachteile

Ein häufiger Nachteil ist, dass Änderungen am Quellcode oft zu einem vollständigen Neustart der Applikation führen. Zwar bietet React Native Funktionen wie „Fast Refresh“, dennoch kann es insbesondere bei größeren Projekten zu Verzögerungen kommen, was den Entwicklungsprozess ausbremst. [84]

Ein weiterer Nachteil betrifft die Performance. Bei sehr komplexen oder speicherintensiven Anwendungen kann React Native nicht mit der Effizienz nativer Apps mithalten. Dies liegt unter anderem daran, dass React Native eine Brücke verwendet, um zwischen JavaScript und nativen Komponenten zu kommunizieren, was zu Engpässen führen kann. [85]

Im direkten Vergleich mit Alternativen wie Flutter, das eine eigene Rendering-Engine verwendet, zeigt sich, dass React Native bei der Ausführungsgeschwindigkeit und bei Animationen oft etwas hinterherhinkt. [86]

Zudem sind für viele native Funktionen spezielle Plugins oder eigene native Module notwendig, was zusätzliche Entwicklungskompetenz in Java (Android) oder Swift/Obj-C (iOS) erfordert. Dies kann den Entwicklungsaufwand erhöhen und widerspricht ein Stück weit dem Versprechen einer rein plattformübergreifenden Lösung. [87]

3.3.2 Angular

Das Angular Framework, entwickelt und gepflegt von Google, ist ein weit verbreitetes Open-Source-Framework zur Entwicklung von Single Page Applications (SPAs). Es basiert auf TypeScript und bietet eine vollständige, strukturierte Lösung für die Erstellung moderner Webanwendungen mit Komponentenarchitektur, Dependency Injection, Routing und integrierten Werkzeugen zur Testautomatisierung. Angular eignet sich besonders für große, skalierbare Anwendungen im Enterprise-Umfeld. [88, 89]

Ein wesentlicher Vorteil von Angular ist die Möglichkeit, Anwendungen als sogenannte Progressive Web Apps (PWAs) zu entwickeln. PWAs kombinieren die Vorteile klassischer Webanwendungen

mit jenen nativer Apps – sie sind im Browser ausführbar, aber gleichzeitig offlinefähig, installierbar und können Push-Benachrichtigungen versenden. Angular bietet mit dem offiziellen Paket @angular/pwa eine einfache Möglichkeit, bestehende Anwendungen um PWA-Funktionalitäten zu erweitern. Dieses Paket integriert u.a. automatisch ein konfigurierbares Service Worker Setup, ein manifest.webmanifest sowie Caching-Strategien, die durch Angulars Build-Prozess optimiert werden. [90, 91]

Dank dieser Integration ermöglicht Angular die Entwicklung robuster, performanter Webanwendungen, die sich wie native Apps verhalten und auch unter instabilen Netzwerkbedingungen zuverlässig funktionieren. Die Kombination aus einem modernen Framework und PWA-Technologien fördert sowohl die User Experience (UX) als auch die Erreichbarkeit von Anwendungen über verschiedenste Geräte hinweg. [90, 91]

Vorteile

Angular bietet eine ganze Reihe an Vorteilen, die es besonders für größere, skalierbare Webanwendungen attraktiv machen. Einer der größten Pluspunkte ist die große Auswahl an vorgefertigten Komponenten und Modulen, die direkt im Angular-Ökosystem verfügbar sind. Diese Komponenten sparen Entwicklungszeit und sorgen für eine konsistente Benutzeroberfläche. [92]

Ein weiteres starkes Argument ist die Unterstützung für Progressive Web Apps (PWAs). Angular bietet eine einfache Möglichkeit, Webanwendungen mit Offline-Fähigkeiten, Push-Benachrichtigungen und Installationsfunktionen auszustatten und das mit nur wenigen zusätzlichen Schritten im Entwicklungsprozess. [90]

Zudem punktet Angular durch ein einfaches und strukturiertes Testen. Dank des eingebauten Test-Frameworks und der klaren Architektur können sowohl Unit-Tests als auch End-to-End-Tests effizient umgesetzt werden, was die Wartbarkeit und Qualität der Anwendung verbessert. [93]

In puncto Kompatibilität ist Angular sehr stark: Es unterstützt alle modernen Browser und sorgt mit Polyfills auch für Rückwärtskompatibilität zu älteren Versionen. Diese breite Browserunterstützung macht Angular zu einer stabilen Basis für Webanwendungen im professionellen Umfeld. [94]

Ein weiterer wichtiger Punkt ist die Zukunftssicherheit: Angular wird von Google entwickelt und aktiv weiter gepflegt, was regelmäßige Updates, moderne Features und langfristigen Support garantiert. Für viele Unternehmen ist das ein zentraler Entscheidungsfaktor. [88]

Nicht zuletzt bietet Angular eine klare und einheitliche Struktur. Durch das MVC-ähnliche Architekturmodell, Dependency Injection, Module und Services ist die Codebasis auch in größeren Teams gut wartbar und skalierbar. Das Framework fördert bewährte Best Practices und sorgt so für sauberen, wartbaren Code. [95]

Nachteile

Trotz seiner weitreichenden Möglichkeiten bringt Angular auch einige Nachteile mit sich, die bei der Entscheidung für ein Framework bedacht werden sollten. Einer davon ist die Tatsache, dass Angular keinen direkten Zugriff auf native Funktionen von Mobilgeräten bietet. Anders als Frameworks wie Flutter, die speziell für mobile Anwendungen entwickelt wurden und nativen Code kompilieren können, ist Angular rein auf Webtechnologien ausgelegt. Das bedeutet, dass Funktionen wie Kamera, GPS oder Bluetooth nicht ohne externe Hilfsmittel oder hybride Ansätze genutzt werden können. Für viele Webprojekte ist das jedoch unproblematisch – insbesondere dann, wenn keine plattformspezifischen Features benötigt werden, wie es etwa bei klassischen Unternehmens-Dashboards oder Verwaltungsanwendungen der Fall ist. [96]

Ein weiterer Nachteil besteht in der vergleichsweise kleineren und weniger aktiven Community. Im Gegensatz zu React oder Vue.js, die eine riesige Entwicklerbasis und unzählige frei verfügbare Pakete, Tutorials und StackOverflow-Beiträge bieten, fällt der Community-Support bei Angular spürbar kleiner aus. Das bedeutet: Wenn bei der Entwicklung Probleme auftreten, findet man unter Umständen nicht sofort eine passende Lösung oder ein Beispiel aus der Praxis. Auch Drittanbieter-Bibliotheken und UI-Komponenten sind nicht ganz so zahlreich wie in anderen Ökosystemen. Zwar bietet Angular selbst viele Werkzeuge mit, aber gerade bei sehr individuellen Anforderungen kann der begrenzte Umfang an Community-Erweiterungen spürbar sein. [97]

Besonders ins Gewicht fällt auch die vergleichsweise hohe Einstiegshürde. Während moderne Frameworks wie React oder Vue.js häufig mit einem minimalen Setup starten und eine flache Lernkurve haben, verlangt Angular bereits zu Beginn ein solides Grundverständnis seiner Architektur und Arbeitsweise. Entwickler*innen müssen sich mit verschiedenen Kernkonzepten auseinandersetzen, wie beispielsweise Dependency Injection, RxJS und Observables, Angular Module, Services, Decorators, und Templates mit eigener Syntax. Diese Komplexität hat zwar den Vorteil, dass große Projekte klar strukturiert und skalierbar bleiben – für Einsteiger oder kleinere Teams kann sie jedoch überfordernd wirken und den Projektstart verzögern. Besonders für jene, die aus dem klassischen Webumfeld kommen oder vorher mit React gearbeitet haben, ist die Umgewöhnung spürbar. [98]

3.3.3 Ionic

Ionic ist ein Open-Source-Framework zur Entwicklung von Cross-Plattform-Apps, das es Entwickler*innen ermöglicht, mobile, Web- und Desktop-Anwendungen mit einer einzigen Codebasis zu erstellen. Ursprünglich im Jahr 2013 veröffentlicht, basiert Ionic auf Webtechnologien wie HTML, CSS und JavaScript und nutzt moderne Frontend-Frameworks wie Angular, React oder Vue als Grundlage. Ionic setzt dabei auf Web Components und verwendet das von Capacitor unterstützte Hybrid-Konzept, um native Funktionalitäten auf mobilen Geräten zugänglich zu machen. [99, 100]

Ein zentrales Ziel von Ionic ist die nahtlose Benutzererfahrung, die sich in nativen UI-Komponenten widerspiegelt, welche plattformübergreifend identisch oder plattformspezifisch dargestellt werden können. Durch die Integration von Capacitor, einer nativen Laufzeitumgebung von Ionic, können Apps direkt auf native APIs wie Kamera, GPS, Benachrichtigungen oder Datei-Handling zugreifen, vergleichbar mit klassischen nativen Apps, aber bei wesentlich reduziertem Entwicklungsaufwand. [101]

Besonders hervorzuheben ist, dass Ionic auch eine hervorragende Unterstützung für Progressive Web Apps (PWAs) bietet. Mit seinem Fokus auf responsive, performante und offlinefähige Webtechnologien eignet sich Ionic besonders gut für den Aufbau moderner PWAs, die sich installierbar verhalten und wie native Apps auf mobilen Geräten nutzbar sind. [102]

Durch die Kombination von plattformübergreifender Flexibilität, nativer Performance durch Capacitor und der Einbindung beliebter Frameworks bietet Ionic eine praxisnahe und produktive Umgebung für die moderne App-Entwicklung. [101] [100]

Vorteile

Ionic ist ein beliebtes Framework für die plattformübergreifende App-Entwicklung und bietet eine Reihe von Vorteilen, die es vor allem für Webentwickler*innen attraktiv machen. Ein zentraler Pluspunkt ist, dass es sich bei Ionic um eine Webapplikation handelt, die wie eine native App aussieht und sich auch so anfühlt. Das sogenannte "native Look & Feel"wird durch den Einsatz von Webtechnologien wie HTML, CSS und JavaScript in Verbindung mit einer nativen Hülle (über Capacitor oder Cordova) ermöglicht. So können Anwendungen erstellt werden, die optisch und funktional kaum von echten nativen Apps zu unterscheiden sind. [99]

Ein weiterer wesentlicher Vorteil von Ionic ist, dass nur eine einzige Codebasis für alle Plattformen benötigt wird, egal ob Android, iOS, Web oder Desktop. Diese plattformübergreifende Entwicklung spart Zeit und Ressourcen, da Änderungen und neue Features nicht mehrfach

implementiert werden müssen. Auch die Wartung der App wird dadurch erheblich vereinfacht. [103]

Ionic punktet außerdem durch eine große Auswahl an Plugins, mit denen sich native Funktionen wie Kamera, GPS, Push-Benachrichtigungen oder Dateispeicher unkompliziert integrieren lassen. Dank Capacitor, dem modernen nativen Laufzeit-Tool von Ionic, ist die Plugin-Nutzung stabiler und moderner als bei früheren Versionen mit Cordova. Auch Drittanbieter-Bibliotheken lassen sich einfach einbinden, was die Entwicklung zusätzlich beschleunigt. [104]

Ein praktisches Feature ist zudem, dass Ionic-Apps direkt im Browser getestet werden können. Entwickler*innen können damit schnell eine Vorschau ihrer App sehen, ohne sie ständig auf ein echtes Gerät oder einen Emulator übertragen zu müssen. Gerade in frühen Entwicklungsphasen beschleunigt dies den Workflow enorm. [105]

Nachteile

Trotz seiner Flexibilität und Benutzerfreundlichkeit bringt Ionic auch einige Nachteile mit sich, die bei der Auswahl eines passenden Frameworks berücksichtigt werden sollten. Ein wesentlicher Punkt ist, dass bei Codeänderungen die Anwendung vollständig neu geladen werden muss, da es, im Gegensatz zu Flutter, kein echtes Hot Reload gibt. Das kann den Entwicklungsprozess verlangsamen, da Entwickler*innen nach jeder Änderung mehr Zeit auf das erneute Starten und Testen der App verwenden müssen. [106]

Ein weiterer Nachteil liegt in der Performance, insbesondere bei komplexeren oder speicherintensiven Anwendungen. Da Ionic auf Webtechnologien wie HTML, CSS und JavaScript basiert und in einer WebView läuft, kann es in Situationen, in denen viele Berechnungen oder rechenintensive Prozesse anfallen, zu spürbaren Leistungsengpässen kommen. Im Vergleich zu Frameworks wie Flutter oder React Native, die näher am nativen Code arbeiten, fällt Ionic bei der Ausführungsgeschwindigkeit in vielen Fällen etwas ab. [107] [108]

Auch wenn Ionic viele native Funktionen über Plugins verfügbar macht, sind Plugins oft zwingend notwendig, um auf bestimmte Systemfunktionen zuzugreifen, etwa auf Kamera, Bluetooth, GPS oder Dateisysteme. Diese Abhängigkeit von Plugins kann zu Problemen führen, insbesondere wenn Plugins veraltet, nicht mehr gepflegt oder inkompatibel mit bestimmten Geräten sind. Zwar bietet Capacitor eine moderne Lösung zur Plugin-Verwaltung, dennoch bleibt Ionic im Bereich „echter“ nativer Funktionen weniger flexibel als speziell dafür entwickelte Frameworks. [109]

3.3.4 Flutter

Flutter ist ein von Google entwickeltes Open-Source-Framework zur plattformübergreifenden Entwicklung moderner Benutzeroberflächen. Es ermöglicht die Erstellung von mobilen, Web- und Desktop-Anwendungen auf Basis einer einzigen Codebasis. Die erste stabile Version wurde 2018 veröffentlicht und hat sich seither als ernstzunehmende Alternative zu klassischen nativen und hybriden Frameworks etabliert. [110]

Technologisch unterscheidet sich Flutter deutlich von anderen Cross-Plattform-Ansätzen wie React Native oder Ionic. Während viele Frameworks auf native UI-Komponenten der jeweiligen Plattform zurückgreifen, verwendet Flutter seine eigene Rendering-Engine namens Skia, mit der Benutzeroberflächen vollständig unabhängig von der darunterliegenden Plattform dargestellt werden können. Dadurch ergibt sich eine hohe Konsistenz in der Darstellung sowie eine bemerkenswerte Performance – sowohl auf iOS als auch auf Android und anderen Plattformen. [111]

Flutter nutzt die Programmiersprache Dart, die ebenfalls von Google entwickelt wurde. Dart kombiniert objektorientierte Strukturen mit einer modernen Syntax und ermöglicht sowohl Just-in-Time als auch Ahead-of-Time Compilation. Dies ermöglicht unter anderem das beliebte Feature Hot Reload, bei dem Änderungen im Quellcode sofort im laufenden Zustand der App sichtbar werden. [112]

Flutter hat eine ausgeprägte Widget-basierte Architektur. Nahezu jedes UI-Element ist als Widget implementiert, was eine deklarative und komponentenorientierte Entwicklung fördert. Darüber hinaus unterstützt Flutter sowohl Material Design als auch Cupertino-Design, wodurch sich Apps visuell an die Designrichtlinien von Android bzw. iOS anpassen lassen. [113]

Zunehmend wird Flutter auch für Progressive Web Apps (PWAs) eingesetzt. Die Webunterstützung befindet sich zwar noch im Entwicklungsstatus, bietet aber bereits die Möglichkeit, plattformunabhängige Anwendungen im Browser auszuführen, inklusive Offlinefähigkeit und App-Installation. [114]

Insgesamt stellt Flutter eine leistungsstarke und moderne Entwicklungsumgebung dar, die sich durch hohe Performance, ein flexibles UI-System und einen plattformübergreifenden Ansatz auszeichnet. Vor allem für Teams, die gleichzeitig mehrere Plattformen bedienen möchten, bietet Flutter einen enormen Effizienzgewinn. [114]

Vorteile

Flutter bietet eine Vielzahl von Vorteilen, die es zu einer beliebten Wahl für die plattformübergreifende App-Entwicklung machen. Einer der größten Pluspunkte ist das sogenannte Hot Reload: Diese Funktion ermöglicht es Entwickler*innen, Änderungen am Code in Echtzeit zu sehen, ohne die gesamte App neu starten zu müssen. Nur die neu hinzugefügten oder geänderten Teile werden neu geladen, was den Entwicklungsprozess deutlich beschleunigt. [115] Ein weiterer entscheidender Vorteil ist die Plattformunabhängigkeit. Mit Flutter kann derselbe Code sowohl für Android, iOS, Web als auch für Desktop-Anwendungen verwendet werden. Das spart nicht nur Entwicklungszeit, sondern reduziert auch den Wartungsaufwand, da kein doppelter Code für verschiedene Plattformen notwendig ist. [116]

Im Vergleich zu anderen Frameworks wie Ionic zeigt Flutter vor allem bei komplexeren Anwendungen eine höhere Performance. Dies liegt unter anderem daran, dass Flutter direkt auf nativen Code kompiliert und nicht auf WebView-Komponenten angewiesen ist. [117]

Zudem macht Flutter es sehr einfach, aus einer mobilen App eine Webanwendung oder Progressive Web App zu erstellen. Dank der einheitlichen Codebasis und der guten Unterstützung für Webtechnologien ist die Umwandlung schnell und effizient möglich. [114] Ein praktisches Feature ist auch die automatische Generierung von Farbschemata. Flutter bietet eine integrierte Funktion, mit der basierend auf einer Primärfarbe ein komplettes Farbschema erzeugt werden kann. [118]

Nachteile

Trotz vieler Vorteile bringt Flutter auch einige Nachteile mit sich, die bei der Wahl des Frameworks berücksichtigt werden sollten. Ein häufig genannter Kritikpunkt ist, dass Flutter nicht über so viele etablierte Bibliotheken und Tools verfügt wie beispielsweise Ionic. Gerade für bestimmte Spezialfälle oder weniger verbreitete Funktionen müssen Entwickler*innen daher öfter auf eigene Lösungen zurückgreifen oder auf weniger gepflegte Drittanbieterpakete ausweichen. [119]

Ein weiteres Problem betrifft die Darstellung komplexer Animationen und Vektorgrafiken über Plugins. Hier kommt es immer wieder zu Einschränkungen, insbesondere im Web-Bereich, da die Unterstützung für solche Plugins teils noch experimentell oder nicht vollständig ausgereift ist. [120]

Zudem ist Flutter Web derzeit noch nicht ideal für Inhalte wie Blogs oder andere stark SEO-abhängige Seiten. Da Flutter keine klassische HTML-Struktur verwendet und Inhalte oft nicht

direkt im DOM stehen, kann es von Suchmaschinen schwerer indexiert werden. Für Projekte mit starkem Fokus auf Suchmaschinenoptimierung ist Flutter Web deshalb aktuell nicht die erste Wahl. [121]

Ein technisches Hindernis ist auch, dass Flutter Web nicht auf Standardtechnologien wie JavaScript, TypeScript oder HTML basiert, sondern auf Dart. Dadurch ist die Integration in bestehende Web-Projekte schwieriger und es fehlt die gewohnte Flexibilität von klassischen Webframeworks. [122]

Schließlich basiert Dart auf einem nominalen Typsystem, was bedeutet, dass Typen nur dann kompatibel sind, wenn sie explizit so definiert wurden. Das unterscheidet sich etwa von strukturellen Typsystemen wie in TypeScript und kann in manchen Fällen zu Einschränkungen in der Wiederverwendbarkeit und Flexibilität von Code führen. [123]

3.3.5 Prototypen

Im Rahmen der Diplomarbeit wurden Prototypen eingesetzt, um zentrale Funktionen, Benutzeroberflächen und Abläufe frühzeitig zu testen und iterativ zu verbessern. Der Einsatz von Prototypen ist eine bewährte Methode in der Softwareentwicklung, insbesondere im agilen Kontext, da sie es ermöglichen, Ideen schnell sichtbar und erlebbar zu machen, bevor eine vollständige Implementierung erfolgt.

Ein Prototyp dient dabei als Vormodell oder funktionsreduzierte Vorversion der späteren Anwendung. Durch die frühzeitige Visualisierung von Abläufen, Layouts oder Interaktionen kann frühzeitig Feedback eingeholt, Missverständnisse mit Stakeholdern vermieden und die Benutzerfreundlichkeit überprüft werden. Dadurch lassen sich potenzielle Probleme oder konzeptionelle Schwächen frühzeitig erkennen und beheben – was Zeit und Ressourcen in der späteren Entwicklungsphase spart.

Prototyping unterstützt zudem die kommunikative Schnittstelle zwischen Entwicklung und Design. Komplexe Anforderungen lassen sich durch klickbare Mockups, einfache Codebeispiele oder visuelle Abläufe klarer vermitteln. Außerdem hilft ein Prototyp dabei, die technische Umsetzbarkeit bestimmter Funktionen zu evaluieren, etwa im Hinblick auf Performance, Kompatibilität oder Machbarkeit innerhalb des gewählten Frameworks.

Es wurden in allen Frameworks mindestestens ein Prototyp angefertigt, der zeigt, wie die Daten vom Backend eingelesen werden können. Insgesamt hat die Erstellung von Prototypen wesentlich dazu beigetragen, den Entwicklungsprozess strukturiert, nutzerzentriert und effizient zu gestalten. Der finale Prototyp wird in Absatz ETF-APP Anwendung 3.4 genauer beschrieben.

3.3.6 Entscheidung für React Native

Für den Prototyp wurde sich für React Native entschieden, da es für die Anforderungen der Anwendung die sinnvollste und effizienteste Lösung bietet. Die App benötigt keine tiefgreifenden nativen Gerätefunktionen wie Kamera, Sensoren oder GPS, sondern konzentriert sich im Wesentlichen auf die Anbindung an eine REST-API sowie die Verwaltung von dynamischen Daten, um die Kennzahlen der ETFs darzustellen. [**?**]

React Native bietet hierfür eine ideale Grundlage: Da es auf TypeScript basiert und mit React ein sehr etabliertes Frontend-Framework nutzt, lässt sich die Anwendung schnell und effizient entwickeln. Die plattformübergreifende Struktur bedeutet, dass ich nur eine Codebasis für Android und iOS brauche, was den Entwicklungs- und Wartungsaufwand deutlich reduziert, ohne dabei auf eine native Optik verzichten zu müssen. [79]

Darüber hinaus unterstützt React Native ein schnelles Entwicklungs- und Testverfahren durch Live Reload und Fast Refresh, was den Workflow deutlich verbessert, besonders bei häufiger UI-Anpassung. In Kombination mit einer gut strukturierten Backend-API in Spring Boot lassen sich so moderne, performante Apps erstellen, ohne sich mit unnötiger Komplexität auf der nativen Ebene beschäftigen zu müssen. [80]

3.4 ETF-APP Anwendung

3.4.1 Aufbau des Programms

```
etf-app-frontend
    app
        _layout.tsx
        fund.tsx
        index.tsx
    filter
        filter.tsx
        searchFilter.tsx
    model
        allocation.tsx
        breakdownValue.tsx
        chartType.tsx
        fund.tsx
        holding.tsx
        portfolioBreakdown.tsx
        prc.tsx
        spotPrice.tsx
    styles
        colorScheme.tsx
        fundDetail.tsx
        fundStyle.tsx
        layoutStyle.tsx
    app.json
    babel.config.js
    env.d.tsx
    package.json
    tsconfig.json
```

Die Ordnerstruktur des Projekts orientiert sich an den Konventionen von Expo Router, einer modernen Routing-Lösung für React Native basierend auf dem Dateisystem. Im Wurzelverzeichnis befinden sich neben Konfigurationsdateien wie package.json, tsconfig.json und babel.config.js auch projektweite Definitionen in env.d.tsx sowie app.json, welches zentrale App-Einstellungen wie Name, Icon oder Deep Linking beinhaltet. [124]

Das zentrale Element für das Routing bildet der Ordner app/. Er enthält verschiedene „Bildschirmdateien“, die als Routen fungieren. Beispielsweise wird index.tsx als Startseite (/) interpretiert, fund.tsx als /fund und _layout.tsx als gemeinsames Layout für alle darunterliegenden Routen verwendet. Die Verwendung von _layout.tsx ist eine Besonderheit des Expo Routers, da damit übergeordnete Layout-Komponenten wie Navigationsleisten oder Wrapper definiert werden können, die sich mehrere Seiten teilen. [125, 126]

Zusätzlich zur Routenstruktur gibt es klar voneinander getrennte Module für spezifische Funktionalitäten. Der Ordner filter/ enthält Komponenten zur Filterung und Suche von Fonds. filter.tsx und searchFilter.tsx liefern dabei UI-Komponenten oder Logik zur Eingrenzung der dargestellten Fonds. Im Ordner model/ sind die zentralen Typdefinitionen und Datenmodelle abgelegt, die zur Beschreibung von Fonds, Allokationen, Marktpreisen oder Portfoliobestandteilen verwendet werden. Diese Aufteilung folgt dem Prinzip der Trennung von Präsentation und Datenmodell. [127]

Unter styles/ befinden sich gestylte Komponenten oder Style-Definitionen. Beispielsweise legt colorScheme.tsx zentrale Farbwerte fest, während fundDetail.tsx, fundStyle.tsx und layoutStyle.tsx für spezifische visuelle Anpassungen einzelner Ansichten zuständig sind. Diese Modularisierung erlaubt eine bessere Wiederverwendbarkeit und Wartbarkeit. [128]

Diese Struktur unterstützt die Skalierbarkeit der Anwendung, indem sie sowohl die funktionale Trennung (z.B. in filter, model, styles) als auch die routenbasierte Navigation mittels Expo Router berücksichtigt. Durch die Kombination von typensicherem TypeScript, modularer Komponentenstruktur und dem Dateibasierten Routing wird eine klare, wartbare Architektur geschaffen, die sich gut für größere React Native Projekte eignet. [128]

3.4.2 Axios

Für das Laden der Daten vom Backend ist die Entscheidung auf Axios gefallen. Axios ist ein Third-Party-Package. Mit folgendem Befehl wird es installiert:

```
npm install axios
```

Für die Datenkommunikation mit dem Backend wurde im Rahmen dieses Projekts das HTTP-Client-Modul Axios eingesetzt. Obwohl React Native bereits über die native Fetch API verfügt, bietet Axios mehrere Vorteile, die letztlich ausschlaggebend für die Entscheidung waren. Einer der zentralen Vorteile besteht darin, dass Axios die Antwort eines HTTP-Requests, als JSON-Datenstring,automatisch in ein TypeScript-Objekt umwandelt. Im Gegensatz dazu muss bei der Fetch API dieser Schritt explizit über Methoden wie response.json() durchgeführt werden. [129][130][131][132]

In der Implementierung dieser Anwendung wird ausschließlich die GET-Methode von Axios verwendet, um Daten vom Server abzurufen. Dabei erfolgt der Aufruf asynchron mithilfe der modernen TypeScript-Schlüsselwörter async und await. Diese Syntax abstrahiert den Umgang mit sogenannten Promises, die eine zentrale Rolle in der asynchronen Programmierung in TypeScript spielen. Ein Promise stellt ein Objekt dar, das ein zukünftiges Ergebnis eines noch nicht abgeschlossenen Vorgangs beschreibt – etwa den Abschluss eines HTTP-Requests. [129][130][131][132]

Durch die Verwendung von async/await wird der asynchrone Code lesbarer und ähnelt syntaktisch einem synchronen Ablauf. Das Schlüsselwort await pausiert die Ausführung der Funktion, bis das Promise aufgelöst ist, d.h. bis eine Antwort empfangen oder ein Fehler zurückgegeben wurde. Dies reduziert die Komplexität, die durch klassische Callback- oder .then()-Ketten entsteht, und erhöht gleichzeitig die Wartbarkeit des Codes. [129][130][131][132]

Im folgenden Beispiel wird gezeigt, wie Axios implementiert wird:

Listing 39: Axios Implementierung

```
1       async function fetchData() {
2           return await axios.get<Fund[]>(API_URL);
3       }
```

3.4.3 Cors

Wenn ein React Native Frontend mit einem Java Spring Boot Backend kommunizieren soll – etwa beim Abrufen von Daten oder beim Senden von Formularen, dann spielt CORS (Cross-Origin Resource Sharing) eine entscheidende Rolle. Besonders in der Entwicklungsumgebung befinden sich Frontend und Backend oft auf unterschiedlichen Ursprüngen, also Domains oder Ports. Ein typisches Beispiel: Das Backend läuft auf http://localhost:8080 (Spring Boot), während das React Native Projekt über den lokalen Entwicklungsserver z.B. auf http://localhost:8081 ausgeführt wird. [133, 134] Standardmäßig blockieren Webbrowser und Mobil-Emulatoren solche Cross-Origin-Anfragen aus Sicherheitsgründen, um zu verhindern, dass unerlaubte Webseiten

auf sensible Daten zugreifen. Damit der Browser bzw. das React Native Runtime-Environment die Verbindung zum Backend zulässt, muss der Server explizit angeben, welche Ursprünge (z.B. localhost:8081) Zugriff erhalten dürfen. Diese Konfiguration erfolgt im Spring Boot Backend über sogenannte CORS-Regeln. [81] Eine gängige Möglichkeit, CORS global zu konfigurieren, besteht darin, eine eigene WebMvcConfigurer-Klasse im Backend anzulegen. In dieser Klasse wird definiert, für welche Endpunkte (z.B. /api/**) und welche Ursprünge Anfragen erlaubt sind. Auch die erlaubten HTTP-Methoden wie GET, POST oder DELETE sowie Header können hier angegeben werden. Diese Konfiguration stellt sicher, dass React Native ohne Fehlermeldungen wie CORS policy: No 'Access-Control-Allow-Origin' header mit dem Backend interagieren kann. [135]

3.4.4 Recharts

Reachrts ist ein Third-Party-Plugin, das für das Darstellen von Graphen genutzt werden kann. Recharts dient zum Anzeigen von Area Charts, Bar Charts, Line Charts, Pie Charts und viele weitere Graphen. Bevor Recharts verwendet wird, muss es noch importiert werden. Das unteren Codebeispiel zeigt, wie eine Line Chart Komponente aufgebaut wird. Die Daten die dargestellt werden, sollten als Key/Value Paare gespeichert werden. Die Daten müssen als Parameter an Line Chart übergeben werden. Nebenbei können Höhe und Breite der Line Chart eingestellt werden. XAxis ist die X-Achse des Graphen, hier ist als dataKey angegeben, welche Daten an der X-Achse angezeigt werden. Bei der Y-Achse als YAxis kann der Wertebereich der Y-Achse als den Parameter domain konfiguriert werden. Im Beispiel 40 werden dataMin und dataMax angegeben, das heißt, dass der Bereich der Werte an der Y-Achse angezeigt wird, von Minimum bis Maximum von value ist. Tooltip ermöglicht dem Benutzer, das Anzeigen der Werte, wenn er mit dem Cursor über den Punkt, wo der Cursor steht, fährt. CartestainGrid sorgt dafür, dass ein Gitter angezeigt wird, im Beispiel ist es mithilfe des Parameters stroke grau gefärbt. Mit Line wird die Linie des Graphen dargestellt, hier wird als dataKey value übergeben. Nebenbei wird hier noch der Linientyp linear, das sorgt dafür, dass die Linie von Punkt zu Punkt linear dargestellt wird. Wenn monotone als Typ übergeben ist, wird die Linie abgeflacht. Wenn dot auf false gesetzt ist, werden keine Punkte an jedem Datenpunkt angezeigt. Brush dient dazu, dass der Benutzer einen Wertebereich mit einen Schieber auswählen kann. Hier wird als dataKey name angegeben damit sich der Wertebereich auf der X-Achse ändert. Im Beispiel wurde noch die Höhe und Breite des Schiebers eingestellt. [136] Das folgende Beispiel zeigt, wie eine Line Chart in ein Programm implementiert werden kann:

Listing 40: Line Chart in React Native

```
import { CartesianGrid, Line, LineChart, XAxis, YAxis, Tooltip, Brush, } from
    'recharts';

<LineChart width={800} height={400} data={data} >
    <XAxis dataKey="name" />
    <YAxis domain={['dataMin', 'dataMax']} />
    <Tooltip /> { }
    <CartesianGrid stroke="#eee" />
    <Line type="linear" dataKey="value" stroke={COLOR_SCHEME[0]} dot={false} />
    <Brush
        dataKey="name"
        height={30}
        travellerWidth={15}
    />
 </LineChart>
```

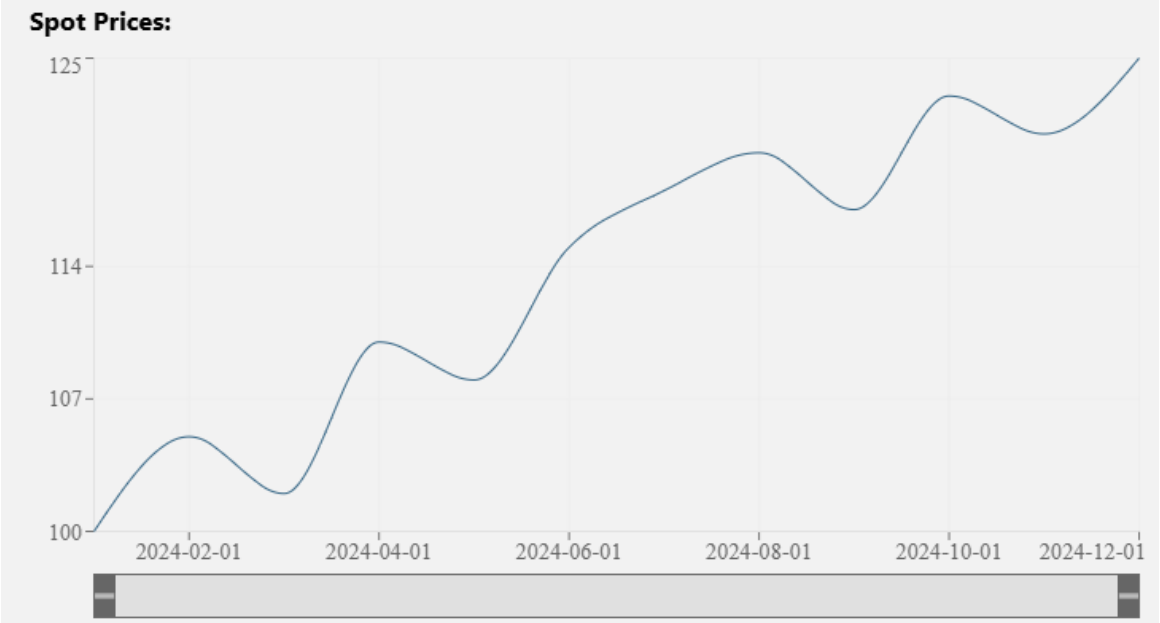


Abbildung 24: Line Chart

Neben der Line Chart wird auch eine Pie Chart verwendet. In dem unteren Beispiel 41 zeigt, wie so eine Pie Chart aussehen kann. Die Parameter cx und cy sind die X- und Y-Koordinaten von dem Mittelpunkt des Kreises. OuterRatios bestimmt die Größe des Kreises und dataKey sorgt dafür, dass value als Daten für die Zellen genommen wird. Mit Label wird der Name zu den Werten für die Zellen angezeigt. In der PieChart Komponente gibt es eine Pie Komponente. Die Pie hat einen Parameter für die Daten, der Datentyp ist der gleiche, wie bei der Line Chart. Bei dem Beispiel wird durch das ganze Datenarray iteriert. So werden mit Cell die einzelnen Zellen des Graphen angezeigt. Tooltip zeigt die Details zu einer Zelle an, wenn der Benutzer mit dem Cursor drüber fährt. Damit es eine Legende gibt, muss die Komponente Legend in hinzugeschrieben werden. [136]

Listing 41: Pie Chart in React Native

```
import { CartesianGrid, Line, LineChart, XAxis, YAxis, Tooltip, Brush, } from
    'recharts';

<PieChart width={400} height={450}>
    <Pie data={data} cx={200} cy={200} outerRadius={150} dataKey="value" label>
```

```
5               {data.map((entry, index) => (
6                   <Cell key={`cell-${index}`} fill={COLOR_SCHEME[index %
                        COLOR_SCHEME.length]} />
7               ))}
8           </Pie>
9           <Tooltip />
10          <Legend />
11      </PieChart>
```

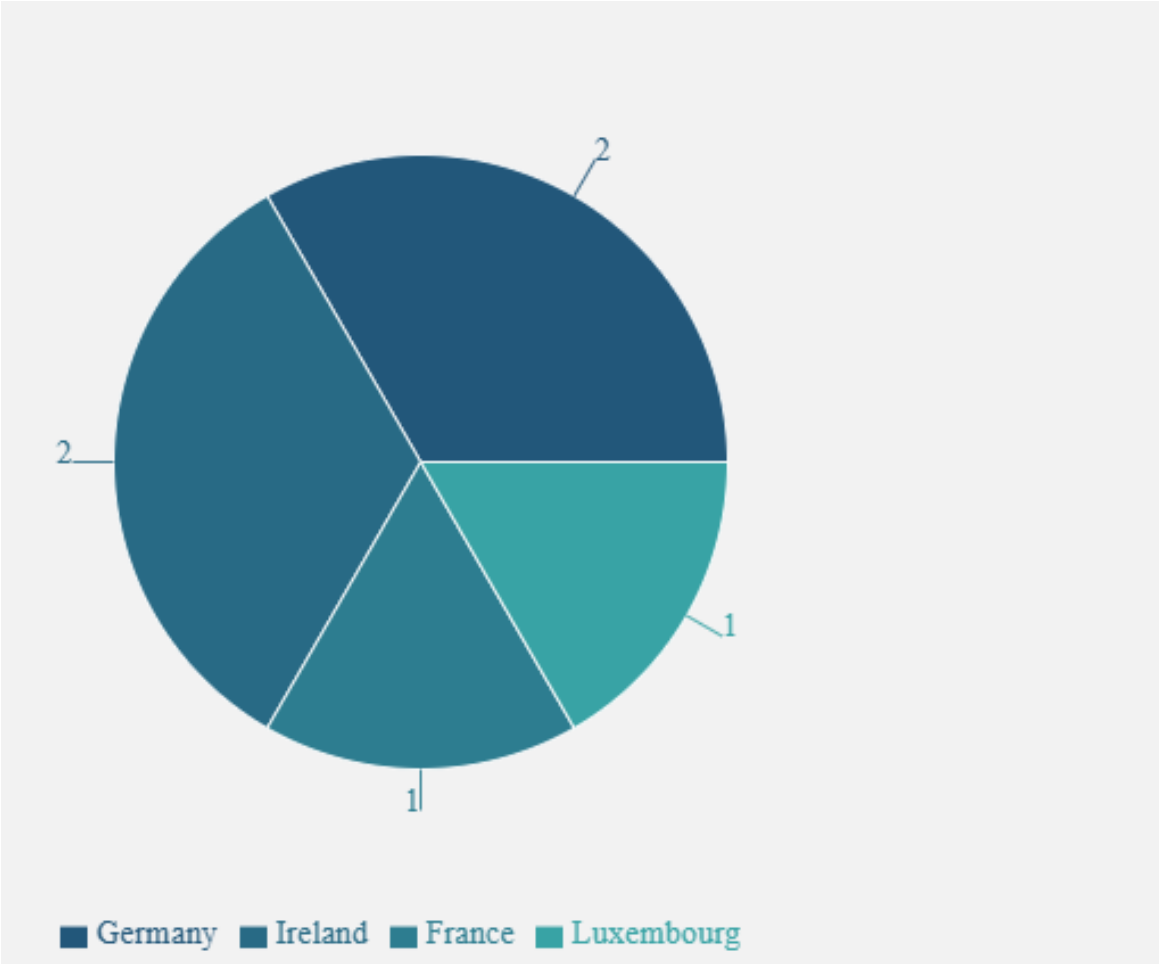


Abbildung 25: Pie Chart

3.4.5 Filter

Die Funktion countryFilter prüft, ob ein bestimmter ETF in Bezug auf eine Land gefiltert werden soll oder nicht. Dabei gibt sie true zurück, wenn der ETF angezeigt werden soll, und false, wenn nicht.

Zuerst wird eine Variable noCountrySet auf true gesetzt. Sie dient dazu festzustellen, ob überhaupt eines der Länder in der übergebenen Liste als aktiv gefiltert markiert ist.Dann geht die Funktion alle Länder in der countries-Menge durch. Wenn eines dieser Länder als gefiltert (isFiltered === true) markiert ist, wird noCountrySet auf false gesetzt, also es gibt mindestens ein aktives Filterkriterium.

Anschließend prüft die Funktion, ob das Land des Fonds, das speichert, die Variable fund.domicile, auch das übergebene Land enthält. Falls ja, wird sofort true zurückgegeben – der Fonds wird also behalten, weil er zu einem der aktiv gefilterten Länder gehört. Wenn kein Land als Filter ausgewählt wurde, wird true zurückgegeben – der Fonds wird also nicht rausgefiltert, weil keine Einschränkung gesetzt wurde. In allen anderen Fällen, also wenn Länder gefiltert werden, aber dieser Fonds zu keinem dieser Länder gehört, gibt die Funktion false zurück, der Fonds wird nicht angezeigt.

Listing 42: Länderfilter

```
import { FilteredCountries } from "../model/filteredCountries";
import { Fund } from "../model/fund";

export function countryFilter(fund: Fund, countries: Set<FilteredCountries>): boolean {

    let noCountrySet: boolean = true;

    for (const country of countries) {
        if (country.isFiltered === true) {
            noCountrySet = false;
              if (fund.domicile?.includes(country.country)) {
                return true;
            }
        }
    }

    if (noCountrySet) {
        return true;
    }
    return false;
}
```

Die Funktion überprüft, ob ein bestimmter Investmentfonds zu einem eingegebenen Suchbegriff passt. Dabei wird der Suchbegriff zunächst in Kleinbuchstaben umgewandelt, um sicherzustellen, dass die Groß- oder Kleinschreibung keine Rolle spielt. Anschließend wird geprüft, ob dieser Suchbegriff im Namen des Fonds vorkommt. Falls ja, gilt der Fonds als passend und wird berücksichtigt. Wenn der Suchbegriff nicht im Namen zu finden ist, wird überprüft, ob er in der ISIN, also der eindeutigen Identifikationsnummer des Fonds, vorkommt. Sollte auch dort keine Übereinstimmung gefunden werden, wird zuletzt noch geschaut, ob das Herkunftsland des Fonds den Suchbegriff enthält. Wenn in einem dieser drei Felder der Suchbegriff auftaucht, wird der Fonds als Treffer zurückgegeben. Andernfalls wird er ausgeschlossen. Das folgende Beispiel macht genau so etwas:

Listing 43: Suchfilter

```
import { Fund } from "../model/fund";

export function searchFilter(fund: Fund, pattern: string): boolean {
    const lowerCasePattern: string = pattern.toLowerCase();
    if (fund.name?.toLowerCase().includes(lowerCasePattern)) {
        return true;
    }
    if (fund.isin.toLowerCase().includes(lowerCasePattern)) {
        return true;
    }
    if (fund.domicile?.toLowerCase().includes(lowerCasePattern)) {
        return true;
```

```
13      }
14      return false;
15  }
```

3.5 XML Import

Ein großer Teil des Projektes besteht daraus, relevante Daten aus großen XML-Files (siehe Abschnitt 2.2.7), welche jeweils die Daten zu einem ETF enthalten, auszulesen und in die Datenbank zu speichern. Für diesen Teil ist der Dataimporter verantwortlich. Es soll möglich sein, alle XML-Files, die dem Format entsprechen, in den ”hot directory”-Ordner zu legen und die Daten auszulesen.

3.5.1 Technologien

Jackson

Bei der Evaluierung der passenden Technologie zum Lesen der XML-Files wurde sich für Jackson (siehe Abschnitt 2.3.5) entschieden, da es die komplexen XML-Files mithilfe der XmlMapper-Klasse in eine Baumstruktur aus JSON-Nodes umwandeln kann. Da einige Teile der XML-Files rekursiv abgearbeitet werden müssen, ist diese Struktur von Vorteil. Weiters bietet es die oben genannte Möglichkeit, die Daten aus der XML-Datei direkt in ein Objekt einer passenden Klasse zu mappen, solange sich die Attribute unter denselben JsonNode befinden. Das kann besonders bei großen Klassen, wie der Fund-Klasse, entscheidend sein. Auch Apache Xerces (siehe Abschnitt 2.3.6) wäre eine Möglichkeit gewesen. In diesem Fall wäre die Idee gewesen, die Daten zu DOM zu parsen, um darauf zuzugreifen. Für das Parsen zu DOM ist Apache Xerces auch geeignet. Das Problem ist allerdings, dass bei DOM die Daten nicht automatisch eingesetzt werden, sondern dass jedes Attribut einzeln ausgelesen und in ein Objekt eingesetzt werden müsste. Im folgenden Beispiel wird dargestellt, wie auf DOM zugegriffen werden müsste(das Document ist doc):

Listing 44: XML DOM Reading

```
1  doc.getDocumentElement().normalize();
2  Book book = new Book();
3  Element element = doc.getDocumentElement();
4  NodeList nodeList = element.getChildNodes();
5  for (int i = 0; i < nodeList.getLength(); i++) {
6      Node node = nodeList.item(i);
7      if (node.getNodeType() == Node.ELEMENT_NODE) {
8          Element childElement = (Element) node;
9          switch (childElement.getTagName()) {
10             case "title":
11                 book.setTitle (childElement.getTextContent());
12                 break;
13             case "author":
14                 book.setAuthor(childElement.getTextContent());
```

```
15                      break;
16                  }
17              }
18          }
```

Besonders bei großen Klassen (wie Fund) kann dieser Code lang und zeitaufwendig werden. Allerdings kann das Jackson mit nur einer Zeile Code erledigen.

Alternativen

Die Struktur der XML-Files wurde beim Auslesen der Allocations zum Problem. Denn diese Allocations sind unbestimmt tief und speichern selbst unterschiedlich viele weitere Allocations. Es wurde schnell klar, dass die einzige Möglichkeit, diese Allocations auszulesen, eine rekursive Abarbeitung ist. Zuallererst wurde nach anderen XML-Deserialisierungs-Technologien gesucht, welche eine Methode ähnlich der readValue(JsonNode node)-Methode von Jackson (siehe Abschnitt 2.3.5) mit rekursivem Durchlauf der Subnodes bieten. Das Problem mit der von Jackson ist, dass sie nur eine Ebene gleichzeitig analysieren kann. Da es keine Technologie gibt, die über diese Methode verfügt, gab es keine Alternativen und diese Methode musste selbst implementiert werden. Die Baumstruktur der JSON-Nodes ist ein guter Ausgangspunkt, da dadurch jede Allocation geordnet und in richtiger Reihenfolge einzeln verarbeitet werden kann.

Mapstruct

Da es jede Klasse mehrmals gibt, aber diese in das selbe Format gebracht werden sollen, um in die Datenbank gespeichert werden zu können, müssen diese gemappt werden. Da die Attribute alle ziemlich gleich heißen bei den Businessobjects und den DAO-Objekten, ist MapStruct besonders nützlich. Bei Klassen wie Fund hätte sich der Mappingcode über viele Zeilen gezogen. Das ist besonders fehleranfällig, da möglicherweise falsche Attribute gemappt werden oder einige Attribute vergessen werden. Hier ist ein Beispiel, wie Mapstruct in diesem Projekt verwendet wird:

Hier werden Objekte der XmlFund-Klasse zu FundDAO-Objekten gemappt:

Listing 45: Mapping with MapStruct

```
1  @Mapper
2  public interface DAOMapper {
3      DAOMapper INSTANCE = Mappers.getMapper(DAOMapper.class);
4
5      @Mapping(target = "isin", source = "source.ISIN")
6      FundDAO fundToFundDAO(XmlFund source);
7  }
```

Im Dataimporter-Abschnitt gibt es einen DAO-Mapper. Er verfügt für jedes Businessobjekt, mit zugehöriger DAO-Klasse, mindestens eine Mappingfunktion.

3.5.2 DAO-Pattern

Grundsätzlich sollte der Aufbau des XML-Teils des Projekts einem DAO-Muster (siehe Abschnitt 2.2.8) entsprechen. Leider wurden aber die Klassen etwas falsch benannt und erfüllen nicht exakt den Anforderungen des DAO-Musters. Folgende Funktionen erfüllen die Klassen in diesem Teil:

BusinessObject : Die BuisnessObjects tragen in diesem Projekt etwas andere Namen. Beispielsweise heißt die Fund-Klasse, die der Dataimporter verwendet XmlFund.

DataAccessObject (DAO) : Die Datenbankoperationen werden im jeweiligen Repository durchgeführt. In diesen Klassen werden Funktionen des CrudRepository implementiert. In Falle des Dataimporters enthalten die Repository-Klassen nur eine Insert-Methode, da andere nicht benötigt werden.

DataSource : Es gibt in diesem Abschnitt keine Klasse, die die Datenverbindung verwaltet.

DataTransferObject (DTO) : In unserem Fall erfüllen die DAO-Klassen, wie AllocationDAO, eigentlich die Arbeit der DTO-Klassen.

Abgesehen vom DAO-Muster gibt es kaum andere Objekte, da die Funktion des Backends lediglich die Speicherung der Daten aus verschiedenen Quellen in einer Datenbank ist. Die Business Objects bzw. die DTO-Objekte könnten als POJOs bezeichnet werden.

Business Objects

Die Business Objects haben kaum Funktionen. Sie sind lediglich zum Speichern der Daten zuständig. Alle Klassen sehen vom Aufbau her gleich aus. Diese Klassen haben Annotationen von Lombok (siehe Abschnitt 2.3.8). Um genau zu sein, @Data, um alle Getter, Setter, hashCode, equals, toString und einen Required-Arguments-Constructor zu generieren. Dazu noch einen All-Arguments-Constructor und einen No-Arguments-Constructor. Die Parameter ”staticConstructor=of” und ”staticName=of” sorgen dafür, dass die Konstruktoren private sind und eine statische Fabrikmethode erzeugt wird. Sie halten die Klasse übersichtlich, da sie die Getter, Setter und Konstruktoren selbst erzeugen. Hier wurde als Beispiel die XmlAllocation-Klasse verwendet:

Listing 46: example Business Object

```
@Data(staticConstructor = "of")
@AllArgsConstructor(staticName = "of")
@NoArgsConstructor
public class XmlAllocation {
    private String name;
    // for Level or type (example: CountryExposure)
    private String type;
    private List<XmlBreakdownValue> breakdownValues;
    private XmlAllocation allocation;
}
```

Da die Allocations im XML-File sehr verschachtelt sind und diese rekursiv verarbeitet werden müssen, können sie andere Allocations speichern.

Data Access Objects

Hier werden die Datenbankoperationen durchgeführt. Diese Klassen implementieren das CrudRepository-Interface. Dabei handelt es sich um einen Teil des Spring Data JPA-Frameworks. Dieses stellt nützliche Methoden für CRUD-Operationen auf relationale Datenbanken bereit. In den Generics muss angegeben werden, welche Entität sie handhaben müssen (hier AllocationDAO) und den Datentyp der ID dieser Entität(hier String). Einige dieser nützlichen Methoden sind:

1. save (speichert eine gegebene Entität)
2. findById (gibt die Entität mit der angegebenen ID zurück)
3. findAll (gibt alle Instanzen dieses Typs zurück)
4. count (gibt die Anzahl der Entitäten zurück)
5. deleteById (löscht die Entität mit dieser ID)

[137]

In diesem Fall wird aber die insert-Methode verwendet. Diese sorgt dafür, dass ein Eintrag in der Datenbank erstellt wird. Die ”@Modifying”-Annotation wird benötigt, wenn eine create-/update-/oder delete-Operation auf die Datenbank ausgeführt wird. In @Query wird die SQL-Query geschrieben. Jedes Wort vor dem ”:” steht, ist ein Parameter, der beim Methodenaufruf angegeben werden muss. Beispielsweise wird hier ”:name” in der Query benötigt. Diese Variable wird gefüllt durch den Parameter mit @Param(”name”) davor.

Listing 47: example Data Access Object

```
public interface AllocationRepository extends CrudRepository<AllocationDAO, String> {
    @Modifying
    @Query("INSERT INTO Allocation (name, type, group_id, sale_position, isin) VALUES
        (:name, :type, :group_id, :sale_position, :isin)")
    void insert(@Param("name") String name, @Param("type") String type, @Param("group_id")
        Integer group_id, @Param("sale_position") String sale_position, @Param("isin")
        String isin);
}
```

Data Transfer Objects

Die Data Transfer Objects (DTOs) sind die einzigen Dateien, die sich beide Teile des Backends teilen. Wie oben bereits beschrieben, wurde diese Klasse falsch benannt. Auch hier wurde wieder Lombok (siehe Abschnitt 2.3.8) verwendet, um Boilerplate-Code zu reduzieren. Im Grunde handelt es sich bei diesen Objekten um das Format, wie es sich die Datenbank erwartet. Die @Table(”allocation”)-Annotation wird verwendet, um diese Klasse mit der Tabelle ”allocation” in der Datenbank zu verknüpfen. Dementsprechend müssen auch die Attribute mit den Spalten der Tabelle übereinstimmen. @Id wird verwendet, um die eindeutige ID eines Eintrags dieser Tabelle zu kennzeichnen. Hier ein Beispiel für ein Data Transfer Object unseres Projekts anhand der AllocationDAO-Klasse:

Listing 48: example Data Transfer Object

```
@Data(staticConstructor = "of")
@AllArgsConstructor(staticName = "of")
@Table("allocation")
@NoArgsConstructor
public class AllocationDAO {
    // PK
    @Id
    private String name;
    // PK
    private String type;
    // FK for Group
    private int groupId;
    // PK, FK for Portfoliobreakdown
    private String salePosition;
    // PK, FK for Portfoliobreakdown
    private String isin;
}
```

3.5.3 Controller

Der Controller ist sozusagen der Ausgangspunkt dieses Teils der Anwendung. Hier wird lediglich der DataimportService initialisiert und danach wird in der init()-Methode die importData()-Methode aufgerufen, die dafür sorgt, dass alle XML-Dateien ausgelesen werden und in die Datenbank gespeichert werden. Diese Methode ist mit @PostConstruct gekennzeichnet, was bedeutet, dass sie gleich nach der Dependency Injection ausgeführt wird.

Listing 49: Controller Class

```
    @PostConstruct
    public void init() {
        dataimportService.importData();
    }
```

3.5.4 Dataimport Service

Bei diesem Service handelt es sich um das Herzstück des XML-Import-Teils. Hier werden alle XML-Files ausgelesen und in die richtigen Objekte gespeichert. Gestartet wird der Vorgang in der importData()-Methode. Hier werden alle Files im Hotdirectory-Ordner überprüft, und wenn es sich dabei um ein XML-File handelt, wird die Funktion readXml-aufgerufen. Diese gibt das bereits ausgelesene XMLFund-Objekt zurück, und dieses wird an die writeToDatabase-Methode gegeben, um es in die Datenbank zu speichern.

Listing 50: find XML-Files

```
1   public void importData() {
2           // search for every XML-File in the hotDirectory and read them and write them into
                the DB
3           File directory = new File(HOT_DIRECTORY_URL);
4           List<File> xmlFiles = List.of(directory.listFiles((dir, name) ->
                name.toLowerCase().endsWith(FILETYPE)));
5           xmlFiles.forEach(xmlFile -> {
6               XmlFund fund = readXml(xmlFile);
7               writeToDatabase(fund);
8           });
9           System.out.println("Loading successful: " + xmlFiles.size() + " files");
10      }
```

Der Wert der Konstante FILETYPE ist ”.xml”

XML lesen

Zuallererst wird die ”readXml”-Funktion aufgerufen. Dort wird ein ”XmlMapper”-Objekt und ein ”ObjectMapper”-Objekt von Jackson (siehe Abschnitt 2.3.5) erstellt. Diese Funktion ist dafür verantwortlich, eine gute Basis mit vielen generellen Informationen zu extrahieren und den Prozess des XML-Lesens zu steuern. Dafür wird zuallererst ein neues XmlFundShareClass-Objekt angelegt. Darin werden lediglich 2 JsonNodes gespeichert. Dieses Objekt ist nur dazu da, um die Wurzel leichter zu nennen, um nicht jedes Mal readValue() aufrufen zu müssen, sondern stattdessen root (der Name der Variable) aufrufen zu können. Die meisten Werte, die für Fund aus dem XML-File ausgelesen werden müssen, befinden sich unter dem ”FUND_ATTRIBUTES_NODE”. Da die treeToValue-Methode von ObjectMapper nur Werte unter demselben JsonNode zuordnen kann, wurde dieser Node zum Mappen verwendet, um die größtmögliche Anzahl an Attributen auf einmal in XmlFund speichern zu können. Allerdings gibt es einige Werte, die im ganzen XML-File auf unterschiedlichen Ebenen verteilt liegen (beispielsweise ISIN, welche die ID des Fonds darstellt). Deshalb gibt es die extractGeneralData-Methode, die diese Daten aus dem riesigen XML-Baum ”cherrypicken” muss. Um die genauen Aktien dieses Fonds auszulesen, wird die extractPortfolio-Funktion aufgerufen, und für die genauen Portfoliowerte ist die getAllPortfolioBreakdowns-Funktion zuständig.

Listing 51: reading XML

```
public XmlFund readXml(File file) {
    try {
        root = xmlMapper.readValue(file, XmlFundShareClass.class);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    XmlFund fund;
    try {
        fund = objectMapper.treeToValue(root.getFund().path(FUND_ATTRIBUTES_NODE),
            XmlFund.class);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    extractGeneralData(root, fund);
    fund.setHoldings(extractPortfolio(root, objectMapper));
    fund.setPortfolioBreakdowns(getAllPortfolioBreakdowns(root));
    return fund;
}
```

Aktien des Fonds auslesen

Dieser Teil der Applikation stellte eine weniger große Herausforderung dar, da es die sogenannten ”Holdings” bereits in einer passenden Struktur im XML-File gibt. Es musste lediglich zum Knoten ”Holdings” im XML-File navigiert werden, da sich darunter eine Liste der ”HoldingDetail”-Knoten befindet. Wenn die .get() Methode auf HoldingDetail ausgeführt wird, wird im JsonNode ein Array aller ”HoldingDetail”-Knoten gespeichert, durch das iteriert werden kann.

Listing 52: iterate through HoldingDetail

```
 JsonNode portfolio =
     root.getFund().path(PORTFOLIO_LIST_NODE).path(PORTFOLIO_NODE).path(HOLDING_NODE);
JsonNode holdingsNode = portfolio.get(HOLDING_DETAIL_NODE);
if (holdingsNode != null && holdingsNode.isArray()) {
    holdingsNode.forEach(holding -> {
    //Logik
    });
}
```

Jeder HoldingNode speichert alle Informationen, die ein XmlHolding-Objekt benötigt, auf derselben Ebene und mit dem selben Namen der Attribute. Aus diesem Grund kann wieder die readValue()-Funktion der ObjectMapper-Klasse verwendet werden, um die meisten Werte zu mappen. Es müssen nur 2 Attribute manuell ausgelesen werden, da diese im XML-File denselben Namen ”_Id” haben.

PortfolioBreakdown auslesen

Die größte Herausforderung des XML-Lesens war das Auslesen der PortfolioBreakdowns eines Fonds. Der Grund dafür ist, (wie oben bereits erwähnt) dass diese unterschiedlich viele, unterschiedlich tiefe von uns genannte ”Allocation”-Knoten haben, die alle unterschiedliche Namen tragen. Die bisherige Lösung readValue() vom ObjectMapper reicht in diesem Fall nicht mehr aus, um diese Informationen zu extrahieren. Diese Daten müssen ausgelesen werden, da sie

wichtige Analysewerte beinhalten. Um sich das vorstellen zu können, ist hier ein kleines Beispiel anhand der IndiaAssetAllocation:

Listing 53: example Allocation

```
<IndiaAssetAllocation>
    <IndiaAssetTypeBreakdown Level="1">
        <BreakdownValue Type="11">99.74273</BreakdownValue>
    </IndiaAssetTypeBreakdown>
    <IndiaAssetTypeBreakdown Level="2">
        <BreakdownValue Type="1110">99.62907</BreakdownValue>
    </IndiaAssetTypeBreakdown>
    <IndianCustomAssetAllocation Level="1">
        <BreakdownValue Type="2">99.62907</BreakdownValue>
        <BreakdownValue Type="6">0.37093</BreakdownValue>
    </IndianCustomAssetAllocation>
    <IndianCustomAssetAllocation Level="2">
        <BreakdownValue Type="21">98.2656</BreakdownValue>
        <BreakdownValue Type="22">1.04767</BreakdownValue>
    </IndianCustomAssetAllocation>
</IndiaAssetAllocation>
```

Eine Lösung, um diese ”Allocation”-Nodes dennoch sinnvoll zu verarbeiten, ist eine rekursive Methode. Diese Methode heißt im DataimportService ”processAllocationNode(JsonNode node, XmlPortfolioBreakdown breakdown, XmlAllocation parent)”. Der JsonNode ist der aktuelle zu verarbeitende Node, breakdown ist das XmlPortfolioBreakdown-Objekt, in das die Daten gespeichert werden sollen, und parent ist die Allocation über den aktuellen Knoten. Der erste Aufruf der Methode erfolgt mit folgenden Werten: node = der ”PortfolioBreakdown”-Knoten, der verarbeitet werden soll; breakdown = ein leeres XmlPortfolioBreakdown-Objekt; parent = null. Um durch alle Elemente von JsonNode zu iterieren, gibt es die Funktion .fields(). Diese liefert einen Iterator, der über die Key-Value-Paare iteriert. Der Schlüssel eines JsonNodes ist der Name des Tags im XML-File. Danach wird ein Iterable-Objekt erzeugt. ”StreamSupport.stream(iter.spliterator(), false)” erzeugt einen Stream aus den Key-Value-Paaren, der sequentiell durchlaufen werden kann. iter.spliterator() konvertiert den Iterator in einem ”Spliterator”. Das ist notwendig, da sich die stream()-Methode einen Spliterator erwartet. In der forEach-Schleife kann mittels field.getValue() auf den JsonNode zugegriffen werden.

Listing 54: iterate through JsonNodes

```
Iterator<Map.Entry<String, JsonNode>> fields = node.fields();
Iterable<Map.Entry<String, JsonNode>> iter = () -> fields;
Stream<Map.Entry<String, JsonNode>> stream = StreamSupport.stream(iter.spliterator(),
    false);
stream.forEach(field -> {
    //Logic for a Key-Value-Pair
}
```

In der Schleife soll zuallererst festgestellt werden, ob es sich um eine Allocation/einen BreakdownValue oder um ein einzelnes Attribut handelt. Nach genauer Analyse der Unterschiede dieser Klassen wurde Folgendes festgestellt: Ein Knoten ist eine Allocation / ein BreakdownValue, wenn die Attribute nicht mit dem Prefix ”_” beginnen und wenn dieser Knoten Kinderelemente

besitzt, eine Elternallocation hat oder kein Array ist. Sollte das nicht auf einen Knoten zutreffen, handelt es sich um ein Attribut des Portfoliobreakdown-Knotens. Dieses Attribut ist Teil des Primärschlüssels der PortfolioBreakdown-Entität und muss demnach gespeichert werden. Ansonsten kann der Knoten weiterverarbeitet werden. Wenn der Key des Paares ”Breakdown-Value” ist, ist ein Blatt des JsonNode-Baums erreicht. Der Value des Paares ist eine Liste aus BreakdownValues, die an eine weitere Funktion gesendet wird. Dort werden die Daten extrahiert und als Liste von XmlBreakdownValues in das parent-Element dieses Knotens gespeichert. Wenn der Key gänzlich unbekannt ist, handelt es sich um eine Allocation. Es wird ein neues XmlAllocation-Objekt angelegt und alle notwendigen Daten werden darin gespeichert. Danach wird in einer Schleife, die durch die Kindelemente dieser Allocation iteriert, die Funktion erneut aufgerufen (rekursiv). Dieses Mal mit folgenden Werten: node = der Kindknoten der Allocation, breakdown = breakdown, parent = die derzeitige Allocation). Beim ”Unwinding” der Rekursion wird die allocation der XmlAllocation-Liste von breakdown hinzugefügt, und die Funktion endet.

In Datenbank speichern

Nachdem der gesamte Fond nun in eine XmlFund-Klasse geladen wurde, müssen alle Daten in die Datenbank geladen werden. Dazu müssen alle Business-Objekte (Xml...) in die Data Transfer Objekte (...DAO) umgewandelt werden. Für das wird der DAO-Mapper (3.5.1) verwendet. Die saveFund-Methode erwartet ein Objekt der FundDAO-Klasse. Die Methode ruft lediglich die insert-Funktion des FundRepositorys auf und gibt true oder false zurück, je nachdem, ob dieser Fond bereits in der Datenbank gespeichert ist. Sollte der Fond bereits in der Datenbank existieren, existieren auch die dazugehörigen Daten wie Holdings und PortfolioBreakdowns in der Klasse. Nach Absprache mit unserem Betreuer wurde klar, dass die Daten nicht aktualisiert werden müssen, sollten sie bereits existieren. Der Grund dafür ist, dass sich die XML-Files nicht verändern sollten.

Listing 55: save Fund in Database

```
1 private void writeToDatabase(XmlFund fund) {
2     FundDAO daoFund = DAOMapper.INSTANCE.fundToFundDAO(fund);
3     boolean alreadyExists = saveFund(daoFund);
4     if (!alreadyExists) {
5         saveAllDaoHoldings(fund);
6         getAllPortfolioBreakdownDAOs(fund);
7     }
8 }
```

Erwähnenswert ist außerdem, dass die im Xml als Allocations identifizierten Objekte nicht alle in die Allocations-Tabelle gespeichert werden sollen. Grund dafür ist eine spätere Änderung der DB. Allocations könnten nun auch Groups sein. Eine Group ist im Grunde eine übergeordnete

Allocation. Sie unterscheidet sich von Allocations darin, dass sie kein Elternelement hat, jedoch selbst ein Elternelement für andere Allocations ist. Nicht aber für BreakdownValues. Die restlichen Objekte können ganz normal mit dem DAOMapper gemappt werden und mit deren jeweiligen Repositories in die Datenbank gespeichert werden.

4 Ergebnis

4.1 Spring Rest API

4.1.1 Erwartungen

Die ursprüngliche Aufgabenstellung der Firma lautete: „Ein Java-Spring-REST-Backend, das Daten im JSON-Format zur Verfügung stellt, sowie die Implementierung mehrerer Unit-Tests.“

Im Laufe mehrerer Besprechungen kristallisierte sich heraus, dass eine API entwickelt werden soll, welche häufig veränderte Daten des Risk Services mit eher statischen Daten aus einer Datenbank kombiniert.

4.1.2 Ergebnisse

Das Endergebnis ist eine REST-API mit zwei Endpunkten (siehe Abbildung 26). Beide Endpunkte lesen Daten sowohl aus einer Datenbank als auch vom Risk Service aus und stellen diese einem Frontend zur Verfügung.

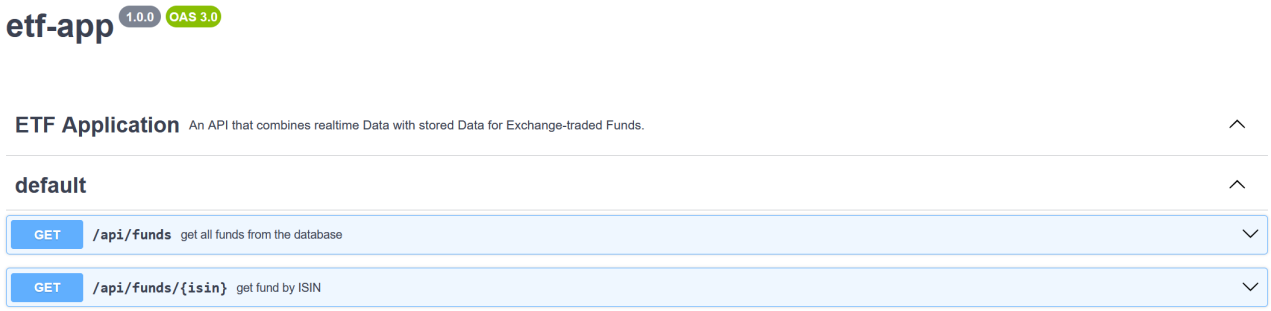


Abbildung 26: openapi.yml für die erstellte Rest API

Der erste Endpunkt (siehe Abbildung 27) gibt eine Liste aller ETFs im JSON-Format zurück. Er dient dazu, einem Frontend eine Übersicht über sämtliche in der Datenbank gespeicherten ETFs bereitzustellen.

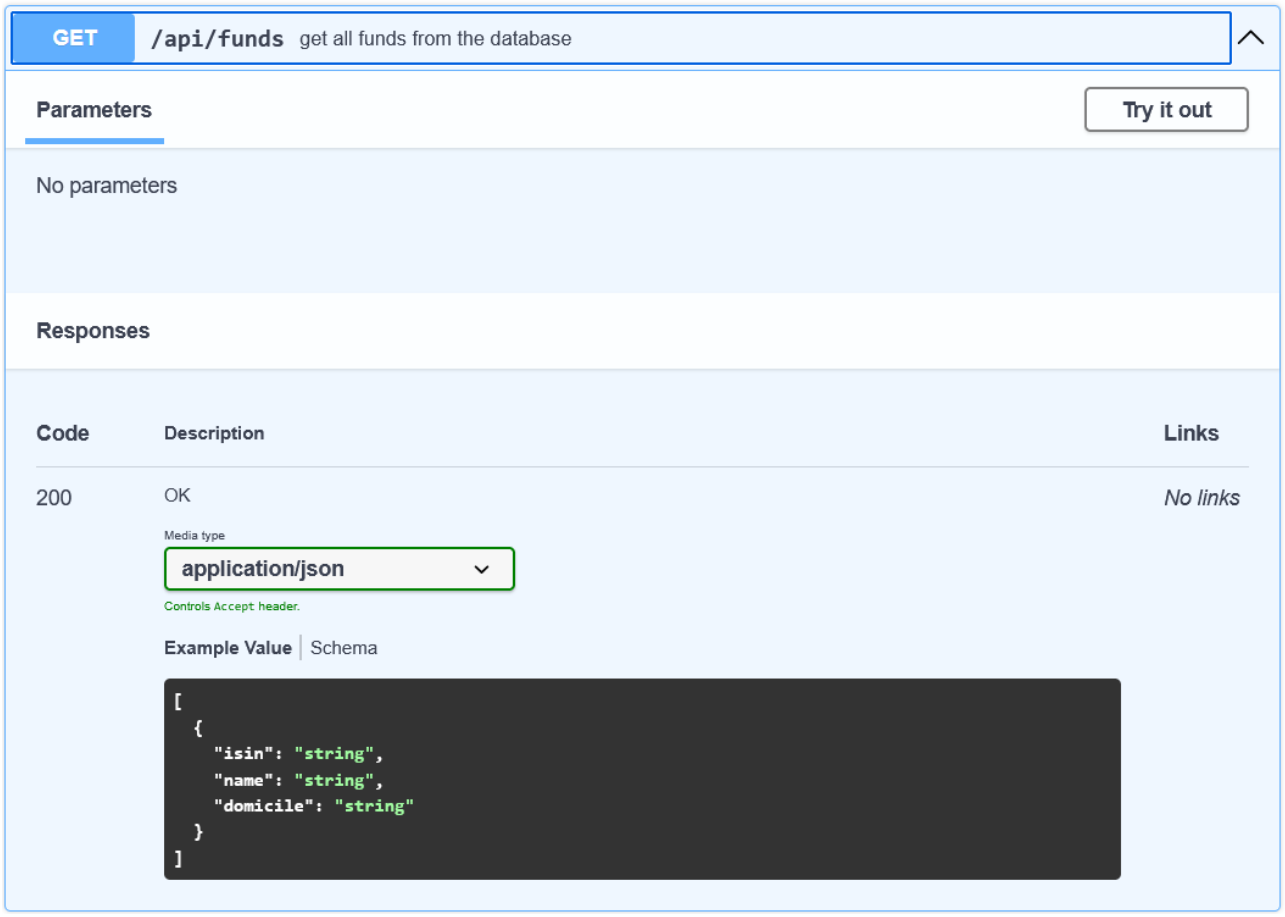


Abbildung 27: Erster Endpunkt, welcher alle ETFs in der Datenbank zurückgibt

Beim zweiten Endpunkt (siehe Abbildung 28) wird eine ISIN erwartet – dabei handelt es sich um eine eindeutige Identifikationsnummer für einen ETF. Als Antwort liefert die API detaillierte Informationen zu dem jeweiligen ETF. Zusätzlich werden über eine zweite API weitere Daten wie Kursverläufe und von der Kooperationsfirma berechnete Kennzahlen eingebunden.

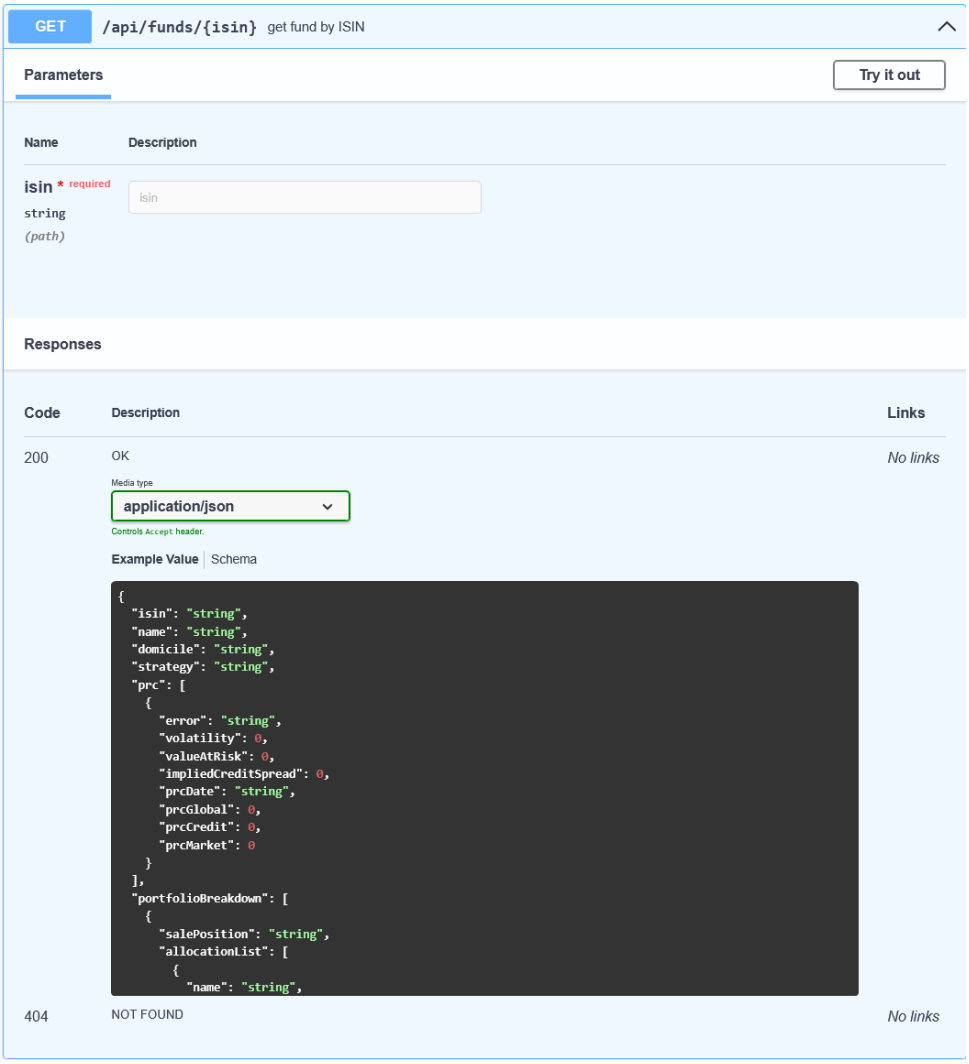


Abbildung 28: Zweiter Endpunkte für Details über einen ETF

4.2 Data Importer

4.2.1 Erwartungen

Die Aufgabe für den Data Importer lautete: "Das Sammeln von Testdaten aus externen Quellen (XML-Files) und danach das Speichern dieser in eine relationale Datenbank."
Um das Programm testen zu können, wurden einige XML-Files zum Testen zur Verfügung gestellt. Nach einigen Gesprächen mit unseren Ansprechpartner wurde klar, dass XML-Files, die ausgelesen werden sollen und in die Datenbank gespeichert werden sollen, in ein sogenanntes "hotDirectory"gelegt werden sollen. Es kann davon ausgegangen werden, dass diese die richtige Struktur aufweisen.

4.2.2 Ergebnis

Das Endergebnis ist eine SpringBoot-Anwendung mit einem "HotDirectory". Darin befindet sich eine Anleitung, die den simplen Prozess für den Nutzer / die Nutzerin beschreibt. Es müssen lediglich die XML-Files in diesem Verzeichnis abgelegt werden, und die Anwendung ausgeführt werden. Sollten sich Files darin befinden, die nicht auf ".xmlënden, stellt das kein Problem dar. Danach muss nur die Anwendung ausgeführt und gewartet werden. Je nachdem, wie lang diese Files sind, kann der Vorgang einige Minuten in Anspruch nehmen. Danach sind alle ETFs, bei denen die XML-Struktur der erwarteten Struktur entsprach, in der Datenbank gespeichert. Da es keine Benutzeroberfläche gibt, sind die einzigen Informationen, die der Nutzer / die Nutzerin bekommt, Fehlermeldungen oder Erfolgsinformationen in der Konsole.

4.3 Database

4.3.1 Erwartung

Als die Aufgaben verteilt wurden, war schon relativ klar, was das Ziel ist. Nämlich eine funktionierende PostgreSQL-Datenbank und ein dazugehöriges Datenmodell. Das Datenmodell soll ETfs darstellen, diese werden von der Firma über den Riskservice und mehrere XMLs bereitgestellt. Von Anfang an war klar, dass eine tiefere Analyse der Struktur nötig ist. Weiters muss sich darüber Gedanken gemacht werden, mit welchen Technologien die Datenbank läuft. Von Anfang an ist kommuniziert worden, dass diese Arbeit eine Grundlage für ein großes Projekt der Firma ist und dass das Datenmodell dementsprechend sauber sein muss. Denn wenn das Modell fehlerhaft ist, muss das ganze Projekt von vorne neu überdacht werden.

4.3.2 Ergebnis

Nach vier Wochen Arbeit ist eine Docker-Datenbank und ein Datenmodell entstanden.

Das Datenmodell 29 stellt einen ETF kompakt dar. In den Klassen sind viele Werte gespeichert, diese Werte gehen dann ins Detail und geben alle nötigen Informationen zu einem ETF.

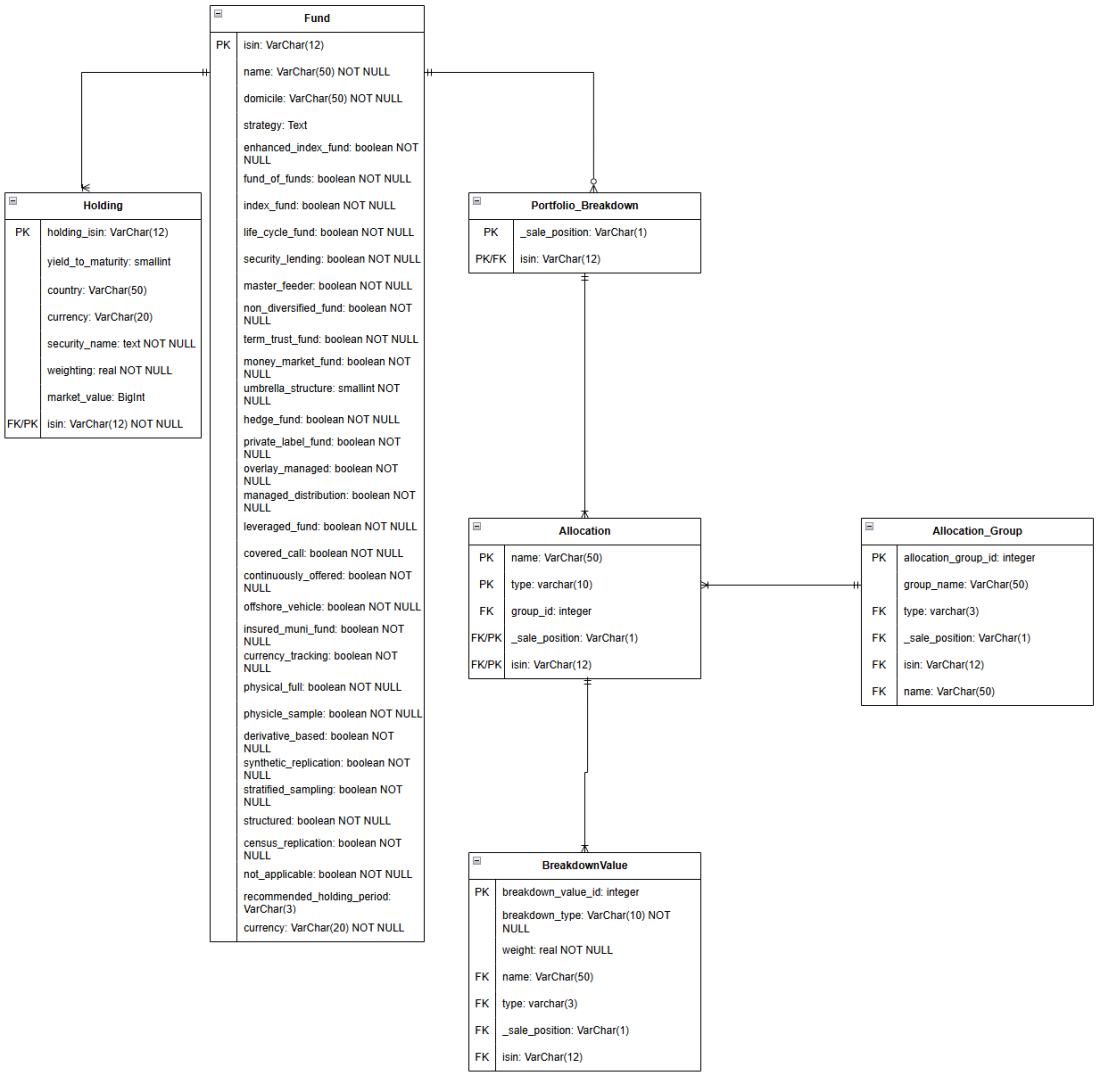


Abbildung 29: Ergebnis

Die zu verwendenden Werte sind nach Rücksprachen vorgegeben worden, weil nicht alle einen Sinn für die Auswertung haben. Alle Daten auszuwerten und zu verstehen, würde der Gruppe nicht viel bringen, außer dass man das Endprodukt, das verkauft wird, besser versteht. Die Datenbank wird von den Java-Anwendungen verwendet und das Datenmodell kann auch mit neuen ETfs problemlos umgehen.

4.4 Frontend

4.4.1 Erwartung

Die Aufgabe war ein Cross Platform Frontend Framework zu evaluieren und mit diesem einen Prototypen zu entwickeln. Für dieses sollte man ein bekanntes Framework nehmen. Des Weiteren sollte man mithilfe dieses Framework einen Prototypen entwickeln, der alle ETFs anzeigt. Auch

ETFs nach Ländern filtern sollte das Frontend können. Nebenbei sollte es noch eine Möglichkeit geben, nach ETFs zu suchen. Zu einem einzelnen ETF sollte man sich Details anzeigen lassen können.

4.4.2 Ergebnis

Es wurde in den Wochen des Praktikum einen Protype entwickelt, der, wie man in der Abbildung ËTF Übersicht"30 sehen kann. Des Weiteren kann man nach einzelnen ETFs suchen und sie nach Ländern filtern lassen. In der Abbildung ËTF Detail Seite"31 kann man sehen, wie die Detailseite eines einzelnen ETFs aussieht. Hier werden alle wichtigen Daten eines ETFs angezeigt. Die Grapik, die in der Abbildung ist, bildet die Preise der Preise über einen Zeitraum ab.

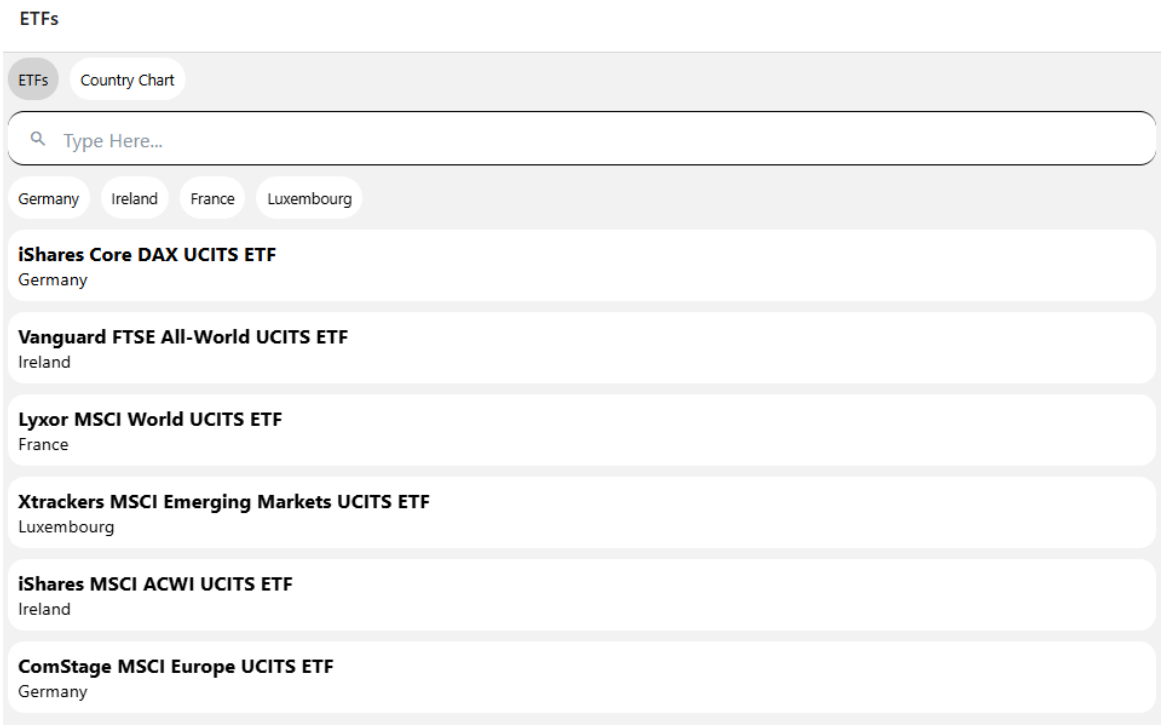


Abbildung 30: ETFs Übersicht

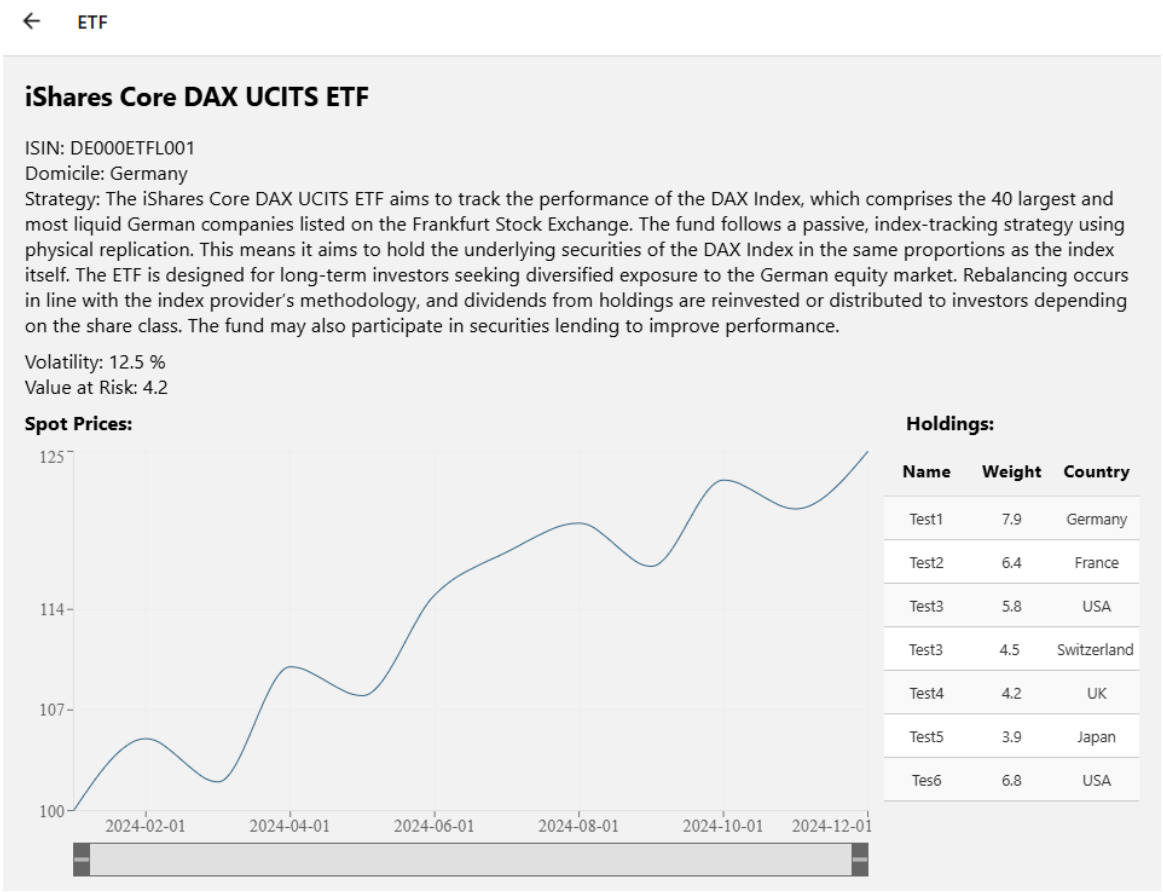


Abbildung 31: ETF Detail Seite

5 Resümee

5.1 Spring Rest API

Alles in allem ist das Projekt sehr erfolgreich verlaufen. Im Backend-Teil gibt es dennoch kleinere Probleme, die man anders hätte lösen können. Bezüglich des Datenschemas hätte ich entweder einen Code-First- oder einen Database-First-Ansatz wählen sollen. Zum Zeitpunkt der Diplomarbeit war mir dieser Ansatz jedoch nicht bekannt. Außerdem war die Benennung der Modellklassen nicht korrekt. Die DAO-Klassen entsprechen nicht dem typischen DAO-Schema. Die Aufteilung auf die beiden FundServices hätte ich besser im Voraus planen müssen. Es wäre zudem sinnvoller gewesen, die Logik auf mehrere Services zu verteilen.

5.2 XML-Importer

Grundsätzlich wurden die Ziele erreicht, wenn auch nicht perfekt. Würde das Projekt wiederholt werden, gäbe es einige Dinge, die anders hätten gemacht werden sollen. Der wahrscheinlich größte Fehler dieses Teils war die falsche Benennung der Klassen. Die Xml...-Klassen würden so beibehalten werden, aber ansonsten stimmen die Klassennamen nicht mit denen des DAO-Musters vorgesehenen überein. Beispielsweise würden die ...DAO-Klassen umbenannt werden in ...DTO-Klassen, da es sich dabei um die Data-Transfer-Objects handelt. Die eigentlichen DAO-Klassen sind in diesem Teil die Repository-Klassen. Als eine besonders große Hürde hat sich die rekursive Methode enthüllt. Es mussten immer alle Details im Auge behalten und bedacht werden, wie diese unterschiedlichen Maßnahmen in einer Methode untergebracht werden können, ohne dabei die anderen Knoten zu beeinflussen. Es wäre bestimmt besser und unkomplizierter gewesen, diesen Teil Stück für Stück zu programmieren, anstatt die ganze Methode auf einmal. Eine etwas spätere Änderung in der Datenbankstruktur (etwa nach 1,5 Wochen) führte dazu, dass einige Codeteile überarbeitet werden mussten. Der Teil, in dem die Daten "gecherrypicked"werden, ist natürlich nicht ideal und wenn dieses Projekt wiederholt werden würde, würde nach einer anderen, besseren Lösung gesucht werden.

5.3 Datenbank

Im Bereich Datenbanken ist das Projekt sehr gut verlaufen. Durch regelmäßige Besprechungen und Rücksprachen waren die Ziele auch klar definiert. Bei Problemen sind auch schnell Lösungen gefunden worden, einfach weil das Team gut miteinander kommuniziert hat. Das einzige Problem, das auch im Team zu größeren Diskussionen geführt hat, war die Handhabung der Allocations, aber selbst für dieses Problem wurden Lösungen gefunden. Was meinem Arbeitsfortschritt sehr geschadet hat, war, dass ich oft nicht die Details beachtet habe. Bevor überhaupt die größeren Themen besprochen werden konnten, mussten etliche Rechtschreibfehler ausgebessert werden. Weiters hätte mehr Zeit in Recherche nicht geschadet, viele Fehler wären erst gar nicht passiert, wenn ich mich von Anfang an genauer in das Thema eingelesen hätte. Im Großen und Ganzen kann ich aber sagen, dass ich glücklich mit dem Projekt, der Firma und der Gruppe bin. Ich habe einiges gelernt und weiß wie ich beim nächsten mal produktiver arbeiten kann.

5.4 Frontend

Die Ziele wurden im Bereich Frontend umgesetzt, es gibt natürlich Dinge, die man anders hätte machen können, aber ich denke, es ist mir ganz gut gelungen. Bei Expo Router gab es anfänglich ein paar Probleme mit der Umsetzung, aber am Ende ist es etwas geworden. Ich bin sehr froh, dass ich mich bei dem Laden der Daten für axios anstelle von fetch entschieden zu haben, weil es mir den Zugriff auf die Daten vom Backend wesentlich erleichtert haben. Alles in Allem ist es mir gut gelungen, auch wenn ich das Projekt nochmal machen würde, würde ich sicher einiges ändern.

Literaturverzeichnis

[1] M. Helmich, „RESTful Webservices (1): Was ist das überhaupt?" REST API. Online verfügbar: https://www.mittwald.de/blog/webentwicklung-design/restful-webservices-1-was-ist-das-uberhaupt

[2] K. P. Nilang Patel, *Java 9 Dependency Injection: Write loosely coupled code with Spring 5 and Guice*. Packt Publishing Ltd, 2018.

[3] „Flutter - Google Developers," https://flutter.dev, 2025.

[4] „React Native - Meta," https://reactnative.dev, 2025.

[5] „.NET MAUI – Microsoft Learn," https://learn.microsoft.com/en-us/dotnet/maui/, 2025.

[6] „Ionic Framework," https://ionicframework.com, 2025.

[7] „Thoughtworks Technology Radar," https://www.thoughtworks.com/radar, 2025.

[8] „Progressive Web Apps – Google Developers," https://developer.google.com/web/progressive-web-apps, 2025.

[9] „Progressive Web Apps (PWAs) – Mozilla Developer Network," https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps, 2025.

[10] „Your First Progressive Web App – web.dev (Google)," https://web.dev/learn/pwa/, 2025.

[11] „Progressive Web Apps (PWAs) on Windows – Microsoft Docs," https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/, 2025.

[12] „Expo – An open-source platform for making universal native apps with React," https://expo.dev, 2025.

[13] „Expo Documentation," https://docs.expo.dev, 2025.

[14] R. Zicari, „Objectmodel," Objectmodel. Online verfügbar: https://www.odbms.org/2010/06/introduction-to-databases-object-and-object-relational-databases/

[15] J. W. schmidt, *Relational Database Systems: Analysis and Comparison*. Springer Verlag.

[16] databricks, „ACID Transactions," ACID. Online verfügbar: https://www.databricks.com/glossary/acid-transactions

[17] M. Saraswatipura, „Database Concurrency in PostgreSQL," Concurrency-Methods. Online verfügbar: https://www.red-gate.com/simple-talk/databases/postgresql/database-concurrency-in-postgresql/

[18] S. J. Bigelow, „Open core vs. open source: What's the difference?" Open-Source-Core. Online verfügbar: https://www.techtarget.com/searchitoperations/tip/Open-core-vs-open-source-Whats-the-difference

[19] redhat, „Was sind Microservices?" Micorservices. Online verfügbar: https://www.redhat.com/de/topics/microservices/what-are-microservices

[20] C. Harris, „Microservice-Architektur," Microservices-Funktionen. Online verfügbar: https://www.atlassian.com/de/microservices/microservices-architecture/

[21] P. Onyeanuna, „How APIs Power Microservices Architectures 101 Guide," API. Online verfügbar: https://www.getambassador.io/blog/apis-microservices-architectures-guide

[22] statista, „Die beliebtesten Programmiersprachen weltweit laut PYPL-Index im März 2025," beliebtesteProgrammiersprache. Online verfügbar: https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index/

[23] W. editors, „Java (Programmiersprache)," java. Online verfügbar: https://de.wikipedia.org/wiki/Java_(Programmiersprache)

[24] C. Ullenboom, *Java ist auch eine Insel*. Rheinwerk Computing, 2023.

[25] J. Team, „Was ist JFrog Artifactory?" JFrog. Online verfügbar: https://jfrog.com/de/blog/what-is-artifactory-jfrog/

[26] J. Erolin, „Git, GitHub, and GitLab: What's the Difference?" was-ist-git. Online verfügbar: https://www.bairesdev.com/blog/git-github-and-gitlab-whats-the-difference/

[27] git, „Getting Started - A Short History of Git," git-history. Online verfügbar: https://git-scm.com/book/ms/v2/Getting-Started-A-Short-History-of-Git

[28] C. Möhring, „GitHub vs. GitLab," gitlab. Online verfügbar: https://www.heise.de/tipps-tricks/GitHub-vs-GitLab-4597154.html

[29] S. Pittet, „Continuous integration vs. delivery vs. deployment," CD-CI. Online verfügbar: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[30] geeksforgeeks, „Git Features," git. Online verfügbar: https://www.geeksforgeeks.org/git-features/

[31] Docker, „DockerL," Docker. Online verfügbar: https://docs.docker.com

[32] A. S. Gillis, „Containerbasierte Virtualisierung," Containerbasierte-Virtualisierung. Online verfügbar: https://www.computerweekly.com/de/definition/Containerbasierte-Virtualisierung

[33] gridscale, „Was ist Containervirtualisierung?" was-ist-Containervirtualisierung. Online verfügbar: https://gridscale.io/community/glossar/was-ist-containervirtualisierung/

[34] I. Buchanan, „Container vs. virtuelle Computer," Container-vs-VM. Online verfügbar: https://www.atlassian.com/de/microservices/cloud-computing/containers-vs-vms

[35] R. Powell, „Benefits of containerization," containerization. Online verfügbar: https://circleci.com/blog/benefits-of-containerization/#c-consent-modal

[36] R. Hat, „What is a hypervisor?" hypervisor. Online verfügbar: https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

[37] rheinwerk, „Dämonprozesse," daemon. Online verfügbar: https://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap07-011.htm

[38] E. Mell, „Docker Engine," docker-engine. Online verfügbar: https://www.computerweekly.com/de/definition/Docker-Engine

[39] cto, „Differences Between a DockerFile, Docker Image, and Docker Container," DockerFile-Image. Online verfügbar: https://cto.ai/blog/docker-image-vs-container-vs-dockerfile/#:~:text=A%20Dockerfile%20is%20the%20Docker,to%20build%20the%20image%20itself

[40] geeksforgeeks, „What is Docker Hub?" DockerHub. Online verfügbar: https://www.geeksforgeeks.org/what-is-docker-hub/

[41] docker, „Networking overview," Networking-eins. Online verfügbar: https://docs.docker.com/engine/network/

[42] J. Walker, „Docker Networking – Basics, Network Types Examples," Networking-zwei. Online verfügbar: https://spacelift.io/blog/docker-networking/

[43] ——, „Docker Networking – Basics, Network Types Examples," Networkstack. Online verfügbar: https://spacelift.io/blog/docker-networking/

[44] ——, „Docker Compose – What is It, Example Tutorial," Docker-Compose. Online verfügbar: https://spacelift.io/blog/docker-compose

[45] S. D. Korry Douglas, *PostgreSQL: A Comprehensive Guide to Building, Programming, and Administering PostgresSQL Databases*. Sams Publishing, 2003.

[46] „What Is PostgreSQL?" official PostgreSQL. Online verfügbar: https://www.postgresql.org/docs/current/intro-whatis.html

[47] PostgreSQL, „A Brief History of PostgreSQL," Postgre-History. Online verfügbar: https://www.postgresql.org/docs/current/history.html

[48] ibm, „What is PostgreSQL?" what-is-PostgreSQL. Online verfügbar: https://www.ibm.com/think/topics/postgresql

[49] PostgreSQL, „What is PostgreSQL?" why-PostgreSQL. Online verfügbar: https://www.postgresql.org/about/

[50] geeksforgeeks, „PostgreSQL – Data Types," PostgreSQL-DataTypes. Online verfügbar: https://www.geeksforgeeks.org/postgresql-data-types/

[51] A. Zanini, „Understanding PostgreSQL Data Integrity," PostgreSQL-Dataintegrity. Online verfügbar: https://www.dbvis.com/thetable/understanding-postgresql-data-integrity/

[52] PostgreSQL, „Performance Tips," PostgreSQL-Perfomanc. Online verfügbar: https://www.postgresql.org/docs/8.1/performance-tips.html#:~:text=PostgreSQL%20devises%20a%20query%20plan,tries%20to%20select%20good%20plans

[53] ——, „Reliability," PostgreSQL-Reliability. Online verfügbar: https://www.postgresql.org/docs/current/wal-reliability.html

[54] ——, „Security," PostgreSQL-security. Online verfügbar: https://www.postgresql.org/docs/7.0/security.htm

[55] edb, „Mastering PostgreSQL: Strategies for Scalability and High Performance," PostgreSQL-Scaling. Online verfügbar: https://www.enterprisedb.com/scaling-postgresql-high-availability-and-performance

[56] The tutorialpoint Team, „JUnit - Overview," Short introduction to JUnit. Online verfügbar: https://www.tutorialspoint.com/junit/junit_overview.htm

[57] Microsoft, „XML für Anfänger," XML-Basics. Online verfügbar: https://support.microsoft.com/de-de/office/xml-f%C3%BCr-anf%C3%A4nger-a87d234d-4c2e-4409-9cbc-45e4eb857d44

[58] geeksforgeeks, „POJO vs Java Beans," Explaination of POJO and JavaBeans. Online verfügbar: https://www.geeksforgeeks.org/data-access-object-pattern/

[59] ——, „Data Access Object(DAO) Design Pattern," Explaination of DAO. Online verfügbar: https://www.baeldung.com/java-dao-pattern

[60] Spring Team, „Introduction to Spring Framework," Spring documentation. Online verfügbar: https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html

[61] Spring, „Dependency Injections Spring Tutorial," Online Tutorial. Online verfügbar: https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

[62] N. O. Dashrath Mane, Ketaki Chitnis, „The Spring Framework: An Open Source Java Platform for Developing Robust Java Applications," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 2013.

[63] R. R. Karanam, *Mastering Spring 5.0.* Packt Publishing Ltd, 2017.

[64] Spring, „Spring Cloud OpenFeign," die offizielle Feign Dokumentation. Online verfügbar: https://docs.spring.io/spring-cloud-openfeign/docs/current/reference/html/

[65] M.-H. F. G. J. Heinisch, C., *Java als erste Programmiersprache.* Vieweg+Teubner, 2011.

[66] S. Team, „Spring Data Relational," Spring JDBC. Online verfügbar: https://docs.spring.io/spring-data/relational/reference/jdbc/why.html

[67] Auth0, „Was ist OAuth 2.0?" OAuth Grundlagen. Online verfügbar: https://auth0.com/de/intro-to-iam/what-is-oauth-2

[68] Javapoint, „Jackson Tutorial," Online Tutorial. Online verfügbar: https://www.javatpoint.com/jackson

[69] The Apache Xerces Project, „Xerces Project Charter," Introduction to the project. Online verfügbar: https://xerces.apache.org/xerces2-j/charter.html

[70] ——, „XNI Design Details," Details to the Framework. Online verfügbar: https://xerces.apache.org/xerces2-j/xni-design.html

[71] Gunnar Morling, Andreas Gudian, Sjaak Derksen, Filip Hrisafov and the MapStruct community, „MapStruct 1.6.3 Reference Guide," die offizielle Mapstruct Dokumentation. Online verfügbar: https://mapstruct.org/documentation/stable/reference/html/

[72] The Project Lombok Authors, „Project Lombok," Official Lombok Website. Online verfügbar: https://projectlombok.org

[73] S. Team, „Annotation Interface AutoConfigureMockMvc," Spring MockMvc. Online verfügbar: https://docs.spring.io/spring-boot/3.3/api/java/org/springframework/boot/test/autoconfigure/web/servlet/AutoConfigureMockMvc.html

[74] Meta (ehemals Facebook), „React Native - Getting Started," https://reactnative.dev/docs/getting-started, 2024.

[75] LogRocket, „React Native vs. Native App Development: How to choose the right framework," https://blog.logrocket.com/react-native-vs-native-app-development/, 2023.

[76] Meta, „React Native Components and APIs Overview," https://reactnative.dev/docs/components-and-apis, 2024.

[77] Meta (ehemals Facebook), „React Native Blog – Aktuelle Entwicklungen und Ankündigungen," https://reactnative.dev/blog, 2024.

[78] React, „React Native," die offizielle React Native Dokumentation. Online verfügbar: https://reactnative.dev/

[79] LogRocket, „React Native vs. Native App Development: How to choose the right framework," https://blog.logrocket.com/react-native-vs-native-app-development/, 2023.

[80] Simform, „Top Benefits of Cross-Platform App Development," https://www.simform.com/blog/cross-platform-app-development-benefits/, 2023.

[81] Meta, „React Native Networking – Fetch and APIs," https://reactnative.dev/docs/network, 2024.

[82] Expo, „Config Plugins – Customize the native code in Expo projects," https://docs.expo.dev/guides/config-plugins/, 2024.

[83] ——, „Introduction to Expo," https://docs.expo.dev/, 2024.

[84] Meta, „Fast Refresh in React Native," https://reactnative.dev/docs/fast-refresh, 2024.

[85] LogRocket, „React Native vs. Native App Development: How to choose the right framework," https://blog.logrocket.com/react-native-vs-native-app-development/, 2023.

[86] Simform, „Flutter vs. React Native vs. Ionic – What to Choose in 2023," https://www.simform.com/blog/flutter-vs-react-native-vs-ionic/, 2023.

[87] Meta, „Native Modules – Bridging JavaScript and Native Code," https://reactnative.dev/docs/native-modules-intro, 2024.

[88] A. Team, „About Angular – Developed by Google," https://angular.io/about, 2024.

[89] ——, „Angular Documentation – Overview," https://angular.io/docs, 2024.

[90] ——, „Progressive Web Apps with Angular," https://angular.io/guide/service-worker-intro, 2024.

[91] ——, „@angular/pwa Package – Add PWA support to Angular apps," https://angular.io/guide/pwa, 2024.

[92] ——, „Angular Components Overview," https://angular.io/components, 2024, Zugriff am 6. April 2025.

[93] ——, „Testing Angular Applications," https://angular.io/guide/testing, 2024.

[94] ——, „Browser Support for Angular," https://angular.io/guide/browser-support, 2024.

[95] ——, „Angular Architecture Overview," https://angular.io/guide/architecture, 2024.

[96] LogRocket, „Angular vs. React vs. Vue – A comparison of front-end frameworks," https://blog.logrocket.com/angular-vs-react-vs-vue/, 2023.

[97] S. of JavaScript, „State of JS 2022 – Front-End Frameworks Usage and Satisfaction," https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/, 2022.

[98] S. Magazine, „A Comparison of Angular, React and Vue," https://www.smashingmagazine.com/2020/01/angular-react-vue-comparison/, 2020.

[99] I. Team, „Ionic Documentation – Native Look Feel with Web Technologies," https://ionicframework.com/docs, 2024.

[100] ——, „Why Ionic – Build once, run anywhere," https://ionicframework.com/why-ionic, 2024.

[101] C. Team, „Capacitor Documentation – Native APIs and Plugins," https://capacitorjs.com/docs/plugins, 2024.

[102] I. Team, „Ionic PWA Toolkit – Build Progressive Web Apps with Ionic," https://ionicframework.com/pwa, 2024.

[103] ——, „Why Ionic – One Codebase for All Platforms," https://ionicframework.com/why-ionic, 2024.

[104] C. Team, „Capacitor Plugins – Native Access for Web Apps," https://capacitorjs.com/docs/plugins, 2024.

[105] I. Team, „Testing in Ionic Angular Apps," https://ionicframework.com/docs/angular/testing, 2024.

[106] ——, „Hot Reload vs. Live Reload – What's the Difference?" https://ionicframework.com/blog/hot-reload-vs-live-reload-whats-the-difference/, 2023.

[107] LogRocket, „Flutter vs. Ionic – A performance and UX comparison," https://blog.logrocket.com/flutter-vs-ionic-comparison/, 2023.

[108] Simform, „Flutter vs. React Native vs. Ionic – What to Choose in 2023," https://www.simform.com/blog/flutter-vs-react-native-vs-ionic/, 2023.

[109] C. Team, „Capacitor Plugins – Native Access for Web Apps," https://capacitorjs.com/docs/plugins, 2024.

[110] F. Team, „Flutter – Build apps for any screen," https://developers.google.com/flutter, 2024.

[111] ——, „Rendering with Flutter – Skia Engine," https://flutter.dev/docs/resources/architectural-overview#rendering-layer, 2024.

[112] D. Team, „Dart Language Overview," https://dart.dev/language, 2024.

[113] F. Team, „Flutter Widgets – Everything is a widget," https://flutter.dev/docs/development/ui/widgets, 2024.

[114] ——, „Flutter Web Support," https://flutter.dev/web, 2024.

[115] ——, „Flutter Hot Reload," https://flutter.dev/docs/development/tools/hot-reload, 2024.

[116] G. Developers, „Flutter – Build apps for any screen," https://developers.google.com/flutter, 2024.

[117] TechRepublic, „Flutter vs. Ionic: Which mobile development framework is better?" https://www.techrepublic.com/article/flutter-vs-ionic/, 2023.

[118] F. A. Docs, „ColorScheme class – Material 3," https://api.flutter.dev/flutter/material/ColorScheme-class.html, 2024.

[119] LogRocket, „Flutter vs. Ionic – A performance and UX comparison," https://blog.logrocket.com/flutter-vs-ionic-comparison/, 2023.

[120] F. G. I. Tracker, „Issue #14641 – Problems rendering animations and vector graphics," https://github.com/flutter/flutter/issues/14641, 2018.

[121] F. Team, „Flutter Web SEO – What You Need to Know," https://medium.com/flutter/flutter-web-seo-what-you-need-to-know-8d342aed6d3b, 2020.

[122] S. Overflow, „Why is Flutter Web not using HTML/CSS/JS directly?" https://stackoverflow.com/questions/57772849/why-is-flutter-web-not-using-html-css-js-directly, 2019.

[123] D. Team, „Dart Language – Type System Overview," https://dart.dev/language/type-system, 2024.

[124] Expo, „Routing in Expo – Expo Router," https://expo.dev/router, 2024.

[125] ——, „app.json / app.config.js – Project Configuration," https://docs.expo.dev/workflow/configuration/, 2024.

[126] ——, „Using TypeScript with Expo," https://docs.expo.dev/guides/typescript/, 2024.

[127] R. N. Community, „Best Practices for React Native Project Structure," https://reactnative.dev/docs/structure, 2023.

[128] Expo, „Styling in Expo – Best Practices," https://docs.expo.dev/guides/styling/, 2024.

[129] Axios. (n.d.) Axios Documentation. Online verfügbar: https://axios-http.com/docs/intro

[130] Mozilla. (n.d.) Using Fetch. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

[131] ——. (n.d.) Promise – JavaScript. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[132] ——. (n.d.) async function – JavaScript. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

[133] S. Team, „Spring Framework Documentation – Web MVC CORS Support," https://docs.spring.io/spring-framework/reference/web/webmvc/cors.html, 2024.

[134] Baeldung, „CORS with Spring Boot," https://www.baeldung.com/spring-cors, 2023.

[135] M. W. Docs, „HTTP Access Control (CORS)," https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS, 2024.

[136] Recharts, „Reacharts," die offizielle Recharts Dokumentation. Online verfügbar: https://recharts.org/en-US/api

[137] G. Team, „Spring Boot – CrudRepository with Example," Spring CrudRepository. Online verfügbar: https://www.geeksforgeeks.org/spring-boot-crudrepository-with-example/

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

Schüler	**Individuelle Themenstellung**
Jan Oppitz	Java Spring Backend
Armin Hintersteiniger	PostgreSQL Datenbank
Gabrielle Praher	Frontend Evaluierung
Adrian Tagwerker	Einlesen in Datenbank

Tabelle 1: Individuelle Themenstellungen der Schüler

B Meilensteine

Meilenstein	**Datum**
Das Schema der XML-Files beschafft	08.07.2024
Datenbank Schema analysiert	09.07.2024
Liste von Cross-Plattform Frontend Frameworks erstellt	11.07.2024
in Java Spring eingearbeitet	12.07.2024
Datentypen für die API erstellt	17.07.2024
Erste Auswahl aller möglichen Frameworks durchgeführt.	17.07.2024
Ko Kriterien gefunden und nähere Auswahl getroffen	18.07.2024
Datenbank Datentypen erstellt	18.07.2024
Für Framework entschieden	19.07.2024
Endpunkte/ Library für Datentypen erstellt	22.07.2024
Skript zum Einlesen der XML-Files erstellt und in passende Objekte gespeichert	22.07.2024
Erstes Datenbank Konzept visualisiert	25.07.2024
Die Daten vom RISK-Service in die Datenbank importiert	25.07.2024
Endpunkte für ETFs erstellt	29.07.2024
Erste DB-Version erstellt	31.07.2024
Abfrage von Risk-Service automatisiert	01.08.2024
Prototyp mithilfe des ausgewählten Frameworks erstellt	01.08.2024
UNIT-Tests für die Requests erstellt	02.08.2024
DB implementiert	02.08.2024

Tabelle 2: Projektmeilensteine mit Datum

C ChatGPT

Bei der Erstellung der entsprechend gekennzeichneten Teile dieser Diplomarbeit wurde ChatGPT-4 eingesetzt. Dieses Tool ist unter der Adresse https://chat.openai.com/chat online verfügbar. In den folgenden Abschnitten sind die Benutzereingaben und die dazugehörigen Ausgaben von ChatGPT dargestellt.

C.1 LaTeX

Eingabe: Erkläre in einem Satz was Latex ist.

Ausgabe: LaTeX ist ein textbasiertes Schriftsatzsystem, das hauptsächlich zur Erstellung von wissenschaftlichen Dokumenten verwendet wird, da es präzise Formatierung und mathematische Formeln unterstützt.

Eingabe: Nenne eine Quelle für die obige Formulierung

Ausgabe: Die Formulierung basiert auf einer allgemeinen Definition von LaTeX. Eine ähnliche Beschreibung findest du auf der offiziellen LaTeX-Projektseite:

Quelle: https://www.latex-project.org/about/

Projektübergabe-Bestätigung

Projektname: ETF-App

Projektverantwortlicher: Armin Hintersteininger

Datum der Übergabe: 30.6.2024

Hiermit wird bestätigt, dass das oben genannte Projekt am 30.7.2024 ordnungsgemäß und vollständig übergeben wurde.

Die Übergabe erfolgte an Simon Primetzhofer, und umfasst folgende Inhalte und Zuständigkeiten:

Projektdokumentation: sämtliche Konzepte, Betriebsanleitung

Technische Ressourcen:

Projekt welches im Praktikum erstellt wurde.

Die Übergabe wurde in gegenseitigem Einverständnis und mit Klarheit über alle offenen Punkte durchgeführt. Beide Parteien bestätigen, dass sie sämtliche notwendigen Informationen erhalten bzw. bereitgestellt haben.

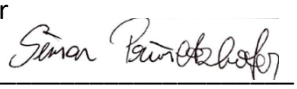
Unterschriften:

Armin Hintersteininger

Unterschrift: _____

Datum: _____

Simon Primetzhofer

Unterschrift: 

Datum: 29.01.2025