ETF Analyzer Pro

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/07/2025 – 26/03/2026

Projektmitglieder: Michael Gillhofer
Lukas Leitner

Betreuer: Prof. Mag. Michael Holzmann

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbstständig, 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 hat das Projektteam das generative KI-Tool ChatGPT lediglich für Erklärungen zum besseren Verständnis technisch komplexer Zusammenhänge verwendet.

Perg, 25.03.2026

Ort, Datum

Unterschrift, Michael Gillhofer

Perg, 25.03.2026

Ort, Datum

Unterschrift, Lukas Leitner

Danksagung

An dieser Stelle möchten wir uns bei all denjenigen bedanken, die uns während der Anfertigung der Diplomarbeit unterstützt und motiviert haben.

Zuerst gebührt unser Dank Herrn Professor Mag. Michael Holzmann, der unsere Diplomarbeit betreut und begutachtet hat und Frau Professor DI (FH) Gudrun Heinzelreiter-Wallner, MSc, BEd, die uns im Rahmen des Unterrichts die Grundlagen des wissenschaftlichen Arbeitens vermittelt hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchten wir uns herzlich bedanken.

Ein besonderer Dank gilt allen Mitwirkenden der uni software plus GmbH, die uns bei administrativen Aufgaben unterstützt haben. Vor allem gilt der Dank Dipl.-Ing. Simon Primetzhofer, der uns als engagierter und zuverlässiger Ansprechpartner und mit seiner fachkundigen Kompetenz zur Seite gestanden ist.

Abschließend möchten wir auch unseren Eltern danken, die uns während unserer Ausbildung an der HTL Perg und insbesondere in der Umsetzungsphase der Diplomarbeit stets unterstützt und uns mentalen Rückhalt gegeben haben.

Abstract

The basic idea behind the **ETF Analyzer Pro** is to develop a web application for investors in the ETF sector. So-called *Exchange Traded Funds* are a popular financial product and, unlike individual shares, spread capital across multiple companies and, depending on their composition, across different sectors, countries or currency regions. However, choosing the right fund to suit one's needs presents challenges, particularly for beginners, as many key metrics and attributes are difficult to understand or consist purely of numerical values. The application addresses this through a clear, graphical presentation of the data, as well as the ability to search for, filter and compare any funds.

Various technologies were used to implement the project. The backend utilises Java Spring Boot and delivers the required data and functionalities to the frontend, which runs on Angular. Furthermore, an external interface is used to retrieve current price values for an ETF. Finally, the option is provided to import new funds from structured XML files and persist them in the PostgreSQL database. When implementing the user interface, particular attention was paid to ensuring a clear and interactive presentation, which is why various predefined components from PrimeNG were used.

The project builds on several undergraduate and master's theses and is intended as an extension or completion of these. The front-end will be completely re-implemented based on a specified design, whilst the back-end will be expanded to include a fund comparison function. uni software plus GmbH is acting as the project partner, and supervision is provided by Dipl.-Ing. Simon Primetzhofer.

The result is a fully functional web application that provides users with easier access to the world of investment in the form of exchange-traded funds. Furthermore, thanks to its modular structure, the entire application is easy to maintain and expand.

Kurzfassung

Die Grundidee der **ETF Anaylzer Pro** besteht darin, eine Webapplikation für Anlegerinnen und Anleger im Bereich der ETFs zu implementieren. Sogenannte *Exchange Traded Funds* sind ein beliebtes Finanzprodukt und streuen das Kapital, im Gegensatz zu Einzelaktien, auf mehrere Unternehmen und abhängig von der Zusammensetzung auf verschiedene Branchen, Länder oder Währungsregionen auf. Jedoch bringt die richtige Auswahl eines auf sich zugeschnittenen Fonds vor allem für Einsteigerinnen und Einsteiger Herausforderungen mit sich, da viele wichtige Kennzahlen und Attribute schwer verständlich oder reine Zahlenwerte sind. Die Applikation löst dies durch eine übersichtliche, grafische Aufbereitung der Daten, sowie den Möglichkeiten nach beliebigen Fonds zu suchen, zu filtern und diese auch miteinander zu vergleichen.

Zur Umsetzung des Projekts kamen verschiedene Technologien zum Einsatz. Das Backend verwendet Java Spring Boot und liefert die geforderten Daten und Funktionalitäten an das Frontend, das mit Angular funktioniert. Des weiteren wird eine externe Schnittstelle verwendet, um aktuelle Kurswerte eines ETFs zu erhalten. Abschließend wird die Möglichkeit geboten, aus strukturierten XML-Dateien neue Fonds einzulesen und diese in der PostgreSQL-Datenbank zu persistieren. Bei der Implementierung der Benutzeroberfläche musste vor allem auf eine übersichtliche und interaktive Darstellung geachtet werden, und deswegen wurden diverse vordefinierte Komponenten von PrimeNG verwendet.

Das Projekt setzt auf mehrere Diplom- bzw. Masterarbeiten auf und wird als Erweiterung bzw. Fertigstellung gesehen. Das Frontend wird basierend auf einem vorgegebenen Design zur Gänze neu implementiert, während das Backend um die Vergleichsfunktion mehrerer Fonds ergänzt wird. Als Projektpartner fungiert die uni software plus GmbH und die Betreuung erfolgt durch Dipl.-Ing. Simon Primetzhofer.

Das Ergebnis ist eine voll funktionsfähige Webanwendung, die Nutzerinnen und Nutzern einen einfacheren Zugang zur Welt der Kapitalanlage in Form von ETFs ermöglicht. Ebenso ist die komplette Anwendung aufgrund ihres modularen Aufbaus leicht wart- und erweiterbar.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Das Projekt **ETF Analyzer Pro** wurde ins Leben gerufen, um interessierten Anlegerinnen und Anlegern einen einfacheren Einstieg in die Welt der ETFs zu ermöglichen. Die Schwierigkeit liegt darin, zwischen einer Vielzahl von Fonds den richtigen für seine Ziele zu finden. ETFs unterscheiden sich grundlegend in ihren Eigenschaften und Anlagestrategien, dadurch ist eine konkrete Gegenüberstellung ohne genauere Vorkenntnisse im Finanzsektor nur schwer möglich.

Die entwickelte Applikation soll genau dieses Problem lösen. Im Rahmen einer vorangegangenen Diplomarbeit wurden bereits einige Teilbereiche implementiert. Dadurch können bereits ETFs in einer Datenbank persistiert, über einen XML-Importer geladen und mittels einer API zur Verfügung gestellt werden. Ebenso wurde ein Frontend-Framework evaluiert und ein erster Prototyp in Angular implementiert.

Die Aufgaben des Projektteams in dieser Diplomarbeit belaufen sich auf die Weiterentwicklung der Applikation und werden im folgenden genauer erklärt:

- Implementierung eines neuen Frontends auf Basis des Frameworks Angular. Besonderer Fokus liegt dabei auf einer ansprechenden und benutzerfreundlichen Gestaltung, da die Benutzeroberfläche produktreif umgesetzt werden soll.
- Erweiterung um die Funktion *ETF-Comparison*. Diese soll es ermöglichen, eine Vielzahl von Fonds aufgrund ihrer Eigenschaften miteinander zu vergleichen und die besten Werte hervorzuheben. Für die Realisierung muss nicht nur eine Benutzeroberfläche umgesetzt, sondern auch die gesamte Logik im Backend geschaffen werden.

1.2 Projektumfeld

Bei der Umsetzung der Diplomarbeit waren viele Personen involviert. Im folgenden Abschnitt werden diese Beteiligten und ihre Rollen im Projekt näher vorgestellt.

1.2.1 Projektteam

Das Projektteam setzt sich aus Michael Gillhofer und Lukas Leitner zusammen, beide Schüler der Höheren technischen Bundeslehranstalt Perg. Aufgrund der Erfahrung in vorangegangenen Projekten hat sich Michael um die Backendfunktionalitäten und die Datenbankanbindung gekümmert, während Lukas die Gestaltung des Frontends übernommen hat. Im weiteren Projektverlauf ist auch Michael zur Frontendentwicklung übergegangen.

1.2.2 Betreuer

Die Diplomarbeit wurde seitens der Schule durch Fachkoordinator Prof. Mag. Michael Holzmann betreut. Durch seine Erfahrungen im Bereich Webentwicklung und API-Programmierung konnte er uns sowohl technisch als auch theoretisch bei der Umsetzung des Projekts unterstützen.

1.2.3 Auftraggeber

Auftraggeber für dieses Projekt ist die uni software plus GmbH mit Sitz in Perg. Sie entwickeln maßgeschneiderte Softwaresysteme und bieten Consulting in Bereichen wie Datenanalyse, künstliche Intelligenz und Simulation an, um komplexe betriebliche Probleme zu lösen. Vor allem hat sich die Firma auf Industrie- und Finanzanwendungen spezialisiert, was auch an der umgesetzten Themenstellung im Rahmen dieser Diplomarbeit zu entnehmen ist. Das vollständige Projekt wurde von einer Schweizer Bank in Auftrag gegeben.

1.2.4 Ansprechpartner

Seitens des Auftraggebers wurden wir von Dipl.-Ing. Simon Primetzhofer betreut. Von der Projektplanung über die Implementierung bis hin zur Verfassung dieser Diplomarbeit ist er für Fragen zur Verfügung gestanden. Darüber hinaus hat er uns als erfahrener Software Engineer vertiefte Einblicke in industrielle Entwicklungsprozesse ermöglicht.

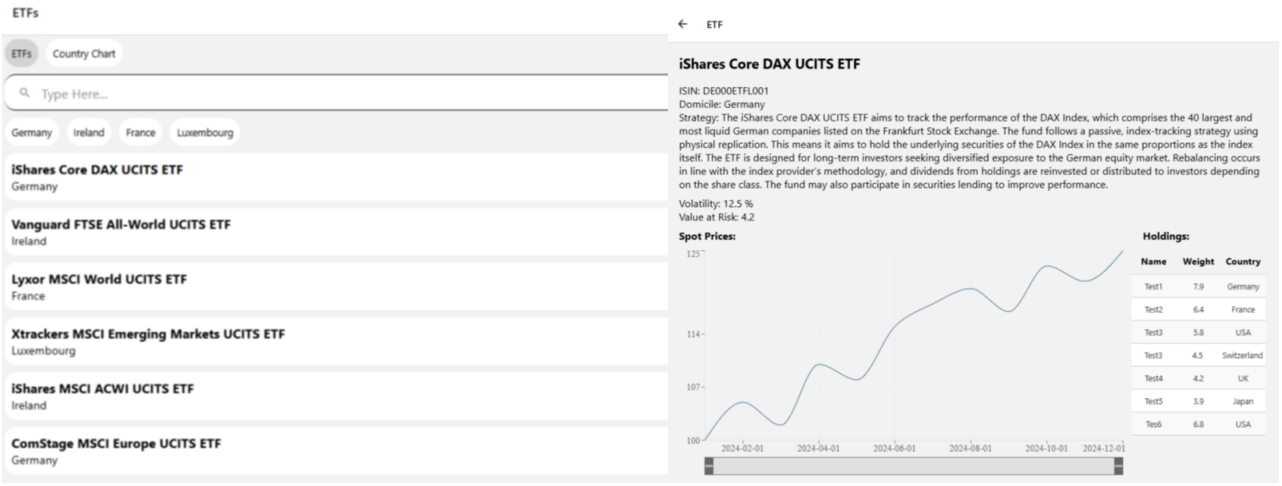
1.3 Ausgangssituation der Implementierung

Die Anforderungsanalyse und Softwarearchitektur ist vor der Implementierung bereits vorhanden gewesen. [1] Auf dieser aufbauend wurde, wie in 1.1 beschrieben, im Rahmen einer vorherigen Diplomarbeit die Grundstruktur der Datenbank festgelegt, Testdaten aus XML-Dateien eingelesen, ein Backendschnittstelle zur Datenbank implementiert und eine Frontendevaluierung

inklusive Implementierung eines Prototyps durchgeführt. [2] Folgend werden die bestehenden Komponenten näher beschrieben.

Angular Frontend Aufgrund der durchgeführten Evaluierung der Frontendtechnologien wurde Angular als Framework ausgewählt. Zwar wurden bereits einige grundlegende Funktionalitäten und Anzeigen implementiert, jedoch folgt es noch keinem vorgefertigtem, konsistenten Design.

Bereits vorhanden ist eine listenartige Darstellung von ETFs, inklusive einer Such- und Filterfunktion nach Ländern. In einer Detailansicht können dann genauere Informationen und ein Kursverlauf angezeigt werden.



(a) Listenansicht [2] (b) Detailansicht [2]

Abbildung 1: Screenshots des alten Prototyps

Datenquellen Die Anzeige der Daten im Frontend basiert auf drei Datenquellen: der Risk Service Schnittstelle (2.6.1), den XML-Dateien und einer PostgreSQL Datenbank. (3.1.1) Der Hauptlieferant für Daten ist die Risk Service Schnittstelle. Diese ermöglicht den Zugriff auf den Preisverlauf, Attribute und weitere Daten von ETFs. Weiters werden die Informationen über ETFs mithilfe einer vom Backend separaten Dataimporter Applikation, die auf XML-Dateien zugreift, ergänzt. Insgesamt sind fünf XML-Dateien vorhanden. Dadurch entstehen Testdaten, welche durch weitere SQL Skripts erweitert werden. Die hinzugefügten Informationen sind in einer PostgreSQL Datenbank persistiert. Das Datenbankschema ist modelliert und normalisiert. Alle Tabellen, um ETFs, deren Holdings, Gruppierungen von ETFs in Peer Groups und so weiter zu speichern, sind vorhanden.(2.1.2)

Java Spring Boot Backend Das Backend liefert dem Frontend eine einheitliche REST-Schnittstelle für alle Datenquellen. Der Code ist in Repository, Service und Controller unterteilt. (2.1.1) Das Repository ist die Schnittstelle zur PostgreSQL Datenbank und definiert die Datenbankoperationen. Der Service greift auf das Repository zu, kombiniert diese mit der Risk Service Schnittstelle und enthält die Geschäftslogik. Der Controller nimmt die Anfragen entgegen, wandelt die Parameter das passende Format um und ruft die Methoden des Service auf. Die Funktionalität für den Zugriff auf eine Liste von ETFs ist vorhanden. Weiters sind genauere Detailinformationen zu einem ETF verfügbar. Außerdem können mehrere ähnliche ETFs in einer Peer Group (2.1.2) abgefragt werden.

1.4 Projektinhalt

Aufbauend auf diesem Fundament wird das Frontend von Grund auf neu implementiert, die Funktionalitäten im Backend erweitert und weitere Testdaten in die Datenbank eingefügt.

Angular Frontend Damit das Projektteam eine benutzerfreundliche und stilistisch konsistente Oberfläche umsetzen kann, wurde vom Auftraggeber ein Design bei einer externen Firma in Auftrag gegeben. Die einzelnen Ansichten wurden auf der Plattform Figma bereitgestellt. Ebenso konnte auf eine klickbare Variante zurückgegriffen werden. Diese ist über den Browser erreichbar gewesen und hat zu einer leichteren Einarbeitungsphase im Projekt verholfen, da der Workflow dadurch leicht veranschaulicht werden konnte.

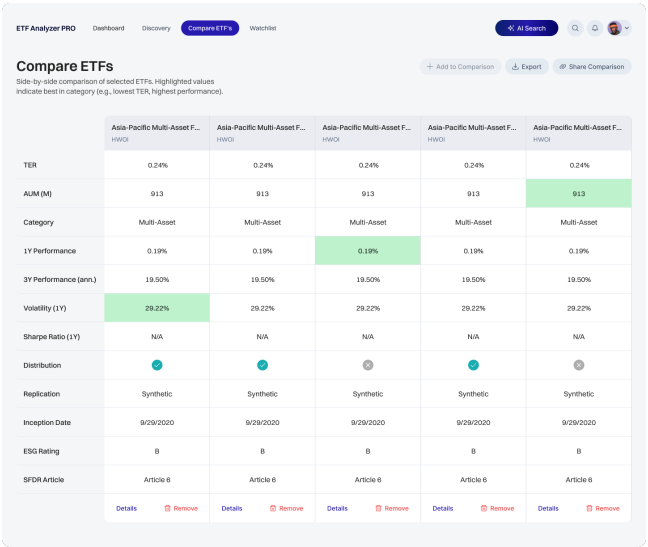


Abbildung 2: Figma Ansicht - Comparison Seite

Neben diesem Entwurf existiert auch eine weitere Designversion, mit der die Implementierungsarbeiten begonnen worden sind. Etwa zur Hälfte des vierwöchigen Praktikums wurde der aktuelle Figma-Entwurf fertiggestellt und geteilt, wodurch das Styling des bereits implementierten Codes angepasst werden musste.

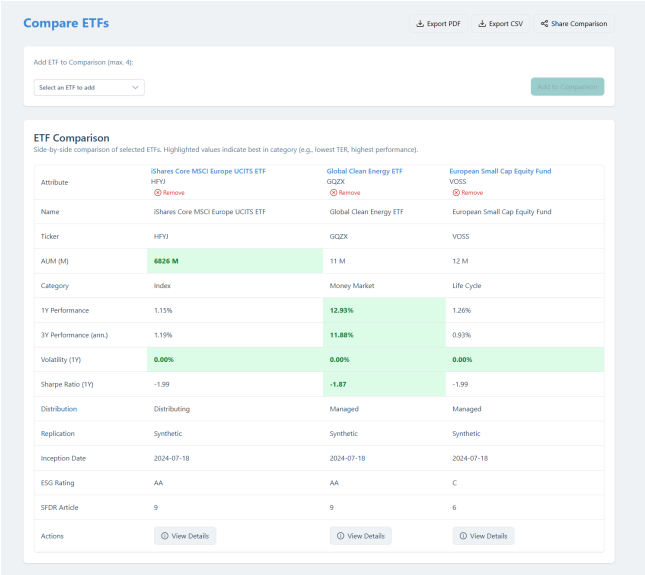


Abbildung 3: Alte Figma Ansicht - Comparison Seite

Java Spring Boot Backend Erweiterung Das Backend beinhaltet viele benötigte Funktionalitäten, wie den Listenzugriff auf ETFs, die Verwaltung von ähnlichen ETFs in einer Peer Group oder den Zugriff auf den Preisverlauf eines ETFs. Zusätzlich dazu müssen noch weitere Endpunkte implementiert werden, um die für das Frontend benötigten Informationen bereitstellen zu können. Dieses zeigt in der Entdeckungskomponente (3.2.6) mehrere ETFs in einer tabellarischen Liste an und auf der rechten Seite gibt es Filteroptionen. Dafür wird der Listenzugriff mit Pagination, Sortierung und Filterung (2.1.1) ausgestattet. Zusätzlich benötigt die Vergleichskomponente (3.2.7) Kennzahlen, wie zum Beispiel die maximum Drawdown, die noch nicht als Attribute, in der Ausgangssituation bereitgestellt werden. Außerdem wird für den Vergleich ein Ranking (3.4.3) aufgrund dieser Kennzahlen zur Verfügung gestellt.

Testdatenerweiterung Das Datenbankschema ist nicht vollständig, um alle Attribute des Figma Designs im Frontend bereitzustellen. Deswegen werden weitere Attribute zu ETFs hinzugefügt, die Bereiche, wie Handel, Risiko und akkumulierte Kennzahlen zu Rendite abdecken. Weiters werden für die Testung der Funktionalitäten ergänzende Testdaten generiert. Die Herausforderung dabei sind die verschiedenen Datenquellen (1.3), vor allem die Risk Service

Schnittstelle (2.6.1), aus denen die Testdaten stammen. Beispielsweise müssen valide ISINs (3.4.4) gefunden werden, weil die Risk Service Schnittstelle bei zufällig generierten keinen Preisverlauf findet.

2 Grundlagen und Methoden

2.1 Theoretische Grundlagen

Nachfolgend werden theoretischen Hintergründe und Methoden erläutert, die für diese Arbeit von Relevanz sind.

2.1.1 Spring Boot Backend

Controller-Service-Repository Muster Die Architektur im Backend besteht aus drei Schichten: Controller, Service und Repository (2.1.1) (siehe Abbildung 4). Das *Repository* ist für den Datenbankzugriff und das Speichern von Daten zuständig. Der *Service* implementiert die Geschäftslogik und greift auf das Repository zu. Der *Controller* stellt eine REST-Schnittstelle (2.1.1) zum Service zur Verfügung. *Models* sind die Domainenobjekte, die zwischen den Schichten verschickt werden. [3] [4]

Abbildung 4: Backend Architektur

Die Aufteilung in verschiedene Schichten vereinfacht die Applikation zu testen, weil Aufgaben voneinander isoliert werden. Wenn der Controller getestet werden soll, wird die Serviceschicht gemockt. Ein *Mock* ist ein Objekt oder ein Funktion, die stellvertretend für eine Implementierung während des Testingprozesses ist. [5] Dadurch kann kein Fehler in der Geschäftslogik die Anzeige verfälschen und Grenzfälle können durchprobiert werden. Wenn der Service getestet werden soll, wird die Repositoryschicht gemockt. Dadurch kann die Geschäftslogik unabhängig von der Datenbank getestet werden. [3] [4]

Data Transfer Objects Muster Das Ziel von DTO ist den Netzwerkverkehr zu minimieren. Wenn mehrere Models in eines zusammengefasst werden, reduziert das die Anfragen des Clients. Wenn unnötige Attribute weggelassen werden können, müssen weniger Daten übers Netzwerk verschickt werden. Die Modelattribute müssen den Attributen der DTOs zugeordnet werden.

Boilerplate Code ist Code, welcher mehrmals ohne große Variation wiederholt wird. [6] Die manuelle Zuordnung von Attributen erzeugt Boilerplate Code. Deswegen gibt es eigene Bibliotheken, welche diesen Boilerplate Code automatisch generieren können. Im Rahmen dieser Arbeit wurde die Bibliothek *MapStruct* (2.1.1) verwendet. [7]

MapStruct Ein *Compiler* wandelt Instruktionen in Maschinensprache um. [8] MapStruct generiert Boilerplate Code während des Codekompilierens. Deshalb ist die Bibliothek hochperformant. Außerdem werden Fehler direkt vom Compiler erkannt und treten nicht erst während der Laufzeit auf. MapStruct setzt auf die Strategie *Convention over Configuration*. Dabei werden sinnvolle Standardeinstellungen angenommen, damit Entwicklerinnen und Entwickler weniger konfigurieren müssen. [9] [10] [11]

Die Klasse *CarMapper*, welche zuständig ist Attribute zuzuordnen, muss mit der @Mapper Annotation versehen werden. Zuordnungsmethoden von einer Klasse (z.B. Car) zu einem DTO (z.B. CarDto) werden in der Schnittstelle nur deklariert und nicht implementiert. Jede Namensveränderung eines Attributs kann mit einer @Mapping Annotation angegeben werden. Dabei bezieht sich *source* auf die ursprünglichen Attributsnamen und *target* auf den gewünschten. Der Zugriff auf eine Instanz der Klasse CarMapper erfolgt über eine statische Variable, welche per Konvention *INSTANCE* genannt wird. Sie wird mithilfe von Dependency Injection (2.1.1) initialisiert.

Listing 1: Beispiel MapStruct

```
1 @Mapper
2 public interface CarMapper {
3
4     CarMapper INSTANCE = Mappers.getMapper( CarMapper.class );
5
6     @Mapping(source = "numberOfSeats", target = "seatCount")
7     CarDto carToCarDto(Car car);
8 }
```

[10]

Dependency Injection Muster *Dependency Injection* entkoppelt eine Klasse von ihren Abhängigkeiten, indem das Instanziieren an ein weiteres Objekt übergeben wird. Das *Dependency Inversion* Prinzip besagt, dass (a) Module auf höheren Ebenen nicht von Modulen auf niedrigeren Ebenen abhängen dürfen. Beide sollten von Abstraktionen abhängen. (b) Abstraktionen dürfen nicht von Details abhängen. Details sollten von Abstraktionen abhängen. [12] Abhängigkeiten werden durch Abstraktionen, wie zum Beispiel Schnittstellen, dargestellt.

Im MapStruct Beispiel (siehe Listing 1) ist CarMapper die Schnittstelle und die Klasse *Mappers* kümmert sich um die Instanziierung. Dies ist ein Beispiel für *Methodeninjektion*. Dabei wird die Abhängigkeit als Parameter an eine Methode übergeben. Im Gegensatz dazu werden bei der *Konstruktorinjektion* die Abhängigkeiten als Parameter im Konstruktor deklariert. Dadurch können Instanzen beim Testen leichter gemockt werden. Als dritte Möglichkeit gibt es *Feldinjektion*. Dabei werden Abhängigkeiten direkt in die Membervariablen der Klasse injiziert. [13] [14]

Spring IoC containers verwalten mehrere Java Beans. Beans werden in XML-Konfigurationsdateien definiert. Eine weitere Definitionsmöglichkeit ist die @Component Annotation. Davon abgeleitet sind @Repository, @Service und @Controller, die die stereotypische Rolle 2.1.1 in der Applikation genauer spezifizieren. [15] [16] [17]

Repository Das Repository nutzt die Funktionalitäten von Spring Data JDBC [18] für den Datenbankzugriff. Jede Schnittstelle erbt von einer zentralen *Repository-Schnittstelle*. Eine Objekt wird eindeutig durch den Typ der Klasse und durch eine Identifikationsmembervariable identifiziert. Die Typen beider müssen als Typenparameter an die Repository-Schnittstelle übergeben werden. CRUD sind die fundamentalen Datenbankoperationen: Schreiben, Lesen, Verändern und Löschen. Die *CrudRepository-Schnittstelle* generiert alle CRUD Operationen für ein Objekt. Dadurch wird Boilerplatecode vermieden. *Pagination* ist der Prozess, Inhalt in diskrete Seiten aufzuteilen. [19] Dadurch wird die Benutzerin oder der Benutzer nicht mit Informationen überschwemmt und die Leistung wird gesteigert, weil nur Teile der Daten geladen werden. [19] Die *PagingAndSortingRepository-Schnittstelle* fügt Pagination- und Sortierfunktionalitäten zu der CrudRepository-Schnittstelle hinzu. Für Pagination muss die Zahl gewünschten Seite und die Anzahl an Objekte pro Seite angegeben werden. Bei der Sortierung werden die Spaltennamen angegeben und ob es entweder aufsteigend oder absteigend sortiert werden soll. Neben den vordefinierten Funktionalitäten können die Schnittstellen erweitert werden. Eigene Abfragen werden mit der @Query Annotation definiert (siehe Listing 3). [20] Die Abfrage wird als String an die Annotation übergeben, während Parameter im String mit einem Doppelpunkt gekennzeichnet sind. Die @Param Annotation definiert den dazugehörigen Parameter in der Methodensignatur (siehe Listing 3). [21]

Listing 2: Beispiel Repository mit @Query Annotation

```
interface UserRepository extends CrudRepository<User, Long> {

    @Query("select * from User u where u.email = :email")
    User findByEmail(@Param("email") String email);

}
```

Für bestimmte Operationen [20] kann Spring Data JDBC die Abfrage ohne String-Parameter zusammenbauen. Somit produziert die folgende Schreibweise dasselbe Resultat, wie im obigen Codebeispiel (siehe Listing 3):

Listing 3: Beispiel Repository mit @Query Annotation

```
interface UserRepository extends CrudRepository<User, Long> {

    User findByEmail(String email);

}
```

[22]

REST *Representational State Transfer (REST)* ist ein Architekturstil, um Hypermedia-Inhalte zu übertragen. REST ist optimiert für Web Applikationen und setzt auf eine einheitliche Schnittstelle. Die Kommunikation erfolgt zwischen einem Client und Server. Eine Anfrage des Clients an den Server ist zustandslos und enthält alle Informationen, die zur Verarbeitung der Anfrage benötigt werden. Zusätzlich muss der Client angeben, ob die Anfrage zwischengespeichert werden darf. In Architekturen mit mehreren Schicht kann der Server mit weiteren Servern über REST kommunizieren. Der Client sieht immer nur die erste Schicht. Optional kann sich der Client auch Code dynamisch vom Server herunterladen. [23, p.76 - 85]

Zwischen Client und Server werden *Ressourcen* verschickt. Jede Information, die eindeutig identifizierbar ist, wie zum Beispiel ein Dokument, Video oder eine Sammlung anderer Ressourcen, kann eine Ressource sein. Die Identifikation heißt *Ressource Identifier*. Die Ressource wird durch eine *Representation* dargestellt. Diese besteht aus Daten in verschiedenen Formaten und Metadaten, die die Daten beschreiben. Das Datenformat wird als *MediaType* bezeichnet und ist zum Beispiel JSON. Zusätzlich wird *Control Data* mitgesendet, um die aufgeforderte Aktion des Servers zu beschreiben. [23, p.86 - 92]

REST verwendet *Connectoren*, um Aktionen zu kapseln. Diese Schnittstellen werden verwendet, damit die Implementierung ohne Änderung des Zugriffs ausgetauscht werden. Client, Server,

Cache, Resolver und Tunnel sind Connectoren. Der Client baut eine Verbindung auf und der Server lauscht, ob ein Client eine Verbindung aufbauen möchte. Ein *Cache* speichert Ergebnisse zwischen. Ein *Resolver* übersetzt einen Ressource Identifier in Netzwerkaddressen. DNS ist ein Resolver, der eine URI zu einer IP-Adresse auflöst. Ein *Tunnel* leitet die Netzwerkkommunikation weiter. [23, p.92 - 96]

REST *Components* sind die Softwarebestandteile, die über Connectoren miteinander kommunizieren. Der *User Agent* verwendet die Clientschnittstelle, um eine Verbindung aufzubauen. Der *Origin Server* erhält diese Anfrage über die Serverschnittstelle. Dazwischen können optional *Proxies* oder *Gateways* sein, die Anfragen weiterleiten. Der Unterschied zwischen beiden ist, dass sich der Client entscheiden kann, ob er einen Proxy verwendet oder nicht. [23, p.96]

Migration Wenn sich das Datenbankschema ändert, müssen die Daten vom alten System in das neue migriert werden. *Flyway* ist ein Werkzeug, das bei der Datenbankmigration hilft. Zuerst versucht Flyway Schemaverlaufstabelle *flyway_schema_history* zu finden. Existiert diese nicht, wird eine neue erstellt. Dann wird nach Migrationen gesucht. Eine *Migration* enthält alle Änderungen einer Datenbank von einer Version zur nächsten und wird durch zum Beispiel eine SQL Datei beschrieben. Wenn eine leere Datenbank vorliegt, werden alle Änderungen in Reihenfolge der Versionsnummer angewendet und in die flyway_schema_history Tabelle eingetragen. [24] [25]

2.1.2 ETFs

Begriff und Definition Exchange Traded Funds (ETFs) sind börsengehandelte Investmentfonds. Sie bündeln eine Gruppe von Wertpapieren in einem Fonds und können wie einzelne Aktien an der Börse gehandelt werden. ETFs bilden Aktienindizes, wie zum Beispiel den DAX (Deutscher Aktienindex), MSCI World oder S&P 500, nach und versuchen deren Kursentwicklung möglichst exakt zu replizieren. [26]

Ein Aktienindex ist ein Kennwert, der die Entwicklung einer bestimmten Gruppe von Aktien widerspiegelt. Er dient als Maßstab zur Beurteilung der allgemeinen Marktentwicklung eines Landes, einer Region oder eines gewissen Marktsegments. Der Wert eines solchen Index setzt sich aus der Summe von Aktienkursen multipliziert mit der Gewichtung der einzelnen Unternehmen zusammen und wird in Punkten dargestellt. Ein Aktienindex selbst ist jedoch kein handelbares Finanzprodukt, somit ist ein direkter Kauf, zum Beispiel des DAX, nicht möglich. Um dieselbe Wertentwicklung zu erzielen, müsste eine Anlegerin oder ein Anleger theoretisch alle 40 Aktien des DAX in den entsprechenden Gewichtungen erwerben und regelmäßig anpassen, wenn sich

die Zusammensetzung des Index ändert. Dies ist in der Realität nur mit großem Aufwand und Transaktionskosten realisierbar. [26]

ETFs bieten eine Lösung für dieses Problem, denn durch ihre automatische Nachbildung von Indizes können Anlegerinnen und Anleger einfach breitgefächerte Investitionen vornehmen. Ein DAX-ETF ermöglicht auf diese Weise effiziente Investitionen in die 40 größten deutschen Aktiengesellschaften, die an der Frankfurter Börse gehandelt werden. [26]

Funktionsweise und Struktur von ETFs Die zuvor beschriebene Nachbildung von Aktienindizes kann auf unterschiedliche Weise erfolgen. Bei der **physischen Replikation** kauft der Fonds die im Index enthaltenen Wertpapiere direkt. Dies kann entweder vollständig oder teilweise erfolgen. Bei der vollständigen Replikation werden alle Anlagen gekauft, während bei der teilweisen Replikation nur eine repräsentative Teilmenge physisch erworben wird. Die zweite Möglichkeit bildet die **synthetische Replikation**, hierbei wird der Index über sogenannte Swaps abgebildet. In diesem Fall geht der ETF-Anbieter ein Tauschgeschäft mit einem Partner (z.B. einer Bank) ein, welcher sich verpflichtet, dem Anbieter genau die Rendite 2.1.2 zu zahlen, welche der Index liefern würde. Als Gegenleistung bekommt der Partner eine Gebühr und die Rendite aus dem Sicherheitsportfolio des ETF-Anbieters. Diese Methode wird vor allem bei Anlagen in schwer zugänglichen oder regulierten Märkten eingesetzt. [27]

ETFs unterscheiden sich außerdem in der Behandlung von Erträgen. **Ausschüttende ETFs** zahlen Erträge regelmäßig an die Anlegerinnen und Anleger aus, während **thesaurierende ETFs** diese automatisch wieder in den Fonds reinvestieren. Durch diese Wiederanlage steigt der Wert des ETFs und damit auch der Wert der gehaltenen Anteile. [28]

Kennzahlen und Attribute von ETFs Für die Bewertung und den Vergleich von ETFs werden bestimmte Kennzahlen und Attribute herangezogen. Diese bilden auch die Grundlage für die in dieser Arbeit entwickelte ETF-Analyzer-App. Im Folgenden werden die wichtigsten Kennzahlen und Attribute von ETFs erläutert:

- **ISIN**
  Die ISIN ist eine international standardisierte Kennnummer, die jedes Wertpapier eindeutig identifiziert. [29]
- **Gesamtvermögen (AUM - Assets under Management)**
  Das verwaltete Gesamtvermögen beschreibt die Summe aller im Fonds befindlichen Vermögenswerte, ausgedrückt in der jeweiligen Fondswährung. [30]

- **Gesamtkostenquote (TER - Total Expense Ratio)**
  Die TER gibt die jährlich anfallenden Verwaltungs- und Betriebskosten des Fonds als Prozentsatz des durchschnittlichen Fondsvermögens an. [31]

- **Rendite (Return / Performance)**
  Die Rendite beschreibt den prozentuellen Zuwachs eines ETFs über einen bestimmten Zeitraum. Die häufigsten Analysen des Returns laufen über ein Jahr, allerdings auch Vergleiche bis hin zur Auflage des ETFs sind ebenfalls üblich. [32]

- **Volatilität und Sharpe Ratio**
  Die Volatilität misst die Schwankungsintensität der Fondsrenditen über einen bestimmten Zeitraum und dient als Indikator für das Marktrisiko. [33] Die Sharpe Ratio setzt die erzielte Rendite ins Verhältnis zum eingegangenen Risiko und ermöglicht damit eine Leistungsbewertung unter Einrechnung des Risikos. [34]

- **VAR - Value at Risk**
  Der Value at Risk schätzt den potenziellen maximalen Verlust eines Fonds innerhalb eines bestimmten Zeitraums. [32]

- **Fondsdomizil und Währung**
  Das Fondsdomizil bezeichnet den rechtlichen Sitz des Fonds und bestimmt die regulatorischen sowie steuerlichen Rahmenbedingungen. [35] Die Fondswährung wiederum beeinflusst das Währungsrisiko, insbesondere bei internationalen ETFs. [36]

- **Benchmark**
  Die Benchmark ist der Referenzindex, dessen Wertentwicklung der ETF nachbildet. [26]

- **Holdings**
  Unter Holdings versteht man die im Fonds enthaltenen Wertpapiere. [37]

- **Ticker-Symbol**
  Der Ticker ist ein meist alphanumerisches Kürzel, unter dem ein ETF an einer Börse gehandelt wird. Da ein ETF an mehreren Handelsplätzen notiert sein kann, existieren häufig mehrere Ticker für ein und dasselbe Produkt. [38]

- **Strategie**
  Die Strategie beschreibt den Ansatz, mit dem der ETF seinen Referenzindex abbildet oder ein bestimmtes Anlageziel verfolgt. Unterschieden wird unter anderem zwischen passiven oder regelbasierten Strategien. [39]

- **ESG-Rating**

  Das ESG-Rating bewertet ETFs anhand von Umwelt-, Sozial- und Governance-Kriterien. Es dient Investorinnen und Investoren als Orientierungshilfe für nachhaltige Geldanlagen. [40]

- **Peer Group**

  Die Peer Group umfasst ETFs mit vergleichbarer Anlagestrategie, gleichem Marktsegment oder ähnlicher Risikostruktur. Ein Vergleich innerhalb dieser Gruppe erlaubt eine relative Bewertung von Kosten, Rendite und Risiko. [41]

- **Auflegungsdatum (Inception Date)**

  Das Auflegungsdatum gibt an, seit wann ein ETF am Markt verfügbar ist. [42]

- **Emittent (Issuer)**

  Der Emittent ist die Fondsgesellschaft, die den ETF auflegt und verwaltet. Die Reputation und Erfahrung des Emittenten gelten als qualitative Faktoren bei der Produktauswahl. [43]

- **Maximum Drawdown**

  Der Maximum Drawdown misst den größten prozentualen Verlust eines ETFs zwischen einem Höchst- und einem darauffolgenden Tiefststand innerhalb eines definierten Zeitraums. Diese Kennzahl dient der Abschätzung extremer Verlustrisiken. [44]

- **Zeitformate und Betrachtungszeiträume**

  Zeitformate bzw. Betrachtungszeiträume definieren den Zeitraum, über den Kennzahlen berechnet werden. In dieser Arbeit werden Abkürzungen verwendet, M1 steht dabei beispielsweise für einen Monat, M3 für drei Monate.

Vorteile, Risiken und Relevanz von ETFs ETFs haben in den vergangenen Jahren, gemessen am Investitionsvolumen, stark an Bedeutung gewonnen und sind heutzutage nicht mehr vom Finanzmarkt wegzudenken. [45] Sowohl private als auch institutionelle Anlegerinnen und Anleger setzen vermehrt auf die Fonds. [46] Zu den Vorteilen zählen unter anderem die breite Diversifikation, da Anlegerinnen und Anleger durch den Kauf automatisch in eine Vielzahl von Unternehmen investieren. Dies reduziert das Risiko einzelner Fehlinvestitionen erheblich. Außerdem spielt die Transparenz eine große Rolle, denn als Investorin oder Investor ist es möglich zu jeder Zeit die aktuelle Zusammensetzung des Fonds einzusehen. Durch den Handel an der Börse ermöglicht dies schnelle Entscheidungen und Reaktionen. [47]

Allerdings sind auch ETFs nicht risikofrei, denn wie jedes börsengehandelte Wertpapier unterliegt es dem Marktrisiko. Fällt der zugrundeliegende Index, so sinkt auch der ETF in seinem Wert. Darüber hinaus können Tracking Errors, also Unterschiede in der Performance von ETF und Index entstehen. Dies tritt auf, sobald der ETF seinen Index nicht präzise nachbildet. Internationale ETFs bringen zudem ein Währungsrisiko mit sich, wenn sich Wechselkurse negativ auf die Rendite auswirken. [48] [49]

ETFs ermöglichen einen einfachen Einstieg in den Finanzmarkt, werden jedoch auch von erfahrenen Investorinnen und Investoren zunehmend in ihren Anlagestrategien integriert. Dieser stetig wachsende Stellenwert spiegelt die Relevanz der Fonds wider. [50]

2.1.3 Angular Frontend

DOM Das *Document Object Model* ist die baumartige Repräsentation der HTML-Struktur einer Webseite im Arbeitsspeicher des Browsers. Über standardisierte DOM-APIs können diese Objekte zur Laufzeit manipuliert werden, etwa durch das Hinzufügen oder Entfernen von Elementen oder das Verändern von Attributen. [51]

Direkte DOM-Manipulation wird in Angular vermieden, da sie die Change Detection umgeht. Falls ein DOM-Zugriff dennoch erforderlich ist, stellt Angular mit dem `Renderer2` eine Schnittstelle bereit. [52] [53]

Komponenten Angular ermöglicht es, eine Webapplikation in mehrere einzelne Bausteine zu unterteilen, die einfach wiederverwendet werden können. Diese sogenannten *Komponenten* setzen sich dabei aus folgenden drei Bestandteilen zusammen: [54]

- **Template (HTML)**
  Definiert die View, also den Aufbau der Benutzeroberfläche. Es beschreibt, welche Inhalte angezeigt werden. Mittels Angular-spezifischer Syntax kann man sich dynamisch auf Variablen der TS-Klasse koppeln und bei Arrays beispielsweise iterieren. [54]
- **Logic (TS)**
  Beinhaltet die Logik der Komponente. Hier werden Daten verarbeitet, Zustände verwaltet sowie Ereignisse (Events) behandelt. [54]
- **Style (SCSS)**
  Definiert die visuelle Gestaltung der Benutzeroberfläche. Mithilfe von SCSS können strukturierte und wiederverwendbare Styles erstellt werden, die Farben, Layouts und Abstände festlegen. [54]

Der `@Component`-Decorator in der TypeScript-Datei definiert eine Komponente. Neben einem Selector, also dem Namen, mit dem die Komponente später aufgerufen werden kann, werden Dateipfade zur HTML-Datei und zur Styling-Datei angegeben. Weiters können seit der Einführung von Standalone Components diese ihre Imports direkt innerhalb der Komponenten-Metadaten (Decorator) deklarieren. [55]

Die komponentenbasierte Struktur fördert neben der Wiederverwendbarkeit ebenso die Kapselung. Komponenten kommunizieren über klar definierte Schnittstellen mittels `@Input`- und `@Output`-Decorator. Dadurch entsteht eine hierarchische Struktur aus Parent- und Child-Komponenten. [56] [57]

Listing 4: Standalone Angular-Komponente mit @Component-Decorator

```
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-user-card',
  standalone: true,
  imports: [CommonModule, RouterModule],
  templateUrl: './user-card.component.html',
  styleUrls: ['./user-card.component.scss']
})
export class UserCardComponent {

}
```

Durch Angular-Lifecycle-Hooks können gezielte Handlungen an definierten Zeitpunkten im Lebenszyklus der Komponente ausgeführt werden. Ein Beispiel hierfür ist die Funktion `ngOnInit()`, welche bei der Initialisierung der Komponente aufgerufen wird. [58]

Services *Services* werden zur Auslagerung von Logik verwendet. Code, welcher nicht spezifisch zu einer Komponente gehört bzw. von mehreren verwendet wird, wird einmal implementiert und dann zur Verfügung gestellt. Dies wird über die Angular Dependency Injection (2.1.1) gelöst, denn durch den `@Injectable`-Decorator wird ein Service injizierbar. Das bedeutet Angular kümmert sich automatisch um die Instanzierung und Bereitstellung. Der Service kann so an beliebiger Stelle in anderen Komponenten verwendet werden, wobei er nicht mehrfach instanziert werden muss. [59] [60]

Typische Aufgaben von Services sind z.B. die API-Kommunikation, Datenpersistierung oder die Abbildung von übergeordneter Geschäftslogik. Die weitere Trennung zwischen Benutzeroberfläche und Logik fördert die Übersichtlichkeit, Testbarkeit und Wartbarkeit der Anwendung.

Routing Angular implementiert das Konzept der Single Page Application (SPA), bei dem die gesamte Anwendung innerhalb einer einzigen HTML-Seite läuft. Während eine klassische Webapplikation bei jeder Navigation zu einer anderen URL eine Anfrage an den Webserver schickt und die ganze Seite mit einer neuen ersetzt, so wird bei Angular lediglich der Inhalt dynamisch ausgetauscht. Eine Netzwerkanfrage muss nur beim ersten Aufruf der Seite gestellt werden. [61]

Die Navigation erfolgt durch den Angular Router, der auf Basis von definierten Routen die richtigen Komponenten einbindet. Die Routen werden zentral festgelegt und beinhalten immer den URL-Pfad mit der dazugehörigen Komponente. Zusätzlich kann auch ein Route Guard, wie `canActivate` verwendet werden, um Zugriffsbeschränkungen zu implementieren. [62] [61]

Skeleton *Skeleton* Screens dienen dazu, der Benutzerin oder dem Benutzer zu zeigen, dass eine Anwendung noch lädt. Sie zeigt in etwa wie die Seite im geladenen Zustand aussehen wird und dient als Platzhalter. Dies vermittelt den Endbenutzerinnen und Endbenutzern eine Illusion kürzerer Wartezeit und geben ihnen Zeit sich mental auf den Inhalt vorzubereiten. Es kommt nicht zu einer plötzlichen Reizüberflutung, wenn der Ladevorgang beendet ist. [63]

Abbildung 5: Exemplarische Implementierung eines Skeleton Screens

NGXS Statemanagement *NGXS* ist ein State-Management-Framework für Angular, das eine zentrale Verwaltung von Zuständen einer Anwendung ermöglicht. In komplexeren Anwendungen müssen Zustände, wie beispielsweise Filtereinstellungen, komponentenübergreifend gespeichert werden, damit diese auch nach Navigationsvorgängen erhalten bleiben und korrekte API-Abfragen durchgeführt werden können. Zur Umsetzung setzt NGXS auf drei Prinzipien. Alle Zustände liegen in einem zentralen Store, während Änderungen ausschließlich über definierte Actions erfolgen können. Neue Zustände werden unveränderlich erzeugt. Das bedeutet, dass ein neues Zustandsobjekt instanziiert wird, anstatt den bestehenden Zustand direkt zu verändern. [64]

Auch hier erfolgt die Definition über die Decorators. `@State` definiert einen State, `@Action` einen Action-Handler und `@Selector` einen Selector mit dessen Hilfe es möglich ist auf Teile des Zustands zuzugreifen. [65] [64]

Durch diesen strukturierten Ansatz verbessert NGXS die Wartbarkeit und Testbarkeit der Anwendung, da Zustandsänderungen nachvollziehbar und reproduzierbar sind. [66]

Signals, Observable Pattern Ein *Observable* repräsentiert eine Datenquelle, die Werte über Zeit liefert. Observer abonnieren diese Quelle und reagieren auf neue Daten. Dieses Push-Modell ermöglicht die Verarbeitung asynchroner Ereignisse wie HTTP-Anfragen oder Benutzereingaben. [67]

Mit Angular 16 wurden *Signals* eingeführt. Ein Signal speichert einen Wert und verfolgt automatisch Abhängigkeiten. Funktionen wie computed() erzeugen abgeleitete Werte, während effect() Aktionen ausführt, sobald sich abhängige Signale ändern. Sie stellen eine Alternative zum bisherigen Change-Detection-Mechanismus dar, der auf Zone.js basiert und Änderungen global auslöst. Ziel ist es, langfristig weniger von Zone.js abhängig zu sein und Updates gezielter sowie performanter durchzuführen. [68] [69]

Signals eignen sich primär für den lokalen, synchronen Zustand, während Observables für asynchrone Datenströme verwendet werden.

Ng Content Projection *Content Projection* ermöglicht es, Inhalte von außen in eine Komponente einzubetten. Das `<ng-content>` Element fungiert als Platzhalter und wird dann durch den tatsächlichen Inhalt ersetzt. Hat man mehrere Projektionsbereiche, so kann man diese mittels dem `select`-Attribut definieren. Dieses Konzept erhöht die Wiederverwendbarkeit von Komponenten. [70]

2.2 Technologien

Folgend werden alle Technologien, die das Projektteam zur Implementierung der Arbeit verwendet hat, gegliedert nach Anwendungsbereich erklärt.

2.2.1 Angular Frontend

NPM *npm*, ehemals *Node Package Manager* ist Bestandteil von NodeJS und ist verantwortlich für die Installation, Versionierung und Verwaltung von Bibliotheken. Gespeichert werden diese

Pakete in einer öffentlichen Datenbank, der sogenannten Registry, verwaltet. Die Webseite dient zur Suche von passenden Paketen und die Installation erfolgt über die Kommandozeile. [71]

Angular besteht aus vielen einzelnen Paketen, welche bei Bedarf installiert werden müssen. Die zentrale Konfigurationsdatei dieser Abhängigkeiten heißt `package.json` und definiert alle benötigten Paketnamen, einschließlich ihrer Versionsnummer, wobei diese entweder genau oder mittels des Semantic-Versioning-Standard beschrieben wird. Dieser Standard erlaubt die Angabe eines Versionsbereichs. Die Installation erfolgt in das Verzeichnis `node_modules` und die Informationen über die exakt heruntergeladenen Versionen werden im `package-lock.json` gespeichert. Möchte man auf einem anderen Gerät dieselben Versionen installieren und eine App so zum Laufen bringen, benötigt man lediglich diese Datei.

Abbildung 6: Logo NPM [72]

Angular *Angular* ist ein TypeScript-basiertes Framework zur Entwicklung von modernen Webanwendungen. Die Kombination aus Client-Side Rendering (CSR) und der Single Page Application Architektur sorgt für eine gute User Experience und weniger Serverlast, da sowohl die Erzeugung des HTML-Codes als auch der dynamische Austausch von Inhalten auf Seiten der Benutzerin oder des Benutzers erfolgen. Die modulare Aufteilung in Komponenten (2.1.3) erlaubt eine einfache Wiederverwendung von bereits implementierten Bausteinen der Benutzeroberfläche. Services (2.1.3) sind abgekapselte Teile der Logik, welche ebenso einfach an jeder benötigten Stelle eingebunden werden können. [73]

Um ein Angular Projekt aufzusetzen, muss die Angular CLI installiert werden. Dies funktioniert über npm und dem Befehl `npm i @angular/cli`. Danach erfolgen alle projektspezifischen Befehele über diese CLI. Dies reicht von der Erstellung des eigentlichen Projekts, der Generierung von Komponenten und Services bis hin zum Start der Applikation. [74] [75]

Entwickelt wird Angular von einem Team von Google, wobei die Code-Basis Open-Source ist und jedem zur Verfügung steht. [73] [76]

Abbildung 7: Logo Angular [77]

NodeJS *NodeJS* ist eine JavaScript-Runtime, welche die Voraussetzung zur Entwicklung von Angular bildet. Sie erlaubt es, JavaScript serverseitig auszuführen und wird beispielsweise für den Build-Prozess oder die Projektgenerierung verwendet. Standardmäßig läuft Node innerhalb eines Threads, anstatt jeder eingehenden Anfrage einen neuen zuzordnen. Operationen werden nicht-blockierend und asynchron ausgeführt, dadurch wird eine Verarbeitung vieler gleichzeitiger Requests möglich. [78]

Abbildung 8: Logo NodeJS [79]

NX Monorepo Ein *Monorepo* ist ein Repository, in dem mehrere Projekte oder Applikationen gemeinsam verwaltet werden. Diese Architektur verbessert Code-Sharing, Wartbarkeit und Zusammenarbeit zwischen Teams, da die verschiedenen Teile logisch getrennt, jedoch organisatorisch zusammenhängend sind. Diese Zusammenhänge werden über klare Abhängigkeiten geregelt. [80] [81] Beispielsweise können durch Monorepos mehrere Angular Applikationen und viele von mehreren Projekten verwendete Libraries in einem gemeinsamen Respository gebündelt werden. Beispiele für solche Libraries sind unter anderem visuelle Komponenten (2.1.3), als auch Services 2.1.3 oder APIs, welche den Zugriff auf Daten ermöglichen. Anhand eines Dependency Graphs kann man erkennen, welche Applikation welche Libraries nutzt, um Abhängigkeiten zu erkennen und gegebenenfalls zu entfernen.

NX ist ein Build-System für solche Monorepos und hilft dabei, die Entwicklung und Integration schnell und skalierbar zu halten. Dies wird durch verschiedene Mechanismen sichergestalt. Dazu zählen unter anderem Cachching, sodass kein Code öfter gebaut wird oder intelligente Aufgabenverteilung. NX kann mittels npm (2.2.1) installiert und auch bereits bestehende Projekte können migriert werden. [82] [83]

Abbildung 9: Logo NX [84]

Yarn *Yarn* ist eine Alternative zum zuvor beschriebenen npm (2.2.1) und dient ebenfalls der Installation, Versionierung und Verwaltung von Abhängigkeiten. Im Grunde funktioniert Yarn

ähnlich wie npm. Im `package.json` werden die benötigten Pakete definiert, im `node_modules`-Verzeichnis installiert und die tatsächlich installierten Versionen in der `yarn.lock`-Datei festgehalten. Yarn kann mittels npm und dem Befehl `npm i yarn` installiert werden. [85]

Yarn und npm sind funktional vergleichbar, unterscheiden sich aber in ihrer Implementierung und Anwendungsbereichen. Ersteren hat man im Jahr 2016 entwickelt, um damalige Schwachstellen von npm zu verbessern. Aus diesem Grund hat Yarn einen größeren Fokus auf Sicherheitsmechanismen, effizienter Installation durch Parallelisierung und gutes Offline-Chaching. Weiters hat sich Yarn mit seinem Workspaces-Konzept bereits früh auf die Monorepo-Architektur spezialisiert und diese nativ unterstützt. Dadurch können mehrere Packages in einem Repo für spezifische Teilprojekte organisiert werden und eine trotz alle dem ein eine zentrale Installation der Abhängigkeiten geboten werden. [86] [87]

Abbildung 10: Logo Yarn [88]

SCSS *SCSS* ist eine einfache Erweiterung von CSS. Sie ermöglicht die Gestaltung von Webapplikationen und erweitert den Funktionsumfang durch Verschachtelungen, der Möglichkeit Variablen zu definieren und sogenannten Mixins. Bei letzterem handelt es sich um wiederverwendbare Code-Segmente, die einmal definiert werden und an beliebiger Stelle verwendet werden können. All diese Faktoren erleichtert Entwicklerinnen und Entwicklern ihre Arbeit, da eine bessere Lesbarkeit und Struktur innerhalb ihres Stylings ermöglicht wird. Vor dem Build wird SCSS zu normalem CSS kompiliert. [89]
Beide Varianten können bei der Erstellung einer Applikation als Gestaltungsoption ausgewählt werden.

Abbildung 11: Logo SCSS [90]

Tailwind CSS *Tailwind CSS* ist ein Utility-First-CSS-Framework, das einfache CSS-Klassen zur Verfügung stellt, die direkt im HTML-Template eingebunden werden können. Auf diese Weise können vordefinierte Bausteine, wie zum Beispiel `p-4`, verwendet werden, um Padding

zu erzeugen. Im Gegensatz zu klassischem CSS müssen Entwicklerinnen und Entwickler also keine eigenen Styles für jede Komponente schreiben. Stattdessen verwenden sie vorhandene Utility-Klassen, was die Entwicklung deutlich beschleunigt. [91]

Technisch funktioniert Tailwind CSS mittels eines Just-In-Time (JIT) Compiler, der die HTML-Dateien durchscannt und nur tatsächlich genutzte CSS-Klassen in die finale Styling-Datei übernimmt. [92] Die Konfiguration erfolgt über die globale CSS-Datei. [91]

Abbildung 12: Logo Tailwind CSS [93]

2.2.2 Spring Boot Backend

Spring Boot Ein *Framework* ist eine organisierte Menge an Werkzeugen und Bibliotheken. [94] *Spring* (siehe Logo in Abbildung 13) ist das bekannteste Java Framework. [95] *Spring Boot* vereinfacht den Prozess eine eigenständige Applikation mit Spring zu entwickeln. Spring Boot setzt auf das Controller-Service-Repository Muster (2.1.1)

Abbildung 13: Logo Spring Boot [96]

2.2.3 PostgreSQL Datenbank

Datenbank *PostgreSQL* ist ein Open Source Object-Relational-DBMS. *Open-Source-Software* ist Software, die jeder inspizieren, verändern und verbessern kann. [97] Ein *DMBS* ist für die Erstellung und Verwaltung von Datenbank zuständig. [98] In dem *relationalen* Modell speichern Datenbanken Informationen in Form von Tabellen. Diese stehen miteinander in Beziehung - in Relation. [99] PostgresSQL unterstützt auch objektorientierte Ansätze, wie benutzerdefinierte Datentypen oder Funktionen. [100]

Funktionen Die Funktionalität einer PostgreSQL Datenbank kann durch Funktionen erweitert werden. [101]. Eine Funktion führt ein oder mehrere SQL Statements aus und returniert einen Rückgabewert. Ohne Rückgabewert wird sie Stored Procedure genannt. Dabei werden vier verschiedene Typen [102] unterschieden: *Query Language Functions* führen eine Liste von SQL

Statements aus und returnieren das Ergebnis der letzten Query, *Internal Functions* und *C-Language Functions* sind beide in C geschrieben [103] [104]. Internal Functions werden statisch gelinked und C-Language Functions werden in dynamisch ladbare Objekte kompiliert. *Procedural Language Funktionen* [105] ist ein Überbegriff für alle Funktionen, die in anderen Sprachen als SQL und C geschrieben wurden. Die Basisdistribution enthält zum Beispiel die Sprache PL/pgSQL [105].

Vorteil von Funktionen Funktionen werden direkt am Datenbankserver ausgeführt. Dadurch muss eine Client-Applikation nur einen SQL Aufruf machen anstatt mehrerer, was den Netzwerkoverhead reduziert. Außerdem können Funktionen vorkompiliert werden und dadurch erhöht sich die Performance. Weiters ist die Logik zentral organisiert. Das erhöht die Wiederverwendbarkeit und verringert Redundanten Code in verschiedenen Applikationen [106] [107]

2.3 Entwicklungssysteme

Die Umsetzung einer solchen Arbeit benötigt verschiedenste Systeme zur Entwicklung, Ausführung oder dem Test. Nachfolgend werden die verwendeten Entwicklungssysteme genauer erläutert.

2.3.1 Webstorm und IntelliJ

Webstorm (siehe Logo in Abbildung 14a) und IntelliJ (siehe Logo in Abbildung 14b) sind beides Entwicklungsumgebungen von Jetbrains. Webstorm konzentriert sich auf die Programmiersprachen JavaScript und TypeScript und Intellij auf Java und Kotlin. Die Entwicklungsumgebungen unterstützen durch Codevervollständigung, -qualitätsanalyse und Anzeige von Dokumentation beim Programmieren. Außerdem helfen sie bei der Codeumstrukturierung, indem zum Beispiel beim Umbenennen automatisch Referenzen ändert. Nicht nur die Editierung von Code, sondern auch die Navigation wird erleichtert. Die IDEs zeigt die Projektstruktur übersichtlich und erlaubt Textsuche überall oder zum Beispiel nur beschränkt auf Dateien. Weiters erlauben Webstorm und IntelliJ das Auffinden aller Verwendungen von zum Beispiel einer Methode und umgekehrt, von einer Verwendung zu der Definition zu springen. Ein Terminal ist direkt in die IDEs integriert, genauso wie eine graphische Benutzeroberfläche, um Versionsmanagement mit Git (2.3.3) zu verwalten. [108] [109]

(a) Logo Webstorm [110]

(b) Logo IntelliJ [111]

Abbildung 14: Jetbrains Logos

2.3.2 Figma

Figma (siehe Logo in Abbildung 15) ist ein Werkzeug um Designs zu erstellen. Zu Beginn kann entweder mit einem leeren Design gestartet oder ein Template verwendet werden. Danach erlaubt des Designwerkzeug die Editierung von Farben, Formen, Text und vielem mehr. Gemeinsam können mehrere Teammitglieder gleichzeitig an einem Produkt arbeiten. Diese Designs werden zum Beispiel für Webseiten, Social Media oder Mobilapplikationen verwendet. Weiters wird die Umwandlung von Designs in Code unterstützt und Figma stellt sogar eigene MCPs zur Verfügung, welche es künstlicher Intelligenz erlaubt mit Figma Designs zu arbeiten. [112]

Abbildung 15: Logo Figma [113]

2.3.3 Git

Versionsverwaltungen speichern Änderungen einer Datei über einen Zeitraum. Git (siehe Logo in Abbildung 16) ist eine Versionsverwaltung und kann über eine Kommandozeile oder verschiedenste grafische Benutzeroberflächen gesteuert werden. Mit diesem Werkzeug können Veränderungen ausgewählter Dateien zurückgesetzt und über Zeit miteinander verglichen werden. Außerdem kann genau festgestellt werden, wer etwas verändert hat. Git ist eine verteilte Versionsverwaltung. Deshalb wird der gesamte Veränderungsverlauf lokal und auf dem Server gespeichert. Dadurch wird ein vollständiges Backup erzeugt, was die Ausfallsicherheit erhöht. Im Gegensatz dazu speichern zentralisierte Versionsverwaltungen Veränderungen nur am Server. [114]

Ein *Code Repository* ist ein Speicherort für Quellcode und andere Softwareentwicklungsressourcen, welches von einer Versionsverwaltung gesteuert wird. Vollständiges Kopieren und hoch- und herunterladen von Änderungen ermöglicht die gemeinsame Verwaltung eines Code Repositories. Beim Zusammenführen mehrere Code Repositories können Konflikte entstehen, wenn mehrere Entwicklerinnen und Entwickler an derselben Datei Änderungen vorgenommen haben. Dabei

müssen sich die Entwicklerinnen und Entwickler einigen, welche Änderungen übernommen und welche verworfen werden sollen. [115]

Abbildung 16: Logo Git [116]

2.3.4 Docker

Containerisierung ist der Prozess eine Applikation mit allen Abhängigkeiten zu isolieren. Dadurch kann Software in einer konsistenten Umgebung bereitgestellt werden. Ohne Containerisierung können Fehler auftreten, wenn die Umgebung gewechselt wird: zum Beispiel von einem Entwicklungslaptop zum Server. *Virtuelle Maschinene* abstrahieren das gesamte Betriebssystem und die darunterliegende Infrastruktur weg. Im Gegensatz dazu emuliert ein Container kein Betriebssystem, sondern greift auf denselben Kernel zu. Somit ist ein Container viel ressourcensparender. Docker (siehe Logo in Abbildung 17) ist eine Platform für die Containerisierung von Applikationen. [117] [118]

Abbildung 17: Logo Docker [119]

2.3.5 Chrome DevTools

Chrome DevTools (siehe Logo in Abbildung 18) ist eine Sammlung von Werkzeugen, die direkt in Chrome (2.4.1) integriert ist. Die Werkzeuge sind in verschiedene Register gruppiert. Das Register *Elemente* erlaubt die Anzeige und Veränderung des DOMs [120]. Im Register *Console* befindet sich eine integrierte Console, die Nachrichten anzeigt und beliebigen JavaScript Code ausführen kann. [121]. Quellcodedateien können im Register *Quellen* editiert werden. Das *Netzwerkregister* bietet Analysemöglichkeiten für Netzwerkverkehr und Ladezeiten. [122]. Leistungsindikatoren befinden sich im Register *Leistung*. [123] Weiters kann im *Lighthouseregister* eine Punktezahl für die Bereiche Leistung, Bedienungshilfen, Best Practices und SEO zugewiesen werden. [124] Im Register *App* können zum Beispiel Cookies oder lokale Speicher angezeigt werden. [125]Darüber hinaus gibt es noch ein Vielzahl weiterer Werkzeuge. [126] [127]

Abbildung 18: Logo Chrome Devtools [128]

2.4 Sonstige verwendete Software

Die Planung, Kommunikation und das Verfassen der Arbeit hat ebenso Technologien benötigt, die nachfolgend erklärt werden.

2.4.1 Chromium

Chromium ist ein Open-Source Browserprojekt, welches 2008 gestartet wurde und heute als Basis für diverse Browser fungiert. Neben Google Chrome zählen hierzu auch Microsoft Edge, Opera oder Brave. Diese konkreten Ausführungen implementieren dann wiederum ihre eigenen Funktionen.[129]

Abbildung 19: Logo Chromium [130]

2.4.2 Clockify

Clockify ist ein webbasiertes Projektentwicklungstool zur Zeiterfassung. In einem gemeinsamen Workspace können Beteiligte ihren Arbeitsaufwand eintragen und bestimmten Projekten und Aufgabenbereichen, wie zum Beispiel Frontend, Backend oder Testing zuordnen. Mithilfe dieser App ist es möglich den genauen Projektverlauf zu verfolgen und nachvollziehbarer zu gestalten. [131]

Abbildung 20: Logo Clockify [132]

2.4.3 Discord

Discord ist ein Kommunikationstool mit Möglichkeiten sich sowohl in Sprach- und Videochats, als auch rein textuell zu verständigen. Durch das Erstellen einzelner Server oder Chatgruppen lassen sich Themen, also zum Beispiel verschiedener Projekte, gut voneinander abkapseln und der Zugriff von Usern leicht beschränken. [133] [134]

Abbildung 21: Logo Discord [135]

2.4.4 Rocket.Chat

Rocket.Chat ist ein Kommunikationstool, das grundsätzlich die gleichen Funktionalitäten wie Discord (2.4.3) bietet, dabei jedoch einen klareren Fokus auf Unternehmen hat. Zusätzlich gibt es die Möglichkeit, das Programm auf eigenen Servern zu betreiben, was einen hohen Grad an Kontrolle und Sicherheit mit sich bringt. [136]

Abbildung 22: Logo Rocket.Chat [137]

2.4.5 Overleaf

Overleaf ist ein webbasierter Editor für LaTeX-Dokumente. Die automatische Versionsverwaltung erlaubt ein gleichzeitiges Arbeiten mehrerer Personen an einem Dokument. Eine integrierte PDF-Vorschau ermöglicht es der Benutzerin oder dem Benutzer aktuelle Änderungen am Text oder der Formatierung sofort einzusehen. [138]

Abbildung 23: Logo Overleaf [139]

2.5 Bibliotheken und Plugins

Bei der Implementierung wurden vor allem im Frontend einige externe Bibliotheken und Plugins verwendet, welche den Umsetzungsprozess vereinfachen und sonst oft aufwändige Funktionalitäten fertig zur Verfügung stellen. Nachfolgend wird sowohl Nutzen als auch Funktionsweise erklärt.

2.5.1 PrimeNG

PrimeNG ist der Ableger von PrimeTek für Angular-Applikationen. PrimeTek stellt vorgefertigte Open-Source-Komponenten zur Gestaltung von Benutzeroberflächen bereit. Neben Angular werden auch React, Vue.js und Java unterstützt. Der Einsatz dieser Bibliothek reduziert den Implementierungsaufwand und gewährleistet ein konsistentes visuelles Erscheinungsbild. [140] Zur Einbindung in das Projekt reicht die Installation über einen beliebigen Paketmanager und

der Eintrag in der `app.config.ts`-Datei. Es muss `providePrimeNG` in der Liste von Providern angegeben werden.

Verwendete Components Im Folgenden werden die in der Diplomarbeit genutzten Komponenten sowohl in ihrer Nutzung als auch in ihrer Funktionsweise genauer beschrieben.

AutoComplete *AutoComplete* ist ein Eingabefeld, welches automatisch während des Schreibens Vorschläge in Echtzeit liefert. Hierzu wird im Hintergrund eine Liste aller Möglichkeiten gehalten und gefiltert. Im Zuge der Diplomarbeit wurde die Komponente in der Suchleiste (3.2.9) eingesetzt, um ETFs anhand ihres Namens zu finden.

Zum Importieren muss `import AutoCompleteModule from 'primeng/autocomplete';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den `<p-autoComplete />`-Tag. [141]

Checkbox *Checkbox* verhält sich wie ein standardmäßiges HTML-Auswahlkästchen, bietet jedoch den Vorteil frei gestaltbar zu sein und ermöglicht daher ein konsistentes Erscheinungsbild mit der restlichen Applikation. Eingesetzt ist die Komponente in der Entdeckungskomponente (3.2.6) geworden, um ETFs in einer Liste auszuwählen.

Zum Importieren muss `import CheckboxModule from 'primeng/checkbox';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den `<p-checkbox />`-Tag. [142]

IconField Die *IconField*-Komponente umschließt ein Eingabefeld und ein Icon. Dieser sogenannte Wrapper kümmert sich um die korrekte visuelle Darstellung und Positionierung des Icons innerhalb der Eingabe. Dabei handelt es sich nicht um ein eigenständiges Anzeigeelement mit Logik, sondern um eine Hilfskomponente für das Layout. Das Einsatzgebiet liegt ebenfalls in der Suchleiste (3.2.9), um eine kleine Lupe anzuzeigen. Diese kennzeichnet der Benutzerin oder dem Benutzer, dass es sich um ein Suchfeld handelt.

Zum Importieren muss `import IconFieldModule from 'primeng/iconfield';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den `<p-iconfield />`-Tag. [143]

InputText Die *InputText*-Komponente erweitert das Standardeingabefeld um die Gestaltungsmöglichkeiten. Dies schafft wiederum ein konsistentes Gesamtbild der Applikation. Anwendung hat das Textfeld bei der Eingabe eines ETF-Namens gefunden.

Zum Importieren muss `import InputTextModule from 'primeng/inputtext';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<input />`-Tag. Jedoch wird darin `pInputText` als Attribut festgelegt. [144]

Select Der *Select* ist eine Komponente zur Auswahl eines Elements aus einer Liste von Optionen. Diese Liste muss im Hintergrund in der TS-Datei gespeichert werden. Durch die Nutzung kann beispielsweise eine Kategorie im Peer Group Ranking (3.2.8) ausgewählt werden.

Zum Importieren muss `import SelectModule from 'primeng/select';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-select />`-Tag. [145]

MultiSelect Mithilfe eines *MultiSelects* können mehrere Elemente aus einer Liste ausgewählt werden. Hierzu wird im Hintergrund nicht nur eine Sammlung aller möglichen Optionen geführt, sondern auch eine mit allen Selektierten. Die Komponente wird bei der Auswahl von ETFs für den gemeinsamen Vergleich verwendet. Eine konkrete Implementierung befindet sich auch in der Vergleichskomponente (3.2.7).

Zum Importieren muss `import MultiSelectModule from 'primeng/multiselect';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-multiselect />`-Tag. [146]

SelectButton Ein *SelectButton* bildet eine Menge von Optionen als Button-Gruppe ab. Diese ist visuell hervorgehoben und ermöglicht der Benutzerin oder dem Benutzer je nach Konfiguration eine Einfach- oder Mehrfachauswahl. In der Diplomarbeit wurde bei der Auswahl der Zeitformate für die Renditengrafkomponente (3.2.9) auf diese gesetzt.

Zum Importieren muss `import SelectButtonModule from 'primeng/selectbutton';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-selectbutton />`-Tag. [146]

Slider Durch den Einsatz der *Slider*-Komponente ist eine Auswahl eines numerischen Wertes oder Wertebereichs durch Ziehen eines Schiebereglers möglich. Durch eine Skala gewinnt die Benutzerin oder der Benutzer eine intuitive Werteanpassung und kann logische bzw. vorgesehene Wertebereiche besser abschätzen. Eingesetzt wurde der Regler bei der Filterung (3.2.6) der ETFs nach numerischen Kennzahlen.

Zum Importieren muss `import SliderModule from 'primeng/slider';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-slider />`-Tag. [147]

Button PrimeNG stellt ebenso eine *Button*-Komponente zur Verfügung, welche das Standardelement um die erweiterten Gestaltungsmöglichkeiten ergänzt. Besonders einfach ist die Einbindung von Icons über das `icon`-Attribut im HTML.

Zum Importieren muss `import ButtonModule from 'primeng/button';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-button />`-Tag. [148]

Paginator Mithilfe eines *Paginators* können große Datenmengen in mehrere Seiten unterteilt werden. Weiters ermöglicht er die Navigation zwischen den Seiten. Das System wurde in der Entdeckungskomponente (3.2.6) verwendet. Eine genauere Beschreibung findet sich im Kapitel Paginator (3.2.9).

Zum Importieren muss `import PaginatorModule from 'primeng/paginator';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-paginator />`-Tag. [149]

Divider Der *Divider* fügt eine horizontale oder vertikale Linie ein und dient zur visuellen Trennung von Inhalten. Dies verbessert die Struktur, Lesbarkeit und Orientierung der Webseite. In der Diplomarbeit fand dieses Element bei der Implementierung der Essenziellen Informationen auf der ETF Detailansichtskomponente (3.2.8).

Zum Importieren muss `import DividerModule from 'primeng/divider';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-divider />`-Tag. [150]

Popover Durch ein *Popover* können Inhalte in einem Overlay angezeigt werden. Dieses wird über anderen Komponenten eingeblendet und steht somit immer im Vordergrund. In der Applikation ermöglicht das Popover die Sprachenauswahl (3.2.9).

Zum Importieren muss `import PopoverModule from 'primeng/popover';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-popover op/>`-Tag. [151]

Breadcrumb Beim *Breadcrumb* handelt es sich um eine Komponente zur visuellen Darstellung der Navigationshierarchie innerhalb einer Webseite. Sie zeigt den aktuellen Standort der Benutzerin oder des Benutzers und ermöglicht das Zurückspringen auf übergeordnete Seiten. Eingesetzt wurde diese Anzeige bei der Implementierung der ETF Detailansichtskomponente (3.2.8), um ein Navigation auf die Entdeckungskomponente (3.2.6) zu ermöglichen.

Zum Importieren muss `import BreadcrumbModule from 'primeng/breadcrumb';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-breadcrumb />`-Tag. [152]

Charts PrimeNG *Charts* dienen zur grafischen Darstellung von Daten in Form von Diagrammen. Sie basieren auf der Open-Source-Bibliothek `Chart.js`. Durch die Komponente konnte im Angular-Projekt der Kursverlauf von ETFs leicht visualisiert werden. Die Implementierung befindet sich in der Renditengrafikkomponente (3.2.9).

Zum Importieren muss `import ChartModule from 'primeng/chart';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-chart />`-Tag. [153] [154]

Skeleton Die *Skeleton*-Komponente ermöglicht es Platzhalter während Ladeprozessen anzuzeigen (siehe 2.1.3). In der Diplomarbeit wurden Skeleton-Elemente auf jeder Seite verbaut, die aktiv Daten aus dem Backend abfragt. Denn hier kann es bei schlechter Performance zu Wartezeiten kommen.

Zum Importieren muss `import SkeletonModule from 'primeng/skeleton';` in der TS-Datei angegeben werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-skeleton />`-Tag. [155]

Toast Durch ein *Toast*-Element können kurzzeitige Systemnachrichten in einem zusätzlichen Overlay angezeigt werde. Die Information erscheint meist am Rand und hält eine Benutzerin oder einen Benutzer über den Status oder Ergebnis einer Aktion informiert. In der Applikation findet der Toast beim Hinzufügen von ETFs in den Comparison Basket (siehe zum Beispiel bei der Vergleichskomponente (3.2.7)) einzug und liefert Informationen über den Erfolg.

Zum Importieren muss `import ToastModule from 'primeng/toast';` in der TS-Datei angegeben und der `MessageService` injected werden. Im HTML-Template erfolgt die Einbindung über den normalen `<p-toast />`-Tag. [156]

Icons PrimeNG verwendet standardmäßig *PrimeIcons* als Bibliothek für Icons. Es wird eine einfache Integration über CSS-Klassen ermöglicht. Nach der Installation reicht es, bei einem HTML-Element `class="pi pi-<Iconname>"` anzugeben, und das gewünschte Symbol wird angezeigt. [157]

PrimeIcons ermöglichen eine schnelle und konsistente Einbindung von standardisierten UI-Symbolen. Im Vergleich werden externe `.svg`-Icons vor allem bei individuellen Firmendesigns,

wie z.B. Logos, eingesetzt. In diesen Fällen sind oft präzise und mehrfarbige Formen und Muster gefordert. [158]

Preset PrimeNG verwendet ein *Preset-Konzept* zur Gestaltung der Komponenten. Dabei werden die wichtigsten Designeinstellungen zentral in der Angular-Anwendung definiert. Es gibt Standardpresets, die optional erweitert oder überschrieben werden können. Somit lassen sich auch projektspezifische Designanforderungen umsetzen. Aktuell gibt es vier vorgefertigte Presets, darunter Aura, Material, Lara und Nora. [159] [160]

Die Konfiguration eines Presets erfolgt in einem dreistufigen Token-System. Primitive Tokens definieren grundlegende Designwerte wie Farben oder Abstände, Semantic Tokens weisen diesen Werten eine funktionale Bedeutung im Benutzerinterface zu (z. B. Primärfarbe oder Hintergrund), und Component Tokens legen schließlich das konkrete Styling einzelner Komponenten fest. [160]

2.5.2 NX Console

Die *NX Console* ist eine installierbare Erweiterung für Entwicklungsumgebungen wie z.B. WebStorm (2.3.1). In Projekten, welche die NX Monorepo-Struktur (2.2.1) verfolgen, ist die Konsole besonders relevant, da die NX-Workflows damit abgebildet werden können. [161]

Zu den NX-Workflows gehören beispielsweise die Anzeige des Abhängigkeitsgraphen oder das Generieren neuer Libraries. Vor allem für letzteres wird es am häufigsten genutzt. Die Konsole stellt hierfür eine eigene Generate UI zur Verfügung, in der die neue Komponente komplett konfiguriert werden kann. Dadurch lässt sich eine konsistente Library-Struktur aufbauen. [161] [162]

2.5.3 ngx-translate

Die *ngx-translate*-Bibliothek ist eine weit verbreitete Möglichkeit, eine Angular-Applikation in mehreren Sprachen bereitzustellen. Mit ihr erfolgt ein Sprachenwechsel dynamisch während der Laufzeit, und der Inhalt der Webseite wird automatisch ausgetauscht. [163]

Zugrunde liegen hierfür sogenannte *Translation Files* im `.json`-Format. In solch einer Datei werden die Übersetzungen mittels Schlüssel-Wert-Paaren festgelegt. So kann beispielsweise für den Schlüssel *"yes"* die Übersetzung *"ja"* festgelegt werden. Die Dateien können weiter verschachtelt werden, so ist es beispielsweise üblich, nach Komponenten oder einzelnen, kleineren

Bausteinen der Benutzeroberfläche zu gruppieren. Abgelegt werden diese unter dem Verzeichnis `/public/i18n`.
Die Übersetzung übernimmt die `translate`-Pipe oder der `TranslateService`, ebenso kann man mittels `TranslateService.onLangChange` auf Sprachwechsel reagieren. Im HTML-Template wird nun anstelle des hartkodierten Texts der Schlüssel mit Weiterleitung an die Pipe angegeben. Diese Weiterleitung erfolgt durch das `|` Zeichen. [164] [165] [166]

Hinsichtlich der Lade-Strategie wird zwischen statischem und dynamischem Laden unterschieden. Bei ersterer Variante werden alle Sprachdateien bereits bei der Initialisierung der Applikation geladen und stehen dann sofort zur Verfügung. Dies ist zwar die einfachste Methode, jedoch skaliert diese bei vielen Sprachen bzw. vielen Übersetzungen nicht besonders gut. Das dynamische Laden schafft Abhilfe. Es ermöglicht das Nachladen von Übersetzungen mittels des `TranslateHttpLoader`. Dies reduziert die initiale Ladegröße der Anwendung. [164]

2.5.4 Prettier, ESLint

Um eine einheitliche, saubere Codebasis zu erreichen, können in ein Projekt Formatierungs- und Linting-Werkzeuge integriert werden.

Ein weitverbreiteter Formatter ist *Prettier*. Es definiert klare Regeln für Einrückungen, Zeilenumbrüche, Anführungszeichen und weitere stilistische Aspekte des Programmcodes. Diese Standards werden in der Regel beim Speichern einer Datei überprüft und angewendet. Hierzu wird der Code genommen und von Grund auf neu ausgegeben. [167]

Ergänzend kann *ESLint* verwendet werden. Hierbei werden ebenso Regeln festgelegt, allerdings im Bezug auf die Qualität des Codes. Unbenutzte Variablen, unnötige Zuweisungen an Variablen oder ein zwingender Rückgabewert bei einer Methode sind mögliche Konventionen. Verstöße werden als Warnungen bzw. Fehler angezeigt. [168] [169]

Kurzgesagt ist Prettier für die Formatierung und Linter für das Finden von Fehlern verantwortlich. [170]

2.6 Verwendete Schnittstellen

Die Applikation verwendet im Hintergrund verschiedene vorgefertigte Schnittstellen, um mit dahinterliegenden Services zu kommunizieren.

2.6.1 Risk Service

Der *Risk Service* ist eine REST Schnitstelle (2.1.1), die Daten zu ETFs bereitstellt. Der Zugriff auf die Risk Service Schnittstelle ist in mehrere Java Schnittstellen unterteilt. Die HTTP Methoden sind in Schnittstellen mit der Endung -Api mit *Spring MVC* [171] Annotationen definiert. Davon abgeleitet sind Schnittstellen mit der Endung -ApiClient, welche mit @FeignClient Annotationen versehen sind, damit sich *OpenFeign* [172] um die Übersetzung des Java Codes in einen HTTP Client kümmert. Zum Beispiel stellt die SpotPriceApi Schnittstelle eine Methode zur Verfügung, um die Preisliste eines ETFs innerhalb eines Zeitraums zu bekommen.

2.6.2 Artifactory

Ein *Artifact* ist eine Zusammenstellung von Projektressourcen, wie zum Beispiel kompiliertem Programmcode, Bibliotheken und deren Abhängigkeiten oder andere Ressourcen, wie z.B. Bilder. [173] *Artifactory* ist eine Softwarelösung zur Verwaltung von Artifacts. Der Risk Service (2.6.1) ist zur Zeit der Implementierung auf Artifactory zur Verfügung gestellt worden. Danach wurde auf Gitlab umgestiegen.

3 Implementierung

3.1 Technischer Überblick

Die Implementierung des *ETF Analyzer Pro* besteht aus vielen Einzelkomponenten, die zusammen das große Ganze bilden und die Applikation verwendbar machen. Wie bereits in den Technologien (2.2) erklärt, wurden bei der Umsetzung Angular im Frontend, Spring Boot im Backend und PostgreSQL als Datenbank eingesetzt.

Nachfolgend wird das Zusammenspiel der Teile genau erläutert.

3.1.1 Architektur

Damit die Benutzerin oder der Benutzer im Frontend eine Anzeige bekommt und mit der Applikation interagieren kann, müssen im Hintergrund viele Prozesse laufen und miteinander kommunizieren.

PostgreSQL - Datenbank Die Datenhaltung im System übernimmt eine PostgreSQL-Datenbank (2.2.3). In ihr werden die Stammdaten aller ETFs gespeichert und verwaltet. Es umfasst beispielsweise Attribute wie ISIN (2.1.2), Name oder Holdings (2.1.2). Ohne einer laufenden Datenbank hat die Anwendung keine ETFs zum Anzeigen.

Risk Service - Externe API Der Risk Service (2.6.1) ist eine zur Verfügung gestellte externe Schnittstelle um Daten zu ETFs abzurufen. Im Kontext dieser Arbeit wurde sie verwendet um die Spot-Prices, also die Preise von ETFs zu bestimmten Zeitpunkten, zu beziehen. Der Service ist der einzige Teil auf den das Projektteam keinen Einfluss hat und darauf vertrauen muss, dass dieser rund um die Uhr erreichbar ist. Sollte dies nicht der Fall sein, so können die Kursverläufe, Informationen zu Preisen und dynamisch berechnete Performance-Kennzahlen nicht dargestellt werden.

Spring Boot - Backend Das Spring Boot Backend (2.2.2) führt die Daten aus der Datenbank und dem Risk Service zusammen und bildet eine zentrale Schnittstelle für das Frontend, um Daten abrufen und Aktionen darauf ausführen zu können. Neben den Operationen auf der

Datenbank werden weitere Endpunkte angeboten, um vor allem rechenintensive bzw. komplexe Berechnungen im Frontend zu vermeiden. Ohne ein lauffähiges Backend gibt es keine Anbindung an die Daten und das Frontend wäre leer.

Frontend - Angular Das Angular Frontend (2.2.1) ist die Komponente, mit der Anwenderinnen und Anwender direkt interagieren. Sie muss benutzerfreundlich und ansprechend implementiert sein und stabil laufen, da dies entscheidend für eine effiziente Nutzung, hohe Akzeptanz und einen zuverlässigen Betrieb der Anwendung ist.

Optional: Spring Boot - Data-Importer Der Data-Importer wird verwendet, um neue ETFs aus einer XML-Datei zu importieren und in der Datenbank zu persistieren. Damit muss dieses in Spring Boot (2.2.2) implementierte Tool nicht zwingend im Hintergrund arbeiten, damit die Applikation korrekt läuft. Der Einsatz beschränkt sich auf das Ergänzen von ETFs oder das initiale Laden bei der Neuerstellung der Datenbank.

Während des Implementierungsprozesses hat das Projektteam die Entwicklung lokal durchgeführt und neue Funktionen regelmäßig auf ein Git-Repository (2.3.3) geladen. Die Datenbank ist hierbei in einem Docker-Container (2.3.4) gelaufen.

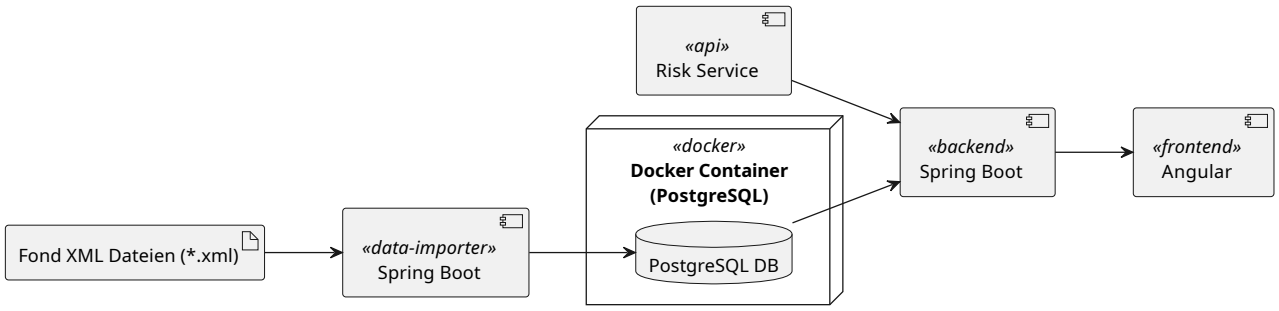


Abbildung 24: Aufbau der Architektur

3.1.2 Projektstruktur: Angular Frontend

Der Aufbau der Angular-Applikation unterscheidet sich durch den Einsatz der Nx Monorepo Architektur (2.2.1) grundlegend von einem standardmäßigen Projekt. Im Folgenden wird die Verzeichnisstruktur vereinfacht dargestellt.

```
etf-app-frontend/
├── apps/
│   ├── etf-app-frontend/
│   └── etf-app-frontend-e2e/
└── libs/
```

```
├── feature-.../
└── shared/
    ├── data-access/
    ├── styles/
    ├── types/
    └── ui/
```

Die zwei wichtigsten Verzeichnisse sind `apps` und `libs`. Im ersteren befindet sich das eigentliche Angular-Projekt, das Komponenten einbindet und ein dazu passendes `e2e`-Verzeichnis enthält, um sogenanntes End-to-End-Testing durchzuführen. Sie sorgen dafür, dass eine Applikation von Anfang bis Ende korrekt funktioniert und imitieren dabei die Art und Weise, wie eine Benutzerin oder ein Benutzer mit dieser umgehen würde [174]. Die Tests hat das Projektteam aufgrund von Zeitmangel nicht umgesetzt.

Im `libs`-Verzeichnis erfolgt die tatsächliche Implementierung. Der Code wird hierbei in in sich geschlossene Bausteine aufgeteilt und kann auch ganze Unterseiten enthalten . Das Namensschema ist im Projektteam auf *`feature`-Name der Komponente* festgelegt. Werden Teile des Codes öfter benötigt, so liegen diese im `shared`-Verzeichniss. Dies umfasst sowohl datenverarbeitende Schritte, wie zum Beispiel das NGXS Statemanagement (2.1.3), Models oder Angular Services (2.1.3), im Unterordner `data-access`, globale Gestaltungsrichtlinien unter `styles`, Typescript-Typen in `types` oder Komponenten der Benutzeroberfläche in `ui`. Die Besonderheit ist, dass diese UI-Bestandteile keine Logik implementieren, sondern nur für die Anzeige gedacht sind. Die Daten werden mittels `@Input` und `@Output` zwischen den Komponenten ausgetauscht (siehe Angular Komponenten 2.1.3).

3.1.3 Projektstruktur: Spring Boot Backend und Dataimporter

Sowohl das Backend als auch der Dataimporter sind im Java Spring Boot Framework (2.2.2) implementiert. Das Backend stellt eine klassische REST-API (2.1.1) zur Verfügung, während der Dataimporter zum Importieren von ETFs aus XML-Dateien dient.

Backend

Im Backend wird ein klassischer Projektaufbau verfolgt. Zur besseren Strukturierung und Wiederfindbarkeit von Code ist dieser in verschiedene logische Unterverzeichnisse aufgeteilt.

```
etfbackend/
├── configuration/
└── controller/
```

```
├── dto/
├── exceptions/
├── mapper/
├── repository/
├── services/
└── util/
```

Die Hauptlogik der API befindet sich in `controller`, denn hier werden alle Endpunkte definiert. Um die korrekten Daten zu liefern, greifen sie auf Services (unter `services`) zu, die wiederum auf Repositories (unter `repository`) zugreifen. Diese Abstraktion dient dazu, die tatsächliche Abfrage aus der Datenbank, zusätzliche verarbeitende Schritte wie Formatierung und die Beantwortung der Anfragen voneinander zu trennen. Das Backend folgt damit einer klassischen dreischichtigen Architektur (2.1.1), bestehend aus Controller-, Service- und Repository-Schicht. Dies hält die eigentlichen Endpunkte schlank und ermöglicht eine einfachere Einarbeitung in den Code.
In den verbleibenden Verzeichnissen befinden sich allerdings ebenso wichtige Komponenten, ohne die das Backend nicht funktionieren würde. In `configuration` werden Sicherheitseinstellungen festgelegt, beispielsweise, wer Anfragen senden darf. Die sogenannten DTOs (siehe 2.1.1) sind Objekte, die an die Benutzerin und den Benutzer gesendet werden und darauf ausgelegt sind, den Datenverkehr zu minimieren. Dies gelingt durch das zusammenfassen mehrerer Modelle in ein Objekt. Diese Klassen liegen unter `dto`. Zur Transformation zwischen der Datenbankabbildung und den DTOs wird im Verzeichnis `mapper` ein Mapper implementiert.
Abschließend befinden sich im Unterverzeichnis `exceptions` einige individuell implementierte Fehlerklassen, welche im Fehlerfall genauere Beschreibungen liefern. Das `util`-Paket implementiert lediglich eine Klasse zur zufälligen String-Generierung, um Testdaten zu simulieren.

Dataimporter Der Importer ist zur besseren Übersichtlichkeit ebenfalls in Pakete unterteilt und folgendermaßen aufgebaut.

```
dataimporter/
├── common/
├── exceptions/
├── model/
├── repo/
└── services/
```

Im Unterverzeichnis `common` befindet sich ein Mapper, der zur automatisierten Transformation der eingelesenen XML-Datenmodelle in entsprechende Java-Objekte dient. Um im Fehlerfall aussagekräftigere Rückmeldungen an die Benutzerin oder den Benutzer zu geben, wird unter

`exception` eine individuelle Exception definiert, welche den Fehler im Importprozess beschreibt. Im `model`-Verzeichnis werden die Java-Klassen definiert, welche die Daten aus dem importierten XML-Dokument repräsentieren.
Die schlussendliche Logik befindet sich unter `services`, wo die Daten eingelesen werden und der Aufruf zur Persistierung in der Datenbank erfolgt. Die eigentliche Speicherung mittels SQL-Queries erfolgt in den Repositories, die im Unterverzeichnis `repo` abgelegt sind.

Da sowohl das Backend als auch der Data-Importer auf gemeinsame Datenmodelle zugreifen, werden einheitliche Data-Access-Objects (DAOs) verwendet. Aus diesem Grund gibt es ein weiteres Projekt namens `commons`, das diese Klassen zentral verwaltet und den anderen Teilprojekten zur Verfügung stellt. Im Kontext dieser Arbeit stehen DAOs für die Abbildung der Datenstruktur einzelner Entitäten und repräsentieren somit die entsprechenden Tabellen der Datenbank. Diese Klassen bilden die Grundlage für Datenbankoperationen, da die Repositories damit arbeiten.

3.2 Angular Frontend

Das Frontend ist von Grund auf neu implementiert. Die einzelnen Bestandteile werden im Folgenden genauer veranschaulicht.

3.2.1 Workflow: Tailwindcss

Wie bereits im Projektinhalt (1.3) beschrieben, wurde dem Projektteam ein im Internet gehostetes Design zur Verfügung gestellt. Neben der leichteren Einarbeitung in den Kontext und den Workflow der Webseite hat sie uns ebenfalls bei der Umsetzung geholfen. Mithilfe der Entwicklertools des Browsers (2.3.5) ist es möglich, den DOM (2.1.3) anzuzeigen und jedes Element, einschließlich dessen CSS-Styling, einzeln zu betrachten. Durch diese Einsicht konnten viele Teile der Gestaltung übernommen und eine genaue Replikation des Designs gewährleistet werden.

Abbildung 25: Beispiel: DOM inklusive Styling

3.2.2 Layout Visualisierung

Jede Angular Applikation ist in Komponenten unterteilt. Die folgenden Abbildungen enthalten Screenshots von der Webseite. Dabei ist jede verwendete Angular Komponente eingerahmt in den Farben Blau, Rot und Grün. Die Farben entsprechen der hierachischen Ebene im HTML (2.1.3). Dafür wird eine CSS (2.1.3) Klasse *visualize* verwendet, welche eine CSS Variable für die Farbe und ein HTML Attribute für den Text verwendet. Wenn dieselbe Komponente mehrmals zum Beispiel in einer Liste vorkommt, dann sind die einzelnen Komponenten mit einer gestrichelten Linie umrahmt und die Liste ist eigens eingerahmt. In der rechten oberen Ecke befindet sich eine eindeutige Nummer für jede Komponente (zum Beispiel: Bereich 2.1). Unterbei ist eine Übersicht aller Komponenten und Routen der Webapplikation:

Tabelle 1: Übersicht der Routen und Komponenten der Webapplikation

Seite	**`Route`**	**Funktion**
Hauptseite (3.2.5)	`/`	Generelle Statistiken
Entdeckungsseite (3.2.6)	`/search`	Suche nach ETFs
Detailansichtsseite (3.2.8)	`/fund/{isin}`	Detailansicht eines spezifischen ETFs
Vergleichsseute (3.2.7)	`/compare`	Vergleich mehrere ETFs

3.2.3 Navigationsleiste

Jede Seite hat oberhalb die Navigationsleiste und unterhalb die Fußzeile (3.2.4). Dazwischen wird mithilfe von Angular Routing (2.1.3) zwischen den einzelnen Komponenten gewechselt.

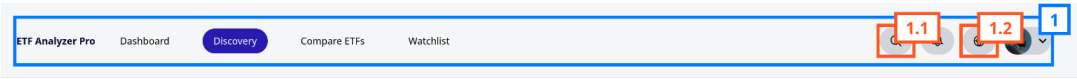


Abbildung 26: Navigationsleiste

Funktionalität Die *lib-feature-navbar* Komponente (siehe Bereich 1 Abbildung 26) zeigt auf der linken Seite Verlinkungen zu den verschiedenen Webseiten an und auf der rechten Seite andere Funktionalitäten. Durch Klick auf das Suchsymbol (siehe Bereich 1.1 Abbildung 26) kann mit dem Namen nach einem ETF gesucht werden (siehe Abbildung 27).

Abbildung 27: Suche nach ETFs

Durch Klick auf das Globussymbol (siehe Bereich 1.2 Abbildung 27) kann eine Sprache ausgewählt werden (siehe Abbildung 28).

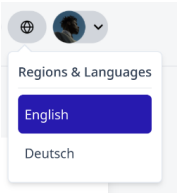


Abbildung 28: Sprachauswahl

Navigationsmöglichkeiten In der Navigationsleiste gibt es folgende Verlinkungen:

- ETF Analyzer Pro (3.2.5)
- Discovery (3.2.6)
- Compare ETFs (3.2.7)

3.2.4 Fußzeile

Die Fußzeile (siehe Abbildung 29) zeigt statische Informationen an.

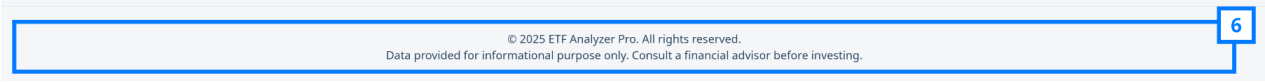


Abbildung 29: Fußzeile

3.2.5 Hauptseite

Die Hauptseite ist der Einstiegspunkt in die Applikation. Sie zeigt generelle Statistiken zu ETFs an und leitet auf weiterführende Komponente zur Suche, zum Vergleich und zur detaillierteren Ansicht von ETFs weiter.

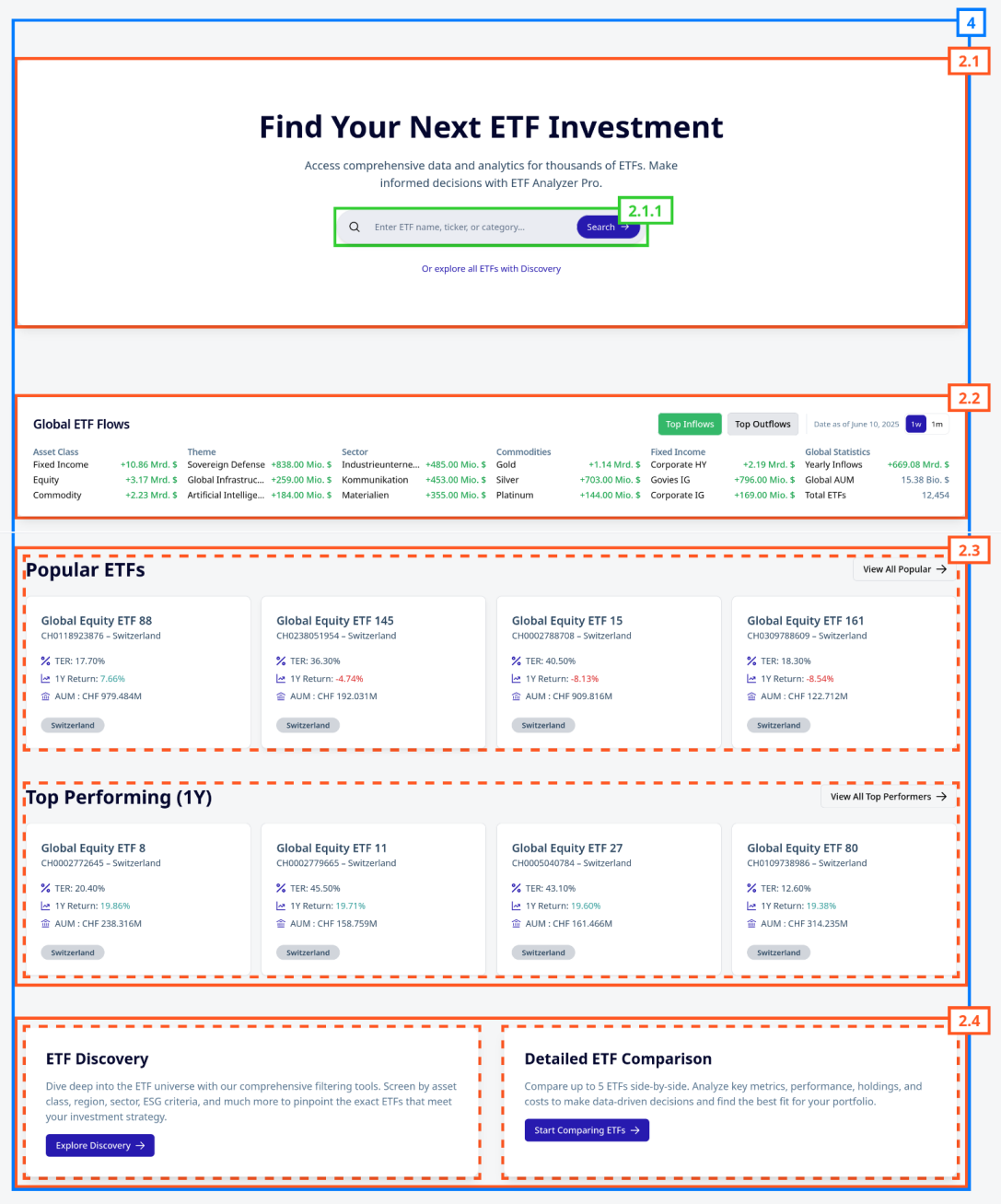


Abbildung 30: Home Page

Funktionalität Im Bereich *2.1* (siehe Abbildung 30) befindet sich die *lib-ui-header-card*. Innerhalb erlaubt die Suchkomponente im Bereich *2.1.1* der Benutzerin oder dem Benutzer eine Textsuche nach ETFs. Nach Eingabe von Zeichen in die Suche werden bis zu fünf Suchergebnisse angezeigt (siehe Abbildung 31).

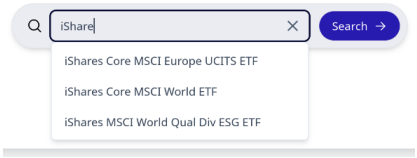


Abbildung 31: Suche von ETFs

Darunter befindet sich in *2.2* die *lib-ui-etf-flows* Komponent, welche eine Übersicht der globalen Veränderungen von gruppierten ETFs zeigt. Zur Zeit der Implementierung hat es noch keine Referenzdaten gegeben. Deswegen sind diese Daten hardkodiert. Bei *2.3* wird darunter zweimal die *lib-ui-top-etf-card* Komponente (siehe Abbildung 32) angezeigt. Bei den Popular ETFs wird nach Volatilität (2.1.2) und bei den Top Performing ETFs nach der Jahresrendite (2.1.2).

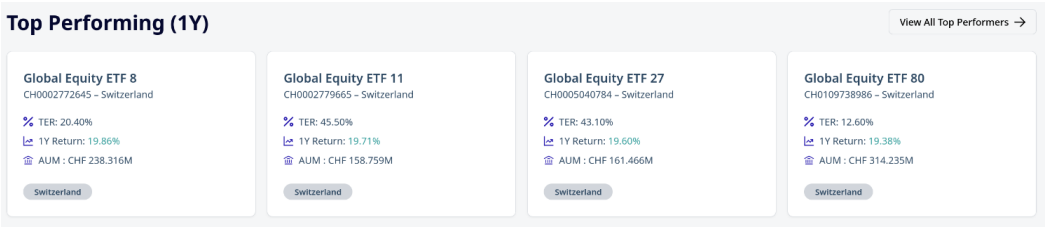


Abbildung 32: Top Performing ETFs

Weiterführende Verlinkungen zur Entdeckungskomponente (3.2.6) und Vergleichskomponente (3.2.7) befinden sich bei *2.4*. Die Anzeige der Überschrift, des beschreibenden Textes und des Buttons wird durch die Komponente *lib-ui-button-card* (siehe Abbildung 33) zur Verfügung gestellt.

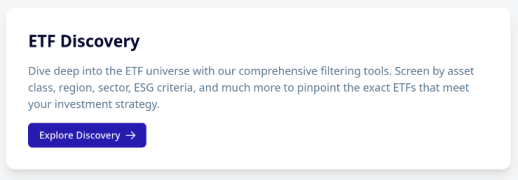


Abbildung 33: Button Card

Navigationsmöglichkeiten Neben der Navigationsleiste gelingt das Routing (2.1.3) zu anderen Komponenten über verschiedene Wege. Jedes angezeigte ETF in Abbildung 31 kann für die Navigation zur ETF Detailansichtskomponente (3.2.8) benutzt werden. Jedes ETF bei Abbildung 32 ist weiß hinterlegt und verlinkt auf die dazugehörige Detailansichtskomponente. In der rechten oberen Ecke der Abbildung 32 navigiert der View All Top Performers Button durch Klick auf die Entdeckungskomponente (3.2.6) und es wird automatisch die Sortierung nach Jahresrendite ausgewählt. Der primärgefärbte Button in Abbildung 33 verlinkt auf die Entdeckungskomponente

3.2.6. Dieselbe Komponente ist rechts daneben in Abbildung 30 zu sehen und verlinkt auf die Vergleichskomponente (38).

3.2.6 Entdeckungskomponente

Die Hauptfunktion der Entdeckungskomponente ist die Anzeige und Suche aller ETFs in einer Tabelle. Nachdem ein passendes ETF gefunden wurde, kann eine Detailansicht aufgerufen oder es mit anderen ETFs verglichen werden.

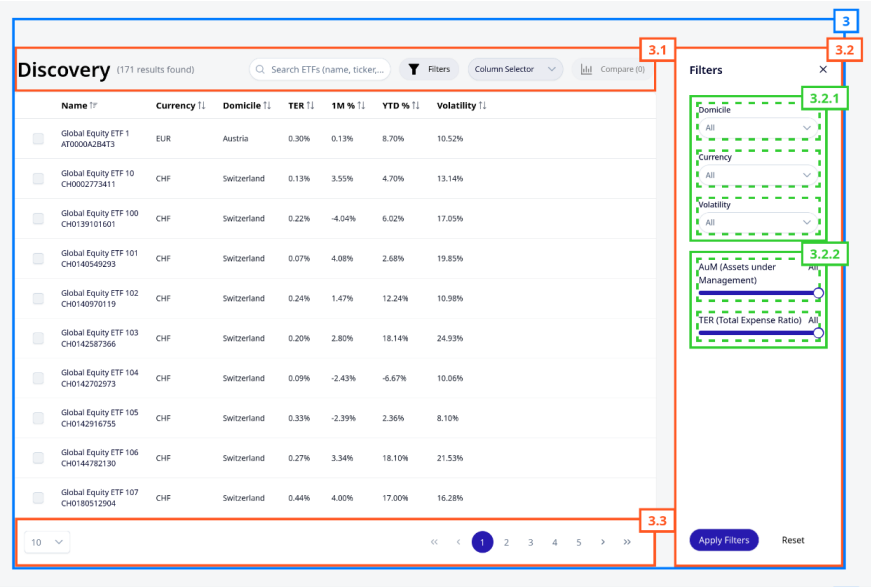


Abbildung 34: Entdeckung von ETFs

Funktionalität Alle ETFs haben nicht auf einer Seite Platz. Deshalb wird nur ein Teil unter der Verwendung von Pagination (2.1.1) angezeigt. Die Steuerung der Pagination erfolgt über die Paginatorkomponente (3.2.9) im Bereich *3.3*. Weitere Einschränkungen der Spalten und Filterung werden durch die Kopfzeilenkomponente (3.2.6) im Bereich *3.1* und Filterkomponente (3.2.6) im Bereich *3.2* bereitgestellt. Innerhalb der tabellarischen Anzeige erlaubt ein Klick auf die Spalten dessen Sortierung (2.1.1) nach auf- oder absteigender Reihenfolge. Eingestellte Filter und Sortierungen werden über NGXS Statemanagent (2.1.3) im Hintergrund zwischengespeichert. Dadurch bleiben diese Eigenschaften auch nach Hin- und Zurücknavigation bestehen. Die aktuelle Reihenfolge (2.1.1) wird durch das Icon neben der Spalte angegeben. Die Sortierung nach mehreren Spalten gleichzeitig wird nicht unterstützt. Durch Auswahl mehrerer ETFs über die Checkbox auf der linken Seite, können bis zu fünf ETFs ausgewählt werden.

Kopfzeile In der Mitte der Abbildung 35 gibt es eine Texteingabe, welche für die Filterung der angezeigten ETFs verwendet wird. Rechts daneben kann durch Button Klick auf Filters die

Filterkomponente (3.2.6) ein- und ausgeblendet werden, falls mehr Platz für die Anzeige der ETFs benötigt wird.

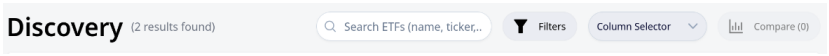


Abbildung 35: Kopfzeile

Der Column Selector Button öffnet auf Klick eine Liste von Spaltennamen (siehe Abbildung 36). Die Kontrollkästchen daneben ermöglichen die selektive Auswahl der angezeigten Spalten in der Tabelle. Die Spalte für den Namen ist verpflichtend und kann nicht deselektiert werden.

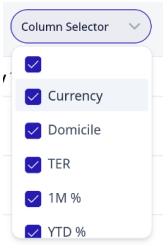


Abbildung 36: Spaltenauswahl

Filter Die Komponente *lib-feature-filter* befindet sich im Bereich *3.2* der Abbildung 34. Die Filter unterteilen sich in selektive Filter im Bereich *3.2.1*, welche Auswahlkomponenten (3.2.9) anzeigen, und Schieberegler im Bereich *3.2.2*, welche Schiebereglerkomponenten (3.2.9) anzeigen. Jedes eindeutige Land und jede eindeutige Währung kann für die Filterung selektiert werden. Die Volatilität wird nach vorgefertigten Intervallen, wie Low (0 - 10%) oder Medium (10 - 20%), ausgewählt. Schieberegler ermöglichen die Filterung nach Gesamtvermögen zwischen 0 und 60 Millionen des Währungsformates und nach Gesamtkostenquote von 0 bis 1. Die Grundauswahl der Filter ist auf den Wert Äll"gesetzt. Die Anwendung der Filter auf die Liste von ETFs erfolgt beim Klick auf den Apply Filters Button. Das Filtermenü kann über den Filters Button in der Kopfzeile (3.2.6) ein- und ausgeblendet werden.

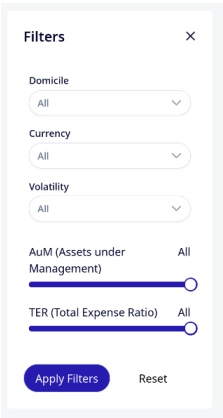


Abbildung 37: Filter

Navigationsmöglichkeiten Bevor ein ETF ausgewählt wird ist der Compare Button (siehe Abbildung 35) ausgegraut. Sobald eine Auswahl mithilfe der Checkboxen (2.5.1) erfolgt ist, erlaubt dieser durch Klick die Navigation zur Vergleichskomponente (3.2.7). Weiters können die Detailinformationen 3.2.8 zu einem ETF mithilfe eines Klicks auf den Namen angezeigt werden.

3.2.7 Vergleichskomponente

In der Vergleichskomponente können mehrere ETFs über Kennzahlen miteinander verglichen werden. Wenn ein interessantes ETF gefunden wurde, kann die Detailansicht aufgerufen werden.

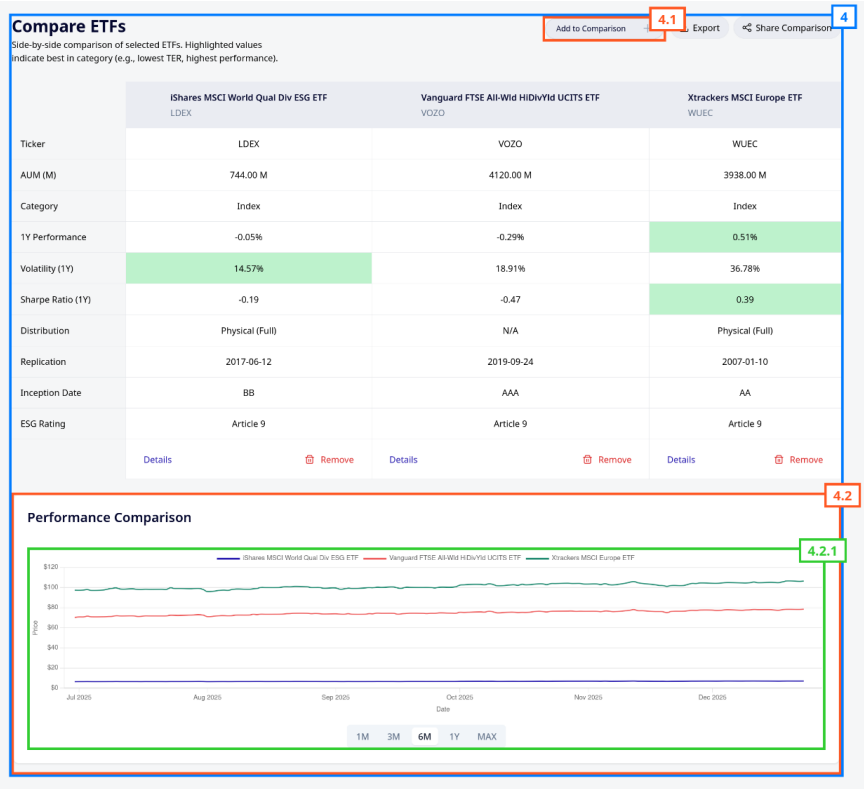


Abbildung 38: Vergleich von ETFs

Funktionalität Die Vergleichskomponente erlaubt den Vergleich der Kennzahlen selektierter ETFs. Diese werden tabellarisch nebeneinander angezeigt. Im NGXS Statemanagement (2.1.3) wird eine Liste von Identifkationen der ausgewählten ETFs gespeichert. Die Liste wird als *Comparison Basket* bezeichnet. Der Add To Comparison Button im Bereich *4.1* der Abbildung 38 öffnet durch Klick eine Mehrfachauswahl (siehe Abbildung 39) und kann für das Hinzufügen von bis zu fünf ETFs zum Comparison Basket verwendet werden. Dabei wird die Multiselect PrimeNg Komponente verwendet (2.5.1).

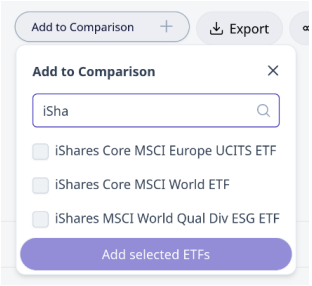


Abbildung 39: Mehrfachauswahl von ETFs

Außerdem bietet der Export Button eine Konvertierung der HTML Ansicht zu einem PDF. Unterhalb jedes ETFs befindet sich ein Remove Button, um diesen aus dem Comparison Basket entfernen zu können. Unterhalb der tabellarischen Ansicht der Hauptkennzahlen im Bereich *4.2* wird die Performance der ETFs im Comparison Basket in einem Grafen (3.2.9) visualisiert.

Navigationsmöglichkeiten Zu jedem ETF kann die ETF Detailansichtskomponente (3.2.8) durch Klick auf den Details Button links unterhalb eines ETFs angezeigt werden.

3.2.8 ETF Detailansichtskomponente

Die ETF Detailansichtskomponente zeigt und visualisiert Informationen zu einem ETF. Die wesentlichen Informationen werden in der Kopfzeile angezeigt. Darunter ist ein Menü, um zwischen den einzelnen Visualisierungen und Ansichten auf das ETF durchschalten zu können.

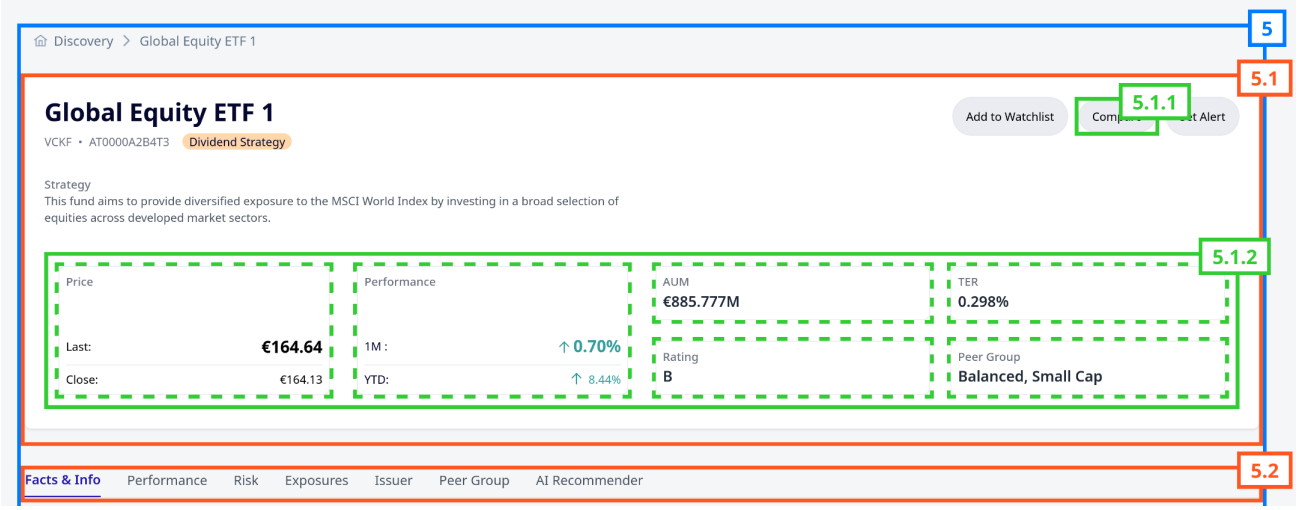


Abbildung 40: Essenzielle Informationen

Funktionalität Grundlegende Informationen zu einem ETF werden in der *lib-ui-essential-card* Komponente (siehe Bereich *5.1* Abbildung 40) dargelegt. Diese stellt im Bereich *5.1.2*

Informationen zu Ticker, Namen, Strategie, Preis, Rendite, Gesamtvermögen, ESG-Rating,er Gesamtkostenquote und Peer Group dar (2.1.2). Der aktuelle Pfad wird in mithilfe der Breadcrumb PrimeNG Komponente (2.5.1) in links oben angezeigt (siehe Abbildung 40). Der Compare Button (siehe Bereich *5.1.1* Abbildung 40) erlaubt das Hinzufügen des ETFs zum Comparison Basket, damit dieser infolgedessen mit anderen ETFs in der Vergleichskomponente (3.2.7) verglichen werden kann und den Zugriff auf einen Link zum Peer Group Ranking (3.2.8). Unterhalb der grundlegenden Informationen befindet sich die *lib-ui-menu* Komponente (siehe Bereich 5.2 Abbildung 40) zum Öffnen von detaillierteren Informationen und Visualisierungen. Es folgt eine Liste mit Verlinkungen zu den einzelnen Menüpunkten, die durch eine Divider PrimeNG Komponente (2.5.1) ausgewählt werden können:

- Facts & Info (41)
- Performance (42)
- Risk (43)
- Exposures (44)
- Issuer (45)
- Peer Group (46)

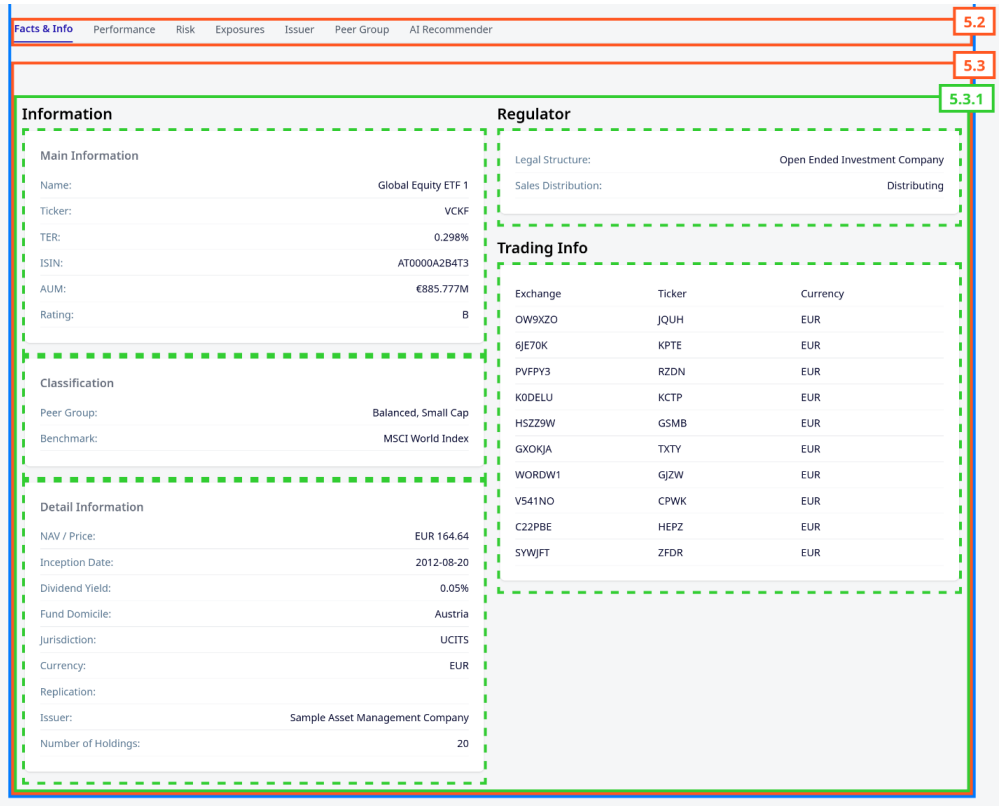


Abbildung 41: Hauptinformationen

Funktionalität Die *lib-feature-main-information* (siehe Bereich 5.3 Abbildung 41) bietet eine weitaus detailreichere Ansicht über die Informationen eines ETFs als die essenzielle Informationskomponente (siehe Abbildung 40). Die Informationen sind in Regulator, Trading Info und Information mit den Unterkategorien Main Information, Classification und Detail Information gruppiert. Jede Unterteilung ist in eine eigene *lib-ui-card* (siehe Bereich 5.3.1 Abbildung 41) verpackt, welche diese visuell vom Hintergrund abhebt.

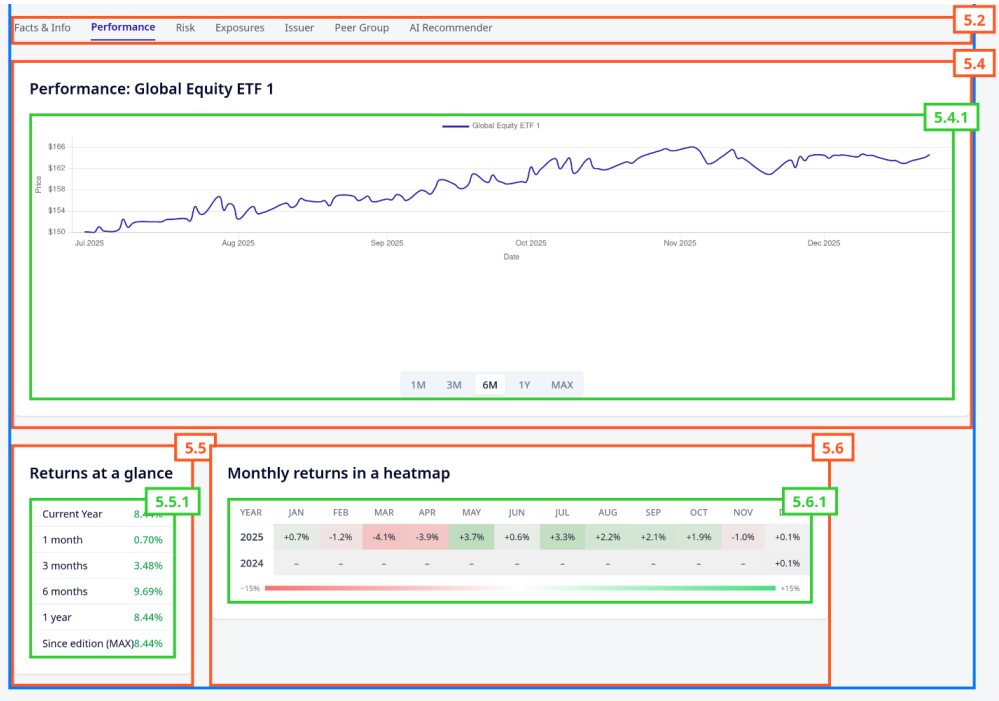


Abbildung 42: Performance Charts

Funktionalität Die Rendite eines ETFs wird über Zeiträume von Jahren gemessen. Den Verlauf der Rendite wird in der Renditengrafkomponente (3.2.9) dargestellt. Die Rendite zu den Zeitpunkten vom aktuellen Jahr, 1 Monat, 3 Monate, 6 Monate, 1 Jahr oder seit dem Auflegungsdatum des ETFs (2.1.2) wird links unten angezeigt (siehe Bereich *5.5* Abbildung 42). Zusätzlich bietet die Heatmap Komponente (3.2.9) (siehe Bereich 5.6 Abbildung 42) eine Übersicht der aktuellen Änderungsrate des Kursverlaufes.

Abbildung 43: Risikoanalyse

Funktionalität Die *lib-feature-risk-analysis* Komponente (siehe Bereich 5.7 Abbildung 43) zeigt Daten zur Einschätzung des Risikos eines ETFs in einer Tabelle an. In der ersten Spalte befinden sich die verschiedenen Metriken. Die weiteren Spalten beinhalten die Werte in verschiedenen Zeitintervallen. Eine Kenngröße, der momentane Risikorendite (5), wird zusätzlich mit den Farben Rot, Gelb und Grün und einem Balken visualisiert. Wenn die Zahl negativ ist, wird sie in Rot dargestellt andernfalls in Grün. Die Kästchen im Balken korrespondieren zu den Schwellen 5%, 20%, 30% und 50%. Sobald eine Schwelle erreicht ist, übernimmt das Kästchen die Farbe der Zahl. Wenn keine Schwelle erreicht wird und die Risikorendite deshalb unter 5% liegt, wird der unterste Balken gelb eingefärbt. Der Balken ist mit folgendem HTML implementiert:

Listing 5: Risikorendite

```
<div class="flex flex-col w-2 gap-[2px]">
    @let color = (value >= 5) ? 'trafficLight-green' : (value >=
       -5) ?
      'trafficLight-yellow' : 'trafficLight-red';
    @for (threshold of
      thresholds(); track threshold) {
      <div
        class="h-1.5 rounded-sm bg-gray-200"
        [ngClass]="Math.abs(value) > threshold ? 'bg-' + color :
          ''"
      ></div>
    }
</div>
```

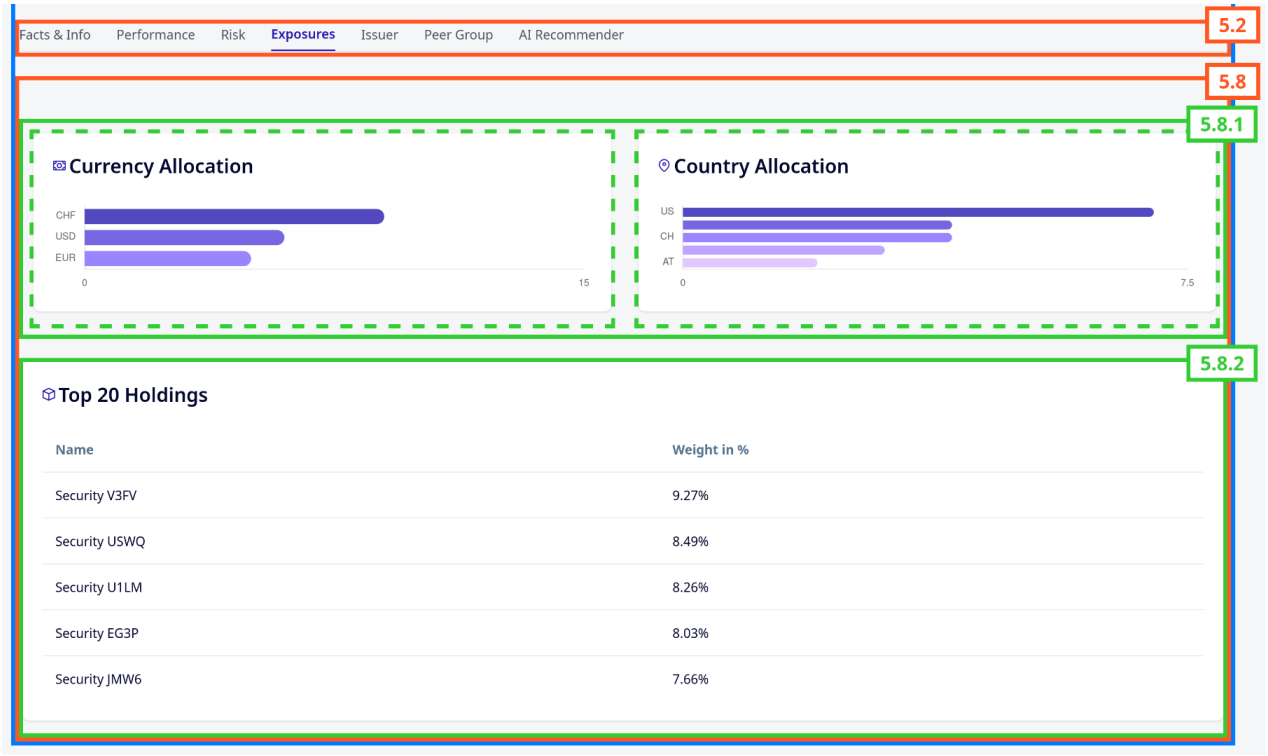


Abbildung 44: Zusammensetzung

Funktionalität Zu jedem ETF gehören mehere Holdings (2.1.2). Die Forderungskomponente bietet eine Übersicht und Visualisierungen aller Holdings eines ETFs. Holdings werden in verschiedenen Währungen gehandelt und kommen aus verschiedenen Ländern. Im Balkendiagramm (3.2.9) wird der Anteil der Holdings gruppiert nach Währung und auf der rechten gruppiert nach Land an dem ETF visualisiert (siehe Bereich 5.8.1 Abbildung 44). Unterbei ist eine Liste der Top 20 Holdings (siehe Bereich 5.8.2 Abbildung 44) sortiert nach der Gewichtung im ETF. Es werden für Demonstrationszwecke nur fünf Holdings angezeigt, weil dadurch das Bild kompakter ist.

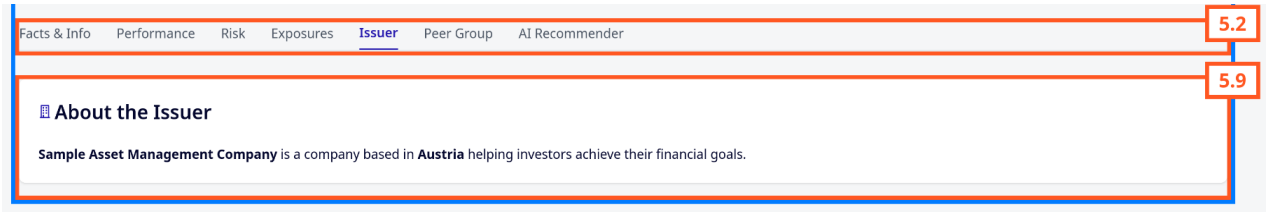


Abbildung 45: Emmitent

Funktionalität Die *lib-ui-issuer* Komponente (siehe Bereich 5.9 Abbildung 45) zeigt den Emittenten des ETFs 2.1.2 an.

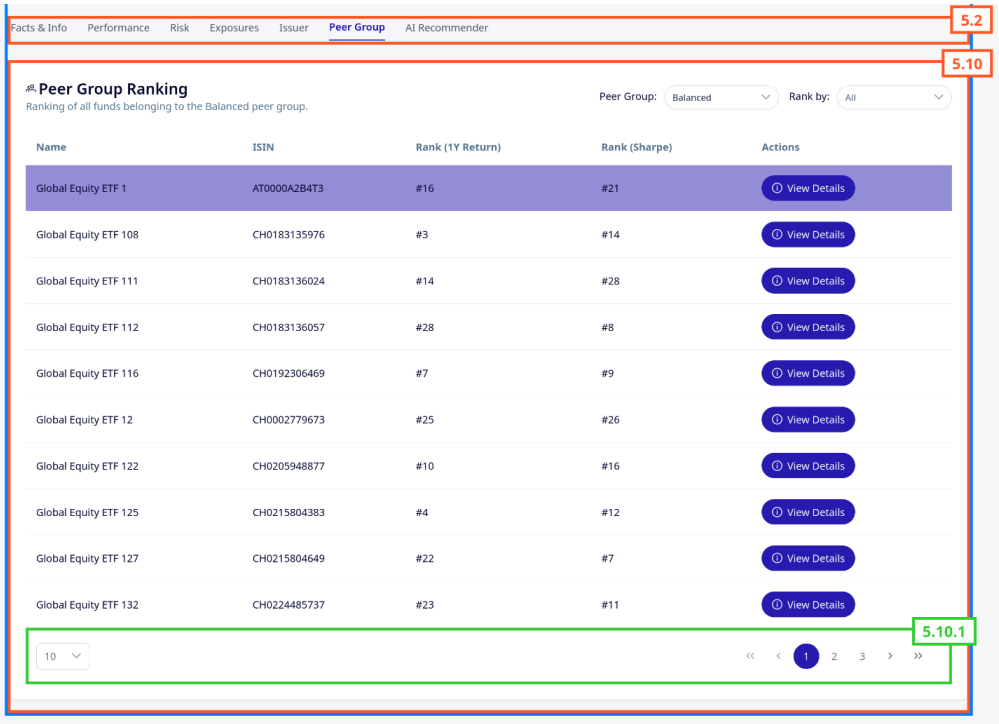


Abbildung 46: Peer Group Ranking

Funktionalität In der ETF Detailansichtskomponente werden die Informationen zu einem ETF geladen - dem *aktuellen ETF*. Die *lib-feature-peer-group-ranking* zeigt ETFs zugehörig zur ausgewählten Peer Group 2.1.2 des aktuellen ETFs an. Das aktuelle ETF wird mit lila hinterlegt, damit es leichter zu erkennen ist. Alle ETFs dieser Peer Group werden in einem Ranking angezeigt. In der Kopfzeile kann mit der Auswahlkomponente (3.2.9) zu anderen Peer Groups gewechselt die Sortierung, das Ranking, innerhalb einer Peer Group angepasst werden. Das Peer Group Ranking verwendet Pagination 3.2.9 (siehe Bereich 5.10.1 Abbildung 46). Die Nummer eins unter der Kategorie Jahresrendite wird durch eine goldene Trophäe und goldener Hinterlegung des #1 hervorgehoben (siehe Abbildung 47).

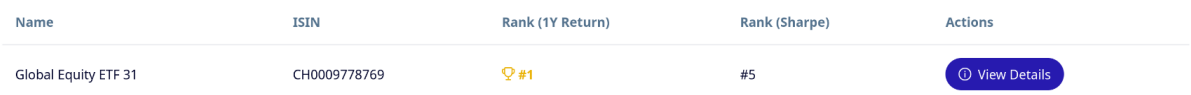


Abbildung 47: Peer Group Ranking

Navigationsmöglichkeiten Nachdem ein passendes ETF in der Tabelle gefunden wurde, leitet ein Klick auf den View Details Button (siehe Abbildung 47) zu der ETF Detailansichtskomponente (3.2.8) weiter.

3.2.9 UI Komponenten

Nachfolgend wird ein Auszug der wichtigsten Komponenten (2.1.3) der Applikation genauer vorgestellt. Diese sind allesamt im `libs`-Verzeichnis zu finden (siehe Projektstruktur Angular Frontend 3.1.2).

Paginatorkomponente Die *Paginatorkomponente* erlaubt innerhalb einer Liste die Anzahl der angezeigten ETFs einzustellen und über die verschiedenen Seiten von ETFs zu navigieren (2.1.1). Die Kapselung der beiden Funktionalitäten in eine Komponente erlaubt die Wiederverwendung in der Entdeckungskomponente (3.2.6) und der Peer Group Ansicht der Detailansichtskomponente (3.2.8). Für die Navigation wird der PrimeNG Paginator (2.5.1) verwendet. Für die begrenzte Anzahl der angezeigten ETFs stehen die Werte 10, 20 oder 30 zur Auswahl. Dabei ist für das Testen der Funktionalität ausschlaggebend, dass mehr als 30 Testdaten (3.4.4) eingefügt sind.

Auswahlkomponente Die *Auswahlkomponente* ermöglicht die Auswahl eines Wertes. Die Komponente verkapselt die Funktionalität des PrimeNG Select (2.5.1). Dadurch bleibt das Aussehen einer Auswahl konsistent und Übersetzungsfunktionalitäten können zentral in einer Komponente verwaltet werden. Anstatt des PrimeNG Select wurde zu Beginn ein nativer Select-Tag verwendet. Aufgrund der Herausforderungen, den Tag anzupassen, wurde danach die Bibliothek verwendet (3.2.10). Als Parameter wird eine Liste von Schlüssel-Wert Paaren übergeben. Der Schlüssel wird mit NGX-Translate (2.5.3) übersetzt und der Wert dient als Rückgabewert. Die Auswahlkomponente wird im Filter der Entdeckungskomponente (3.2.6) und der Peer Group Ansicht der Detailansichtskomponente (3.2.8) eingesetzt.

Schiebereglerkomponente Mit der *Schiebereglerkomponente* kann eine Zahl zwischen einem minimalen und maximalen Wert ausgewählt werden. Aufbauend auf dem PrimeNG Slider (2.5.1) erlaubt die Komponente die Formatierung mit einem Suffix, zum Beispiel "%", und die Zahl unter Angabe eines Musters zu formatieren. Die Komponente wird im Filter der Entdeckungskomponente (34) verwendet.

Suchleiste Eine wichtige Komponente in der Applikation ist die *Suchleiste*. Die Komponente kann durch die Übergabe eines zusätzlichen Parameters entweder in einer großen oder kleinen Ausführung angezeigt werden. Erstere Variante kommt auf der Startseite (31) zum Einsatz, während die andere an mehreren Stellen vorkommt und somit den Standard für eine Suche

bildet. Unter anderem kommt sie in der Navigationsleiste (27) vor.
Unabhängig von der Darstellung verwendet die Implementierung die PrimeNG-Komponente AutoComplete. Wie unter den verwendeten Bibliotheken erklärt (2.5.1) ermöglicht sie die Eingabe mit automatischer Filterung der Auswahloptionen. Weiter wird bei der kleinen Ausführung auf ein IconField (2.5.1) gesetzt, um bei der Eingabe auch noch ein Icon einer Lupe anzuzeigen.

Sprachenauswahl Die *Sprachenauswahl* befindet sich in der Navigationsleiste (siehe 28). Dabei wird auf die PrimeNG-Komponente Popover zurückgegriffen, um das Entscheidungsfenster in einem Dialog im Vordergrund anzuzeigen. Wenn sich die Auswahl ändert, so wird automatisch die ganze Webseite übersetzt.

Renditengrafkomponente Mit der *Renditengrafkomponente* werden die Kursverläufe eines ETFs in einem Liniendiagramm dargestellt. Hierzu stellt PrimeNG die Komponente Charts (2.5.1) zur Verfügung, durch die verschiedene Arten von Diagrammen angezeigt werden können. Bereits standardmäßig wird beim Überfahren eines Datenpunkts mit der Maus ein Dialog angezeigt, der detaillierte Informationen zu diesem Datenpunkt enthält.

Damit die Benutzerin oder der Benutzer den Zeithorizont der Daten einschränken kann, hat das Projektteam zusätzlich einen PrimeNG SelectButton (2.5.1) eingebunden. Dieser ermöglicht eine Auswahl von einem Monat, drei Monaten, sechs Monaten, einem Jahr oder keiner Begrenzung. Eine Erweiterung in der Logik ermöglicht es der Komponente, mehrere ETFs zur Anzeige zu übergeben. Hierzu muss der `boolean`-Parameter `multipleLines=true` gesetzt und die Preise in einem zweidimensionalen Array übergeben werden. Die erste Dimension beschreibt hierbei die verschiedenen Fonds, während die Zweite die einzelnen Preise im Verlauf der Zeit repräsentiert. In diesem Fall werden aus dem Array mehrere Datasets extrahiert und dem Graf überlassen.

Im nachfolgenden Codesegment (6) wird die Aufbereitung der Datensätze aus einem zweidimensionalen Array zur Darstellung im Diagramm veranschaulicht, wobei `rawSets` die übergebenen Preise und `this.data()` die Anzeigekonfiguration des Grafen sind. Der Aufruf `this.getColor(index)` liefert eine individuelle Farbe zurück, um eine Unterscheidbarkeit zwischen den Fonds zu gewährleisten.

Listing 6: Aufbereitung der Datensätze aus einem zweidimensionalen Array

```
1 const rawSets = this.rawData as SpotPrice[][];
2 this.data().datasets = rawSets.map((dataset, index) => {
3   const pts = dataset
4     .map(d => ({
```

```
5         t: parseISO(d.priceDate).getTime(),
6         y: d.price
7       }))
8       .filter(p => p.t >= start && p.t <= now);
9
10    if (index === 0) {
11      this.data().labels = pts.map(p => p.t);
12    }
13
14    return {
15      label: this.labels[index] ?? `Dataset ${index + 1}`,
16      data: pts.map(p => p.y),
17      fill: false,
18      borderWidth: 2,
19      pointRadius: 0,
20      tension: 0.4,
21      borderColor: this.getColor(index)
22    };
23  });
```

Einsatz findet die Komponente einerseits in der Performance-Analyse der Detailansichtskomponente (siehe Abbildung 42), andererseits in der Vergleichskomponente (siehe Abbildung 38), wobei es genau hier vorkommen kann, dass mehrere ETF-Kursverläufe gleichzeitig sichtbar sind. Dies hängt von der Anzahl der Fonds im Vergleichskorb ab.

Heatmap Die *Heatmap* dient Anwenderinnen und Anwendern dazu, einen Eindruck von der Kursentwicklung jedes Monats eines Jahres zu gewinnen. Hierfür werden die übergebenen Preise vorverarbeitet und danach im Template in einer Tabelle ausgegeben. Zunächst werden die vorhandenen Kursdaten nach Datum sortiert. Anschließend werden die Datenpunkte nach Jahr und Monat gruppiert, wobei für jeden Monat der erste und der letzte Kurswert gespeichert werden. Auf Basis dieser beiden Werte wird die monatliche Rendite berechnet, indem der letzte Preis durch den ersten Preis des Monats dividiert und schließlich eins subtrahiert wird. Die Hintergrundfarbe einer Zelle macht sowohl positive als auch negative Entwicklungen schnell erkennbar.

Die Komponente kommt in der Performance-Analyse der Detailansichtskomponente (siehe Abbildung 42) zum Einsatz.

Balkendiagramm-Container Der *Balkendiagramm-Container* dient zur Darstellung eines hervorgehobenen UI-Elements, das ein Icon, eine Überschrift sowie ein Balkendiagramm enthält. Auch diese grafische Abbildung hat das Projektteam mittels der PrimeNG-Komponente Charts umgesetzt.

Anwendung findet die Anzeige bei den Informationen zur Zusammensetzung des ETFs in der Detailansichtskomponente (siehe Abbildung 44).

3.2.10 Herausforderungen

Aussehen der Auswahloptionen Anfangs wurde für die Auswahloptionen der Kriterien des Filters (siehe Abbildung 37) ein nativer Select-Tag verwendet. Das Problem bei der Anpassung des Aussehens von nativen Select-Tags ist die tiefe Verstrickung mit dem Betriebssystem. [175] Deshalb wurde die Auswahlkomponente von PrimeNg (2.5.1) verwendet, um die Auswahloptionen farblich konsistent zu halten.

Tabellenspalten Die Spalten in der Entdeckungskomponente (3.2.6) sollen sich immer auf der linken Seite befinden und nur die minimale Breite einnehmen. Zusätzlich muss die Tabelle die restliche Breite bis zu den Filtern auf der rechten Seite auffüllen. Deshalb wird die Tailwind CSS Klasse w-full 2.2.1 verwendet. Diese kann entweder auf die letzte Spalte (siehe Listing 7) angewendet oder ein colgroup Tag (siehe Listing 8) verwendet werden. Der Vorteil des colgroup Tags ist, dass kein leerer Tag am Ende benötigt wird, der keine semantische Bedeutung hat.

Listing 7: Tabellenspalten auf der linken Seite mit th-Tag

```
1 <th class="w-full"/>
```

Listing 8: Tabellenspalten auf der linken Seite mit colgroup

```
1 <colgroup>
2   <col span="{{tableHeaders().length}}">
3   <col class="w-full">
4 </colgroup>
```

Abstrakte Klassen für Dependency Injection Das Angular Frontend wurde implementiert, während das Spring Boot Backend noch nicht alle anzuzeigenden Daten geliefert hat. Deswegen gibt es für jeden Angular Service (2.1.3) zwei Varianten: die eine greift auf das Backend zu und die andere mockt die Daten. Der Code in den Komponenten soll unabhängig davon sein, welche der beiden Varianten verwendet wird. Deswegen verwenden alle Komponenten eine abstrakte Basisklasse, anstatt der konkreten Implementierungen. Schnittstellen können nicht für Dependency Injection (2.1.1) verwendet werden, weil diese nur während der Kompilierzeit existieren. Im Gegensatz dazu verweilen abstrakte Klassen bis zur Laufzeit.

3.3 Spring Boot Backend

Das Spring Boot Backend wurde entsprechend der benötigten Funktionalitäten erweitert. Die einzelnen Methoden und Herausforderungen werden nachfolgend erklärt.

3.3.1 Funktionalitäten

Die bestehenden Funktionalitäten des Backends (1.3) wurden erweitert.

Methode getFundByIsIn Die in der Ausgangssituation vorhandene Methode *getFundByIsIn* liefert Detailinformationen zu einem bestimmten ETF. Im Rahmen dieser Arbeit wurde ein weiterer Parameter, um Preisstatistiken für mehrere Zeitpunkte zu berechnen, hinzugefügt. Die Preisstatistiken (siehe Bereich 5.1.2 in Abbildung 40 und Bereich 5.5 in Abbildung 42) werden in der ETF Detailansichtskomponente (3.2.8) gezeigt.

Methode getAllFundsFiltered Die Methode *getAllFundsFiltered* liefert eine Liste zu ETFs und wird von der Entdeckungskomponente (3.2.6) für Pagination,Pagination Sortierung und Filterung (2.1.1) verwendet. Die Parameter für Pagination und Sortierung sind analog zur getAllComparisonFunds Methode 3.3.1. Alle Filterparameter sind übersichtlich in eine Klasse gruppiert. Im Repository (2.1.1) werden ETFs mithilfe von Derived Query Methoden gefiltert.

Methode getAllComparisonFunds Die Methode *getAllComparisonFunds* returniert Informationen zum Vergleich mehrere ETFs 3.4.3 identifiziert durch eine Liste von ISINs (2.1.2). Sie erlaubt Pagination und Sortierung (2.1.1). Der Sortierparameter besteht aus dem Spaltennamen. Optional kann davor das Minuszeichen "gesetzt werden, um absteigend zu sortieren. Für die Pagination wird ein Parameter für den Startindex und einer für die Anzahl der zu returnierenden ETFs benötigt. Der Kurs von ETFs verändert sich über die Zeit. Deswegen müssen ETFs zu einem bestimmten Zeitpunkt verglichen werden. Dafür wird Anzahl der Monate mit dem Prefix "M"(z.B. "M12"für 1 Jahr zuvor) 3.3.2 übergeben.

Sonstige Die Methoden *getAllDomiciles* und *getAllCurrencies* listen alle eindeutigen Länder und Währungen auf, welche für die Auswahlmöglichkeiten von selektiven Filtern (3.2.9) in der Entdeckungskomponente (3.2.6) verwendet werden.

3.3.2 Herausforderungen

Zeitformat Die Übergabe der Zeit kann in verschiedenen Formaten erfolgen. Millisekunden haben den Nachteil, dass die großen Zahlenbereiche bei Daten schwer lesbar sind für Menschen. Ein Datum als Zeichenkette formatiert zu übergeben ist unpraktisch, weil Zeitpunkt ein vielfaches eines Monats sind und Daten viel feingranularer auf Tagesebene arbeiten. Dadurch können leicht Off-by-one Fehler entstehen. Konstanten mit Enumerationen zu definieren ist zu einschränkend, weil festgelegt werden muss, welche einzelnen Zeitpunkte gültig sind und bei Erweiterungen der Quellcode angepasst werden muss. Deshalb und aufgrund der Konsistenz mit der .xml Datei wurde die Anzahl der Monate mit dem Prefix "M"(z.B. "M12"für 1 Jahr zuvor) als Datumsformat verwendet.

Benutzerdefinierte Query Für die Rankingattribute 3.4.3 muss eine benutzerdefinierte Query [176] verwendet werden, welche Pagination und Sortierung 2.1.1 nicht unterstützten . Deshalb wurde Pagination mit einem NamedParameterJdbcTemplate [177] implementiert. Zuerst wird das SQL Statement zusammengebaut: Es greift auf die Datenbankfunktion *get_comparison_funds* 3.4.3 zurück und nutzt folgende Abfrageklauseln: *LIMIT* legt die maximale Anzahl an Datensätzen fest. *OFFSET* beschreibt, wie viele Datensätze übersprungen werden sollen, vor dem ersten Datensatz des *Result Sets* (engl. für Ergebnismenge). *ORDER BY* sortiert das Ergebnis der Query entweder aufsteigend oder absteigend. Bei der Programmierkonvention *Camel Case* werden Wörter, anstatt durch Leerzeichen getrennt, direkt aneinander geschrieben. Das erste Wort wird kleingeschrieben und alle weiteren werden großgeschrieben. Bei der Programmierkonvention *Snake Case* werden Wörter kleingeschrieben und durch Unterstriche voneinander getrennt. Weiters nutzen die Spaltennamen im Frontend Camel Case und in der Datenbank wird Snake Case verwendet. Deshalb werden die Spaltennamen zuerst mit einer Map von Camel Case in Snake Case umgewandelt, bevor diese für die Sortierung verwendet werden.

3.4 PostgreSQL Datenbank

Auf Datenbankebene wurde das Schema erweitert, eine Funktion implementiert und Testdaten generiert. Die einzelnen Umsetzungen werden nachfolgend erläutert.

3.4.1 Datenmodell

In der Abbildung 48 ist ein Teildatenmodell mit allen Tabellen zu sehen, die für unser Projekt relevant sind. Die Farben deuten an, ob die Tabelle hinzugefügt (Blau), erweitert (Grün) oder aus der Ausgangssituation (1.3) ohne Änderung übernommen (Gelb) wurde.

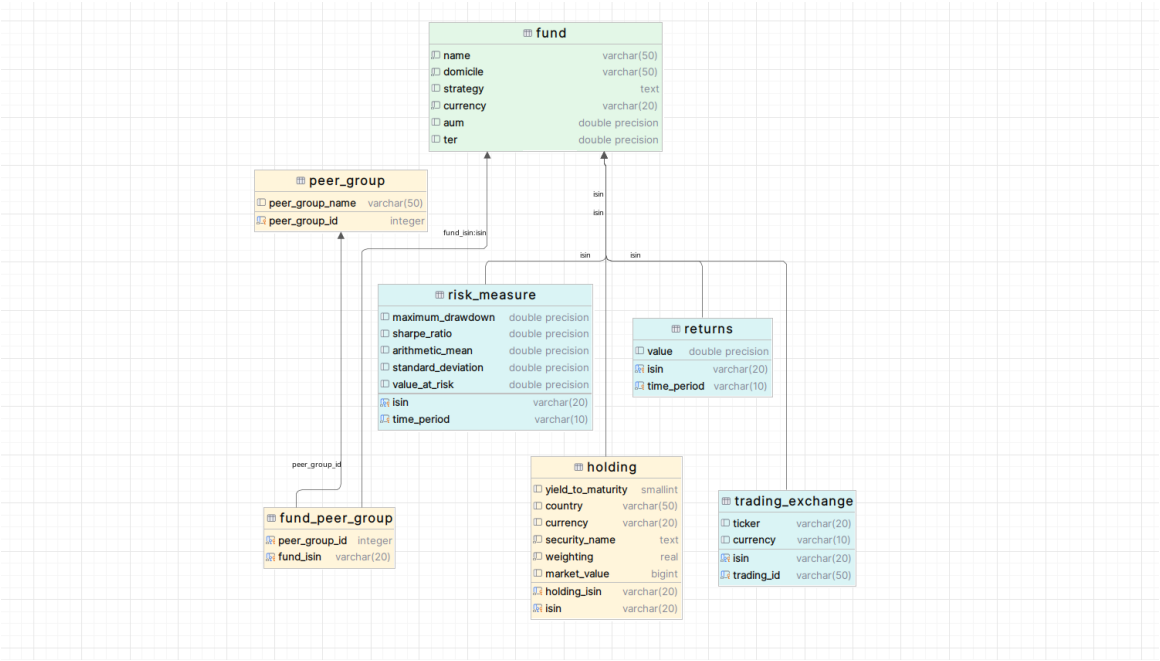


Abbildung 48: ERD

3.4.2 Schemaerweiterung

In der Datenbank sind zu Beginn (1.4) nicht alle Attribute vorhanden gewesen, welche für das Design benötigt werden. Deswegen wurde das Datenbankschema mit folgenden Tabellen und Attributen erweitert. Als Quelle dienen .xml Dateien (siehe Abbildung 24).

Navigation in XMLDatei Die einzelnen Attribute der Tabellen werden aus den XML-Dateien ausgelesen. Die Dateien sind eins bis vier Megabyte groß und jede Datei wurde manuell nach passenden XML Tags durchsucht worden, weil es keine Dokumentation für die Attribute gibt. Der größte Teil der Datei ist eine Liste von Holdings (2.1.2). Bei jeder Liste muss nur das erste Element angesehen werden, weil die restlichen demselben Schema folgen. Diese Strategie wurde verwendet, die gesamte Datei durchzugehen und das Datenbankschema nach im Design enthaltenen Attributen zu erweitern. Diese wurden in Tabellen gruppiert und normalisiert.

Tabelle trading_exchange Die Tabelle *trading_exchange* speichert Informationen über die Handelsbörse. Jeder Datensatz speichert die Handelsidentifikation (z.B. OW9XZO), der Ticker

(z.B. JQUH), die Handelswährung (z.B. EUR für Euro) und die ISIN des Fonds (2.1.2) gespeichert. Die Informationen werden sind unter dem Menüpunkt Facts & Info (siehe Abbildung 41) in der ETF Detailansichtskomponente (3.2.8) ersichtlich.

Tabelle risk_measures Die Tabelle *risk_measures* speichert Risikometriken (2.1.2) über verschiedene Zeiträume. Jeder Datensatz speichert Zeitraum (z.B. M12), höchsten Wertverlust, Sharpe Ratio, Risikorendite, Volatilität und Risikowert 2.1.2. Pfade zum Zugriff auf die .xml Datei wurden in einer *Config* Klasse mit Konstanten definiert. Die Werte der Tabelle *risk_measures* werden dem Pfad /ClassPerformance/Performance/TrailingPerformance/RiskAndRating/RiskAnalysis/RiskMeasures/RiskMeasuresDetail entnommen, welche enthält eine Liste folgender Einträge:

Listing 9: XML Daten

```
<RiskMeasuresDetail TimePeriod="M12" Type="71">
  <EndDate>2023-10-31</EndDate>
  <ArithmeticMean>0.925</ArithmeticMean>          <!-- Return -->
  <StandardDeviation>14.99</StandardDeviation>  <!-- Volatility -->
  <Skewness>0.26</Skewness>
  <Kurtosis>-1.60</Kurtosis>
  <MaximumDrawdown>-9.32</MaximumDrawdown>
  <SharpeRatio>0.40</SharpeRatio>
  <SortinoRatio>0.70</SortinoRatio>
  <AverageGain>4.70</AverageGain>
  <AverageLoss>-2.87</AverageLoss>
</RiskMeasuresDetail>
```

Die Informationen sind unter dem Menüpunkt Risk (siehe Abbildung 43) in der ETF Detailansichtskomponente 3.2.8 ersichtlich.

Tabelle returns Die Tabelle *returns* speichert die Rendite eines ETFs über mehrere Zeiträume. Jeder Datensatz speichert den Zeitraum (z.B. M12) und die Rendite. Die Monats- und Jahresrendite werden aus diesen Werten entnommen und sind in der Entdeckungskomponente (3.2.6) zu sehen. Die Daten zur Rendite können auch von der Risk-Schnittstelle 2.6.1 abgefragt werden und dadurch kann sich die Monats- und Jahresrendite berechnet werden. Allerdings unterstützt die Schnittstelle nicht, nur akkumulierte Resultate zu returnieren und die Werte aus einem Zeitraum von einem Jahr zu berechnen hat während der Implementierung bei 170 ETFs 40 Sekunden gedauert. Deshalb wurden die Daten nicht von der Risk-Schnittstelle, sondern von den .xml Dateien genommen.

Tabelle fund Die Tabelle *fund* speichert generelle Informationen eines ETFs. Es wurde die bereits bestehende Tabelle mit 34 Attributen auf 48 erweitert. Jeder Datensatz speicher zusätzlich das Gesamtvermögen (AuM), Gesamtkostenquote (TER), Dividendenrendite, Auslegungsdatum, ob es eine akkumulierte Verteilung ist, Schlusskurs, Rechtstruktur, Gerichtsbarkeit, Namen und Land des Emittenten, Volatilität, Monats- und Jahresrendite (2.1.2).

Speicherung von Tabellen mit zusammengesetztem Schlüssel Zur Zeit der Implementierung hat JDBC keine zusammengesetzten Schlüssel unterstützt [178]. Zum Beispiel gehören die Attribute Zeitpunkt und ISIN in der Tabelle returns (3.4.2) zum Primärschlüssel. In der Umsetzung wird nur ISIN als Primärschlüssel definiert. Um Duplikate zu vermeiden überprüft die Einfügemethode zuerst, ob schon ein Datensatz mit derselben ISIN und demselben Zeitpunkt existiert. Nach der Implementierung am 27. Juli 2025 [179] ist Unterstützung für zusammengesetzte Schlüssel zu JDBC hinzugefügt worden.

3.4.3 ETF Ranking

Für das Ranking von ETFs innerhalb einer Peer Group (2.1.2) oder für den Vergleich von ETFs in der Vergleichskomponente (3.2.6) wurde die Funktion *get_comparison_funds* (2.2.3) implementiert. Diese ergänzt für jede Spalte in der Tabelle risk_measure (3.4.2) ein ganzzahliges Rankingattribute, welches für die Sortierung verwendet wird. Rankingattribute werden für alle Risikometriken der Tabelle risk_measure (3.4.2) hinzugefügt. Innerhalb einer Peer Group (siehe Abbildung 46) kann nach Jahresrendite und Sharpe Ratio sortiert werden. Die erlaubten Sortierungen sind auf die Attribute vom Figma Design beschränkt. In der Vergleichskomponente werden die Rankingattribute für die grüne Hinterlegung der besten Attribute verwendet (siehe Abbildung 38).

3.4.4 Test data

In jeder XML-Datei steht die Information für ein ETF. In dem Dataimporter Projekt gibt es XML-Dateien für sechs verschiedene ETFs. Das erschwert das Testen der Applikation. Vor allem im Bereich Pagination, Sortierung und Filterung im Frontend. Deshalb wurden weitere Testdaten in Form von SQL Einfügekommandos mit ChatGPT generiert. Allerdings gibt es ein Problem: Die Daten zu Rendite kommen von der Risk-Schnittstelle (2.6.1), welche echte ISIN als Identifikation benötigt.

Valide ISIN finden Die Klasse RiskService (2.6.1), welche für die Abfrage der Renditedaten genutzt wird, stellt keine Methoden zur Verfügung, um eine Liste von ETFs zu bekommen. Allerdings returniert die Methode InstrumentsApiClient.getInstrumentsUniverse() eine Liste von ETFs. Diese Liste wurde nach Einträgen mit mehr als 100 Renditedaten gefiltert und danach in eine Liste von ISINs mit einer Java Methode konvertiert, welche folgend angeführt wird:

Listing 10: Methode zur Findung valider ISINs

```
1  private final InstrumentsApiClient instrumentsApiClient;
2  public void testApiClients() {
3      ResponseEntity<List<ServiceBreakdownInstrument>> instruments2
          = instrumentsApiClient.getInstrumentsUniverse();
4      List<String> isins =
          instruments2.getBody().stream().map(ServiceBreakdownInstrument::getIs
5      List<String> filtered = new ArrayList<>();
6      for (String isin : isins) {
7          List<SpotPriceHistory> spotPriceHistoryList =
              getSpotPriceHistoryByIsin(isin);
8          if (spotPriceHistoryList == null ||
              spotPriceHistoryList.isEmpty()) continue;
9          int size =
              spotPriceHistoryList.get(0).getSpotPrices().size();
10         if (size > 100){
11             filtered.add(isin);
12         }
13     }
14 }
```

Generierung der Testdaten Zur Generierung der Testdaten wurde ChatGPT verwendet. Der Prompt (siehe Listing 11) beschreibt zuerst den generellen Kontext, wofür die Testdaten generiert werden sollen. Danach werden die Anzahl der Einfügekommandos pro Tabelle und weitere Instruktionen, welche die Anforderungen genauere spezifizieren erwähnt. Weiters werden die validen ISINs aufgelistet und zum Schluss ein Beispiel Einfügekommando definiert. Allerdings ist es häufig zu statistischen Fehlern gekommen. Zum Beispiel wurde ein Boolean bei einem Einfügekommando für die Tabelle Fund 3.4.2 vergessen. Deshalb wurde die Reihenfolge der Attribute im Einfügekommando geändert, sodass zwischen den Booleans keine numerischen Werte mehr platziert sind. Die fertigen Einfügekommandos wurden von ChatGPT in eine Datei namens *test_etf_data_inserts.sql* geschrieben. Nach jedem weiteren Versuch eine semantisch korrekte Datei zu bekommen, wurde von ChatGPT ein neuer Suffix angehängt: zuerst *_full*, danach *_final* und zum Schluss die funktionierende Version mit *_v2*. Der ChatGPT Prompt wird folgend verkürzt gezeigt. Die verkürzten Stellen sind an drei Punkten ... ersichtlich:

Listing 11: ChatGPT Prompt

```
I want to generate Test Data for ETFs. Please generate
   Insertcommands from the example structure below. Do not insert
   multiple rows with the same Insert. Use seperate ones.
Before inserting the funds, generate 10 Peer_Group later used for
   the Fund_Peer_Group assignments
Per fund should be the following quantities:
- 1 public.fund entry
  - Give it a realistic name (not Fund CH0002769351)
  - The strategy should be at least 10 Words long.
  - Be careful to provide the exact amount of booleans needed (27
     booleans)
...

Generate the above mentioned quantities for each of the following
   ISIN values in a downloadable file:
[AT0000A2B4T3, CH0000422433, ...]

Example Structure:
INSERT INTO public.fund ...
INSERT INTO public.holding ...
```

Migrations Error Das Erstellen von Tabellen und das hinzufügen von Testdaten findet in separaten Migrationen (2.1.1) statt. Zusätzliche Testdaten werden vom Dataimporter (1.4) eingefügt. Die Tabellen-Migrationen müssen vor dem Dataimporter angewendet werden, weil dieser sonst keine Daten einfügen kann. Die Testdaten-Migrationen müssen nach dem Dataimporter angewendet werden, weil diese auf die Testdaten des Dataimporters zugreifen. Das Problem ist, dass Flyway (2.1.1) immer alle Migrationen in dem *resources/db/migrations* Ordner anwendet. Deshalb müssen zuerst (1) die Testdaten-Migrationen aus dem Ordner entfernt. Dann (2) die Tabellen-Migrationen angewendet werden, indem das Backend gestartet wird. Danach (3) der Dataimporter gestartet werden. Zum Schluss muss (4) das Backend mit den Testdaten-Migrationen neugstartet werden.

4 Ergebnis

Anschließend sind die einzelnen Komponenten der vorliegenden Arbeit mit ihren Funktionalitäten beschrieben. Die Anwendung setzt sich zusammen aus einem Angular Frontend, Spring Boot Backend und einer PostgreSQL Datenbank.

4.1 Angular Frontend

Das Frontend ist die Schnittstelle zu den Anwendenden. Es verwendet die REST-API des Spring Boot Backends unter Benutzung von Angular Services (2.1.3), um auf die Geschäftslogik zuzugreifen. Auf der Hauptseite (3.2.5) sind generelle Statistiken zu ETFs zu sehen. Weiters kann nach ETFs in einem Schnellzugriff gesucht werden (siehe Abbildung 31) und vorsortierte Listen von ETFs werden angezeigt. Die Entdeckungskomponente (3.2.6) bietet eine tabellarische Sichtweise über alle vom Backend geladenen ETFs. Dabei wird nur ein Teil aller ETFs mithilfe von Pagination (2.1.1) angezeigt. Zusätzlich können die ETFs sortiert und nach verschiedenen Kriterien gefiltert (3.2.6) werden. Danach kann entweder ein ETF direkt angeklickt werden, um auf die Detailansichtskomponente (3.2.8) zu gelangen, oder mehrere ETFs ausgewählt werden und zuvor in der Vergleichskomponente (3.2.7) miteinander verglichen werden. Die beste Kennzahl aller ausgewählten ETFs wird in grün hinterlegt. Somit ist auf einen Blick ersichtlich, wenn ein ETF in vielen Metriken besser ausfällt als die anderen. Der erste Eindruck über die ETFs durch die Kennzahlen kann darunter mit einer Renditengrafkomponente (3.2.9) verfeinert werden. Nachdem eine passendes ETF gefunden wurde, kann auf die Detailansichtskomponente (3.2.8) unter Verwendung eines Buttons navigiert werden. Diese zeigt oberhalb die wichtigsten Metriken zum ETF. Darunter können verschiedene, detailliere Informationsansichten und weitere Visualisierungen und Statistiken in einem Menü ausgewählt werden. Die einzelnen Ansichten sind in der Detailansichtskomponente genauer beschrieben.

4.2 Spring Boot Backend

Das Spring Boot Backend implementiert wesentliche Funktionalitäten der Geschäftslogik und stellt diese über eine REST-API zur Verfügung. Dafür werden Daten vom Risk Service (2.6.1) und

der PostgreSQL Datenbank herangezogen. Im Zuge dieser Arbeit sind vor allem Erweiterungen für den Vergleich mehrerer ETFs (3.3.1) implementiert worden. Dafür wird die Stored Procedure *get_comparison_funds* (3.4.3) verwendet. Außerdem beinhalten essentielle Erweiterungen die Unterstützung von Pagination, Sortierung und Filterung (2.1.1) für den Listenzugriff auf ETFs.

4.3 PostgreSQL Datenbank

Die Kerndatenquelle des Projektes ist die PostgreSQL Datenbank. Das Schema ist mit zusätzlichen Attributen und Tabellen (3.4.1) erweitert worden. Außerdem sind zusätzlich Testdaten, ergänzend zu den vom Dataimporter (1.4) eingefügten, in der Datenbank mittels mehreren Einfügekommandos in einer SQL-Datei erzeugt worden. Die Datei wurde mithilfe von ChatGPT (3.4.4) generiert. Als spezielle Herausforderung hat sich das Finden valider ISINs entpuppt und es sind mehrere Anpassungen des Prompts notwendig gewesen, um fatale Fehler, wie zum Beispiel fehlende Attribute, von ChatGPT zu verhindern.

5 Resümee

Wir haben im Rahmen der Durchführung dieses Projektes viele neue Erkenntnisse gewonnen und wichtige Eindrücke sowie Vorgehensweisen im Bereich der Softwareentwicklung gesammelt. Viele Grundlagen waren durch den Unterricht an der HTL Perg bereits bekannt, konnten jedoch noch weiter vertieft werden. Dies betrifft alle Teilbereiche der entwickelten Applikation.

Angular - Frontend Bei der Entwicklung der Benutzeroberfläche wurden besonders viele neue Technologien eingesetzt. Insbesondere die Verwendung der Monorepo-Architektur mit der entsprechenden Aufteilung der Komponenten in verschiedene Verzeichnisse musste Anfangs richtig verstanden und umgesetzt werden, damit die Projektstruktur sowohl richtig als auch konsistent bleibt. Darüber hinaus war der Umgang mit PrimeNG, NGXS-Statemanagement, Signals und die Gestaltung mittels TailwindCSS neu.

Spring Boot - Backend & Dataimporter Die Implementierung von APIs in Java war grundsätzlich bereits bekannt. Allerdings mussten wir uns in den bestehenden Code einlesen und Spring-Boot-spezifische Elemente wie Annotationen verstehen und bei der anschließenden Erweiterung korrekt anwenden.

PostgreSQL - Datenbank Der Umgang mit relationalen Datenbanken war uns bereits vertraut. Allerdings mussten auch hier wichtige Besonderheiten berücksichtigt werden. Komplett neu war wiederum die Implementierung einer Stored Procedure.

Neben den technischen Herausforderungen stellte auch die Projektabwicklung und Planung in gewissen Zügen eine neue Erfahrung dar. Wir haben zum ersten Mal über eine vierwöchige Zeitspanne intensiv an einem Produkt gearbeitet und dabei einen Auftraggeber im Hintergrund gehabt. Die Kommunikation mit diesem war die wichtigste Erkenntnis aus diesem Projekt und hätte rückblickend verbessert werden können, um beispielsweise das Missverständnis hinsichtlich der angefertigten Designs zu vermeiden.

Glossar

API Application Programming Interface

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

DBMS Database Management System

DNS Domain Name System

DOM Document Object Model

DTO Data Transfer Object

ERD Entity Relationship Diagram

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

ISIN International Securities Identification Number

JDBC Java Database Connectivity

JSON Javascript Object Notation

MCP Model Context Protocol

REST Representational State Transfer

SEO Search Engine Optimization

SQL Structured Query Language

URI Uniform Ressource Identifier

XML Extensible Markup Language

Literaturverzeichnis

[1] S. Primetzhofer, „A case study on eliciting architecturally-significant requirements in the finance domain / Author Simon Primetzhofer, BSc," Master's thesis, JKU Linz, 2025.

[2] J. P. G. T. A. Hintersteininger, Armin; Oppitz, „ETF-APP," Master's thesis, HTL Perg, 2025.

[3] T. Collings, „Controller-Service-Repository," *Medium*, 2021. Online verfügbar: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

[4] T. Kelly, „Service Layer Pattern in Java With Spring Boot," *foojay.io*, 2025. Online verfügbar: https://foojay.io/today/service-layer-pattern-in-java-with-spring-boot/

[5] E. Elliott, „Mocking is a Code Smell," *Medium*, 2017.

[6] A. AWS. Was ist Boilerplate-Code? Online verfügbar: https://aws.amazon.com/de/what-is/boilerplate-code/

[7] O. Yılmaz, „The DTO Pattern (Data Transfer Objects)," *Medium*, 2023.

[8] C. Dictionary. Online verfügbar: https://dictionary.cambridge.org/dictionary/english/compiler

[9] J. Miller, „Patterns in Practice - Convention Over Configuration," *Microsoft Learn*, 2009. Online verfügbar: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-in-practice-convention-over-configuration

[10] Mapstruct. Online verfügbar: https://mapstruct.org/

[11] T. Bormans, „A guide to MapStruct with Spring Boot, Vavr Lombok." *Medium*, 2024. Online verfügbar: https://medium.com/@tijl.b/a-guide-to-mapstruct-with-spring-boot-vavr-lombok-d5325b436220

[12] R. C. Martin, *Agile Software Development*. Prentice Hall, 2003. Online verfügbar: https://books.google.at/books?id=0HYhAQAAIAAJ&redir_esc=y

[13] Thorben, „Dependency Injection Explained (With Examples)," *Stackify*, 2023. Online verfügbar: https://stackify.com/dependency-injection/

[14] sardar khan, „Understanding Dependency Injection: A Powerful Design Pattern for Flexible and Testable Code," *Medium*, 2023. Online verfügbar: https://medium.com/@sardar.khan299/understanding-dependency-injection-a-powerful-design-pattern-for-flexible-and-testable-code-5e1161dd37dd

[15] Spring. Online verfügbar: https://docs.spring.io/spring-framework/reference/core/beans/introduction.html

[16] ——. Online verfügbar: https://docs.spring.io/spring-framework/reference/core/beans/classpath-scanning.html

[17] ——. Online verfügbar: https://docs.spring.io/spring-framework/reference/core/beans/definition.html

[18] spring. Online verfügbar: https://docs.spring.io/spring-data/relational/reference/index.html

[19] Vishnuravichandran, „Understanding Pagination: A Guide for Developers," *Medium*, 2024. Online verfügbar: https://medium.com/@vishnuravichandran.28/understanding-pagination-a-guide-for-developers-dffe3d547d60

[20] spring. Online verfügbar: https://docs.spring.io/spring-data/relational/reference/jdbc/query-methods.html

[21] J. EE. Online verfügbar: https://jakarta.ee/learn/docs/jakartaee-tutorial/current/persist/persistence-intro/persistence-intro.html

[22] spring. Online verfügbar: https://docs.spring.io/spring-data/relational/reference/repositories/core-concepts.html

[23] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, UNIVERSITY OF CALIFORNIA, IRVINE, 2000. Online verfügbar: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf

[24] Flyway. Getting started with Flyway. Online verfügbar: https://documentation.red-gate.com/flyway/getting-started-with-flyway

[25] N. Fairuz, „Data Migration Fundamentals, but made relatable," *Medium*, 2025.

[26] J. Sixel, „Was sind ETFs? Exchange Traded Funds erklärt!" *finanzfluss.de*, 2023. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/etf/

[27] M. Thomaser, „ETF Replikationsmethoden im Vergleich," *finanzfluss.de*, 2025. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/replikationsmethoden/

[28] J. Sixel, „Ausschüttende oder thesaurierende ETFs?" *finanzfluss.de*, 2023. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/ausschuettend-oder-thesaurierend/

[29] J. Chen, S. Silberstein, und V. Velasquez, „Understanding ISIN: Definition, Uses, and Importance in Global Finance," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/i/isin.asp

[30] J. Chen, G. Scott, und T. Li, „Assets Under Management (AUM): Definition, Calculation, and Example," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/a/aum.asp

[31] A. Hayes, A. Drury, und M. Rosenston, „Total Expense Ratio (TER): Definition and How to Calculate," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/t/ter.asp

[32] S. Stoll, „Die wichtigsten Kennzahlen für Fonds und ETFs erklärt: Cockpit-Check für Anlege," *netfonds.de*, 2025. Online verfügbar: https://www.netfonds.de/blog/details/wichtigste-fonds-etf-kennzahlen-erklaert

[33] A. Hayes, C. Potters, und V. Velasquez, „Volatility: Meaning in Finance and How It Works With Stocks," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/v/volatility.asp

[34] J. Fernando, G. Scott, und K. Munichiello, „Sharpe Ratio: Definition, Formula, and Examples," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/s/sharperatio.asp

[35] Finanzfluss, „Fondsdomizil bei ETFs," *finanzfluss.de*, 2020. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/fondsdomizil/

[36] A. Bilandzija, „Währungsrisiken bei ETFs," *finanzfluss.de*, 2020. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/waehrungsrisiko/

[37] J. Chan und T. Brock, „Holdings: Definition in Investing and Their Role in Diversity," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/h/holdings.asp

[38] A. Hayes, T. J. Catalano, und A. Jackson, „Stock Symbol (Ticker Symbol): Abbreviation for a Company's Stock," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/s/stocksymbol.asp

[39] J. Arzadon. Active or passive ETFs: how do you decide? Online verfügbar: https://www.firstlinks.com.au/active-passive-etfs-strategies

[40] I. Team, G. Scott, und B. Petrick, „Environmental, Social, and Governance (ESG) Investing: What It Is How It Works," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/e/environmental-social-and-governance-esg-criteria.asp

[41] A. Hayes, „Understanding Peer Groups: Definition, Uses, Examples, Pros Cons," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/terms/p/peer-group.asp

[42] TradingView, „Inception date," *TradingView*. Online verfügbar: https://www.tradingview.com/support/solutions/43000748181-inception-date/

[43] E. Glossary, „ETF Issuer Definition," *etf.com*. Online verfügbar: https://www.etf.com/sections/etf-issuer-definition

[44] A. Hayes, G. Scott, und Y. Perez, „Understanding Maximum Drawdown (MDD): Key Insights and Formula," *Investopedia*. Online verfügbar: https://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp

[45] ETFGI, „Global ETF Assets Reach Record High of US$18.81 Trillion at end of September according to new research from ETFGI," *ETFGI*, 2025. Online verfügbar: https://etfgi.com/news/press-releases/2025/10/global-etf-assets-reach-record-high-us1881-trillion-end-september

[46] S. D. Simpson, J. Mansa, und J. Ma, „The Rise of Exchange-Traded Funds: A Historical Overview," *Investopedia*, 2025. Online verfügbar: https://www.investopedia.com/articles/exchangetradedfunds/12/brief-history-exchange-traded-funds.asp#toc-the-expansion-of-etfs-transforming-investment-strategies

[47] J. Sixel, „Die Vorteile von ETFs," *finanzfluss.de*, 2023. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/vorteile/

[48] ——, „Risiken beim Investieren in ETFs," *finanzfluss.de*, 2023. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/risiken/

[49] ——, „Tracking Error Tracking Differenz erklärt," *finanzfluss.de*, 2023. Online verfügbar: https://www.finanzfluss.de/etf-handbuch/tracking-error-tracking-difference/

[50] I. Atamanchuk, H. Qureshi, und J. Weinberg, „A Close Look at Exchange-Traded Funds and Their Investors," Investment Company Institute, Tech. Rep. 7, Sep. 2025, Accessed: 2026-03-21. Online verfügbar: https://www.ici.org/system/files/2025-09/per31-07.pdf

[51] Mozilla. Document Object Model (DOM). Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[52] Angular. Using DOM APIs. Online verfügbar: https://angular.dev/guide/components/dom-apis

[53] S. Peiris. What is Renderer2 in Angular? Online verfügbar: https://medium.com/@saranipeiris17/directly-accessing-dom-elements-directly-accessing-the-dom-involves-using-native-javascript-methods-6c91ed3a933d

[54] Angular. Components. Online verfügbar: https://angular.dev/essentials/components

[55] ——. Components. Online verfügbar: https://angular.dev/guide/components

[56] ——. Accepting data with input properties. Online verfügbar: https://angular.dev/guide/components/inputs

[57] ——. Custom events with outputs. Online verfügbar: https://angular.dev/guide/components/outputs

[58] ——. Component Lifecycle. Online verfügbar: https://angular.dev/guide/components/lifecycle

[59] ——. Creating and using services. Online verfügbar: https://angular.dev/guide/di/creating-and-using-services

[60] ——. Dependency injection in Angular. Online verfügbar: https://angular.dev/guide/di

[61] ——. Angular Routing. Online verfügbar: https://angular.dev/guide/routing#

[62] ——. Define routes. Online verfügbar: https://angular.dev/guide/routing/define-routes

[63] S. Tankala. Skeleton Screens 101. Online verfügbar: https://www.nngroup.com/articles/skeleton-screens/

[64] F. Prodromou. All you need to know to jumpstart with NGXS. Online verfügbar: https://angular.love/all-you-need-to-know-to-jumpstart-with-ngxs

[65] NGXS. STATE. Online verfügbar: https://www.ngxs.io/concepts/state

[66] ——. WHY. Online verfügbar: https://www.ngxs.io/introduction/why

[67] RXJS. Introduction. Online verfügbar: https://rxjs.dev/guide/overview

[68] Angular. Signals. Online verfügbar: https://angular.dev/essentials/signals

[69] S. Alam. What is Zone.js in Angular? Online verfügbar: https://medium.com/@sehban.alam/what-is-zone-js-in-angular-e0029c21c32f

[70] Angular. Content projection with ng-content. Online verfügbar: https://angular.dev/guide/components/content-projection

[71] npm. About npm. Online verfügbar: https://docs.npmjs.com/about-npm

[72] W. Commons, „File:Npm-logo.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Npm-logo.svg&oldid=1054332928

[73] Angular. What is Angular? Online verfügbar: https://angular.dev/overview

[74] ——. Installation. Online verfügbar: https://angular.dev/installation

[75] ——. The Angular CLI. Online verfügbar: https://angular.dev/tools/cli

[76] ——. The Open-Source GitHub-Repository. Online verfügbar: https://github.com/angular/angular

[77] W. Commons, „File:Angular wordmark gradient.png — Wikimedia Commons, the free media repository," 2025, [Online; accessed 21-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Angular_wordmark_gradient.png&oldid=1054352181

[78] NodeJS. Introduction to Node.js. Online verfügbar: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[79] W. Commons, „File:Node.js logo.svg — Wikimedia Commons, the free media repository," 2026, [Online; accessed 18-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Node.js_logo.svg&oldid=1151644190

[80] S. Sasindu. Monorepo Architecture: A Complete Guide for Modern Development. Online verfügbar: https://medium.com/@sasindusathiska/monorepo-architecture-a-complete-guide-for-modern-development-828584f05534

[81] Nx. What is a monorepo? Online verfügbar: https://monorepo.tools/#what-is-a-monorepo

[82] ——. What is Nx? Online verfügbar: https://nx.dev/docs/getting-started/intro

[83] ——. Add to an Existing Project. Online verfügbar: https://nx.dev/docs/getting-started/start-with-existing-project

[84] N. C. Team. NX Logo. Online verfügbar: https://miro.medium.com/v2/resize:fit:720/format:webp/0*8tu6dgB0zeyiz-vo.png

[85] Yarn. Introduction. Online verfügbar: https://yarnpkg.com/getting-started

[86] T. C. N. Experts. Yarn vs. NPM: Which Package Manager You Should Choose, and Why? Online verfügbar: https://www.aquasec.com/cloud-native-academy/supply-chain-security/yarn-vs-npm/

[87] Yarn. Workspaces. Online verfügbar: https://yarnpkg.com/features/workspaces

[88] W. Commons, „File:Yarn-logo-kitten.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 21-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Yarn-logo-kitten.svg&oldid=1051923643

[89] H. Europe. SCSS – Was kann die praktische Stylesheet-Sprache? Online verfügbar: https://www.hosteurope.de/blog/scss-was-kann-die-praktische-stylesheet-sprache/

[90] W. Commons, „File:Sass Logo Color.svg — Wikimedia Commons, the free media repository," 2020, [Online; accessed 21-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Sass_Logo_Color.svg&oldid=498144031

[91] H. S. Nair. Introduction to Tailwind CSS. Online verfügbar: https://www.geeksforgeeks.org/css/introduction-to-tailwind-css/

[92] F. Deinhard. Was ist Tailwind CSS? Online verfügbar: https://www.it-schulungen.com/wir-ueber-uns/wissensblog/was-ist-tailwind-css.html

[93] W. Commons, „File:Tailwind CSS logo with dark text.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 21-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:

Tailwind_CSS_logo_with_dark_text.svg&oldid=1104128075

[94] luthfurc, „What is a framework ?" *Reddit*, 2020. Online verfügbar: https://www.reddit.com/r/learnprogramming/comments/i63bmd/what_is_a_framework/

[95] Jetbrains. Online verfügbar: https://www.jetbrains.com/lp/devecosystem-2023/java/

[96] I. Pivotal Software. Online verfügbar: https://spring.io/img/spring.svg

[97] What is open source? Online verfügbar: https://dspace.htl-perg.ac.at/handle/htl-perg/1933

[98] What Is a Database Management System (DBMS)? Online verfügbar: https://neo4j.com/news/what-is-a-database-management-system-dbms

[99] Oracle. (2021) What Is a Relational Database? (RDBMS)?

[100] Keshav, „PostgreSQL for Everyone: Introduction, Features Architecture (Part 1)," *Towards Dev - Medium*, 2024. Online verfügbar: https://towardsdev.com/postgresql-for-everyone-introduction-features-architecture-part-1-37838cf06bc7

[101] Postgres. Online verfügbar: https://www.postgresql.org/docs/current/extend-how.html

[102] ——. Online verfügbar: https://www.postgresql.org/docs/current/xfunc.html

[103] ——. Online verfügbar: https://www.postgresql.org/docs/current/xfunc-internal.html

[104] ——. Online verfügbar: https://www.postgresql.org/docs/current/xfunc-c.html

[105] ——. Online verfügbar: https://www.postgresql.org/docs/current/xplang.html

[106] R. Kumar, „PostgreSQL - Introduction to Stored Procedures," *Geeks for Geeks*, 2025. Online verfügbar: https://www.geeksforgeeks.org/postgresql/postgresql-introduction-to-stored-procedures/

[107] I. Redaktion, „SQL STORED PROCEDURE – So funktioniert Automatisierung mit SQL," *IONOS*, 2024. Online verfügbar: https://www.ionos.at/digitalguide/server/konfiguration/sql-stored-procedure/

[108] Jetbrains. Online verfügbar: https://www.jetbrains.com/intellij/features/

[109] ——. Online verfügbar: https://www.jetbrains.com/webstorm/features/

[110] W. Commons, „File:WebStorm Icon.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:WebStorm_Icon.svg&oldid=1008636300

[111] ——, „File:JetBrains WebStorm Product Icon.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:JetBrains_WebStorm_Product_Icon.svg&oldid=1008693722

[112] Figma. Online verfügbar: https://www.figma.com/

[113] W. Commons, „File:Figma-logo.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Figma-logo.svg&oldid=1056524841

[114] Git. Online verfügbar: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[115] ——. Online verfügbar: https://github.com/resources/articles/what-are-code-repositories

[116] W. Commons, „File:Git-logo.svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Git-logo.svg&oldid=1046522439

[117] M. Artis, „What is containerization?" *Github Articles*, July 29, 2024. Online verfügbar: https://github.com/resources/articles/containerization

[118] Docker. Online verfügbar: https://docs.docker.com/get-started/docker-overview/

[119] W. Commons, „File:Docker logo (until 2025).png — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Docker_logo_(until_2025).png&oldid=1117282987

[120] C. D. Tools. Online verfügbar: https://developer.chrome.com/docs/devtools/dom?hl=de

[121] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/console?hl=de

[122] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/network/overview?hl=de

[123] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/performance/overview?hl=de

[124] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/lighthouse?hl=de

[125] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/application?hl=de

[126] ——. Online verfügbar: https://developer.chrome.com/docs/devtools

[127] ——. Online verfügbar: https://developer.chrome.com/docs/devtools/overview

[128] ——. Online verfügbar: https://github.com/ChromeDevTools/devtools-logo

[129] Chromium. Online verfügbar: https://www.chromium.org/Home/

[130] W. Commons, „File:Chromium logo with wordmark.png — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Chromium_logo_with_wordmark.png&oldid=1115266330

[131] Clockify. (2025) Clockify – Zeiterfassung und Projektmanagement. Online verfügbar: https://clockify.me/de/

[132] W. Commons, „File:Clockify.png — Wikimedia Commons, the free media repository," 2024, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Clockify.png&oldid=862259349

[133] Discord. Discord. Online verfügbar: https://discord.com/

[134] Devxhub, „Tips and Tricks for Efficient Project Management with Discord," *Medium*, 2023. Online verfügbar: https://devxhub.medium.com/tips-and-tricks-for-efficient-project-management-with-discord-38a6a8c34589

[135] W. Commons, „File:Discord Color Text Logo (2015-2021).svg — Wikimedia Commons, the free media repository," 2025, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Discord_Color_Text_Logo_(2015-2021).svg&oldid=1016342730

[136] Rocket.Chat. Rocket.Chat. Online verfügbar: https://de.rocket.chat/

[137] W. Commons, „File:Rocket.Chat Logo.svg — Wikimedia Commons, the free media repository," 2020, [Online; accessed 21-March-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Rocket.Chat_Logo.svg&oldid=503434535

[138] Overleaf. Overleaf. Online verfügbar: https://www.overleaf.com/about/features-overview

[139] W. Commons, „File:Overleaf Logo.svg — Wikimedia Commons, the free media repository," 2024, [Online; accessed 18-February-2026]. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Overleaf_Logo.svg&oldid=973265315

[140] PrimeNG. PrimeNG. Online verfügbar: https://primeng.org/

[141] ——. AutoComplete. Online verfügbar: https://primeng.org/autocomplete

[142] ——. Checkbox. Online verfügbar: https://primeng.org/checkbox

[143] ——. IconField. Online verfügbar: https://primeng.org/iconfield

[144] ——. InputText. Online verfügbar: https://primeng.org/inputtext

[145] ——. Select. Online verfügbar: https://primeng.org/select

[146] ——. MultiSelect. Online verfügbar: https://primeng.org/multiselect

[147] ——. Slider. Online verfügbar: https://primeng.org/slider

[148] ——. Button. Online verfügbar: https://primeng.org/button

[149] ——. Paginator. Online verfügbar: https://primeng.org/paginator

[150] ——. Divider. Online verfügbar: https://primeng.org/divider

[151] ——. Popover. Online verfügbar: https://primeng.org/popover

[152] ——. Breadcrumb. Online verfügbar: https://primeng.org/breadcrumb

[153] ——. Charts. Online verfügbar: https://primeng.org/chart

[154] Chart.js. Chart JS Docs. Online verfügbar: https://www.chartjs.org/docs/latest

[155] PrimeNG. Skeleton. Online verfügbar: https://primeng.org/skeleton

[156] ——. Toast. Online verfügbar: https://primeng.org/toast

[157] ——. Icons. Online verfügbar: https://primeng.org/icons

[158] D. K. Team. When to Use SVGs in Modern Web Design. Online verfügbar: https://www.webbb.ai/blog/when-to-use-svgs-in-modern-web-design

[159] PrimeNG. Configuration. Online verfügbar: https://primeng.org/configuration

[160] ——. Styled Mode. Online verfügbar: https://primeng.org/theming/styled

[161] nx. Nx Console. Online verfügbar: https://nx.dev/docs/guides/nx-console

[162] ——. Explore your Workspace. Online verfügbar: https://nx.dev/docs/features/explore-graph

[163] ngx translate. Repository: @ngx-translate/core. Online verfügbar: https://www.npmjs.com/package/@ngx-translate/core

[164] ——. Loading Translation Files. Online verfügbar: https://ngx-translate.org/getting-started/translation-files/

[165] ——. ngx-translate TranslateService API Reference. Online verfügbar: https://ngx-translate.org/reference/translate-service-api/

[166] ——. How to Translate Angular Components with ngx-translate. Online verfügbar: https://ngx-translate.org/getting-started/translating-your-components/

[167] PrimeNG. What is Prettier? Online verfügbar: https://prettier.io/docs/

[168] ESLint. Getting Started with ESLint. Online verfügbar: https://eslint.org/docs/latest/use/getting-started

[169] ——. Rules Reference. Online verfügbar: https://eslint.org/docs/latest/rules

[170] Prettier. Prettier vs. Linters. Online verfügbar: https://prettier.io/docs/comparison

[171] Spring. Online verfügbar: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[172] O. Feign. Online verfügbar: https://github.com/OpenFeign/feign?tab=readme-ov-file

[173] Jetbrains. Online verfügbar: https://www.jetbrains.com/help/idea/working-with-artifacts.html

[174] Angular. End to End Testing. Online verfügbar: https://angular.dev/tools/cli/end-to-end

[175] A. Bone, „Future of CSS: Select styling without the hacks," *dev*, 2025.

[176] Spring. Online verfügbar: https://docs.spring.io/spring-data/relational/reference/jdbc/query-methods.html

[177] ——. Online verfügbar: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/NamedParameterJdbcTemplate.html

[178] Kango, „Why are there no Composite Keys in Spring Data JDBC?" *Reddit*, 2024. Online verfügbar: https://www.reddit.com/r/SpringBoot/comments/1ehhfgc/why_no_composite_keys_in_spring_data_jdbc/

[179] J. Schauder, „Spring Data JDBC and R2DBC 4.0 will support Composite IDs," *Spring Blog*, 2026. Online verfügbar: https://spring.io/blog/2025/07/22/spring-data-jdbc-composite-id

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Projektorganisation

Die Organisation ist in Projekten dieser Größenordnung sehr wichtig, um ein gutes Ergebnis zu erzielen.

A.1 Projektverlauf

Bereits in einem seiner früheren Praktika bei der uni software plus GmbH (1.2.3) hat Michael Gillhofer mit den Unternehmensverantwortlichen über eine Möglichkeit zur gemeinsamen Umsetzung einer Diplomarbeit gesprochen und dabei eine Zusicherung erhalten.

Ende November 2024 erhielt das Projektteam dann die endgültige Zusage und einen groben Themenüberblick. Dieser wurde durch die Ausarbeitung des Kooperationsformulars und des Projektantrags konkreter, und Fragen konnten im ersten gemeinsamen Meeting Anfang Februar 2025 geklärt werden. Mit der Implementierung ging auch ein vierwöchiges Praktikum einher, im Zuge dessen die gesamte Programmierarbeit vollbracht werden konnte. Die Arbeitszeit beläuft sich daher auf etwa 308 Stunden. (38.5 Wochenstunden * 4 Wochen * 2 Personen) Durch wöchentliche Updates wurde auch Prof. Mag. Holzmann als Betreuer (1.2.2) auf dem Laufenden gehalten.

Vor und nach dem Praktikum erfolgt die Kommunikation mit dem Auftraggeber über E-Mail und Discord (2.4.3). Vor allem erwies sich Discord als praktisches Tool, da es eine schnellere und unkompliziertere Möglichkeit zum Austausch von Text- und Dateiinhalten bietet als ein klassischer E-Mail-Verkehr. Der Betreuer erhielt Fragen und Anmerkungen über E-Mail oder persönlich im Rahmen regulärer Unterrichtseinheiten.

A.2 Meilensteine

Folgende Meilensteine wurden für die Diplomarbeit definiert:

Tabelle 2: Meilensteine und geplante Erreichungsdaten

Meilenstein	**Datum**	**Name**
Arbeitsgerät ist aufgesetzt, Zugriff auf den Prototypen ist gewährleistet und die Software zum Ausführen des Prototyps ist installiert.	08.07.2025	Lukas Leitner und Michael Gillhofer
Frontendbasis ist implementiert (Unterteilung des Figmas in Angular Komponenten; Globale Styles, wie Font Sizes, Font Scale, Spacing System, Naming Convention, PrimeNG Theme).	11.07.2025	Lukas Leitner
Schema der Datenbank ist erweitert und Testdaten sind in die Datenbank eingefügt.	11.07.2025	Michael Gillhofer

Meilenstein	Datum	Name
Die Endpunkte für die Fund Comparison Funktionalität sowie Erweiterungen für Filterung, Pagination und Sortierung einer Liste von ETFs sind im Backend implementiert und getestet.	13.07.2025	Michael Gillhofer
Vergleich von ETFs basierend auf Kennzahlen ist im Frontend implementiert.	14.07.2025	Michael Gillhofer
Anzeige einer Liste von ETFs mit Unterstützung für Filterung, Pagination und Sortierung ist im Frontend implementiert.	17.07.2025	Michael Gillhofer
Detailansicht zu einem bestimmten ETF ist im Frontend implementiert.	20.07.2025	Lukas Leitner
Navigationsleiste, Footer und Startseite sind implementiert.	24.07.2025	Lukas Leitner
Quality-of-Life-Verbesserungen im Frontend sind implementiert (z. B. Ladeanimationen, Mehrsprachenunterstützung).	26.07.2025	Lukas Leitner und Michael Gillhofer
Code Review im Frontend und Backend ist umgesetzt (z. B. Linting, Signals, Stored Procedures, ...).	30.07.2025	Lukas Leitner und Michael Gillhofer
Die Applikation ist manuell getestet und eventuelle Bugs sind behoben.	30.07.2025	Lukas Leitner und Michael Gillhofer

B Aufgabenverteilung

In den beiden nachfolgenden Inhaltsverzeichnissen werden die verfassten Kapitel der Arbeit pro Projektmitglied aufgelistet.

B.1 Inhaltsverzeichnis Leitner Lukas

B.2 Inhaltsverzeichnis Gillhofer Michael

C Plakat

Abbildung 49: Plakat

D Logo

Abbildung 50: Logo