



WESTSÄCHSISCHE HOCHSCHULE ZWICKAU
STUDIENGANG INFORMATIK

BACHELORARBEIT

Entwicklung einer Service-Struktur zur Suche, Filterung und Abfrage von ETF- und Fondsdaten

Tommy-Lee Bannert
Matrikel: 40760

Wintersemester 2023/24

Betreuer	Einrichtung
Prof. Dr. Frank Grimm	Westsächsische Hochschule Zwickau
Prof. Dr. Wolfgang Golubski	Westsächsische Hochschule Zwickau
Jan Zimmermann	flatexDEGIRO AG

5. Januar 2024

Abstract

In dieser Bachelorarbeit wird die Entwicklung einer Service-Struktur zur Suche, Filterung und Abfrage von ETF- und Fondsdaten vorgestellt. Der Fokus liegt auf der Modernisierung einer bestehenden Anwendung, ursprünglich in Go entwickelt, durch die Überführung in das moderne Quarkus-Framework. Ziel ist es, die Datenverarbeitung und -bereitstellung zu optimieren und manuelle Prozesse, durch die Verwendung von Schnittstellen, zu automatisieren. Es werden die Herausforderungen und Einschränkungen des bestehenden Services analysiert und eine Lösung zur Verbesserung der Effizienz und Performance durch die Anwendung von Quarkus, Hibernate und anderen Technologien vorgeschlagen. Die Arbeit beleuchtet die Implementierung und Bewertung der neuen Service-Struktur, die eine effektivere Handhabung und Analyse von ETF- und Fondsdaten ermöglicht.

Inhaltsverzeichnis

1 Einleitung	5
1.1 Motivation	5
1.2 Ziele	5
1.3 Vorgehensweise	6
2 IST-Analyse	7
2.1 Analyse der bestehenden Datenquellen	7
2.1.1 Morningstar	7
2.1.1.1 Morningstar Direct	7
2.1.2 ENTAX/Rp312	9
2.1.3 InfoZone	9
2.2 Bestehende Prozesse	9
2.2.1 Export der Rohdaten	9
2.2.2 Anpassung der Rohdaten	9
2.2.3 Übertragung zum Service	9
2.3 Bestehender Service	10
2.3.1 Bestehende Endpunkte	10
2.3.2 Struktur, Code-Qualität und Dokumentation	12
2.3.3 Wartbarkeit	12
2.4 Probleme des bestehenden Services	13
2.5 Oberfläche	15
2.5.1 Desktop Ansicht	15
2.5.2 App Ansicht	16
3 SOLL-Analyse	18
3.1 Initiale Anforderungen des Unternehmens	18
3.2 Optimierung bestehender Datenquellen	18
3.2.1 Morningstar Direct	19
3.3 Integration neuer Datenquellen	19
3.3.1 Morningstar Top 10 Holdings	19
3.4 Entwürfe	19
4 Anforderungen	22
4.1 Funktionale Anforderungen	23
4.2 Nicht-funktionale Anforderungen	24
5 Konzept	25
5.1 Architektur	25
5.2 Vorteile der Trennung	25
5.3 Fund Management Suite	27

5.3.1	Fund Commons	27
5.3.2	Fund Retrieval Service	27
5.3.2.1	Verantwortlichkeiten	27
5.3.2.2	Entwicklungs- und Änderungsprozess	28
5.3.2.3	Bedeutung und Vorteile	28
5.3.3	Fund Discovery Service	28
5.3.3.1	Verantwortlichkeiten	28
5.3.3.2	Entwicklungs- und Änderungsprozess	28
5.3.3.3	Bedeutung und Vorteile	29
5.3.4	Fund Discovery Service API	29
5.4	Datenverarbeitung und -mapping	30
5.5	Datenbankentscheidungen	30
5.6	Volltextsuche und Dynamische Filterung	30
5.7	Datenbank	30
5.8	Automatisierte Tests mit Mocks	32
6	Verwendete Technologien	33
6.1	Quarkus	33
6.1.1	Live Coding und Hot-Reloading	33
6.1.2	Continuous Testing	34
6.1.3	Fokus auf Cloud- und Microservice-Umgebungen	34
6.1.4	Unterstützung und Integrationen	34
6.1.5	Konfigurationsverwaltung	35
6.1.6	Dev Services	35
6.1.7	GraalVM und Quarkus	35
6.1.7.1	Vergleich von JVM mit nativer Kompilierung	35
6.1.8	Vergleich zu Alternativen wie Spring-Framework	35
6.2	Hibernate	36
6.2.1	Hibernate ORM	36
6.2.1.1	Panache	36
6.2.2	Hibernate Search	37
6.2.2.1	Elasticsearch	37
6.2.3	Hibernate Validator	37
6.2.4	Hibernate Envers	37
7	Methodik	38
7.1	Prozesse und Technologien für die Entwicklung	38
7.1.1	Semantic Versioning	38
7.1.2	Conventional Commits	38
7.1.3	Git-Hooks	38
7.2	Entwicklungsumgebung	39
8	Implementierung	40
8.1	Erzeugen der Projekt-Strukturen	40
8.2	Entwicklungs- und Anpassungsprozesse	41
8.3	Fund Retrieval Service (FRS)	41
8.3.1	Datenvalidierung und -management	41
8.3.1.1	Datenformat-Handling	41
8.3.1.2	Datenvalidierungsframework	41

8.3.2	Datenprozesse	42
8.3.2.1	CSV-Import von Morningstar Direct	42
9	Ergebnisse	43
9.1	Bewertung der Implementierung	43
9.2	Probleme und Lösungen	43
9.2.1	Syntaxfehler CSV Zeilenumbrüche	43
9.2.2	Proxy und SSL-Zertifikate	43
9.2.3	Ausfall von CI/CD-Pipeline-Runners	43
9.3	Vergleich mit den Anforderungen	44
10	Auswertung	45
11	Ausblick	46
12	Schlussfolgerung	47
	Selbstständigkeitserklärung	48
	Literaturverzeichnis	49
	Abkürzungsverzeichnis	50
	Abbildungsverzeichnis	51
	Tabellenverzeichnis	51

Kapitel 1

Einleitung

Angesichts aktueller globaler Konflikte wie dem Ukraine-Krieg, den Spannungen um Taiwan und Israel sowie dem zunehmenden Einkommensgefälle, die weltweit zu finanzieller Unsicherheit beitragen, gewinnen private Investitionsmöglichkeiten zunehmend an Bedeutung. Besonders die in den 1990er-Jahren eingeführten börsengehandelten Fonds (ETFs) wurden als Instrumente für Privatanleger entwickelt, die passiv und indexorientiert investieren wollen. Im Vergleich zu traditionellen offenen Fonds bieten ETFs mehrere Vorteile, insbesondere in Bezug auf die Flexibilität des Handels. Sie verfügen über eine hohe Liquidität und sind häufig kosteneffizient strukturiert. Damit erleichtern ETFs den Aufbau diversifizierter Portfolios mit geringen Investitionssummen. Dadurch sind ETFs insbesondere für Privatanleger verschiedener Einkommensgruppen von Bedeutung. Die immer weiter zunehmende Popularität von ETFs spiegelt diese Vorteile wider. Die Vorteile von ETFs gegenüber anderen Finanzprodukten sind besonders für Neobroker von Bedeutung, da sie die Kundenzielgruppe deutlich erweitern und als einfacher und risikoarmer Einstiegspunkt für neue Kund:innen dienen. [8, S. 117] Diese Bachelorarbeit befasst sich mit der Entwicklung einer Service-Struktur zur Suche, Filterung und Abfrage von ETF- und Fondsdaten für den Neobroker flatexDEGIRO. Dabei soll deren bestehende Anwendung durch eine Neuimplementierung in Java 17 und dem Quarkus-Framework manuelle Prozesse automatisieren und optimieren. Kunden soll der Einstieg in die Welt der Finanzprodukte so einfach wie möglich gemacht werden. Die Zufriedenheit der Kund:innen soll durch neue innovative Features und ein skalierbares, ausfallsicheres System gesteigert werden. Damit wird zudem ein Ausbau des Markteinflusses angestrebt.

1.1 Motivation

Vorstellung des Themas

1.2 Ziele

Ziel dieser Arbeit ist es, die Anwendung zu modernisieren und alte Funktionalitäten in das neue Framework zu implementieren. Es wird damit eine Optimierung der Datenverarbeitung, der Datenbereitstellung, der Datenstruktur, der Datenabfrage und Datenfilterung angestrebt.

- Überführen der alten Funktionalität in das neue Framework

- Optimierung der Datenverarbeitung
- Optimierung der Datenbereitstellung
- Optimierung der Datenstruktur
- Optimierung der Datenabfrage
- Optimierung der Datenfilterung

1.3 Vorgehensweise

Die Entwicklung umfasste eine klassische Anforderungsanalyse mit IST- und SOLL-Analyse. Die Ergebnisse dieser Analyse wurden als funktionale und nicht funktionale Anforderungen definiert. Auf Basis der Anforderungen wurde ein Konzept entwickelt und dieses anschließend implementiert.

Kapitel 2

IST-Analyse

2.1 Analyse der bestehenden Datenquellen

2.1.1 Morningstar

Diese Informationen wurden aus den Vertragsunterlagen mit Morningstar vom 21. September 2023 entnommen. Die Unterlagen wurden für die Neugestaltung des ETF-Search-Service und den damit verbundenen Ersatz der manuell erstellten Comma-separated values (CSV)-Datei bereitgestellt und sind daher geschützt. Laut den bestehenden Verträgen mit Morningstar ist die Datenabfrage auf die Graphical User Interface (GUI)-Anwendung Morningstar Direct beschränkt.

2.1.1.1 Morningstar Direct

Die Daten können als Datei in den Formaten Excel, CSV, Tab Delimited oder Extensible Markup Language (XML) exportiert werden. Diese Datei kann auf einem (S)File Transfer Protocol (FTP)-Server abgelegt oder per E-Mail versendet werden. Aufgrund der vertraglichen Bedingungen dürfen Daten von bis zu 8.000 verschiedenen ISINs pro Jahr und die Environmentail, Social and Governance (ESG)-Daten von bis zu 400 verschiedenen ISINs pro Jahr exportiert werden.

- Automatische Exporte können in Morningstar Direct erstellt werden.
- Die Anwendung ermöglicht Dateieporte in Excel, CSV, Tab Delimited und XML-Formaten.
- Die exportierte Datei kann entweder direkt auf dem lokalen Rechner gespeichert, auf einem konfigurierbaren (S)FTP-Server gespeichert oder per E-Mail gesendet werden.
- Das Dateiformat, zusätzliche Definitionen des Dateiformats, z. B. das Trennzeichen in CSV, das Intervall und das Ziel können konfiguriert werden.
- Um einen (S)FTP-Server als Exportziel zu nutzen, müssen die Netzwerkadresse, der Benutzername und das Passwort eingegeben werden.
- Export auf lokalen Speicher oder Versand per E-Mail ist für eine zukunftssichere Automatisierung nicht praktikabel.

Derzeit ist nicht bekannt, wie und von wo der automatische Export ausgeführt wird. Der automatische Export könnte auf verschiedene Weisen ausgeführt werden:

- ausschließlich durch den Rechner, auf dem Morningstar Direct läuft
- ausschließlich durch eine externe Entität
 - der konfigurierte automatische Export könnte auf einer externen Entität gespeichert werden
 - die externe Entität könnte die konfigurierten automatischen Exporte ausführen
 - dies könnte erfolgen, um die Ausführung auch dann zu gewährleisten, wenn Morningstar Direct nicht verfügbar oder in Betrieb ist
- eine Kombination aus beidem

Jedes dieser Szenarien birgt erhebliche Sicherheitsrisiken. Wenn der automatische Export durch eine externe Entität ausgeführt würde, kann die Integrität der exportierten Daten nicht garantiert werden, da

1. die externe Entität nicht auf böswillige Aktivitäten überwacht werden kann,
2. kein direkter Zugriff auf die externe Entität besteht,
3. böswillige Aktivitäten nicht erkannt und daher nicht darauf reagiert werden kann, wie
 - (a) Kompromittierung der Zugangsdaten für den (S)FTP-Server,
 - (b) Hochladen von böswilligen Daten oder Dateien auf den (S)FTP-Server ohne Validierung
4. keine vertraglich bindenden Details für die externe Entität existieren,
5. die Daten möglicherweise bereits manipuliert wurden, bevor sie auf den (S)FTP-Server hochgeladen werden,
6. die Daten zwischen dem Hochladen auf den (S)FTP-Server und dem Import durch den ETF-Suchdienst manipuliert werden könnten

Sollte der automatische Export durch einen lokalen Client ausgeführt werden, würden dennoch Unternehmensprozesse und -richtlinien verletzt, wie

1. die Beschaffung externer Daten muss über konforme Schnittstellen erfolgen,
2. alle Anwendungen im Unternehmen müssen
 - (a) validierbar,
 - (b) genehmigt,
 - (c) dokumentiert,
 - (d) zugangskontrolliert,
 - (e) versionkontrolliert sein
3. alle geschäftskritischen Anwendungen im Unternehmen erfordern
 - (a) testbar in automatisierter Weise zu sein,
 - (b) erfolgreich durch alle Bereitstellungsumgebungen zu gelangen

2.1.2 ENTAX/Rp312

Es werden die Daten aus ENTAX, konkret dem Rp312 benötigt. Die Daten sind aufgeteilt in flatex und flatex-at. Die relevanten Daten sind sparplanfähig und "handelbar bis". Es wurde scheinbar bereits eine Schnittstelle für den Export dieser Daten eingerichtet, jedoch wurde diese noch nicht getestet.

2.1.3 InfoZone

Die Bezeichnungen, aktuellen Handelspreise und die Preishistorie der Assets sollen anhand der ISIN von InfoZone abgerufen werden.

2.2 Bestehende Prozesse

Derzeit wird der ETF-Search-Service stark von manuellen Prozessen geprägt.

2.2.1 Export der Rohdaten

Die Rohdaten werden aus Morningstar Direct als CSV-Datei exportiert

Delimiter-Zeichen	Semikolon (;)
Zeichenkodierung (Character Encoding)	UTF-8 mit BOM
Textqualifikator (Text Qualifier)	(")
Escape-Zeichen	
Datumsformate	
Dezimaltrennzeichen	(,)
Zeilenendezeichen (Line Ending Character)	Semikolon (;)
Nullwerte	Semikolon (;)

Tabelle 2.1: Charakteristiken der CSV-Datei

2.2.2 Anpassung der Rohdaten

Anschließend wird die exportierte Datei in die vorhandene Excel-Datei eingefügt. Neben der bereits eingefügten CSV-Datei enthält diese Excel-Datei fünf zusätzliche Tabellen, die den Inhalt der Excel-Dateien anpassen und um zusätzliche Informationen ergänzen. Anpassungen an diesen Daten werden basierend auf den Konfigurationen der Zuordnungstabellen (siehe Tabelle) vorgenommen und erfordern zusätzliche manuelle Eingriffe.

Nach Abschluss dieses Prozesses werden die Daten als CSV-Dateien exportiert. Die CSV-Dateien repräsentieren die Länder Deutschland und Österreich.

2.2.3 Übertragung zum Service

Die angepasste CSV-Datei wird nun einem neu erstellten Service-Ticket in JIRA angehängt. Der Entwickler des ETF-Search-Services wird als Empfänger dieses Tickets gesetzt. Der Empfänger soll den Anhang überprüfen und die Abteilung IT-Infrastruktur als neuen

Tabelle 2.2: Mapping-Tabellen

Bezeichnung	Quelle	Aktualisierungsrate
Assetklasse & Branche	Morningstar	wird von Morningstar aktualisiert, aber bisher als statisch betrachtet; sollte ca. einmal im Jahr überprüft werden
Region & Länder	Morningstar	wird von Morningstar aktualisiert, aber bisher als statisch betrachtet; sollte ca. einmal im Jahr überprüft werden
Sparplanfähigkeit	ENTAX/Rp312	TBD
Currency	intern	TODO
Trades 12M	Business Intelligence	täglich, wöchentlich, monatlich

Empfänger setzen. Zuvor werden die benötigten ‘curl’-Befehle, mit denen die angehängten Daten an den Service gesendet werden, in die Beschreibung des Service-Tickets eingefügt. Ein Mitarbeiter der Abteilung führt diese Befehle aus und die CSV-Dateien werden über den REST-Endpunkt hochgeladen.

2.3 Bestehender Service

Der bisher verwendete Service ist in Go implementiert. Die Bezeichnungen und Preise der Assets werden anhand der ISIN von InfoZone abgerufen werden.

2.3.1 Bestehende Endpunkte

URI	Beschreibung	HTTP-Methode
/v1/:client/etfs/searchparams	HTTP-Handler-Funktionen, die Suchparameter für ETFs oder Fonds basierend auf einem Client-Parameter zurückgeben.	GET
/v1/:client/funds/searchparams		GET

/v1/:client/upload	Application Programming Interface (API)-Endpunktmethodene zum Importieren von CSV-Daten. Bei korrekter Anfrage werden die Daten asynchron importiert, und die Zeit, die für den Import benötigt wird, wird gemessen und protokolliert. Fehler werden während des gesamten Prozesses entsprechend behandelt und an den Client gemeldet.	POST
/v1/:client/etfs	API-Endpunkte, die auf Anfragen mit bestimmten Kriterien reagieren, ETFs basierend auf diesen Kriterien sucht und das Ergebnis im JSON-Format zurückgibt.	GET
/v1/:client/funds	API-Endpunkte, die auf Anfragen mit bestimmten Kriterien reagieren, Fonds basierend auf diesen Kriterien sucht und das Ergebnis im JSON-Format zurückgibt.	GET
/v1/development	Mechanismus zur Aktualisierung der Preisentwicklung von Wertpapieren basierend auf ihren ISINs. Die Aktualisierung erfolgt asynchron und nutzt Konkurrenz, um die Effizienz zu erhöhen.	PATCH
/v1/:client/details/:isin	Endpunkte, die Details zu Wertpapieren anhand ihrer ISINs abrufen können. Der erste Endpunkt (GetDetailsV1) holt Details zu einem einzelnen Wertpapier, während der zweite (GetDetailsV2) in der Lage ist, Details zu mehreren Wertpapieren in einem einzigen Aufruf abzurufen.	GET

/v2/:client/details		GET
---------------------	--	-----

Tabelle 2.3: Endpunkte des aktuellen Service

2.3.2 Struktur, Code-Qualität und Dokumentation

Die strukturelle Analyse des ETF-Search-Service offenbart eine durchdachte und modular aufgebaute Architektur. Diese modulare Struktur, erkennbar durch die Aufteilung des Codes in spezifische Pakete wie `pgstore` und `api`, ermöglicht eine klare Trennung der Verantwortungsbereiche innerhalb des Services. Jedes Paket fokussiert sich auf einen bestimmten Aspekt der Service-Funktionalität, was zu einer erhöhten Übersichtlichkeit und einer besseren Wartbarkeit des Codes beiträgt. Insbesondere die Trennung von Datenzugriffslogik (`pgstore`) und API-Schnittstellen (`api`) ist ein Hinweis auf die Einhaltung des Prinzips der Trennung von Interessen (Separation of Concerns), welches eine grundlegende Komponente softwaretechnischer Best Practices darstellt. Die Konsistenz des Codes wird durch die Verwendung wiederkehrender Muster und Konventionen innerhalb des gesamten Projekts unterstützt. Beispielsweise folgen die Dateien `get_funds_search_params.go` und `get_etf_search_params.go` einem ähnlichen Muster in ihrer Struktur und Funktionsweise, was nicht nur die Wiederverwendbarkeit des Codes erhöht, sondern auch die Einarbeitungszeit für neue Entwickler verringert. Diese Konsistenz ist ein Indikator für eine durchdachte Entwicklungsstrategie und spiegelt sich in einer vereinfachten Wartung und Skalierbarkeit des Services wider. Die Qualität des Codes lässt sich anhand verschiedener Kriterien beurteilen. Zum einen zeigt der Code eine hohe Lesbarkeit, die durch klare Strukturierung, sinnvolle Benennung von Variablen und Funktionen sowie die Verwendung von Paketimporten erreicht wird. Zum anderen weist der Code eine ordnungsgemäße Fehlerbehandlung auf, wie in der Datei `update_development.go` zu beobachten, was auf eine robuste und resiliente Anwendung hindeutet. Die Nutzung externer Pakete und Bibliotheken wie `opentracing` und `httprouter` zeigt außerdem eine Orientierung an aktuellen technologischen Standards und fördert somit die langfristige Tragfähigkeit und Modernität des Services. Die Lesbarkeit des Codes ist ein wesentlicher Faktor, der die Wartbarkeit und Erweiterbarkeit eines Softwareprojekts beeinflusst. Der ETF-Search-Service zeichnet sich durch eine klare und verständliche Codierung aus, was sich in einer strukturierten Anordnung von Funktionen und logischen Blöcken manifestiert. Die Lesbarkeit wird weiterhin durch die konsistente Verwendung von Namenskonventionen und eine logische Strukturierung der Code-Module unterstützt. Dies erleichtert nicht nur das Verständnis des bestehenden Codes, sondern auch die Implementierung neuer Funktionen und die Fehlersuche. Abschließend kann festgehalten werden, dass der ETF-Search-Service in seiner aktuellen Form eine solide Basis in Bezug auf Struktur, Konsistenz, Qualität und Lesbarkeit bietet. Diese Eigenschaften tragen maßgeblich zur Wartbarkeit und Skalierbarkeit des Services bei und bilden somit eine wichtige Grundlage für die zukünftige Entwicklung und Erweiterung des Projekts.

2.3.3 Wartbarkeit

Die Wartbarkeit von Software ist ein kritischer Aspekt, der die Effizienz und Effektivität der langfristigen Pflege und Weiterentwicklung eines Softwareprodukts bestimmt. Im Kontext des vorliegenden ETF-Search-Services ist die Wartbarkeit durch verschiedene Schlüsselfaktoren charakterisiert, die im Folgenden detailliert betrachtet werden.

1. Modularität und Code-Strukturierung: Der Service zeigt eine klare modulare Strukturierung, wobei spezifische Funktionalitäten in getrennten Paketen (api, pgstore) organisiert sind. Diese Modularität fördert eine Trennung der Verantwortlichkeiten, was essenziell für die Wartbarkeit ist. Jedes Modul kapselt eine bestimmte Funktionalität, wodurch Änderungen, Erweiterungen oder Fehlerbehebungen in einem Teil des Systems minimale Auswirkungen auf andere Teile haben. Dies reduziert die Komplexität bei der Fehlersuche und erleichtert das Verständnis des Systems für neue Entwickler.

2. Lesbarkeit und Code-Qualität: Die untersuchten Code-Samples des Services zeigen eine hohe Lesbarkeit und eine strukturierte Nutzung von Go-Programmierkonventionen. Die Verwendung klarer Funktionsnamen und strukturierter Paketimportierung trägt wesentlich zur Verständlichkeit des Codes bei. Die Lesbarkeit des Codes ist ein entscheidender Faktor für die Wartbarkeit, da sie die Einarbeitungszeit für neue Teammitglieder reduziert und die Effizienz der Fehlerbehebung erhöht.

3. Dokumentation und Code-Kommentierung: Trotz des Vorhandenseins einiger Dokumentationsdateien, insbesondere für externe Abhängigkeiten, fehlt es an einer umfassenden, projektspezifischen Dokumentation. Detaillierte Code-Kommentare und eine ausführliche Dokumentation der Systemarchitektur, Geschäftslogik und Datenflüsse sind für die nachhaltige Wartbarkeit des Services unerlässlich. Eine kontinuierliche Dokumentation ermöglicht ein schnelleres Verständnis der Systemlogik und erleichtert zukünftige Anpassungen sowie Fehlerbehebungen.

4. Testbarkeit und Automatisierung: Obwohl das Repository Anzeichen von Continuous Integration (CI) durch die Existenz einer .gitlab-ci.yml-Datei zeigt, wurde in der Stichprobe keine umfassende Testabdeckung oder Testautomatisierung festgestellt. Die Implementierung von Unit-Tests, Integrationstests und automatisierten Regressionstests ist entscheidend für die Wartbarkeit. Sie ermöglicht es, Änderungen sicher vorzunehmen, indem sichergestellt wird, dass neue Entwicklungen nicht zu unbeabsichtigten Fehlern in bestehenden Funktionen führen.

5. Wartungsstrategien und -praktiken: Für eine effektive Wartung ist es wichtig, dass klare Strategien und Praktiken etabliert sind. Dazu gehören regelmäßige Code-Reviews, das Festlegen von Codierungsstandards und die kontinuierliche Dokumentationspflege. Die Integration dieser Praktiken in den Entwicklungsprozess trägt wesentlich zur nachhaltigen Wartbarkeit und Qualitätssicherung bei.

6. Sicherheitsaspekte: Die Sicherheit von Software, insbesondere in einem sensiblen Bereich wie Finanzdienstleistungen, ist ein essenzieller Faktor für die Wartbarkeit. Eine umfassende Sicherheitsanalyse, einschließlich der Überprüfung der Authentifizierung, Autorisierung, Datenvalidierung und Fehlerbehandlung, ist erforderlich, um potenzielle Sicherheitslücken proaktiv zu identifizieren und zu schließen.

Der ETF-Search-Service weist in Bezug auf seine Wartbarkeit sowohl Stärken als auch Verbesserungspotenziale auf. Während die modulare Struktur und die Code-Lesbarkeit positiv hervorzuheben sind, bedarf es einer verstärkten Fokussierung auf umfassende Dokumentation, systematische Teststrategien und Sicherheitsaspekte, um eine nachhaltige und effiziente Wartbarkeit des Services zu gewährleisten.

2.4 Probleme des bestehenden Services

Der bestehende Service hat mehrere Probleme. 1. Der Service - Jede Anpassung der CSV-Datei erfordert auch eine Anpassung des Services. - keine Pagination - keine OpenAPI-Spezifikation - keine automatisierten Tests - keine automatisierte Datenaktualisierung -

keine Validierung der Daten - keine logische Verknüpfung der Daten - Durch die fehlende Historie, Datensicherung und Recovery können Inhalte nicht nachgeprüft werden. Dies ist insbesondere für einen Finanzdienstleister relevant. Falls etwa ein Kunde aufgrund der bereitgestellten Informationen eine Investition getätigt hat, die ihm einen finanziellen Verlust verursacht, könnte der Kunde argumentieren, dass die Daten fehlerhaft sind und deshalb der Finanzdienstleister für den entstandenen Schaden verantwortlich ist. In dieser Situation wäre es für den Finanzdienstleister kompliziert nachzuweisen, welche Daten dem Kunden bereitgestellt wurden, ob die Daten tatsächlich fehlerhaft waren und falls die Daten fehlerhaft waren, ob der Finanzdienstleister oder die zugrunde liegende Rohdatenquelle der Ursprung dieser Fehler war. Der bestehende ETF-Suchdienst, der auf einer manuell erstellten CSV-Datei basiert, ist mit zahlreichen Herausforderungen und Einschränkungen konfrontiert, die seine Effektivität und Zuverlässigkeit als Instrument für Finanzdienstleistungen signifikant beeinträchtigen. Diese Probleme erstrecken sich von technischen Aspekten bis zu solchen, die Datenintegrität und -sicherheit betreffen. Zunächst bedingt jede Anpassung der CSV-Datei eine manuelle Modifikation des Services. Dieser Prozess ist nicht nur zeitaufwendig, sondern steigert auch das Risiko für Fehler und Inkonsistenzen in den Daten. Die fehlende Automatisierung der Datenaktualisierung resultiert in einer verzögerten Reaktion auf Änderungen und neue Informationen, was für einen dynamischen Markt wie den ETF-Markt unzureichend ist. Des Weiteren bietet der Service keine Pagination, was die Handhabung großer Datensätze erschwert und die Benutzerfreundlichkeit reduziert. Das Fehlen einer OpenAPI-Spezifikation begrenzt die Interoperabilität sowie die Integration mit anderen Systemen, was wiederum die Skalierbarkeit und die Erweiterungsmöglichkeiten des Services einschränkt.

2.5 Oberfläche

2.5.1 Desktop Ansicht

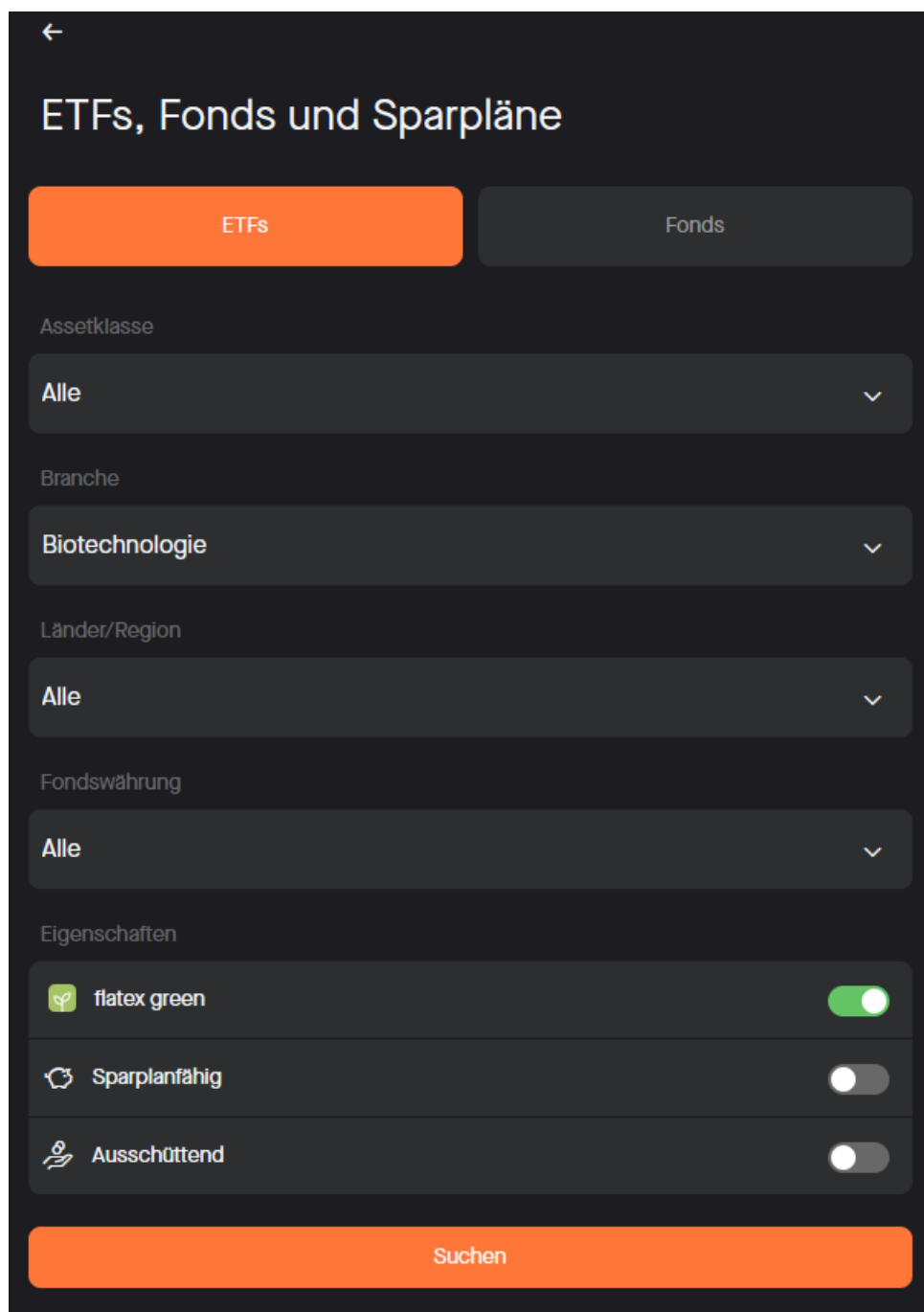


Abbildung 2.1: Suchmaske für ETFs und Fonds

< Zurück ETFs

Beliebteste Größte

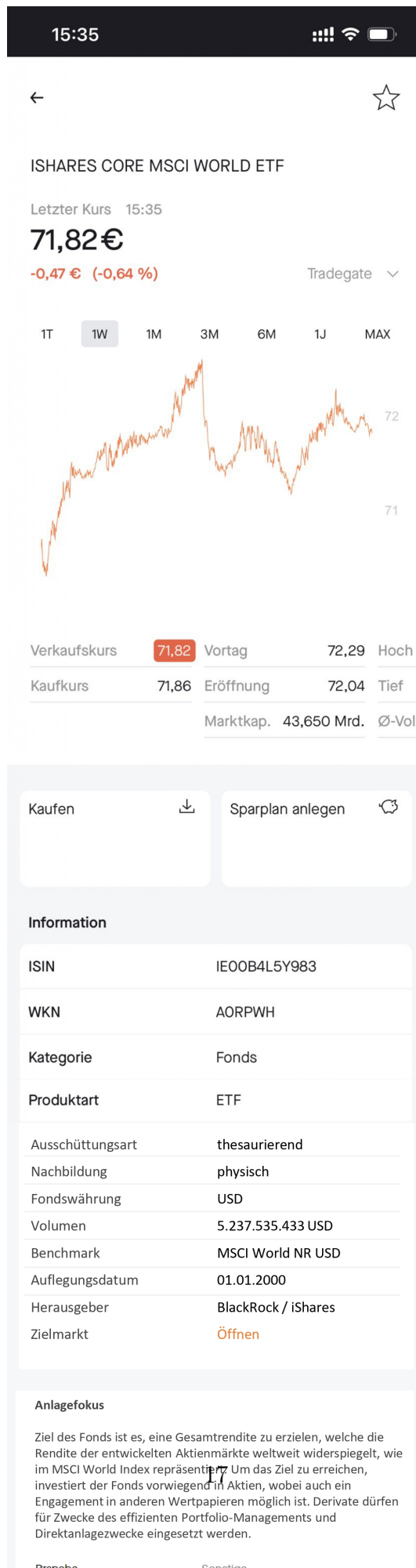
Top-Treffer

<p>ETF </p> <p>ISHSIII-CORE MSCI WLD DLA IE00B4L5Y983</p> <p>Land/Region Global</p>	<p>Entwicklung 1 Jahr 10,68 % / 7,34 €</p> <p>Kurs 76,30 €</p>	<p>Entwicklung 5 Jahre 53,79 % / 26,61 €</p> <p>Volumen 57.409.680,436 US Dollar</p>	<p>Assetklasse Aktien</p> <p>Branche Sonstige</p>
<p>ETF </p> <p>X(IE)-MSCI WORLD 1C IE00BJOKDQ92</p> <p>Land/Region Global</p>	<p>Entwicklung 1 Jahr 10,56 % / 7,94 €</p> <p>Kurs 83,44 €</p>	<p>Entwicklung 5 Jahre - / -</p> <p>Volumen 11.375.824,446 US Dollar</p>	<p>Assetklasse Aktien</p> <p>Branche Sonstige</p>
<p>ETF </p> <p>ISHSII-GL.CL.ENERGY DLDIS IE00B1XNHC34</p> <p>Land/Region Global</p>	<p>Entwicklung 1 Jahr -32,94 % / -3,67 €</p> <p>Kurs 7,53 €</p>	<p>Entwicklung 5 Jahre 73,89 % / 3,18 €</p> <p>Volumen 4.172.018,296 US Dollar</p>	<p>Assetklasse Aktien</p> <p>Branche Alternative E</p>
<p>ETF </p> <p>ISHSVII-CORE S+P500 DLACC IE00B5BMR087</p> <p>Land/Region USA</p>	<p>Entwicklung 1 Jahr - / -</p> <p>Kurs 423,18 €</p>	<p>Entwicklung 5 Jahre 70,60 % / 174,28 €</p> <p>Volumen 62.038.105,744 US Dollar</p>	<p>Assetklasse Aktien</p> <p>Branche Sonstige</p>
<p>ETF </p> <p>X(IE)-MSCI EM.MKTS 1CDL IE00BTJRMPP35</p> <p>Land/Region Emerging Markets</p>	<p>Entwicklung 1 Jahr 2,50 % / 1,11 €</p> <p>Kurs 45,24 €</p>	<p>Entwicklung 5 Jahre 9,32 % / 3,86 €</p> <p>Volumen 4.820.196,332 US Dollar</p>	<p>Assetklasse Aktien</p> <p>Branche Sonstige</p>

Abbildung 2.2: Ergebnisliste

2.5.2 App Ansicht

Abbildung 2.3: App Ansicht



Kapitel 3

SOLL-Analyse

3.1 Initiale Anforderungen des Unternehmens

Die initialen Anforderungen an die Software umfassen eine Reihe spezifischer Vorgaben, die vom Unternehmen definiert wurden. Das Unternehmen möchte zukünftige Entwicklungen auf Basis des Quarkus-Frameworks realisieren. Daher soll das Quarkus-Framework mit Java 17 verwendet werden. Dazu wurden Bibliotheken des Frameworks teilweise an die Vorgaben des Unternehmens angepasst und Werkzeuge zur Erleichterung des Entwicklungsstarts geschaffen. Für die Projekterstellung stehen dafür drei Maven Arche Types zur Verfügung, die automatisch die geforderten Projektstrukturen und Grundlagen anlegen. Insgesamt stehen drei verschiedene Maven Arche Types zur Verfügung:

- für Microservices
- für API-Spezifikationen und Client-Stubs
- Definition mittels Helm

Die Software soll als Docker-Image(s) bereitgestellt werden. Weiterhin ist die Implementierung von DAP-Tracing zur Verfolgung von Laufzeiten vorgesehen. Ein weiterer Schwerpunkt liegt auf der Ablösung der CSV-Datei durch direkte Kopplungen an die relevanten Datensysteme, um den manuellen Aufwand bei der Datenerfassung zu minimieren. Dies erfordert die Entwicklung automatisierter Tests, einschließlich Mocks für die Datenquellen. Zudem sollen notwendige Anwendungsfälle (Use Cases) zur flexiblen Suche nach ETF-Produkten ermittelt und die entsprechenden Datenquellen, wie Stammdaten-systeme, Kursdaten und Statistiken, definiert werden. Ferner sollen die Architektur und die benötigten Schnittstellen für den API-Client und die Backends konkretisiert werden. Zugleich sind auch neue Anforderungen an die API zu berücksichtigen, welche zusätzliche oder veränderte Funktionen im Vergleich zum aktuellen Stand des Services umfassen.

3.2 Optimierung bestehender Datenquellen

Das zentrale Ziel ist die Ablösung des manuellen Prozesses, der bisher das Exportieren, Importieren, Anpassen und erneute Exportieren und Importieren von CSV-Dateien umfasste, durch eine direkte Integration mit den Datenquellen. Diese Integration ermöglicht eine nahtlose und weniger fehleranfällige Datenübertragung. Zur Sicherstellung der Datenqualität soll eine Validierung der zu importierenden Daten implementiert werden.

Dabei sollen die Syntax und Semantik der Daten validiert werden. Die Automatisierung der Datenaktualisierung ist ein weiterer wichtiger Punkt. Durch regelmäßige und automatische Updates wird sichergestellt, dass die Daten stets aktuell sind. Dies ist besonders wichtig, da ETF- und Fondsdaten sich schnell ändern können. Eng verbunden damit ist die Automatisierung der Datenverarbeitung, welche die Effizienz des Systems verbessert und menschliche Fehler reduziert. Durch die Automatisierung können große Datenmengen schneller und präziser verarbeitet werden. Ein weiteres Ziel ist die logische Verknüpfung der Daten. Durch das Herstellen von Beziehungen zwischen verschiedenen Datensätzen können komplexere Analysen und Abfragen durchgeführt werden, die tiefgehende Einblicke in die Daten bieten. Die Reduktion von Dopplungen, Redundanzen und Inkonsistenzen ist ebenfalls von hoher Bedeutung. Durch das Eliminieren von überflüssigen oder widersprüchlichen Daten wird die Datenbasis schlanker und aussagekräftiger. Weiterhin wird eine verbesserte Aufgliederung der Regionen und Länder angestrebt, die die Rohdaten effektiv verwendet. Damit sollen detailliertere und gezieltere Analysen und Abfragen von ETF- und Fondsdaten ermöglicht werden. Diese Aufgliederung hilft dabei, spezifischere Einsichten in verschiedene geografische Märkte zu gewinnen und neue Funktionen zu implementieren.

3.2.1 Morningstar Direct

Um eine klare Trennung zu gewährleisten und Vertragsverletzungen zu verhindern, sollten die Daten getrennt exportiert werden. Daher wird angestrebt eine Datei mit bis zu 8.000 verschiedenen ISINs ohne ESG-Bewertungen und eine Datei mit bis zu 400 verschiedenen ISINs mit ESG-Bewertungen zu exportieren. Es wurde bisher nicht festgelegt, ob der Export der ESG-Daten als Ergänzung der Hauptdatei fungieren soll oder die bis zu 400 festgelegten ISINs vollständig aus der Hauptdatei losgelöst und mit allen selektierten Daten exportiert werden soll. Der Export soll im CSV-Format erfolgen.

3.3 Integration neuer Datenquellen

3.3.1 Morningstar Top 10 Holdings

Mit der Umsetzung der neuen Architektur soll die Integration der „Morningstar Top 10 Holdings“ Datenquelle erfolgen. Diese Quelle liefert die zehn größten Assets eines Fonds zurück.

3.4 Entwürfe

Die Suchfunktion soll eine Volltextsuche ermöglichen. Zusätzlich werden ein Fuzzy-Matching, um Tippfehler zu korrigieren und eine Autovervollständigung, um die Suche zu vereinfachen, verwendet. Die Funddaten sollen über eine Representational State Transfer (REST)-API abrufbar sein. Die REST-API soll eine dynamische Filterung der Funddaten ermöglichen.

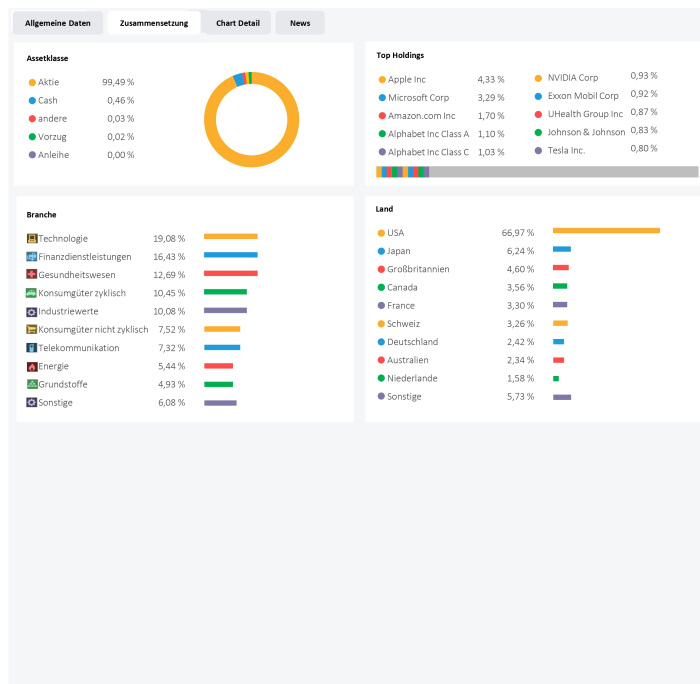


Abbildung 3.1: Suchmaske für ETFs und Fonds

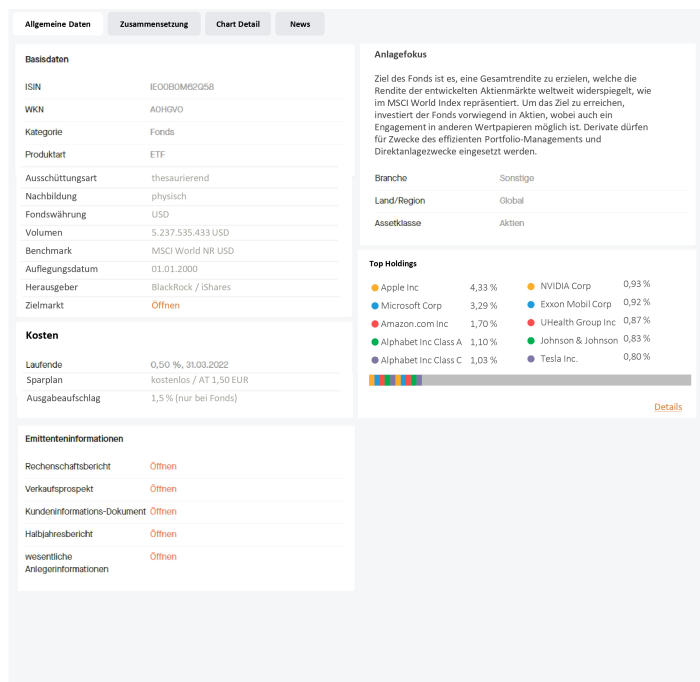


Abbildung 3.2: Suchmaske für ETFs und Fonds

The screenshot displays a financial search interface with several sections:

- Allgemeine Daten** (General Data): Includes fields for ISIN (IE00B0M2Q258), WKN (A0H9V0), Kategorie (Fonds), Produktart (ETF), Ausschüttungsart (thesaurie), Nachbildung (physisch), Fondswährung (USD), Volumen (5.237.53), Benchmark (MSCI Wld), Auflegungsdatum (01.01.20), Herausgeber (BlackRock), and Zielmarkt (Offnen).
- Kosten** (Costs): Shows Laufende (0,50 %, 31.03.2022), Sparplan (kostenlos / AT 1,50 EUR), and Ausgabeaufschlag (1,5 % (nur bei Fonds)).
- Emittenteninformationen** (Issuer Information): Lists documents like Rechenschaftsbericht, Verkaufsprospekt, Kundeninformations-Dokument, Halbjahresbericht, and wesentliche Anlegerinformationen, each with an 'Offnen' link.
- Anlagefokus** (Investment Focus): Contains a descriptive text about the fund's goal to track global developed stock markets.
- Zielmarkt** (Target Market) popup window:
 - Risikoklasse: D
 - Verlusttragfähigkeit: Totalverlust des Einsatzes möglich
 - Kenntnisse & Erfahrungen: Kunde benötigt Basiskenntnisse/-Erfahrungen
 - Kundenkategorien: für alle Kunden geeignet
 - Risikoindikator: mittel bis hoch
 - Risiko/Rendite: ertragsorientiert
 - Anlagehorizont: 3 - 5 Jahre länger als 6 Jahre
- Holdings**: A list of top holdings including Alphabet Inc Class A (1,10%), Alphabet Inc Class C (1,03%), Johnson & Johnson (0,83%), and Tesla Inc. (0,80%).

Abbildung 3.3: Suchmaske für ETFs und Fonds

Kapitel 4

Anforderungen

Die Analyse der bestehenden ETF-Suchfunktion offenbarte eine Reihe von Herausforderungen. Die ursprüngliche Implementierung in Go und die manuelle Verwaltung von CSV-Dateien führten zu einer verzögerten Datenaktualisierung und erhöhten die Gefahr von menschlichen Fehlern. Eine Analyse der Anforderungen ergab die Notwendigkeit, den Service in Java umzusetzen, wobei das Quarkus-Framework verwendet wird, um den manuellen Aufwand zu reduzieren und die Datenintegrität zu verbessern. Die Herausforderung besteht darin, die manuelle Verwaltung dieser CSV-Datei durch eine direkte Integration mit den Datenquellen zu ersetzen, um Echtzeit-Datensynchronisation und eine Verringerung der Fehleranfälligkeit zu erreichen. Die Analyse hat gezeigt, dass die aktuellen Prozesse und Systeme nicht den Sicherheitsstandards des Unternehmens entsprechen. Um den Unternehmensprozessen und -richtlinien zu entsprechen, muss der aktuelle manuelle Export- und Anpassungsprozess durch eine automatisierte Lösung ersetzt werden, die die Morningstar Direct Web Services zur Datenabfrage einbezieht.

4.1 Funktionale Anforderungen

ID	Anforderung
F-1	Die Service-Struktur soll den bisherigen ETF-Search-Service ersetzen und dessen Funktionalitäten übernehmen.
F-2	Die Service-Struktur soll bisher manuelle Prozesse durch Verwendung direkter Schnittstellen automatisieren.
F-3	Die Service-Struktur muss die ETF- und Fondsdaten in einer Datenbank persistieren.
F-4	Die Service-Struktur muss die Bezeichnungen und Kursdaten von ETFs und Fonds aus dem KurseInformations-System (InfoZone) abrufen.
F-5	Die Service-Struktur soll die Erstellung und Auslieferung der aktuellen CSV-Datei programmatisch automatisieren.
F-6	Die Service-Struktur soll Quellen regelmäßig und automatisch importieren.
F-7	Die Service-Struktur soll Filteroptionen dynamisch bereitstellen.
F-8	Die Service-Struktur soll die Selektion ergebnisloser Filteroptionen verhindern.
F-9	Die Service-Struktur soll eine fehlerverzeihende Suchfunktion mit Fuzzy-Matching implementieren.
F-10	Die Service-Struktur soll zu importierende Daten zuvor validieren.

Tabelle 4.1: Funktionale Anforderungen

4.2 Nicht-funktionale Anforderungen

ID	Anforderung
NF-1	Die Service-Struktur muss in Java 17 entwickelt sein.
NF-2	Die Service-Struktur muss das Quarkus-Framework verwenden.
NF-3	Die Service-Struktur muss die Maven Archetypes des Unternehmens verwenden.
NF-4	Die Service-Struktur muss als Docker-Image(s) ausgeliefert werden.
NF-5	Die Service-Struktur muss Überwachung von Laufzeiten unterstützen.
NF-6	Die Service-Struktur muss Prozesse für die CI/CD implementieren.
NF-7	Die Service-Struktur muss das Buildsystem Maven verwenden.
NF-8	Die Service-Struktur muss eine OpenAPI-Spezifikation als Ausgangspunkt für die Entwicklung definieren.
NF-9	Die Service-Struktur muss automatisierte Tests unter Verwendung von Mocks für Datenquellen implementieren.
NF-10	Die Service-Struktur muss das Datenbankmanagementsystem PostgreSQL verwenden.
NF-11	Die Service-Struktur soll zustandslos implementiert werden.
NF-12	Die Service-Struktur soll automatische Unit- und Integrationstests enthalten.
NF-13	Die Service-Struktur soll von externen Abhängigkeiten isolierte Unit-Tests durch Mocking realisieren.
NF-14	Die Service-Struktur soll eine hohe Testabdeckung erreichen.

Tabelle 4.2: Nicht-funktionale Anforderungen

Kapitel 5

Konzept

Das Konzept soll die funktionalen und nicht funktionalen Anforderungen umsetzen. Das Konzept sieht vor, die bestehende Go-Implementierung durch eine Java-basierte Lösung unter Verwendung des Quarkus-Frameworks zu ersetzen. Quarkus wurde aufgrund seiner Eignung für die Entwicklung von Microservices und seiner Unterstützung für eine effiziente Softwareentwicklung ausgewählt. Die Bezeichnung „ETF-Search-Service“ ist unpassend, da die Funktionalität des Service sich nicht nur auf die Suche beschränkt, die Funktionalität auch das Filtern und Bereitstellen von Daten umfasst, die Daten nicht nur ETFs enthalten, sondern auch Fonds umfassen. Getrieben von grundlegenden architektonischen Prinzipien wie dem „Prinzip der Einzelverantwortung“, der „Trennung von Zuständigkeiten“, dem „Minimieren von Kopplungen“ und dem „Maximieren von Zusammenhalt“, wird der ETF-Search-Service entsprechend in zwei spezialisierte Microservices aufgespalten:

1. Fund Retrieval Service (FRS)
2. Fund Discovery Service (FDS)

5.1 Architektur

Die Architektur der Fund Management Suite (FMS) ist in Abbildung dargestellt. Die FMS besteht aus den Modulen Fund Commons und Fund Discovery Service API (FDS-API) und aus den Microservices FRS und FDS. Im Designprozess wurden die Verantwortlichkeiten der beiden Microservices definiert. Der FRS ist für den Import und die Verwaltung der Daten zuständig, während der FDS die Schnittstelle für die Suche und Filterung der Daten bereitstellt. Die Verwendung von Liquibase ermöglicht eine versionierte Verwaltung von Datenbankschemaänderungen und trägt zur Konsistenz und Nachvollziehbarkeit der Datenbankänderungen bei.

5.2 Vorteile der Trennung

Eine Trennung wie diese ermöglicht zusätzlich die Möglichkeit einer stufenweisen Einführung der neuen Service-Struktur. So könnte der FRS zunächst separat eingeführt und damit der manuelle Prozess der CSV-Datei-Erstellung abgelöst werden. Der FDS kann anschließend eingeführt werden.

Single Responsibility Principle Da zwischen importzentrierten Aufgaben (FRS) und Datenabrufoperationen (FDS) unterschieden wird, ist jeder Service mit einem klaren

und eindeutigen Fokus konzipiert. Dies stellt sicher, dass Änderungen in einem Bereich nicht unbeabsichtigt die Funktionalitäten des anderen beeinträchtigen, was zu wartungsfreundlicherem und weniger fehleranfälligerem Code führt.

Separation of Concerns Der Service ist akribisch darauf ausgerichtet, spezifische Aufgaben zu bewältigen. Er FRS konzentriert sich auf die Datenbeschaffung, -validierung und -verwaltung, während der FDS ausschließlich für die Datenverbreitung verantwortlich ist. Diese Abgrenzung sorgt für Klarheit und reduziert Komplexitäten bei Updates oder Fehlerbehebungen.

Minimierung von Kopplung Durch getrennte Services ist eine unabhängige Skalierung, Bereitstellung und Wartung möglich. Wenn eine Funktionserweiterung oder -modifikation im FRS erforderlich ist, wird dadurch nicht notwendigerweise eine Änderung im FDS erforderlich, was Agilität im Entwicklungsprozess und der Bereitstellung ermöglicht.

Maximierung von Kohäsion Durch die Ausrichtung der Services auf ihre Funktionsbereiche wird die interne Kohärenz gestärkt. Funktionen im Zusammenhang mit Datenimport, Validierung und Überwachung sind innerhalb des FRS kohärent gebündelt, was die Entwicklung und das Testen erleichtert.

Chain-of-Trust Der FRS etabliert eine Vertrauenskette mit dem FDS, wobei sichergestellt wird, dass Validierungen beim Import und Abruf konzentriert sind. Dies eliminiert redundante Überprüfungen im FDS, erhöht die Effizienz und reduziert potenzielle Fehlerquellen.

Single Source of Truth Die zentralisierte Datenverwaltung im FRS gewährleistet, dass die von FDS verbreiteten Daten stets konsistent, genau und aktuell sind, was die Datenintegrität über die gesamte Plattform fördert.

Sicherheit Durch die Beschränkung von Datenbankschreiboperationen auf FRS wird die Sicherheit verstärkt. Im Falle von Schwachstellen kann der FDS, selbst wenn kompromittiert, die Datenbank nicht unbeabsichtigt verändern. Dieser geschichtete Ansatz gewährleistet eine erhöhte Datensicherheit.

Optimierung Caching im FDS und gezielte Datenabrufe optimieren die Leistung, um schnelle Antworten auf häufige und entscheidende Abfragen zu gewährleisten.

Skalierbarkeit Eine Microservice-Architektur, die auf Quarkus basiert, gewährleistet nahtlose Skalierbarkeit. Während unser Daten- oder Benutzerstamm wächst, kann jeder Service unabhängig skaliert werden, um die erhöhte Last zu bewältigen.

Flexibilität Dieses Design ist zukunftssicher. Ob es um den Übergang zu neuen Datenquellen oder die Integration mit internen Microservices geht, unsere Architektur ist auf Anpassung ausgerichtet.

Risikomanagement Die Segregation reduziert den Schadensradius. Ein Problem in FRS wird FDS nicht unterbunden und umgekehrt. Diese Segmentierung hilft bei der schnelleren Identifikation und Behebung von Problemen und minimiert Ausfallzeiten.

Das Ziel des Entwurfs war es, eine Service-Struktur zu entwickeln, die eine effiziente Suche, Filterung und Abfrage von ETF- und Fondsdaten ermöglicht. Die Entscheidung, den Service in zwei Microservices zu teilen — den FRS und den FDS — basierte auf dem Prinzip der Trennung von Verantwortlichkeiten. Der FRS konzentriert sich auf die Datenimportfunktionen, während der FDS die Such- und Filterfunktionen bereitstellt. Diese Aufteilung verbessert die Wartbarkeit, Sicherheit und Skalierbarkeit des Gesamtsystems. Da das Frontend zukünftig nur mit dem FDS interagieren und der FDS keine Änderungen an den Datensätzen in der zentralen Datenbank durchführen wird, kann der FDS nur lesend auf die Datenbank zugreifen. Durch diese Einschränkung wird die Trennung von

Verantwortlichkeiten noch einmal verstärkt und zusätzlich die Sicherheit des Systems verbessert. Sollte der FDS kompromittiert werden, können durch diese Einschränkung keine Manipulationen an den Datensätzen durchgeführt werden.

5.3 Fund Management Suite

Die FMS ist die Service-Struktur und umfasst alle Bibliotheken und Services der Domäne.

5.3.1 Fund Commons

Die Fund Commons (FC)-Bibliothek umfasst Logik und Strukturen, die von beiden Services benötigt werden. Durch diese Auslagerung wird Code-Dopplung deutlich verringert und es entsteht eine Single-Source-of-Truth, wodurch Änderungen nur an einer zentralen Stelle notwendig sind. Die Bibliothek enthält alle Hibernate Entitäten, die Migrationskripte für das Datenbankschema (Liquibase Changelogs) und die Initialisierungslogik für die Datenbank. Die Bibliothek wird von den anderen Services als Abhängigkeit eingebunden. Die Bibliothek enthält die Repository- und Service-Klassen zu den Hibernate Entitäten. Die Hibernate Entitäten definieren die Struktur der Datenbank. Domain Driven Design wird verwendet, um die Datenbankstruktur zu definieren. In den Bereitstellungsumgebungen Quality Integration Testing (QIT), User Acceptance Testing (UAT) und Production (PROD) wird das Datenbankschema durch Liquibase verwaltet. Durch die Konfigurations-Profile des Quarkus-Frameworks kann die Kontrolle über das Datenbankschema für das Hibernate ORM in den produktiven Umgebungen deaktiviert werden. Dies ermöglicht eine bessere Kontrolle und Nachvollziehbarkeit der Datenbankstruktur. Die Bibliothek ist unabhängig von den anderen Modulen versionierbar. Durch diese Versionierbarkeit können die Services abweichende Versionen der Bibliothek verwenden. Solange die verwendeten Major-Versionsnummern der Bibliothek identisch sind, bleiben die Services bei Abweichungen der Minor- und Patch-Versionsnummern kompatibel. Daher müssen bei Änderungen nicht alle Services angepasst und aktualisiert werden. Weichen jedoch die Major-Versionsnummern ab, sind die Versionen der Bibliothek nicht miteinander kompatibel. Dadurch können die Services verschiedene Versionen der Bibliothek verwenden, solange keine Breaking Changes eingeführt werden.

5.3.2 Fund Retrieval Service

Der FRS ist für den Import, die Verarbeitung, die Validierung und die Speicherung der Funddaten zuständig. Der Service gewährleistet ein zusammenhängendes und zentralisiertes Datenverwaltungssystem. Nur der FRS kann schreibend auf die geteilte Datenbank zugreifen. Der FRS ist zusätzlich nicht extern erreichbar.

5.3.2.1 Verantwortlichkeiten

Importieren der rohen ETF- und Fond-Daten aus der CSV-Datei, die von Morningstar Direct exportiert wurde. Abrufen der Sparplanfähigkeit aus dem Bankensystem ENT-AX Bericht Rp312. Holen der am häufigsten gehandelten ETF- und Fond-Daten aus der Business Intelligence (BI). Erwerben der ETF- und Fond-Preis- und Namensdaten aus InfoZone. Schemaänderungen und Validierungen mit Liquibase. Verwalten von Daten in-

nerhalb der Datenbank. Überwachen von Datenänderungen nach Importen und Abfragen. Validieren der importierten und abgerufenen Daten.

5.3.2.2 Entwicklungs- und Änderungsprozess

Um den Entwicklungsprozess zu beschleunigen und zu vereinfachen, werden in der DIT-Umgebung Änderungen an den Hibernate Entitäten im FC durchgeführt. Um eine klare, nachvollziehbare, gut dokumentierte und statische Struktur zu gewährleisten, wird bei der Veröffentlichung neuer Versionen automatisch durch einen CI/CD-Prozess ein Liquibase Changelog aus den Änderungen an den Hibernate Entitäten erzeugt und mit der FC-Bibliothek ausgeliefert. In den Umgebungen QIT, UAT und PROD wird die Kontrolle über die Datenbankstruktur für Hibernate ORM deaktiviert. Stattdessen wird die Datenbankstruktur durch Liquibase auf Basis der Changelogs verwaltet.

5.3.2.3 Bedeutung und Vorteile

- **Strukturiertes Datenmanagement:** Beginnend mit dem Datenbankschema wird sichergestellt, dass die Datenebene organisiert, skalierbar und effizient bleibt.
- **Zuverlässige Evolutionen:** Die Verwendung von Liquibase für Schemaänderungen bedeutet, dass Modifikationen des Datenbankschemas versioniert, nachvollziehbar, statisch sind.
- **Integrität und Konsistenz:** Dieser Prozess gewährleistet, dass Daten in einem konsistenten Zustand bleiben und minimiert potenzielle Diskrepanzen und Konflikte.

5.3.3 Fund Discovery Service

Als Middleware dient der FDS als Brücke zwischen Backend FRS und Front-End-Clients und erleichtert die formatierte Datenabfrage, dynamische Suchfunktionen und stellt sichere und optimierte Interaktionen sicher.

5.3.3.1 Verantwortlichkeiten

Bereitstellung einer REST-Schnittstelle zum Abrufen von ETF- und Fond-Daten. Zwischenspeichern häufig angeforderter Daten zur Leistungssteigerung. Durch die Beschränkung auf lesenden Datenbankzugriff wird die Sicherheit erhöht, da Schwachstellen oder Kompromittierungen nicht zu unbeabsichtigten Datenbankmodifikationen führen. Trotz der Beschränkung könnte der FDS zukünftig mittels Kafka dem FRS mitgeteilt und entsprechend umgesetzt werden. Dies bewahrt die Prinzipien der Trennung von Anliegen und der Vertrauenskette und stellt sicher, dass Datenvalidierungen zentralisiert bleiben.

5.3.3.2 Entwicklungs- und Änderungsprozess

Der Entwicklungslebenszyklus für den FDS beginnt mit der Definition einer OpenAPI-Spezifikation, die als Vorlage für die Service-Endpunkte, Antwort-Strukturen und des gesamten API-Verhaltens dient. Modell- und Data Transfer Object (DTO)-Generierung: Unter Verwendung der OpenAPI-Spezifikation als Grundlage werden Modelle und DTOs automatisch generiert, die sowohl dem FDS als auch den Client-Anwendungen, die mit dem Dienst interagieren, dienen.

Integration in *fund-discovery-service*: Das Projekt *fund-discovery-service-api*, das die OpenAPI-Spezifikation enthält, wird als Bibliothek in das Hauptprojekt *fund-discovery-service* integriert. Diese Einbeziehung stellt sicher, dass der FDS die generierten Modelle und DTOs nutzen und die definierten Endpunkte nahtlos umsetzen kann.

Implementierung und Iteration: Die anschließende Entwicklung und Modifikationen sich um diese vordefinierte Struktur, wobei Änderungen oder Erweiterungen des FDS zunächst in der OpenAPI-Spezifikation reflektiert werden.

5.3.3.3 Bedeutung und Vorteile

- **Konsistenz:** Durch den Beginn mit einer strukturierten Spezifikation wird sichergestellt, dass die Entwicklung mit dem definierten API-Vertrag übereinstimmt.
- **Flexibilität:** Änderungen oder Erweiterungen können systematisch durch Aktualisierung der OpenAPI-Spezifikation eingeführt werden, sodass die resultierenden Systemmodifikationen vorhersehbar und organisiert sind.
- **Einfache Integration:** Kund:innen können aufgrund der strikten Einhaltung der OpenAPI-Spezifikation sicher sein, dass sich der Dienst wie erwartet verhält, was die Integration vereinfacht und potenzielle Inkonsistenzen reduziert.
- **Schnelle Entwicklung:** Die automatische Generierung von Modellen und DTOs eliminiert manuelle Programmierfehler und beschleunigt den Entwicklungsprozess.

Der FDS ist für die Bereitstellung der Funddaten zuständig. Der FDS bindet das FDS-API als Abhängigkeit ein. Der FDS implementiert die API-Spezifikation des FDS-API. Der FRS und der FDS sind voneinander entkoppelt. Dadurch kann der FDS unabhängig vom FRS skaliert werden. Alle Endpunkte, die Datensätze ohne zusätzliche Logik zurückgeben, verwenden nur die Hibernate Entitäten und die PostgreSQL-Datenbank. Die Endpunkte für Suche, Filterung und Autovervollständigung verwenden Hibernate Search in Kombination mit einer Elasticsearch-Instanz. Die Verwendung von Hibernate Search mit Elasticsearch ist eine optimal Wahl, da bereits das Hibernate ORM mit der Panache-Erweiterung verwendet wurde. Hibernate Search unterstützt Apache Lucene, Elasticsearch und OpenSearch. Es wurde Elasticsearch verwendet, da das Quarkus-Framework mit den Development-Services automatisch eine Instanz von Elasticsearch (oder OpenSearch) als Docker-Container zur Entwicklungszeit bereitstellen kann. Elasticsearch war bis zur Version 7.10.2 quelloffen und wurde jedoch in den nachfolgenden Versionen unter einer nicht quelloffenen Lizenz veröffentlicht. OpenSearch ist als Fork von Elasticsearch entstanden, um die Weiterentwicklung des Projektes zukünftig quelloffen unter einem anderen Namen fortzuführen. Aus diesen Gründen wäre OpenSearch eine Alternative, jedoch erlaubt das Unternehmen nur die Verwendung von Docker Images, die als „Official“ oder „Certified“ eingestuft wurden. OpenSearch hat jedoch lediglich das Level „Verified“, wodurch es gesondert von der Abteilung IT-Sicherheit genehmigt werden muss. Aufgrund der begrenzten Bearbeitungszeit des Bachelorprojekts wurde stattdessen Elasticsearch mit der Version 7.10.2 verwendet, um langfristig das quelloffene OpenSearch zu verwenden.

5.3.4 Fund Discovery Service API

Das FDS-API beinhaltet die OpenAPI-Spezifikation für den Fund Discovery Service. Dadurch ist die Spezifikation für den FDS getrennt von der Implementierung und kann von

anderen Modulen als Abhängigkeit eingebunden werden. Zusätzlich kann die FDS-API unabhängig von der Implementierung versioniert werden. Eine CI/CD-Pipeline generiert aus der OpenAPI-Spezifikation die Dokumentation und die Client-Bibliothek. Die Client-Bibliothek wird vom FDS eingebunden. Sie enthält die DTO und die Client-Logik für den FDS. Im FDS muss das vom OpenAPI-Generator erstellte Interface implementiert werden.

5.4 Datenverarbeitung und -mapping

Die Daten von Morningstar Direct werden durch Mapping-Tabellen angepasst, um eine konsistente und genaue Datenquelle für den Service zu gewährleisten. Es wurden fünf verschiedene Mapping-Tabellen identifiziert, die jeweils spezifische Attribute wie Assetklasse, Region und Währung abbilden. Diese Tabellen werden regelmäßig überprüft, um sicherzustellen, dass die Daten aktuell bleiben.

5.5 Datenbankentscheidungen

PostgreSQL ist ein etabliertes Datenbankmanagementsystem, das eine umfassende Unterstützung bietet. Im Unternehmen sind Datenbank-Administrator (DBA)s für die UAT- und PROD-Umgebungen und die Entwickler für Development Integration Testing (DIT)- und QIT-Umgebungen verantwortlich.

5.6 Volltextsuche und Dynamische Filterung

Hibernate Search soll für die dynamische Filterung der Fondsdaten mit einem Elasticsearch Index verwendet werden. Daher muss keine zusätzliche Filterung auf der Datenbankebene implementiert werden. Für die dynamische Filterung werden zwei Endpunkte benötigt. Ein Endpunkt zum Abrufen der anwendbaren Filter und die Anzahl an Ergebnissen für die Filter. Die Anzahl an Ergebnissen für die Filter wird über eine Aggregation mittels Hibernate Search und Elasticsearch ermittelt. Diesem Endpunkt kann optional eine Selektion von Filtern übergeben werden. Wenn keine Selektion von Filtern übergeben wird, werden alle Filter zurückgegeben. Ist dies doch der Fall, werden nur die noch anwendbaren Filter zurückgegeben. Dadurch soll sichergestellt werden, dass keine Filterauswahl zu einer leeren Ergebnismenge führen kann. Ein Endpunkt zum Filtern der Funddaten auf Basis der übergebenen Filter.

5.7 Datenbank

Die FMS nutzt die folgenden Standards für die Datenhaltung:

- ISO 639-1 für Sprachen
- ISO 3166-1 Alpha-2 für Länder
- ISO 4217 für Währungen

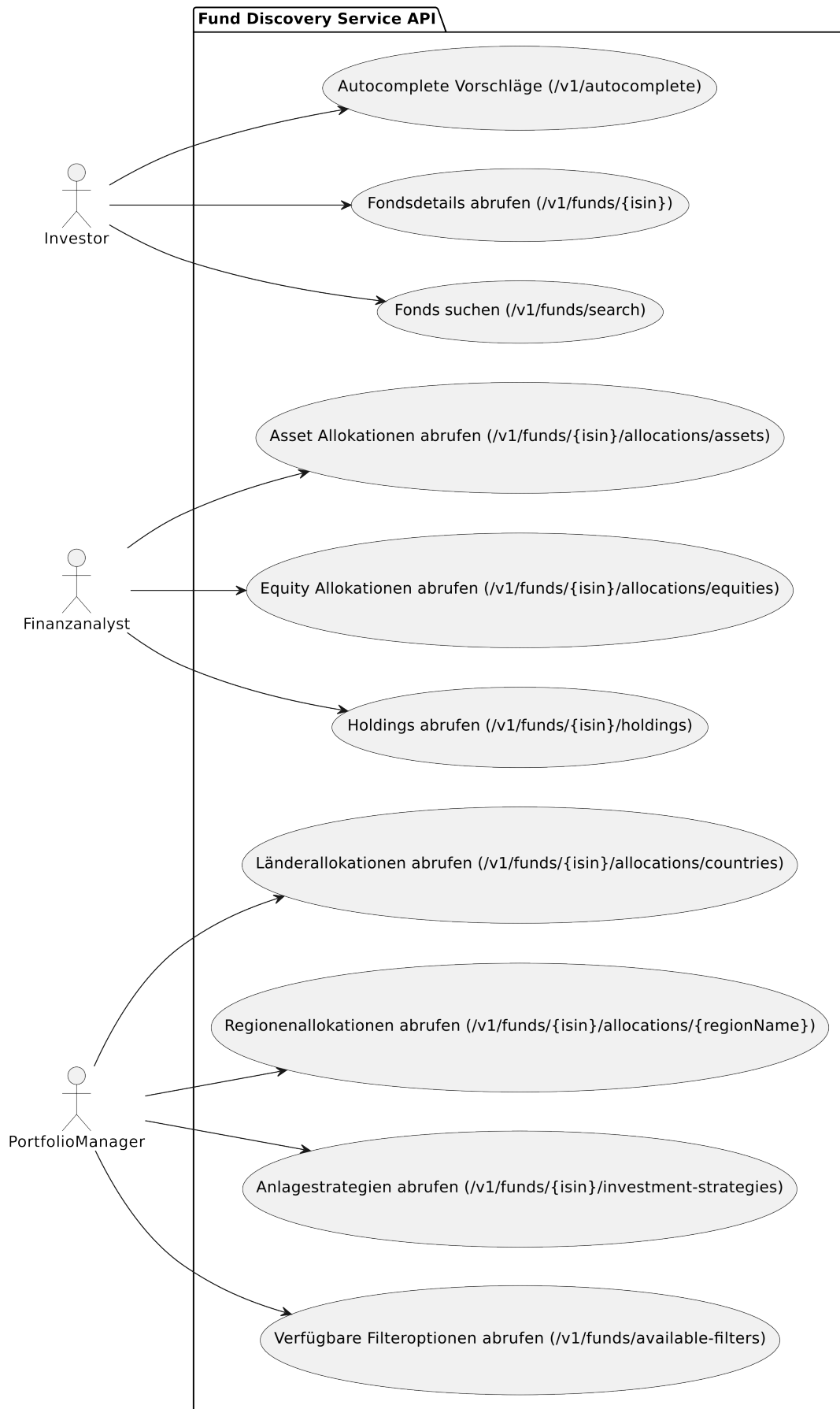


Abbildung 5.1: Anwendungsfälle der REST-Schnittstelle

5.8 Automatisierte Tests mit Mocks

Die Implementierung automatisierter Tests mit Mocks für die Datenquellen war ein wesentlicher Bestandteil des Projekts. Diese Tests ermöglichten es, die Funktionalität und Zuverlässigkeit des Services zu gewährleisten, ohne auf externe Datenquellen angewiesen zu sein.

Kapitel 6

Verwendete Technologien

6.1 Quarkus

Quarkus ist ein Java-Framework, das für die Entwicklung von Mikroservices und serverlosen Systemen konzipiert ist. Das Quarkus-Framework wurde erstmals am 20. März 2019 vom Unternehmen Red Hat Software unter der Apache-2.0-Lizenz [9] veröffentlicht [7, S. 307]. Es entstand als Alternative zu den bestehenden Java-Mikroservice-Stacks und bietet ein Anwendungsframework, das unübertroffene Leistungsvorteile bietet, während es gleichzeitig ein Entwicklungsmodell nutzt, das auf den APIs (Anwendungsprogrammierschnittstellen) beliebter Bibliotheken und Java-Standards basiert, die im Java-Ökosystem seit Jahren praktiziert werden. Quarkus legt den Fokus auf die Produktivität der Entwickler:innen, da diese, Produktivität und Benutzerfreundlichkeit einer Technologie am meisten schätzen [11, S. 6].

6.1.1 Live Coding und Hot-Reloading

Die Live-Coding-Funktion von Quarkus trägt erheblich zur Verbesserung des Entwicklungsprozesses bei, indem sie es ermöglicht, Änderungen sofort zu sehen, ohne die Anwendung neu starten zu müssen. Dies fördert einen iterativen Entwicklungsansatz und ermöglicht schnelles Feedback, was besonders nützlich ist beim Prototyping und beim Testen neuer Ideen. Allerdings gibt es auch Herausforderungen, wie die Notwendigkeit, den Zustand der Anwendung zwischen Neuladungen zu managen. Außerdem kann Live Coding in komplexen Anwendungen mit vielen Abhängigkeiten oder in bestimmten Umgebungen, die spezielle Konfigurationen erfordern, eingeschränkt sein. Die Steigerung der Entwicklerproduktivität ist ein wichtiger Aspekt des Quarkus-Frameworks, insbesondere durch die Hot-Reloading-Funktion. Diese Funktion ermöglicht die Reflexion von Codeänderungen in Echtzeit und ist unabhängig von der verwendeten Entwicklungsumgebung (IDE) verfügbar. Änderungen am Quellcode werden umgehend gespeichert, sodass ein kontinuierliches Testen des Codes möglich ist, ohne dass ein Neustart erforderlich ist, selbst bei Änderungen in der Konfigurationsdatei. Ein weiteres Attribut von Quarkus ist die Fähigkeit, neue Abhängigkeiten in eine Maven POM-Datei in einem aktiven Projekt einzubinden, was zur automatischen Übernahme neuer Bibliotheken führt, ohne dass die Anwendung neu gestartet werden muss. [6, S. 8]

6.1.2 Continuous Testing

Diese Funktion ermöglicht es Entwickler:innen, Tests automatisch im Hintergrund auszuführen, während sie an ihrer Anwendung arbeiten. Sobald eine Änderung am Code vorgenommen wird, führt Quarkus die relevanten Tests durch, ohne dass Entwickler:innen manuell eingreifen müssen. Dies bietet den Vorteil, dass Probleme und Fehler frühzeitig erkannt und behoben werden können, was die Qualität des Codes verbessert und den Entwicklungsprozess beschleunigt.

6.1.3 Fokus auf Cloud- und Microservice-Umgebungen

In der Softwarebranche bestanden Bedenken bezüglich des Einsatzes von Java in Containern. Container müssen schnell starten, wenig Ressourcen verbrauchen, klein sein und Java-Frameworks erfüllen diese Anforderungen oft nicht. Quarkus hingegen ist so konzipiert, dass es Anwendungen ermöglicht, schnell zu starten und eine hervorragende Effizienz sowie Leistung zu bieten. Dabei nutzt es das Konzept der Build-Zeit-Verarbeitung, um so viel Verarbeitung wie möglich von der Laufzeit in die Kompilierungszeit zu verlagern. Dieser Ansatz wurde durch Google Dagger, ein Framework für Abhängigkeitsinjektion für Java und Android, populär gemacht. Bei mobilen Anwendungen sind Laufzeitleistung und Ressourcennutzung essenziell, da sie direkt das Benutzererlebnis beeinflussen. Dagger musste die Verarbeitung von der Laufzeit in die Kompilierungszeit verschieben, um Reaktionszeiten und Batterieverbrauch zu reduzieren. In Unternehmensanwendungen mag der Batterieverbrauch irrelevant sein. Trotzdem ist der Speicherbedarf wichtig, da er direkt die Produktionskosten beeinflusst, was ein kritischer Faktor für die Preisgestaltung im Cloud-Computing ist [11, S. 11].

6.1.4 Unterstützung und Integrationen

Quarkus unterstützt etablierte und verbreitete Java-Bibliotheken und -Standards. Diese umfassen eine Vielzahl beliebter Bibliotheken und Enterprise-Java-Standards, die sowohl im JVM- als auch im nativen Modus genutzt werden können. Quarkus bietet native Unterstützung für eine Vielzahl gängiger Bibliotheken und Standardimplementierungen innerhalb des Java-Ökosystems. Quarkus nutzt etablierte Standards wie MicroProfile und Jakarta EE sowie beliebte Open-Source-Frameworks wie Hibernate, Vert.x, Apache Camel und RESTEasy. Dies ermöglicht Entwickler:innen, ihre Erfahrungen und Kenntnisse aus der Arbeit mit diesen Bibliotheken bei der Verwendung von Quarkus erneut einzusetzen. Die Verwendung von Open-Source-Standards reduziert die Notwendigkeit einer engen Bindung an einen einzelnen Anbieter, da mehrere Anbieter den gleichen Standard implementieren können. Dies ermöglicht es, Anwendungen zu migrieren, die bereits MicroProfile oder Jakarta EE-Standards unterstützen. Die Migrationspfade für bestehende Module sind nicht nur auf die gleichen APIs zurückzuführen, sondern auch auf einfachere Migrationspfade. Es wäre jedoch möglich, eine Anwendung, die bereits MicroProfile-Standards verwendet, ohne Codeänderungen zu migrieren. Quarkus bietet zusätzlich eingeschränkte Unterstützung für einige Spring-APIs, um die Migration für Entwickler:innen zu vereinfachen, die sich an die MicroProfile- und Jakarta EE-APIs anpassen müssen. [11, S. 20]

Bibliotheken werden in das Ökosystem des Frameworks integriert und dabei die Startzeit und der Speicherverbrauch dieser optimiert. Die integrierten Bibliotheken werden als Erweiterungen in Quarkus-eigenen Paketen veröffentlicht. Dies ermöglicht verschiedene Quality-of-Life-Verbesserungen. Es sind bereits viele Bibliotheken als Erweiterungen des

Quarkus-Frameworks integriert, wie die Hibernate-Bibliotheken (ORM, Search, Validator, Envers) [9]. Dabei werden Bibliotheken nicht nur integriert, sondern auch erweitert, wie das Hibernate ORM Panache, das den Umgang mit dem Hibernate ORM vereinfacht und verbessert. [10]

6.1.5 Konfigurationsverwaltung

Eine dieser QoL Verbesserungen ist eine zentrale Konfigurationsverwaltung mit nur einer Konfigurationsdatei. Davor war es notwendig, jede Technologie in einer eigenen Datei zu konfigurieren. Dies trägt zu einem direkterem Development Workflow, gesteigerter Nachvollziehbarkeit und geringerem Wartungsaufwand bei.

6.1.6 Dev Services

Ein zentraler Bestandteil des Quarkus-Frameworks sind die „Development Services“. Diese ermöglichen die automatische Bereitstellung von vorkonfigurierten Services in Entwicklungs- und Testumgebungen. Die Development Services unterstützen eine Vielzahl an Technologien. Dev Services in Quarkus können automatisch die benötigten Docker-Container für Datenbanken, Messaging-Systeme usw. starten. Technologien, wie Elasticsearch. [4] Datenbanken, wie PostgreSQL im Container. [2]

6.1.7 GraalVM und Quarkus

Die native Image-Unterstützung von Quarkus, ermöglicht durch GraalVM, führt zu signifikant schnelleren Startzeiten und reduziertem Speicherverbrauch. In der Praxis bedeutet dies, dass Microservices, die mit Quarkus entwickelt wurden, schneller hochfahren und weniger Ressourcen verbrauchen, was besonders wichtig in Cloud-basierten und containerisierten Umgebungen ist. Dies kann auch die Skalierbarkeit verbessern, da Anwendungen schneller gestartet und gestoppt werden können, um auf Laständerungen zu reagieren. Allerdings erfordert die Erstellung von nativen Images eine sorgfältige Abwägung, da sie den Build-Prozess komplizieren und die Kompatibilität mit bestimmten Bibliotheken beeinträchtigen können.

6.1.7.1 Vergleich von JVM mit nativer Kompilierung

Im Buch *Supercharge Your Applications with GraalVM* wurden die Startzeiten und Docker Image Größen von verschiedenen Auslieferungsarten miteinander verglichen. Dabei ist deutlich zu sehen, dass die nativ kompilierten Auslieferungsarten signifikant schnellere Startzeiten und geringere Docker Image Größen hervorbringen.

6.1.8 Vergleich zu Alternativen wie Spring-Framework

Im Buch „*Beginning Quarkus Framework: Build Cloud-Native Enterprise Java Applications and Microservices*“, das am 17. September 2020 erschien, schrieb der Autor Tayo Koleoso: „Der Markt wird wohl vom Spring Framework dominiert, wobei Spring Boot seine Flaggschiff-Plattform für Mikroservices ist.“ [6, S. 1]

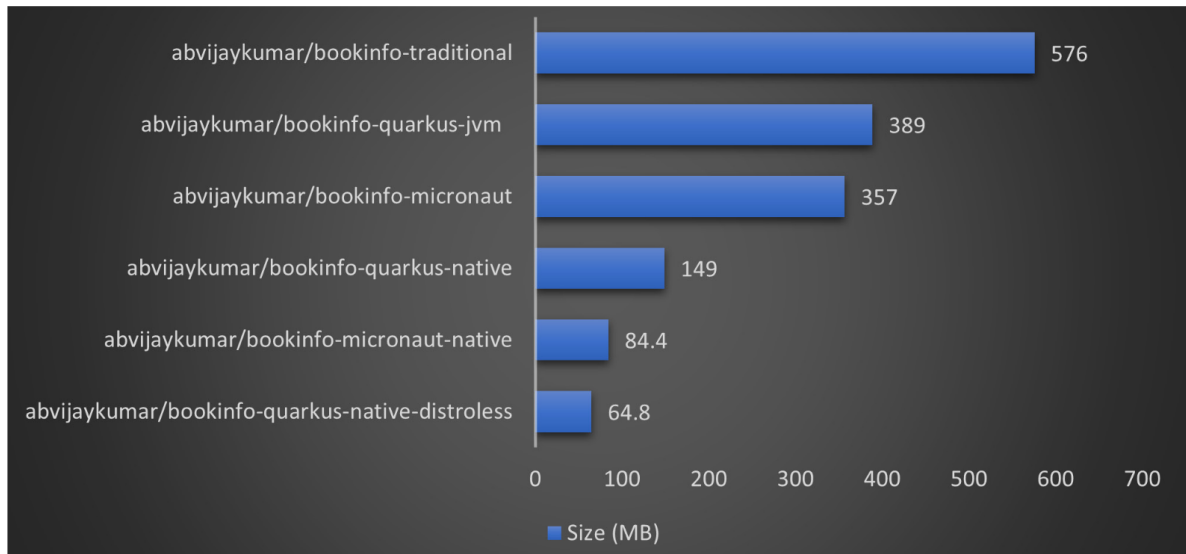


Figure 10.14 – Chart comparing the size of Docker images

Abbildung 6.1: Vergleich

[7, S. 313]

6.2 Hibernate

6.2.1 Hibernate ORM

Hibernate ORM ist ein Teil der Hibernate-Bibliothek und ermöglicht die objektrelationale Abbildung von Java-Objekten auf relationale Datenbanken. Es bietet eine einfache Möglichkeit, Datenbankoperationen wie das Erstellen, Lesen, Aktualisieren und Löschen von Datensätzen durchzuführen. Hibernate ORM (Object-Relational Mapping) ist ein wichtiger Bestandteil der Java-Technologie, der es Entwicklern ermöglicht, relationale Datenbanken effizient mit Java-Objekten zu verbinden. Das Hauptziel von Hibernate ORM ist die Vereinfachung des Entwicklungsprozesses bei der Datenpersistenz und Datenabruf, indem es eine Abstraktionsebene über die traditionelle JDBC (Java Database Connectivity) hinaus bietet. Diese Technologie ermöglicht es Entwicklern, sich auf die Geschäftslogik ihrer Anwendungen zu konzentrieren, ohne sich um die Komplexität von SQL-Abfragen und Datenbankverbindungen kümmern zu müssen. [12]

6.2.1.1 Panache

Hibernate ORM Panache ist eine Erweiterung von Hibernate ORM, die das Active-Record-Muster implementiert. Mit Panache können Datenbankzugriffe direkt in den Entitätsklassen selbst durchgeführt werden, ohne dass ein EntityManager injiziert werden muss. Dies ermöglicht eine vereinfachte und intuitivere Datenbankinteraktion. Hibernate ORM Panache ist eine Erweiterung von Hibernate, die das Arbeiten mit Datenbanken weiter vereinfacht. Es folgt dem Active Record-Modell, bei dem Datenbankoperationen wie das Finden, Einfügen oder Aktualisieren von Daten direkt auf den Entitätsklassen durchgeführt werden können. Diese Methode bietet eine intuitivere und weniger „boilerplate“-lastige Alternative zum traditionellen Ansatz von Hibernate ORM. [1, S. 128][11, S. 216]

6.2.2 Hibernate Search

Hibernate Search ist eine Erweiterung von Hibernate ORM, die die Volltextsuche in den Datenbankinhalten ermöglicht. Es bietet eine leistungsstarke Suchfunktionalität, die auf den Indexierungsfunktionen von Lucene basiert. Hibernate Search ermöglicht eine effiziente und schnelle Suche nach bestimmten Kriterien in den Daten. Hibernate Search ist eine leistungsstarke Erweiterung für Hibernate ORM ist. Sie bietet Volltextsuchfunktionen, die durch Integration mit Suchplattformen wie Apache Lucene, Elasticsearch und OpenSearch ermöglicht werden. Dies erlaubt es Entwicklern, komplexe Suchabfragen über ihre Datenbestände zu erstellen und effizient durchzuführen. [5]

6.2.2.1 Elasticsearch

Elasticsearch ist eine Open-Source-Suchmaschine, die auf Apache Lucene basiert und speziell für die Verarbeitung großer Datenmengen optimiert ist. Hibernate Search kann Elasticsearch als Backend verwenden, um die Suchfunktionalität zu verbessern und die Skalierbarkeit zu erhöhen. Elasticsearch, eine populäre Such- und Analyseengine, wird in Verbindung mit Hibernate Search verwendet, um eine erweiterte Indexierung und Suche von Daten zu ermöglichen. [3]

6.2.3 Hibernate Validator

Hibernate Validator ist ein Validierungsframework, das auf Bean Validation basiert. Es ermöglicht die Definition und Überprüfung von Validierungsregeln für die Eingabedaten. Hibernate Validator stellt sicher, dass die Daten, die in die Datenbank geschrieben werden, den definierten Regeln entsprechen. Hibernate Validator ist eine Implementierung der Jakarta Bean Validation-Spezifikation. Es ermöglicht es Entwicklern, Validierungsregeln in Form von Annotationen direkt in ihren Java-Klassen zu definieren. Diese Annotationen ermöglichen es, die Konsistenz und Validität von Daten zu gewährleisten, bevor sie in der Datenbank gespeichert werden. Entwickler können so etwa sicherstellen, dass E-Mail-Adressen einem gültigen Format entsprechen oder bestimmte Felder nicht leer sind. [1, S. 65–66][13]

6.2.4 Hibernate Envers

Hibernate Envers ist eine Bibliothek, die die Versionierung von Daten in Hibernate ermöglicht. Sie speichert alle Änderungen an den Entitätsdaten und ermöglicht es, zu einem beliebigen Zeitpunkt auf frühere Versionen der Daten zuzugreifen. Hibernate Envers bietet eine einfache Möglichkeit, die Historie von Datenänderungen zu verfolgen und wiederherzustellen. [12]

Kapitel 7

Methodik

7.1 Prozesse und Technologien für die Entwicklung

7.1.1 Semantic Versioning

Das Semantic Versioning basiert auf einem einfachen Schema, das klare Informationen über die Art der vorgenommenen Änderungen enthält. Die Versionsnummern bestehen aus den Major-, Minor und Patch-Nummern und bilden getrennt durch Punkte die vollständige Versionsnummer. Ein Major-Update deutet auf inkompatible API-Änderungen hin, ein Minor-Update auf rückwärtskompatible Neuerungen und ein Patch auf rückwärtskompatible Bugfixes. In Softwareprojekten mit vielen Abhängigkeiten ermöglicht SemVer Entwicklern, Abhängigkeiten effizient zu verwalten und zu aktualisieren, ohne unerwartete Inkompatibilitäten oder Fehler zu befürchten. SemVer bietet einen standardisierten Prozess für das Release-Management, was die Planung und Durchführung von Software-Veröffentlichungen vereinfacht. Viele moderne Tools im Softwareentwicklungsprozess sind für die Arbeit mit Semantic Versioning ausgelegt, was die Automatisierung und Effizienz in der Entwicklung verbessert. In Umgebungen, in denen viele Entwickler zusammenarbeiten, sorgt SemVer für eine gemeinsame Sprache und Verständnis bezüglich der Versionierung, was die Integration verschiedener Beiträge erleichtert. Auf der Grundlage des Semantic Versioning werden die Versionen der Komponenten der Service-Struktur festgelegt. <https://semver.org/>

7.1.2 Conventional Commits

Es wurden die Conventional Commit Vorgaben für die Commit-Nachrichten der Versionskontrolle verwendet. <https://www.conventionalcommits.org/en/v1.0.0/>

7.1.3 Git-Hooks

In den Git-Repositorys der Service-Struktur wurden Git-Hooks implementiert, die eine Validierung und Verknüpfung von JIRA-Ticket-Nummern in den Commit-Nachrichten realisiert.

7.2 Entwicklungsumgebung

Für die Realisierung der FMS wurde die IntelliJ IDE von JetBrains in der virtuellen Desktop-Umgebung des Unternehmens verwendet. Es stehen verschiedene Erweiterungen für die IntelliJ IDEA zur Verfügung, die Entwicklungsprozesse mit dem Quarkus-Framework vereinfachen und erweitern.

Kapitel 8

Implementierung

Die Implementierung des Fund-Management-Suite-Projekts umfasste die Migration des bestehenden Go-basierten ETF-Suchdienstes zu einer Java-Implementierung unter Verwendung des Quarkus-Frameworks. Die Maven Arche Types verwenden JDBI, JDBC und Liquibase. Daher wurde das Design des Datenbankschemas zunächst in Liquibase-Changelogs formuliert. Da die Entitäten Abhängigkeiten untereinander haben und Validierungslogik enthalten sollen, wurde das Hibernate ORM verwendet. Dies hat den Vorteil, dass Relationen und Validierungslogik nicht an mehreren Stellen implementiert werden müssen. Berechtigte Kritik an Hibernate ORM -> daher Datenbankschemaverwaltung in Bereitstellungsumgebungen durch Liquibase. Die Verwendung von Quarkus ermöglicht eine effiziente Entwicklung von Microservices mit automatisierten Tests und CI/CD-Integrationen. Weiterhin wurde ein Design-First-Ansatz mit OpenAPI-Spezifikation verfolgt, um eine standardisierte und leicht verständliche Service-Schnittstelle zu fördern. Die Implementierung umfasst die Automatisierung des Datenimports durch direkte Integrationen mit den Datenquellen, die Implementierung von automatisierten Tests und die Einbindung von CI/CD-Pipelines für eine effiziente Bereitstellung. Die Verwendung von Docker-Containern und die Integration von Überwachungsfunktionen wie Tracing und Health Checks sind ebenfalls Teil der Implementierung.

8.1 Erzeugen der Projekt-Strukturen

Mittels der Maven Archetypes kann die geforderte Struktur für Projekte im geforderten Aufbau und bereits konfigurierten Abhängigkeiten und Verknüpfungen erzeugt werden.

```
mvn archetype:generate -DarchetypeGroupId=com.  
  ↪ flatex.watergate -DarchetypeArtifactId=  
  ↪ microservice -DarchetypeVersion=1.5.1
```

```
mvn archetype:generate -DarchetypeGroupId=com.  
  ↪ flatex.watergate -DarchetypeArtifactId=  
  ↪ microservice-api -DarchetypeVersion=1.5.1
```

```
mvn archetype:generate -DarchetypeGroupId=com.  
  ↪ flatex.watergate -DarchetypeArtifactId=helm -  
  ↪ DarchetypeVersion=1.5.1
```

8.2 Entwicklungs- und Anpassungsprozesse

Um Konsistenz, Wartbarkeit und optimierte Leistung der FMS zu gewährleisten, sind die Entwicklungsprozesse für den FRS und den FDS methodisch definiert. Diese Prozesse spiegeln die architektonischen Entscheidungen und die strategische Bedeutung jedes Dienstes wider und gewährleisten hohe Zuverlässigkeit und Vorhersehbarkeit in ihren jeweiligen Funktionalitäten.

8.3 Fund Retrieval Service (FRS)

8.3.1 Datenvalidierung und -management

Der FRS ist darauf ausgelegt, Daten für ETFs und Fonds nahtlos aus verschiedenen Quellen zu importieren, während die Integrität, Korrektheit und Konsistenz der Daten gewährleistet wird. Um ein hohes Maß an Zuverlässigkeit und Konformität zu gewährleisten, ist es entscheidend, einen robusten Mechanismus zur Validierung sowohl der Syntax als auch der Semantik der eingehenden Daten zu etablieren.

8.3.1.1 Datenformat-Handling

Ziel: Unterstützung verschiedener Datenformate für den Importprozess, um Anpassungsfähigkeit und Flexibilität zu ermöglichen. Unterstützte Formate:

- JSON (für REST APIs)
- CSV (für Morningstar Direct Export)
- XML (für Morningstar REST API)
- (Potenziell um zusätzliche Formate in der Zukunft erweiterbar)

8.3.1.2 Datenvalidierungsframework

Die Syntax und Semantik von zu importierenden Daten muss validiert werden. Dafür eignet sich die Bibliothek Hibernate Validator.

Die Rohdaten der CSV-Datei müssen Um die Authentizität und Korrektheit der Daten zu gewährleisten, wird ein mehrstufiger Validierungsprozess eingesetzt. Dieser Prozess untersucht sowohl die Struktur (Syntax) als auch den Inhalt (Semantik) der importierten Daten.

Syntax-Validierung Stellt sicher, dass die importierten Daten den jeweiligen Formatstandards entsprechen. Zum Beispiel wird validiert, dass eine CSV den Regeln für kommagetrennte Werte entspricht. Semantik-Validierung Konzentriert sich auf den Inhalt und die inhärente Geschäftslogik der Daten. Dies beinhaltet, ist aber nicht beschränkt auf:

- Feldspezifische Validierungen, wie die Sicherstellung, dass ein finanzieller Wert nicht negativ ist.
- Kreuzfeldvalidierungen, wie die Überprüfung, dass ein Enddatum nach einem Startdatum liegt.
- Einhaltung spezifischer Geschäftsbeschränkungen oder -regeln.

8.3.2 Datenprozesse

8.3.2.1 CSV-Import von Morningstar Direct

Da die Exportziele von Morningstar Direct nicht mit Unternehmensvorgaben vereinbar waren, muss vorläufig der manuelle Export beibehalten werden. Die manuelle Anpassung der exportierten Daten ist nicht mehr erforderlich. Die exportierte CSV-Datei kann nun dem FRS mittels eines Service-Tickets in JIRA übermittelt werden. Das Service-Ticket muss manuell erstellt und die Datei angehängt werden. Bisher wurde nicht geklärt, inwiefern dieser Prozess in Zukunft ablaufen soll. Entsprechend könnte der bisherige Prozess und die verantwortlichen Personen bestehen bleiben. Dieser Prozess hat hohes Verbesserungspotenzial. Jackson Parsing Validate Mapping Persistierung

Kapitel 9

Ergebnisse

9.1 Bewertung der Implementierung

Während der Bearbeitungszeit konnten die Grundfunktionalitäten umgesetzt werden. Die Umsetzung ist gut strukturiert und durchdacht. Die Entwicklung kann dadurch fortgeführt und finalisiert werden. Während der Umsetzung wurden Probleme in der initialen Konzeption erkannt und diese entsprechend verbessert. Durch die Verwendung der reaktiven Hibernate-Bibliotheken könnte der Import-Prozess der Rohdaten beschleunigt werden. Hibernate Search unterstützt jedoch Hibernate Reactive nicht. Daher konnte eine Umstellung auf ein reaktives Hibernate nicht erfolgen.

9.2 Probleme und Lösungen

9.2.1 Syntaxfehler CSV Zeilenumbrüche

Während der Implementierung wurde festgestellt, dass die zu importierende CSV-Datei von der allgemeinen CSV-Syntax abweicht. Dies führte zu Abstürzungen und Fehlermeldungen während des Importvorgangs. Die Bezeichnungen in der Kopfzeile enthalten Zeilenumbrüche (LF). Diese Syntaxfehler müssen daher vor der Verarbeitung behoben werden. Dafür wird die Zeichenkette traversiert und die fehlerhaften Zeichen entfernt und durch Leerzeichen ersetzt.

9.2.2 Proxy und SSL-Zertifikate

Das Unternehmen verwendet einen Proxy, der verwendet werden muss. Der Proxy verwendet eigene SSL-Zertifikate und CA-Authoritäten, um verschlüsselten Netzwerkverkehr entschlüsseln und überprüfen zu können. Jeder Dienst muss diesen Proxy und die Zertifikate verwenden, was zusätzliche Konfigurationen erfordert.

9.2.3 Ausfall von CI/CD-Pipeline-Runners

Während der Entwicklungszeit sind die CI/CD-Pipeline-Runners ausgefallen, was die Entwicklung blockierte.

9.3 Vergleich mit den Anforderungen

In der Bearbeitungszeit konnten nicht alle Anforderungen umgesetzt werden.

Kapitel 10

Auswertung

Die Evaluation des Projekts umfasste das Testen der Funktionalität und Leistung des Services. Die Verwendung von Liquibase für die Datenbankänderungsverwaltung und die Implementierung von Best Practices, wie sie in den Quarkus-Workshops des Unternehmens vorgeschlagen wurden, trugen zur Sicherstellung einer soliden und wartbaren Codebasis bei. Die Entscheidungen für die Datenbankstruktur, die Verwendung von Maven-Archetypen und die Auslieferung als Docker-Images wurden auf Basis von Branchenstandards und den spezifischen Anforderungen des Projekts getroffen. Die erfolgreiche Migration zu Java und die Integration in das Quarkus-Framework werden als wesentliche Erfolgsfaktoren für die Modernisierung und Skalierbarkeit der Anwendung angesehen.

Kapitel 11

Ausblick

- Zukünftig sollte die „Morningstar Direct“ GUI-Anwendung durch eine direkte REST-Schnittstelle, die von Morningstar bereitgestellt wird, ersetzt werden.
- (Potenzielle zukünftige Überlegung) Integration mit anderen internen Mikroservices nach Bedarf.
- (Potenzielle zukünftige Überlegung) Implementierung asynchroner Endpunkte mit Kafka.

Kapitel 12

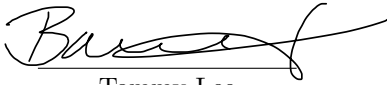
Schlussfolgerung

Der vorliegende Ansatz bietet eine umfassende Modernisierung eines ETF-Suchdienstes, indem die Vorteile des Quarkus-Frameworks genutzt werden und durch die Trennung in zwei Microservices eine effiziente und wartbare Architektur geschaffen wird. Die Automatisierung des Datenimports und die verbesserte Datenverarbeitung und -bereitstellung versprechen eine erhöhte Benutzerzufriedenheit und eine Reduktion der Fehleranfälligkeit. Die Entwicklung einer Service-Struktur zur Suche, Filterung und Abfrage von ETF- und Fondsdaten unter Verwendung des Quarkus-Frameworks hat zu einem modernen, skalierbaren und wartbaren System geführt. Die Implementierung folgte einem Design-First-Ansatz, wobei Sicherheitsaspekte und Datenintegrität im Vordergrund standen. Durch die Verwendung von PostgreSQL, Liquibase und anderen Best Practices der Software-Entwicklung konnte ein zuverlässiger und effizienter Service geschaffen werden, der die Anforderungen des Projekts erfüllt.

Selbstständigkeitserklärung

Hiermit versichere ich, Tommy-Lee Bannert, dass ich die vorliegende Bachelorarbeit mit dem Titel „Entwicklung einer Service-Struktur zur Suche, Filterung und Abfrage von ETF- und Fondsdaten“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist bislang nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Zwickau, den 5. Januar 2024



Tommy-Lee
Bannert

Literaturverzeichnis

- [1] A. S. Bueno und J. Porter, *Quarkus Cookbook*. O'Reilly Media, Inc., Juli 2020.
- [2] „Dev Services for Databases.“ (), Adresse: <https://quarkus.io/guides/databases-dev-services> (besucht am 26.10.2023).
- [3] „Dev Services for Elasticsearch.“ (), Adresse: <https://quarkus.io/guides/elasticsearch-dev-services> (besucht am 01.11.2023).
- [4] „Dev Services Overview.“ (), Adresse: <https://quarkus.io/guides/dev-services> (besucht am 26.10.2023).
- [5] „Hibernate Search guide.“ (), Adresse: <https://quarkus.io/guides/hibernate-search-orm-elasticsearch> (besucht am 01.11.2023).
- [6] T. Koleoso, *Beginning Quarkus Framework: Build Cloud-Native Enterprise Java Applications and Microservices*. Berkeley, CA: Apress, Sep. 2020, 311 S.
- [7] A. B. V. Kumar, *Supercharge Your Applications with GraalVM*. Aug. 2021, 356 S.
- [8] F. I. Lessambo, „Exchange traded funds (ETF),“ in *International Finance*. Cham: Springer International Publishing, 2021, S. 117–124.
- [9] „Quarkus and Maven.“ (), Adresse: <https://quarkus.io/guides/maven-tooling> (besucht am 20.11.2023).
- [10] „Simplified Hibernate ORM with Panache - Quarkus.“ (), Adresse: <https://quarkus.io/guides/hibernate-orm-panache> (besucht am 23.10.2023).
- [11] M. Štefanko und J. Martiška, *Quarkus in Action: Build resilient and scalable, cloud-native, enterprise Java applications using the Quarkus framework*. Manning Early Access Program (MEAP) V4. Manning Publications Co, Jan. 2023, 525 S.
- [12] „Using Hibernate ORM and Jakarta Persistence.“ (), Adresse: <https://quarkus.io/guides/hibernate-orm> (besucht am 23.10.2023).
- [13] „Validation with Hibernate Validator.“ (), Adresse: <https://quarkus.io/guides/validation> (besucht am 23.10.2023).

Abkürzungsverzeichnis

API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Delivery
DTO	Data Transfer Object
REST	Representational State Transfer
XML	Extensible Markup Language
CSV	Comma-separated values
FMS	Fund Management Suite
FC	Fund Commons
FRS	Fund Retrieval Service
FDS	Fund Discovery Service
FDS-API	Fund Discovery Service API
BI	Business Intelligence
DIT	Development Integration Testing
QIT	Quality Integration Testing
UAT	User Acceptance Testing
PROD	Production
DBA	Datenbank-Administrator
ESG	Environmentail, Social and Governance
GUI	Graphical User Interface
FTP	File Transfer Protocol

Abbildungsverzeichnis

2.1	Suchmaske für ETFs und Fonds	15
2.2	Ergebnisliste	16
2.3	App Ansicht	17
2.4	App Ansicht	17
3.1	Suchmaske für ETFs und Fonds	20
3.2	Suchmaske für ETFs und Fonds	20
3.3	Suchmaske für ETFs und Fonds	21
5.1	Anwendungsfälle der REST-Schnittstelle	31
6.1	Vergleich	36

Tabellenverzeichnis

2.1	Charakteristiken der CSV-Datei	9
2.2	Mapping-Tabellen	10
2.3	Endpunkte des aktuellen Service	12
4.1	Funktionale Anforderungen	23
4.2	Nicht-funktionale Anforderungen	24