



Handelskammer Bremen
für Bremen und Bremerhaven

Abschlussprüfung Sommer 2025
Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit
BotHQ – ein Discord-Bot-Framework



Prüfungsausschuss: Bernd Heidemann
Katrín Deeken

Abgabedatum: 30. Mai 2024

Prüfungsbewerber: Philipp Batelka
Jan Mahnken
Daniel Quellenberg
Fabian Reichwald
Justus Sieweke
Christopher Spencer

Ausbildungsbetrieb: Europaschule Schulzentrum SII Utbremen
Meta-Sattler-Str. 33
28217 Bremen

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	III
Quelltextverzeichnis	III
1 Einleitung	1
1.1 Discord und Discord Bots	1
1.2 Projektbeschreibung	2
1.3 Projektziel	2
2 Projektumfeld	2
2.1 Produktportfolio der HiTec GmbH	2
2.2 Entstehung der Projektidee	3
3 Projektvorbereitung	3
3.1 Ist-Analyse	3
3.2 Soll-Analyse	4
3.3 Detaillierte Zielsetzung des Projekts	4
3.4 Anforderungen und Technologien	5
4 Projektdurchführung	7
4.1 Vorgehensmodell	7
4.2 Umsetzung	8
4.2.1 Klassenmodell	8
4.2.2 Datenhaltung	10
4.2.3 Datenbankstruktur und Datenbankmodell	10
4.2.4 Beziehungen zwischen den Tabellen	12
4.2.5 Detaillierte Beschreibung der Beziehungen	12
4.2.6 Design Patterns	13
4.3 Qualitätssicherung	14
4.3.1 Code Reviews	14
4.3.2 Teststrategien	14
4.3.3 Testfälle	15
4.3.4 Zusammenfassung der Testergebnisse	15
5 Wirtschaftliche Betrachtung	15
5.1 Marktuntersuchung	15
5.1.1 Zielgruppe	15
5.1.2 Marktvolumen und Marktpotential	16
5.1.3 Konkurrenzanalyse	17
5.2 Marketingmix	17

5.3	Kostenplanung	18
5.3.1	Personalkosten	18
5.3.2	Sachmittelkosten	19
5.3.3	Gesamtkosten des Projektes	20
5.1	Wirtschaftlichkeitsberechnung	20
5.1.1	Gewinnschwellenberechnung	20
5.1.2	Amortisationsrechnung	21
6	Projektabschluss	21
6.1	Erreichung des Projektziels	21
6.2	Änderungen zur anfänglichen Planung	22
6.3	Fazit	23
A	Anhang	IV
A.1	Glossar	IV
A.2	Literaturverzeichnis	VIII
A.3	Quellcode	IX

Abbildungsverzeichnis

1	Gantt-Diagramm der Projektphasen	9
2	Datenbankmodell	10

Tabellenverzeichnis

1	Stundensätze der Projektmitglieder	18
2	Gesamtkosten pro Woche	18
3	Hardware-Kosten	19
4	Software-Kosten	19
5	Zusatzkosten	19
6	Gesamtkosten Sachmittel	20
7	Gesamtkosten des Projektes	20

Quelltextverzeichnis

1	Singleton-Entwurfsmuster in Java	13
2	Factory-Entwurfsmuster in Java	13
3	Observer-Entwurfsmuster in Java	14
4	Beispielcode für einen ReadyListener in JDA	IX

1 Einleitung

1.1 Discord und Discord Bots

Discord ist eine kostenlose Kommunikationsplattform, die ursprünglich für die Gaming-Community entwickelt wurde, mittlerweile jedoch von einer Vielzahl von Communities und Organisationen genutzt wird. Die Plattform bietet umfassende Funktionen für Text-, Sprach- und Video-Kommunikation sowie für die Organisation von Gruppen.

Ein grundlegendes Konzept von Discord ist der *Server*. Ein Server ist eine dedizierte Instanz, die von einer Gruppe von Benutzern genutzt wird, um zu kommunizieren und Inhalte zu teilen. Jeder Server kann mehrere *Kanäle* enthalten, die weiter in Text- und Sprachkanäle unterteilt sind. Textkanäle dienen der schriftlichen Kommunikation und dem Austausch von Dateien, während Sprachkanäle für Echtzeit-Audio-Gespräche verwendet werden.¹ Discord nutzt eine *Client-Server-Architektur* ▶², bei der alle Daten über zentrale Server verarbeitet und gespeichert werden. Diese Server sind in verschiedenen Rechenzentren weltweit verteilt, um niedrige Latenzzeiten und hohe Verfügbarkeit zu gewährleisten. Die Kommunikation zwischen dem Discord-Client (verfügbar für Desktop, Web und Mobilgeräte) und den Servern erfolgt über das *HTTPS-Protokoll* ▶ und *WebSocket-Verbindungen* ▶.³

Ein besonders mächtiges Feature von Discord ist die Unterstützung von *Bots*. Bots sind automatisierte Programme, die über die *Discord API* ▶ interagieren können. Die API bietet eine Vielzahl von *Endpunkten* ▶, die es Entwicklern ermöglichen, Nachrichten zu senden, Benutzerinformationen abzurufen, Kanäle zu verwalten und auf Ereignisse zu reagieren. Bots werden häufig genutzt, um Server zu moderieren, Spiele zu integrieren, Musik abzuspielen und vieles mehr.

Im Gegensatz zur *Discord API* ▶ handelt es sich bei der *Java Discord API (JDA)* um eine Java-Bibliothek, die eine einfache Nutzung der Discord API ermöglicht. Sie abstrahiert viele der komplexen Aspekte der API und bietet eine benutzerfreundliche Schnittstelle für die Bot-Entwicklung.

Discord legt großen Wert auf Sicherheit und Datenschutz. Alle Datenübertragungen sind mit TLS (Transport Layer Security) verschlüsselt, um die Vertraulichkeit und Integrität der Daten zu gewährleisten. Benutzer haben die Kontrolle über ihre Privatsphäre-Einstellungen und können festlegen, wer sie kontaktieren kann und welche Informationen öffentlich sichtbar sind. Discord hat auch Richtlinien und Maßnahmen zum Schutz vor Spam, Missbrauch und Belästigung, um eine positive und sichere Umgebung für alle Benutzer zu gewährleisten.⁴

1 Discord 2024a.

2 Alle Begriffe, die im Glossar erklärt werden, sind mit diesem Zeichen ▶ gekennzeichnet und führen per Klick direkt zur Erklärung.

3 Discord 2024b.

4 Discord 2024c.

1.2 Projektbeschreibung

In dieser Arbeit wird die Entwicklung eines modularen Discord-Bot-Frameworks als *Cloud-Service* ▶ beschrieben. Dieses Projekt zielt darauf ab, ein innovatives und modulares Framework für Discord-Bots zu entwickeln, das auf der *Java Discord API* ▶⁵ basiert. Das Framework wird es den Nutzer:innen ermöglichen, benutzerdefinierte Plugins zu erstellen, zu laden und über eine leicht zu bedienende Weboberfläche zu verwalten. Um den Betrieb und die Wartung zu vereinfachen, wird das Framework als Cloud-Service angeboten, sodass die Nutzer keine eigene Hosting-Infrastruktur bereitstellen müssen. Das Projekt beinhaltet die Entwicklung einer *REST-API* ▶ für die Kommunikation zwischen der Weboberfläche und dem Backend, sowie die Implementierung von vorgefertigten Plugins, die grundlegende Bot-Funktionalitäten abdecken. Durch diese Lösung wird eine hohe Flexibilität und Erweiterbarkeit gewährleistet, um den unterschiedlichen Anforderungen der Nutzer gerecht zu werden.

1.3 Projektziel

Das Hauptziel dieses Projekts ist es, ein leistungsfähiges und flexibles Discord-Bot-Framework zu entwickeln, das sowohl für Anfänger als auch für fortgeschrittene Nutzer:innen zugänglich ist. Das Framework soll es ermöglichen, verschiedene Plugins nahtlos zu integrieren und zu verwalten, ohne dass tiefgehende technische Kenntnisse erforderlich sind. Ein weiteres Ziel ist die Bereitstellung als Cloud-Service, um den Nutzern die Komplexität des eigenen Hostings abzunehmen und gleichzeitig eine hohe Verfügbarkeit und Skalierbarkeit sicherzustellen. Dank der REST-API können Benutzeränderungen und Konfigurationen in Echtzeit verarbeitet werden. Die benutzerfreundliche Oberfläche soll Nutzer in die Lage versetzen, ihre Bots einfach zu konfigurieren und anzupassen, was die allgemeine Benutzererfahrung erheblich verbessert.

2 Projektumfeld

2.1 Produktportfolio der HiTec GmbH

Die HiTec GmbH ist ein mittelgroßes IT-Systemhaus und seit 15 Jahren mit den folgenden Produkten und Dienstleistungen am Markt vertreten:

- **Entwicklung** Erstellung eigener Softwareprodukte und individueller Softwarelösungen
- **IT-Systembereich** Lieferung und Verkauf von IT-Komponenten sowie die Planung und Installation von Netzwerken
- **Consulting** Beratung von Anwendern zu neuen Technologien, Anwendungen und IT-Security
- **Support** Betreuung von IT-Systemen (sowohl Hard- als auch Software)

⁵ Austin Keener 2024a,b.

Alle Dienstleistungen werden von eigenen Abteilungen mit spezialisiertem Personal ausgeführt. IT-Dienstleistungen werden in der Regel von der IT-Abteilung erbracht, während die Entwicklung von Softwareprodukten von der Entwicklungsabteilung durchgeführt wird.

2.2 Entstehung der Projektidee

Die Projektidee entstand im Rahmen eines Auftrags, nach dem ein Schulungsprojekt durchzuführen war, das zur Entwicklung vorgeschriebene Frameworks und Technologien nutzt. Das Thema war hierbei dem bearbeitenden Personal freigestellt. Die Idee eines modularen Discord-Bot-Frameworks wurde von einem Teammitglied vorgeschlagen, das in der Vergangenheit Erfahrung mit der Entwicklung von Discord-Bots gesammelt hatte. Diese Idee wurde von der Projektleitung aufgegriffen und als Projektidee angenommen.

3 Projektvorbereitung

3.1 Ist-Analyse

Derzeit verfügt die HiTec GmbH nicht über ein Discord-Bot-Framework oder anderweitige Dienstleistungen in Bezug auf Discord. Dies hat zur Folge, dass die HiTec GmbH keine Kunden betreuen kann, die Discord zur Kommunikation nutzen, weder Intern noch mit Kunden.

Des Weiteren wäre die HiTec GmbH bei eigener Nutzung von Discord zu Kommunikationszwecken von Dienstleistungen von Drittanbietern abhängig, um die Anwendungsmöglichkeiten von Discord für den eigenen Anwendungsfall anzupassen und zu optimieren. Folgerichtig ist es notwendig, ein eigenes Discord-Bot-Framework zu entwickeln, das die Anforderungen der Kunden erfüllt und die Nutzung von Discord für die HiTec GmbH optimiert.

Im Vorfeld des Projekts wurden Interviews mit aktuellen Nutzern geführt, um die Anforderungen und Erwartungen an das Discord-Bot-Framework zu ermitteln. Dabei wurden folgende Schwachstellen identifiziert:

- **Ineffiziente Plugin-Verwaltung:** Die derzeitige Plugin-Verwaltung ist umständlich und zeitaufwändig, da Plugins manuell hinzugefügt und konfiguriert werden müssen.
- **Fehlende Automatisierungsprozesse:** Es fehlen Automatisierungsprozesse, um wiederkehrende Aufgaben zu vereinfachen und zu beschleunigen.
- **Unzureichende Benutzeroberflächen:** Die Benutzeroberflächen sind unübersichtlich und nicht intuitiv zu bedienen, was zu Fehlern und Verzögerungen führt.
- **Mangelnde Dokumentation:** Die Dokumentation des Discord-Frameworks ist unvollständig und schwer verständlich, was die Nutzung erschwert.

- **Fehlende Schulungen:** Es gibt keine Schulungen oder Schulungsmaterialien, um neue Nutzer in die Nutzung des Discord-Frameworks einzuführen.
- **Mangelnde Integration:** Das Discord-Framework ist nicht vollständig in die bestehenden Systeme der HiTec GmbH integriert, was zu Inkonsistenzen und Datenverlust führen kann.
- **Fehlende Erweiterbarkeit:** Das Discord-Framework ist nicht einfach erweiterbar, um neue Funktionen und Anforderungen hinzuzufügen.

Schnell ließ sich daraus der anvisierte Soll-Zustand ableiten, der im folgenden Abschnitt beschrieben wird.

3.2 Soll-Analyse

Der angestrebte Soll-Zustand sieht vor, dass ein Cloud-Service der HiTec GmbH für Discord-Bots existiert, der auf der *Java Discord API* ▶⁶ basiert. Dieser Service soll den Nutzern über eine Weboberfläche ermöglichen, Plugins zu ihren Discord-Servern hinzuzufügen und zu verwalten. Zudem soll ein Framework entwickelt werden, das das einfache Hinzufügen neuer Plugins durch die HiTec GmbH ermöglicht. Dies erlaubt es der HiTec GmbH, eine modulare Dienstleistung für Kunden anzubieten, die Discord zu Kommunikationszwecken nutzen, und diese auf Wunsch der Kunden flexibel anzupassen.

3.3 Detaillierte Zielsetzung des Projekts

Der Service wird den Kunden als Webservice zur Verfügung gestellt, sodass sie ihre Server und Plugins eigenständig verwalten können. Die Plattform wird responsive gestaltet, mit besonderem Augenmerk auf benutzerfreundlichem Design und fehlerfreier Funktionalität.

Beim Öffnen der Webseite wird der Nutzer aufgefordert, sich zu authentifizieren. Dieser Authentifizierungsdienst wird von der Discord-API bereitgestellt.

Auf der Webseite soll der Nutzer eine Übersicht seiner Discord-Server sowie der verfügbaren Plugins sehen können. Die Liste der Server wird von der Discord-API bereitgestellt, während die Liste der Plugins von der eigenen Backend-API des Services stammt. Den Nutzern soll es ermöglicht werden, Plugins und deren Einstellungen für jeden Server individuell und in Echtzeit zu verwalten. Eine Datenbank im Backend des Services speichert die vorgenommenen Einstellungen ab und das Backend übernimmt die Konfigurationen automatisch.

Zusätzlich soll ein Framework zum Erstellen von Plugins für den Inhaber des Services bereitgestellt werden. Entwicklern soll es ermöglicht werden, Code zu schreiben, der mit einem Discord-Server interagiert. Zudem soll der Entwickler eine Konfigurationsdatei bereitstellen können, die Parameter enthält, die vom Endnutzer verändert werden können. Diese Konfigurationsdatei wird beim Aufruf der Plugin-Einstellungen durch den Nutzer an das Frontend übermittelt, welches daraus dynamisch ein Interface für den Nutzer generiert.

⁶ Austin Keener 2024a,b.

3.4 Anforderungen und Technologien

Zur Durchführung des Projekts ist die Nutzung der folgenden Tools und Technologien vorgesehen:

Java Spring Framework ist ein Open-Source-Framework zur Entwicklung von eigenständigen Anwendungen, die auf der Java Virtual Machine (JVM) laufen. Spring bietet eine Abhängigkeitsinjektionsfunktion, bei der Objekte ihre eigenen Abhängigkeiten definieren können, die dann später injiziert werden. Dadurch können modulare Anwendungen aus lose gekoppelten Komponenten erstellt werden, was sich sehr gut für Netzanwendungen und Mikroservices eignet. Zusätzlich beinhaltet das Framework auch Funktionen und Unterstützung für klassische Aufgaben wie zum Beispiel Ausnahmebehandlung, Typumsetzung, Internationalisierung und weitere.

Java Spring Boot ist ein Tool, das die Entwicklung von Anwendungen mit dem Java Spring Framework durch drei Kernfunktionen vereinfacht:

- Autokonfiguration
- Ein »Opinionated«-Konfigurationsansatz
- Die Fähigkeit, eigenständige Anwendungen zu erstellen

Eine REST API ist ein Interface, das den Beschränkungen von REST unterliegt. REST steht für »Representational State Transfer« und wurde vom US-amerikanischen Informatiker Roy Fielding entwickelt. Es handelt sich dabei um eine Sammlung von Architekturbeschränkungen mit einem Schwerpunkt auf Maschine-zu-Maschine-Kommunikation. Bei einer Anforderung über eine REST-API werden Ressourcen in Form eines HTTP-Formats an einen Client oder Endpunkt geschickt. RESTful-APIs zeichnen sich durch folgende Merkmale aus:

- Client/Server-Architektur, die Anforderungen per HTTP verwaltet
- Zustandslose Client/Server-Kommunikation
- Cachingfähige Daten
- Eine einheitliche Schnittstelle

Angular ist ein auf TypeScript basierendes Front-End-Framework zur Entwicklung von Webanwendungen. Es wird als Open-Source-Plattform von einer Community aus Einzelpersonen und Firmen, angeführt durch Google, entwickelt. Angular verwendet eine Komponentenarchitektur, um Webanwendungen zu erstellen, bei der jeder Teil der Anwendung in unabhängige, wiederverwendbare Komponenten aufgeteilt wird. Zusätzlich liefert Angular viele Tools zur Webentwicklung, wie Routing- und Navigationssysteme, eine leistungsfähige Templating-Sprache und weitere. Vorteile von Angular sind unter anderem:

- Reduzierter Code
- Einfache Refaktorisierung
- Gute Testbarkeit

- Wiederverwendbare Code-Komponenten
- Große Community und ständige Weiterentwicklung

PostgreSQL, allgemein »Post-GRES« ausgesprochen, ist eine Open-Source-Datenbank, die für ihre Zuverlässigkeit, Flexibilität und Unterstützung offener technischer Standards bekannt ist. Im Gegensatz zu anderen RDBMS (Relational Database Management Systems) unterstützt PostgreSQL sowohl relationale als auch nicht-relationale Datentypen. Dies macht sie zu einer der konformsten, stabilsten und ausgereiftesten relationalen Datenbanken, die aktuell verfügbar sind. Es wurde ursprünglich 1986 als Nachfolger von INGRES, einem in den frühen 1970er Jahren begonnenen Open-Source-SQL-Projekt für relationale Datenbanken, entwickelt und war die Idee von Michael Stonebraker, einem Informatikprofessor in Berkeley.

Discord ist ein Service für die Online-Kommunikation, Instant Messaging, Sprach- und Videokonferenzen. Es ist sowohl als Webanwendung als auch als Client-Software auf allen gängigen Betriebssystemen nutzbar. Ursprünglich war Discord vornehmlich für die Videospiel-Szene gedacht, um während des Spielens Informationen mit Mitspielern austauschen zu können. Inzwischen wird Discord jedoch von einer großen Anzahl an Online-Communities genutzt. Das Konzept von Discord stammt von Jason Citron, der OpenFeint, eine Social-Gaming-Plattform für mobile Spiele, gegründet hatte, und von Stanislav Vishnevskiy, dem Gründer von Guildwork, einer weiteren Social-Gaming-Plattform.

Die Java Discord API (JDA) ist eine umfassende Programmbibliothek, die Entwicklern ermöglicht, Discord-Bots in der Programmiersprache Java zu erstellen und zu verwalten. JDA bietet eine hohe Abstraktionsebene, um die Kommunikation zwischen einem Java-Programm und der Discord-API zu erleichtern. Mit JDA können Entwickler auf verschiedene Funktionen und Ereignisse in Discord zugreifen, wie das Empfangen und Senden von Nachrichten, das Verwalten von Servern, Kanälen und Benutzern sowie das Reagieren auf verschiedene Interaktionen innerhalb von Discord.

Zu den Hauptfunktionen der JDA gehören:

- **Nachrichtenverwaltung:** Senden, Bearbeiten und Löschen von Nachrichten in Textkanälen.
- **Benutzerverwaltung:** Abrufen von Benutzerinformationen, Verwalten von Rollen und Berechtigungen.
- **Ereignisbehandlung:** Reagieren auf Ereignisse wie Nachrichtenempfang, Benutzerbeitritt und -austritt, Reaktionen und vieles mehr.
- **Sprachunterstützung:** Unterstützung für die Audioübertragung in Sprachkanälen.

JDA ist besonders für seine einfache Handhabung und umfangreiche Dokumentation bekannt, die es sowohl Anfängern als auch erfahrenen Entwicklern ermöglicht, leistungsstarke und funktionsreiche Discord-Bots zu erstellen. Die Bibliothek wird aktiv gepflegt und weiterentwickelt, um stets mit den neuesten Änderungen und Funktionen der Discord-API kompatibel zu bleiben.

4 Projektdurchführung

4.1 Vorgehensmodell

Für die Projektdurchführung wurde das agile Vorgehensmodell *Scrum* ⁷ gewählt. Scrum ist ein weit verbreitetes Framework für die agile Softwareentwicklung, das iterative und inkrementelle Prozesse unterstützt. Es besteht aus festen Rollen, Ereignissen und Artefakten, die bei der iterativen Softwareentwicklung unterstützen. Aufgrund der kurzen Projektdauer und der unregelmäßigen Verfügbarkeit der Teammitglieder wurde eine angepasste Version von Scrum verwendet, die die Flexibilität und Anpassungsfähigkeit des Frameworks beibehält, aber an die spezifischen Anforderungen des Projekts angepasst ist.

Der Methodologie von Scrum folgend wurde das Projekt in drei Sprints unterteilt, wobei die Sprints abweichend von der Regel zwischen drei und sechs Wochen dauerten. Diese zusätzliche Flexibilität ermöglichte es dem Team, sich besser an die Anforderungen und den Fortschritt des Projekts anzupassen. Jeder Sprint wurde mit einem Sprint-Planungsmeeting gestartet, in dem die Ziele und Aufgaben für den Sprint festgelegt wurden. Während des Sprints fanden regelmäßige Weekly-Meetings⁷ statt, um den Fortschritt zu überprüfen und Hindernisse zu beseitigen. Am Ende jedes Sprints wurde ein Sprint Review Meeting abgehalten, um die erreichten Ziele zu präsentieren und Feedback zu sammeln. In Abbildung 1 auf Seite 9 sind die drei Sprints und ihre Phasen dargestellt.

Normalerweise sind Rollen und Verantwortlichkeiten in Scrum klar definiert. In diesem Projekt wurden die Rollen jedoch flexibel gehandhabt, da die Teammitglieder unterschiedliche Fähigkeiten und Verfügbarkeiten hatten. Die Teammitglieder übernahmen je nach Bedarf verschiedene Rollen, um sicherzustellen, dass die Anforderungen des Projekts erfüllt wurden. Die wichtigsten Rollen waren:

- **Projektleitung:** Scrum sieht die Rolle des Projektleiters nicht vor, dennoch erschien es dem ganzen Team wichtig, eine Person zu haben, die verantwortlich für die Planung, Koordination und Überwachung des Projekts zeichnete – Christopher Spencer. Er stellte sicher, dass die Ziele erreicht und Hindernisse beseitigt wurden.
- **Backend:** Die Implementierung der Backend-Funktionalität und die Integration von Datenbanken und APIs programmierten in erster Linie Christopher Spencer und Justus Sieweke.
- **Frontend:** Die Implementierung der Benutzeroberfläche und die Interaktion mit dem Backend erledigte Jan Mahnken.
- **Plugins:** Die Implementierung von Plugins des Bot-Frameworks wurde in Arbeitsteilung von Philipp Batelka, Jan Mahnken, Daniel Quellenberg, Fabian Reichwald, Justus Sieweke und Christopher Spencer erledigt. Die Entwickler arbeiteten eng zusammen, um die Anforderungen umzusetzen und den Code zu überprüfen.

⁷ Scrum sieht zwar Daily-Meetings vor, doch aufgrund der unregelmäßigen Verfügbarkeit der Teammitglieder wurden wöchentliche Meetings abgehalten.

- **Dokumentation und Marketing:** Verantwortlich für die Erstellung und Pflege der Projektdokumentation, der Projektpräsentation und des Marketingpitches waren Philipp Batelka, Daniel Quellenberg und Fabian Reichwald. Sie sorgten dafür, dass alle wichtigen Informationen und Entscheidungen dokumentiert wurden.

Ein großer Vorteil der verteilten Verantwortlichkeiten lag darin, dass die unterschiedlichen Fähigkeiten und Erfahrungen der Teammitglieder optimal genutzt werden konnten. Dies ermöglichte es jedem Einzelnen, das Team genau in den Aufgaben zu unterstützen, für die sie am besten geeignet waren.

Als Projektmanagement-Tool wurde *Jira* ► eingesetzt. Die Planung, Durchführung und Überwachung des Projekts wurde dadurch klar und transparent. In Jira wurden die Anforderungen, Aufgaben, Sprints und Fortschritte des Projekts verwaltet. Dies ermöglichte es dem Team, den Überblick über den Projektfortschritt zu behalten, Hindernisse zu identifizieren und die Zusammenarbeit zu verbessern. Für die Versionsverwaltung waren *Git* ► in Verbindung mit *GitHub* ► die Werkzeuge der Wahl. Git ermöglichte es dem Team, den Code effizient zu verwalten, Änderungen nachzuverfolgen und Versionskonflikte zu lösen. GitHub wurde als zentrale Plattform für die Zusammenarbeit genutzt, um den Code zu teilen, *Pull-Requests* ► zu überprüfen und Feedback zu geben. Die Entscheidung, viele Repositories zu erstellen, ermöglichte es, den Code für verschiedene Komponenten und Module getrennt zu halten und die Verantwortlichkeiten klar zu trennen.

4.2 Umsetzung

4.2.1 Klassenmodell

Das Klassenmodell zeigt die wichtigsten Klassen, ihre Attribute und Methoden sowie die Beziehungen zwischen den Klassen.

Zu den Hauptklassen gehören:

User Repräsentiert einen Benutzer der Anwendung mit Attributen wie `id`, `username`, `password` und `roles`.

Plugin Repräsentiert ein Plugin mit Attributen wie `id`, `name`, `description`, `enabled` und `version`.

Server Repräsentiert einen Discord-Server mit Attributen wie `id`, `name`, `owner_id`, `created_at` und `updated_at`.

ServerPlugin Verknüpft Server und Plugins und speichert Konfigurationsinformationen für jedes Plugin auf einem Server.

Die Beziehungen zwischen den Entitäten sind durch Assoziationen, Aggregationen und Kompositionen gekennzeichnet. Zum Beispiel hat die Klasse **Server** eine Aggregation von **Plugin**, was bedeutet, dass ein Server mehrere Plugins haben kann, aber ein Plugin ohne einen Server existieren kann.

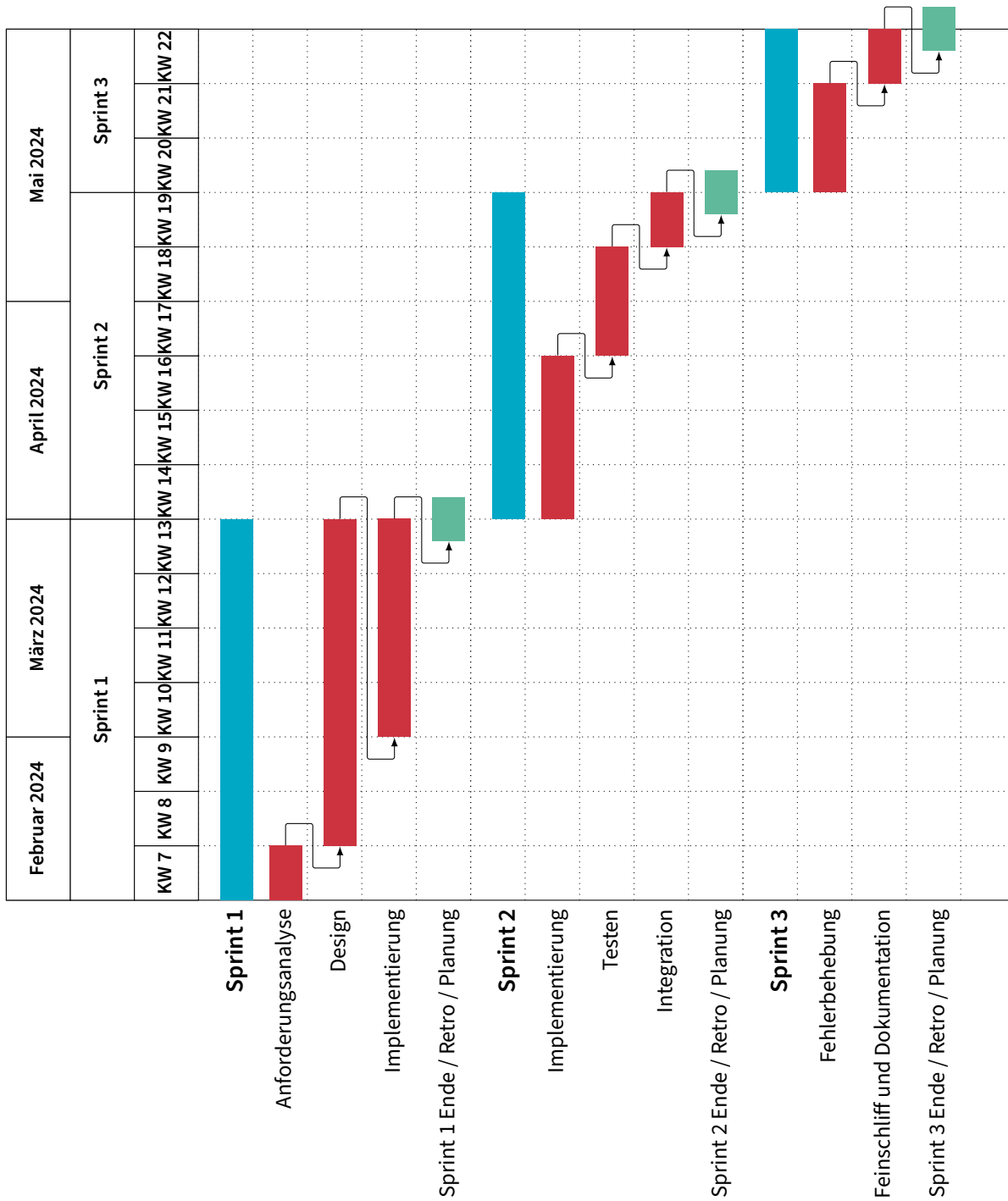


Abbildung 1: Gantt-Diagramm der Projektphasen

4.2.2 Datenhaltung

Für die Datenhaltung wurde eine relationale PostgreSQL-Datenbank erstellt. Die Datenbank enthält Tabellen für Benutzer, Server, Plugins und andere Entitäten. Die Wahl fiel auf PostgreSQL aufgrund seiner Robustheit, Zuverlässigkeit, Erweiterbarkeit und Leistungsfähigkeit. PostgreSQL bietet umfassende ACID-Compliance, erweiterte SQL-Funktionen und eine hohe Sicherheit, was es zur idealen Wahl für die Anforderungen dieses Projekts macht.

Die Datenbank besteht aus vier Haupttabellen: `userinfo`, `configs`, `plugins` und `servers`. Jede dieser Tabellen speichert spezifische Informationen, die für den Betrieb und die Verwaltung der Anwendung notwendig sind. Nachfolgend werden die Tabellen und ihre Spalten detailliert beschrieben, gefolgt von einer Darstellung der Beziehungen zwischen den Tabellen.

4.2.3 Datenbankstruktur und Datenbankmodell

Die Datenbank besteht aus vier Haupttabellen: `users`, `plugins`, `servers` und `server_plugins`. Jede dieser Tabellen speichert spezifische Informationen, die für den Betrieb und die Verwaltung der Anwendung notwendig sind. Nachfolgend werden die Tabellen und ihre Spalten detailliert beschrieben, gefolgt von einer Darstellung der Beziehungen zwischen den Tabellen.

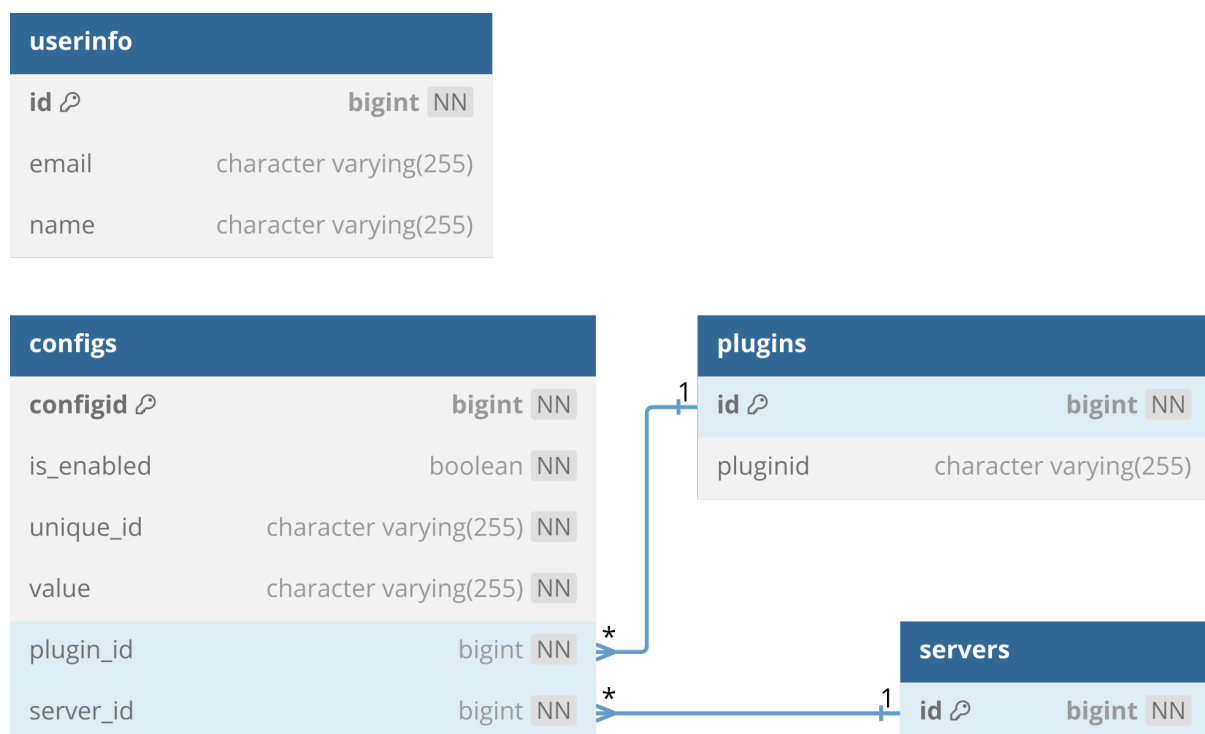


Abbildung 2: Datenbankmodell

Tabelle `userinfo` Die Tabelle `userinfo` speichert Informationen über die Benutzer der Anwendung. Jede Zeile repräsentiert einen einzelnen Benutzer.

- **id** (BIGINT, PRIMARY KEY, NOT NULL): Eindeutige Identifikationsnummer des Benutzers. Dies ist der Primärschlüssel der Tabelle.
- **email** (VARCHAR(255), NOT NULL): Die E-Mail-Adresse des Benutzers, die für Benachrichtigungen und Authentifizierungen verwendet wird.
- **name** (VARCHAR(255), NOT NULL): Der Name des Benutzers, der zur Identifikation und Personalisierung der Benutzererfahrung verwendet wird.

Tabelle `plugins` Die Tabelle `plugins` enthält Informationen über die verschiedenen Plugins, die in der Anwendung verwendet werden können. Jedes Plugin stellt eine eigenständige Erweiterung der Funktionalität dar.

- **id** (BIGINT, PRIMARY KEY, NOT NULL): Eindeutige Identifikationsnummer des Plugins. Dies ist der Primärschlüssel der Tabelle.
- **pluginid** (VARCHAR(255), NOT NULL): Die Plugin-ID, die das Plugin eindeutig identifiziert. Diese ID wird verwendet, um das Plugin innerhalb der Anwendung zu referenzieren.

Tabelle `servers` Die Tabelle `servers` speichert Informationen über die Discord-Server, auf denen der Bot installiert ist. Diese Informationen sind wichtig, um die verschiedenen Instanzen des Bots auf den verschiedenen Servern zu verwalten.

- **id** (BIGINT, PRIMARY KEY, NOT NULL): Eindeutige Identifikationsnummer des Servers. Dies ist der Primärschlüssel der Tabelle.
- **pluginid** (VARCHAR(255), NOT NULL): Die Plugin-ID, die dem Server zugeordnet ist. Diese ID hilft, die Plugins zu verwalten, die auf den einzelnen Servern installiert sind.

Tabelle `configs` Die Tabelle `configs` speichert spezifische Konfigurationen für jedes Plugin auf einem Server. Diese Konfigurationen ermöglichen es, die Funktionsweise der Plugins auf den jeweiligen Servern anzupassen.

- **configid** (BIGINT, PRIMARY KEY, NOT NULL): Eindeutige Identifikationsnummer der Konfiguration. Dies ist der Primärschlüssel der Tabelle.
- **is_enabled** (BOOLEAN, NOT NULL): Ein Flag, das angibt, ob die Konfiguration aktiviert (TRUE) oder deaktiviert (FALSE) ist. Dies ermöglicht das einfache Ein- und Ausschalten von Plugin-Funktionen.
- **unique_id** (VARCHAR(255), NOT NULL): Eine eindeutige ID, die zur Identifikation der Konfiguration dient. Diese ID wird verwendet, um spezifische Konfigurationseinstellungen zu referenzieren.

- **value** (VARCHAR(255), NOT NULL): Der Wert der Konfiguration. Dies könnte z.B. eine bestimmte Einstellung oder ein Parameter sein, der die Funktionsweise des Plugins beeinflusst.
- **plugin_id** (BIGINT, NOT NULL): Die ID des Plugins, zu dem diese Konfiguration gehört. Diese Spalte verweist auf `plugins.id` und ermöglicht es, die Konfigurationen spezifischen Plugins zuzuordnen.
- **server_id** (BIGINT, NOT NULL): Die ID des Servers, auf dem das Plugin installiert ist. Diese Spalte verweist auf `servers.id` und hilft, die Konfigurationen den entsprechenden Servern zuzuordnen.

4.2.4 Beziehungen zwischen den Tabellen

Die Tabellen sind durch verschiedene Beziehungen miteinander verknüpft, um die Datenintegrität zu gewährleisten und die Struktur der Anwendung abzubilden.

- **Server und Plugins:** Ein Server (`servers`) kann mehrere Plugins (`plugins`) verwenden. Diese Beziehung wird durch die Verknüpfungstabelle `configs` realisiert. Die Tabelle `configs` enthält die Fremdschlüssel `configs.server_id` und `configs.plugin_id`, die auf `servers.id` bzw. `plugins.id` verweisen. Diese Tabelle ermöglicht es, die Konfiguration jedes Plugins für jeden Server individuell zu speichern.

4.2.5 Detaillierte Beschreibung der Beziehungen

Server und Plugins (1:n Beziehung)

- **Primärschlüssel:** `servers.id`, `plugins.id`
- **Fremdschlüssel:** `configs.server_id`, `configs.plugin_id`
- **Beschreibung:** Ein Server kann mehrere Plugins verwenden, und ein Plugin kann auf mehreren Servern verwendet werden. Diese 1:n-Beziehung wird durch die `configs`-Tabelle vermittelt, die die IDs der Server und Plugins sowie die Konfigurationseinstellungen für die Plugins auf den jeweiligen Servern speichert.

Die Datenbankstruktur und das Modell bieten die Grundlage für die Verwaltung von Benutzern, Discord-Servern und Plugins. Die klar definierten Beziehungen zwischen den Tabellen gewährleisten eine hohe Datenintegrität und ermöglichen eine flexible und erweiterbare Architektur für die Anwendung. Die Tabelle `userinfo` speichert Benutzerinformationen, die Tabelle `servers` speichert Serverinformationen, die Tabelle `plugins` speichert Plugin-Informationen, und die Tabelle `configs` speichert die Zuordnungen und Konfigurationen der Plugins zu den jeweiligen Servern.

4.2.6 Design Patterns

Während der Entwicklung wurden mehrere *Design Patterns* ► eingesetzt, um die Codequalität und Wartbarkeit zu verbessern. Die folgenden Design Patterns wurden verwendet:

Singleton Ein Singleton-Entwurfsmuster wurde verwendet, um sicherzustellen, dass nur eine Instanz einer Klasse erstellt wird und auf diese Instanz global zugegriffen werden kann. Dies wurde für Klassen wie die Konfigurationsverwaltung verwendet, um sicherzustellen, dass nur eine Instanz der Konfiguration existiert und von verschiedenen Teilen der Anwendung verwendet werden kann. Bei einem Singleton-Designmuster wird eine statische Instanz der Klasse erstellt und auf diese Instanz über eine statische Methode zugegriffen. In Java kann dies durch die Verwendung des Schlüsselworts `static` und einer privaten Konstruktor-Methode erreicht werden:

```
1 public class Configuration {
2     private static Configuration instance = null;
3     private Configuration() {
4         // private constructor to prevent instantiation
5     }
6     public static Configuration getInstance() {
7         if (instance == null) {
8             instance = new Configuration();
9         }
10        return instance;
11    }
12 }
```

Quelltext 1: Singleton-Entwurfsmuster in Java

Factory Ein Factory-Entwurfsmuster wurde verwendet, um die Erstellung von Objekten zu kapseln und die Flexibilität und Wiederverwendbarkeit des Codes zu erhöhen. Dies wurde für die Erstellung von Server- und Plugin-Objekten verwendet, um die Komplexität der Erstellung und Konfiguration dieser Objekte zu reduzieren. Bei einem Factory-Entwurfsmuster wird eine separate Fabrikmethode verwendet, um Objekte zu erstellen und zu konfigurieren. Dies ermöglicht es, die Erstellung von Objekten zu kapseln und die Implementierungsdetails zu verbergen. In Java kann dies durch die Verwendung einer statischen Fabrikmethode erreicht werden:

```
1 public class ServerFactory {
2     public static Server createServer(String name, User owner) {
3         Server server = new Server();
4         server.setName(name);
5         server.setOwner(owner);
6         return server;
7     }
8 }
```

Quelltext 2: Factory-Entwurfsmuster in Java

Observer Ein Observer-Entwurfsmuster wurde verwendet, um eine lose Kopplung zwischen Objekten zu ermöglichen und Event-Handling-Mechanismen zu implementieren. Dies wurde für die Implementierung von Benachrichtigungen und Ereignissen in der Anwendung verwendet. Bei einem Observer-Entwurfsmuster werden Beobachter- und Subjektoobjekte verwendet, um Änderungen in einem Objekt zu überwachen und darauf zu reagieren. Dies ermöglicht es, Änderungen in einem Objekt zu verfolgen und andere Teile der Anwendung darüber zu informieren. In Java kann dies durch die Verwendung von Observer- und Observable-Klassen erreicht werden:

```
1 public class Subject extends Observable {
2     public void notifyObservers(Object arg) {
3         setChanged();
4         super.notifyObservers(arg);
5     }
6 }
7 public class Observer implements java.util.Observer {
8     public void update(Observable o, Object arg) {
9         // handle update
10    }
11 }
```

Quelltext 3: Observer-Entwurfsmuster in Java

Diese Design Patterns wurden ausgewählt, weil sie bewährte Lösungen für häufig auftretende Probleme bieten und die Codebasis strukturierter und erweiterbarer machen.

4.3 Qualitätssicherung

4.3.1 Code Reviews

Während der Entwicklung wurden regelmäßige Code-Reviews durchgeführt, um sicherzustellen, dass der Code qualitativ hochwertig und fehlerfrei ist. Die Code-Reviews wurden von den Teammitgliedern durchgeführt, die nicht an der ursprünglichen Implementierung beteiligt waren. Dies ermöglichte es, potenzielle Fehler und Probleme frühzeitig zu identifizieren und zu beheben. Die Code-Reviews konzentrierten sich auf Aspekte wie Code-Qualität, Lesbarkeit, Wartbarkeit, Performance und Sicherheit. Die Ergebnisse der Code-Reviews wurden dokumentiert und in die weitere Entwicklung integriert.

4.3.2 Teststrategien

Für die Qualitätssicherung wurden verschiedene Teststrategien eingesetzt, um sicherzustellen, dass die Anwendung stabil und fehlerfrei funktioniert.

- **Unit Tests:** Diese Tests wurden auf der kleinsten Code-Ebene durchgeführt, um sicherzustellen, dass einzelne Methoden und Funktionen korrekt arbeiten. Dabei kamen JUnit und Mockito zum Einsatz.

- **Integrationstests:** Diese Tests überprüfen das Zusammenspiel mehrerer Komponenten und Dienste der Anwendung. Hierbei wurde insbesondere die Integration der Datenbank und der REST-API getestet.
- **Systemtests:** Diese Tests wurden durchgeführt, um das gesamte System unter realistischen Bedingungen zu überprüfen. Dazu gehörten Tests der Benutzeroberfläche und der API-Endpunkte.

4.3.3 Testfälle

Konkrete Testfälle:

- **Benutzerregistrierung:** Testfall zur Überprüfung der erfolgreichen Registrierung eines neuen Benutzers. Erwartetes Ergebnis: Der Benutzer wird in der Datenbank gespeichert und kann sich anmelden.
- **Plugin-Aktivierung:** Testfall zur Überprüfung der erfolgreichen Aktivierung eines Plugins auf einem Server. Erwartetes Ergebnis: Das Plugin wird aktiviert und die entsprechende Konfiguration wird gespeichert.
- **Server-Verwaltung:** Testfall zur Überprüfung der korrekten Erstellung und Verwaltung von Servern. Erwartetes Ergebnis: Server werden korrekt in der Datenbank gespeichert und können vom Benutzer verwaltet werden.

4.3.4 Zusammenfassung der Testergebnisse

Die durchgeführten Tests haben gezeigt, dass die Anwendung stabil und funktional ist. Alle kritischen Funktionen wurden erfolgreich getestet und funktionieren wie erwartet. Es wurden keine schwerwiegenden Fehler gefunden, und kleinere Bugs wurden behoben. Die Anwendung erfüllt die gestellten Anforderungen und ist bereit für den Einsatz.

5 Wirtschaftliche Betrachtung

5.1 Marktuntersuchung

5.1.1 Zielgruppe

Die Hauptzielgruppen für BothQ lassen sich in mehrere Kategorien einteilen. Zunächst sind da die Gamer und Gaming-Communities. Discord ist bei Gamern sehr beliebt, und viele Gaming-Communities verwenden Bots, um Server zu verwalten, Spielinformationen bereitzustellen, Events und Turniere zu organisieren und vieles mehr.

Des Weiteren zählen technikbegeisterte Personen und Entwickler zur Zielgruppe. Diese Personen interessieren sich für die Programmierung und Entwicklung von Bots, verfügen jedoch möglicherweise nicht über die technischen Kenntnisse, um einen Discord-Bot von Grund auf zu erstellen und zu konfigurieren.

Auch Unternehmen und Organisationen, die Discord für die Teamkommunikation nutzen, sind eine wichtige Zielgruppe. Diese Unternehmen könnten Bots zur Automatisierung von Aufgaben, für den Kundenservice und andere organisatorische Zwecke einsetzen. Schließlich gehören auch Content Creator und Streamer zu den Zielgruppen. Streamer auf Plattformen wie Twitch und YouTube nutzen häufig Discord-Server, um mit ihrer Community zu interagieren, und könnten Bots zur Moderation und zur Steigerung des Engagements einsetzen.

Die Erwartungen der Zielgruppe an BotHQ umfassen eine einfache Installation und Einrichtung, Benutzerfreundlichkeit, vielfältige Plugins und Erweiterungen, Zuverlässigkeit und Stabilität, Sicherheit, guten Kundensupport und Dokumentation, Flexibilität und Anpassbarkeit sowie die Integration mit anderen Diensten.

5.1.2 Marktvolumen und Marktpotential

Um das Marktvolumen für BotHQ zu bestimmen, betrachten wir zunächst die Nutzerbasis von Discord, die weltweit über 250 Millionen registrierte Nutzer und über 100 Millionen aktive Nutzer pro Tag umfasst. Besonders beliebt ist die Plattform bei Gamern, Technik-Enthusiasten und Community-Organisatoren. Es wird geschätzt, dass etwa 30% der Discord-Server mindestens einen Bot verwenden. Geht man davon aus, dass 30% der aktiven Discord-Nutzer Bots verwenden, ergibt sich ein potenzielles Marktvolumen von ca. 30 Millionen aktiven Nutzern weltweit, die an BotHQ interessiert sein könnten.

Der Markt für Discord-Bots weist zudem ein erhebliches Wachstumspotenzial auf. Mit der zunehmenden Verbreitung von Discord in verschiedenen Communities und Branchen steigt auch die Nachfrage nach Bots, die bestimmte Aufgaben automatisieren und Server effizienter verwalten können. In vielen Bereichen wächst der Bedarf an Automatisierungslösungen, und Discord-Bots können eine Vielzahl von Aufgaben übernehmen, von der Moderation über die Bereitstellung von Informationen bis hin zur Interaktion mit der Community.

Zusätzlich eröffnen neue Entwicklungen in der API-Integration sowie die zunehmende Marktexpansion von Discord neue Möglichkeiten für BotHQ. Während Discord ursprünglich vor allem von Gamern genutzt wurde, hat sich die Plattform inzwischen auf andere Bereiche wie Bildung, Unternehmenskommunikation und soziale Netzwerke ausgeweitet. Diese neuen Marktsegmente bieten weitere Wachstumsmöglichkeiten für BotHQ. Innovative Geschäftsmöglichkeiten durch Premium-Dienste, personalisierte Lösungen und Support-Dienste können zusätzliche Einnahmequellen erschließen und die Nutzerbindung erhöhen.

5.1.3 Konkurrenzanalyse

Im Bereich der Discord-Bot-Management-Plattformen gibt es mehrere Konkurrenzprodukte, die ähnliche Dienste anbieten. Einige der wichtigsten Konkurrenten sind Dyno Bot, Mee6, Carl-bot, GAwesome Bot, Nightbot und BotGhost. Diese Bots bieten umfangreiche Verwaltungsfunktionen und zeichnen sich durch verschiedene Stärken und Schwächen aus.

BotHQ bietet jedoch mehrere einzigartige Funktionen und Vorteile, die es von der Konkurrenz abheben. Im Gegensatz zu vielen Konkurrenzprodukten, die eine manuelle Konfiguration erfordern, bietet BotHQ eine vollständig automatisierte Installation und Konfiguration von Bots und Plugins, was die Benutzerfreundlichkeit deutlich erhöht. Darüber hinaus ermöglicht BotHQ eine einfache Aktivierung und Konfiguration von Plugins, ohne den Bot neu starten oder Dateien manuell bearbeiten zu müssen. Eine intuitive und leicht zu navigierende Benutzeroberfläche stellt sicher, dass auch Nutzer ohne technische Vorkenntnisse den vollen Funktionsumfang des Bots nutzen können.

BotHQ legt großen Wert auf Sicherheit und Datenschutz und bietet regelmäßige Updates und Sicherheitspatches an, um sicherzustellen, dass die Bots und die Daten der Nutzer geschützt sind. Der umfangreiche Kundensupport und die aktive Community von BotHQ bieten den Nutzern bei Problemen und Fragen Unterstützung. Zusätzlich unterstützt BotHQ eine Vielzahl von Plugins und ermöglicht es Entwicklern, eigene Plugins zu erstellen und zu integrieren, um die Funktionalität des Bots kontinuierlich zu erweitern.

5.2 Marketingmix

Die Produktpolitik von BotHQ umfasst verschiedene Produktvarianten, um den unterschiedlichen Bedürfnissen der Zielgruppen gerecht zu werden. Dazu gehören eine kostenlose Basisversion und kostenpflichtige Premiumversionen mit erweiterten Funktionen.

Die Preispolitik von BotHQ verfolgt eine Freemium-Preisstrategie. Diese Strategie ermöglicht es, eine breite Nutzerbasis zu erreichen und gleichzeitig durch Premium-Dienste Einnahmen zu generieren.

In der Kommunikationspolitik setzt BotHQ auf verschiedene Kanäle, darunter Social Media, Influencer-Marketing, Partnerschaften mit Discord-Communities und gezielte Online-Werbung. Ein starkes Augenmerk wird zudem auf Content-Marketing gelegt, um die Sichtbarkeit und das Engagement zu erhöhen.

Die Distributionspolitik von BotHQ konzentriert sich hauptsächlich auf den Online-Vertrieb über die eigene Website sowie über Plattformen wie GitHub und Discord selbst. Diese Kanäle ermöglichen eine breite Verfügbarkeit und einfache Zugänglichkeit für die Nutzer.

5.3 Kostenplanung

Im Folgenden werden die potenziellen Kosten, die im Laufe des Projekts entstehen können, im Detail erläutert. Dies betrifft die Berechnung der Personalkosten, die jeweiligen Stundensätze, die Kosten für Sachmittel und die Gesamtkosten für die Durchführung des Projekts.

5.3.1 Personalkosten

Das Projekt startete am 14. Februar 2024 und endet am 30. Mai 2024. Die Arbeitszeit ist für jeden Mittwoch von 8:10 bis 13:20 Uhr mit zwei Pausen je 20 Minuten geplant. Dies ergibt eine Arbeitszeit pro Mittwoch von 4 Stunden und 30 Minuten (270 Minuten) abzüglich 40 Minuten Pause, also 3 Stunden und 50 Minuten (230 Minuten oder 3.83 Stunden). Der Zeitraum umfasst insgesamt 16 Wochen.

Person	Stundensatz (EUR)
Fabian Reichwald	50
Phillipp Batelka	45
Justus Sieweke	55
Daniel Quellenberg	40
Jan Mahnken	50
Christopher Spencer	60

Tabelle 1: Stundensätze der Projektmitglieder

Die Berechnung der Personalkosten erfolgt wie folgt:

$$\begin{aligned}\text{Personalkosten pro Woche} &= (\text{Fabian Reichwald} + \text{Phillipp Batelka} + \text{Justus Sieweke} + \\ &\quad \text{Daniel Quellenberg} + \text{Jan Mahnken} + \text{Christopher Spencer}) \\ &\quad \times \text{Arbeitszeit pro Woche} \\ &= (50 + 45 + 55 + 40 + 50 + 60) \times 3.83 \\ &= 300 \times 3.83 \\ &= 1149 \text{ EUR/Woche}\end{aligned}$$

Gesamtkosten pro Woche	1149 EUR
Anzahl der Wochen	16
Gesamtkosten	18384 EUR

Tabelle 2: Gesamtkosten pro Woche

Die gesamten Personalkosten für 6 Personen über den Zeitraum von 3 Sprints (16 Wochen) betragen somit **18.384 EUR**.

5.3.2 Sachmittelkosten

Für die Durchführung des Projekts fallen auch Sachmittelkosten an. Diese umfassen Hardware-, Software- und Zusatzkosten.

Hardware	Anzahl	Kosten (EUR)
Laptop (Windows)	2	2000
Laptop (Linux)	2	1800
Laptop (Mac)	2	3000
Gesamtkosten Hardware		6800

Tabelle 3: Hardware-Kosten

Um den Entwicklern flexible und leistungsfähige Arbeitsmittel zur Verfügung zu stellen, werden 6 Laptops benötigt. Verschiedene Betriebssysteme (Windows, Linux, Mac) sind erforderlich, um sicherzustellen, dass die entwickelten Anwendungen auf verschiedenen Plattformen getestet und optimiert werden können.

Software	Anzahl	Kosten (EUR)
JetBrains Lizenz (Full Package)	6	1500
Webhosting	1	200
Domain	1	10
SSL-Zertifikat	1	50
Gesamtkosten Software		1760

Tabelle 4: Software-Kosten

JetBrains-Lizenzen werden benötigt, um dem Team hochwertige Entwicklungsumgebungen zur Verfügung zu stellen. Webhosting, Domain und SSL-Zertifikat sind notwendig, um die entwickelte Anwendung sicher und zugänglich im Internet bereitzustellen.

Cloud & Backup	Anzahl	Kosten (EUR)
Backup-Lösung	1	300
Cloud-Speicher	1	200
Gesamtkosten Zusatzkosten		500

Tabelle 5: Zusatzkosten

Eine Backup-Lösung ist wichtig, um Datenverlust zu vermeiden und die Sicherheit der Projektdaten zu gewährleisten. Cloud-Speicherung ist erforderlich, um eine zentrale Speicherung und einen einfachen Zugriff auf Projektdateien für alle Teammitglieder zu ermöglichen.

Kostenkategorie	Kosten (EUR)
Hardware	6800
Software	1760
Zusatzkosten	500
Gesamtkosten Sachmittel	9060

Tabelle 6: Gesamtkosten Sachmittel

5.3.3 Gesamtkosten des Projektes

Die gesamten Projektkosten setzen sich aus den Personalkosten und den Sachmittelkosten zusammen.

Kostenkategorie	Kosten (EUR)
Personalkosten	18384
Sachmittelkosten	9060
Gesamtkosten des Projektes	27444

Tabelle 7: Gesamtkosten des Projektes

Ergebnis

Die gesamten Projektkosten betragen **27.444 EUR**.

5.1 Wirtschaftlichkeitsberechnung

Bei der Gewinnschwellenberechnung und der Amortisationsrechnung handelt es sich um wichtige Instrumente zur Bewertung der Wirtschaftlichkeit eines Projekts. Diese Berechnungen geben Aufschluss darüber, ab welchem Punkt das Projekt profitabel wird und wie lange es dauert, bis sich die Investitionen amortisiert haben.

5.1.1 Gewinnschwellenberechnung

Die Gewinnschwellenberechnung bestimmt, ab welcher Menge verkaufter Einheiten das Projekt Gewinn erzielt. Angenommen, das Produktabo wird zu einem Preis von 10 EUR pro Monat angeboten.

- **Gesamtkosten des Projektes:** 27.444 EUR
- **Preis des Produktabos:** 10 EUR pro Monat
- **Gewinnschwelle:**

$$\frac{27.444 \text{ EUR}}{10 \text{ EUR/Monat}} = 2.744,4 \text{ Monate}$$

Das bedeutet, es müssen 2.745 Abos für einen Monat verkauft werden, um die Kosten zu decken.

5.1.2 Amortisationsrechnung

Die Amortisationsrechnung berechnet die Zeit, die benötigt wird, um die Investition durch die Einnahmen zu decken. Angenommen, es werden monatlich 500 Abos verkauft.

- **Monatliche Einnahmen:**

$$500 \text{ Abos} \times 10 \text{ EUR/Monat} = 5.000 \text{ EUR/Monat}$$

- **Amortisationszeit:**

$$\frac{27.444 \text{ EUR}}{5.000 \text{ EUR/Monat}} = 5,49 \text{ Monate}$$

Das bedeutet, dass es ungefähr 5,5 Monate dauert, bis sich die Investition in das Projekt amortisiert hat.

6 Projektabschluss

6.1 Erreichung des Projektziels

Ziel des Projekts war die Entwicklung eines modularen Frameworks auf Basis eines bestehenden Discord-Frameworks für Java, das als Cloud-Service angeboten wird. Benutzer sollten in der Lage sein, selbst geschriebene Plugins über ein Webinterface zu laden und zu konfigurieren, ohne das Framework selbst hosten zu müssen. Im Folgenden ein detaillierter Überblick über die erreichten und nicht erreichten Ziele:

Erreichte Ziele

Modulares Framework: Ein eigenständiges modulares Framework wurde erfolgreich entwickelt. Nutzer können selbst geschriebene Plugins laden, aktivieren, deaktivieren und konfigurieren.

Cloud-Service Bereitstellung: Das Framework wird erfolgreich als Cloud-Service angeboten, sodass die Nutzer keinen eigenen Hosting-Service benötigen.

Discord-Bot Integration: Der entwickelte Discord-Bot kann von Nutzern auf eigene Server geladen werden. Über die Weboberfläche können Einstellungen vorgenommen und Plugins verwaltet werden.

Weboberfläche und REST-API: Die Weboberfläche ermöglicht es Nutzern, ihre Instanz des Frameworks zu verwalten. Eine REST-API wurde entwickelt und integriert, um die Weboberfläche mit dem REST-Server zu verbinden, Anfragen zu verarbeiten und Daten in der Datenbank zu speichern.

Plugins: Die folgenden Plugins wurden erfolgreich implementiert:

- Begrüßung neuer Mitglieder in einem definierten Channel
- Mitteilung beim Verlassen eines Mitglieds
- Kicken/Bannen von Mitgliedern, mit oder ohne Text-Grund
- Dynamische Rollenvergabe anhand von Emoji-Reaktionen
- Broadcast von Nachrichten mit Discord-Embed-Support

Nicht erreichte Ziele

Erweiterung der Plugins: Keine zusätzlichen Plugins über die geplanten hinaus wurden implementiert, obwohl dies optional war.

Erweiterte REST-API Schnittstellen: Weitere Default-REST-API-Schnittstellen, die über die Mitglieds- und Rollenaufstellung hinausgehen, wurden nicht hinzugefügt.

Default REST-API Funktionen: Standard-REST-API zur Auflistung aller Mitglieder und deren Rollen ohne zusätzliche Plugins wurde nicht implementiert.

Zusammenfassung

Das Projekt wurde weitgehend erfolgreich abgeschlossen. Das modulare Framework wurde wie geplant entwickelt und als Cloud-Service zur Verfügung gestellt. Nutzer können Plugins über eine benutzerfreundliche Weboberfläche laden und konfigurieren. Alle geplanten Standard-Plugins und die Kernfunktionalität der REST-API wurden implementiert. Einige zusätzliche Funktionen und Erweiterungen wurden nicht umgesetzt, was jedoch die Kernfunktionalität und die Erreichung der Hauptziele des Projekts nicht beeinträchtigt. Insgesamt wurde das Hauptziel erreicht, ein robustes und benutzerfreundliches modulares Framework für Discord-Bots zu schaffen, das den Nutzern flexible Anpassungsmöglichkeiten bietet, ohne dass sie selbst hosten müssen.

6.2 Änderungen zur anfänglichen Planung

Im Laufe des Projekts wurden einige Änderungen an der ursprünglichen Planung vorgenommen. Diese Änderungen ergaben sich aus praktischen Herausforderungen und neuen Erkenntnissen während der Umsetzungsphase. Im Folgenden werden die wichtigsten Änderungen und die Gründe für diese Änderungen aufgeführt:

Anpassungen der Datenbank

Während der Entwicklung mussten wir die Datenbankstruktur mehrfach anpassen. Diese Änderungen waren notwendig, um neue Anforderungen und Optimierungen zu berücksichtigen, die während der Implementierung auftraten.

Designanpassungen

Das ursprüngliche Design ließ sich nicht immer 1:1 übernehmen. Anpassungen wurden vorgenommen, um die Benutzerfreundlichkeit zu verbessern und technische Einschränkungen zu überwinden.

Verzicht auf Mitgliederliste

Ursprünglich war geplant, eine Liste aller Mitglieder und deren Rollen über die REST-API bereitzustellen. Diese Funktion wurde entfernt, da sie gegen die Nutzungsbedingungen (ToS) von Discord verstoßen könnte, wenn sie nicht unbedingt erforderlich ist. Discord legt großen Wert auf den Datenschutz der Mitglieder und hat strenge Einschränkungen für den Zugriff auf solche Daten.

Zusammenfassung

Die im Laufe des Projekts vorgenommenen Änderungen waren notwendig, um sowohl technischen als auch rechtlichen Herausforderungen zu begegnen. Die Anpassungen der Datenbank und des Designs trugen dazu bei, die Funktionalität und Benutzerfreundlichkeit des Frameworks zu verbessern. Der Verzicht auf eine Mitgliederliste war eine bewusste Entscheidung, um den Nutzungsbedingungen von Discord zu entsprechen und mögliche rechtliche Probleme zu vermeiden. Trotz dieser Änderungen wurden die Hauptziele des Projekts erreicht und das entwickelte Framework bietet eine robuste und flexible Lösung für die Verwaltung von Discord-Bots als Cloud-Service.

6.3 Fazit

Teamarbeit

Die Teamarbeit hat insgesamt gut funktioniert. Alle Teammitglieder haben sich engagiert und die meisten Aufgaben wurden in gemeinsamer Anstrengung erfolgreich bewältigt. Vereinzelt gab es jedoch Probleme in der Zusammenarbeit, die zu Verzögerungen und zusätzlichem Koordinationsaufwand führten. Diese Herausforderungen konnten durch offene Kommunikation und Anpassungsfähigkeit gemeistert werden.

Persönlicher Gewinn

Das Projekt hat viele wertvolle Erfahrungen und Erkenntnisse auf persönlicher Ebene gebracht. Es wurde deutlich, wie wichtig eine gründliche und flexible Planung ist, um auf unerwartete Herausforderungen effektiv reagieren zu können. Die Zusammenarbeit im Team hat gezeigt, wie wichtig klare Kommunikation und gegenseitige Unterstützung für ein erfolgreiches Projekt sind. Durch regelmäßige Reflexionen konnten wir uns kontinuierlich verbessern und aus Fehlern lernen.

Auch die technischen Fähigkeiten wurden deutlich erweitert. Wir haben vertiefte Kenntnisse in der Entwicklung von Webanwendungen mit Angular sowie in der Backend-Entwicklung mit Java gewonnen. Zudem erhielten wir Einblicke in die Erstellung und Verwendung von RESTful APIs für die Kommunikation zwischen verschiedenen Systemen sowie in die Datenbankverwaltung und -optimierung. Insgesamt hat das Projekt sowohl die fachlichen als auch die sozialen Kompetenzen gestärkt und wertvolle praktische Erfahrungen vermittelt, die für zukünftige Projekte von großem Nutzen sein werden.

A Anhang

A.1 Glossar

C

Client-Server-Architektur Die Client-Server-Architektur ist ein Netzwerkmodell, bei dem Aufgaben und Dienste zwischen zwei Hauptkomponenten aufgeteilt werden: dem Client und dem Server. Der Client stellt Anfragen, um auf Ressourcen oder Dienste zuzugreifen, die vom Server bereitgestellt werden. Der Server verarbeitet diese Anfragen und liefert die entsprechenden Antworten oder Ressourcen zurück. Dieses Modell ist weit verbreitet in der Entwicklung von Webanwendungen, bei denen der Client in der Regel ein Webbrowser ist, der Anfragen an einen Webserver stellt. 1

Cloud-Service Ein Cloud-Service ist ein Dienst, der über das Internet bereitgestellt wird und auf Cloud-Computing-Technologien basiert. Cloud-Services ermöglichen es Benutzern, auf Anwendungen, Daten und Ressourcen zuzugreifen, ohne dass sie physische Hardware oder Software installieren oder verwalten müssen 2

D

Design Patterns Design Patterns sind bewährte Lösungsmuster für häufig auftretende Probleme in der Softwareentwicklung. Sie bieten eine strukturierte und wiederverwendbare Möglichkeit, Probleme zu lösen und Entwurfsentscheidungen zu treffen. Design Patterns helfen Entwicklern, bewährte Praktiken zu nutzen, Code zu organisieren, die Wartbarkeit zu verbessern und die Lesbarkeit zu erhöhen 13

Discord API Die Discord API ist eine Programmierschnittstelle, die Entwicklern Zugriff auf die Funktionen und Daten von Discord bietet. Mit der API können Entwickler Bots und Anwendungen erstellen, die mit Discord-Servern und -Benutzern interagieren. Die API ermöglicht das Senden und Empfangen von Nachrichten, das Verwalten von Servern und Kanälen, das Abrufen von Benutzerinformationen und vieles mehr. Die Discord API nutzt REST für die meisten ihrer Funktionen und bietet Echtzeit-Updates über WebSockets. 1

E

Endpunkt Ein Endpunkt ist eine spezifische URL innerhalb einer API, die eine bestimmte Funktion oder Ressource repräsentiert. Endpunkte werden verwendet, um verschiedene Operationen wie das Abrufen von Daten, das Senden von Informationen oder das Ausführen von Aktionen zu ermöglichen. Jeder Endpunkt wird durch eine HTTP-Methode (GET, POST, PUT, DELETE usw.) und eine URL definiert. Zum Beispiel könnte ein Endpunkt für das Abrufen von Benutzerinformationen in einer API die URL `/users/{id}` haben und mit der HTTP-Methode GET aufgerufen werden. 1

G

Git Git ist ein verteiltes Versionskontrollsystem, das von Linus Torvalds entwickelt wurde. Es wird häufig in der Softwareentwicklung eingesetzt, um Änderungen an Quellcode und Dateien zu verfolgen, zu verwalten und zu teilen. Git ermöglicht es Entwicklern, Änderungen in einem Repository zu speichern, Branches zu erstellen, Konflikte zu lösen und Änderungen zu veröffentlichen. Es bietet Funktionen wie Versionsverwaltung, Branching, Merging und Rebase. Git ist bekannt für seine Geschwindigkeit, Flexibilität und Skalierbarkeit 8, *Siehe auch* GitHub

GitHub GitHub ist eine Webplattform für die Versionskontrolle und Zusammenarbeit an Softwareprojekten, die auf Git basiert. Es bietet Funktionen wie Issue-Tracking, Pull-Requests, Code-Reviews, Wikis und Projektmanagement-Tools. GitHub wird von Millionen von Entwicklern und Teams weltweit genutzt, um Quellcode zu hosten, zu teilen und zusammenzuarbeiten. Es ist bekannt für seine Benutzerfreundlichkeit, soziale Funktionen und umfangreichen Integrationsmöglichkeiten 8, *Siehe auch* Git

H

HTTPS-Protokoll Das HTTPS-Protokoll (Hypertext Transfer Protocol Secure) ist eine Erweiterung des HTTP-Protokolls, die eine sichere Kommunikation über ein Computernetzwerk ermöglicht. HTTPS verwendet das Transport Layer Security (TLS) Protokoll, um die Daten zwischen dem Webbrowser und dem Webserver zu verschlüsseln. Dies stellt sicher, dass die übertragenen Daten vertraulich und vor Manipulationen geschützt sind. HTTPS ist besonders wichtig für Webseiten, die sensible Daten wie Passwörter, Kreditkarteninformationen und persönliche Informationen verarbeiten. 1

J

Java Discord API Die Java Discord API (JDA) ist eine umfassende Programmbibliothek, die Entwicklern ermöglicht, Discord-Bots in der Programmiersprache Java zu erstellen und zu verwalten. JDA bietet eine hohe Abstraktionsebene, um die Kommunikation zwischen einem Java-Programm und der Discord-API zu erleichtern. Mit JDA können Entwickler auf verschiedene Funktionen und Ereignisse in Discord zugreifen, wie das Empfangen und Senden von Nachrichten, das Verwalten von Servern, Kanälen und Benutzern sowie das Reagieren auf verschiedene Interaktionen innerhalb von Discord. 2, 4

Jira Jira ist ein weit verbreitetes Tool zur Projekt- und Aufgabenverwaltung, das von Atlassian entwickelt wurde. Es wird häufig in der Softwareentwicklung und im Projektmanagement eingesetzt, um agile Methoden wie Scrum und Kanban zu unterstützen. Mit Jira können Teams Projekte planen, Aufgaben verfolgen, Fortschritte visualisieren und Berichte erstellen. Es bietet Funktionen wie Issue-Tracking, Sprint-Management, Roadmaps und umfangreiche Integrationen mit anderen Entwicklungs- und Kollaborationstools 8, *Siehe auch* Scrum

P

Pull-Requests Ein Pull-Request ist eine Funktion in Git-basierten Versionskontrollsystemen wie GitHub, die es Entwicklern ermöglicht, Änderungen an einem Repository vorzuschlagen und zu diskutieren. Ein Pull-Request enthält die vorgeschlagenen Änderungen, eine Beschreibung des Zwecks

und der Auswirkungen der Änderungen sowie eine Diskussionsplattform für Feedback und Kommentare. Andere Entwickler können den Pull-Request überprüfen, kommentieren und genehmigen, bevor die Änderungen in das Hauptrepository übernommen werden. Pull-Requests sind ein wichtiger Bestandteil des Code-Review-Prozesses und fördern die Zusammenarbeit und Qualitätssicherung im Entwicklerteam 8, *Siehe auch*

R

Repository Ein Repository (auch Repo genannt) ist ein Speicherort für Quellcode, Dateien und Ressourcen, die in einem Versionskontrollsystem wie Git verwaltet werden. Ein Repository enthält die gesamte Historie der Änderungen, die an den Dateien vorgenommen wurden, sowie Metadaten wie Commits, Branches und Tags. Entwickler können ein Repository klonen, um eine lokale Kopie des Codes zu erhalten, Änderungen vorzunehmen und sie dann in das Hauptrepository zurückzuführen. Repositories werden häufig verwendet, um Quellcode für Softwareprojekte zu speichern und zu teilen *Siehe auch* Git, &

REST-API Eine REST-API (Representational State Transfer Application Programming Interface) ist ein Architekturstil für APIs, der auf den Prinzipien von REST basiert. REST-APIs verwenden HTTP-Anfragen, um auf Ressourcen zuzugreifen und diese zu manipulieren. Diese Ressourcen werden durch URLs identifiziert, und die gängigen HTTP-Methoden (GET, POST, PUT, DELETE) werden verwendet, um verschiedene Operationen auszuführen. REST-APIs sind stateless, was bedeutet, dass jede Anfrage alle Informationen enthält, die der Server benötigt, um sie zu verarbeiten, ohne den Zustand zwischen den Anfragen zu speichern. REST-APIs sind bekannt für ihre Einfachheit, Flexibilität und Skalierbarkeit. 2

S

Scrum Scrum ist ein Framework für die agile Softwareentwicklung, das iterative und inkrementelle Prozesse unterstützt. Es besteht aus festen Rollen, Ereignissen und Artefakten, die die Zusammenarbeit im Team fördern und die kontinuierliche Lieferung von funktionierenden Software-Inkrementen ermöglichen. Zu den Rollen gehören der *Product Owner*, der *Scrum Master* und das *Entwicklungsteam*. Die wichtigsten Ereignisse sind der *Sprint*, das *Sprint Planning*, der *Daily Scrum*, das *Sprint Review* und die *Sprint Retrospective*. Zu den Artefakten zählen das *Product Backlog*, das *Sprint Backlog* und das *Increment*. Scrum ermöglicht eine flexible und adaptive Entwicklung, indem es regelmäßig Feedback einholt und das Projekt kontinuierlich an die sich ändernden Anforderungen anpasst. 7

W

WebSocket-Verbindungen WebSocket-Verbindungen sind ein Kommunikationsprotokoll, das eine bidirektionale, Echtzeit-Kommunikation zwischen einem Client (z.B. einem Webbrowser) und einem Server ermöglicht. Im Gegensatz zu traditionellen HTTP-Verbindungen, die unidirektional sind und für jede Datenübertragung eine neue Verbindung aufbauen, bleibt eine WebSocket-Verbindung während der gesamten Kommunikationssitzung offen. Dies ermöglicht eine effizientere und schnellere Datenübertragung, die besonders für Anwendungen wie Chat-Dienste, Online-Spiele und Echtzeit-Datenfeeds nützlich ist. WebSockets sind in der WebSocket-Spezifikation

des IETF (Internet Engineering Task Force) definiert und nutzen das ws:// oder wss:// Schema für unverschlüsselte bzw. verschlüsselte Verbindungen. 1

A.2 Literaturverzeichnis

Austin Keener, Florian Spieß (2024a). *JDA (Java Discord API) Documentation*. Version v5.0.0-beta.24. URL: <https://jda.wiki/introduction/jda/> (besucht am 25. 05. 2024).

– (2024b). *JDA: Java Discord API*. Version v5.0.0-beta.24. URL: <https://github.com/discord-jda/JDA> (besucht am 20. 05. 2024).

Discord (2024a). *Beginner's Guide to Discord*. URL: <https://support.discord.com/hc/en-us/articles/360045138571-Beginner-s-Guide-to-Discord> (besucht am 22. 05. 2024).

– (2024b). *Discord Developer Portal*. URL: <https://discord.com/developers/docs/intro> (besucht am 25. 05. 2024).

– (2024c). *Discord Privacy Policy*. URL: <https://discord.com/privacy#2> (besucht am 25. 05. 2024).

A.3 Quellcode

```
1  public class ReadyListener implements EventListener
2  {
3      public static void main(String[] args)
4          throws InterruptedException
5      {
6          // Note: It is important to register your ReadyListener before
6          building
7          JDA jda = JDABuilder.createDefault("token")
8              .addEventListeners(new ReadyListener())
9              .build();
10
11         // optionally block until JDA is ready
12         jda.awaitReady();
13     }
14
15     @Override
16     public void onEvent(GenericEvent event)
17     {
18         if (event instanceof ReadyEvent)
19             System.out.println("API is ready!");
20     }
21 }
```

Quelltext 4: Beispielcode für einen ReadyListener in JDA