Dokumentation zur betrieblichen Projektarbeit

— Mittelstufenprojekt —

BotHQ

Ein Discord-Bot-Framework

Prüfungsausschuss:	Bernd Heidemann Katrin Deeken
Abgabedatum:	29. 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

Tabellenverzeichnis

Quelltextverzeichnis

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 *Endpunkt* ➤, 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 einfach zu erstellen, zu laden und über eine benutzerfreundliche 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 vertreten:

- **Entwicklung**: Erstellung eigener Softwareprodukte
- **IT-Systembereich**: Lieferung und Verkauf von IT-Komponenten sowie die Planung und Installation von Netwerken
- **Consulting**: Beratung von Anwender 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. Jede Abteilung hat ihren eigenen Abteilungsleiter der eng mit anderen Abteilungsleiter zusammenarbeitet.

2.2 Entstehung der Projektidee

Die Projektidee enstand im Rahmen eines Auftrages, nach dem ein Schulungsprojekt durchzuführen war, dass 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 dem Entwickeln von Discord-Bots gesammelt hat.

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.

Desweiterem 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.

3.2 Soll-Analyse

Soll-Zustands ist es, das ein Cloud Service der HiTec Gmbh für Discord-Bots existieren soll, das auf der *Java Discord API* ▸[6] basiert. Der Service soll Nutzern über eine Weboberfläche ermöglichen, Plugins zu Discord-Servern hinzuzufügen und zu verwalten. Desweiteren soll ein Framework entwickelt werden, welches ein einfaches hinzufügen von neuen Plugins von Seiten der HiTec Gmbh ermöglicht. Dies erlaubt es der HiTec Gmbh eine modulare Dienstleistung für Kunden, die Discord zu Kommunikationszwecken nutzen, anzubieten und diese auf Wunsch der Kunden einfach anzupassen.

3.3 Projektziel detailliert

Der Service soll dem Kunden als Webservice zur verfügung gestellt werden, damit er die Verwaltung seiner Server und Plugins selbstständig durführen kann. Die Platform soll responsive gestaltetwerden mit dem Fokus auf einem benutzerfreundlichen Design und fehlerfreier Funktionalität.

Beim öffnen der Webseite soll der Nutzer dazu aufgefordert werden, sich zu authentifizieren. Der Service zur authentifizierung wird hierbei von der Discord-API bereit gestellt.

6 Austin Keener 2024a,b.

Auf der Website soll der Nutzer eine Liste Seiner Discord-Server und der verfügbaren Plugins angezeigt bekommen. Die Liste der Server wird von der Discord-API bereitgestellt, die Liste der Plugins wird wiederum von der eigenen Backend-API des Services. Nutzern soll es möglich sein, Plugins und die Einstellungen dieser für jeden Server einzeln und in Echtzeit verwalten zu können. Eine Datenbank im Backend des Services Speichert die so vorgenommenen Einstellungen ab. Das Backend soll die vom Nutzer vorgenommenen Einstellungen automatisiert vornehmen.

Desweiteren soll ein Framework zum erstellen von Plugins für den Inhaber des Services existieren. Einem Entwickler soll es ermöglicht werden, Code schreiben zu können, der mit einem Dicord-Server interagieren kann. Zusätzlich soll der Entwickler eine Konfigurationsdatei bereitstellen können. Diese soll Parameter enthalten, die vom Endnutzer manipuliert können sollen. Hierzu wird die Konfigurationsdatei beim Aufruf der Plugin-Einstellungen durch den nutzer an das Frontend übermittel, was dann daraus dynamisch ein Interface für den Nutzer generiert.

3.4 Anforderungen und Technologien

- Auflistung und Beschreibung der Anforderungen:
- Technologische Anforderungen (z.B. REST-API, Datenbanken)
- Zielplattformen und benötigte Technologien (z.B. Cloud-Services)

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* ▸ 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-Meetings[7] 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 7 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, Jahn 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.
- **Dokumentation und Marketing**: Verantwortlich für die Erstellung und Pflege der Projektdokumentation, der Projektpräsentation und des Marketingpitches waren Philipp Batelka, Daniel

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

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.

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.

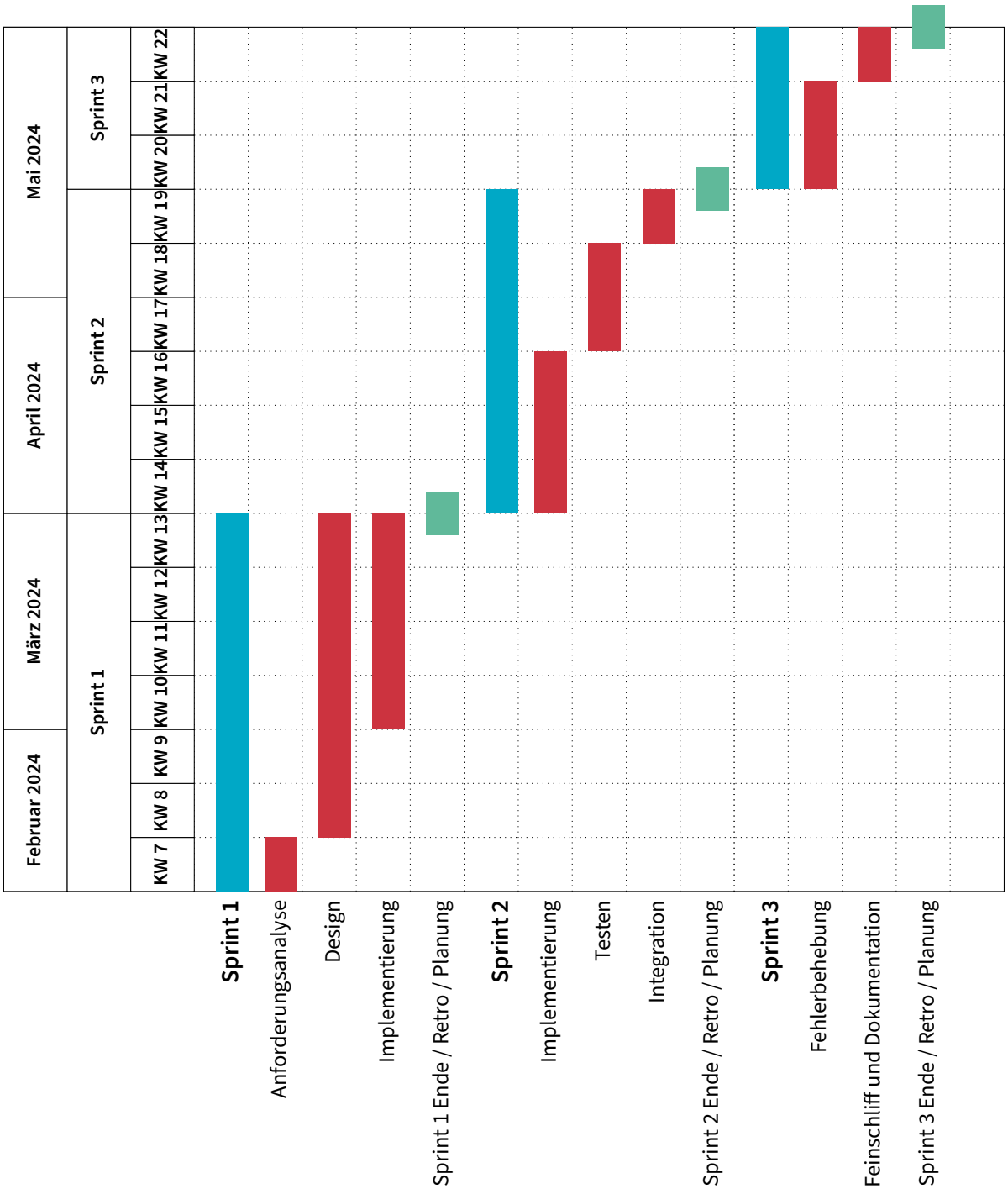

Abbildung 1: Gantt-Diagramm der Projektphasen

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.

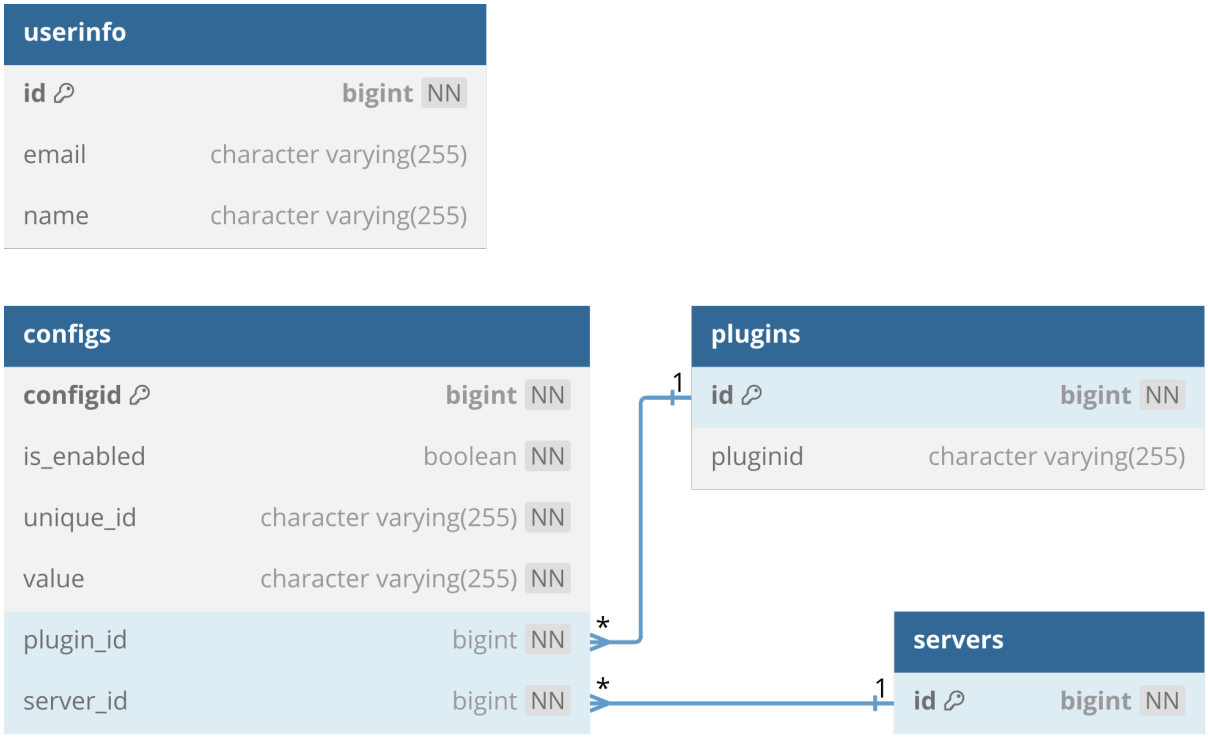

Abbildung 2: Datenbankmodell

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 Subjektobjekte 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 überprüften 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

- Beschreibung der Zielgruppe:
  - Welche Erwartungen hat die Zielgruppe an das Produkt?

5.1.2 Marktvolumen und Marktpotential

- Analyse des Marktvolumens und -potentials:
  - Wie groß ist der Markt für das Produkt?
  - Welche Wachstumsmöglichkeiten gibt es?

5.1.3 Konkurrenzanalyse

- Analyse der Wettbewerbssituation:
  - Welche Konkurrenzprodukte gibt es und wie unterscheiden sie sich?

5.2 Marketingmix (4P)

5.2.1 Produktpolitik

- Beschreibung der Produktpolitik:
  - Welche Produktvarianten werden angeboten?

5.2.2 Preispolitik

- Festlegung der Preispolitik:
  - Welche Preisstrategie wird verfolgt?

5.2.3 Kommunikationspolitik

- Beschreibung der Kommunikationsstrategie:
  - Wie wird das Produkt beworben?

5.2.4 Distributionspolitik

- Beschreibung der Distributionspolitik:
  - Wie gelangt das Produkt zum Endkunden?

5.3 Kostenplanung

5.3.1 Personalkosten

- Aufschlüsselung der Personalkosten:
  - Kosten für die beteiligten Mitarbeiter und deren Einsatzzeiten

5.3.2 Sachmittelkosten

- Aufschlüsselung der Sachmittelkosten:
  - Kosten für benötigte Hardware und Software

5.4 Wirtschaftlichkeitsberechnung

5.4.1 Gewinnschwellenberechnung

- Berechnung der Gewinnschwelle:
  - Ab welcher Menge erzielt das Projekt Gewinn?

5.4.2 Amortisationsrechnung

- Berechnung der Amortisationszeit:
  - Wie lange dauert es, bis sich das Projekt amortisiert hat?

6 Projektabschluss

6.1 Erreichung des Projektziels

- Zusammenfassung, ob das Projektziel erreicht wurde:
  - Welche Ziele wurden erreicht und welche nicht?

6.2 Änderungen zur anfänglichen Planung

- Darstellung der Änderungen zur ursprünglichen Planung:
  - Welche Änderungen wurden während des Projekts vorgenommen und warum?

6.3 Fazit

- Persönliches Fazit:
  - Wie hat die Teamarbeit funktioniert?
  - Was hat das Projekt persönlich gebracht?

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

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 10

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 6, *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 Projektmangement-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 6, *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, 3

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 6, *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 6, *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. 5

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
               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