Dokumentation zur betrieblichen Projektarbeit

— Mittelstufenprojekt —

BotHQ – ein Discord-Bot-Framework

Untertitel

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-guide**.
2 Alle Begriffe, die im Glossar erklärt werden, sind mit diesem Zeichen ➤ gekennzeichnet und führen per Klick direkt zur Erklärung.
3 **discord-developer**.
4 **discord-privacy**.

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 **jda-github**; **jda-wiki**.

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 **jda-github**; **jda-wiki**.

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.

Die Entscheidung für Scrum wurde aus mehreren Gründen getroffen. Erstens ermöglicht Scrum eine flexible und anpassungsfähige Entwicklung, die besonders nützlich ist, wenn sich Anforderungen im Laufe des Projekts ändern können. Zweitens fördert Scrum eine enge Zusammenarbeit und kontinuierliche Kommunikation im Team, was zu einer besseren Koordination und höherer Produktivität führt. Drittens erleichtert die iterative Natur von Scrum eine regelmäßige Überprüfung und Anpassung des Projekts, was zu einer höheren Qualität des Bot-Framework geführt hat.

Der Methodologie von Scrum folgend wurde das Projekt in drei Sprints unterteilt, wobei jeder Sprint eine feste Dauer von vier Wochen hatte. Dies ermöglichte es dem Team, sich auf kurzfristige Ziele zu konzentrieren und regelmäßig Fortschritte zu präsentieren. Die in unserem Fall wöchentlichen Stand-up-Meetings (Daily Scrum) förderten die Transparenz und halfen dabei, Hindernisse schnell zu identifizieren und zu beseitigen. Durch die Sprint Reviews konnte kontinuierlich Feedback vom ganzen Team eingeholt und in die nächste Planungsphase integriert werden. Insgesamt führte der Einsatz von Scrum zu einer besseren Planbarkeit, höheren Flexibilität und einer kontinuierlichen Verbesserung der Projektarbeit.

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:

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.

- Wahl der Datenhaltung:
- Beschreibung der Datenbankstruktur und des Datenbankmodells

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.

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

- **id** (INTEGER, PRIMARY KEY, AUTOINCREMENT): Eindeutige Identifikationsnummer des Benutzers. Dies ist der Primärschlüssel der Tabelle und wird automatisch inkrementiert.
- **username** (TEXT, UNIQUE, NOT NULL): Der Benutzername des Nutzers, der zur Anmeldung verwendet wird. Dieser muss eindeutig sein.
- **password** (TEXT, NOT NULL): Der Hashwert des Passworts des Benutzers. Aus Sicherheitsgründen wird das Passwort nicht im Klartext gespeichert.
- **roles** (TEXT): Rollen des Benutzers (z.B. Administrator, Benutzer). Diese Spalte kann mehrere Rollen in einem bestimmten Format enthalten (z.B. durch Kommata getrennt).

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** (INTEGER, PRIMARY KEY, AUTOINCREMENT): Eindeutige Identifikationsnummer des Plugins. Dies ist der Primärschlüssel der Tabelle und wird automatisch inkrementiert.
- **name** (TEXT, UNIQUE, NOT NULL): Der Name des Plugins.
- **description** (TEXT, NOT NULL): Eine Beschreibung des Plugins, die dessen Funktionalität erklärt.
- **enabled** (BOOLEAN, NOT NULL, DEFAULT FALSE): Ein Flag, das angibt, ob das Plugin aktiviert (TRUE) oder deaktiviert (FALSE) ist.
- **version** (TEXT, NOT NULL): Die Versionsnummer des Plugins.
- **created_at** (DATETIME, DEFAULT CURRENT_TIMESTAMP): Das Datum und die Uhrzeit, wann das Plugin erstellt wurde.

Tabelle `servers` Die Tabelle `servers` speichert Informationen über die Discord-Server, auf denen der Bot installiert ist.

- **id** (INTEGER, PRIMARY KEY, AUTOINCREMENT): Eindeutige Identifikationsnummer des Servers. Dies ist der Primärschlüssel der Tabelle und wird automatisch inkrementiert.

- **name** (TEXT, NOT NULL): Der Name des Discord-Servers.
- **owner_id** (INTEGER, NOT NULL, FOREIGN KEY REFERENCES users(id)): Die ID des Benutzers, der der Besitzer des Servers ist. Diese Spalte verweist auf `users.id`.
- **created_at** (DATETIME, DEFAULT CURRENT_TIMESTAMP): Das Datum und die Uhrzeit, wann der Server in der Datenbank erstellt wurde.
- **updated_at** (DATETIME, DEFAULT CURRENT_TIMESTAMP): Das Datum und die Uhrzeit, wann der Server zuletzt aktualisiert wurde.

Tabelle `server_plugins` Die Tabelle `server_plugins` verknüpft die Server und die Plugins miteinander und speichert spezifische Konfigurationen für jedes Plugin auf einem Server.

- **server_id** (INTEGER, NOT NULL, FOREIGN KEY REFERENCES servers(id)): Die ID des Servers, auf dem das Plugin installiert ist. Diese Spalte verweist auf `servers.id`.
- **plugin_id** (INTEGER, NOT NULL, FOREIGN KEY REFERENCES plugins(id)): Die ID des Plugins, das auf dem Server installiert ist. Diese Spalte verweist auf `plugins.id`.
- **enabled** (BOOLEAN, NOT NULL, DEFAULT FALSE): Ein Flag, das angibt, ob das Plugin auf diesem Server aktiviert (TRUE) oder deaktiviert (FALSE) ist.
- **configurations** (TEXT): Ein JSON-String, der die Konfigurationseinstellungen des Plugins speichert. Dieser String enthält die spezifischen Einstellungen, die der Benutzer für das Plugin auf diesem Server vorgenommen hat.

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.

- **Benutzer und Server**: Ein Benutzer (`users`) kann mehrere Server (`servers`) besitzen. Dies wird durch die Fremdschlüsselbeziehung `servers.owner_id` zu `users.id` dargestellt. Ein Benutzer kann somit mehrere Server verwalten.
- **Server und Plugins**: Ein Server (`servers`) kann mehrere Plugins (`plugins`) verwenden. Diese Beziehung wird durch die Verknüpfungstabelle `server_plugins` realisiert. Die Tabelle `server_plugins` enthält die Fremdschlüssel `server_plugins.server_id` und `server_plugins.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

Benutzer und Server (1:n Beziehung)

- **Primärschlüssel**: `users.id`
- **Fremdschlüssel**: `servers.owner_id`

- **Beschreibung**: Ein Benutzer kann mehrere Server besitzen. Jeder Server hat einen Besitzer, der in der `owner_id`-Spalte der `servers`-Tabelle gespeichert ist.

Server und Plugins (n:m Beziehung)

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

4.2.6 Zusammenfassung

Die Datenbankstruktur und das Modell bieten eine robuste 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 `users` speichert Benutzerinformationen, die Tabelle `servers` speichert Serverinformationen, die Tabelle `plugins` speichert Plugin-Informationen, und die Verknüpfungstabelle `server_plugins` speichert die Zuordnungen und Konfigurationen der Plugins zu den jeweiligen Servern.

4.2.7 Design Patterns

- Einsatz von Design Patterns:
- Welche Design Patterns werden verwendet und warum?

4.3 Qualitätssicherung

4.3.1 Teststrategien

- Beschreibung der Teststrategien:
- Welche Tests werden durchgeführt (Unit Tests, Integrationstests, Systemtests)?

4.3.2 Testfälle

- Konkrete Testfälle:
  - Beschreibung der durchgeführten Testfälle und deren Ergebnisse

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

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

5.3.1 Personalkosten

Projektdaten

Projektstart:	14. Februar 2024
Projektende:	29. Mai 2024
Arbeitszeit:	Jeden Mittwoch von 8:10 bis 13:20 mit 2 Pausen je 20 Minuten

Berechnung der Arbeitszeit

Die Arbeitszeit pro Mittwoch beträgt 4 Stunden und 30 Minuten (270 Minuten) abzüglich 40 Minuten Pause, also 3 Stunden und 50 Minuten (230 Minuten oder 3.83 Stunden).

$$\text{Arbeitszeit pro Woche} = 3.83 \text{ Stunden}$$

Der Zeitraum von 14. Februar 2024 bis 29. Mai 2024 umfasst insgesamt 16 Wochen.

Stundensätze

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

Berechnung der Personalkosten

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

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

Ergebnis

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

5.0.1 Sachmittelkosten

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

5.1 Wirtschaftlichkeitsberechnung

5.1.1 Gewinnschwellenberechnung

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

5.1.2 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

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

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

R

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

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 1: Beispielcode für einen ReadyListener in JDA