OST
Ostschweizer
Fachhochschule

Discord-Bot für Vereine

Semester- und Bachelorarbeit im Frühlingssemester 2025

Ausdruck:
06.06.2025

Team / Autoren:
Vanessa Alves und Marco Schnider

Referent:
Prof. Dr. Frieder Loch

Koreferent:
Dr. Michael Sollfrank

Gegenleser:
Hannes Badertscher

Abstract

Diese Arbeit behandelt die Konzeption, Implementierung und initiale Evaluation eines Discord-Bots zur automatisierten Verwaltung vereinsrelevanter Prozesse. Das Ziel bestand darin, ein System zu entwickeln, welches insbesondere die Mitgliederverwaltung, Eventorganisation und Gamification innerhalb einer bestehenden Discord-Infrastruktur integriert. Besonderer Fokus lag auf der Implementierung eines rollenbasierten Berechtigungssystems, der Synchronisation von Events über die Google Calendar API und der Integration eines wissenschaftlich fundierten Gamification-Systems zur Förderung der Mitgliederaktivität und -motivation.

Der Discord-Bot wurde in TypeScript unter Verwendung der Bibliothek `discord.js` entwickelt und nutzt die Discord-API für Nutzerinteraktionen. Vereinsrelevante Daten werden in einer MongoDB-Datenbank gespeichert, die eine dokumentenorientierte Struktur für Mitglieder, Rollen, Events und Punktestände bietet. Die Integration des Google Calendar wurde mittels der offiziellen Google API `Node.js`-Clientbibliothek realisiert, wodurch eine automatisierte Synchronisierung von Events ermöglicht wird. Das rollenbasierte Berechtigungskonzept nutzt Discord-Rollen in Verbindung mit Slash-Befehlen, um Administrations- und Nutzerfunktionen zu steuern. Das Deployment erfolgt derzeit manuell über Docker-Container auf einem dedizierten Server.

Für die Entwicklung des Gamification-Systems wurde zunächst eine Mitgliederumfrage durchgeführt, um motivationsfördernde Elemente wie Punktesystem, Ranglisten und Belohnungen nutzerzentriert zu gestalten. Aufbauend auf diesen Ergebnissen, sowie unter der Berücksichtigung von wissenschaftlichen Metaanalysen zum Thema Gamification, wurde das Gamification-Konzept entwickelt. Die Evaluation des Gesamtsystems erfolgte durch strukturierte Usability-Tests und Interviews.

Der entwickelte Discord-Bot wurde prototypisch auf einem realen Vereinsserver implementiert und evaluiert. Die Usability-Tests bestätigten, dass die zentralen Funktionen wie Mitgliederregistrierung, Eventbenachrichtigungen und Punkteverwaltung von den Nutzenden als benutzerfreundlich wahrgenommen wurden und eine intuitive Bedienung ermöglichten. Das Gamification-System basiert auf einem Punktesystem, das für Aktivitäten wie Eventteilnahmen, erledigte Aufgaben und Spenden Punkte vergibt. Eine Mitgliederumfrage belegte, dass saisonale Ranglisten, transparente Regeln und Sachprämien als motivierend wahrgenommen wurden, während übermässiger Wettbewerb und komplexe Punkteberechnungen tendenziell kritisch gesehen wurden. Aussagen zur langfristigen Wirkung des Systems auf die Vereinsorganisation können aufgrund der limitierten Einsatzdauer noch nicht getroffen werden, jedoch wurde das System insgesamt als potenziell wertvoll für die Förderung von Engagement und die Automatisierung von Verwaltungsprozessen eingeschätzt.

Perspektiven für eine künftige Erweiterung umfassen die Implementierung eines Admin-Dashboards, um die Verwaltung von Mitgliederdaten, Ranglisten und Transaktionen zu erleichtern, die Integration atomarer Transaktionen zur Sicherstellung einer konsistenten Datenverarbeitung und die Entkopplung der Discord-spezifischen Logik von den Datenbankmodellen. Zudem sollten eine automatische Bereitstellung, automatisierte Kanal-Erstellung sowie Webhooks für Echtzeit-Updates berücksichtigt werden, um das System langfristig effizienter und benutzerfreundlicher zu gestalten. Insgesamt leistet der entwickelte Discord-Bot einen wertvollen Beitrag zur modernen Vereinsorganisation und bildet eine solide Basis für weitere Ausbaustufen.

Management Summary

Ausgangslage

Discord hat sich seit seiner Einführung im Jahr 2015 von einer reinen Gaming-App zu einem vielseitigen Kommunikationswerkzeug entwickelt, das zunehmend von Jugend- und Freizeitorganisationen als digitaler Treffpunkt genutzt wird. Viele Online-Vereine verwenden Discord als zentrale Plattform, um ihre Mitglieder zu vernetzen, Events zu organisieren und die Zusammenarbeit zu fördern. Trotz der umfangreichen Funktionen dieser Plattform sind bestimmte Verwaltungsaufgaben wie die Aufnahme neuer Mitglieder oder die Rollenvergabe ohne ergänzende Softwarelösungen oft aufwändig und fehleranfällig. Dies kann besonders bei einer grossen Mitgliederzahl oder häufigen Veranstaltungen zu organisatorischen Herausforderungen führen. Darüber hinaus entstehen durch die Verwaltung personenbezogener Daten zusätzliche Anforderungen an den Datenschutz.

Das Ziel dieser Arbeit war es, eine spezifische Erweiterung für Discord in Form eines Bots zu entwickeln, der diese Herausforderungen adressiert. Der Bot soll die Mitgliederverwaltung automatisieren und das Engagement der Mitglieder durch ein Gamification-System fördern. Zudem wurde ein Datenschutzkonzept erstellt, um die personenbezogenen Daten der Mitglieder zu schützen.

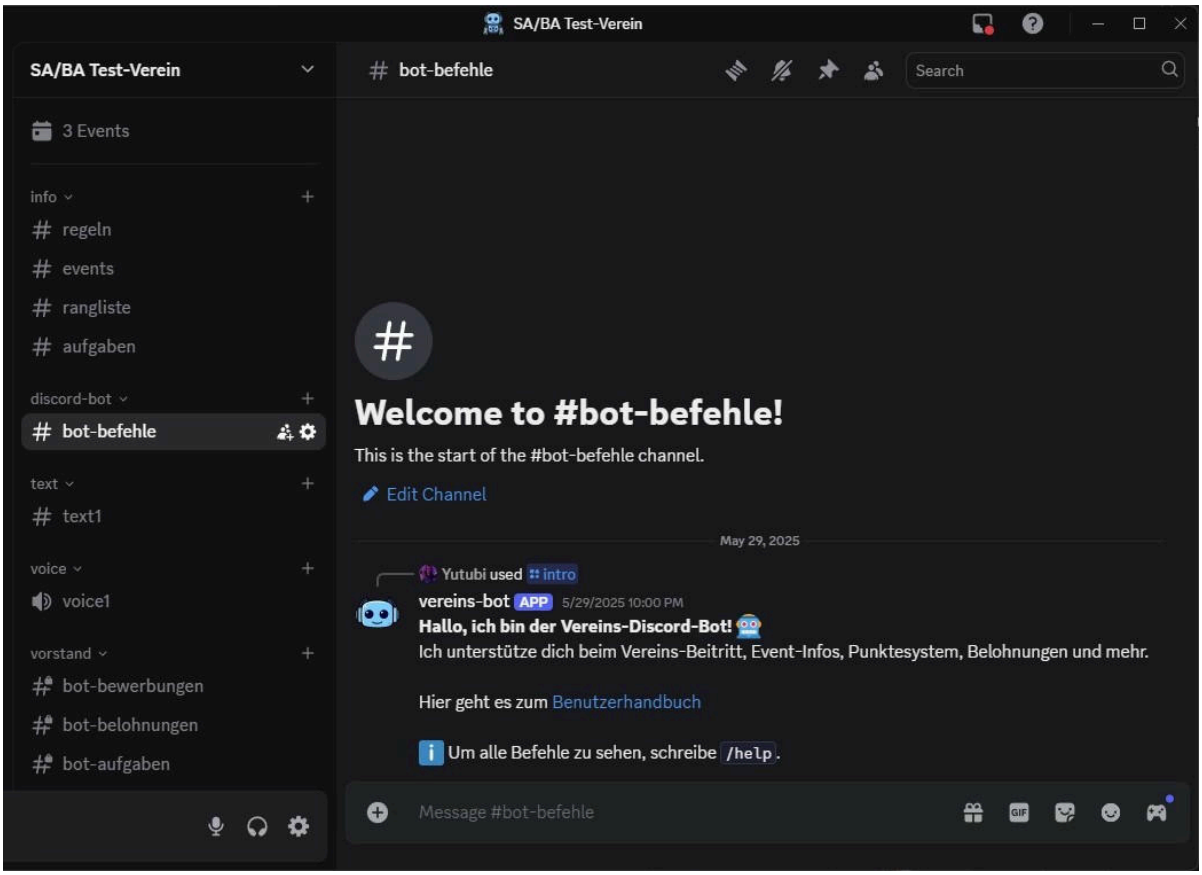


Abb. 1: Der Discord-Bot in einem Test-Server inklusive Willkommensnachricht und Hinweisen zur Bot-Benutzung

Vorgehen und Technologien

Zu Beginn der Arbeit wurden bestehende Discord-Bots und Vereins-Software analysiert, um die Anforderungen besser zu verstehen und einen eigenen Lösungsansatz zu entwickeln. Auf Basis wissenschaftlicher Studien zu Gamification und den Ergebnissen einer Mitgliederumfrage wurde ein passendes Gamification-Konzept erarbeitet. Parallel dazu entstand ein Sicherheitskonzept, das sich an den Anforderungen der Datenschutzgrundverordnung orientiert.

Die technische Umsetzung erfolgte mit TypeScript und der Discord.js-Bibliothek. Mitgliederdaten, Punkte und Events werden in einer MongoDB-Datenbank gespeichert, um eine flexible Verwaltung zu ermöglichen. Für die Synchronisierung von Events wurde die Google Calendar API integriert. Nach der Implementierung wurde ein Usability-Test durchgeführt, um die Benutzerfreundlichkeit zu validieren und Optimierungen abzuleiten.

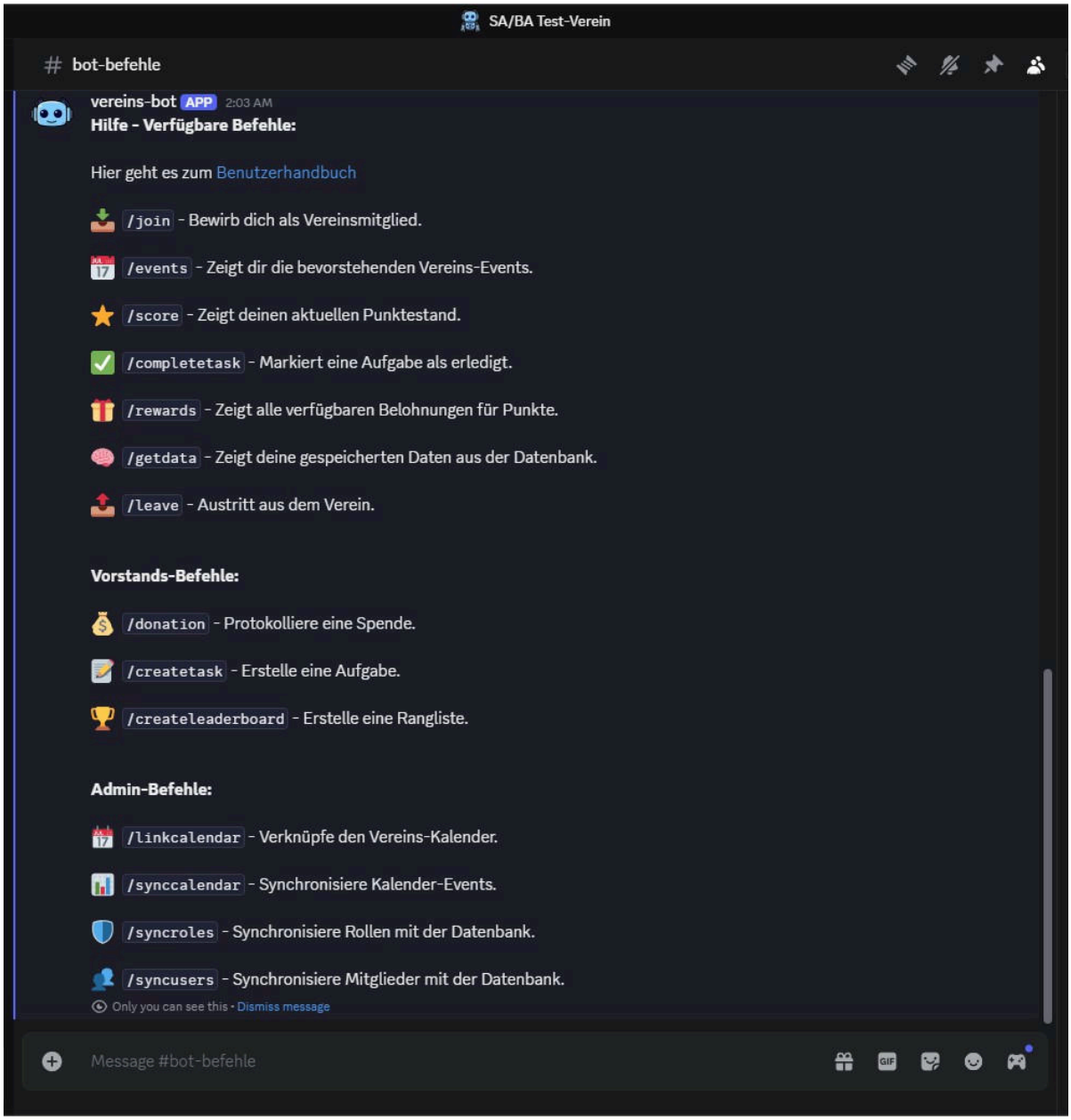


Abb. 2: Übersicht aller verfügbaren Befehle des Discord-Bots

Ergebnisse

Das Ergebnis dieser Arbeit ist ein einsatzfähiger Discord-Bot, der speziell für Vereine entwickelt wurde. Er automatisiert die Aufnahme neuer Mitglieder, vergibt Rollen auf dem Discord-Server und informiert Mitglieder über bevorstehende Events. Das implementierte Gamification-System ermöglicht es, für Vereinsaktivitäten Punkte zu sammeln und diese gegen Belohnungen einzutauschen. Eine Rangliste zeigt die aktivsten Mitglieder und motiviert zur Teilnahme.

Der Bot wurde erfolgreich in einer realen Vereinsumgebung getestet und konnte seine wichtigsten Funktionen dabei unter Beweis stellen. Die Usability-Tests zeigten, dass der Bot insgesamt eine gute Benutzerfreundlichkeit bietet und die Kernfunktionen von den Nutzenden intuitiv genutzt werden können. Das Punktesystem wurde zwar von einigen Mitgliedern als motivierend empfunden, jedoch zeigte sich, dass ein Teil der Mitglieder diesem System noch zurückhaltend gegenübersteht. Dies deutet darauf hin, dass das Punktesystem künftig noch zielgerichteter auf die Vereinsaktivitäten und die Interessen der Mitglieder angepasst werden sollte, um langfristig eine stärkere Motivation zu erzielen.

Trotz der erfolgreichen Umsetzung im Testbetrieb gibt es noch einige Punkte, die für den praktischen Einsatz weiterentwickelt werden sollten. So fehlt beispielsweise ein Admin-Dashboard, mit dem die Verwaltung von Mitgliedern, Events und Punkten einfach gesteuert werden kann. Die Punktevergabe kann momentan nicht losgelöst von Aufgaben oder Spenden ausgeführt werden. Für die Verwaltung von Discord-Kanälen und die Anpassung von Rollen sind aktuell noch manuelle Schritte notwendig. Eine automatische Bereitstellung des Bots ist derzeit noch nicht realisiert. Dies zeigt, dass der Bot eine solide Basis bietet, aber in einigen Bereichen noch Potenzial zur Verbesserung besteht, um den Betrieb für Vereine langfristig zu erleichtern und noch effizienter zu gestalten.

Abb. 3: Allzeit Rangliste im Ranglisten-Kanal des Discord-Servers

Inhaltsverzeichnis

Kapitel I
Produkt Dokumentation

1 Einleitung

Viele Online Communitys und Vereine, nutzen Discord als zentrale Plattform für die Kommunikation und Organisation. Während Discord von sich aus Funktionen zur Mitgliederverwaltung, Rollenvergabe und Eventorganisation bietet, stossen Vereine mit einer wachsenden Mitgliederzahl und Fluktuationen an organisatorische Grenzen. Aus diesem Grund befasst sich diese Arbeit damit, wie sich die Organisation und Verwaltung von solchen Vereinen mittels eines Discord-Bots automatisieren und vereinfachen lässt. Ausserdem wird untersucht, wie durch ein Gamification-System die Interaktion auf dem Server erhöht und umgesetzt werden kann. Ein Discord-Bot kann als flexible Erweiterung eines Discord-Servers betrachtet werden, die eine einfache Integration ermöglicht.

Die identifizierten Herausforderungen werden anhand des Vereins „EGSwiss“ analysiert. Das Endprodukt ist jedoch keine massgeschneiderte Lösung ausschliesslich für diesen Verein, sondern ein allgemeines Hilfsmittel für die Verwaltung von Online Vereinen in Discord.

1.1 Realbeispiel: EGSwiss

Der Gaming Verein EGSwiss ist eine Community, die sich auf die Förderung der Gemeinschaft in der Schweizer Gaming-Szene spezialisiert hat. Neben regelmässigen Events verwaltet der Verein eine aktive Discord-Community, die unter anderem für die Vereinsverwaltung genutzt wird.

Durch die aktive Rolle von Vanessa Alves als Mitgründerin und aktuelle Präsidentin des Vereins stehen für diese Arbeit wertvolle organisatorische Einblicke sowie die Möglichkeit einer praxisnahen Evaluation zur Verfügung.

Aktuell werden im Vorstand viele Prozesse manuell oder über externe Tools wie Microsoft Excel, Google Calendar und der Discord-Rollenverwaltung gesteuert. Dies führt zu einem erhöhten Verwaltungsaufwand, inkonsistenter Datenhaltung und potenziellen Fehlerquellen.

1.2 Problemstellung

Die folgenden Aspekte werden als Herausforderungen bei der Verwaltung von Online Vereinen identifiziert und werden im Verein EGSwiss entweder durch manuelle Prozesse oder noch gar nicht umgesetzt:

- Mitgliederverwaltung: Die Aufnahme von neuen Mitgliedern in die Vereinsliste, Aktualisierung der Rollen und Verwaltung der Austritte.
- Rollenmanagement: Die korrekte und automatisierte Vergabe von Rechten und Rollen an Mitglieder basierend auf ihrem Vereinsstatus.
- Eventmanagement: Die Erstellung und Verwaltung von Event-Erinnerungen sowie Teilnehmerbenachrichtigungen.
- Gamification: Ein einheitliches System, um Mitgliederaktivitäten zu erfassen, zu belohnen und um die Vereinsaktivität zu steigern.

Um diese Anforderungen zu bewältigen, wird ein Discord-Bot entwickelt und untersucht, wie eine solche Lösung für online Vereine wie EGSwiss konzipiert, implementiert und in die bestehende Infrastruktur eingebunden werden kann.

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung eines Discord-Bots, der zentrale Verwaltungsprozesse automatisiert und nahtlos in bestehende Systeme integriert. Um das zu erreichen, werden die in der folgenden Tabelle definierten Hauptziele verfolgt.

Hauptziel	Beschreibung
Automatisierte Mitgliederverwaltung	Aufnahme neuer Mitglieder über ein Formular oder manuell und automatische Synchronisation mit der Vereinsdatenbank. Automatische Zuweisung und Verwaltung von Discord-Rollen basierend auf dem Mitgliedsstatus. Erinnerung an Mitgliedsbeiträge und Verwaltung der Austritte.
Integration eines Event-Kalenders	Anbindung an die Google Calendar API zur Verwaltung von Events. Automatische Benachrichtigung von interessierten Mitgliedern bei bevorstehenden Events.
Gamification-System	Punktevergabe für Aktivitäten im Discord-Server. Belohnungssystem für Mitglieder mit hoher Aktivität. Möglichkeit, Punkte für Vergünstigungen im Vereins-Shop oder andere Belohnungen einzulösen.

Tab. 1: Beschreibung der Hauptziele für dieses Projekt

Die Automatisierung der Vereinsverwaltung durch einen Discord-Bot bietet signifikante Vorteile hinsichtlich Effizienz, Skalierbarkeit und Benutzerfreundlichkeit. Im weiteren Verlauf der Arbeit werden die konkreten technischen Anforderungen definiert (siehe Abschnitt 2), Konzepte für die Gamification (siehe Abschnitt 4) und Sicherheit (siehe Abschnitt 3) vertieft, die Architektur (siehe Abschnitt 5.2) im Detail beschrieben und die Implementierung (siehe Abschnitt 7) sowie die Evaluation des Systems durchgeführt (siehe Abschnitt 8).

1.4 Rahmenbedingungen

Dieses Projekt wird als Semesterarbeit / Bachelorarbeit durchgeführt. Der Schwerpunkt liegt auf der Konzeption, Entwicklung und Evaluation eines Discord-Bots für Vereine, um die Vereinsorganisation und Interaktion der Mitglieder zu verbessern. Das Projekt umfasst die Planung, Implementierung sowie die Durchführung von Usability-Tests.

- **Arbeitstyp**: Semesterarbeit (240 Stunden) + Bachelorarbeit (360 Stunden)
- **Gesamtarbeitsaufwand**: 600 Stunden
- **ECTS-Punkte**: 8 ECTS (Semesterarbeit) und 12 ECTS (Bachelorarbeit)

1.5 Stand der Technik

Bevor im weiteren Verlauf die Architektur und die Funktionalitäten des geplanten Systems erläutert werden, soll zunächst eine Analyse bestehender Lösungen erfolgen. Im Bereich der Gamification existieren bereits mehrere etablierte Discord-Bots, die Nutzeraktivitäten erfassen und durch Belohnungssysteme fördern.

Darüber hinaus stellt die Veranstaltungsplanung ein weiteres relevantes Anwendungsfeld dar. Hier kommen Bots zum Einsatz, die eine Integration mit Kalenderdiensten wie Google Calendar ermöglichen, um Events effizient innerhalb von Discord zu organisieren und zu kommunizieren.

Ziel dieses Kapitels ist es, bestehende Lösungen im Bereich Discord-Gamification und Eventverwaltung zu analysieren und zentrale Unterschiede sowie Abgrenzungspotenziale gegenüber dem eigenen Projekt herauszuarbeiten.

1.5.1 Gamification Bots

Es existieren bereits Discord-Bots, welche für ein Gamification-System benutzt werden können. Die Systeme konzentrieren sich darauf, Interaktionen innerhalb des Discord-Servers zu fördern.

MEE6[1] ist ein Discord-Bot mit verschiedenen Funktionen wie benutzerdefinierte Befehle, Ranglisten, Benachrichtigungen für Beiträge auf Social-Media, Moderationswerkzeuge und mehr. Darunter auch ein System, bei dem man Punkte sammeln kann und dafür bestimmte Vorteile bekommt, wie zum Beispiel Zugriff auf exklusive Kanäle im Server. Nutzer können Punkte

[1] https://mee6.xyz/de/

sammeln, in dem sie Nachrichten schreiben. Der Fokus liegt hier also lediglich darauf, die Aktivität im Server zu steigern, nicht jedoch die allgemeine Aktivität eines Vereins.

Ein weiterer verbreiteter Bot ist Arcane[2], damit können Nutzer nicht nur Punkte für Aktivität in Textchats, sondern auch in Sprachchats sammeln. Den aktivsten Nutzern können verschieden Rollen zugeteilt werden. Ausserdem gibt es Ranglisten bei denen die aktivsten Nutzer der Woche oder des Monats dargestellt werden. Auch bei Arcane liegt der Fokus darauf, die Aktivität im Discord-Server zu steigern.

1.5.2 Kalender Bots

Für die Eventverwaltung und Terminplanung gibt es ebenfalls spezialisierte Bots, die eine Synchronisation mit Google Calendar ermöglichen. Diese Bots erleichtern die Organisation von Veranstaltungen, bieten jedoch keine Verbindung zu Gamification-Systemen.

Chronicle Bot[3] erlaubt es Events in Google Calendar zu planen und diese automatisch mit Discord Events zu synchronisieren. Zudem können Erinnerungen für anstehende Events automatisch im Discord-Server gesendet werden. Diese Benachrichtigungen können von Administratoren beliebig personalisiert werden. Der Bot ist lediglich auf Kalender Funktionen fokussiert und kann nicht direkt mit einem Punktesystem verbunden werden.

Eine Alternative zu Chronicle Bot ist sesh[4]. Dieser Bot bietet ähnliche Funktionen wie Chronicle Bot. Zusätzlich zu Synchronisationen mit Google Calendar bietet sesh auch eine Umfrage-Funktion und die Möglichkeit Event-Teilnehmern Rollen zuzuteilen.

1.5.3 Weitere Tools

Es konnte kein Discord-Bot identifiziert werden, der gezielt auf die Vereinsverwaltung ausgerichtet ist. Allerdings gibt es ausserhalb von Discord andere Software für die Vereinsverwaltung.

Plattformen wie WildApricot[5] bieten Webseiten-Ersteller, Online Kalender oder Zahlungsmanagement Funktionen. Andere Anbieter wie MTH Software[6] legen den Fokus auf Mitglieder- und Rechnungsverwaltung. Vereinssoftware mit einem integrierten Gamification-System konnte nicht gefunden werden.

Genially[7] bietet allgemeine Gamification-Tools die von Firmen, Schulen und Vereinen genutzt werden können. Der Fokus liegt hierbei ausschliesslich auf Animationen und Audio welche Beiträge eine Video-Spiel-Ästhetik verleihen soll. Ausserdem können Quizze erstellt werden.

1.5.4 Bewertung bestehender Systeme und Abgrenzung

Die analysierten Discord-Bots im Bereich Gamification, wie MEE6 und Arcane, fördern hauptsächlich die Aktivität innerhalb des Discord-Servers. Sie belohnen Nutzer für das Schreiben von Nachrichten oder die Teilnahme in Sprachkanälen. Ein Bezug zu vereinsbezogenen Aufgaben oder realen Aktivitäten fehlt jedoch.

Auch Kalender-Bots wie Chronicle und sesh ermöglichen zwar eine effektive Eventplanung in Verbindung mit Google Calendar, verfügen jedoch über keine gamifizierten Elemente zur Steigerung der Teilnehmerzahl oder organisatorische Mithilfe im Verein.

Externe Softwarelösungen zur Vereinsverwaltung bieten umfassende Verwaltungsfunktionen, sind jedoch nicht mit Discord verknüpft und enthalten keine spielerischen Anreize.

Das vorgestellte Projekt grenzt sich von bestehenden Lösungen ab, indem es Gamification mit Veranstaltungs- und Mitgliederverwaltung verbindet. Ziel ist es, das Engagement im Verein sowohl auf Discord als auch bei physischen Vereinsaktivitäten gezielt zu fördern.

[2] https://arcane.bot/
[3] https://chroniclebot.com/
[4] https://sesh.fyi/
[5] https://www.wildapricot.com/
[6] https://www.mth-software.de
[7] https://genially.com/features/gamification/

2 Anforderungen

Dieses Kapitel stellt die Anforderungen an den Discord-Bot für den Verein systematisch dar. Die Anforderungen basieren auf klar definierten User Stories , welche die Bedürfnisse der Nutzer abbilden. Aus diesen User Stories werden die Functional Requirements abgeleitet, welche die konkreten Funktionen des Bots spezifizieren. Darauf aufbauend definieren die Use Cases , wie diese Anforderungen in der geplanten Umsetzung realisiert werden. Neben den Functional Requirements werden auch die Non-Functional Requirements berücksichtigt, um Qualitätsmerkmale wie Sicherheit, Skalierbarkeit und Benutzerfreundlichkeit sicherzustellen. Zudem erfolgt eine Priorisierung der Anforderungen , damit die wichtigsten Funktionen frühzeitig implementiert werden.

2.1 Priorisierung der Anforderungen

Damit der Entwicklungsprozess auf die wichtigsten Features fokussiert bleibt, haben diese eine höhere Priorität. Dadurch kann zügig ein Prototyp entwickelt, getestet und auf Basis von Nutzerfeedback iterativ verbessert werden. Die nicht optionalen Use Cases und Non-Functional Requirements bilden das MVP (Minimum Viable Product) und stellen die funktionale Basis des Projekts dar.

Die Nachfolgende Auflistung beschreibt die Prioritäts-Kategorien und deren Bedeutung:

Prioritäts-Kategorie	**Beschreibung**
Hoch (MVP) – Essenziell für die erste Version	Diesese Anforderungen sind direkt mit den Kernanforderungen des MVP verknüpft und müssen für eine funktionale Erstversion vorhanden sein.
Mittel – Wichtige Funktionen nach dem MVP	Diese Anforderungen ergänzen das System sinnvoll, sind jedoch nicht kritisch für die erste Version.
Niedrig – Erweiterungen für spätere Versionen	Diese Anforderungen verbessern die Benutzerfreundlichkeit und Verwaltung, sind aber nicht erforderlich für den ersten MVP-Release.

Tab. 2: Beschreibung der Prioritäts-Kategorien

2.2 Legende zum Erfüllungsgrad der Anforderungen

Die Resultate der Anforderungen werden jeweils in den Anforderungstabellen dokumentiert. Zusätzlich wird der Status farblich hervorgehoben, um eine schnelle Übersicht über den Erfüllungsgrad zu ermöglichen.

Farbe	**Status**
	Ziel der Anforderung wurde erreicht.
	Ziel der Anforderung wurde zum Teil oder über einen anderen Weg erreicht.
	Ziel der Anforderung wurde nicht erreicht.

Tab. 3: Beschreibung der Resultats-Kategorien

2.3 User Stories

Die User Stories erfassen die Bedürfnisse und Erwartungen der Nutzer an den Discord-Bot und formulieren sie aus der Perspektive der Endanwender. Sie bilden die Grundlage für die Functional Requirements , die daraus abgeleitet werden. Die Analyse konzentriert sich auf die Nutzerrollen Vorstand, Mitglied und Nicht-Mitglied.

2.3.1 User Story: Vorstandsmitglied

Als Vorstandsmitglied...

- möchte ich, dass Neumitglieder durch ein Formular automatisch in die Vereinsliste aufgenommen werden, um die Verwaltung zu vereinfachen.
- möchte ich, dass die Rollen der Mitglieder automatisch vergeben werden (Nicht-Mitglied, Mitglied, Vorstand), um sicherzustellen, dass jeder die richtigen Berechtigungen hat.
- möchte ich Mitglieder nach einer bestimmten Zeit benachrichtigen oder entfernen können, um die Mitgliederliste aktuell zu halten.
- möchte ich die Google Calendar API verwenden, um unsere Vereins-Events automatisch in Discord zu integrieren, damit alle Mitglieder auf dem Laufenden bleiben.
- möchte ich Erinnerungen an interessierte Teilnehmer eines kommenden Events senden, damit die Teilnahmequote steigt.
- möchte ich sicherstellen, dass sensible Befehle nur von autorisierten Rollen genutzt werden, um die Sicherheit der Mitgliederdaten zu gewährleisten.
- möchte ich, dass die Speicherung von Mitgliederdaten DSGVO-konform erfolgt, um rechtliche Vorgaben einzuhalten.
- möchte ich ein Gamification-System implementieren, um die Aktivität auf dem Server zu fördern und die Mitgliederbindung zu stärken.
- möchte ich eine Rangliste der aktivsten Mitglieder einsehen, um die Strategien und Belohnungen entsprechend anzupassen.
- möchte ich über ein Admin-Dashboard Mitglieder, Rollen und Events verwalten können, um die Organisation übersichtlicher und effizienter zu gestalten.
- möchte ich Gamification-Daten einsehen, um die Motivation der Mitglieder besser zu verstehen und zu fördern.

2.3.2 User Story: Mitglied

Als Mitglied...

- möchte ich benachrichtigt werden, wenn ich zur jährlichen General- / Hauptversammlung eingeladen werde, um meine Teilnahme rechtzeitig planen zu können.
- möchte ich direkt benachrichtigt werden, wenn ich auf den „interessiert“-Button für ein Event klicke, damit ich keine wichtigen Informationen verpasse.
- möchte ich die Möglichkeit haben, meine Daten jederzeit löschen zu lassen, um meine Privatsphäre zu schützen.
- möchte ich Punkte für meine Aktivitäten im Server sammeln, um Belohnungen und Rabatte im Vereins-Shop zu erhalten.
- möchte ich an Umfragen und Events teilnehmen, um meine Punkte zu steigern und in der Rangliste aufsteigen zu können.

2.3.3 User Story: Nicht-Mitglied

Als Nicht-Mitglied...

- möchte ich ein einfaches Formular ausfüllen, um mich für den Verein zu bewerben, damit ich beitreten kann und die entsprechende Rolle erhalte.

2.4 Functional Requirements

Die Functional Requirements leiten sich aus den User Stories ab und spezifizieren die erwarteten Funktionen des Discord-Bots. Sie beschreiben, welche Features der Bot implementieren muss, um die definierten Anforderungen zu erfüllen. Zudem sind sie eine zentrale Grundlage für die Use Cases , die detailliert darstellen, wie Nutzer mit dem Bot interagieren.

2.4.1 Mitgliederverwaltung

ID	**FR1**
Name	Aufnahme neuer Mitglieder
Beschreibung	• Der Discord-Bot muss ein Formular bereitstellen, mit dem sich neue Mitglieder bewerben können. • Die eingegebenen Daten müssen in einer Vereinsliste beziehungsweise Datenbank persistent gespeichert werden. • Der Vorstand muss bei Neueintritten benachrichtigt werden. • Der Vorstand muss über den Discord-Bot den Status von Mitgliedern ändern können.

Tab. 4: Beschreibung Functional Requirement 1

ID	**FR2**
Name	Automatische Rollenvergabe

Beschreibung	• Der Discord-Bot muss Mitglieder basierend auf ihrem Status automatisch in eine der folgenden Rollen einteilen:
	  ▸ Vorstand
	  ▸ Mitglied
	  ▸ Offene Bewerbung
	  ▸ Nicht-Mitglied, beziehungsweise Gast
	  ▸ Weitere Rollen (beispielsweise Esports-Team oder Discord-Moderator)
	• Die Rollen müssen dynamisch konfigurierbar sein.

Tab. 5: Beschreibung Functional Requirement 2

ID	**FR3**
Name	Verwaltung und Entfernung von Mitgliedern
Beschreibung	• Mitglieder sollen automatisch eine Erinnerung zur Zahlung ihrer Mitgliedgebühren erhalten. • Der Discord-Bot muss Mitglieder entfernen können, wenn diese nicht mehr zum Verein gehören. • Der Vorstand muss die Möglichkeit haben, Mitglieder nach einer festgelegten Zeitspanne zu benachrichtigen oder zu entfernen, um die Mitgliederliste aktuell zu halten.

Tab. 6: Beschreibung Functional Requirement 3

2.4.2 Eventkalender-Anbindung

ID	**FR4**
Name	Automatische Event-Benachrichtigungen
Beschreibung	• Der Discord-Bot muss über die Google Calendar API oder iCal Events aus dem Vereinskalender abrufen. • Events müssen in einem festgelegten Discord-Kanal angekündigt werden.

Tab. 7: Beschreibung Functional Requirement 4

ID	**FR5**
Name	Erinnerungen an Events
Beschreibung	• Der Discord-Bot muss automatische Erinnerungen für Events versenden. • Teilnehmer, die auf den „interessiert“-Button in Discord geklickt haben, müssen gezielt erwähnt werden.

Tab. 8: Beschreibung Functional Requirement 5

2.4.3 Sicherheit und Datenschutz

ID	**FR6**
Name	Rechteverwaltung für Befehle
Beschreibung	• Der Discord-Bot muss die Zugriffsrechte kontrollieren, damit nur bestimmte Rollen sensible Befehle nutzen können.

Tab. 9: Beschreibung Functional Requirement 6

ID	**FR7**
Name	Opt-in für Datenspeicherung
Beschreibung	• Mitglieder müssen explizit der Speicherung ihrer Daten zustimmen (z. B. durch das Akzeptieren der Discord-Server-Regeln und Datenschutzrichtlinien).

Tab. 10: Beschreibung Functional Requirement 7

ID	**FR8**
Name	DSGVO-Konformität
Beschreibung	• Der Discord-Bot darf nur notwendige Daten speichern. • Die Daten müssen vertraulich behandelt werden. • Mitglieder müssen über den Discord-Bot jederzeit Einsicht in ihre gespeicherten Daten erhalten. • Die Löschung der Daten soll nach dem Vereins-Austritt und einer zusätzlich definierten Bedenkfrist erfolgen.

Tab. 11: Beschreibung Functional Requirement 8

2.4.4 Gamification-System

ID	**FR9**
Name	Punkte und Ränge für Aktivität
Beschreibung	• Der Discord-Bot muss gewisse Aktivitäten der Mitglieder auf dem Discord-Server erfassen. (z. B. Umfragen ausfüllen, an Events teilnehmen) • Mitglieder müssen Punkte für bestimmte Aktionen erhalten. • Das System muss fair bleiben und darf keine offensichtliche Manipulationsmöglichkeiten bieten.

Tab. 12: Beschreibung Functional Requirement 9

ID	**FR10**
Name	Belohnungen für Interaktionen

Beschreibung	• Der Discord-Bot muss Mitglieder über verfügbare Gamification-Interaktionen benachrichtigen.
	• Der Discord-Bot vergibt Punkte für:
	  ▸ Umfragen-Teilnahmen
	  ▸ Event-Teilnahmen
	  ▸ Spenden
	  ▸ Weitere Interaktionen

Tab. 13: Beschreibung Functional Requirement 10

ID	**FR11**
Name	Rangliste und Punktesystem
Beschreibung	• Es muss eine Rangliste geben, welche die aktivsten Mitglieder anzeigt. • Mitglieder müssen ihre aktuellen Punkte einsehen können.

Tab. 14: Beschreibung Functional Requirement 11

ID	FR12
Name	Belohnungen für Punkte (z. B. Vereins-Merch)
Beschreibung	• Mitglieder müssen ihre gesammelten Punkte für ausgeschriebene Belohnungen einlösen können. • Der Discord-Bot sollte mit dem Shop-System über eine API oder Webhooks kommunizieren können. (Optional)

Tab. 15: Beschreibung Functional Requirement 12

2.4.5 Admin-Dashboard (optional)

ID	FR13
Name	Verwaltung von Mitgliedern, Rollen sowie Events
Beschreibung	Ein Web-Interface (Admin-Dashboard) muss bereitgestellt werden, in dem Vorstandsmitglieder folgende Funktionen zur Verfügung haben: • Mitglieder und deren Rollen verwalten • Events inklusive Erinnerungen erstellen, bearbeiten und löschen • Gamification-Daten einsehen

Tab. 16: Beschreibung Functional Requirement 13

ID	FR14
Name	Steuerung der Bot-Funktionen
Beschreibung	• Das Admin-Dashboard muss ermöglichen, die Zugriffsrechte auf Befehle zu bearbeiten (welche Rollen welche Befehle nutzen dürfen). • Das Admin-Dashboard muss ermöglichen, automatisierte Nachrichten zu konfigurieren.

Tab. 17: Beschreibung Functional Requirement 14

ID	FR15
Name	Gamification-Datenverwaltung
Beschreibung	• Das Admin-Dashboard muss ermöglichen, detaillierte Gamification-Statistiken und Mitgliedsaktivitäten einzusehen und zu analysieren.

Tab. 18: Beschreibung Functional Requirement 15

2.4.6 Skalierbarkeit und Automatisierung

ID	FR16
Name	Webhooks und API-Integration
Beschreibung	• Der Discord-Bot muss über Webhooks und APIs mit externen Systemen kommunizieren: ‣ Google Calendar (Event-Benachrichtigungen) ‣ Vereinswebseite / Shop (Punkte-Updates, Transaktionen) (optional)

Tab. 19: Beschreibung Functional Requirement 16

ID	FR17
Name	Automatisches Deployment
Beschreibung	• Der Discord-Bot muss mit Docker sowie CI/CD automatisch bereitgestellt und aktualisiert werden.

Tab. 20: Beschreibung Functional Requirement 17

2.4.7 Testing und Qualitätssicherung

ID	**FR18**
Name	End-to-End-Tests für Discord-Befehle
Beschreibung	• Es müssen e2e-Tests durchgeführt werden, um die Discord-Bot-Funktionalität zu überprüfen.

Tab. 21: Beschreibung Functional Requirement 18

ID	**FR19**
Name	Unit-Tests für Bot-Logik
Beschreibung	• Der Code muss mit Mocking für discord.js getestet werden, um Fehler frühzeitig zu erkennen.

Tab. 22: Beschreibung Functional Requirement 19

2.5 Use Cases

Die Use Cases beschreiben detailliert die Interaktionen zwischen Nutzern und dem Discord-Bot und zeigen, wie die Functional Requirements umgesetzt werden. Sie definieren zentrale Abläufe für die Automatisierung von Vereinsfunktionen wie Mitgliederverwaltung, Event-Ankündigungen und Gamification. Darüber hinaus unterstützen sie die Systemarchitektur, indem sie Anforderungen an Sicherheit, Skalierbarkeit und Benutzerfreundlichkeit konkretisieren.

ID	**UC1**
Name	Mitglied Bewerbung und Rollenvergabe (FR1 , FR6 , FR7)
Akteur	Nicht-Mitglied, Discord-Bot, Vorstand
Beschreibung	Ein Nicht-Mitglied kann sich über ein Bewerbungsformular für den Verein bewerben. Nach der manuellen Prüfung durch den Vorstand vergibt der Bot die entsprechende Rolle.
Voraussetzung	Das Nicht-Mitglied hat Zugriff auf das Bewerbungsformular.
Standard-Verlauf	1. Das Nicht-Mitglied füllt das Bewerbungsformular aus 2. Nach einer expliziten Zustimmung der Datennutzungs-Bedingungen wird das Formular abgesendet. (UC7) 3. Der Discord-Bot speichert die Daten in der Vereinsdatenbank und informiert den Vorstand. 4. Der Vorstand prüft die Bewerbung und setzt den Status des Mitglieds, wenn das ausführende Vorstandsmitglied die nötigen Zugriffsrechte auf diesen Befehl hat.
Alternativ-Verlauf	• Falls das Bewerbungsformular unvollständig oder fehlerhaft ist, gibt der Bot eine entsprechende Fehlermeldung aus und fordert eine Korrektur an. • Falls die Bewerbung abgelehnt wird, erhält das Nicht-Mitglied eine Benachrichtigung. • Falls das ausführende Vorstandsmitglied nicht die nötigen Zugriffsrechte auf diesen Befehl hat, wird eine Fehlermeldung ausgegeben.
Nachbedingung	Das Mitglied ist in der Vereinsdatenbank registriert und hat eine zugewiesene Rolle oder wurde abgelehnt.
Priorität	Hoch
Resultat	Ein Nicht-Mitglied kann den Bewerbungsprozess über den `/join`-Befehl starten. Das Nicht-Mitglied kann seine Daten im Bewerbungsformular angeben und muss am Ende noch die Datenschutzrichtlinien annehmen. Zudem werden Zahlungsinformationen über den Mitgliederbeitrag angezeigt. Der Vorstand wird im Bewerbungskanal (welcher nur für Vorstandsmitglieder sichtbar ist) über die Bewerbung informiert und kann diese nach der manuellen überprüfung annehmen oder ablehnen. In beiden Fällen wird das Nicht-Mitglied mit einer Nachricht informiert. Die Annahme führt zur Mitgliedschaft im Verein.

Tab. 23: Beschreibung Fully dressed Use Case 1

ID	UC2
Name	Statusänderung und automatische Rollenverwaltung (FR2)
Akteur	Mitglied, Vorstand, Discord-Bot
Beschreibung	Änderungen im Mitgliedsstatus führen zu einer automatischen Rollenaktualisierung im Discord-Server.
Voraussetzung	Das Mitglied ist in der Vereinsdatenbank registriert.
Standard-Verlauf	1. Der Status eines Mitglieds ändert sich (z. B. durch Zahlung des Mitgliedbeitrages oder Annahme einer Bewerbung). 2. Der Discord-Bot reagiert auf die Statusänderung und vergibt automatisch die entsprechende Rolle.
Alternativ-Verlauf	• Falls das Mitglied bereits die korrekte Rolle hat, wird keine Statusänderung durchgeführt. • Falls erforderlich, kann der Vorstand Rollen manuell über den Bot anpassen.
Nachbedingung	Der Mitgliedsstatus ist aktuell, und die zugewiesene Rolle entspricht dem Status.
Priorität	Hoch
Resultat	Einem Nicht-Mitglied wird durch Annahme einer Bewerbung die Rolle Mitglied zugewiesen. Ändert ein Vorstandsmitglied die Rolle eines Mitglieds im Discord-Server oder in der Datenbank kann diese Änderung mit einem Befehl synchronisiert werden.

Tab. 24: Beschreibung Fully dressed Use Case 2

ID	UC3
Name	Mitgliedsbeitragserinnerung und automatische Entfernung (FR3 , FR13)
Akteur	Discord-Bot, Mitglied, Vorstand
Beschreibung	Der Bot erinnert Mitglieder an ausstehende Zahlungen und kann sie bei Nichtzahlung entfernen.
Voraussetzung	Ein Mitglied hat eine ausstehende Mitgliedsgebühr.
Standard-Verlauf	1. Der Discord-Bot prüft am Fälligkeitsdatum, ob ein Mitglied den Beitrag bezahlt hat. 2. Falls nicht, sendet der Bot Erinnerungsnachrichten. 3. Nach Fristablauf benachrichtigt der Bot den Vorstand, der das Mitglied über den Bot aus dem Verein entfernen kann.
Alternativ-Verlauf	Falls das Mitglied zahlt, werden keine weiteren Erinnerungen gesendet.
Nachbedingung	Mitglieder ohne Zahlung werden aus dem Server entfernt.
Priorität	Mittel
Resultat	Nicht-Mitglieder werden beim Beitritt in den Verein aufgefordert die Mitgliedsgebühr zu bezahlen. Eine Jährliche Zahlungserinnerung mit automatischer Entfernung des Mitglieds aus dem Server wurde nicht umgesetzt, jedoch ist es möglich die Zahlungfrist als Event aufzunehmen und so die Mitglieder darüber zu informieren.

Tab. 25: Beschreibung Fully dressed Use Case 3

ID	UC4
Name	Eventverwaltung und Ankündigung (FR4)
Akteur	Vorstand, Discord-Bot
Beschreibung	Events werden über den Bot verwaltet und angekündigt.
Voraussetzung	Der Vereinskalender ist mit dem Bot verbunden.
Standard-Verlauf	1. Der Vorstand erstellt oder bearbeitet ein Event im Google Calendar oder Admin-Dashboard. 2. Der Discord-Bot ruft die Event-Daten ab und postet eine Ankündigung im Event-Kanal. 3. Mitglieder können sich für Events anmelden, indem sie auf den „Interessiert“-Button klicken.
Alternativ-Verlauf	Falls ein Event abgesagt wird, synchronisiert der Bot den Kalender und informiert somit die Teilnehmer.
Nachbedingung	Event-Informationen sind aktuell im Discord-Kanal.
Priorität	Hoch
Resultat	Der Vorstand kann Events direkt im Discord-Server oder im Google Calendar erstellen und bearbeiten. Events die auf Discord erstellt oder bearbeitet werden, werden direkt im Google Calendar aktualisiert. Änderungen die im Google Calendar durchgeführt werden, werden periodisch (aktuell alle 10 Minuten) oder manuell durch einen Befehl synchronisiert. Mitglieder können sich im Discord mithilfe des „Interessiert“-Buttons auf ein Event anmelden. Der Bot informiert die Teilnehmer bei einer Event-Absage über eine automatische Aktualisierung der Event-Übersicht im entsprechenden Kanal, in den Discord-Events sowie im Google Calendar.

Tab. 26: Beschreibung Fully dressed Use Case 4

ID	UC5
Name	Event-Erinnerungen senden (FR5)
Akteur	Discord-Bot, Mitglieder
Beschreibung	Der Bot sendet automatische Erinnerungen für geplante Events.
Voraussetzung	Ein Event ist angekündigt.
Standard-Verlauf	1. Der Bot prüft, wer sich für ein Event interessiert hat. 2. Erinnerungen werden zu festgelegten Zeitpunkten im Event-Kanal versendet, wobei die interessierten Teilnehmer markiert werden.
Alternativ-Verlauf	Falls es derzeit keine Events gibt, wird dementsprechend keine Handlung vorgenommen.
Nachbedingung	Teilnehmer haben rechtzeitig Event-Erinnerungen erhalten.
Priorität	Hoch
Resultat	Der Bot prüft periodisch (aktuell alle 10 Minuten) ob ein Event innerhalb der nächsten Stunde beginnt. Ist dies der Fall, werden angemeldete Nutzer im Event-Kanal mit Markierungen darüber informiert.

Tab. 27: Beschreibung Fully dressed Use Case 5

ID	UC6
Name	Opt-in Zustimmung zur Datenspeicherung (FR7)
Akteur	Nicht-Mitglied, Discord-Bot
Beschreibung	Bevor Daten gespeichert werden, muss das Nicht-Mitglied der Speicherung zustimmen.
Voraussetzung	Das Nicht-Mitglied tritt dem Discord-Server bei oder bewirbt sich als Mitglied.
Standard-Verlauf	1. Beim Beitritt wird auf die Datenschutzerklärung hingewiesen. 2. Erst nach Zustimmung werden Daten gespeichert und Rollen vergeben.
Alternativ-Verlauf	Falls keine Zustimmung erfolgt, wird die Bewerbung automatisch abgelehnt und es können auf dem Server keine Punkte gesammelt werden.
Nachbedingung	Daten werden nur mit Zustimmung gespeichert.
Priorität	Hoch
Resultat	Beim Beitritt auf den Discord-Server werden Nutzer auf die Datenschutzerklärung hingewiesen. Nutzerdaten werden erst bei der Bewerbung gespeichert, bei welcher die Nutzer die Datenschutzrichtlinien annehmen müssen. Stimmen die Nutzer der Datenschutzrichtlinien nicht zu wird die Bewerbung abgelehnt.

Tab. 28: Beschreibung Fully dressed Use Case 6

ID	UC7
Name	Einsicht und Verwaltung gespeicherter Daten (FR8)
Akteur	Mitglied, Discord-Bot
Beschreibung	Mitglieder können ihre gespeicherten Daten einsehen und durch einen Vereins-Austritt die Daten löschen lassen.
Voraussetzung	Das Mitglied ist in der Vereinsdatenbank gespeichert.
Standard-Verlauf	1. Das Mitglied ruft seine gespeicherten Daten per Befehl ab. 2. Es kann seine aktuell in der Datenbank gespeicherten Daten einsehen.
Alternativ-Verlauf	1. Das Mitglied will seine Daten Löschen und tritt aus dem Verein aus. 2. Es wird benachrichtigt über das Lösch-Datum. 3. Es kann innerhalb der Bedenkfrist wieder in den Verein eintreten und behält die gespeicherten Daten.
Nachbedingung	Mitgliederdaten sind abrufbar und verwaltbar.
Priorität	Hoch
Resultat	Ein Mitglied kann mit dem `/getdata`-Befehl alle über über dessen Person gespeicherte Daten einsehen. Um die Daten zu löschen kann das Mitglied mit dem `/leave`-Befehl den Verein verlassen. Die Daten werden nach einer definierten Frist automatisch gelöscht.

Tab. 29: Beschreibung Fully dressed Use Case 7

ID	UC8
Name	Punktesystem und Rangliste (FR9 , FR10 , FR11)
Akteur	Mitglied, Discord-Bot
Beschreibung	Mitglieder sammeln Punkte für Aktivitäten und der Bot verwaltet eine Rangliste.
Voraussetzung	Das Gamification-System ist aktiviert.
Standard-Verlauf	1. Mitglieder erhalten Punkte für Aktivitäten (z. B. Event-Teilnahme, Spenden). 2. Der Discord-Bot speichert den Punktestand und zeigt diesen auf Anfrage an. 3. Mitglieder können die Rangliste einsehen.
Alternativ-Verlauf	Falls das Punktesystem deaktiviert ist, werden keine Punkte gesammelt.
Nachbedingung	Punkte werden korrekt verwaltet und angezeigt.
Priorität	Hoch
Resultat	Mitglieder können Aufgaben annehmen und bekommen für diese nach einer Überprüfung eines Vorstandsmitgliedes Punkte. Der Punktestand kann mit dem `/score`- oder auch mit dem `/getdata`-Befehl eingesehen werden. Im „ranglisten“-Kanal sind alle Ranglisten ersichtlich.

Tab. 30: Beschreibung Fully dressed Use Case 8

ID	UC9
Name	Manipulationsschutz im Gamification-System (FR9 , FR10)
Akteur	Mitglied, Discord-Bot, Vorstand
Beschreibung	Der Vorstand ist bei jeder Punkte-Transaktion involviert und kontrolliert durch manuelle freigabe die Punktevergabe. Die Gamification-Daten werden nachvollziehbar dokumentiert und können jederzeit durch den Vorstand eingesehen werden, um Missbrauch zu verhindern.
Voraussetzung	Das Gamification-System ist aktiv.
Standard-Verlauf	1. Mitglieder sammeln Punkte durch verschiedene Aktivitäten. 2. Der Vorstand bestätigt die Puntkevergabe und kann diese einsehen. 3. Falls eine Manipulation erkannt wird, reagiert der Vorstand entsprechend der Situation und kann die Punktebewegungen nachvollziehen, um den Grund dafür herauszufinden.
Alternativ-Verlauf	Falls keine Manipulation festgestellt wird, wird keine Handlung vorgenommen.
Nachbedingung	Die Gamification bleibt fair, und das System verhindert Missbrauch.
Priorität	Hoch
Resultat	Jede durchgeführte Aktivität muss durch ein Vorstandsmitglied validiert werden, bevor Punkte dafür ausgeschrieben werden. Dies geschieht in einem nur für Vorstandsmitglieder sichtbaren Aufgaben-Kanal. Somit haben alle Vorstandsmitglieder einen Überblick über die verteilten Punkte und können Manipulationsversuche erkennen. Zudem sind alle Punkte-Bewegungen als Transaktion in der Datenbank gespeichert und somit nachvollziehbar.

Tab. 31: Beschreibung Fully dressed Use Case 9

ID	UC10
Name	Punkte für Belohnungen einlösen (FR12)
Akteur	Mitglied, Discord-Bot, Vorstand, Vereins-Shop-System (optional)
Beschreibung	Mitglieder können gesammelte Punkte gegen Belohnungen, welche der Vorstand definiert, einlösen.
Voraussetzung	Das Mitglied hat genügend Punkte.
Standard-Verlauf	1. Das Mitglied wählt über den Bot eine passende Belohnung aus. 2. Die Anfrage wird vom Vorstand (oder automatisch per Webhook im Vereins-Shop) bearbeitet. 3. Das Mitglied bekommt die Belohnung zugestellt. 4. Das Vorstandsmitglied (oder der Vereins-Shop) bestätigt die Übermittlung.
Alternativ-Verlauf	• Falls die Datenbank nicht erreichbar ist, gibt der Bot eine Fehlermeldung aus. • Falls das optionale Shop-System nicht erreichbar ist, erhält das Mitglied eine Fehlermeldung.
Nachbedingung	Der Punktestand wird entsprechend der Einlösung aktualisiert.
Priorität	Mittel
Resultat	Mitglieder können über den Discord-Bot ihre Punkte für Belohnungen eintauschen. Belohnungen müssen von einem Vorstandsmitglied bearbeitet werden. Das Mitglied erhält eine Benachrichtigung sobald die Belohnung bearbeitet und zugestellt wurde.

Tab. 32: Beschreibung Fully dressed Use Case 10

ID	UC11 (optional)
Name	Verwaltung von Mitgliedern und Bot-Funktionen im Admin-Dashboard (FR13 , FR14 , FR15)
Akteur	Vorstand
Beschreibung	Der Vorstand kann Mitglieder, Rollen und die Command-Whitelist über ein Admin-Dashboard verwalten.
Voraussetzung	Das Admin-Dashboard ist eingerichtet und mit der Vereinsdatenbank verbunden.
Standard-Verlauf	1. Der Vorstand kann über das Admin-Dashboard Mitglieder, Rollen, Events, und die Command-Whitelist verwalten. 2. Änderungen werden in die Datenbank übernommen und vom Bot automatisch verarbeitet.
Alternativ-Verlauf	Falls falsche Eingaben gemacht werden, zeigt das Dashboard eine Fehlermeldung.
Nachbedingung	Änderungen sind korrekt gespeichert und umgesetzt.
Priorität	Niedrig
Resultat	Das Admin-Dashboard wurde nicht umgesetzt.

Tab. 33: Beschreibung Fully dressed Use Case 11

ID	UC12
Name	Webhooks und API-Integration (FR16)
Akteur	Discord-Bot, Google Calendar, Vereinswebseite (optional), Vereins-Shop-System (optional)
Beschreibung	Der Discord-Bot kommuniziert über Webhooks und APIs mit externen Systemen, um Benachrichtigungen, Kalender-Updates und Punkteverwaltung durchzuführen.
Voraussetzung	Externe Systeme bieten eine API / Webhook-Schnittstelle.
Standard-Verlauf	1. Ein Event oder eine Aktion (z. B. ein neues Kalenderevent) wird in einem externen System erstellt oder aktualisiert. 2. Der Discord-Bot empfängt die API / Webhook-Nachricht und verarbeitet die Informationen entsprechend. 3. Der Bot aktualisiert den Status im Discord-Server.
Alternativ-Verlauf	• Falls das externe System nicht erreichbar ist, werden die Daten zwischengespeichert und später erneut gesendet. • Falls das API-Format fehlerhaft ist, gibt der Bot eine Fehlernachricht aus.
Nachbedingung	Die Informationen aus externen Systemen wurden korrekt im Discord-Bot verarbeitet.
Priorität	Hoch
Resultat	Events von Google Calendar werden periodisch über HTTP-Requests oder manuell per Befehl abgefragt und im Discord-Server aktualisiert. Eine Anbindung zur Vereins-Webseite oder Vereins-Shop-System wurde nicht umgesetzt.

Tab. 34: Beschreibung Fully dressed Use Case 12

ID	UC13
Name	Automatisches Deployment (FR17)
Akteur	Entwickler, GitLab CI/CD-System
Beschreibung	Änderungen am Code werden automatisch in der Produktivumgebung ausgerollt, um manuelle Fehler zu vermeiden.
Voraussetzung	Ein funktionierendes CI/CD-Setup mit Docker ist vorhanden.
Standard-Verlauf	1. Ein Entwickler pusht Änderungen ins Repository. 2. Der CI/CD-Runner startet die Build- und Test-Pipeline. 3. Falls alle Tests erfolgreich sind, wird das neue Image erstellt und das Deployment ausgelöst.
Alternativ-Verlauf	Falls ein Test fehlschlägt, wird das Deployment abgebrochen und eine Benachrichtigung an das Entwicklerteam gesendet.
Nachbedingung	Der Bot ist in der neuesten Version online.
Priorität	Mittel
Resultat	Eine Deployment Pipeline wurde nicht umgesetzt. Das Deployment findet manuell über einen Docker Container statt.

Tab. 35: Beschreibung Fully dressed Use Case 13

ID	UC14
Name	End-to-End-Tests für Discord-Befehle (FR18)
Akteur	Testsystem, Discord-Bot
Beschreibung	Automatische Tests überprüfen die vollständige Funktionalität der Bot-Befehle.
Voraussetzung	Eine Testumgebung für den Discord-Bot ist eingerichtet.
Standard-Verlauf	1. Der Testsystem-Runner sendet verschiedene Befehle an den Bot. 2. Der Bot reagiert und sendet entsprechende Nachrichten zurück. 3. Die Antworten werden mit den erwarteten Ergebnissen verglichen.
Alternativ-Verlauf	Falls ein Test fehlschlägt, wird ein Fehlerbericht generiert.
Nachbedingung	Alle getesteten Befehle funktionieren wie vorgesehen.
Priorität	Hoch
Resultat	Die End-to-End-Tests wurden manuell durchgeführt. Alle implementierten Befehle funktionieren korrekt.

Tab. 36: Beschreibung Fully dressed Use Case 14

ID	UC15
Name	Unit-Tests für Bot-Logik (FR19)
Akteur	Entwickler, Unit-Test-Framework
Beschreibung	Einzelne Code-Komponenten werden isoliert getestet, um Fehler frühzeitig zu erkennen.
Voraussetzung	Der Code ist modular genug, um isoliert getestet zu werden.
Standard-Verlauf	1. Eine Code-Änderung wird vorgenommen. 2. Das Unit-Test-Framework führt Tests gegen Mock-Daten durch. 3. Falls alle Tests erfolgreich sind, wird die Änderung übernommen.
Alternativ-Verlauf	Falls ein Test fehlschlägt, wird eine Fehlermeldung zurückgegeben und das Problem muss behoben werden.
Nachbedingung	Die getesteten Code-Komponenten funktionieren korrekt.
Priorität	Mittel
Resultat	Die Unit-Tests fokussieren sich darauf, dass es bei Befehlen zu keinem Absturz des Bots kommt. Die Tests müssen manuell gestartet werden. Für den Test-Report der Unit-Tests siehe Abschnitt 13 .

Tab. 37: Beschreibung Fully dressed Use Case 15

2.6 Non-Functional Requirements

Die Non-Functional Requirements definieren die Qualitätsmerkmale des Discord-Bots, darunter Sicherheit, Performance, Skalierbarkeit und Usability. Sie beeinflussen die Architektur- und Designentscheidungen, um eine zuverlässige und effiziente Nutzung des Bots sicherzustellen. Zudem stellen sie sicher, dass das System auch bei steigender Nutzung robust und erweiterbar bleibt.

ID	NFR1
Beschreibung	Die Code Qualität entspricht dem im Linter definierten Anforderungen.
Anforderungen	Maintainability - Analyzability
Priorität	Hoch
Messung	Es wird manuell überprüft ob der Linter Fehlermeldungen ausgibt.
Testen	Nach jedem Commit
Resultat	Der Linter gibt keine Fehlermeldungen aus.

Tab. 38: Beschreibung Non-Functional Requirement 1

ID	NFR2
Beschreibung	Die Testabdeckung der Applikation beträgt mindestens 80%.
Anforderungen	Maintainability - Testability
Priorität	Hoch
Messung	Test Coverage wird mittels des verwendeten Testing-Frameworks bestimmt.
Testen	Am Ende
Resultat	Bei den Tests wurden vor allem die Discord-Befehle priorisiert. Eine Testabdeckung der gesamten Applikation von 80% konnte nicht erreicht werden.

File	% Stmts	% Branch	% Funcs	% Lines
All files	40.14	22.98	19.82	41.72

Abb. 4: Testabdeckung aller Dateien

src/platform/discord/commands	64.42	50	48.78	65.03
donationModal.ts	28.57	16.66	25	29.16
events.ts	100	100	100	100
getdata.ts	74.35	51.02	14.28	74.35
help.ts	90.9	100	50	90.9
intro.ts	100	100	100	100
leave.ts	50	35.71	40	50
ping.ts	100	100	100	100
syncroles.ts	72.91	56.09	70	74.44
syncusers.ts	65.51	48.27	50	66.26
src/platform/discord/commands/calendar	93.75	75	100	93.75
linkcalendar.ts	85.71	50	100	85.71
synccalendar.ts	100	100	100	100
src/platform/discord/commands/join	98.11	91.3	75	98.11
join.ts	98.11	91.3	75	98.11
src/platform/discord/commands/leaderboard	55.55	9.09	66.66	55.55
createLeaderboard.ts	44.82	0	50	44.82
score.ts	100	100	100	100
src/platform/discord/commands/task	95.65	75	100	98.5
claimTask.ts	85.71	40	100	94.73
completetask.ts	100	100	100	100
createTaskModal.ts	100	100	100	100

Abb. 5: Testabdeckung der Discord-Befehle

Tab. 39: Beschreibung Non-Functional Requirement 2

ID	NFR3
Beschreibung	Fehler werden gefangen und entsprechend behandelt. Es soll nicht zum Absturz des Bots kommen.
Anforderungen	Reliability - Fault tolerance
Priorität	Hoch
Messung	Es werden Usability-Tests durchgeführt, bei denen die Tester gezielt aufgefordert werden, den Bot zum Absturz zu bringen.
Testen	Am Ende
Resultat	Fehler werden implizit behandelt. Bei den Usability-Tests kam es zu keinem Absturz des Bots.

Tab. 40: Beschreibung Non-Functional Requirement 3

ID	NFR4
Beschreibung	Log-Dateien mit Fehler- und Erfolgsmeldungen werden erfasst, damit allfällige Fehler besser gefunden werden können.
Anforderungen	Maintainability - Analyzability
Priorität	Niedrig
Messung	Manuelle Tests
Testen	Jede Woche
Resultat	Log-Dateien wurden nicht umgesetzt. Fehler- und Erfolgsmeldungen werden im Discord-Server angezeigt.

Tab. 41: Beschreibung Non-Functional Requirement 4

ID	NFR5 (optional)
Beschreibung	Das Design des Admin-Dashboards ist ausgelegt auf Desktop-Systeme. Es werden etablierte Design-Praktiken verwendet um die Usability des Dashboards zu gewährleisten.
Anforderungen	Compatibility - Interoperability
Priorität	Mittel
Messung	Es werden Usability-Tests durchgeführt.
Testen	Am Ende
Resultat	Das Admin-Dashboard wurde nicht umgesetzt.

Tab. 42: Beschreibung Non-Functional Requirement 5

ID	NFR6 (optional)
Beschreibung	Das Admin-Dashboard kann auf den Browsern Google Chrome und Firefox verwendet werden.
Anforderungen	Compatibility - Interoperability
Priorität	Mittel
Messung	Es werden Usability-Tests durchgeführt.
Testen	Am Ende
Resultat	Das Admin-Dashboard wurde nicht umgesetzt.

Tab. 43: Beschreibung Non-Functional Requirement 6

ID	NFR7
Beschreibung	Der Datenschutz der Nutzer des Discord-Bots ist gewährleistet.
Anforderungen	Security
Priorität	Hoch
Messung	Es wird ein Sicherheitskonzept erstellt, welche als Grundlage für die Implementation verwendet wird. Zudem werden Usability-Tests durchgeführt.
Testen	Am Ende
Resultat	Die Daten werden sicher auf dem Vereinsserver gespeichert. Nutzer können ihre gespeicherten Daten einsehen. Backups der Daten werden regelmässig erstellt und verschlüsselt (siehe Abschnitt 3.6).

Tab. 44: Beschreibung Non-Functional Requirement 7

ID	NFR8	
Beschreibung	Das Gamification-System ist gegen Manipulation geschützt.	
Anforderungen	Security	
Priorität	Hoch	
Messung	Es wird ein Gamification-Konzept erstellt, welches Schritte zur Erkennung und Verhinderung von Manipulation enthält. Dieses Konzept wird als Grundlage für die Implementation verwendet. Zudem werden Usability-Tests durchgeführt.	
Testen	Am Ende	
Resultat	Punkte können nur durch Vorstandmitglieder verteilt werden. Alle Vorstandmitglieder können die Punktevergabe einsehen. Dadruch können Manipulationsversuche erkannt und Gegenmass-nahmen bestimmt werden.	

Tab. 45: Beschreibung Non-Functional Requirement 8

ID	NFR9	
Beschreibung	Der Discord-Bot ist intuitiv bedienbar und die verschiedenen Funktionen leicht verständlich.	
Anforderungen	Usability	
Priorität	Hoch	
Messung	Es wird ein Usability-Test durchgeführt.	
Testen	Während der Konstruktionsphase	
Resultat	Der Usability-Test war insgesamt erfolgreich. Genauere Informationen können im Anhang gefunden werden (siehe Abschnitt Testreports).	

Tab. 46: Beschreibung Non-Functional Requirement 9

3 Sicherheitskonzept

Das Sicherheitskonzept beschreibt, wie das System gegen Manipulation und Datenlecks geschützt wird. Es definiert bestimmte Technologien und Vorgehensweisen die verwendet werden, um den Schutz der Daten zu gewährleisten.

3.1 Ziele

Bevor das Sicherheitskonzept ausgearbeitet wird, müssen Ziele definiert werden welche durch das Konzept erfüllt werden. Diese Ziele lauten wie folgt:

1. Die Speicherung von so wenig persönlicher Nutzerdaten wie nötig.
2. Die Privatsphäre der Nutzer hat eine hohe Priorität. Die Daten werden DSGVO-konform gespeichert und Nutzer können ihre Daten einsehen sowie löschen.
3. Die Datenspeicherung muss transparent sein. In der Nutzungsvereinbarung muss klar ersichtlich sein, aus welchen Gründen bestimmte Daten gespeichert werden.

Wie diese Ziele gemessen und erreicht werden wird im Folgenden definiert.

3.2 Sicherheit und Rechteverwaltung

Um die Sicherheit des Systems und den Schutz der Nutzerdaten zu gewährleisten, werden die nachfolgenden Methoden implementiert.

3.2.1 OAuth2-Authentifizierung

Discord erlaubt es durch OAuth2-Authentifizierung, auf bestimmte Daten der Discord API zuzugreifen. Anwendungen können durch explizite Zustimmung des Mitglieds geschützte Informationen, wie dessen Freundesliste, einsehen. Der Zugriff auf solche Nutzerdaten ist im aktuellen System jedoch nicht vorgesehen und wird daher auch nicht eingesetzt. (Discord-Contributors, 2025)

3.2.2 Role-Based Access Control

Mit Role-Based Access Control (RBAC) kann der Zugriff auf Daten basierend auf den Nutzerrollen kontrolliert werden. Dabei gibt es folgende Rollen:

- **Vorstandsmitglied**: Vorstandsmitglieder haben vollen Zugriff auf alle Daten, sofern sie über Zugangsdaten zur Datenbank verfügen. Logs der Aktivitätsdaten sind in Form von Benachrichtigungen in den jeweiligen Discord-Kanälen gegeben.
- **Mitglied**: Mitglieder haben Einsicht auf Ihre eigenen Daten.
- **Nicht-Mitglied**: Nicht-Mitglieder haben keinen Zugriff auf irgendwelche Daten, bis auf die öffentlich zugänglichen Ranglisten, Events und ausgeschriebenen Aufgaben.

3.3 Datenschutz und DSGVO

Da der Bot personenbezogene Daten verarbeitet (z.B. Discord-IDs, Mitgliederdaten, Event-Teilnahmen), muss der Datenschutz, insbesondere im Kontext der europäischen Datenschutz-Grundverordnung DSGVO[8] und des schweizerischen Datenschutzgesetzes DSG[9] sichergestellt werden.

Folgende Daten werden erfasst und verarbeitet:

- Discord-IDs
- Mitgliedsrollen
- Status und Rechte
- Gamification-Daten
- Nutzungsprofile
- Event-Teilnahmen

[8] https://dsgvo-gesetz.de/
[9] https://www.fedlex.admin.ch/eli/cc/2022/491/de

- Aktivitätsdaten

Jede Verknüpfung der Discord-IDs mit anderen Daten muss datenschutzkonform behandelt werden, da die IDs auf spezifische Benutzer zurückzuführen sind. Alle gespeicherten Daten sind für das korrekte Funktionieren des Systems notwendig.

3.4 Massnahmen

Um die relevanten Datenschutzverordnungen einzuhalten und die Sicherheit und Privatsphäre der Nutzer zu gewährleisten müssen folgende Massnahmen umgesetzt werden:

1. **Einwilligung und Transparenz (Opt-in):** (Use Case 6)
   - Mitglieder müssen der Datenspeicherung aktiv zustimmen.
   - Klare Hinweise und Regeln müssen im Discord-Server in einem Datenschutz-Kanal ersichtlich sein.
2. **Datenminimierung und Zweckbindung:**
   - Nur notwendige Daten werden gespeichert.
   - Daten, die nach einer gewissen Zeit nicht mehr erforderlich sind, werden gelöscht.
3. **Datenzugriffsrechte und Löschung:**
   - Mitglieder können die Löschung ihrer Daten anfordern. Dies führt jedoch zum Vereins-Austritt, weil die Daten zwingend nötig sind für normale Vereinstätigkeiten.
   - Die Daten werden nach dem Vereins-Austritt sowie nach drei Monaten Bedenkfrist automatisch gelöscht.
   - Mitglieder können auf ihre Daten einsehen. (Use Case 7)

3.5 Datenspeicherung

Ein wichtiger Punkt in der Sicherheit der Daten, ist die Frage, wo diese gespeichert werden. Hier gibt es zwei Möglichkeiten:

1. Self-Hosting
2. Hosting auf einem Cloud-Service (Amazon Web Services, Google Cloud Platform, usw.)

Da die Sicherheit der Daten und die Privatsphäre der Nutzer hohe Priorität hat, kommt die zweite Möglichkeit, Hosting auf Cloud-Services, nicht infrage. Viele der grossen Cloud-Service-Anbieter sind aus den USA, welche durch den CLOUD Act[10] verpflichtet sind, den US-Behörden Zugriff auf im Internet gespeicherte Daten zu geben. Dies ist ebenfalls der Fall, wenn die Speicherung der Daten nicht in den USA erfolgt.

Um die Kontrolle über die gespeicherten Daten zu gewährleisten, ist daher Self-Hosting die beste Wahl. Weitere Begründungen für die Wahl des Hostings werden im Abschnitt 11.11 beschrieben.

Um die Privatsphäre der Nutzer sicherzustellen dürfen bestimme Arten von Daten unter keinen Umständen gespeichert werden.

- Passwörter
- Private Nachrichten
- Discord-Chat-Verläufe
- IP-Adressen

Allgemein muss den Nutzern klar dargestellt werden welche Daten aus welchen Gründen gesammelt werden. Daten die nicht benötigt werden, dürfen nicht gespeichert werden. Jegliche Änderungen müssen klar und ersichtlich kommuniziert werden, damit die Nutzer die Möglichkeit haben zu reagieren. Im Anhang ist ein Beispiel für eine valide Umsetzung der Datenschutzrichtlinien des Discord-Bots zu finden (vgl. Anhang „Datenschutzrichtlinie-Beispiel.pdf“). Dieser wird im Regel-Kanal des Bots öffentlich zugänglich gemacht.

3.6 Sicherheit des Servers

Um die Sicherheit der Daten auf dem Server zu gewährleisten, muss sichergestellt werden, dass nur autorisierte Personen physischen sowie auch digitalen Zugriff auf den Server haben.

[10] https://www.congress.gov/bill/115th-congress/house-bill/4943

3.6.1 Direkter Datenbank-Zugriff

Ein sicherer Zugriff auf die Datenbank von ausserhalb (z. B. über MongoDB Compass) ist nur für autorisierte Vorstandsmitglieder vorgesehen. Für die Authentifizierung wird aktuell ein Passwortschutz eingesetzt. Weitere Sicherheitsmechanismen, wie beispielsweise Zwei-Faktor-Authentifizierung , sind in der aktuellen Version nicht implementiert, sollten aber in zukünftigen Versionen berücksichtigt werden. Eine detaillierte Beschreibung zur Umsetzung des externen Datenbank-Zugriffs ist im Anhang zu finden (siehe Abschnitt 12).

3.6.2 Backups

Um Datenverluste zu verhindern, sollen regelmässige Backups der Datenbank durchgeführt werden. Diese Backups müssen auf separaten Servern oder externen Speichermedien gespeichert werden. Um unbefugten Zugriff zu verhindern, müssen die Backups zusätzlich verschlüsselt werden. Neben der technischen Umsetzung stellt sich die Frage nach Verantwortlichkeiten (z. B. Vorstand oder Server-Administrator) sowie nach einer klar definierten Backup-Strategie.

Die Backup-Strategie sieht vor, dass automatisierte, tägliche Backups erstellt werden, die eine revisionssichere Wiederherstellung ermöglichen. Zusätzlich sollen jederzeit manuelle Backups möglich sein, beispielsweise vor grösseren Updates oder Wartungsarbeiten. Die Aufbewahrungsdauer der Backups soll so gewählt werden, dass ein ausreichender Zeitraum für Rollbacks oder Datenwiederherstellungen besteht, ohne jedoch unnötig Speicherplatz zu belegen. Auch die Einhaltung von Datenschutzbestimmungen (DSGVO) ist zu berücksichtigen, insbesondere durch die Verwendung von Verschlüsselung und die sichere Speicherung der Backups.

Die technische Umsetzung dieses Backup-Systems ist im Kapitel Abschnitt 7.2 beschrieben.

3.7 Schulung

Da Vorstandsmitglieder Zugriff auf sensible Daten haben, ist es wichtig sie entsprechend zu schulen. Die Schulung kann folgende Punkte umfassen:

1. Datenschutz und rechtliche Grundlagen
   - Grundlegende Einführung in die DSGVO.
   - Welche Daten dürfen erfasst werden?
   - Welche Rechte haben Mitglieder im Bezug auf ihre gespeicherten Daten?
2. Sichere Datenverarbeitung und Datenspeicherung
   - Vermeidung unnötiger Datenspeicherung.
   - Umgang mit Anfragen zur Datenlöschung.
3. Zugriffskontrolle und Berechtigungen
   - Sensibilisierung für den Schutz von Admin-Konten.
   - Keine Datenweitergabe an Dritte.
4. Schutz vor Betrug
   - Erkennen von Manipulationsversuchen.
   - Punktevergabe durch 4-Augen-Prinzip.
5. Allgemeine IT-Sicherheit
   - Schutz vor Phishing-Angriffen.
   - Umgang mit verdächtigen Nachrichten.
6. Massnahmen
   - Was tun im Falle eines Datenlecks?
   - Verantwortlichkeiten und Kommunikationswege.

In den Datenschutzrichtlinien des jeweiligen Vereins sollten diese Punkte möglichst adressiert und transparent beschrieben werden (vgl. Anhang „Datenschutzrichtlinie-Beispiel.pdf“).

4 Gamification-Konzept

Gamification beschreibt den gezielten Einsatz spieltypischer Elemente wie Punkte, Ranglisten oder Belohnungen in nicht-spielerischen Kontexten, um Motivation und Engagement zu fördern.

In diesem Kapitel wird untersucht, wie Gamification in einem Discord-Server für einen Verein implementiert werden kann. Ziel ist es, die Mitgliederbeteiligung zu erhöhen und den Vorstand bei der Organisation von Events zu unterstützen. Das Konzept basiert auf wissenschaftlichen Erkenntnissen und wird durch eine Umfrage unter den Mitgliedern des Vereins EGSwiss (siehe Abschnitt 1.1) ergänzt.

4.1 Wissenschaftliche Analyse

Nielsen (2006) erläutert, dass 90% der Nutzer in Online-Communitys lediglich Zuschauer sind, während 9% der Nutzer einen kleinen Beitrag leisten und 1% für fast alle Aktivitäten verantwortlich sind. Dies zeigt das viele Communitys von einer kleinen aktiven Gruppe getragen werden. Um eine nachhaltige Community zu fördern müssen Anreize geschaffen werden, welche mehr Nutzer motiviert sich zu involvieren, wobei ein Gamification-System helfen kann. (Nielsen, 2006)

In einer umfassenden Metaanalyse von Sailer und Homner (2020) wird die Wirksamkeit von Gamification auf drei zentralen Ebenen belegt:

- Kognitive Effekte z. B. gesteigerte Informationsverarbeitung.
- Motivationsbedingte Effekte z. B. gesteigerte Beteiligung durch intrinsische Motivation.
- Verhaltensbedingte Effekte z. B. erhöhte Teilnahme oder Aktivität

Besonders effektiv zeigte sich Gamification dann, wenn soziale Interaktion (z. B. durch Ranglisten) mit spielerischer Konkurrenz kombiniert wurde, jedoch in einem konstruktiven, unterstützenden Rahmen. Bei Nutzer mit einer niedrigen Platzierung kann sich eine Rangliste jedoch demotivierend auswirken. (Sailer & Homner, 2020)

Eine ergänzende Studie von Looyestyn et al. (2017) analysierte explizit die Wirkung von Gamification auf Online-Programme. Von 15 qualitativ bewerteten Studien zeigten 12 signifikant positive Effekte auf das Engagement. Besonders wirksam waren Gamification-Elemente wie Punkte, Ranglisten und Belohnungen. Die Effekte waren insbesondere bei kurzfristigem Einsatz stark ausgeprägt, wohingegen die Wirkung über längere Zeiträume tendenziell abnahm. Auch Folgeeffekte des Engagements wie Lernleistung und Gesundheitsverhalten verbesserten sich in mehreren Studien. (Looyestyn et al., 2017)

Insgesamt stützen beide Studien die Annahme, dass Gamification eine wirkungsvolle Methode ist, um Beteiligung, Motivation und bestimmte Zielverhaltensweisen in digitalen Kontexten gezielt zu fördern.

Das Gamification-Konzept überträgt die wissenschaftlichen Erkenntnisse auf den Vereinskontext. Zentraler Bestandteil ist ein über einen Discord-Bot umgesetztes Gamification-System, das die folgenden Mechanismen enthält.

- Punkte für Aktivitäten (Events, Aufgaben, Umfragen etc.)
- Ranglisten (saisonal, jährlich, allzeit)
- Belohnungssystem (Rabatte, Merchandise)
- Soziale Interaktion durch transparente Wettbewerbsmechanismen

Die Kombination dieser Elemente zielt auf eine Erhöhung Engagements im Verein ab, indem sowohl direkte Anreize als auch soziale Motivation durch Sichtbarkeit in der Gemeinschaft geschaffen werden.

4.2 Nutzergruppen Analyse

Um die Bedürfnisse und Ängste der Nutzer im Gamification-Konzept abzudecken, wurde eine Umfrage durchgeführt (vgl. Anhang „Gaming Umfrage - Plan.pdf“). Hierbei wurden vor allem Personen aus dem Umfeld des Gamingvereins EGSwiss (siehe Abschnitt 1.1) zur Teilnahme aufgefordert. Die Zielgruppe dieser Umfrage besteht aus den folgenden Gruppen:

- Nicht-Mitglieder, die Discord benutzen
- Mitglieder, die kein Discord benutzen
- Mitglieder, die Discord benutzen
- Vorstandsmitglieder, die Discord benutzen

Es werden 13 teilnehmende verzeichnet, wobei 69.2% aus Mitglieder oder Vorstand bsteht und die restlichen 30.8% Nicht-Mitglieder sind. (siehe Abb. 6)

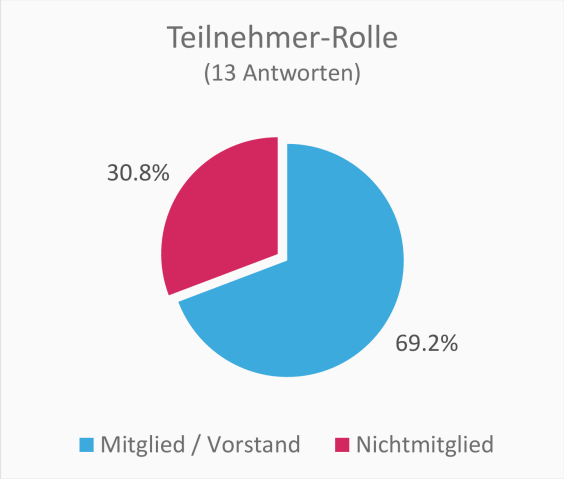


Abb. 6: Umfrage Diagramm: Teilnehmer-Rolle

Die Selbsteinschätzung der Befragten hinsichtlich ihrer Erfahrungen in der Teilnahme und Organisation von Online- sowie Offline-Events weist auf eine gleichmässige Zusammensetzung der Teilnehmer hin. Diese Vielfalt spiegelt sich auch in der Bandbreite der Vereinszugehörigkeitsdauer wider und ermöglicht somit eine differenzierte Betrachtung der Perspektiven sowohl langjähriger als auch neuer Mitglieder. Die Analyse des Zusammenhangs zwischen Mitgliedschaftsdauer und Aktivitätsverhalten im Verein zeigt ein konstantes Verhältnis, was auf ein bereits bestehendes, stabiles Engagement innerhalb des Vereins hinweist. Somit ist in diesem Aspekt Gamification nicht unbedingt notwendig, für eine insgesamte Verbesserung aber auch nicht hinderlich. Zu beachten ist ausserdem, dass diese Umfrage tendenziell von Mitgliedern ausgefüllt wurde, welche von Grund auf bereits mehr Interesse am Verein zeigen. (siehe Abb. 7)

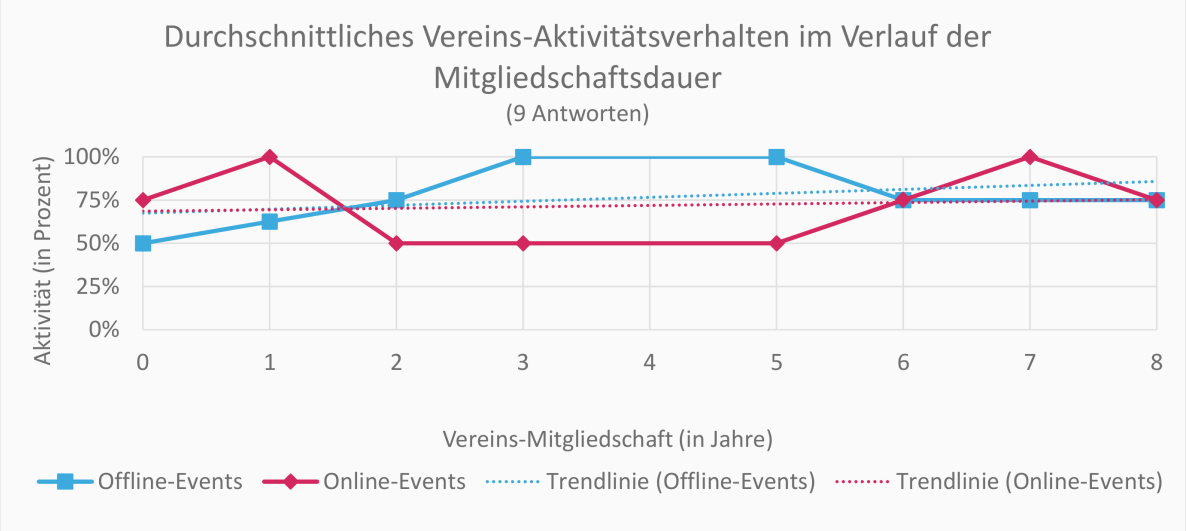


Abb. 7: Umfrage Diagramm: Durchschnittliches Vereins-Aktivitätsverhalten im Verlauf der Mitgliedschaftsdauer

Die meisten Befragten (92.3%) benutzen bereits den Vereins Discord-Server. Eine Person (7.7%) ist nicht im Discord-Server. (siehe Abb. 8)

Seit der Erstellung des Discord-Servers im Jahr 2016 erfolgte ein gleichmässiger Beitritt neuer Mitglieder über die Zeit hinweg. Eine Analyse des Zusammenhangs zwischen der Mitgliedschaftsdauer und dem Nutzungsverhalten zeigt jedoch eine leicht negative Korrelation. Mit zunehmender Dauer der Discord-Zugehörigkeit nimmt die Aktivität der Mitglieder tendenziell ab. Dieses Muster deutet auf ein Optimierungspotenzial im Bereich der Mitgliederbindung hin. Durch den gezielten Einsatz eines Gamification-Systems könnte insbesondere die langfristige Aktivierung und Einbindung langjähriger Mitglieder gefördert werden. (siehe Abb. 9)

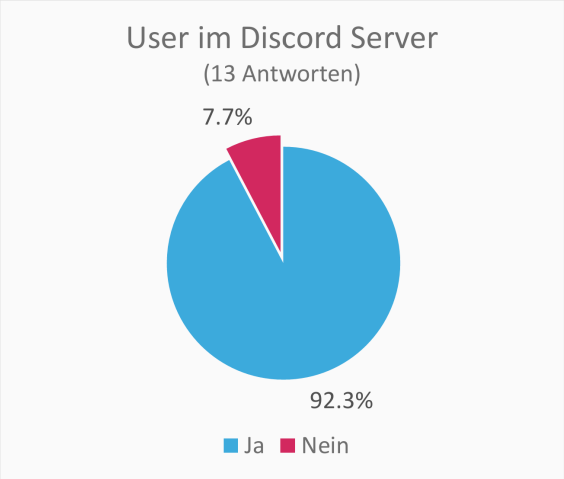


Abb. 8: Umfrage Diagramm: User im Discord-Server

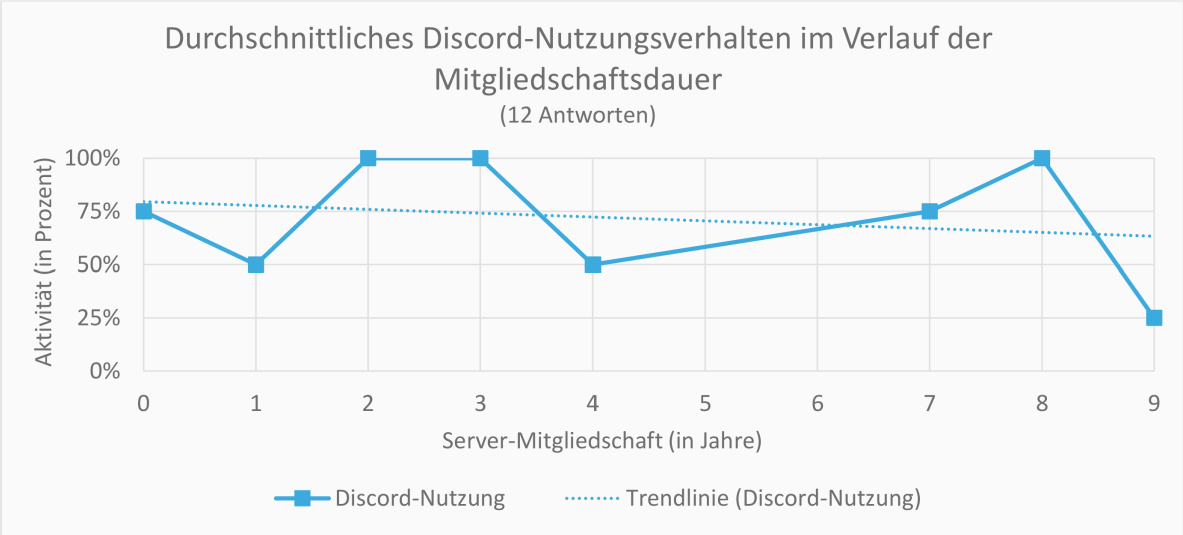


Abb. 9: Umfrage Diagramm: Durchschnittliches Discord-Nutzungsverhalten im Verlauf der Mitgliedschaftsdauer

4.2.1 Teilnahme an Vereinsaktivitäten

Die Hauptmotivation für die Teilnahme an Vereinsaktivitäten ist der soziale Austausch. Viele schätzen es, Gleichgesinnte zu treffen, gemeinsam zu spielen und an Events teilzunehmen. Spass, eine gute Atmosphäre und der Zusammenhalt im Verein sind entscheidend. Auch andere Faktoren wie E-Sports Teams, etwa in „League of Legends“ , spielen eine Rolle.

Häufige Hindernisse für eine aktivere Teilnahme sind Zeitmangel, berufliche Verpflichtungen und Terminkollisionen. Einige Mitglieder sind introvertiert oder nur an bestimmten Vereinsaktivitäten interessiert. Auch geografische Distanzen können eine Hürde darstellen.

Zur Steigerung der Motivation wurden verschiedene Ansätze vorgeschlagen. Dazu gehören der Einsatz von Gamification-Elementen, frühzeitigere Ankündigungen von Events und die Verteilung von Anreizen wie Goodies.

Das Gamification-Konzept für den Discord-Bot berücksichtigt diese Erkenntnisse und zielt darauf ab, flexible und zugängliche Anreize zu schaffen, um die Vereinsaktivität nachhaltig zu fördern.

4.2.2 Einführung eines Gamification-Systems

Die Umfrage zeigt, dass 76.9% der Befragten keinen Mehrwert in ein Punktesystem sehen, wobei 23.1% angaben, dass ein solches System ihre Motivation erhöhen würde. Eine Person spezifizierte, dass selbst zwar keine zusätzliche Motivation daraus gezogen werden könne, jedoch das Potenzial für andere Mitglieder vorhanden sei (siehe Abb. 10). Dieses Ergebniss erscheint zunächst negativ für das Gamification-Konzept, jedoch stellt sich durch weitere Fragen heraus, dass die Bedencken für eine unzureichende oder gar unfaire Umsetzung die Vorteile überwiegt (siehe Abschnitt 4.2.3). Somit muss bei der Definion darauf geachtet werden, dass die Regeln gezielt gestaltet sowie Belohnungen an unterschiedliche Motivationsprofile angepasst werden, um eine breite Akzeptanz zu gewährleisten.

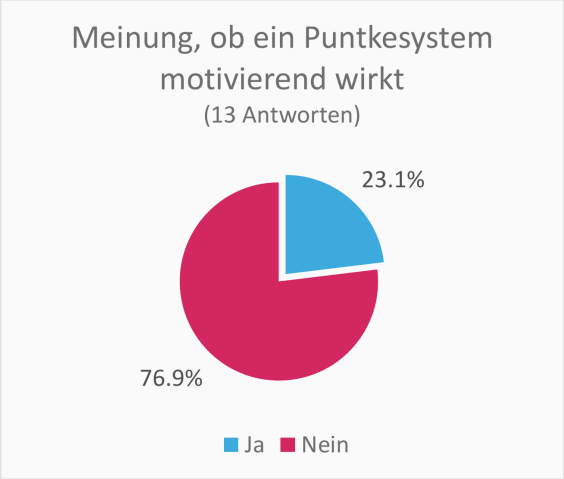


Abb. 10: Umfrage Diagramm: Meinung zum Punktesystem

Die Analyse zeigt, dass Events (z. B. Teilnahme, Organisation) am häufigsten als geeignete Punkteaktivität genannt wurden (n = 9), jedoch nur drei Personen dadurch motiviert würden. Ähnlich verhält es sich bei allgemeinen Aufgaben wie Umfragen oder Vorschlägen (n = 7), bei denen ebenfalls nur drei Personen eine motivierende Wirkung angaben. Tippspiele (n = 7), Mitgliederrekrutierung (n = 6) und Spenden (n = 5) erzielten durchweg geringe motivierende Wirkung (jeweils zwei Stimmen). Eine zusätzliche Idee war die Vergabe von Punkten für Rätsel.

Insgesamt zeigt sich, dass viele Aktivitäten zwar als relevant eingeschätzt werden, jedoch nur wenige tatsächlich zur Motivation beitragen. Ein Gamification-System sollte sich daher auf solche Aufgaben konzentrieren, die sowohl als relevant als auch als motivierend wahrgenommen werden, insbesondere Events und interaktive Vereinsaktivitäten. Zudem muss das Vertrauen in ein nachhaltiges und faires System gestärkt werden, um die Motivation bei den Mitgliedern zu erhöhen.

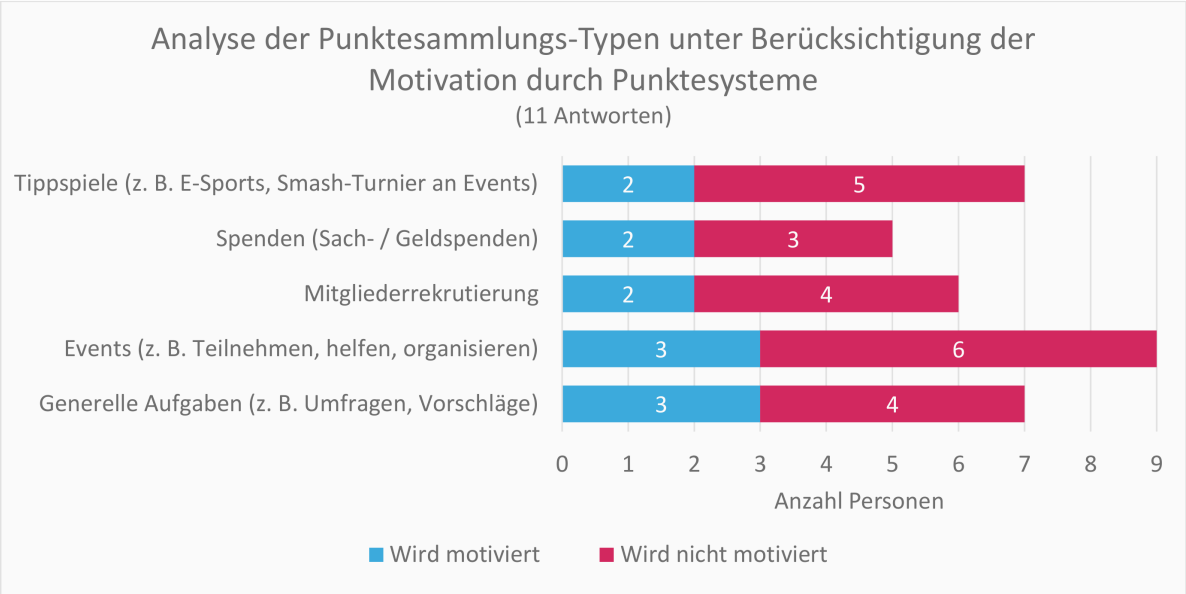


Abb. 11: Umfrage Diagramm: Punktesystem Auswahl

Die Einführung einer Rangliste wurde von 46.2% der Befragten als motivierend angesehen, während 53.8% keinen Mehrwert darin sehen. Dies zeigt, dass eine Rangliste für einige Mitglieder einen positiven Anreiz schaffen kann, während andere möglicherweise keinen Wettbewerbsgedanken verfolgen oder andere Motivationsfaktoren bevorzugen. Eine Rangliste sollte daher optional und ergänzend zu anderen Gamification-Elementen gestaltet werden, um verschiedene Motivationstypen anzusprechen. (siehe Abb. 12)

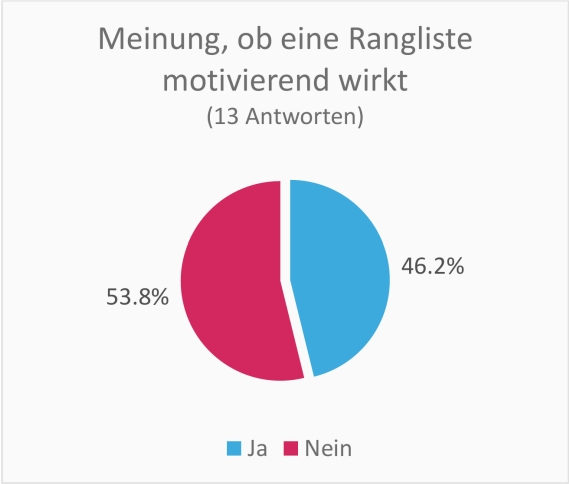


Abb. 12: Umfrage Diagramm: Meinung zur Rangliste

Die Mehrheit der Befragten befürwortet eine jährliche Rangliste mit Zurücksetzung nach einem Vereinsjahr (n = 8, davon 5 motivierend). Auch eine permanente Rangliste erhält Zustimmung (n = 7, davon 5 motivierend). Eine saisonale Rangliste (alle 3 Monate) wird hingegen kaum unterstützt (n = 3, davon 2 motivierend). Vier Teilnehmende lehnen Ranglisten grundsätzlich ab, und eine Person äusserte Bedenken hinsichtlich potenzieller Demotivation bei inaktiveren Mitgliedern. (siehe Abb. 13)

Dieses Ergebniss zeigt, dass eine sorgfältige Gestaltung notwendig ist, um Frustration zu vermeiden. Eine Kombination aus einer dauerhaften Rangliste für langfristige Anerkennung und einer jährlichen Zurücksetzung für neue Chancen könnte eine ausgewogene Lösung sein.

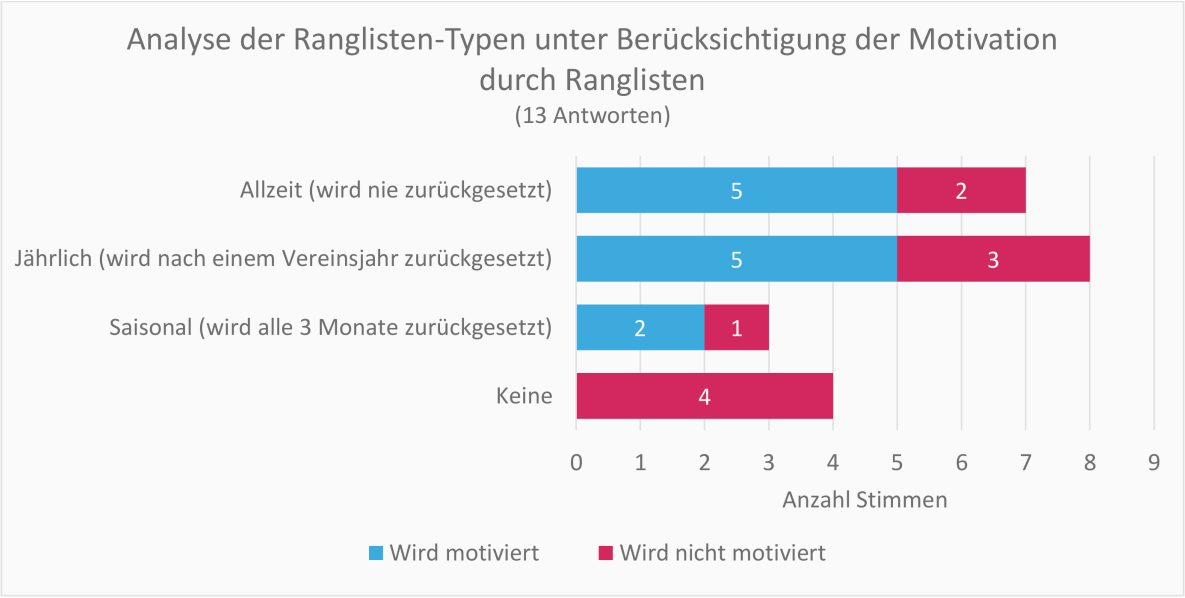


Abb. 13: Umfrage Diagramm: Ranglisten Auswahl

Die Umfrage zeigt, dass Merchandise mit 11 Stimmen die beliebteste Belohnungsart ist. Auch Rabatte im Vereins-Shop (sieben Stimmen) und Gutscheine für Plattformen wie Steam oder Digitec (sechs Stimmen) werden als attraktive Anreize angesehen. Diese Ergebnisse legen nahe, dass Belohnungen einen direkten Bezug zum Verein haben sollten, um die Identifikation mit der Gemeinschaft zu stärken. Merchandise und exklusive Vereinsrabatte bieten hier einen zusätzlichen Mehrwert, während Gutscheine eine flexible Alternative für unterschiedliche Interessen darstellen.

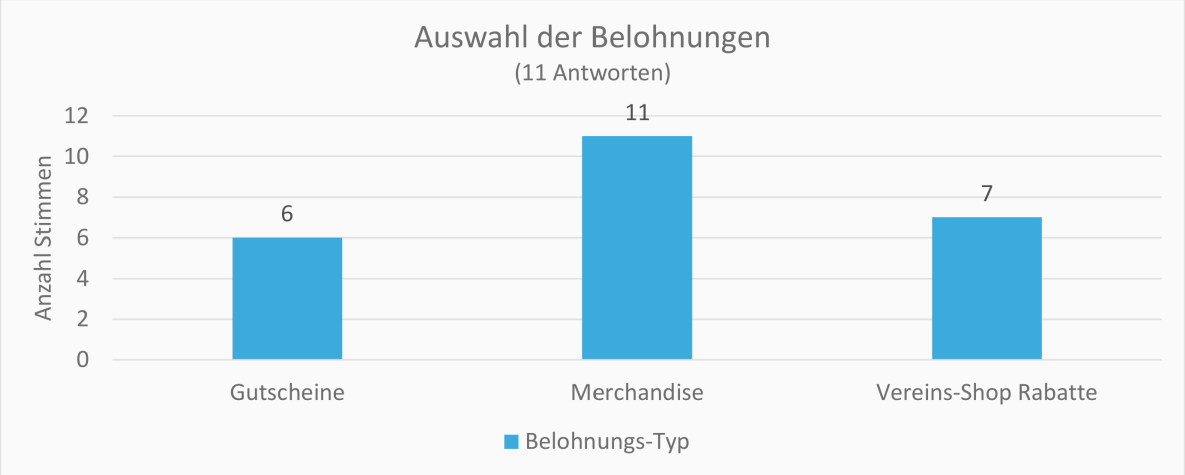


Abb. 14: Umfrage Diagramm: Belohnungen Auswahl

Alle Teilnehmer der Umfrage zeigen eine klare Ablehnung gegenüber der Idee, dass Punkte innerhalb des Vereins Vorteile wie mehr Stimmen bei Abstimmungen verschaffen könnten. Ebenso wurde die Übertragbarkeit von Punkten zwischen Mitgliedern abgelehnt, da auch hier alle 13 Befragten diese Funktion negativ bewerteten. Diese Ergebnisse deuten darauf hin, dass die Mitglieder ein Punktesystem bevorzugen, das auf persönlicher Leistung basiert und keine direkten hierarchischen oder übertragbaren Vorteile bietet. Ein solches System sollte also ausschliesslich zur Anerkennung der aktiven Teilnahme dienen, ohne Einfluss auf Entscheidungsprozesse oder die Verteilung von Ressourcen innerhalb des Vereins zu haben.

4.2.3 Vorschläge und Kritik

Die Befragten haben verschiedene kreative Vorschläge zur Erweiterung des Gamification-Systems. Ein häufiger Vorschlag ist die Einführung von internen Turnieren, bei denen Punkte vergeben werden. Eine andere Idee ist die Möglichkeit, Emotes, Badges oder neue Funktionen freizuschalten, die Mitglieder im Server verwenden oder in ihrem Username anzeigen lassen können. Ein weiteres innovatives Konzept ist die Einführung eines Tamagotchi- oder Pokémon-Systems, bei dem Mitglieder ein virtuelles Wesen mit ihren Punkten „leveln“ können.

Es werden mehrere wichtige Bedenken geäussert. Besonders betont wird, dass Punkte nicht zu einem Zwang führen sollten oder zu ungerechter Behandlung von Mitgliedern, die weniger Punkte haben. Ein weiterer Punkt ist die Sorge, dass die Gamification nicht zu einer „Premium Currency“ oder einem „Pay-to-Win“-System führen dürfe, in dem finanzieller Beitrag die Aktivität übertreffen könnte. Ausserdem wird kritisiert, dass Punkte nicht als Grundlage für mehr Stimmen bei Abstimmungen oder für Vorteile innerhalb des Vereines dienen dürften, da das zu toxischen Dynamiken führen könnte.

Einige Teilnehmer heben hervor, dass Gamification nicht demotivierend für weniger aktive Mitglieder sein sollte und dass eine zu starke Fokussierung auf Punkte den gegenteiligen Effekt haben könnte. Die Idee, Punkte spielerisch zu nutzen, beispielsweise in Turnieren, fand jedoch positive Resonanz, solange dies fair und gleichberechtigt gestaltet wird.

Die Ergebnisse zeigen, dass das Gamification-System sowohl spassige Elemente (wie Turniere und Freischaltungen) als auch klare Grenzen setzen muss, um Gerechtigkeit und Inklusion zu gewährleisten. Punkte sollten auf Engagement und Teilnahme ausgerichtet sein, ohne wirtschaftliche Vorteile oder hierarchische Unterschiede zu schaffen.

4.3 Herausforderungen und Lösungsansätze

Anhand der wissenschaftlichen Analyse (siehe Abschnitt 4.1) sowie der durchgeführten Nutzergruppen Analyse (siehe Abschnitt 4.2) werden die Herausforderungen definiert, welche im Konzept berücksichtigt werden müssen:

- **Wettbewerbsdruck und negative Gruppendynamiken:** Looyestyn et al. (2017) zeigen, dass übermässige Konkurrenz in sozialen Gruppen kontraproduktiv sein kann. Die Einführung mehrerer Ranglisten (allzeit, jährlich, saisonal) bietet differenzierte Teilnahmemöglichkeiten und reduziert die Einstiegshürde. (Looyestyn et al., 2017)
- **Benachteiligung inaktiver Mitglieder:** Personen mit weniger Zeit für Vereinsaktivitäten dürfen nicht demotiviert werden. Die Umfrage hat gezeigt, dass Mitglieder ein System bevorzugen, das sowohl kurzfristige als auch langfristige Erfolge belohnt (siehe Abschnitt 4.2.3). Dies stimmt mit den Erkenntnissen der Metastudie von Looyestyn et al. (2017) überein, wonach kurzfristige Gamification-Impulse besonders effektiv sind, jedoch mit langfristigen Komponenten ergänzt werden sollten. (Looyestyn et al., 2017)
- **Manipulationsschutz:** Eine faire Punktevergabe ist essenziell, da Belohnungen mit echtem Mehrwert angeboten werden. Laut Umfrage erwarten Mitglieder klare Massnahmen zur Verhinderung von Manipulationen (siehe Abschnitt 4.2.3). Massnahmen wie doppelte Verifizierungen durch Vorstandsmitglieder und transparente Protokollierung sollen Manipulation verhindern und wurden durch das Feedback der Mitglieder unterstützt.

4.4 Regeln

Die identifizierten Problemstellen werden durch ein klar definiertes Reglement abgedeckt. Die Punktevergabe unterliegt folgenden Regeln:

1. Mitglieder und Vorstandsmitglieder können Punkte sammeln, wobei Vorstandsmitglieder keine übermässigen Vorteile erhalten dürfen.
2. Punkte können nur durch vom Vorstand vordefinierte Aufgaben gesammelt werden.
3. Regelwidrig gesammelte Punkte werden nicht genehmigt oder rückwirkend entfernt.
4. Punkte sind nicht übertragbar und verfallen bei Austritt aus dem Verein.

4.5 Aufgaben und Aktivitäten

Punktesysteme wirken laut der Self-Determination Theory, auf der die Studie von Sailer und Homner (2020) basiert, als unmittelbare Rückmeldung für Nutzende und stärken dadurch das Gefühl von Kompetenz und Selbstwirksamkeit. Diese direkte Rückmeldung motiviert die Teilnehmenden, weitere Aufgaben zu bewältigen und aktiv zu bleiben. Auch Looyestyn et al. (2017) bestätigen, dass Punktesysteme besonders effektiv für kurzfristige Aktivierungsimpulse sind und Engagement im digitalen Raum nachweislich steigern können. (Looyestyn et al., 2017; Sailer & Homner, 2020)

Punkte können auf verschiedene Arten gesammelt werden. Abb. 15 zeigt den Ablauf der Punktesammlung.

Um Punkte zu sammeln, muss die Person zunächst Mitglied des Vereins sein. Nimmt ein Mitglied an einem Event Teil oder erledigt eine Aufgabe werden dafür Punkte auf das Punktekonto gutgeschrieben. Zudem werden seine Anzahl Gesamtpunkte erhöht, was relevant ist für die Ranglisten. Konvertiert ein Mitglied Punkte zu Rabatten im Vereins-Shop oder für andere Belohnungen um, werden diese lediglich vom Kontostand abgezogen. Wie im Diagramm zu erkennen ist, hat die Punktekonvertierung keinen Einfluss auf die gesammelten Gesamtpunkte der Rangliste. Tritt ein Mitglied aus dem Verein aus werden seine Ranglisteneinträge und sein Kontostand nach einer bestimmten Zeit gelöscht.

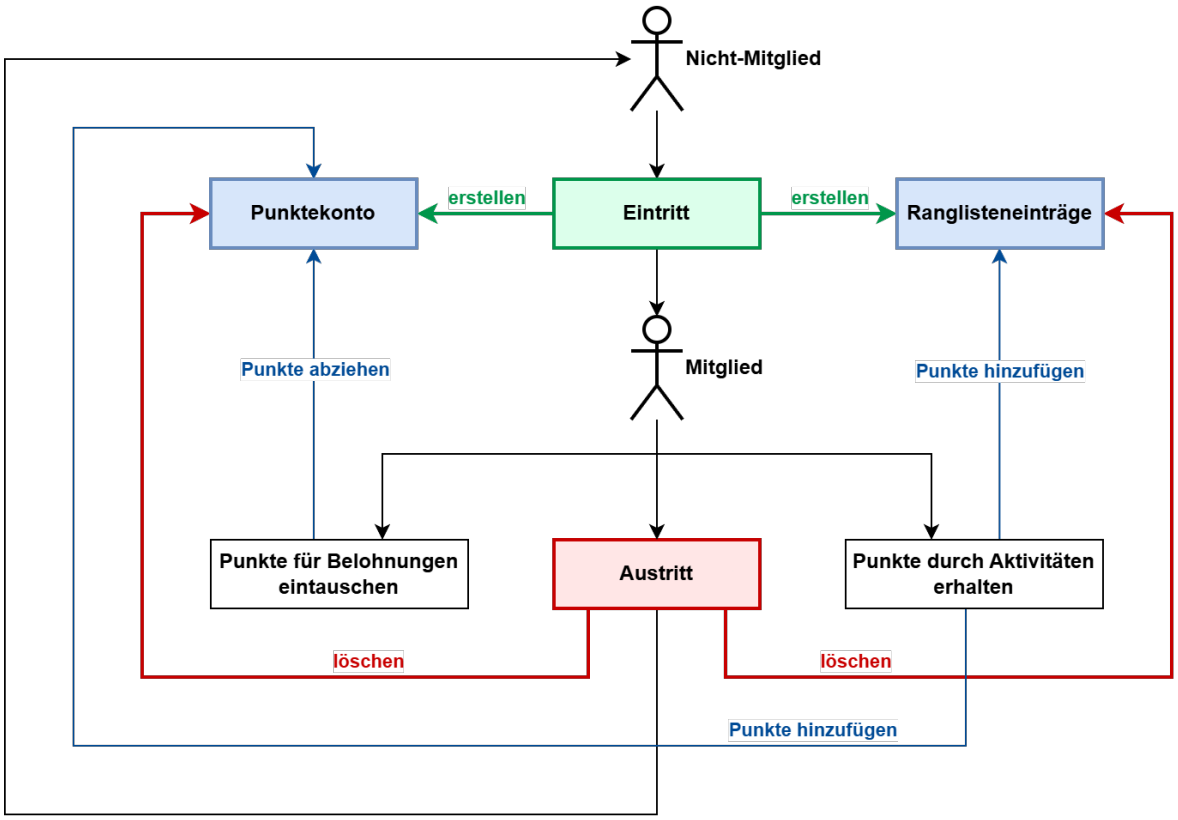


Abb. 15: Gamification Ablaufdiagramm

Das Konzept nutzt gezielt soziale Mechanismen wie kooperative Aufgaben, Gruppenaufgaben und öffentliche Sichtbarkeit von Leistungen. Sailer und Homner (2020) zeigen, dass diese Mechanismen das Bedürfnis nach sozialer Eingebundenheit stärken, welches ein Schlüsselfaktor für nachhaltiges Engagement ist. (Sailer & Homner, 2020)

Aktivitäten werden mit Punkten belohnt. Dies können kleinere Aufgaben sein, wie das Ausfüllen einer Umfrage oder grössere als das Unterstützen bei der Planung eines Events. Die Aufgaben werden über den Discord-Bot an die Mitglieder kommuniziert, wobei die Punkte entsprechend dem Aufwand vom Vorstand festgelegt werden.

Die Umfrage unter den Nutzergruppen hat bestätigt, dass eine Mischung aus kleinen und grösseren Aufgaben die Motivation am besten fördert. Die nachfolgende Reihenfolge der Auflistung entspricht der Beliebtheit dieser vordefinierten Möglichkeiten bei der Umfrage (siehe Abschnitt 4.2.2).

Aktivität zur Punktesammlung	Beschreibung
Events	Für die Teilnahme, Organisation oder Hilfe an Events werden Punkte verteilt. Der Discord-Bot informiert über das Event sowie über die Menge an Punkten, welche für die Teilnahme gesammelt werden können. Der Vorstand kann die Punkte mithilfe des Discord-Bots automatisch an die Mitglieder verteilen welche am Event teilgenommen haben.
Generelle Aufgaben	Punkte können durch die Teilnahme an Umfragen, Vorschläge oder das Erledigen von einmaligen Aufgaben im Verein gesammelt werden. Dadurch werden die Mitglieder dazu ermutigt sich produktiv im Verein zu beteiligen.
Tippspiele	Tippspiele sind eine weitere Möglichkeit die Interaktion im Verein zu steigern. Mitglieder können beispielsweise auf den Sieger von E-Sports Turnieren oder andere Ereignisse tippen. Bei korrekter Vorhersage werden den Mitgliedern Punkte gutgeschrieben, was die aktive Teilnahme und das Engagement erhöht. Wichtig zu erwähnen ist hierbei, dass keine Punkte als Einsatz gesetzt werden können, da es sich nicht um Glücksspiel handeln soll.
Mitgliederrekrutierung	Mitglieder können Punkte sammeln in dem sie neue Mitglieder rekrutieren. Dies motiviert Mitglieder den Verein zu bewerben und hilft ihm zu wachsen.
Spenden	Für jegliche Art von Spenden an den Verein können Mitglieder Punkte erhalten. Punkte die durch Spenden gesammelt werden können sind bewusst eingeschränkt: • Es können maximal ein Mal im Monat Punkte für Geld- und Sachspenden erhalten werden. • Die erhaltenen Punkte sind nur begrenzt vom Spendenbetrag abhängig. • Die Menge der erhaltenen Punkte ist klein. Diese Einschränkungen sind nötig um sicherzustellen, dass das Punktesystem fair bleibt und nicht zu einem Pay-To-Win System mutiert. Die Spenden für einen Verein sind freiwillig und für das Wohl des Vereines. Darum sollten sie nicht Teil eines Art „Cash-back"-System werden, welche durch das eintauschen von Punkten für Rabatte gewährleistet wäre. Eine kleine Menge von Punkten können als Dankeschön verteilt werden.

Tab. 47: Beschreibung von Aktivitäten zur Punktesammlung im Gamification-Konzept

4.6 Rangliste

Ranglisten-basierte Konkurrenz in Kombination mit sozialem Austausch sind gemäss Sailer und Homner (2020) besonders effektiv für verhaltensbezogene Effekte. Auch Looyestyn et al. (2017) erläutern, dass Ranglisten die Interaktion und Plattformnutzung signifikant steigert, insbesondere bei kürzeren Einsätzen oder für spezifische Challenges. Aus diesem Grund werden in diesem Konzept mehrere Ranglisten geführt (zeitlich differenziert), Belohnungen gezielt für aktuelle Leistungen (nicht nur Allzeit-Historie) vergeben und transparente Teilnahmebedingungen geschaffen, um Fairness und soziale Verträglichkeit sicherzustellen. Ausserdem kann der Vorstand die Anzahl sichtbare Einträge bestimmen, damit Benutzer mit einer niedrigen Platzierung nicht ersichtlich sind und somit die Gefahr einer Demotivation geringer ausfällt. (Looyestyn et al., 2017; Sailer & Homner, 2020)

Das System umfasst drei Ranglisten mit unterschiedlichen Motivationsansätzen:

Rangliste für Punkte	Beschreibung
Allzeit	Die Allzeit-Rangliste kann als „Hall of Fame“ gesehen werden. Hier werden die aktivsten und engagiertesten Mitglieder dargestellt. Diese Rangliste verleiht jedoch einzig „bragging rights“ (dt. Angeberechte), Belohnungen werden hier für hohe Platzierungen nicht vergeben. Dies hat folgende Gründe: • Die Allzeit Rangliste spiegelt nicht die aktuelle Aktivität oder das aktuelle Engagement der Mitglieder wider. Werden Belohnungen für hohe Platzierungen vergeben, könnte es vorkommen, dass diese an Mitglieder vergeben werden, die schon länger nicht mehr aktiv am Verein teilhaben. Dies könnte sich demotivierend auf aktive Mitglieder auswirken. • Die Gefahr besteht das Belohnungen immer an die gleichen Mitglieder verteilt werden („Reiche werden reicher“). Auch dies kann sich negativ auf die Motivation anderer Mitglieder auswirken.
Vereins-Jährlich	Die vereins-jährliche Rangliste zeigt die aktivsten Mitglieder des aktuellen Vereinsjahres auf. Hier werden Belohnungen für hohe Platzierungen verteilt. Mehr Informationen dazu im Abschnitt Belohnungen . Die vereins-jährliche Rangliste wird mit der Abhaltung der Generalversammlung zurückgesetzt.
Saisonal	Die saisonale Rangliste gilt immer für 3 Monate. Diese Hilft dabei Neu-Mitglieder schnell in den Verein zu integrieren, da sie nach kurzer Zeit die Chance haben Belohnungen zu gewinnen. Die zusätzliche motivation für Neu-Mitgliedern an verschiedene Vereinsaktivitäten teilzunehmen hat positive Auswirkungen auf den gesamten Verein, da dadurch schnell neue Kontakte geknüpft werden können. Die saisonale Rangliste wird nach jeder Saison zurückgesetzt.

Tab. 48: Beschreibung der Ranglisten für Punkte im Gamification-Konzept

4.7 Belohnungssystem

Belohnungen unterstützen laut der Studie von Sailer und Homner (2020) extrinsische Motivation, können aber auch zur Verstärkung von intrinsisch motiviertem Verhalten beitragen, wenn sie nicht als reines „Bestechungsmittel“ wirken. Die Analyse von Looyestyn et al. (2017) unterstreicht ebenfalls den Nutzen symbolischer Belohnungen als Verstärker für Engagement, insbesondere wenn sie in einem sinnvollen Verhältnis zur erbrachten Aktivität stehen. (Looyestyn et al., 2017; Sailer & Homner, 2020)

Diese Erkenntnisse werden im Gamification-Konzept berücksichtigt, indem Belohnungen gezielt eingesetzt werden, um sowohl kurzfristige Anreize zu schaffen als auch langfristige Bindung an den Verein zu fördern. Die Belohnungen sind so gestaltet, dass sie den Mitgliedern einen echten Mehrwert bieten und gleichzeitig die Identifikation mit dem Verein stärken.

4.7.1 Belohnungen für Ranglistenpositionen

Belegt ein Mitglied einen der oberen Plätze bei der jährlichen oder saisonalen Rangliste, erhält es vom Vorstand definierte Belohnungen. Belohnungen für die jährliche Rangliste fallen hierbei besser aus als für die saisonale Rangliste, damit eine längerfristige Motivation gefördert wird. Nachfolgend ein paar Beispielbelohnungen:

- Merchandise des Vereins (z. B. T-Shirts, Hoodies, etc.)
- Gutscheine (Nicht für den Vereins-Shop, diese kann man sich für Punkte direkt einlösen. Mehr dazu in Abschnitt 4.7.2 .)
- Badges oder Icons in Discord

Nicht geeignet als Belohnung sind weitere Punkte, da für das Sammeln von Punkten, weitere Punkte zu erhalten nicht sehr zufriedenstellend ist. Zudem würde es dazu führen, dass Mitglieder die bereits viele Punkte haben noch mehr erhalten. Dies macht es für andere Mitglieder schwerer aufzuholen und beeinträchtigt somit den Wettbewerb.

4.7.2 Rabatte im Vereins-Shop

Gesammelte Punkte können im Vereins-Shop für Rabatte eingetauscht werden. Die Anzahl Punkte die benötigt werden für bestimmte Rabatte und den maximalen Wert von Rabatten werden durch den Vorstand bestimmt.

4.8 Berechnung des Punktesystems

Ein systematisch entwickeltes Punktesystem stellt eine wesentliche Grundlage zur Förderung der Mitgliederaktivität und zur strukturierten Anerkennung von Engagement im Verein dar. Im Rahmen dieser Arbeit wurde ein Punktesystem konzipiert, das sowohl eine technische als auch eine administrative Perspektive integriert. Die technische Grundlage bildet das Datenbankmodell `Transaction`, welches sämtliche Punktevergaben revisionssicher und nachvollziehbar dokumentiert. Ergänzend wurde eine Excel-Vorlage entwickelt, die dem Vorstand eine flexible Planung, Verwaltung und Analyse der Punktevergabe ermöglicht (vgl. Anhang „pointsystem_calculator.xlsx“).

Die Excel-Vorlage dient dabei als Werkzeug zur Messbarmachung der Punktevergabe und unterstützt die strategische Ausrichtung des Vereins, indem sie eine konsistente und transparente Vergabe von Punkten gewährleistet. Sie ermöglicht zudem dynamische Simulationen, um unterschiedliche Szenarien hinsichtlich Spenden, Aufwänden und Eventteilnahmen durchzurechnen. Diese Funktionalitäten tragen zur langfristigen Optimierung der Anreizstrukturen und zur kontinuierlichen Weiterentwicklung des Systems bei.

Im Folgenden werden die zentralen Funktionalitäten der Excel-Vorlage detailliert erläutert. Ziel ist es, die methodische Umsetzung des Punktesystems umfassend darzustellen und eine Grundlage für die Evaluierung und Weiterentwicklung zu schaffen. Die konkrete technische Umsetzung dieses Punktesystems wird im Kapitel Abschnitt 7.7.1 beschrieben.

4.8.1 Punkteübersicht und Punktekatalog

Die Excel-Vorlage enthält eine tabellarische Auflistung aller Aktivitätsarten, die für die Punktevergabe im Verein relevant sind (z. B. Spenden, Aufgaben, Events). Diese Auflistung dient als standardisierte Entscheidungsgrundlage für den Vorstand und erleichtert die konsistente Vergabe von Punkten.

Technisch werden dabei vordefinierte Aktivitätsarten mit spezifischen Punktewerten verknüpft, welche als Schlüsselwertpaare in der Excel-Datei hinterlegt sind. Diese Struktur erlaubt es dem Vorstand, neue Aktivitäten einfach zu ergänzen und bestehende Punkteschlüssel an die Vereinsbedürfnisse anzupassen.

Definition der Punktevergabe für das Punktesystem					
	Realwert (CHF)	**Punkte**	**Punkte-pro-CHF**	**CHF-pro-Punkt (Belohnungswert ohne Ziel-Kostenanteil)**	**Belohnungswert mit 50% Vereins-Kostenanteil**
Umrechnungskurs	CHF 1.00	15	15	**CHF 1.00**	**CHF 0.50**
	Spendenwert (CHF)	**Punkte**	**Punkte-pro-CHF**	**CHF-pro-Spendenpunkt (Belohnungswert ohne Ziel-Kostenanteil)**	**CHF mit 50% Vereins-Kostenanteil**
Spende	CHF 1.00	5	5	**CHF 0.33**	**CHF 0.17**
Aufwandsentschädigung	**Arbeits-Stunden (h)**	**Punkte**	**Punkte-pro-h**	**CHF-pro-Aufwands-Stunde (Belohnungswert ohne Ziel-Kostenanteil)**	**CHF mit 50% Vereins-Kostenanteil**
Aufgaben (z. B. Social Media, Events)	1	30	30	**CHF 2.00**	**CHF 1.00**
Aufwandsentschädigung		**Punkte**		**CHF-pro-Aufwandspunkt (Belohnungswert ohne Ziel-Kostenanteil)**	**CHF mit 50% Vereins-Kostenanteil**
Event-Teilnahme	--	10	--	**CHF 0.67**	**CHF 0.33**
Umfragen-Teilnahme	--	10	--	**CHF 0.67**	**CHF 0.33**
Mitglieder-Rekrutierung	--	150	--	**CHF 10.00**	**CHF 5.00**

Abb. 16: Punkterechner-Excel - Beispiel einer Punkte-Definition

4.8.2 Dynamische Umrechnungstabellen

Zur Simulation verschiedener Szenarien enthält die Vorlage dynamische Umrechnungstabellen. Diese ermöglichen es dem Vorstand, unterschiedliche Spendenbeträge, Aufwandsstunden für Aufgaben oder Eventteilnahmen in Punkte umzuwandeln und deren Auswirkungen auf den Gesamtpunktestand eines Mitgliedes zu analysieren.

In der Excel-Tabelle ist dies in Form von Formeln implementiert, die in Abhängigkeit von den Eingabewerten automatisch den Punktestand berechnen. Diese Funktion unterstützt den Vorstand bei der Entscheidungsfindung und verhindert fehleranfällige manuelle Berechnungen.

Punktebeispiel pro Jahr (aktives Mitglied)		
Quelle	**Anzahl**	**Punkte**
Spenden (CHF)	30	**150**
Aufgabe (h)	3	**90**
Event-Teilnahme	4	**40**
Umfrage	1	**10**
Mitglieder-Rekrutiertung	1	**150**
Gesamt-Punktzahl		**440**

Abb. 17: Punkterechner-Excel - Beispiel einer jährlichen Punktevergaben an ein aktives Mitglied

4.8.3 Belohnungsgrenzen

Die Excel-Vorlage unterstützt den Vorstand bei der Berechnung von Belohnungsgrenzen und stellt sicher, dass die Kosten für Belohnungen in einem für den Verein tragbaren Rahmen bleiben. Diese Funktion basiert auf einer strukturierten Formelarchitektur, welche transparent die Berechnung der Kosten, Punktewert und Aufwand darstellt.

Berechnung der Punkte-Werte für Belohnungen im Verein					
	Realwert (CHF)	**Ziel-Kostenanteil**	**Zielkosten (CHF)**	**Punktewert**	**Aufwand (h)**
Beschreibung	Wie viel kostet diese Belohnung?	Wie viel Prozent einer Belohnung übernimmt der Verein?	Wie viel der Kosten für eine Belohung übernimmt der Verein?	Wie viele Punkte sollte diese Belohnung mind. kosten?	Wie viele Arbeits-Stunden ist diese Belohnung wert?
Formel	--	--	Zielkosten = Realwert * Ziel-Kostenanteil	Punktewert = Realwert * (Punkte-pro-CHF / Ziel-Kostenanteil)	Stunden = Punktewert / Punkte-pro-h
Belohnung	**Realwert (CHF)**	**Ziel-Kostenanteil (%)**	**Ziel-Kostenanteil (CHF)**	**Punktewert**	**Aufwand (h)**
Rabattcode Vereins-Shop	CHF 5.00	50	**CHF 2.50**	**150**	**5**
Vereins-Merch	CHF 20.00	50	**CHF 10.00**	**600**	**20**
Vereins-Mitgliedschaft	CHF 50.00	50	**CHF 25.00**	**1500**	**50**

Abb. 18: Punkterechner-Excel - Beispiel einer Definition von Belohnungsgrenzen

4.8.3.1 Zielkosten des Vereins

Die Zielkosten definieren, welchen prozentualen Kostenanteil einer Belohnung vom Verein tatsächlich getragen wird und werden folgendermassen berechnet:

$$\text{Zielkosten (CHF)} = \text{Realwert (CHF)} * \text{Ziel-Kostenanteil (\%)} * 0.01$$

Durch diese Berechnung kann der Verein flexibel steuern, wie stark die Beteiligung an den Kosten für die einzelnen Belohnungen sein soll. Ein höherer Ziel-Kostenanteil (z. B. 50 %) bedeutet, dass das Mitglied bereits durch Spenden oder Arbeitsstunden einen Teil der Kosten „finanziert“ hat und somit die Ausgaben des Vereins reduziert werden.

Durch diese Berechnung kann der Verein flexibel festlegen, wie stark die Beteiligung an den Kosten einer Belohnung sein soll. Ein Ziel-Kostenanteil von beispielsweise 50 % bedeutet, dass der Verein die Hälfte der Belohnungskosten selbst trägt. Der verbleibende Teil wird durch eine höhere Punkteanforderung an das Mitglied ausgeglichen. Somit kann der Verein Belohnungen fair finanzieren, ohne das Budget unverhältnismässig zu belasten.

4.8.3.2 Punktekosten

Der Punktewert definiert, wie viele Punkte ein Mitglied mindestens sammeln muss, um die gewünschte Belohnung zu erhalten. Die Berechnung sieht wie folgt aus:

$$\text{Punktewert} = \text{Realwert (CHF)} * \left(\frac{\text{Punkte-pro-CHF}}{\text{Ziel-Kostenanteil (\%)} * 0.01} \right)$$

Mit dieser Formel wird der Ziel-Kostenanteil des Vereins proportional auf die Punkteanforderung des Mitglieds ermittelt. Das bedeutet je geringer der Vereinsbeitrag (Ziel-Kostenanteil), desto mehr Punkte muss das Mitglied sammeln, um die volle Belohnung zu erhalten. Dadurch wird sichergestellt, dass das Mitglied die Kostenbeteiligung des Vereins ausgleicht und eine faire Kostenverteilung entsteht.

4.8.3.3 Aufwandskosten

Zur Berechnung des für eine Belohnung erforderlichen Arbeitsaufwands in Stunden wird der Punktewert durch die Punkte-pro-Stunde-Rate geteilt:

$$\text{Arbeitswert (h)} = \frac{\text{Punktewert}}{\text{Punkte-pro-h}}$$

Diese Formel erlaubt es, Belohnungen auch auf Basis von Arbeitszeit zu quantifizieren und damit eine faire und nachvollziehbare Punktevergabe zu gestalten. Der Vorstand kann so den Aufwand in Stunden den Mitgliedern gegenüber transparent darstellen und den tatsächlichen Wert der Belohnung in Relation zum Engagement setzen.

4.8.4 Planung und Vergleich

Die Vorlage ermöglicht einen Vergleich zwischen Plan- und Ist-Werten der Punktevergabe. Dies umfasst beispielsweise geplante Spenden oder Eventteilnahmen im Vergleich zu tatsächlich dokumentierten Transaktionen in der Datenbank.

Durch diese Funktionalität kann der Vorstand die Effektivität der definierten Punkteanreize überprüfen und gegebenenfalls Anpassungen am Punktesystem vornehmen.

Berechnung der jährlichen Kosten durch das Punktesystem						
	Anzahl	CHF-pro-Spende	Punkte-pro-CHF	Max. Punkte pro Jahr	Vereins-Einnahmen durch alle Spenden (CHF)	
Spenden pro Jahr	5	50	5	1250	CHF 83.33	
	Anzahl	Stunden-pro-Aufgabe	Punkte-pro-h	Max. Punkte pro Jahr	Vereins-Ausgaben für alle Aufgaben (Belohnungswert ohne Ziel-Kostenanteil)	Belohnungswert mit 50% Vereins-Kostenanteil
Aufgaben pro Jahr	30	3	30	2700	-CHF 180.00	-CHF 90.00
	Anzahl	Teilnehmer-pro-Event	Punkte-pro-Event	Max. Punkte pro Jahr	Vereins-Ausgaben für alle Events (Belohnungswert ohne Ziel-Kostenanteil)	Belohnungswert mit 50% Vereins-Kostenanteil
Events pro Jahr	8	15	10	1200	-CHF 80.00	-CHF 40.00
Weitere Aktivitäten pro Jahr	Anzahl	Teilnehmer-pro-Aktivität	Punkte-pro-Aktivität	Max. Punkte pro Jahr	Vereins-Ausgaben für alle Aktivitäten (Belohnungswert ohne Ziel-Kostenanteil)	Belohnungswert mit 50% Vereins-Kostenanteil
Umfragen	2	15	10	300	-CHF 20.00	-CHF 10.00
Mitglieder-Rekrutierung	1	2	150	300	-CHF 20.00	-CHF 10.00
					Umsatz durch Punktesystem pro Jahr (Belohnungswert ohne Ziel-Kostenanteil)	Umsatz durch Punktesystem pro Jahr (Belohnungswert mit 50% Vereins-Kostenanteil)
Gesamter Vereins-Umsatz					-CHF 216.67	-CHF 66.67

Abb. 19: Punkterechner-Excel - Beispiel einer Jahresplanung des Punktesystems

4.9 Evaluierung und Weiterentwicklung

Die Metaanalyse von Looyestyn et al. (2017) zeigt, dass langfristige Untersuchungen nötig sind, um die nachhaltige Wirkung von Gamification zu bewerten. Somit sind regelmässige Evaluierungen notwendig, um die Effektivität des Systems sicherzustellen. Neben Mitgliederbefragungen zur Akzeptanz sollten auch Datenanalysen zur Untersuchung der Punktesammlungs-Dynamik durchgeführt werden. Feedbackgespräche mit dem Vorstand helfen, notwendige Anpassungen vorzunehmen. Eine jährliche Anpassung der Excel-Vorlage wird empfohlen, um die Funktionsweise des Systems an aktuelle Entwicklungen anzupassen. (Looyestyn et al., 2017)

5 Systemanalyse und Architektur

In diesem Kapitel werden die grundlegenden Strukturen und technischen Konzepte des Systems beschrieben. Zunächst wird anhand einer Domänenanalyse modelliert, wie die organisatorischen Abläufe, Rollen und Funktionen eines Vereins digital abgebildet werden können. Darauf aufbauend folgt die Darstellung der Systemarchitektur nach dem C4-Modell.

5.1 Domain Analyse

Im Rahmen der Domänenanalyse wird in diesem Kapitel das Anwendungskonzept eines Discord-Bots zur Vereinsverwaltung und Gamification modelliert (siehe Abb. 20). Ziel ist es, sowohl organisatorische Abläufe als auch Mitgliederbeteiligung durch ein Punktesystem zu unterstützen.

Zentrale Rolle im Modell ist das Vereinsmitglied (eine Obergruppe von Mitglied und Vorstand), welche von Person abgeleitet ist. Diese Struktur ermöglicht die Abgrenzung zwischen Nicht-Mitgliedern und Vereinsmitgliedern. Zusätzlich können Mitglieder in den Vorstand gewählt werden und erhalten dadurch zusätzliche Verantwortlichkeiten und Rechte.

Aufgaben dienen als Hauptquelle für Punkte und sind potenziell an Events gebunden, können aber auch andere Aktivitäten umfassen (z.B. Spenden, Mitgliederrekrutierung). Nur Vereinsmitglieder mit Discord-Zugang können Aufgaben digital als erledigt markieren und dadurch automatisch Punkte erhalten. Diese Einschränkung ergibt sich aus der technischen Basis des Systems, das auf Discord-Interaktionen basiert.

Alle Vereinsmitglieder besitzen ein Punktekonto, unabhängig von Discord. Punkte können auch manuell durch Vorstandsmitglieder vergeben werden, beispielsweise für Mithilfe bei Events oder durch Spenden. Auch das Einlösen von Punkten für Belohnungen erfolgt manuell über Discord. Somit wird sichergestellt, dass Mitglieder, die kein Discord benutzten aktiv eingebunden bleiben, um der Gefahr einer Demotivation durch Plattform-Zwang entgegenzuwirken. Eine Rangliste zeigt den Punktestand aller Mitglieder. Derzeit können nur Discord-Nutzer ihren Rang direkt einsehen. Ein zukünftiges, plattformunabhängiges Dashboard könnte hierbei für mehr Transparenz sorgen (siehe Abschnitt 8.3). Vorstandsmitglieder können ebenfalls Punkte für die Planung von Events oder andere organisatorische Aufgaben erhalten, um Engagement und Motivation innerhalb des Vorstandes zu fördern.

Das nachfolgende Klassendiagramm bildet diese Konzepte detailliert ab, stellt die Beziehungen zwischen den Entitäten transparent dar und dient als Grundlage für die technische Implementierung.

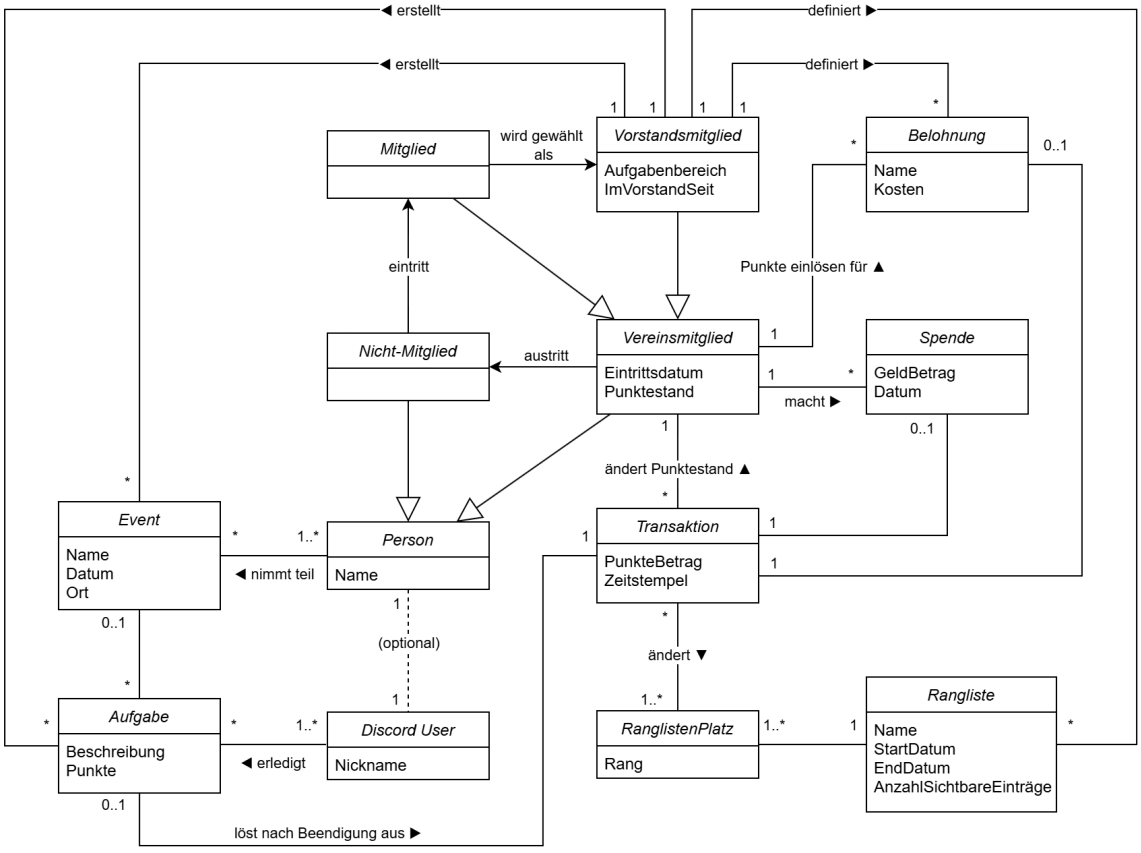


Abb. 20: Domain Model

5.2 C4-Modell

Die Architektur wurde mit dem C4-Modell ausgearbeitet, um eine klare Struktur zu gewährleisten. Bei der Konzeption wurde darauf geachtet, dass der Kommunikationskanal (aktuell Discord) in der Zukunft auch durch andere Clients, wie Microsoft Teams oder ein Web-Dashboard ersetzt werden könnte. Auch der Kalender Provider kann in der Zukunft ausgetauscht werden, aktuell handelt es sich um den Google Calendar. Im Folgenden wird der aktuelle Stand der Architektur beschrieben mit Discord als Kommunikationsclient und Google Calendar als Kalenderprovider.

5.2.1 System Context Diagramm

Das System Context Diagramm zeigt die Gesamtübersicht über die Systeme und deren Nutzer. Nutzer können mit dem Discord-Bot System über einen Discord-Server interagieren. Die Interaktion wird im Container und Component Diagramm genauer beschrieben.

Es gibt drei Arten von Akteuren:
- **Nicht-Mitglied**: Nutzer die teil des Discord-Servers sind, jedoch keine Vereinsmitglieder.
- **Mitglied**: Nutzer die Mitglieder des Discord-Servers und des Vereins sind.
- **Vorstandsmitglied**: Mitglieder des Vereinsvorstandes.

Diese Nutzer interagieren alle mit dem Discord-Server, welcher mit dem Discord-Bot System interagiert. Dieses hat zusätzlich die Verbindung zum Kalenderprovider über welchen es Events hinzufügen und abfragen kann.

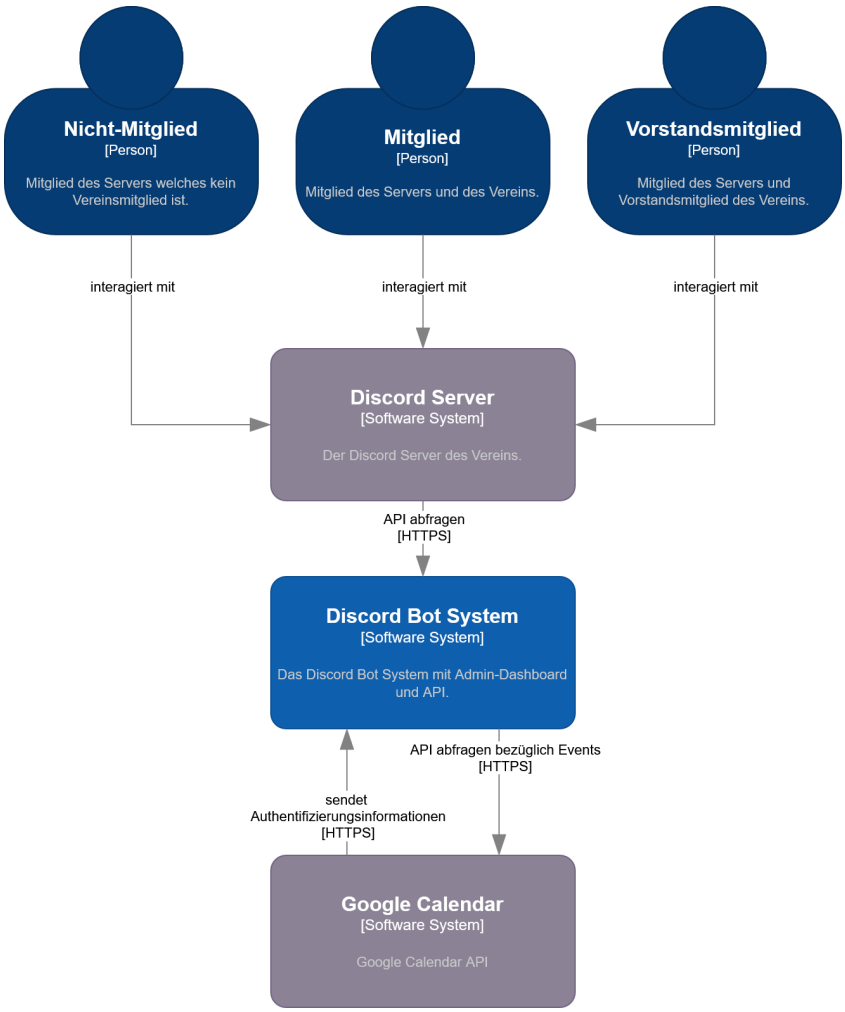


Abb. 21: System Context Diagramm

5.2.2 Container Diagramm

Das Container Diagramm zeigt die verschiedenen Systeme im Discord-Bot System.

Die API Applikation stellt die zentrale Logik der Applikation dar. Sie bietet Endpunkte für die Punkteverwaltung, Event- und Benutzerinformationen. Die API kommuniziert mit dem Kalenderprovider, um Events zu speichern oder abzufragen.

Zudem interagiert die API Applikation mit der Datenbank für die Datenverwaltung.

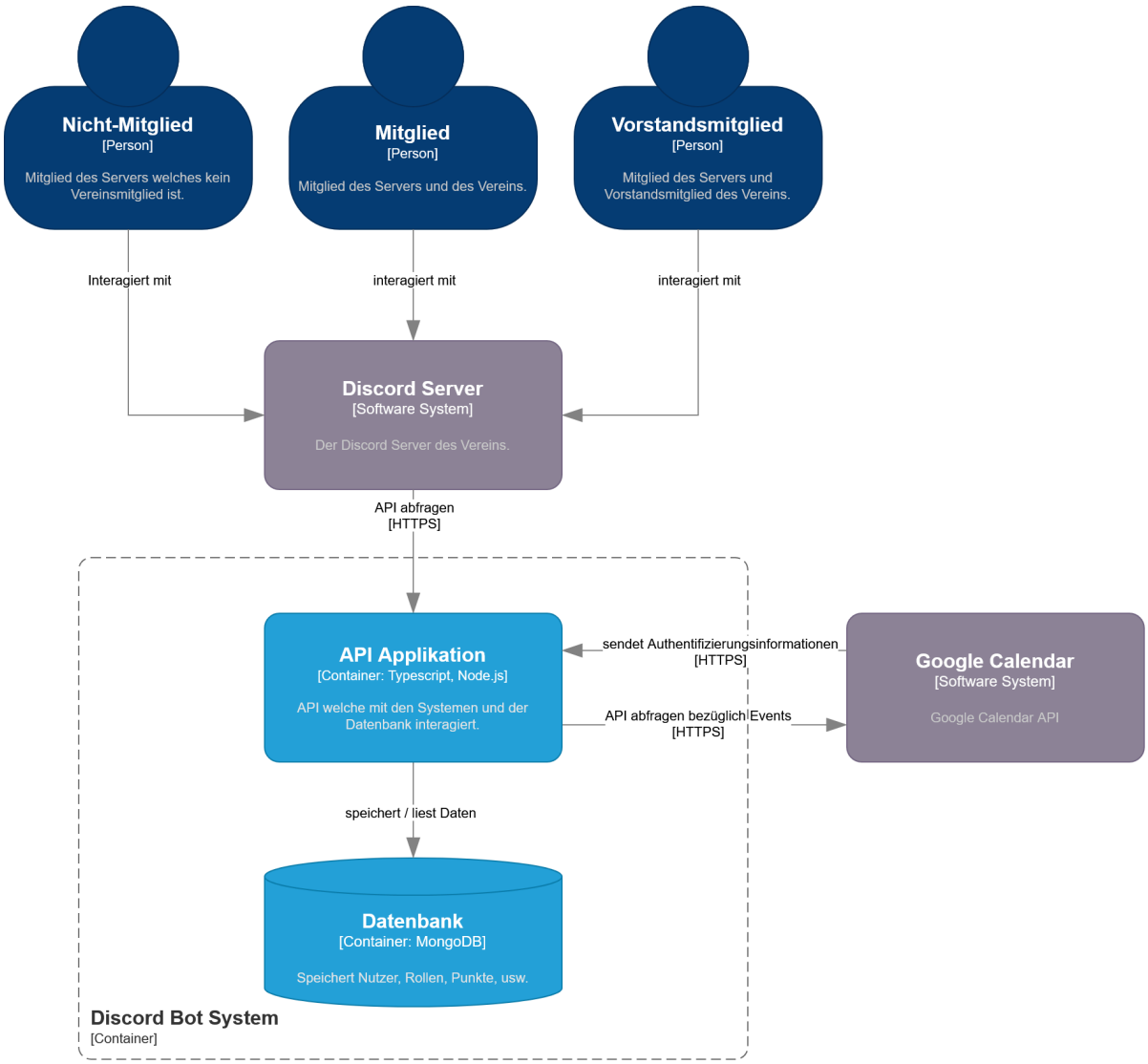


Abb. 22: Container Diagramm

5.2.3 Component Diagramm

Die Applikation ist in verschiedene Komponenten unterteil:

- Der **Discord-Bot**-Container ist verantwortlich für die Anbindung an den Discord-Server. Der Container empfängt eingehende Anfragen und verarbeitet diese entsprechend.
- Der **Commands**-Container enthält die Logik für alle Befehle, die Nutzer über den Discord-Server auslösen können. Die Commands sind Discord spezifisch und nicht plattformneutral implementiert.
- **Gamification, User und Event** bilden die zentrale Geschäftslogik. Sie enthalten Funktionen zur Punktevergabe, Bewerbungs- und Eventmanagement. Diese Komponenten sind plattformunabhängig und haben daher keine Abhängigkeiten zu Discord.
- Die **Datenbank-Komponente** dient als Abstraktionsschicht zur Speicherung und zum Zugriff der Daten in der Datenbank. Dadurch kann die Geschäftslogik von der Datenbanktechnologie getrennt werden. Somit kann die Datenbanktechnologie ausgetauscht werden, ohne das Änderungen in der zentralen Geschäftslogik nötig sind.

- Die **Errors-Komponente** vereinheitlicht das Fehler-Handling und stellt zentrale Fehlertypen bereit. Das Fehler-Handling wurde als eigene Komponente konzipiert, um eine einheitliche und nachvollziehbare Behandlung von Fehlern im gesamten System zu gewährleisten. Durch die Nutzung eines zentralen Result-Typs werden Fehler explizit behandelt und nicht als versteckte Ausnahme zurückgegeben. Damit wird die Robustheit und Wartbarkeit des Systems sichergestellt.
- Die **Kalender-Komponente** agiert als Interface zur Integration von Kalenderdiensten. Aktuell ist hier die Verbindung zum Google Calendar implementiert. Die Architektur lässt jedoch auch die Implementation von anderen Diensten wie Outlook zu. Die Zugriffe von der Geschäftslogik erfolgen über definierte Schnittstellen, sodass die Implementierung bei Bedarf ausgetauscht werden kann.
- Der **Webserver** stellt einen HTTP-Endpunkt bereit, über den die Authentifizierungsinformation des Kalenderanbieters entgegengenommen werden können.

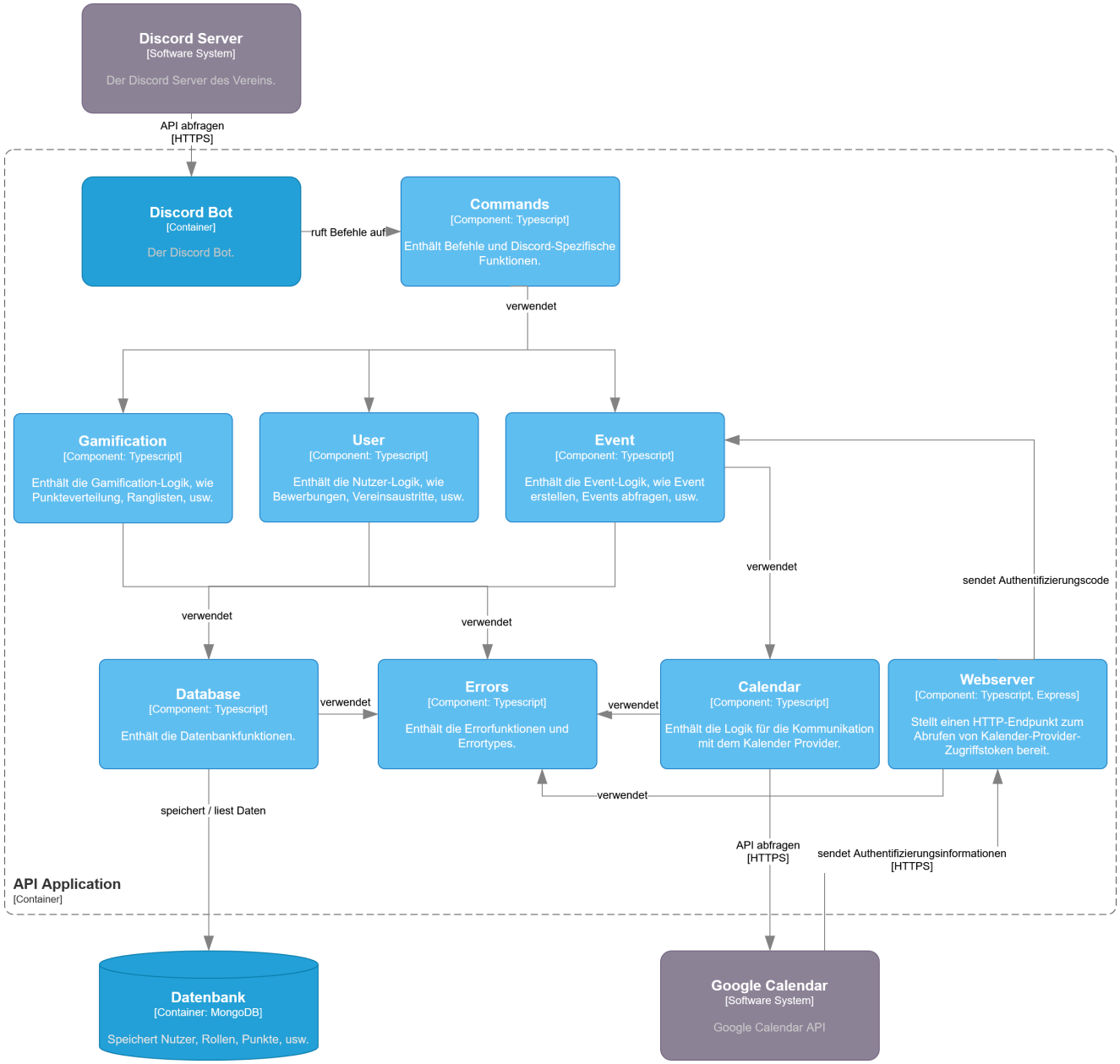


Abb. 23: Component Diagramm

Die konkrete Auswahl der eingesetzten Technologien für die Umsetzung der Architektur, sowie die Begründungen dieser Entscheidungen werden in Abschnitt 11 beschrieben.

6 Qualitätssicherung

Die Qualitätssicherung spielt eine entscheidende Rolle um eine stabile, sichere und fehlerfreie Anwendung bereitzustellen. In diesem Kapitel werden die Teststrategien beschrieben, sowie erläutert wie die Code-Qualität sichergestellt werden kann. Ziel ist es durch Tests und Überprüfungen eine hohe Softwarequalität zu gewährleisten.

6.1 Testkonzept

Das Testkonzept beschreibt die Vorgehensweise zur Sicherstellung der Software-Qualität für den Discord-Bot. Es legt fest, welche Tests durchgeführt werden, wie diese organisiert und dokumentiert werden und welche Tools und Umgebungen dabei zum Einsatz kommen. Ziel ist es, die Einhaltung der funktionalen und nicht-funktionalen Anforderungen sicherzustellen und eine fehlerfreie Anwendung zu gewährleisten.

6.1.1 Vorgehensweise

Um den Erfolg bei dem Testen sicherzustellen ist es wichtig strukturiert vorzugehen. Die Vorgehensweise lautet wie folgt:

1. Mittels der Requirements Tests definieren.
2. Entscheiden wie die Tests durchgeführt werden.
3. Tests durchführen.
4. Fehler und Erfolge dokumentieren.
5. Aufgefallene Fehler, die während den Testen auftauchen, beheben.
6. Tests erneut durchführen.

6.1.2 Testziele

Folgende Funktionen und Elemente sollen getestet werden:

- Alle Use Cases
- Usability
- Security
- Maintainability
- Compatibility
- Reliability

Die Ziele wurden aus den Use Cases (siehe Abschnitt 2.5) und den Non-Functional Requirements (siehe Abschnitt 2.6) übernommen.

6.1.3 Testumgebung

Die Tests werden in der folgend beschriebenen Umgebung durchgeführt:

Client:

Betriebssystem	Windows 11
Browser	Google Chrome, Mozilla Firefox
Plattform	Discord

Tab. 49: Testumgebung Client

Server:

Betriebssystem	Debian GNU/Linux 12 (bookworm)
Typescript	Version 5.8.2
Node.js	Version 23.0.0
Datenbank	MongoDB 8.0.5

Tab. 50: Testumgebung Server

6.1.4 Usability-Tests

Um die Funktionalität und Sicherheit des Bots zu prüfen, werden Usability-Tests mit mehreren Personen durchgeführt. Die Tester werden aufgefordert, verschiedene Funktionen und Abläufe durchzuführen. Die Tests werden mithilfe eines Formulars durchgeführt, welches den Testern die Aufgaben beschreibt. Diese können sie nach der Durchführung bewerten. Bei der Bewertung wird die Klarheit der Durchführung sowie auch der Nutzen der Funktion bewertet. Zusätzlich werden die Tester am Ende aufgefordert, Sicherheitslücken im System zu finden.

6.1.4.1 Testplanung und Vorgehen

Der Usability-Test wurde auf Basis des zuvor entwickelten Usability-Testplans erstellt. Ziel war es, die Benutzerfreundlichkeit der wichtigsten Funktionen für unterschiedliche Benutzerrollen (Nichtmitglied, Mitglied, Vorstand) zu evaluieren. Im Fokus standen die intuitive Bedienbarkeit der Bot-Befehle, die Verständlichkeit der Nutzerinteraktionen sowie die visuelle Gestaltung und Feedbackmechanismen.

Die Tests wurden auf einem dedizierten Discord-Testserver durchgeführt, der dem produktiven Server nachempfunden war. Die Testpersonen nutzten Desktop-Clients (Browser und Discord-App). Jede Testsitzung dauerte etwa 30 Minuten bis maximal zwei Stunden, abhängig von der Benutzerrolle. Die Tests wurden nach dem Think-Aloud-Prinzip durchgeführt, wobei die Teilnehmer gebeten wurden, ihre Gedanken während der Nutzung laut auszusprechen. Dies erlaubte eine tiefere Einsicht in die Denkprozesse und mögliche Verständnisprobleme.

Für die Tests wurden drei verschiedene Zielgruppen berücksichtigt, mit je zwei Testpersonen:

- **Nichtmitglieder**: Personen, die Discord kennen, aber keine Vereinsmitglieder sind.
- **Mitglieder**: Aktive Vereinsmitglieder.
- **Vorstand**: Vorstandsmitglieder oder Personen mit administrativen Aufgaben.

Die Testaufgaben wurden spezifisch auf die Benutzerrollen abgestimmt (vgl. Anhang „Usability Test - Plan.pdf“) und umfassen beispielsweise:

- **Für Nichtmitglieder**: Beitritt zum Discord-Server, Navigation, Ausfüllen der Beitrittsbewerbung.
- **Zusätzlich für Mitglieder**: Anmeldung zu Events, Aufgabenmanagement, Punkteübersicht und -einlösung.
- **Zusätzlich für Vorstandsmitglieder**: Bewerbungen prüfen, Events sowie Ranglisten erstellen und Spenden dokumentieren.

Die Auswertung und die Ergebnisse der Tests werden im Anhang im Abschnitt Testreports dargestellt.

6.1.5 Ablaufplan

Der Ablaufplan beschreibt die verschiedenen Testphasen und deren Durchführung. Die Ergebnisse der Tests liefern wertvolle Informationen zur Stabilität, Funktionalität und Benutzerfreundlichkeit des Systems.

ID	**T1**
Name	Unit-Tests
Beschreibung	Funktionen im Backend werden mit Unit-Tests getestet. Die Tests werden nach jedem Commit automatisch durch GitLab-Pipelines durchgeführt.
Methode	Automatisch
Relevant für	Use-Cases, Functional Requirements
Testumgebung	Client und Server
Intervall	Bei jedem Commit

Tab. 51: Ablauf Unit-Tests

ID	T2
Name	Integration-Tests
Beschreibung	Bei den Integration-Tests wird überprüft, ob die Verbindungen der verschiedenen Systeme (Discord, Backend, Datenbank) richtig funktionieren. Der Test ist erfolgreich bestanden wenn mindestens 95% der Befehle innerhalb von 2 Sekunden beantwortet werden.
Methode	Manuell
Relevant für	Maintainability, Requirements
Testumgebung	Client und Server
Intervall	Am Ende

Tab. 52: Ablauf Integration-Tests

ID	T3
Name	System-Tests
Beschreibung	Am Ende wird für das gesamte System überprüft, ob alle funktionalen und nicht-funktionalen Requirements erfolgreich umgesetzt worden sind.
Methode	Manuell
Relevant für	Alle Testziele
Testumgebung	Client und Server
Intervall	Am Ende

Tab. 53: Ablauf System-Tests

ID	T4
Name	Usability-Tests
Beschreibung	Der Bot wird, wie im Abschnitt Usability-Test genauer beschrieben, von mehreren Testern getestet.
Methode	Manuell
Relevant für	Usability, Security, Requirements, Reliability
Testumgebung	Client und Server
Intervall	Gegen Ende der Konstruktionsphase

Tab. 54: Ablauf Usability-Test

6.2 Code-Qualität

Um die Code-Qualität sicherzustellen wird das Vier-Augen-Prinzip angewendet. Bei jedem Meeting wird der geschriebene Code von den Teammitgliedern überprüft bevor er in den Main-Branch gemerged wird. Weitere Informationen zu Code-Qualität und Gitlab können im Abschnitt 11 gefunden werden.

Um die Einheitlichkeit des Codes sicherzustellen wird ein Linter verwendet. Mehr Informationen dazu können im Code Setup gefunden werden.

6.3 Definition of Done

Damit die Use-Cases als abgeschlossen gelten, müssen folgende Kriterien erfüllt sein:

- Alle zugehörigen Tests wurden erfolgreich durchgeführt.
- Die Testergebnisse sind vollständig dokumentiert.
- Die Coding Guidelines wurden eingehalten (siehe Abschnitt 11.2.1).
- Die Umsetzung entspricht den in den Non-Functional Requirements festgelegten Vorgaben (siehe Abschnitt 2.6).
- Die Funktionalität wurde von dem Projektteam abgenommen und für den Einsatz freigegeben.

7 Implementation

In diesem Kapitel wird die konkrete technische Umsetzung des Discord-Bots beschrieben. Ausgehend von den funktionalen Anforderungen in Abschnitt 2 sowie dem Architekturkonzept in Abschnitt 5.2 , werden die zentralen Komponenten wie Datenbankmodell, Bot-Befehle, Kalenderintegration, Workflows und Gamification-Funktionen detailliert erläutert. Der Fokus liegt auf modularer Struktur, nachvollziehbarer Datenverarbeitung und sicherer Schnittstellennutzung. Zudem werden ausgewählte Implementierungsdetails anhand von Codebeispielen illustriert, um die technische Tiefe und Qualität der Lösung zu belegen.

7.1 Datenbank

Das logische Datenmodell (vgl. Abb. 24) bildet die in der Domänenanalyse (vgl. Abschnitt 5.1) identifizierten Konzepte formal ab und orientiert sich an den im Sicherheitskonzept (vgl. Abschnitt 3) definierten Prinzipien. Es werden ausschliesslich Informationen gespeichert, die für die Umsetzung des Gamification-Systems sowie die Verwaltung des Vereins notwendig sind. Das Modell unterstützt dabei die in Abschnitt 2.5 und Abschnitt 2.6 beschriebenen Use Cases und Non-Functional Requirements.

7.1.1 Wahl des Modellierungsansatzes

Für die Modellierung wurde bewusst das UML-Klassendiagramm gewählt, da es eine objektorientierte Sichtweise auf die Datenstruktur ermöglicht und sich dadurch nahtlos in die geplante technische Umsetzung mit TypeScript integrieren lässt. Im Vergleich dazu bietet das klassische Entity-Relationship-Modell (ERM) eine gute Grundlage für die relationale Modellierung, ist jedoch weniger geeignet für dokumentenorientierte Datenbanken wie MongoDB. UML unterstützt nicht nur die Darstellung von Entitäten, deren Attribute sowie 1:N- und N:M-Beziehungen , sondern erlaubt auch die Modellierung verschachtelter Strukturen, Embedded Documents und optionaler Felder, wie sie für MongoDB typisch sind. Darüber hinaus kann UML direkt in objektorientierten Code (TypeScript-Klassen und Interfaces) übersetzt werden, wodurch eine konsistente Brücke zwischen Design und Implementierung entsteht. Damit eignet sich UML ideal für die Umsetzung der Use Cases im Backend und fördert gleichzeitig eine modulare, flexible und objektorientierte Architektur.

7.1.2 Modellierung der Entitäten

Die zentrale Entität Person bildet die Grundlage für die Mitgliederverwaltung (vgl. UC1 , UC2). Neben den Stammdaten (Name, Kontaktinformationen, Geburtsdatum) enthält sie auch die Discord-spezifischen Attribute wie Discord-ID, Nickname und Rollen. Der Punktestand wird über die Entität Transaktion verwaltet, die jede Punktebewegung dokumentiert und mit einem Statusattribut (z. B. „Ausstehend“, „Erledigt“, „wirdGelöscht“) versehen ist (vgl. UC8 , UC9). Die Historisierung wird somit dezentral über Transaktionen sichergestellt. Diese Struktur unterstützt auch die Anforderungen an Sicherheit und Manipulationsschutz (NRF7 , NRF8).

Die Entitäten Aufgabe und Event sind als eigenständige Auslöser für Gamification-Aktivitäten modelliert (vgl. UC4 , UC5). Beide sind über Zwischentabellen (AufgabenTeilnehmer, EventTeilnehmer) mit den jeweiligen Teilnehmern (Personen) verknüpft. Diese Modellierung erlaubt eine saubere Abbildung von one-to-many -Beziehungen und gewährleistet die notwendige Flexibilität für wiederkehrende Events und individuelle Aufgabenbearbeitungen (vgl. UC8).

Das Modell berücksichtigt Discord-spezifische Anforderungen wie Rollenmanagement durch die Entität Rolle, welche Discord-Rollen samt Farbe, Position und Berechtigungen abbildet (vgl. UC2). Die Zwischentabelle NutzerRolle verwaltet die Mehrfachzuordnung von Rollen zu Personen. Diese Abbildung erleichtert die Synchronisation mit Discord und ermöglicht die dynamische Steuerung der Zugriffskontrolle (vgl. UC1).

Die Entität JoinDataTemp dient der temporären Zwischenspeicherung von Daten des in zwei Teil-Schritte aufgeteilten Bewerbungsformulars und werden gelöscht, sobald der Bewerbungsprozess abgeschlossen ist sowie der Datenschutz akzeptiert wurde (vgl. UC1 , UC6). Die Daten werden nach 30 Minuten automatisch gelöscht, um den Datenschutzanforderungen (NRF7) zu genügen. Eine ausführlichere technische Beschreibung erfolgt in Abschnitt 7.4 .

Transaktionen bilden die Grundlage des Punktesystems (vgl. UC8 , UC9). Jede Transaktion ist einer Person zugeordnet und optional mit einer Aufgabe, einem Event oder einer Belohnung verknüpft. Spenden werden ebenfalls als Transaktionen abgebildet, werden jedoch einzig durch manuelles Hinzufügen eines Vorstandsmitgliedes beziehungsweise nach der Bestätigung des Kassiers ausgelöst, um Missbrauch zu verhindern (vgl. UC9). Belohnungen für Punkte werden in der Entität Belohnung definiert, welche Name, Beschreibung und PunkteKosten enthält. Sobald ein Mitglied eine Belohnung einlöst,

wird eine neue Transaktion erstellt, die den Punkteabzug dokumentiert (vgl. UC10). Diese Architektur stellt sicher, dass alle Punktebewegungen revisionssicher und konsistent gespeichert werden und dass der Punktestand jederzeit korrekt ermittelt werden kann. Die Statusattribute der Transaktionen sind als Enumeration modelliert, was eine klare und konsistente Statusverwaltung erlaubt und Manipulationsschutz bietet (NRF8).

Ranglisten werden über die Entität Rangliste realisiert und enthalten Metadaten wie Name, Start- und Enddatum sowie die Anzahl sichtbarer Einträge (vgl. UC8). Die Entität RanglistenEintrag ordnet eine Person einer Rangliste zu und speichert den aktuellen Punktestand. Die Historisierung wird dabei bewusst nicht direkt in der Rangliste abgebildet, sondern bleibt über die Transaktionen rekonstruierbar. Diese Entkopplung reduziert die Komplexität und verbessert die Performance (NRF3).

Aus diesem logischen Modell wurde anschliessend ein relationales Modell abgeleitet, um die Implementierung der Datenstruktur in Code zu übertragen. Das relationale Modell ist im Anhang dokumentiert und dient als zusätzliche Referenz für eine alternative Datenbanksicht (vgl. Anhang „Relationales Datenmodell.sql“).

7.1.3 Typensicherheit und Datenoperationen

Um eine konsistente und sichere Handhabung der Daten zu gewährleisten (NRF1 , NRF3), wurde bei der Modellierung auf TypeScript-Funktionalitäten wie `Omit` zurückgegriffen. Ein Beispiel ist der Typ `RoleInput`, der aus dem Interface `IRole` abgeleitet wird, jedoch das `_id`-Feld explizit ausschliesst. Dies verhindert, dass MongoDB-IDs manuell gesetzt werden können, und sorgt für eine saubere und sichere Schnittstelle zwischen Backend und Datenbank. (vgl. UC11)

```
const roleSchema = new mongoose.Schema({
    name: { type: String, required: true },
    discordIcon: { type: Buffer<ArrayBufferLike>, default: null },
    discordEmoji: { type: String, default: null },
    discordColor: { type: String, default: null },
    discordPosition: { type: Number, required: true },
    discordPermissions: { type: String, default: null },
    hoist: { type: Boolean, default: false },
    mentionable: { type: Boolean, default: false },
});

export interface IRole {
    _id: mongoose.Types.ObjectId;
    name: string;
    discordIcon?: Buffer<ArrayBufferLike> | null;
    unicodeEmoji?: string | null;
    discordColor?: string | null;
    discordPosition: number;
    discordPermissions?: string | null;
    hoist?: boolean;
    mentionable?: boolean;
}

export type RoleInput = Omit<IRole, "_id">;

export const Role = mongoose.model<IRole>("Role", roleSchema);
```

Listing 1: Model Datenspeicherung von Rollen inklusive der Verwendung von Omit zur Typensicherheit

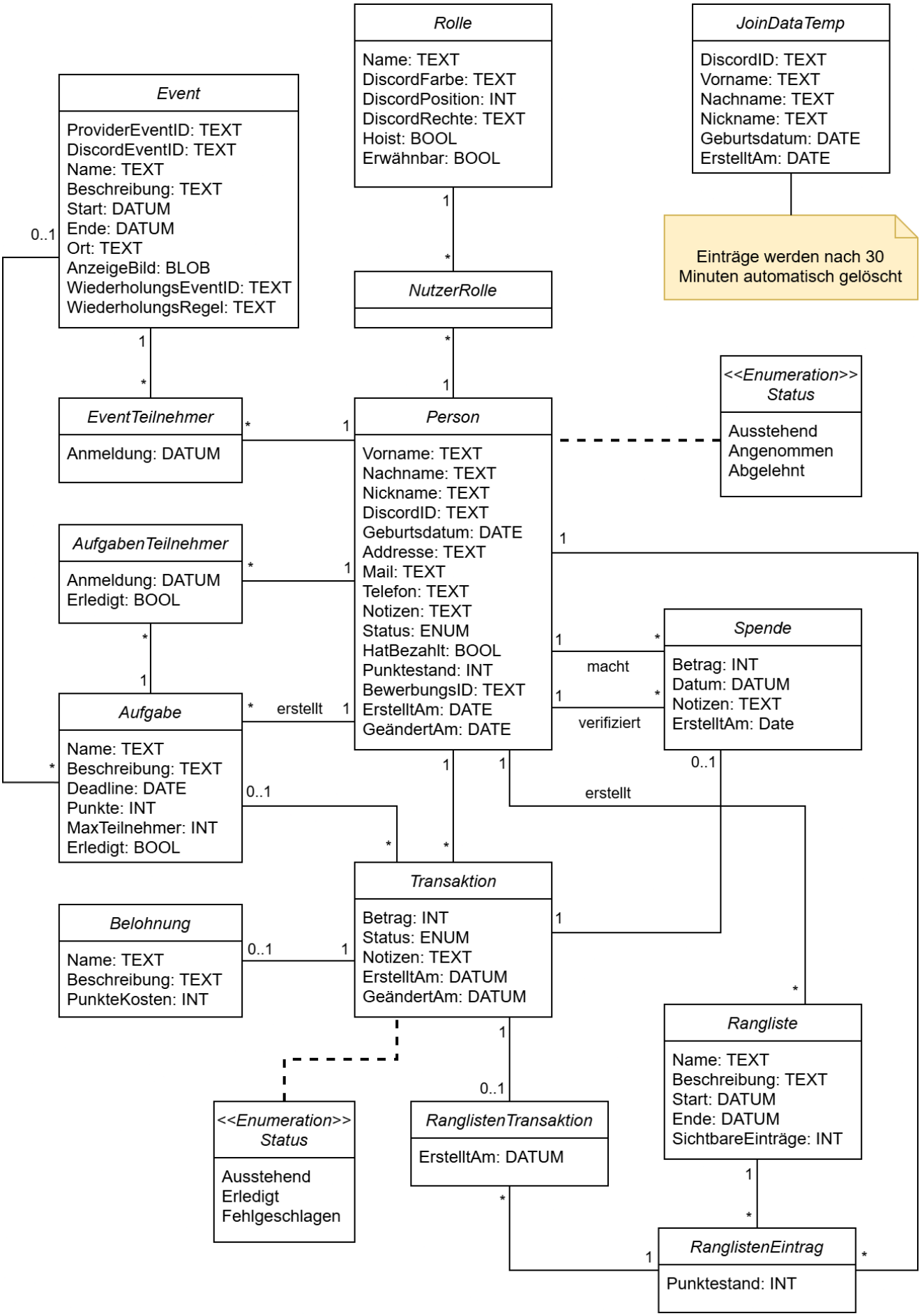


Abb. 24: Logisches Datenmodell

7.2 Backups

Die technische Umsetzung des Backup-Systems basiert auf den Anforderungen und Strategien, die im Sicherheitskonzept unter Abschnitt 3.6.2 beschrieben wurden. Das Backup-System wurde in diesem Projekt durch ein automatisiertes Verfahren realisiert. Ein dedizierter Docker-Service `backup` (vgl. `docker-compose.yml`) erzeugt täglich ein neues Backup der MongoDB-Datenbank. Diese Backups werden standardmässig im Verzeichnis `./data/backup/` gespeichert und nach sieben Tagen automatisch gelöscht, um Speicherplatz effizient zu nutzen. Optional werden die Backups mit GPG verschlüsselt, falls das Verschlüsselungskennwort (`GPG_PW`) in der `.env`-Datei hinterlegt ist.

Für zusätzliche Flexibilität wurde ein manuelles Backup-System implementiert, welches sowohl für Linux/macOS als auch für Windows verfügbar ist. Die Skripte `backup.sh` (Linux/macOS) und `backup.ps1` (Windows) erlauben es dem Vorstand oder einem Server-Administrator, Backups jederzeit on-demand zu erstellen. Diese Skripte nutzen `mongodump`, archivieren die Daten im `.tar.gz`-Format und verschlüsseln sie optional mit GPG. Temporäre unverschlüsselte Dateien werden im Anschluss automatisch gelöscht, um Sicherheitsrisiken zu minimieren.

Die Wiederherstellung der Datenbank erfolgt ebenfalls über automatisierte Skripte: `restore.sh` (Linux/macOS) und `restore.ps1` (Windows). Diese entschlüsseln das Backup-Archiv, extrahieren die Daten und spielen sie über `mongorestore` in den Datenbank-Container ein. Damit ist eine vollständige Wiederherstellung der Datenbank möglich, ohne dass manuelle Konfigurationen erforderlich sind.

Für die korrekte Anwendung dieser Backup- und Restore-Prozesse steht eine detaillierte Anleitung für den Vorstand oder Server-Administrator im BACKUP-README des Projektes zur Verfügung (vgl. Anhang „BACKUP-README.pdf“). Zusätzlich ist eine entsprechende Beschreibung in der Dokumentation im Anhang dieser Arbeit enthalten, die den technischen Ablauf Schritt für Schritt erläutert. Die Implementierung dieses Backup-Systems gewährleistet somit eine regelmässige, sichere und verschlüsselte Sicherung der Datenbankinhalte des Discord-Bots. Gleichzeitig bietet es eine einfache und benutzerfreundliche Möglichkeit zur Wiederherstellung, was die Betriebssicherheit des Systems nachhaltig verbessert.

7.3 Bot-Befehle

Nutzer können durch Befehle auf Funktionen des Discord-Bots zugreifen. Diese Befehle werden als Slash-Befehle implementiert. Im gegensatz zu den alten Discord Befehlen, welche mit einem Ausrufezeichen markiert wurden, bieten Slash-Befehle Funktionen wie Autocomplete, Error-Handling und Berechtigungssteuerung. (kynthia, 2024)

7.3.1 Struktur

Die Befehle des Projektes sind Modular aufgebaut. Jeder Befehl befindet sich in einer separaten Datei und hat folgende Komponenten:

- **data**: Metadaten über den Befehl (Name, Beschreibung, Berechtigungen).
- **execute-Funktion**: Die Funktion welche beim auslösen des Befehls ausgeführt wird.

Im Folgenden ist beispielhaft die gekürzte Implementation des `/intro`-Befehls dargestellt. Zu Beginn wird `interaction.deferReply()` aufgerufen. Dies ist erforderlich, da Discord innerhalb von drei Sekunden eine Rückmeldung vom Bot erwartet (Discord.js-Contributors, 2025a) . Wenn ein Befehl voraussichtlich länger als drei Sekunden benötigt, muss er mit `interaction.deferReply()` gekennzeichnet werden. Discord zeigt in diesem Fall automatisch eine Ladeanzeige im Chat an.

Da der `/intro`-Befehl auf Discord-API-Funktionen zugreift, um beispielsweise den Kanalnamen für den Intro-Text anzupassen, ist ein solches Vorgehen aus Sicherheitsgründen sinnvoll, auch wenn der Befehl im Normalfall innerhalb des Zeitlimits ausgeführt wird. Abschliessend wird die automatisch generierte Ladeanzeige durch die finale Nachricht mit dem Intro-Text ersetzt.

```
  export const data = new SlashCommandBuilder()
    .setName("intro")
    .setDescription("Vorstand-Only: Stellt den Vereins-Bot vor.");

export async function handleIntro(
    interaction: CommandInteraction,
): Promise<void> {
    await interaction.deferReply();
    // Informationen für angepassten Text bereitstellen (hier entstehen Discord API
abfragen)
    let introText = "..."
    await interaction.editReply({ content: introText });
}
```

Listing 2: Implementierungs-Ausschnitt für den Discord-Slash-Befehl: `/intro`

Die Abb. 25 zeigt die Antwort des `/intro`-Befehls.

Abb. 25: Antwort auf den `/intro`-Befehl

7.3.2 Befehls-Registrierung

Um einen erstellten Befehl bei Discord sichtbar zu machen, muss dieser zuerst registriert werden. Es gibt zwei verschiedene Arten der Registrierung:

- **Global:** Globale Befehle gelten für alle Server, die den Bot aktiviert haben. Diese haben eine Verzögerung bei der Registrierung.
- **Pro-Server:** Pro-Server-Befehle gelten nur für spezifisch definierte Server. Hier gibt es keine Verzögerungen beim Registrieren.

Während der Entwicklung werden alle Befehle pro Server registriert, da diese dadurch ohne Verzögerung getestet werden können. Es muss zudem darauf geachtet werden, dass beim Hinzufügen eines neuen Befehls alle Befehle erneut registriert werden müssen, da Discord die Befehlsliste nicht aktualisiert, sondern überschreibt.

```
const rest = new REST({ version: "10" }).setToken(config.DISCORD_TOKEN);

export async function deployCommands(guildId: string) {
  await rest.put(
    Routes.applicationGuildCommands(config.DISCORD_CLIENT_ID, guildId),
    { body: Object.values(commands).map(cmd => cmd.data) }
  );
  console.log("Commands registered.");
}
```

Listing 3: Implementierungs-Ausschnitt der Registrierung der Discord-Slash-Befehle

7.3.3 Zugriffskontrolle der Discord-Befehle

Um die sicherheitskritischen Befehle vor unbefugtem Zugriff zu schützen, wurde ein Wrapper namens `withRoleAccess` implementiert. Dieser kontrolliert, ob die aufrufende Person über eine der erlaubten Rollen verfügt (z. B. „Vorstand“) und bricht andernfalls mit einer Fehlermeldung ab.

Die Prüfung selbst greift auf die `roles.cache`-Daten des Discord-Mitgliedsobjekts zu und vergleicht diese mit den erlaubten Rollennamen.

```
export function withRoleAccess(handler, allowedRoles) {
  return async function (interaction) {
    const ok = await checkRoleAccess(interaction, allowedRoles);
    if (!ok) return;
    await handler(interaction);
  };
}
```

Listing 4: Implementierungs-Ausschnitt des Wrappers für rollenbasierte Zugriffskontrolle von Discord-Befehlen

```
export const execute = withRoleAccess(handleGetDataCommand, [ "Vorstand", "Mitglied" ]);
```

Listing 5: Implementierungs-Ausschnitt der Absicherung eines Discord-Commands mittels rollenbasierter Zugriffskontrolle

7.3.4 Modals und Validierung

Für komplexere Interaktionen, die viele Nutzereingaben erfordern, werden Discord-Modals verwendet. Diese ermöglichen es, strukturierte Eingaben von Nutzern übersichtlich abzufragen. Pro Modal sind maximal fünf Eingabefelder möglich und auch die Validierung der Eingaben erfolgt serverseitig, da durch Einschränkungen seitens der Discord API eigene Input-Kontrollen nicht direkt im Modal implementiert werden können.

Aufgrund der Beschränkung auf fünf Eingabefeldern pro Modal wurden Interaktionen welche mehr als fünf Eingaben benötigten mit sogenannten Optionen gelöst. Diese funktionieren wie Parameter und können einfach hinter den Befehl geschrieben werden. Bei vielen Optionen wurde es jedoch sehr unübersichtlich, wodurch die Nutzerfreundlichkeit nicht mehr gewährleistet war.

Das Problem wird schliesslich dadurch gelöst, dass Formulare, wie beispielsweise das Beitritts-Formular des Vereins, welches sich mit dem Befehl `/join` öffnet, in mehrere Schritte unterteilt werden. Das Nicht-Mitglied gibt die ersten fünf Eingaben im Modal ein und sendet diese an den Bot. Da es auch nicht möglich ist ein weiteres Modal ohne Nutzerinteraktion zu öffnen, wird als Antwort auf das erste Modal eine Nachricht mit Button gesendet, welcher das zweite Modal öffnet. Hier kann das Nicht-Mitglied die restlichen Informationen angeben.

Abb. 26: Nachricht um das nächste Modal zu öffnen

Die Validierung der Eingaben erfolgt erst nach dem Absenden des jeweiligen Modals. Bei fehlerhaftem Input wird das Problem im Chat angezeigt, woraufhin das betroffene Modal erneut geöffnet und korrekt ausgefüllt werden kann. Dadurch werden fehlerhafte oder unvollständige Daten verhindert.

```
export async function handleJoinModal(interaction) {
  const birthdate = interaction.fields.getTextInputValue("birthdate");

  const validBirthdate = validateDate(birthdate);
  if (!validBirthdate.ok) {
    return interaction.reply({
      content: getErrorMessage(validBirthdate.error),
      flags: MessageFlags.Ephemeral,
    });
  }
}
```

Listing 6: Implementierungs-Ausschnitt des Discord-Modals für den `/join`-Befehl mit Validierung

7.3.5 Willkommensnachricht

Beim ersten Beitritt eines neuen Mitglieds auf den Discord-Server wird automatisch eine Willkommensnachricht per Privatnachricht versendet. Dies geschieht über die Utility-Funktion `sendDM`, die eine Direktnachricht an das neue Mitglied sendet. Die Nachricht enthält wichtige Informationen zum Verein und Hinweise zur Nutzung des Bots.

```
client.on("guildMemberAdd", async (member) => {
  const welcomeMessage = `Willkommen auf dem Server **${member.guild.name}**! ...`;
  try {
    await member.send(welcomeMessage);
  } catch {
    console.warn(`DM an ${member.user.tag} fehlgeschlagen.`);
  }
});
```

Listing 7: Implementierungs-Ausschnitt der Willkommensnachricht an neue Mitglieder im Privat-Chat

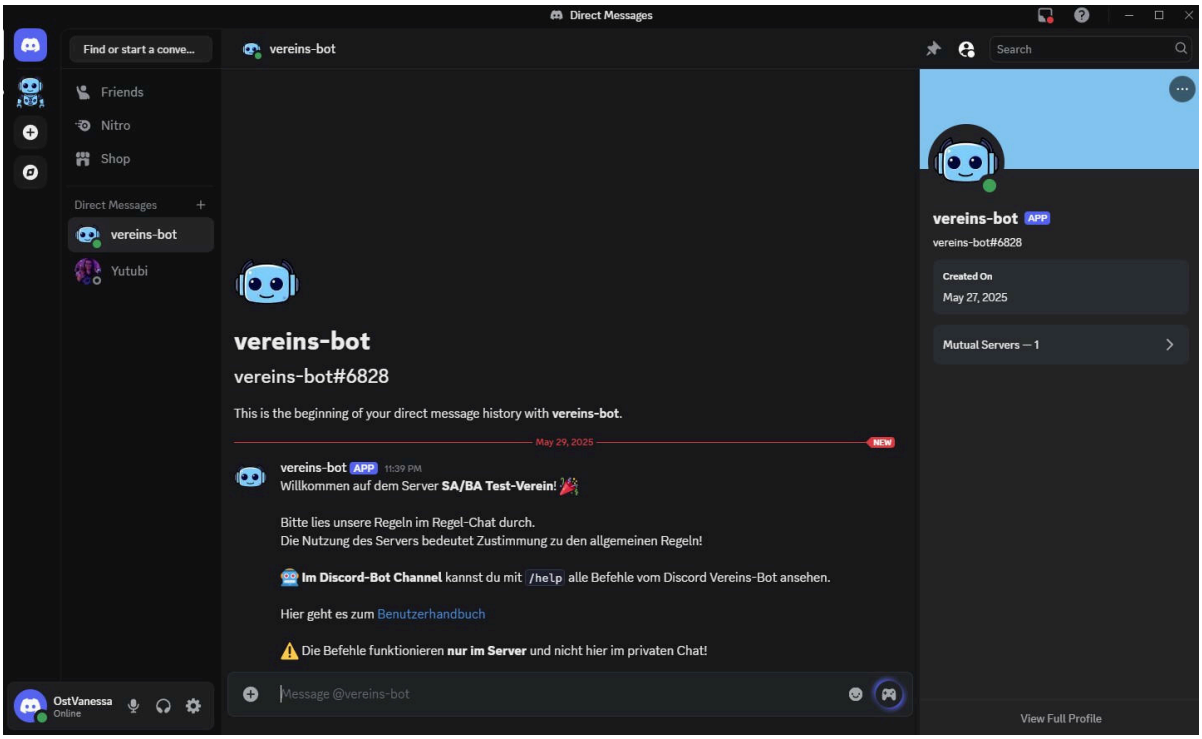


Abb. 27: Automatische Privatnachricht nach dem erstmaligen Server-Beitritt in Discord

7.3.6 Befehls-Übersicht

Mit dem `/help`-Befehl können Mitglieder die für sie verfügbaren Befehle abrufen. Die Darstellung der Befehle ist rollenbasiert implementiert, sodass reguläre Mitglieder ausschliesslich allgemeine Funktionen sehen, während Vorstandsmitglieder zusätzlich Zugriff auf Verwaltungs- und Administrationsbefehle erhalten.

Um die Usability zu optimieren und eine potenzielle Überforderung weniger technikaffiner Vorstandsmitglieder zu vermeiden, wurden bestimmte Befehle wie beispielsweise jene zur Rollen- und Kalendersynchronisation als „Admin-Befehle“ klassifiziert. Diese werden zwar für alle Vorstandsmitglieder angezeigt, sind jedoch primär für technisch versierte Vorstandsmitglieder relevant, die mit diesen Funktionen vertraut sind. Diese Vorgehensweise dient der Transparenz und stellt sicher, dass alle Vorstandsmitglieder über das vollständige Funktionsspektrum informiert sind.

Die Kategorisierung der Befehle in allgemeine, Vorstands- und Admin-Befehle trägt zur Übersichtlichkeit bei und ermöglicht es, dass Vorstandsmitglieder bei Bedarf jederzeit auf alle Funktionen zugreifen können. Technisch betrachtet verfügen alle Vorstandsmitglieder über identische Admin-Rechte, sodass keine Einschränkung der Funktionalität erfolgt. Diese Struktur unterstützt die rollenbasierte Zugriffskontrolle und reduziert gleichzeitig die kognitive Belastung durch eine klare Trennung von Aufgabenbereichen.

```
export const data = new SlashCommandBuilder()
    .setName("help")
    .setDescription(
        "Zeigt eine Übersicht aller verfügbaren Befehle und deren Beschreibung.",
    );

export async function execute(interaction: CommandInteraction) {
    const member = interaction.member as GuildMember;
    const isVorstand = member.roles.cache.some(
        (role) => role.name.toLowerCase() === "vorstand",
    );

    const MANUAL_URI = process.env.MANUAL_URI;

    let helpText = `
**Hilfe - Verfügbare Befehle:**...`;

    if (isVorstand) {
        helpText += `\n\n
**Vorstands-Befehle:**...`;
    }

    await interaction.reply({
        content: helpText,
        flags: MessageFlags.Ephemeral,
    });
}
```

Listing 8: Implementierungs-Ausschnitt des `/help`-Befehls

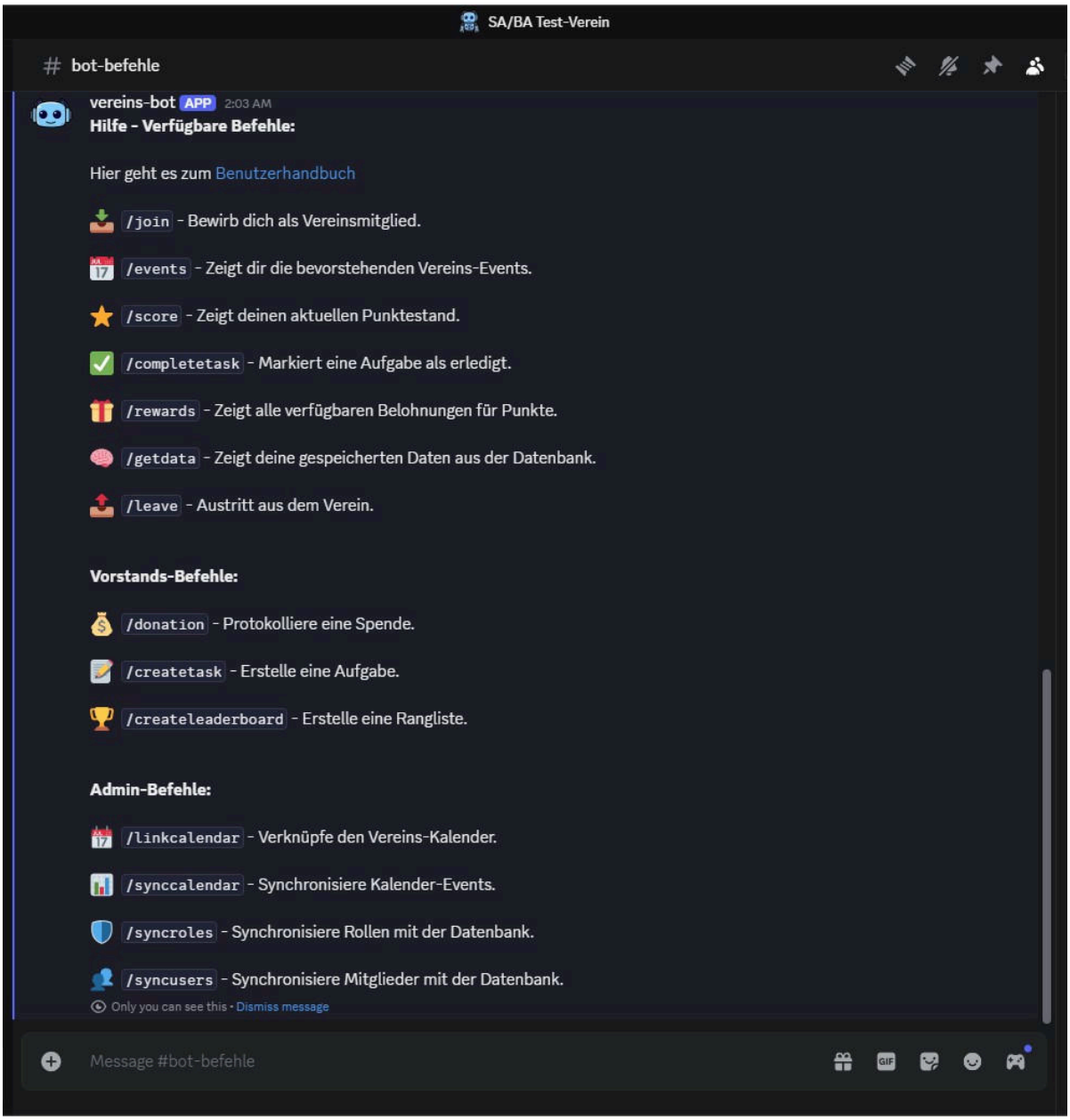


Abb. 28: Ausgabe nach Verwendung des `/help`-Befehls als Person mit Vorstandsrolle

7.4 Ein- und Austritt sowie Rollenzuweisung

Der in Abb. 30 dargestellte Workflow beschreibt den Bewerbungsprozess für neue Mitglieder im Discord-Server eines Vereins. Ein potenzielles Mitglied initiiert die Bewerbung über den Befehl `/join`. Der Bot sendet daraufhin ein mehrstufiges Formular (Abb. 29), in dem die für die Mitgliederverwaltung relevanten Informationen erfasst werden.

Die Daten des ersten Bewerbungsformulars werden temporär in der Datenbanktabelle `JoinTempData` gespeichert. Ursprünglich wurde testweise eine Speicherung in einem globalen Cache im Arbeitsspeicher des Bots implementiert. Diese Lösung erwies sich jedoch für den produktiven Einsatz als nicht tragfähig, da der Cache nicht benutzerspezifisch isoliert ist. Bei gleichzeitigen Bewerbungen mehrerer Nutzer bestünde die Gefahr von Datenüberschreibungen, wodurch Bewerbungsdaten inkonsistent oder unvollständig gespeichert worden wären.

Eine alternative Verwendung von Benutzersessions wurde geprüft, jedoch aufgrund mehrerer technischer und organisatorischer Gründe verworfen. Zum einen würde eine Session-Verwaltung zusätzliche Komplexität in der Implementierung bedeuten, insbesondere in einem asynchronen Event-Driven-Umfeld wie Discord.js. Zum anderen wären Sessions in der Regel flüchtig und könnten bei einem Serverneustart oder Absturz verloren gehen, wodurch die bereits durch die Person eingegeben Daten potenziell nicht wiederhergestellt werden könnten.

Die Speicherung in einer persistenten Datenbank (MongoDB) wurde daher als die geeignetste Lösung identifiziert. Sie gewährleistet eine transaktionssichere, benutzerspezifische Speicherung, auch bei hoher Anzahl paralleler Anfragen und bei Systemabstürzen oder Neustarts. Zudem bietet MongoDB nativ die Möglichkeit, Daten mit einem automatischen Ablaufdatum zu versehen (Time-To-Live-Index). Dadurch werden temporäre Bewerbungsdaten nach einer definierten Zeitspanne (z. B. 30 Minuten) automatisch gelöscht, falls der Bewerbungsprozess nicht abgeschlossen wurde. Diese Funktion unterstützt die Einhaltung des Prinzips der Datenminimierung und vermeidet die Speicherung nicht benötigter oder veralteter Bewerbungsdaten.

Sobald die Bewerbung vollständig abgeschlossen ist und die Datenschutzrichtlinien akzeptiert, wird der finale Eintrag in der `Person`-Tabelle mit dem Status `Pending` erstellt. Anschliessend werden die temporären Daten aus der Tabelle `JoinTempData` gelöscht, um Speicherplatz freizugeben und die Datenhaltung nach dem Prinzip der Datenminimierung umzusetzen.

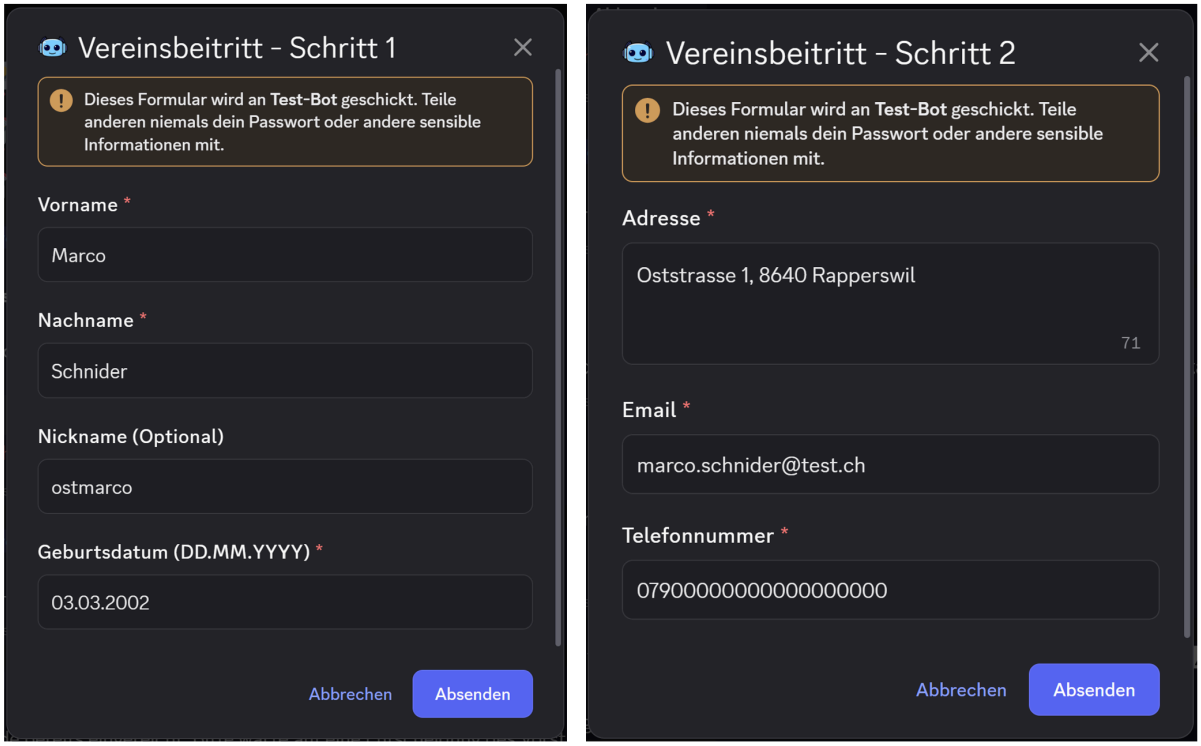


Abb. 29: Formulare der beiden Bewerbungsschritte

Nach Abschluss der Bewerbung wird der Vorstand im Discord-Kanal „bot-bewerbungen“ benachrichtigt. Nach manueller Prüfung der Zahlung kann über Buttons entschieden werden, ob die Bewerbung angenommen oder abgelehnt wird. Bei Annahme aktualisiert der Bot den Datenbankeintrag und vergibt automatisch die entsprechende Rolle. Im Falle einer Ablehnung wird der Datensatz gelöscht und der Bewerber benachrichtigt.

Dieser Workflow zeigt, wie der Discord-Bot durch strukturierte Datenerfassung, gezielte Unterstützung bei Entscheidungen und nahtlose Integration in die Serververwaltung zur effizienten digitalen Vereinsorganisation beiträgt.

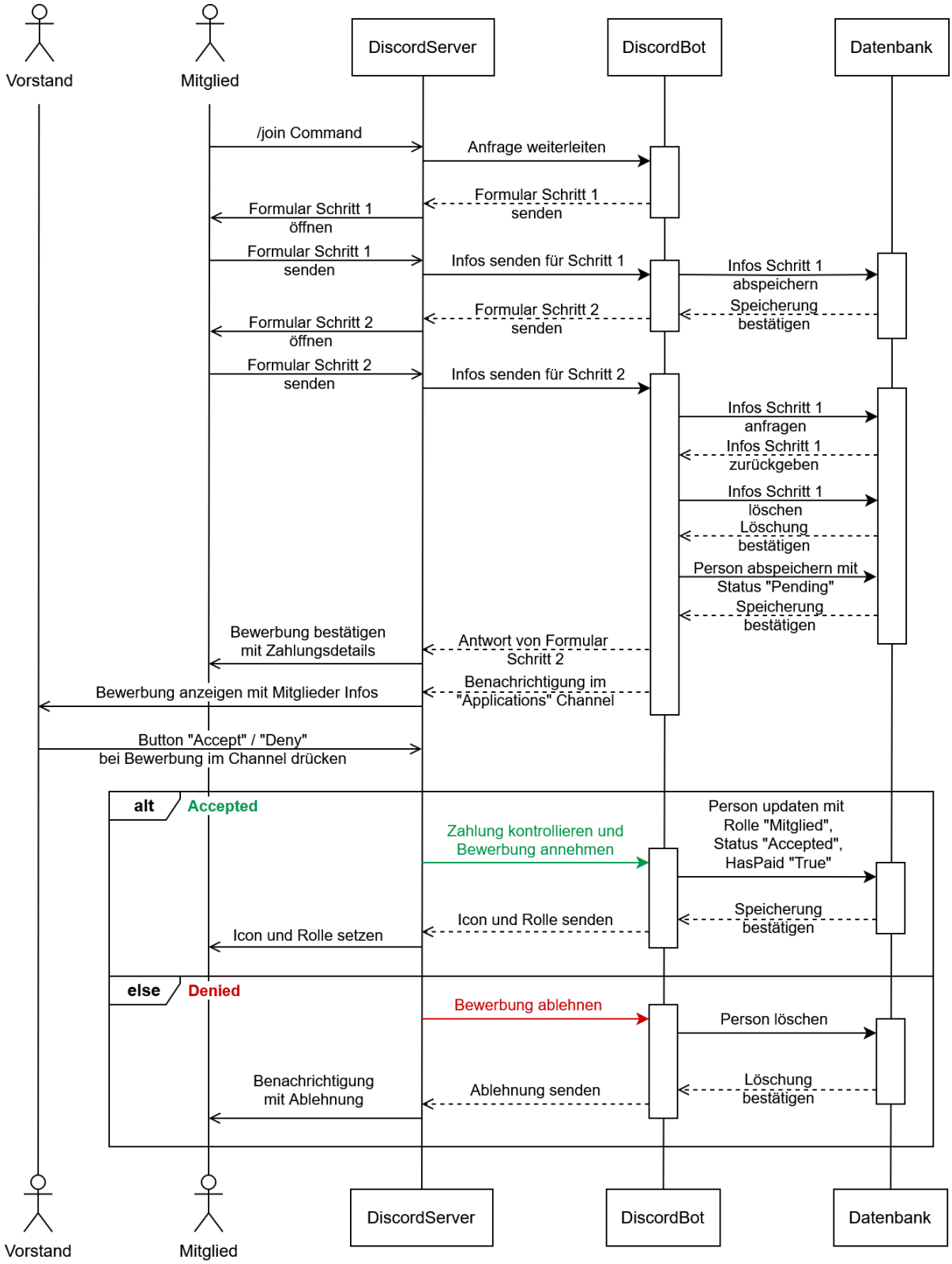


Abb. 30: Workflow zur Bewerbung und Rollenzuweisung

Mit dem Befehl `/leave` kann ein Mitglied den Verein verlassen. Nach Bestätigung wird der Status auf „ToBeDeleted“ gesetzt. In Übereinstimmung mit dem Datenschutzkonzept (siehe Abschnitt 3.4) werden die personenbezogenen Daten nach einer Frist von drei Monaten automatisch gelöscht. Während dieser Übergangszeit werden die Rollen entfernt und die Person über den Ablauf informiert.

```
const deletionDate = new Date();
deletionDate.setMonth(deletionDate.getMonth() + 3);
await UserModel.updatePersonStatusAndDeletionDate(
    person._id,
    "ToBeDeleted",
    deletionDate,
);
await member.roles.remove(role, "User has left the club");
```

Listing 9: Implementierungs-Ausschnitt der Planung einer Mitglieds-Löschung

7.5 Rollen- und Nutzerverwaltung

Ein zentrales Ziel der Anwendung ist die konsistente Synchronisierung von Vereinsmitgliedern und deren Rollen zwischen dem Discord-Server und der Datenbank. Die Umsetzung erfolgt über die beiden Slash-Befehle `/syncusers` und `/syncroles`. Ergänzend erlaubt der Befehl `/getdata` den Nutzern, ihre gespeicherten Daten DSGVO-konform abzurufen. Alle Funktionen sind durch ein rollenbasiertes Berechtigungssystem abgesichert.

Die Implementierung der Befehle `/syncusers` und `/syncroles` ist notwendig, da Discord von sich aus keine native Funktionalität bietet, um manuelle Änderungen wie das Vergeben einer Rolle durch ein Vorstandsmitglied oder das manuelle Anlegen von Rollen automatisch mit einer externen Datenbank abzugleichen. Auch das manuelle Ändern an Mitgliedschaften (z. B. Rollenvergabe) erkennt der Discord-Bot nicht automatisch. Ein theoretischer Ansatz wäre, durch ein zeitgesteuertes Cron-Job-System oder periodische Abfragen solche Änderungen zu erkennen und automatisiert zu synchronisieren. Eine solche permanente Abfrage wurde jedoch bewusst nicht implementiert, da diese Änderungen erfahrungsgemäss nur selten auftreten und eine manuelle Zuweisung von Rollen nicht zum Regelbetrieb gehört. Eine kontinuierliche Abfrage würde darüber hinaus zusätzliche Ressourcen beanspruchen, die in Relation zum Nutzen unverhältnismässig wären.

Aus diesen Gründen wurde die Entscheidung getroffen, die Synchronisation der Rollen und Benutzer explizit durch die manuellen Befehle auszulösen. Diese Vorgehensweise stellt sicher, dass Discord als führende Datenquelle für die Rollen- und Benutzerverwaltung dient und dass alle Änderungen über klar definierte Schnittstellen kontrolliert und nachvollziehbar in die Datenbank übernommen werden.

Besonders beim erstmaligen Setup des Bots im Server ist der manuelle Aufruf dieser Befehle erforderlich, um eine konsistente Ausgangsbasis zwischen Discord und der Datenbank zu schaffen. Eine ausführliche Anleitung zur Einrichtung und zur Verwendung dieser Befehle befindet sich im technischen README der Applikation (vgl. Anhang „README.pdf“). Dieses Vorgehen gewährleistet, dass auch seltene manuelle Änderungen zuverlässig erkannt und verarbeitet werden, wodurch potenzielle Fehler oder Inkonsistenzen in der Datenhaltung vermieden werden. Auf diese Weise bleibt die Integrität der Daten über alle relevanten Systeme hinweg gewahrt.

7.5.1 Synchronisation der Rollen

Der Befehl `/syncroles` ermöglicht es, die in der Datenbank hinterlegten Rollen mit dem Discord-Server abzugleichen. Diese Synchronisation erfolgt bidirektional:

1. **Discord zu Datenbank:** Alle existierenden Rollen im Discord-Server (exklusive Bot-Rollen) werden in die Datenbank übernommen oder aktualisiert.
2. **Datenbank zu Discord:** Rollen, die in der Datenbank gespeichert sind, aber im Discord-Server noch fehlen, werden erstellt.

Die Synchronisation berücksichtigt Rolleneigenschaften wie Berechtigungen, Icons, Farben und Erwähnungsoptionen. Bei Konflikten wird das ausführende Vorstandsmitglied um Bestätigung gebeten, bevor Rollen in der Datenbank überschrieben werden.

Zusätzlich sorgt die Synchronisation dafür, dass Discord-Mitglieder, denen laut Datenbank eine Rolle zugeordnet ist, diese auch im Discord-Server korrekt zugewiesen bekommen.

```
for (const role of dbRoles) {
  const usersWithRole = await UserRole.find({ roleId: role._id }).populate("userId");
  for (const userRole of usersWithRole) {
    const member = guild.members.cache.get(userRole.userId?.discordId);
    if (member && !member.roles.cache.has(discordRole.id)) {
      await member.roles.add(discordRole, "Synchronisiert aus DB");
    }
  }
}
```

Listing 10: Implementierungs-Ausschnitt des Rollenabgleichs von Mitgliedern zwischen Datenbank und Discord

7.5.2 Synchronisation der Mitglieder

Mit dem Befehl `/syncusers` wird überprüft, welche Mitglieder im Discord-Server über die Rollen „Mitglied“ oder „Vorstand“ verfügen. Für jeden dieser Benutzer wird ein Datenbankeintrag vom Typ `Person` angelegt oder aktualisiert. Existieren bereits Einträge mit dem Status `ToBeDeleted`, wird dieser zurück auf `Accepted` gesetzt, sofern die Rolle wieder vorhanden ist. Gleichzeitig wird die Zuordnung zwischen Nutzern und Rollen in der Zwischentabelle `UserRole` gepflegt.

Ein typisches Anwendungsbeispiel ist die automatische Erstellung eines neuen Datenbankeintrags für ein Discord-Mitglied, wenn dieses im Discord-Server eine gültige Rolle erhält, aber in der Datenbank noch nicht existiert.

Die Rollennamen im Discord werden mit den Rolleneinträgen in der Datenbank abgeglichen. Abweichungen werden identifiziert und Änderungen synchronisiert. Ergebnisse und Aktionen werden in einer eingebetteten Discord-Nachricht an die aufrufende Person mit Vorstandsrolle kommuniziert.

```
for (const member of allMembers.values()) {
  // Person finden oder neu anlegen...
  const memberRoleNames = member.roles.cache.map((r) => r.name);
  const matchedDbRoles = await DbRole.find({ name: { $in: memberRoleNames } });
  // UserRole-Einträge hinzufügen / entfernen...
}
```

Listing 11: Implementierungs-Ausschnitt der Synchronisation von Discord-Mitgliedern mit der Datenbank

7.5.3 Nutzerdaten abrufen

Die Implementierung des `/getdata`-Befehls ermöglicht es Mitgliedern, ihre in der Datenbank gespeicherten Daten DSGVO-konform abzurufen. Zu den abgerufenen Informationen gehören:

- Profildaten (Name, Adresse, Geburtsdatum, etc.)
- Zugeordnete Rollen im Discord-Server
- Aufgaben und deren Bearbeitungsstatus
- Getätigte Spenden
- Erhaltene Belohnungen
- Punkte und Transaktionen

Die Ausgabe der Daten erfolgt über eine Discord-Embed-Nachricht, die nur für das jeweilige Mitglied sichtbar ist (Ephemeral Message). Diese Implementierung trägt massgeblich zur Transparenz der Datenverarbeitung bei und unterstützt damit die Einhaltung der gesetzlichen Datenschutzverordnung.

Die Realisierung des `/getdata`-Befehls umfasst die Verarbeitung mehrerer asynchroner Datenbankabfragen, um die verschiedenen Entitäten wie Rollen, Aufgaben und Spenden korrekt zu aggregieren und strukturiert darzustellen. Technisch gesehen wird dabei zunächst die Discord-ID des Mitglieds als Schlüssel zur Identifikation der Person verwendet. Anschliessend werden alle relevanten Daten aus der Datenbank abgefragt und in einem `EmbedBuilder`-Objekt aufbereitet und ausschliesslich dem Mitglied selbst angezeigt (Ephemeral Message). Dies gewährleistet eine übersichtliche und zugleich benutzerfreundliche Präsentation der Daten im Discord-Client.

Durch die Integration diese Befehls wird eine benutzerzentrierte Transparenz geschaffen, die sowohl funktionale als auch rechtliche Anforderungen erfüllt. (Abschnitt 3)

```ts
export const data = new SlashCommandBuilder()
    .setName("getdata")
    .setDescription("Zeigt deine gespeicherten Daten aus der Datenbank.");

export async function handleGetDataCommand(
    interaction: CommandInteraction
): Promise<void> {
    await interaction.deferReply({ flags: MessageFlags.Ephemeral });

    const userDiscordId = interaction.user.id;
    const personResult = await UserModel.getPersonByDiscordId(userDiscordId);
    if (!personResult.ok) {
        await replyWithDeferredError(interaction, personResult.error);
        return;
    }
    const person = personResult.value;

    const userRoles = await UserModel.getUserRoles(person._id);
    const roles = userRoles.ok
        ? userRoles.value.map(r => r.name).join(", ")
        : "Keine";

    const embed = new EmbedBuilder()
        .setTitle(`📊 Datenübersicht für ${person.nickname}`)
        .addFields(
            { name: "Name", value: `${person.firstName} ${person.lastName}` },
            { name: "Discord-ID", value: person.discordId },
            { name: "Rollen", value: roles }
        )
        .setTimestamp();

    await interaction.editReply({ embeds: [embed] });
}
```

Listing 12: Implementierungs-Ausschnitt des `/getdata`-Befehls

7.6 Kalenderintegration

Um geplante Events direkt im Discord-Server anzeigen und verwalten zu können, wurde die Möglichkeit eine Verbindung zu Google Calendar einzurichten implementiert. Um sicherzustellen, dass zukünftige Kalenderanbieter (z.B. Outlook, Apple Calendar) leicht integrierbar sind, wurde ein Interface definiert, welches für jeden Kalenderanbieter implementiert werden muss. Aktuell wurde dieses für den Google Calendar implementiert.

7.6.1 Google API und Authentifizierung

Die Integration basiert auf dem OAuth2 Protokoll, mit dem Nutzer sicher authentifiziert werden, damit auf die Kalenderdaten zugegriffen werden kann. Die Verbindung wurde mithilfe von der Google APIs Library[11] entwickelt. Diese Library bietet einen OAuth2-Client welche das erstellen der Anfragen vereinfacht. Die Implementation folgt diesen Schritten:

1. Ein Server-Admin führt den Command `/linkcalendar` im Discord-Server aus.
2. Der Bot antwortet mit einem Google Kalender Authentifizierungslink welche die Id des Discord-Servers beinhaltet.
3. Der Admin öffnet dem Link und gibt den Zugriff auf den gewünschten Kalender frei.
4. Google leitet nun auf eine vordefinierte Callback-URL weiter, welche von einem Express-Webserver[12] abgefangen wird.
5. Beim Callback wird von Google ein Code mitgesendet, welcher verwendet werden kann, um den Access und Refresh-Token vom OAuth2-Service zu erhalten.
6. Diese Token werden mit der entsprechenden Discord-Server Id (welche auch beim Callback mitgesendet wurde) in der Datenbank gespeichert.
7. Der Token kann nun bei der Abfrage von Kalender-Informationen zur Authentifizierung verwendet werden.

[11] https://github.com/googleapis/google-api-nodejs-client#readme
[12] https://expressjs.com/

7.6.2 Token Verwaltung

Der Access-Token hat eine limitierte Gültigkeit von einer Stunde. Das Ablaufdatum wird jeweils mit dem Token vom Authentifizierungs-Server mitgesendet und in der Datenbank gespeichert. Bei jeder Anfrage auf die Google Calendar Ressource muss zuerst überprüft werden ob der Access-Token noch gültig ist. Ist dies nicht der Fall muss mittels des Refresh-Token ein neuer Access-Token beim Authentifizierungs-Server angefragt werden.

7.6.3 Event-Synchronisation

Events können auf zwei verschiedene Arten erstellt werden. Über den Kalender Provider, hier Google Calendar oder direkt über Discord. Entsprechend müssen die Events synchronisiert werden. Zudem werden die Events in der Datenbank gespeichert, damit sie mit den Aufgaben verbunden werden können. Um mit den verschiedenen Event-Formaten arbeiten zu können, werden diese in ein einheitliches, intern definiertes Event-Format konvertiert.

Discord Events[13] werden automatisch nach ihrer Erstellung mit dem Google Calendar synchronisiert.

Der Google Calendar bietet Webhooks zur automatischen Benachrichtigung, falls neue Events erstellt werden. Um diese Events empfangen zu können ist jedoch eine öffentlich aufrufbare URL notwendig. Da diese aktuell nicht vorhanden ist, werden vom Discord-Bot aus periodisch die Events vom Google Calendar abgefragt. Neue Events werden gespeichert. Events die bereits zuvor gespeichert wurden, werden auf Änderungen geprüft und wenn nötig aktualisiert.

Eine besondere Herausforderung stellte die Handhabung von wiederkehrenden Events dar. Discord verwendet hierfür eine angepasste Version der iCalendar-Wiederholungsregel[14], unterstützt jedoch nur eine begrenzte Auswahl an Wiederholungsmustern. Dabei ist nicht jede logische Kombination erlaubt. So lässt sich beispielsweise ein wöchentliches Event von Dienstag bis Samstag erstellen, nicht jedoch eines von Mittwoch bis Samstag. Ausserdem werden auch andere Wiederholungsmuster abweichend dargestellt. Wenn möglich wird die Wiederholungsregel in ein kompatibles Format konvertiert. Ist dies nicht möglich, werden solche nicht unterstützten Wiederholungen als Einzelereignisse gespeichert, um dennoch eine konsistente Darstellung sicherzustellen. Eine vollständige Liste der zulässigen Wiederholungsoptionen ist in der Discord-Event-Dokumentation[15] einsehbar.

Bei jeder periodischen Event-Synchronisation wird zudem eine Liste mit den anstehenden Events im „event“-Kanal aktualisiert. Diese Liste können Mitglieder zudem mit dem `/event`-Befehl einsehen.

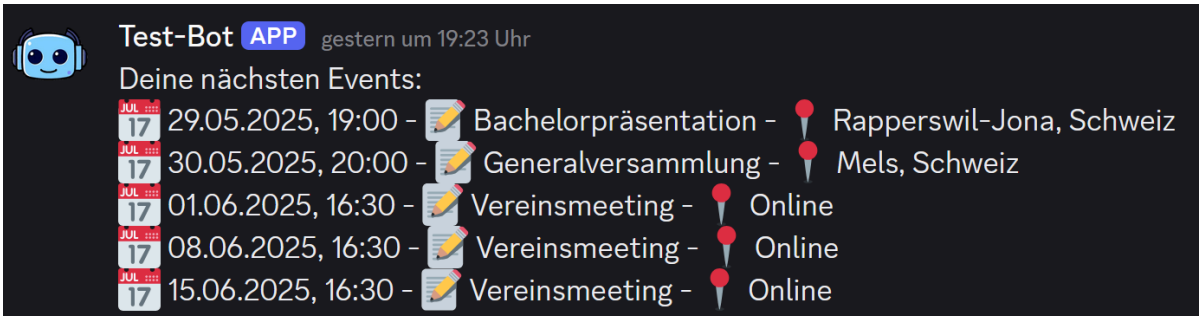


Abb. 31: Event-Übersicht Nachricht

7.6.4 Event Benachrichtigungen

Nutzer können Events direkt in der Discord-Oberfläche einsehen und sich für Benachrichtigungen anmelden. Der Discord-Bot überprüft regelmässig, ob ein Event kurz vor dem Start steht, und sendet rechtzeitig eine Nachricht in einen speziellen Event-Kanal, wobei interessierte Nutzer markiert werden.

[13] https://discord.com/developers/docs/resources/guild-scheduled-event
[14] https://icalendar.org/iCalendar-RFC-5545/3-8-5-3-recurrence-rule.html
[15] https://discord.com/developers/docs/resources/guild-scheduled-event#guild-scheduled-event-recurrence-rule-object

7.7 Gamification

Die Gamification-Komponente des Discord-Bots zielt darauf ab, Mitglieder zu motivieren, sich aktiv am Vereinsleben zu beteiligen. Dies geschieht durch ein Punktesystem, Aufgaben, Spendenmechaniken, Belohnungen und Ranglisten. Die Punktevergabe ist dabei eng mit dem Workflow der Anwendung verknüpft und erfolgt automatisiert über bestimmte Aktionen. Eine konzeptionelle Einführung und Übersicht zur Gamification ist bereits in Abschnitt 4 beschrieben.

7.7.1 Implementierung des Punktesystems

Das Punktesystem basiert in der Implementierung auf dem Datenbankmodell `Transaction`, welches sämtliche Punktevergaben revisionssicher und konsistent dokumentiert. Über definierte Aktionen im Discord-Bot-System werden Punkte automatisch vergeben, beispielsweise für die Teilnahme an Events oder die Erledigung von Aufgaben. Zur Unterstützung der administrativen Planung und Auswertung wurde zusätzlich eine Excel-Vorlage entwickelt, wie in Abschnitt 4.8 ausführlich beschrieben (vgl. Anhang „pointsystem_calculator.xlsx“). Diese ermöglicht es dem Vorstand, das Punktesystem flexibel zu verwalten, Punktewerte zu definieren und zukünftige Punktevergaben transparent nachzuvollziehen. In der Excel-Vorlage werden Spenden ebenfalls berücksichtigt und nach den im README dokumentierten Regeln verrechnet (vgl. Anhang „README.pdf“). Dabei wird der Spendenbetrag während der Dokumentation durch eine Person im Vorstand mit dem Punkte-pro-CHF-Faktor multipliziert. Dieser Faktor kann je nach Vereinsbedürfnissen angepasst und direkt in der Vorlage gepflegt werden.

7.7.2 Aufgaben

Vorstandmitglieder können Aufgaben erstellen, welche dann von Mitgliedern oder anderen Vorstandsmitgliedern durchgeführt werden können. Dieser Prozess wird in der Abb. 33 dargestellt. Um eine Aufgabe zu erstellen wird der Slash-Befehl `/createtask` verwendet. Dieser öffnet ein Formular, bei welchem das Vorstandsmitglied die Daten für die Aufgabe angeben kann. Nach dem absenden des Formulars werden die Daten auf der Datenbank abgespeichert. In einen zweiten Schritt (Abb. 32) erhält das Vorstandsmitglied zwei Select-Menus, mit denen es die verantwortliche Person festlegen und die Aufgabe optional mit einem Event verknüpfen kann.

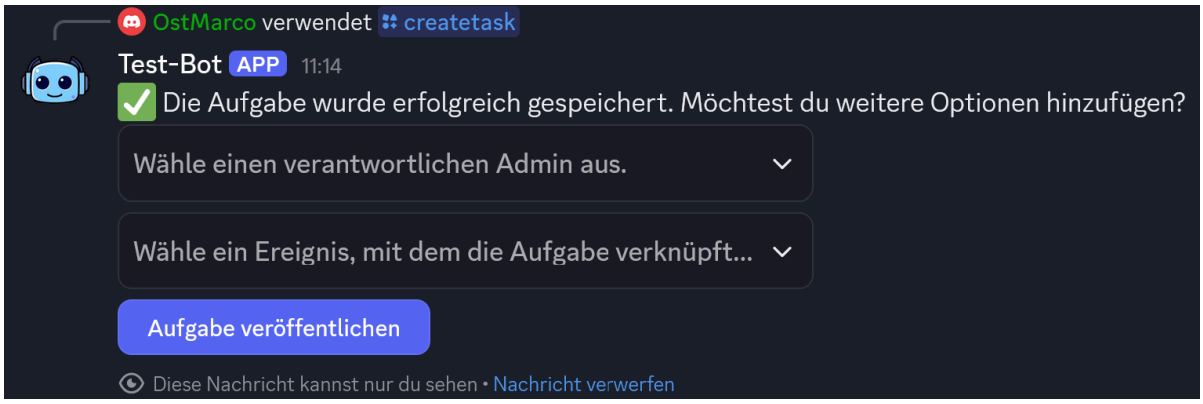


Abb. 32: Select-Menus zur Auswahl der verantwortlichen Person und Verknüpfung mit einem Event

Dieser Ablauf ist nötig, da Discord es nicht erlaubt Select-Menus in ein Formular-Modal einzufügen. Nachdem das Vorstandsmitglied die Daten angegeben hat, werden auch diese in der Datenbank gespeichert und die Aufgabe wird im „aufgaben“-Kanal veröffentlicht.

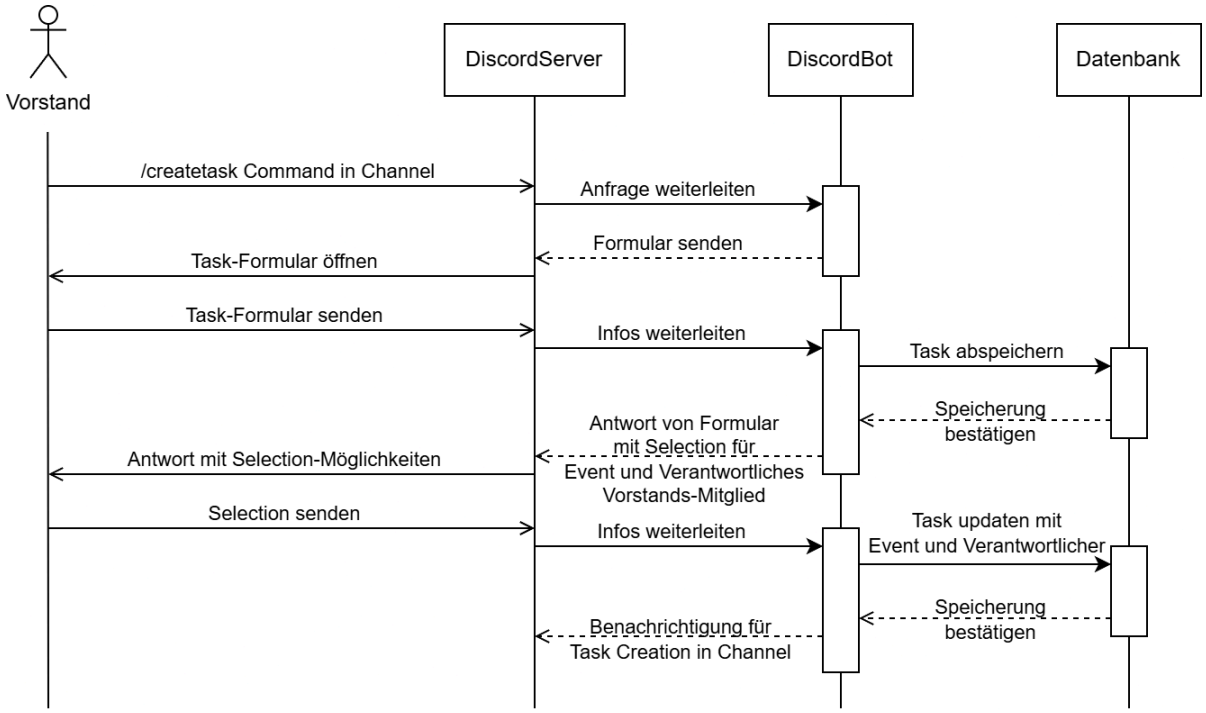


Abb. 33: Workflow zur Aufgabenerstellung

Die Abb. 35 zeigt wie Nutzer eine Aufgabe für sich beanspruchen können. Dazu können sie mit dem „Aufgabe beanspruchen“-Button bei der Aufgabenbeschreibung (Abb. 34) interagieren.

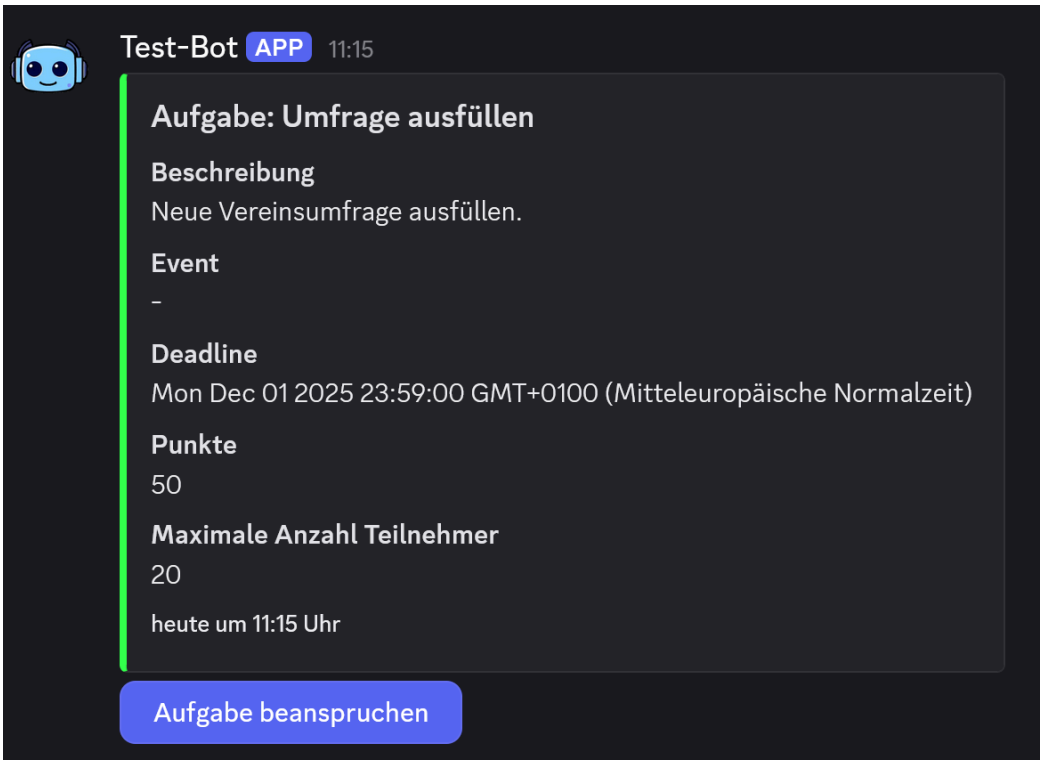


Abb. 34: Aufgabenübersicht im „aufgaben“-Kanal

Bevor die Aufgabe beansprucht werden kann muss zuerst überprüft werden, ob die maximale Anzahl erlaubter Teilnehmende bereits erreicht wurde und ob die betreffende Person diese Aufgabe zuvor bereits beansprucht hat. Ist dies nicht der Fall wird die Teilnahme gespeichert und das Mitglied bekommt eine Rückmeldung.

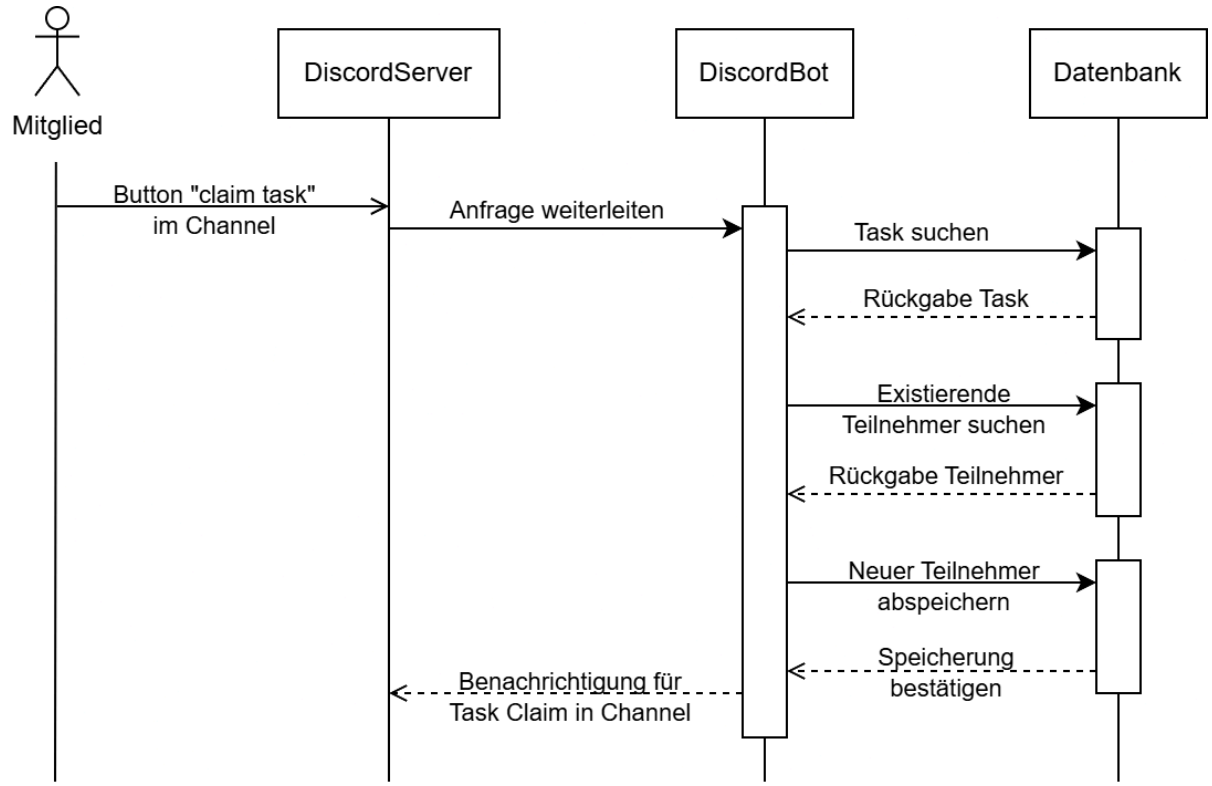


Abb. 35: Workflow zur Aufgabenbeanspruchung

Sobald ein Mitglied eine Aufgabe fertiggestellt hat, kann es dies mithilfe des `/completetask` Commands melden. Der Command bietet dem Mitglied ein Select-Menu mit allen, vom Mitglied angenommenen und noch nicht fertiggestellten Aufgaben. Hier kann es nun die gewünschte Aufgabe aussuchen und dem Vorstand wird eine entsprechende Benachrichtigung gesendet.

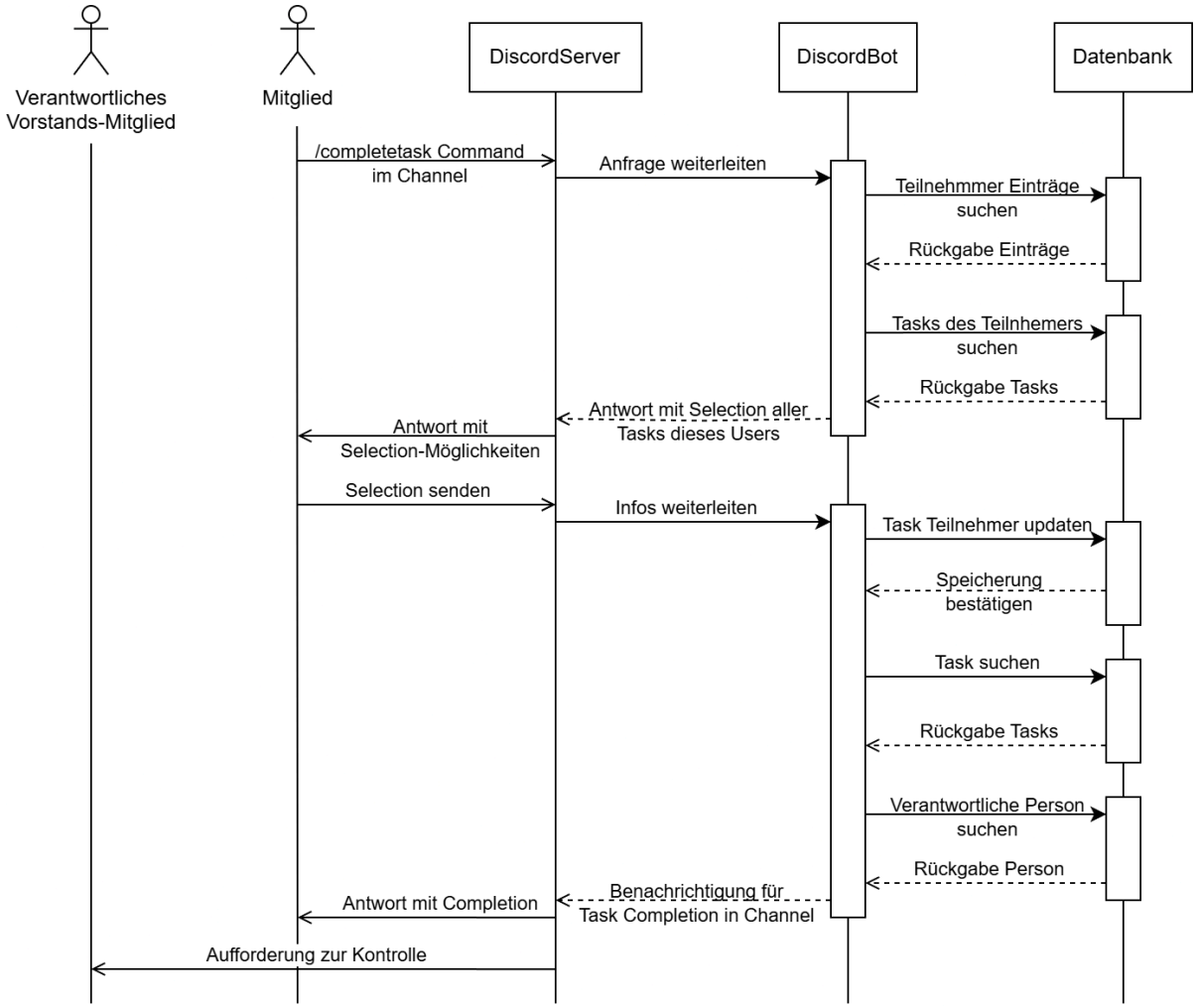


Abb. 36: Workflow zur Aufgabenfertigstellung

Die Benachrichtigung (Abb. 37) die der Vorstand bei der Fertigstellung einer Aufgabe erhält, enthält einen Button um die Fertigstellung zu bestätigen oder abzulehnen.

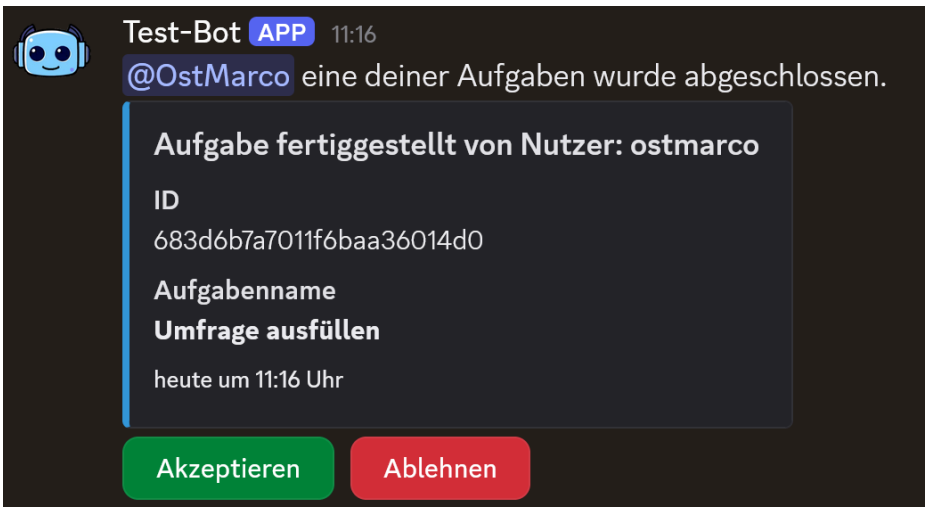


Abb. 37: Benachrichtigung and den Vorstand

Wurde die Aufgabe wunschgemäss umgesetzt, werden dem Mitglied Punkte auf das Konto gutgeschrieben und die relevanten Ranglisteneinträge werden inkrementiert. Wurde die Aufgabe nicht vollständig umgesetzt werden dem Mitglied keine Punkte gutgeschrieben. In beiden Fällen, kann der Vorstand durch ein Formular ein Feedback senden.

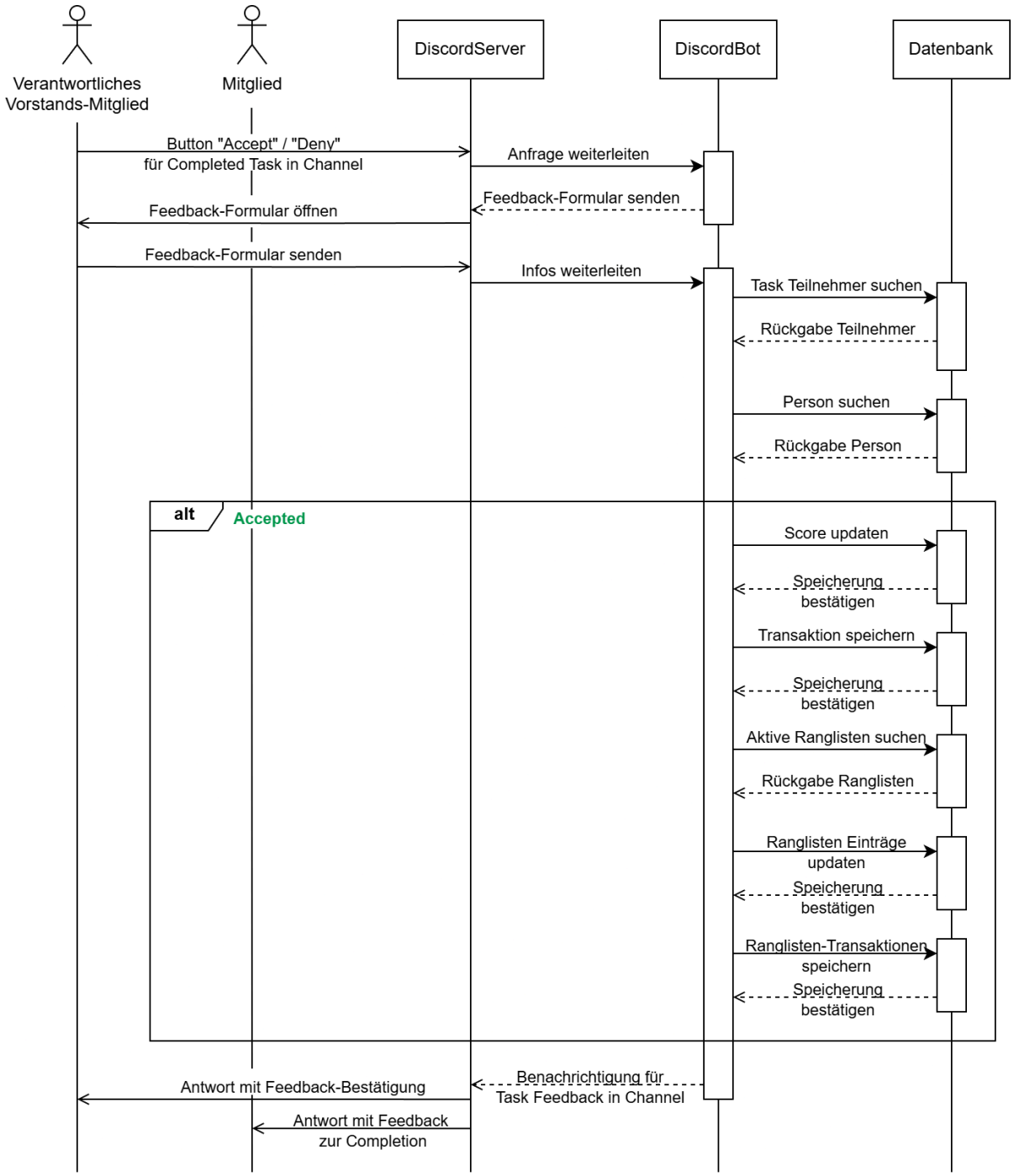


Abb. 38: Workflow zur Aufgabenkontrolle

7.7.3 Spenden

Neben Aufgaben können auch Spenden als Aktivität im Punktesystem berücksichtigt und mit Punkten belohnt werden. Dies ermöglicht es dem Verein, das Engagement seiner Mitglieder bei der finanziellen Unterstützung des Vereins sichtbar zu machen und fair anzuerkennen. Die Implementierung dieser Funktion erfolgt sowohl technisch über das Discord-Bot-System als auch organisatorisch über die Excel-Vorlage (`pointsystem_calculator.xlsx`).

7.7.3.1 Technische Umsetzung im Bot

Im Discord-Bot-System steht dem Vorstand der Slash-Befehl `/donation` zur Verfügung. Dieser erlaubt es, eingegangene Spenden durch ein Modal-Formular zu dokumentieren. Für eine korrekte Erfassung der Spende sind dabei folgende Angaben erforderlich:

- Betrag der Spende in CHF
- Datum der Spende
- Anzahl Punkte, die dem Mitglied für diese Spende gutgeschrieben werden
- Eine kurze Notiz zur Zuordnung (z. B. „Jahresbeitrag 2025“)

Die erfasste Spende wird im Transaktionsmodell gespeichert und automatisch mit der Rangliste synchronisiert.

7.7.3.2 Punktevergabe und Limite

Für die Punktevergabe wurde eine maximale Punktzahl pro Monat definiert, um eine übermässige Punkteakkumulation zu verhindern. Standardmässig sind maximal 500 Punkte pro Monat und Mitglied für Spenden zulässig. Diese Regelung stellt sicher, dass das Punktesystem fair bleibt und keine Wettbewerbsverzerrung durch hohe Einzelspenden entsteht.

7.7.4 Ranglisten

Ranglisten zeigen die von Nutzern innerhalb eines definierten Zeitraums gesammelten Punkte an. Sie können durch den Vorstand mithilfe des Befehls `/createleaderboard` erstellt werden. Im zugehörigen Formular lassen sich Name, Beschreibung, Startdatum, Enddatum sowie die Anzahl sichtbarer Mitglieder in der Ranglistenübersicht festlegen.

Alle von den Nutzern gesammelten Punkte werden allen aktiven Ranglisten hinzugefügt. Dazu wird in der MongoDB nach Ranglisten gesucht, deren Startdatum kleiner oder gleich (`$lte`) und deren Enddatum grösser oder gleich (`$gte`) dem aktuellen Datum ist.

```
const now = new Date();
const activeLeaderboards = await Leaderboard.find({
    startDate: { $lte: now },
    endDate: { $gte: now },
}).lean<ILeaderboard[] | null>();
```

Listing 13: Implementierungs-Ausschnitt der Auslesung der aktiven Ranglisten aus der Datenbank

Jede Punktesammlung eines Mitglieds aktualisiert automatisch die Ranglistenübersicht im „ranglisten“-Kanal. Ein Beispiel dafür ist die Allzeit-Rangliste in Abb. 39 .

Abb. 39: Übersicht der Allzeit-Rangliste

7.7.5 Belohnungen

Gesammelte Punkte können Mitglieder für Belohnungen eintauschen. Eine Übersicht der aktuell verfügbaren Belohnungen ist für Mitglieder über den Befehl `/rewards` im „bot-befehle“-Kanal abrufbar. Der Befehl Antwortet mit einer interaktiven Nachricht welche die verfügbaren Belohnungen anzeigt. Das Mitglied kann mit den Pfeilen durch die verschiedenen Belohnungen navigieren und die gewünschten einfordern. Dies funktioniert, indem die Nachricht bei jeder Interaktion bearbeitet wird, um die richtigen Informationen anzuzeigen. Eingeforderte Belohnung erstellen in der Datenbank eine entsprechende Transaktion mit dem Status `Pending` und es wird eine entsprechende Nachricht an den Vorstand gesendet.

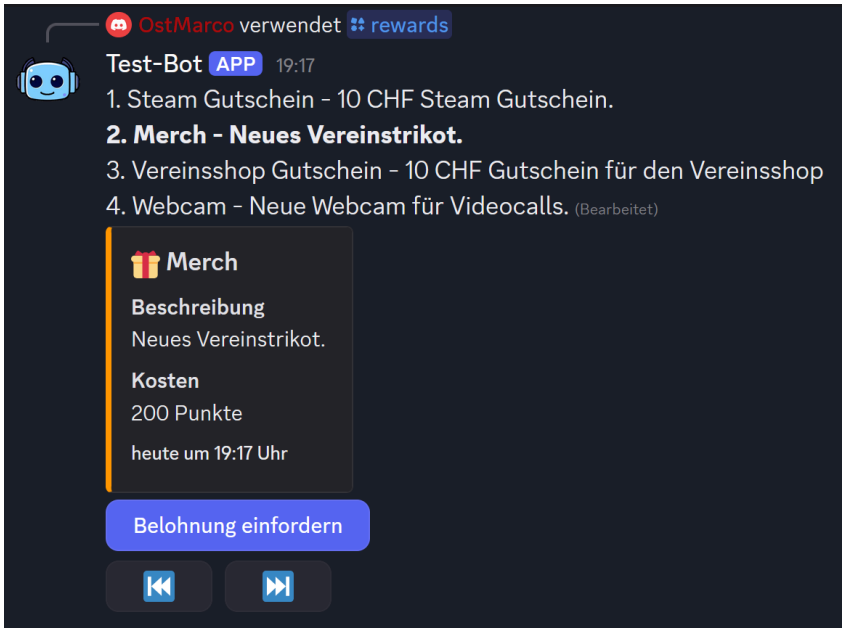


Abb. 40: Übersicht der verfügbaren Belohnungen

Eine Vorstandsmitglied kann, nachdem es die Belohnung bearbeitet hat, diese annehmen oder ablehnen. Bei einer Annahme erhält das Mitglied eine Benachrichtigung das die Belohnung auf dem Weg ist. Kann die Belohnung nicht bearbeitet werden wird das Mitglied auch informiert und dessen Punkte werden auf das Punktekonto zurückerstattet.

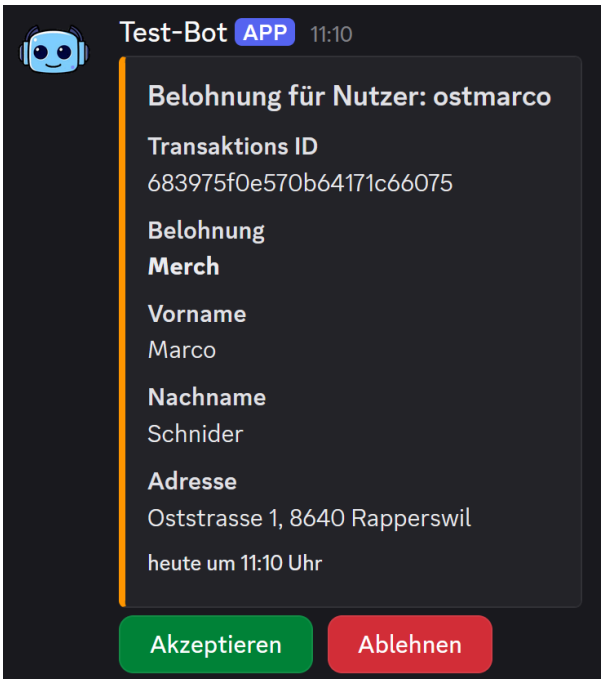


Abb. 41: Nachricht an den Vorstand bei Auswahl einer Belohnung

8 Fazit

Dieses Kapitel fasst die im Rahmen dieser Semester- und Bachelorarbeit erzielten Ergebnisse zusammen und zieht eine Bilanz über die Zielerreichung des Projektes. Es werden die implementierten Funktionen, die Erfüllung der funktionalen und nicht-funktionalen Anforderungen sowie die Resultate aus den Usability-Tests objektiv dargestellt. Die Schlussfolgerung reflektiert diese Resultate, bewertet die Zielerreichung kritisch und zeigt Verbesserungspotenziale für zukünftige Iterationen des Discord-Bots auf.

8.1 Resultate

Die vorliegende Arbeit lieferte einen funktionsfähigen Discord-Bot, der Mitgliederdaten eines Vereins in einer MongoDB-Datenbank verwaltet (UC1), Discord-Rollen automatisch zuweist (UC2) und Events aus Google Calendar synchronisiert (UC4). Diese Funktionen decken die priorisierten Use Cases UC1 , UC2 , UC4 und UC5 ab und wurden erfolgreich im produktiven Testbetrieb validiert. Das Gamification-System wurde prototypisch umgesetzt. Mitglieder sammeln Punkte für Eventteilnahmen, Spenden und Umfragen und sehen ihre Position in einer Rangliste (UC8). Belohnungen können direkt in Discord eingelöst werden (UC10).

Ein serverseitiger Validierungs-Mechanismus wurde implementiert (UC9), der sicherstellt, dass Punkte nur dann vergeben werden, wenn die zugrunde liegende Aktion gültig ist. Konkret prüft der Bot beim beispielsweise beim Beanspruchen einer Aufgabe, ob diese zum gegebenen Zeitpunkt noch existiert, ob das Mitglied die Aufgabe bereits abgeschlossen hat und ob der Anspruch auf Punkte noch nicht eingelöst wurde. Damit wird Manipulation verhindert und die Integrität des Punktesystems gewahrt.

Über den Befehl `/getdata` können Mitglieder sämtliche in der Datenbank hinterlegten Informationen über sie einsehen. Ihre Zustimmung wird durch den Beitritt in den Verein und durch die Nutzung des Discord-Servers sowie Discord-Bots erteilt. Nach dem Vereins-Austritt und einer Bedenkfrist werden sämtliche Daten aus der Datenbank gelöscht (UC7). Damit wird der Datenschutz der Mitglieder gewährleistet (NFR7). Die Synchronisierung mit Google Calendar erfolgt derzeit im Zehn-Minuten-Intervall per Cron-Job und informiert zuverlässig vor Eventbeginn. Webhooks für Echtzeit-Updates sind derzeit nicht implementiert (UC12).

Ein webbasiertes Admin-Dashboard wurde nicht umgesetzt (UC11). Änderungen an Mitgliederdaten, Ranglisten und Transaktionen sind nur direkt in der MongoDB-Datenbank möglich. Das System bietet keine Funktion, mit der Vorstandsmitglieder Transaktionen manuell im Discord ausführen können, wodurch Punkte für Mitglieder ohne Discord-Account derzeit nur durch eine manuelle Anpassung des Punktestandes in der Datenbank vergeben werden können. Eine manuelle Anpassung löst keine automatische Aktualisierung der Rangliste aus.

Das Anlegen von Discord-Kanälen erfolgt nicht automatisiert. Die Definition der Kanalnamen ist im Code festgelegt. Die Positionierung der Bot-Rolle erfordert manuelle Anpassungen in Discord. Cron-Jobs zum Aktualisieren von Eventnachrichten sowie die automatische Aktualisierung von Ranglisten nach jeder Transaktion sind vorhanden.

Die Testabdeckung von 80 Prozent wurde nicht erreicht (NFR2). Es wurden lediglich manuelle Tests der Slash-Befehle durchgeführt. Fehlerbehandlungen wurden implementiert, sodass es während der Usability-Tests zu keinem Absturz kam (NFR3). Atomare Transaktionen für Datenbankaktionen fehlen, ebenso wie eine zusätzliche Authentifizierung (z. B. Zwei-Faktor-Authentifizierung) für den Datenbankzugriff.

In Anbetracht der unterschiedlichen Arbeitsstunden (Bachelorarbeit und Semesterarbeit) wurde der Fokus bewusst auf die priorisierten Kernanforderungen gelegt. Dennoch konnte ein lauffähiger Bot mit den wichtigsten Funktionen bereitgestellt werden.

8.2 Schlussfolgerung

Die Zielsetzung dieser Arbeit bestand darin, ein Discord-basiertes System zu entwickeln, welches die Vereinsverwaltung durch Gamification unterstützt und dabei Datenschutz und Sicherheit berücksichtigt. Dieses Ziel wurde in weiten Teilen erreicht, insbesondere in Bezug auf die Automatisierung von Routineaufgaben, die Mitgliederverwaltung und die Eventorganisation. Der produktive Einsatz des Bots und die positiven Rückmeldungen der Testnutzer verdeutlichen, dass Discord als Plattform grundsätzlich geeignet ist, um Vereinsprozesse zu unterstützen und durch Gamification zu fördern.

Gleichzeitig wurden im Projektverlauf auch Schwächen und offene Baustellen erkennbar. Die zusätzliche Benutzer-Validierung für den Datenbankzugriff sowie der weitreichende administrative Zugriff über MongoDB Compass stellen ein Sicherheitsrisiko dar. Allerdings ist hierbei zu berücksichtigen, dass der Bot ausschliesslich auf einem privaten Vereinsserver

betrieben wird und daher nur autorisierte Vereinsmitglieder Zugriff haben. Diese abgeschlossene Serverumgebung stellt eine gewisse Zugangsbeschränkung dar und reduziert das Risiko unbefugter Zugriffe. Dennoch fehlen atomare Transaktionen, was zu Inkonsistenzen bei der Punktevergabe führen kann, besonders wenn mehrere Operationen gleichzeitig stattfinden. Die Notwendigkeit, viele administrative Aufgaben direkt in der Datenbank vorzunehmen, widerspricht dem Anspruch einer benutzerfreundlichen Lösung und ist fehleranfällig. Ebenso fehlt eine einfache Möglichkeit für Vorstandsmitglieder, Transaktionen manuell über Discord auszulösen, sodass Punkte für bestimmte Aktivitäten momentan nur indirekt oder gar nicht vergeben werden können.

Die in der Architektur geplante saubere Trennung zwischen Discord-Befehlen und Datenbankmodellen konnte aufgrund des hohen Aufwands nicht vollständig realisiert werden. Die bisherige Implementierung zeigt eine bestehende Verbindung zwischen Discord-spezifischer Logik und Datenbankzugriffen, was die Wartbarkeit und Erweiterbarkeit des Systems erschwert. Ein begonnenes Refactoring liegt derzeit als separater Branch vor, konnte aber während des Projektzeitraums nicht vollständig stabilisiert und getestet werden. Dadurch ist der Bot aktuell nicht vollständig entkoppelt und modularisiert, was langfristig die Flexibilität des Systems einschränkt.

Das System ist so konzipiert, dass Punktevergaben und ähnliche Aktionen ausschliesslich nach expliziter Freigabe durch ein Vorstandsmitglied erfolgen. Dieser manuelle Freigabeprozess stellt sicher, dass alle Punktevergaben autorisiert und nachvollziehbar ablaufen, schränkt jedoch die Automatisierung ein. Eine zukünftige Erweiterung könnte hier eine automatisierte Prüf- oder Zahlungskontrolle sein, die bestimmte Punktevergaben nach definierten Regeln automatisch genehmigt und so den Prozess effizienter gestaltet.

Die durchgeführte Nutzergruppen-Analyse zur Gamification ergab ein gemischtes Bild. Einige Mitglieder standen dem Gamification-Ansatz skeptisch gegenüber, während andere dessen Potenzial anerkannten. Eine durchgehend motivierende Wirkung konnte nicht eindeutig nachgewiesen werden, sondern zeigte sich nur selektiv in bestimmten Nutzungsszenarien. Dennoch wurde das Konzept in die Bot-Logik integriert. Die Usability-Tests bestätigten eine insgesamt intuitive Bedienbarkeit des Bots, ohne dass es zu Abstürzen kam. Der Datenschutz konnte durch transparente Datenabfragen und Löschfunktionen für Mitglieder sichergestellt werden. Die Testabdeckung von 80 Prozent konnte jedoch nicht erreicht werden, was die Fehlersicherheit und Wartbarkeit langfristig einschränkt.

Insgesamt bietet der entwickelte Discord-Bot eine solide Grundlage für eine moderne Vereinsverwaltung und Gamification, erfüllt jedoch noch nicht alle Anforderungen in der gewünschten Tiefe. Er bildet ein Fundament, das weiterentwickelt werden muss, insbesondere in den Bereichen Sicherheit, Testabdeckung, Benutzerfreundlichkeit und Systemarchitektur. Die tatsächliche Langzeitwirkung des entwickelten Gamification-Konzeptes bleibt derzeit ebenfalls offen und bedarf weiterer Analysen.

8.3 Ausblick

Für die nächste Entwicklungsphase bietet sich an, die nicht umgesetzten Usability-Test-Rückmeldungen zum Discord-Bot umzusetzen. Darüber hinaus wäre es denkbar, andere Plattformen wie Microsoft Teams einzubinden oder ein plattformunabhängiges Portal zu entwickeln, das Mitgliedern ohne Discord den Zugriff auf ihre Punkte und Aktivitäten ermöglicht. Ein Admin-Dashboard könnte dem Vorstand eine benutzerfreundliche Oberfläche zur Verwaltung der Vereinsdaten bieten und den direkten Datenbankzugriff ablösen. Die Einführung atomarer Transaktionen würde die Konsistenz bei parallelen Punktetransaktionen verbessern. Eine Zwei-Faktor-Authentifizierung könnte zudem den Schutz vor unbefugtem Zugriff erhöhen.

Eine Umstellung von Cron-Jobs auf Webhooks könnte Ereignisse nahezu in Echtzeit abbilden und die Aktualisierung von Events und Ranglisten optimieren. Darüber hinaus sollte der Bot künftig eine feinere Rechteverwaltung sowie eine verbesserte Authentifizierung für die Datenbankanbindung bieten. Die Ergänzung von strukturierten Log-Dateien und einem Monitoring-System würde die Wartbarkeit erhöhen und eine bessere Nachvollziehbarkeit bei Fehlern ermöglichen.

Eine vertiefte Analyse der Gamification-Daten könnte wichtige Erkenntnisse für die langfristige Motivation der Mitglieder liefern und künftige Anpassungen am Punktesystem ermöglichen. Zudem könnte der aktuell manuelle Freigabeprozess für Punktevergaben teilweise automatisiert werden, beispielsweise durch eine Zahlungskontrolle oder definierte Prüfregeln. Dies würde den Workflow beschleunigen, ohne die Integrität des Systems zu gefährden.

Kapitel II
Projekt Dokumentation

9 Projekt Plan

9.1 Zusammenarbeit

Als Projektmanagement-Methode wird Kanban verwendet welches es erlaubt das Projekt agil durchzuführen. Aufgrund der Teamgrösse von zwei Personen wurde sich gegen Scrum entschieden. Der zusätzliche organisatorische Mehraufwand durch Daily Meetings, Sprint Reviews und weitere Scrum Meetings würde keinen grossen Nutzen bringen. Dank der geringen Teamgrösse ist es kein Problem den Überblick über das Projekt zu behalten, sodass eine schlanke Methode wie Kanban eine passende Wahl ist.

Die Zusammenarbeit und Aufgabenaufteilung wird mittels Jira organisiert. Mehr dazu im Abschnitt 11.6 .

9.2 Meetings

Jede Woche findet ein Meeting unter den Teammitgliedern und ein Meeting mit dem Referenten statt.

Im Team-Meeting wird besprochen, welche Aufgaben fertiggestellt wurden und diese gegebenenfalls zusammen angeschaut. Zudem werden die Aufgaben für kommende Woche verteilt.

Beim Meeting mit dem Referenten wird der aktuelle Stand besprochen. Ausserdem kann dieses genutzt werden, um allfällige Fragen zu stellen.

9.3 Minimum Viable Product (MVP)

Das Minimum Viable Product (MVP) beinhaltet die Kernfunktionen des Projektes. Diese sind folgendermassen definiert:

- **Verbindung der Datenbank, Backend und dem Discord-Bot**: Die Verbindung des Backends mit der Datenbank und mit dem Discord-Bot soll vorhanden sein. Das Backend soll Operationen auf der Datenbank ausführen und auf Anfragen vom Discord-Bot antworten können.
- **Mitgliederverwaltung**: Mitglieder sollen manuell in die Datenbank aufgenommen und mit unterschiedlichen Rollen versehen werden können. Der Discord-Bot soll automatisch Benachrichtigungen an Mitglieder senden.
- **Eventkalender**: Der Discord-Bot soll mit dem Google Calendar verknüpft sein und automatisch Erinnerungsnachrichten für dort gespeicherte Events senden.
- **Einfache Gamification**: Das Ziel ist es ein Punktesystemkonzept zu definieren. Mitglieder sollen Punkte für ihr Engagement im Verein erhalten, welche sie für Belohnungen einlösen können. Das System soll gegen Missbrauch geschützt sein und die Mitglieder dazu ermutigen aktiv an Vereins-Events teilzunehmen. Das genaue Konzept ist im Abschnitt 4 zu finden.

9.4 Long-Term Plan

Der Long-Term Plan wurde in 5 verschieden Phasen aufgeteilt. Diese werden folgend genauer beschrieben. Optionale Aufgaben stehen in Klammern.

Phase	Woche	Datum	Ziele	Meilenstein
Inception	1	17.02 - 25.02	Projektplan, Risikomanagement, MVP definieren	M1
Elaboration	2	26.02 - 04.03	Gamificationkonzept	
	3	05.03 - 11.03	Requirements, Architektur	
	4	12.03 - 18.03	Sicherheitskonzept, Tooling	
	5	19.03 - 25.03	Gamification-Umfrage, Testkonzept, DB einrichten, Linux-Server aufsetzen	
	6	26.03 - 01.04	(Admin-Dashboard aufsetzen, Mockups GUI), Backend Discord-Bot	
	7	02.04 - 08.04	Prototyp (Proof of Concept)	M2
Construction	8	09.04 - 15.04	1.2, 2.2, 2.6	
	9	16.04 - 22.04	1.1, 1.4, 2.1, 2.4	
	10	23.04 - 29.04	1.3	
	11	30.04 - 06.05	2.3	
	12	07.05 - 13.05	1.5, (3.1, 3.2, 3.3)	
	13	14.05 - 20.05	1.6, 4.1, 4.2, (3.4, 3.5, 3.6, 2.5)	
	14	21.05 - 27.05	(3.6)	M3
Transition	15	28.05 - 03.06	Dokumentation, Abschluss	
	16	04.06 - 06.06	Fertigstellung und Abgabe Abstract und Dokumentation	M4
Presentation	17	07.06 - 13.06	Präsentation, Poster	M5

Tab. 55: Long-Term Plan

Die Funktionen sind wie folgt definiert:

1. Discord
- **1.1**: Rollenvergabe in Discord
- **1.2**: Events hinzufügen und synchronisieren mit Google Calendar
- **1.3**: Punktevergabe
- **1.4**: Eventbenachrichtigungen
- **1.5**: Datenschutzkonzept umsetzen
- **1.6**: Usability-Tests

2. Backend:
- **2.1**: Rollenverteilung gemäss Mitgliederliste
- **2.2**: Datenbank abfragen
- **2.3**: Gamificationkonzept umsetzen
- **2.4**: Google Calendar API Anbindung und Abspeicherung in Datenbank
- **2.5**: Webhooks zu Vereinswebseite (optional)
- **2.6**: Mit Unit-Tests beginnen

3. Admin-Dashboard (optional)
- **3.1**: Mitglieder verwalten
- **3.2**: Events verwalten
- **3.3**: Rollen verwalten
- **3.4**: Whitelist für Discordbefehle
- **3.5**: Punktesystem verwalten
- **3.6**: Usability-Tests

4. Hosting
- **4.1**: Docker
- **4.2**: Pipelines CI/CD

9.4.1 Inception (Initiierung)

In der Inception-Phase werden die Anforderungen für das Projekt und der Projektplan erstellt, sowie die Risiken identifiziert. In dieser Phase wird zudem die generelle Herangehensweise und die Machbarkeit des Projektes bestimmt.

9.4.2 Elaboration (Ausarbeitung)

In der Elaboration-Phase werden die Anforderungen des Projektes im Detail analysiert. Hier wird das Gamification- und Sicherheitskonzept ausgearbeitet. Die Architektur wird mittels des C4-Modells definiert und ein erster Softwareprototyp wird erstellt. Dieser kann verwendet werden, um erste Systeme zu testen, auf denen dann später aufgebaut werden kann.

9.4.3 Construction (Konstruktion)

Die Construction-Phase wird dazu verwendet das Projekt technisch umzusetzen. In dieser Phase werden die geplanten Features umgesetzt und systematisch getestet. Usability-Tests werden durchgeführt auf dessen Basis Anpassungen vorgenommen werden. Die umgesetzten Features und Tests werden dokumentiert.

9.4.4 Transition (Übergang)

In der Transition-Phase wird das Projekt abgeschlossen. Die Dokumentation wird fertiggestellt und der Abstract wird geschrieben. Die Dokumentation wird in dieser Phase abgegeben.

9.4.5 Presentation (Präsentation)

Da das Projekt neben einer Studienarbeit auch eine Bachelorarbeit ist, wird in der Presentation-Phase die Präsentation erstellt. In dieser Phase arbeitet Marco Schnider, welcher die Bachelorarbeit durchführt, an der Präsentation und am Poster.

9.4.6 Meilensteine

Die folgenden Meilensteine dienen als Hilfe, um die definierten Ziele des Projektes zu erreichen.

M1: Projekt Initiierung
- Die Requirements sind definiert.
- Die Risiken sind definiert.
- Der Projektplan ist definiert.
- Das MVP ist definiert.

M2: Prototypen
- Der erste Prototyp ist erstellt.
- Das Gamificationkonzept, Datenschutzkonzept und Testkonzept ist erstellt.
- Mockups für das Admin-Dashboard sind erstellt.
- Die Datenbank ist aufgesetzt.
- Die Architektur ist definiert.

M3: Fertigstellung des Projektes
- Alle geplanten nicht-optionalen Features sind umgesetzt.
- Alle Funktionen sind erfolgreich getestet.
- Usability-Tests sind durchgeführt und Feedback ist umgesetzt.
- Die Testabdeckung entspricht dem definiertem Wert.

M4: Abgabe

- Die umfassende Dokumentation ist fertiggestellt.
- Das Abstract ist fertiggestellt.

M5: Präsentation

- Die Präsentation ist fertiggestellt.
- Das Poster ist fertiggestellt.

9.5 Risikomanagement

Die folgenden Tabellen zeigen die Risiken des Projektes auf und die Strategien die verwendet werden um diese zu vermindern.

Wahrscheinlichkeit / Schweregrad	1 - Sehr Unwahrscheinlich	2 - Unwahrscheinlich	3 - Gelegentlich	4 - Wahrscheinlich	5 - Oft
4 - Katastrophal	R2				
3 - Kritisch		R3, R7			
2 - Signifikant			R4, R8, R9	R1	
1 - Gering		R5		R6	

Tab. 56: Risiko Matrix

ID	R1
Risiko	Lernkurve
Kommentar	Lernkurve bei der Verwendung von neuen Tools und Technologien.
Präventionsmassnahmen	Es wird so früh wie möglich mit dem Prototypen begonnen, damit genug Zeit ist um das nötige Wissen anzueignen.
Korrekturmassnahmen	Die Features werden in möglichst kleine Aufgaben unterteilt. Wenn ein Teammitglied auf Probleme stösst, kann ein anderes Teammitglied zur Unterstützung hinzugezogen werden.

Tab. 57: Beschreibung Risiko 1

ID	R2
Risiko	Probleme bei Discord
Kommentar	Discord hat Probleme und ist nicht erreichbar.
Präventionsmassnahmen	-
Korrekturmassnahmen	Dieses Risiko muss akzeptiert werden, da nicht dagegen unternommen werden kann.

Tab. 58: Beschreibung Risiko 2

ID	R3
Risiko	Ressourcen limitiert
Kommentar	Ein Teammitglied fällt aus.
Präventionsmassnahmen	Die erledigten Aufgaben werden sauber Dokumentiert.
Korrekturmassnahmen	Kommt es zu einem längeren Ausfall eines Teammitglieds müssen die Arbeiten vom ausgefallenen Teammitglied vom anderen Teammitglied übernommen werden. Der Fokus wird auf Features mit hoher Priorität gelegt.

Tab. 59: Beschreibung Risiko 3

ID	R4
Risiko	Scope Creep
Kommentar	Unklare Definition der Anforderungen, was zu einer Erweiterung des Projekts führt.
Präventionsmass-nahmen	In den Meetings wird der Scope der Aufgaben genau definiert.
Korrekturmass-nahmen	Features mit tiefer Priorität können gestrichen werden.

Tab. 60: Beschreibung Risiko 4

ID	R5
Risiko	Kompatibilitätsprobleme
Kommentar	Tools und Technologien mit verschiedenen Versionen.
Präventionsmass-nahmen	Die verwendeten Versionen von Tools und Technologien werden dokumentiert und jedes Teammitglied verwendet diese.
Korrekturmass-nahmen	Bei unterschiedlichen Versionen wird sich auf eine geeinigt und diese verwendet.

Tab. 61: Beschreibung Risiko 5

ID	R6
Risiko	Zeiteinschätzung
Kommentar	Es kommt zu grossen Abweichungen bei der Zeiteinschätzung.
Präventionsmass-nahmen	Die Zeiteinschätzung wird in den Meeting von den Teammitgliedern besprochen.
Korrekturmass-nahmen	Optionale Features können umgesetzt oder gestrichen werden.

Tab. 62: Beschreibung Risiko 6

ID	R7
Risiko	Änderungen in der Discord API
Kommentar	Es kommt zu fundamentalen Änderungen in der Discord API.
Präventionsmass-nahmen	Funktionen in der API die mit ‚deprecated' markiert sind, werden nicht verwendet.
Korrekturmass-nahmen	Kommt es doch zu fundamentalen Änderungen ausserhalb der ‚deprecated'-Funktionen, muss der Code angepasst werden.

Tab. 63: Beschreibung Risiko 7

ID	R8
Risiko	Manipulation
Kommentar	Das Gamification-System kann missbraucht werden.
Präventionsmass-nahmen	Es werden Usability-Tests durchgeführt um mögliche Lücken im System, die dazu führen könnten das ein Mitglied zu einfach an viele Punkte herankommt, zu erkennen und zu schliessen.
Korrekturmass-nahmen	Es werden Regeln definiert an die sich die Mitglieder halten müssen, tun sie dies nicht können die Punkte zurückgesetzt werden.

Tab. 64: Beschreibung Risiko 8

ID	R9
Risiko	Datenschutz
Kommentar	Datenschutz bedenken in Verbindung mit Discord.
Präventionsmass-nahmen	Es wird ein Datenschutzkonzept erstellt, welches definiert auf welche Daten der Discord-Bot zugreifen kann.
Korrekturmass-nahmen	Benötigt der Discord-Bot bestimmte Nutzerdaten müssen die Nutzer darüber informiert werden.

Tab. 65: Beschreibung Risiko 9

9.5.1 Ausweichplan

Der Ausweichplan dient dazu systematisch auf Risiken zu reagieren, die im Vorfeld nicht bestimmt werden konnten. Der Ablauf ist wie folgt definiert:

1. Das Problem innerhalb einer Aufgabe wird identifiziert und mit dem Teammitglied besprochen.
2. Es wird versucht das Problem so schnell wie möglich zu lösen.
3. Kann das Problem nicht gelöst werden, wird es im nächsten Meeting mit dem Referenten besprochen.

Ziel dieses Ausweichplans ist es, bei auftretenden Problemen, die den Projektverlauf beeinträchtigen, eine klare und strukturierte Vorgehensweise zu gewährleisten. Dadurch kann effizient reagiert und ein reibungsloser Projektfortschritt sichergestellt werden.

10 Arbeitszeitnachweis

In diesem Kapitel wird eine Übersicht über den zeitlichen Aufwand gegeben, der während der Umsetzung des Projektes angefallen ist. Zudem wird die geschätzte Zeit mit der tatsächlich aufgewendeter Zeit verglichen.

10.1 Zeit pro Person

Das folgende Diagramm zeigt die aufgewendete Zeit pro Person. Dabei ist zu beachten, dass Vanessa Alves die Studienarbeit und Marco Schnider die Bachelorarbeit durchgeführt hat.

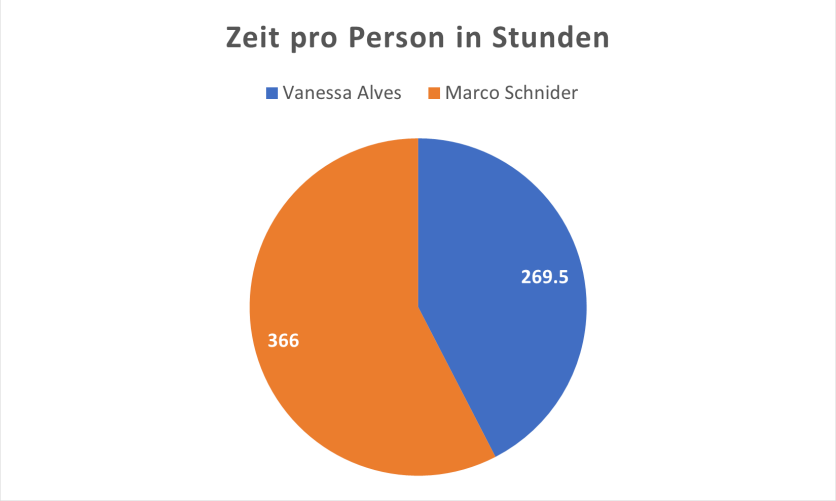


Abb. 42: Zeit pro Person

10.2 Zeit pro Phase

Das folgende Diagramm zeigt die geschätzte und tatsächlich aufgewendete Zeit pro Projektphase und für die Meetings.

Besonders auffällig ist der erhöhte Zeitaufwand bei der Elaboration-Phase. Dies ist insbesondere auf die vertiefte Auseinandersetzung mit dem Datenschutz und dem Gamification-Konzept zurückzuführen. Auch in der Construction-Phase war der Zeitaufwand höher als geplant, dies vor allem wegen unerwarteten Herausforderungen bei der Implementation von Discord-Formularen und bei der Event-Synchronisation.

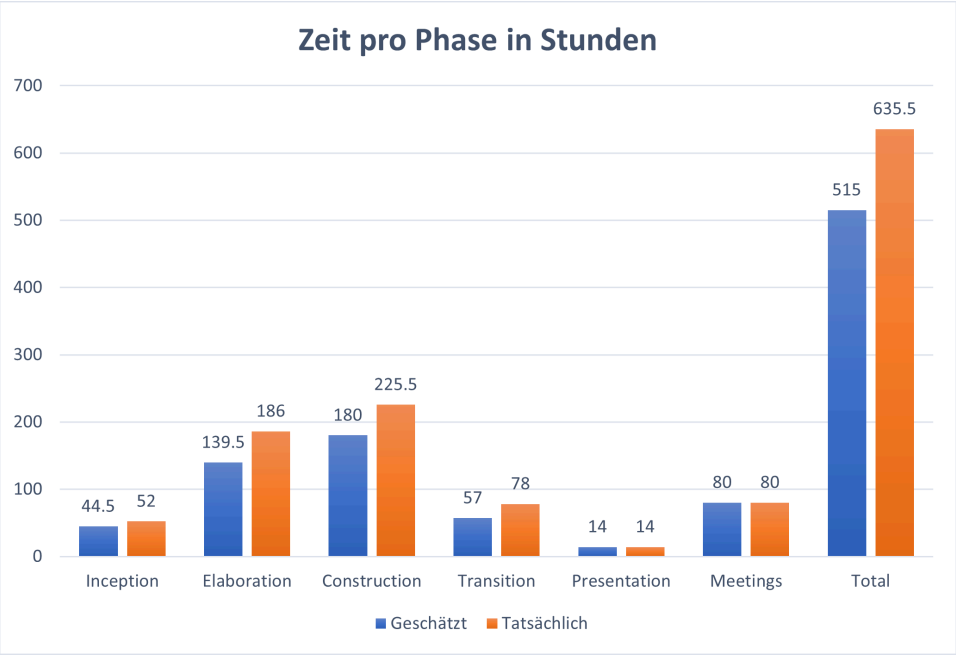


Abb. 43: Zeit pro Phase

11 Tooling und Technologie-Entscheidungen

Dieses Kapitel beschreibt Punkte für das Vorgehen, Regeln und Entscheidungen in den jeweiligen Bereichen dieser Arbeit.

11.1 Dokumentation

Dieses Dokument bildet die schriftliche Dokumentation rund um das Projekt und folgt einem systematischen Vorgang.

11.1.1 Dokumentation Guidlines

1. Dokumentiere auf Deutsch.
2. Dokumentiere mit Typst.
3. Verwende die während diesem Projekt entwickelte Vorlage.
4. Unterteile den Text in eine Typst-Datei pro grösseres Kapitel.
5. Benenne die einzelnen Typst Dateien in Englisch und verwende hierfür kebab-case.
6. Verwende aussagekräftige Titel, welche den Text gut unterteilen.
7. Versehe alle nötigen Titel mit einer kurzen und klaren Referenz-Variable im snacke_case.
8. Verwende eine saubere Formatierung, die dabei, hilft den Text gut lesen zu können.
9. Kennzeichne Referenzen aus externen Quellen als solche.
10. Schreibe für jede Abbildung und Tabelle eine Beschriftung.
11. Schreibe Referenz-Namen der Beschriftungen für Abbildungen und Tabellen im kebab-case.

11.2 Code

Der Code für dieses Projekt wird in der hier beschriebenen Form erstellt.

11.2.1 Code Guidlines

1. Schreibe TypeScript Variablen- und Funktionsnamen im camelCase.
2. Schreibe TypeScript Konstanten in Grossbuchstaben, wobei einzelne Wörter mit Unterstrichen aufgeteilt werden.
3. Schreibe HTML-Element-Bezeichner im kebab-case.
4. Gib Arrays einen Namen im Plural.
5. Schreibe kurze Funktionen mit wenigen Argumenten. Eine Funktion soll nur einen Zweck erfüllen.
6. Verwende let oder const anstelle von var.
7. Schreibe Vergleiche mit === oder !==.
8. Verwende bei Variablen-Zuweisungen Leerzeichen zwischen dem Gleichheitszeichen.
9. Schreibe bei Funktionsheadern die geschweifte Klammer auf die gleiche Zeile.
10. Schreibe Kommentare wo nötig, der Code sollte selbsterklärend geschrieben werden.
11. Verwende für gleiche Konzepte die gleichen Wörter.
12. Überprüfe Änderungen vor dem pushen auf GitLab mit Prettier und ESLint.

11.2.2 Code Setup

Um den Coding-Style einzuhalten werden die Standardregeln von Prettier und ESLint für TypeScript verwendet. Zudem sind bei den Coding Projekten bestimmte Prozesse automatisiert, um die Entwicklung zu erleichtern.

11.3 GitLab

Für die Versionsverwaltung wird GitLab verwendet. Das Projekt ist in drei verschiedene Repositories[16] unterteilt. Eines für die Dokumentation und je eines für den Discord-Bot und das Admin-Dashboard.

[16] https://gitlab.ost.ch/saba-discord_bot

11.3.1 GitLab Guidelines

1. Auf dem Dokumentations-Repository hat jedes Teammitglied seinen eigenen Branch im Format „dev-Initialen“ (z.B. dev-ms)
2. Auf den Code-Repositories (Discord-Bot, Admin-Dashboard) soll für jedes Feature oder Fehler ein jeweiliger Branch im Format „feature-FeatureName“ oder „bug-BugName“ erstellt werden.
3. Das Deployment der Repositories wird je auf einem eigenen Branch namens „deploy“ umgesetzt. Die Sammlung der Commits werden beim mergen des Main-Branches mithilfe der Funktion „Squash“ zusammengefasst, um die Anzahl ähnlicher Commit-Nachrichten zu reduzieren.
4. Bevor Änderungen beim Code in den Develop-Branch gemerged werden, müssen diese von einem anderen Teammitglied überprüft werden.
5. Schreibe Commit-Nachrichten auf Englisch.
6. Beim wöchentlichen Meeting werden die Änderungen vom Develop-Branch in den Main-Branch gemerged.

11.4 Pipelines

Für das automatische Erstellen von PDF und HTML-Dateien werden Pipelines verwendet. Diese werden bei jedem Commit auf den Main-Branch des jeweiligen Repositories ausgeführt.

Das Dokumentations-Repository[17] enthält folgende Jobs:

Job-Name	Beschreibung
protocols_pdf_build	Fasst die einzelnen Markdown-Protokolle in eine Typst Datei zusammen und wandelt diese anschliessend in ein PDF-Dokument um.
main_pdf_build	Generiert aus den Typst Dateien ein PDF-Dokument, welches die komplette Projekt-Dokumentation darstellt.
pages	Erstellt aus der generierten Projekt-Dokumentation eine GitLab Webseite, welche das PDF mithilfe des Browser-internen Viewers darstellt.

Tab. 66: Jobs der Dokumentations-Pipeline

Im Gegensatz zum Dokumentations-Repository enthält das Discord-Bot-Repository[18] nur den folgenden Job in der Pipeline:

Job-Name	Beschreibung
pages	Erstellt aus dem Benutzerhandbuch, welches in Markdown geschrieben ist, eine HTML-Datei, die im Browser angezeigt werden kann.

Tab. 67: Jobs der Discord-Bot-Pipeline

[17] https://gitlab.ost.ch/saba-discord_bot/documentation/-/tree/main?ref_type=heads
[18] https://gitlab.ost.ch/saba-discord_bot/discord-bot/-/tree/main?ref_type=heads

11.5 KI-Tools

KI-Tools werden eingesetzt, um die Effizienz zu steigern und die Qualität der Ergebnisse zu verbessern. Die Einsatzbereiche umfassen:

- **Technische Entscheidung**: Unterstützung bei der Bewertung von Frameworks, Bibliotheken und Architekturansätzen.
- **Programmierung**: Hilfe bei der Fehlersuche, Optimierung und Codegenerierung.
- **Dokumentation**: Unterstützung beim Strukturieren und Formulieren von Textabschnitten, insbesondere für technische Beschreibungen, Kapitelüberschriften und sprachliche Korrektheit.

Die KI wird als Assistenzwerkzeug eingesetzt und dient nicht zur automatisierten Generierung vollständiger Inhalte. Alle durch die KI erzeugten Vorschläge werden kritisch geprüft und bei Bedarf angepasst.

Folgende KI-Tools werden verwendet:

- OpenAI ChatGPT[19]
- Google Gemini[20]
- GitHub Copilot[21]

11.6 Jira

Die Zusammenarbeit und Aufgabenverteilung wird mit Jira[22] verwaltet. Für jede Aufgabe wird im Kanban Board ein Issue erstellt mit den folgenden Merkmalen:

- **Epic**: Die Phase zu dem das Issue gehört.
- **Typ**: Der Typ des Issues (Meeting, Aufgabe, optionale Aufgabe).
- **Geschätzte Zeit**: Die Soll-Zeit des Issues.
- **Verwendete Zeit**: Die tatsächlich aufgewendete Zeit um, den Issue zu beenden.
- **Zugewiesene Person**: Die Person, die das Issue durchführt.

Die Issues werden alle zuerst im Backlog definiert. Im wöchentlichen Team-Meeting werden die Issues, welche in dieser Woche geplant sind in die „Todo“-Spalte geschoben. Wenn die Person, der das Issue zugeteilt wurde, mit der Arbeit beginnt, wird dieser in die „In Progress“-Spalte geschoben. Wurde das Issue umgesetzt, kommt er in die „Review“-Spalte. Alle Issues in der Review Spalte werden im wöchentlichen Meeting besprochen und bei einer erfolgreichen Umsetzung in die „Done“-Spalte verschoben. Falls es noch Verbesserungspotenzial gibt, kommt das Issue zurück in die „Todo“-Spalte.

11.7 Teams

Um Informationen auszutauschen und über ausserordentliche Meetings zu informieren oder nach Hilfe zu fragen, wird Microsoft Teams[23] mit einer eigenen Gruppe verwendet.

11.8 Programmiersprachen

Für die Entwicklung des Discord-Bots stehen verschiedene Programmiersprachen inklusive Bibliotheken für die Verwendung der Discord-API zur Verfügung. (apacheli, 2025)

Zwischen den vielen Möglichkeiten wird TypeScript gewählt, weil das Team damit die meiste Erfahrung hat und somit ein zügiger Start der Entwicklung sichergestellt wird. Zudem ist TypeScript eine geeignete Wahl für die Entwicklung des Discord-Bots, weil es durch die strikte Typisierung die Code-Lesbarkeit und Wartbarkeit verbessert. (Web-Dev-Tutor-Contributors, 2025)

Die TypeScript Discord API Bibliotheken werden anhand der Anzahl Repository-Sterne verglichen, wobei archivierte Bibliotheken wegen der Gefahr von veralteten Versionen ausgeschlossen werden. Die drei am weitesten verbreiteten Bibliotheken mit der höchsten Anzahl werden für die Bot-Entwicklung vorgezogen und genauer analysiert. In diesem Fall

[19] https://chatgpt.com/
[20] https://gemini.google.com
[21] https://github.com/features/copilot
[22] https://www.atlassian.com/software/jira
[23] https://www.microsoft.com/de-ch/microsoft-teams/log-in?market=ch

ist es `discord.js`[24] mit über 25'000 Sternen, `sapphire`[25] mit über 600 und `discordeno.js`[26] mit über 800 Sternen. Dieses Vorgehen stellt sicher, dass die Community, Funktionalität und Dokumentation ausgereift ist.

Discord-Bot Bibliothek	Beschreibung
Discord.js	Die Bibliothek `discord.js` bietet regelmässige Updates, wird offiziell unterstützt und besitzt eine komplette Abdeckung der gesamten Discord API. (Discord.js-Contributors, 2025b)
Sapphire Framework	Die Bibliothek `sapphire` basiert auf `discord.js` und hebt sich ab mit einer sauberen Architektur sowie besserer Modularität. (Sapphire-Contributors, 2025)
Discordeno	Die Bibliothek `discordeno.js` ist eine Alternative zu `discord.js` mit höherer Performance und Skalierbarkeit. (Discordeno-Contributors, 2025)

Tab. 68: Beschreibung von Discord-Bot Bibliotheken für dieses Projekt

Da dieses Team keine vorgängigen Erfahrungen mit der Entwicklung von Discord-Bots hat, wird die Bibliothek `discord.js` gewählt, da sie die grösste Community und viele Ressourcen für eine optimale Unterstützung während der Programmierung bietet.

11.9 Datenbank

Für dieses Projekt wird eine Datenbanklösung benötigt, um Vereinsmitglieder, Rollen, Punkte, Events und weitere Daten effizient zu verwalten. Die nachfolgenden Datenbank-Systeme sind zum einen kostenlos, damit keine Gebühren für den Verein anfallen, und zum anderen den Teammitgliedern bereits bekannt. Aus diesem Grund wurden die folgenden drei Systeme genauer analysiert.

Datenbank-System	Beschreibung
MongoDB Community Edition	Eine dokumentenorientierte NoSQL-Datenbank, die Daten in JSON-ähnlichen Dokumenten speichert. Sie verwendet ein schemaloses Modell und unterstützt horizontale Skalierung. (MongoDB-Contributors, 2024)
PostgreSQL	Eine objektrelationale Datenbank (ORDBMS), die sich durch ACID-Konformität und erweiterbare Funktionen auszeichnet. Sie unterstützt sowohl relationale als auch nicht-relationale Datenstrukturen und Vertikale Skalierung. (PostgrSQL-Contributors, 2025)
SQLite	Eine leichtgewichtige, serverlose SQL-Datenbank, die als Bibliothek in Anwendungen eingebunden wird. Sie ist für lokale Datenspeicherung optimiert und benötigt keine separate Server-Instanz. Es ist keine native Skalierung vorhanden. (SQLite-Contributors, 2023)

Tab. 69: Beschreibung von Datenbank-Systemen für dieses Projekt

Für den Discord-Bot wurde MongoDB gewählt, da es eine flexible und schemalose Speicherung von Daten ermöglicht sowie horizontale Skalierung unterstützt, was für zukünftige Erweiterungen wie ein Gamification-System vorteilhaft ist. Im Gegensatz dazu erfordert PostgreSQL ein festes Schema und eignet sich besser für relationale Daten mit komplexen Beziehungen, was für den dynamischen Bot eher unpraktisch ist. SQLite bietet keine Skalierung und ist bei hoher Schreiblast begrenzt (Hediger, 2025) , was es für eine wachsende Bot-Anwendung weniger geeignet macht. MongoDB bietet daher mehr Flexibilität und Skalierbarkeit für die langfristige Entwicklung des Bots. (MongoDB-Contributors, 2025)

[24] https://github.com/discordjs/discord.js
[25] https://github.com/sapphiredev/framework
[26] https://github.com/discordeno/discordeno

11.10 Deployment

Docker wird für das Container-Management verwendet, um das Deployment auf dem Linux-Server zu vereinfachen. CI/CD mit GitLab automatisiert das Deployment und stellt sicher, dass Änderungen im Code direkt in die Produktivumgebung übernommen werden.

Ein manuelles Deployment wurde verworfen, da es fehleranfällig ist (Jupiterexpress-Contributors, 2025) . Der Einsatz von Kubernetes wurde aufgrund seiner Komplexität für dieses Projekt als überdimensioniert erachtet (Continum-Contributors, 2024) . Weil das Team hauptsächlich Erfahrung mit Docker hat, wurde das als bevorzugte Lösung gewählt, um eine einfache Verwaltung und Bereitstellung der Anwendung zu gewährleisten.

11.11 Hosting

Um den Bot dauerhaft online zu halten und auf Discord-Ereignisse reagieren zu können, ist ein geeignetes Hosting erforderlich. Die Anwendung läuft als Node.js-Service, der kontinuierlich ausgeführt werden muss (Abschnitt 11.8). Es gibt verschiedene Hosting-Optionen, die sich hinsichtlich Skalierbarkeit, Wartung, Kosten und Zuverlässigkeit unterscheiden. Die folgende Tabelle zeigt die in Betracht gezogenen Optionen. Sie wurden ausgewählt, weil entweder die Teammitglieder bereits Erfahrungen mit diesen Anbietern haben oder ein einfaches Deployment eines Discord-Bots über diese Anbieter möglich ist.

Hosting-Option	Anbieter	Beschreibung
Cloud-Plattformen	AWS EC2 (Johnson, 2024)	Diese Plattformen bieten hohe Skalierbarkeit und Flexibilität, sind jedoch komplex in der Einrichtung und können je nach Nutzung hohe Kosten verursachen. Sie eignen sich besonders für grössere Anwendungen. (Amazon-Web-Services-Contributors, 2024a; 2024b; Google-Contributors, o. J.)
	AWS Lambda (Juszko, 2022)	
	Google Cloud (Saltonstall & Stanke, 2021)	
	Heroku (H., 2021)	Dieser Anbieter ist einfach zu verwenden für kleine Anwendungen, jedoch mit Einschränkungen und es ist nicht kostenlos nutzbar. (Heruko-Contributors, 2025)
Discord-Hosting-Services	Replit (Ritza-Contributors, 2022)	Diese Plattformen sind besonders für kleine Bots geeignet. Sie besitzen eine einfache Einrichtung, sind jedoch bei konzentrierter Nutzung mit Kosten verbunden und haben eingeschränkte Ressourcen sowie weniger Kontrolle. (Railway-Contributors, 2025; Replit-Contributors, o. J.)
	Railway (Bongers, 2022)	
Self-Hosting	Raspberry Pi (lokal) (Robu-Contributors, 2024)	Eine kostengünstige Variante, hat aber eine eingeschränkte Leistung und Verfügbarkeit durch die eigene Bereitstellung.
	Virtueller Server (FH OST) (Untersander, o. J.)	Ist in einer stabilen Umgebung mit kostenlosem Zugang für Studierende, allerdings ist die Anwendung einzig im Schul-Netz aufrufbar.
	Vereinsserver	Damit ist die volle Kontrolle gewährleistet und es fallen keine externen Kosten auf, jedoch entsteht ein grösserer Wartungsaufwand.

Tab. 70: Beschreibung von Hosting-Optionen für dieses Projekt

Railway wäre technisch eine gute Wahl, da es sich einfach in die GitLab CI/CD integrieren lässt und ausreichend Ressourcen für einen Discord-Bot bereitstellt. Jedoch entsteht bei dieser Plattform, Replit sowie den anderen Cloud-Plattformen monatliche Kosten, weshalb diese Optionen ausgeschlossen wurden. Ausserdem ist der Datenschutz fragwürdig, da die Server je nach Anbieter ausserhalb der EU liegen und damit potenziell der Datenschutz nicht sichergestellt ist. Abschnitt 3.5

Da ein lokaler Raspberry Pi Server leistungsschwächer ist und somit die Chancen einer Überlastung höher, wird diese Möglichkeit ausgeschlossen. Ein virtueller Server der FH OST wird ebenfalls nicht gewählt, weil der Discord-Bot von aussen für den Discord-Server sichtbar sein muss und dies ohne weitere Abklärung mit der Fachhochschule nicht möglich ist.

Stattdessen wird der bereits vorhandene Linux-Server des Vereins genutzt, der im Fall von EGSwiss Abschnitt 1 auch für das Hosting der Vereinswebseite dient. Der Mehraufwand für Setup und Wartung wird akzeptiert, um die laufenden Kosten für den Verein zu minimieren und um den einen optimalen Datenschutz zu gewährleisten.

Kapitel III
Glossar und Abkürzungsverzeichnis

Opt-In	Ein Verfahren, bei dem Nutzer aktiv ihre Zustimmung zur Verarbeitung oder Speicherung ihrer Daten geben müssen.
Datenschutz-Grundverordnung (DSGVO)	Eine EU-weite Datenschutz-Grundverordnung, die den Schutz personenbezogener Daten und die Rechte der Nutzer auf Datenkontrolle festlegt.
Webhook	Eine automatisierte Methode, mit der ein System Ereignisse oder Daten an ein anderes System sendet, sobald eine bestimmte Aktion eintritt.
CI/CD	Continuous Integration (CI) und Continuous Deployment/Delivery (CD). Ein automatisierter Prozess, bei dem Codeänderungen kontinuierlich integriert, getestet und automatisch oder mit manueller Freigabe in die Produktionsumgebung publiziert werden.
End-to-End (E2E)	End-to-End (E2E) Tests überprüfen den gesamten Anwendungsablauf von Anfang bis Ende, indem sie das System aus der Perspektive eines echten Nutzers testen, um sicherzustellen, dass alle Komponenten wie erwartet zusammenarbeiten.
Datenbank-Zuweisungen (one-to-many / many-to-many)	Beschreibt die Art der Beziehung zwischen zwei Entitäten in einem Datenmodell. Eine 1:N-Beziehung (one-to-many) bedeutet, dass ein Datensatz in der ersten Entität mehreren Datensätzen in der zweiten Entität zugeordnet sein kann. Eine N:M-Beziehung (many-to-many) beschreibt eine Beziehung, bei der mehrere Datensätze in der ersten Entität mehreren Datensätzen in der zweiten Entität zugeordnet sein können.
Time-To-Live (TTL)	Eine Funktion zur automatischen Löschung von temporären Daten nach einer definierten Zeitspanne. Wird häufig in Datenbanken oder Caches eingesetzt, um sicherzustellen, dass veraltete oder nicht benötigte Daten nach Ablauf dieser Zeitspanne automatisch entfernt werden.
Kassier	Eine Vorstands-Rolle im Verein, die für die Verwaltung der Finanzen zuständig ist.
Ephemeral Message	Nachrichten in Discord, die nur für die jeweilige Person sichtbar sind und nach kurzer Zeit automatisch gelöscht werden.
League of Legends	Ein kompetitives Mehrspieler-Computerspiel, in dem zwei Teams gegeneinander antreten.
Think-Aloud	Die Think-Aloud-Methode ist eine qualitative Forschungsmethode zur Usability-Evaluation. Dabei werden Testpersonen gebeten, ihre Gedanken während der Interaktion mit einem System laut auszusprechen. Dies ermöglicht es den Forschenden, Einblicke in die Denkprozesse der Nutzer zu gewinnen, potenzielle Missverständnisse zu erkennen und das Verhalten besser zu verstehen.

GPG	GPG (GNU Privacy Guard) ist ein Werkzeug zur sicheren Verschlüsselung und Signierung von Dateien und Nachrichten. Es implementiert den OpenPGP-Standard (RFC 4880) und bietet eine plattformübergreifende Lösung für kryptographische Aufgaben.
Role-Based Access Control (RBAC)	Role-Based Access Control (RBAC) ist ein Sicherheitskonzept, bei dem der Zugriff auf Ressourcen anhand von Rollen vergeben wird. Benutzer erhalten dabei bestimmte Rollen, die jeweils unterschiedliche Berechtigungen für Funktionen und Daten gewähren. So können Administratoren granular steuern, wer auf welche Teile eines Systems zugreifen darf.
Zwei-Faktor-Authentifizierung (2FA)	Zwei-Faktor-Authentifizierung (2FA) ist ein Sicherheitsmechanismus, bei dem zur Anmeldung an einem System zwei verschiedene Authentifizierungsfaktoren abgefragt werden. Dies können z. B. ein Passwort und ein einmaliger Code (z. B. aus einer App oder per SMS) sein. 2FA erhöht die Sicherheit erheblich, da ein Angreifer beide Faktoren kennen oder besitzen müsste, um sich erfolgreich anzumelden.
Cron-Job	Ein Cron-Job ist ein zeitgesteuerter Prozess in Unix- und Linux-Systemen, der bestimmte Aufgaben (z. B. Skripte oder Befehle) zu festgelegten Zeiten oder Intervallen automatisch ausführt. Er wird häufig zur Automatisierung von wiederkehrenden Aufgaben wie Backups oder Systemüberprüfungen verwendet.

Kapitel IV
Literaturverzeichnis

Amazon-Web-Services-Contributors. (2024a,). *Amazon EC2*. https://aws.amazon.com/de/ec2/

Amazon-Web-Services-Contributors. (2024b,). *AWS Lambda*. https://aws.amazon.com/de/lambda/

apacheli. (2025,). *Discord API Libraries*. https://github.com/apacheli/discord-api-libs

Bongers, C. (2022,). *Hosting a discord bot on Railway*. https://daily-dev-tips.com/posts/hosting-a-discord-bot-on-railway/

Continum-Contributors. (2024,). *Was macht Kubernetes so komplex und herausfordernd?*. https://www.continum.net/kubernetes-komplexitaet/

Discord-Contributors. (2025,). *OAuth2*. https://discord.com/developers/docs/topics/oauth2

Discord.js-Contributors. (2025a,). *Deferred responses*. https://discordjs.guide/slash-commands/response-methods.html#deferred-responses

Discord.js-Contributors. (2025b,). *The most popular way to build Discord bots.* https://discord.js.org/

Discordeno-Contributors. (2025,). *Its time to ditch Eris and Discord.js*. https://discordeno.js.org/

Google-Contributors. *Wir stellen vor: Die neue Art zu clouden.* Abgerufen 17. März 2025, von https://cloud.google.com/?hl=de

H., L. (2021,). *Create & host a Discord bot with Heroku in 5 min*. https://medium.com/@linda0511ny/create-host-a-discord-bot-with-heroku-in-5-min-5cb0830d0ff2

Hediger, P. (2025,). *Concurrency-Probleme mit SQLite: Ein Erfahrungsbericht von SkyPilot*. https://www.computerworld.ch/software/big-data/concurrency-probleme-sqlite-erfahrungsbericht-skypilot-2951842.html

Heruko-Contributors. (2025,). *Getting Started on Heroku with Node.js*. https://devcenter.heroku.com/articles/getting-started-with-nodejs?singlepage=true

Johnson, M. (2024,). *Deploying a Discord Bot using AWS EC2*. https://miguel-codes.medium.com/deploying-a-discord-bot-using-aws-ec2-450d85f8e0af

Jupiterexpress-Contributors. (2025,). *Manuelles Deployment*. https://jupiterexpress.de/glossary/manuelles-deployment/

Juszko, J. (2022,). *Serverless Discord Bot on AWS in 5 Steps*. https://medium.com/better-programming/serverless-discord-bot-on-aws-in-5-steps-956dca04d899

kynthia. (2024,). *Slash Commands FAQ*. https://support-apps.discord.com/hc/en-us/articles/26501837786775-Slash-Commands-FAQ

Looyestyn, J., Kernot, J., Boshoff, K., Ryan, J., Edney, S., & Maher, C. (2017). Does gamification increase engagement with online programs? A systematic review. *PloS one*, *12*(3), e173403–e173403. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0173403

MongoDB-Contributors. (2024,). *Introduction to MongoDB*. https://www.mongodb.com/docs/manual/introduction/

MongoDB-Contributors. (2025,). *SQL- im Vergleich zu NoSQL-Datenbanken*. https://www.mongodb.com/de-de/resources/basics/databases/nosql-explained/nosql-vs-sql

Nielsen, J. (2006,). *The 90-9-1 Rule for Participation Inequality in Social Media and Online Communities*. https://www.nngroup.com/articles/participation-inequality/

PostgrSQL-Contributors. (2025,). *What Is PostgreSQL?*. https://www.postgresql.org/docs/current/intro-whatis.html

Railway-Contributors. (2025,). *Shipping great products is hard. Scaling infrastructure is easy.* https://railway.com/

Replit-Contributors. *Getting Started*. Abgerufen 17. März 2025, von https://docs.replit.com/getting-started/intro-replit

Ritza-Contributors. (2022,). *Building a Discord bot with Node.js and Replit*. https://dev.to/ritza/building-a-discord-bot-with-nodejs-and-replit-3i7f

Robu-Contributors. (2024,). *Host your own Discord Bot on Raspberry Pi 5*. https://robu.in/host-your-own-discord-bot-on-raspberry-pi-5/

Sailer, M., & Homner, L. (2020). The Gamification of Learning: a Meta-analysis. *Educational psychology review*, *32*(1), 77–112. https://link.springer.com/article/10.1007/s10648-019-09498-w

Saltonstall, M., & Stanke, D. (2021,). *Build and run a Discord bot on top of Google Cloud*. https://cloud.google.com/blog/topics/developers-practitioners/build-and-run-discord-bot-top-google-cloud?hl=en

Sapphire-Contributors. (2025,). *Sapphire - Imagine a Framework*. https://www.sapphirejs.dev/

SQLite-Contributors. (2023,). *About SQLite*. https://www.sqlite.org/about.html

Untersander, J. *Department of Computer Science - Virtual Server Kiosk*. Abgerufen 17. März 2025, von https://robu.in/host-your-own-discord-bot-on-raspberry-pi-5/

Web-Dev-Tutor-Contributors. (2025,). *8 Typescript Vorteile, die Sie nicht missen sollten*. https://www.webdevtutor.net/blog/typescript-vorteile

Kapitel V
Abbildungsverzeichnis

Kapitel VI
Tabellenverzeichnis

Kapitel VII
Code-Listings

Kapitel VIII
Anhang

12 Externer Zugriff auf die Datenbank

Dieses Kapitel erläutert die technische und organisatorische Umsetzung des externen Zugriffs auf die Datenbank, die zur Verwaltung von Mitgliederdaten, Events und Gamification-Daten genutzt wird. Ziel ist es, eine einfache und zugleich sichere Verwaltung der Datenbankinhalte zu ermöglichen.

12.1 Hintergrund

Die Anwendung verwendet MongoDB als Datenbanktechnologie, um Vereinsdaten wie Mitgliederdaten, Events, Rollen und Punkte zu speichern. Um Vorstandsmitgliedern oder Admins die Möglichkeit zu geben, bei Bedarf auf die Datenbank zuzugreifen, wird MongoDB Compass eingesetzt. Dieses Tool bietet eine grafische Benutzeroberfläche für die Verwaltung der Datenbankinhalte.

12.2 Aktuelle Umsetzung

In der aktuellen Version ist der externe Zugriff auf die Datenbank über MongoDB Compass für autorisierte Personen (z. B. Vorstandsmitglieder mit IT-Kenntnissen) möglich. Für die Authentifizierung wird ein einfaches Passwortschutzsystem verwendet. Dabei wird beim Einrichten des Discord-Bots auf dem Vereins-Server ein Benutzername und ein Passwort in den Umgebungsvariablen festgelegt, die dann für den Zugriff über Compass genutzt werden müssen. Eine genaue erklärung hierzu ist im README des Discord Bots zu finden (vgl. Anhang README.pdf).

Diese Lösung ermöglicht es, administrative Aufgaben wie das Anpassen von Mitgliedsdaten, die manuelle Korrektur von Punkteständen oder die Überprüfung von Eventdaten ohne direkten Zugriff auf den Server selbst vorzunehmen.

12.3 Sicherheit und Einschränkungen

Das aktuelle Passwortschutzsystem bietet einen grundlegenden Schutz vor unbefugtem Zugriff, ist jedoch nur eine Basisschutzmassnahme. Für den produktiven Einsatz wird empfohlen, zusätzliche Sicherheitsmechanismen zu implementieren, wie zum Beispiel:

- Zwei-Faktor-Authentifizierung (2FA) , um die Sicherheit des Logins zu erhöhen.
- IP-Whitelisting, um nur autorisierte IP-Adressen den Zugriff zu gestatten.
- Rollenbasierte Berechtigungen innerhalb der Datenbank, um nur bestimmte Funktionen für bestimmte Nutzer freizugeben.

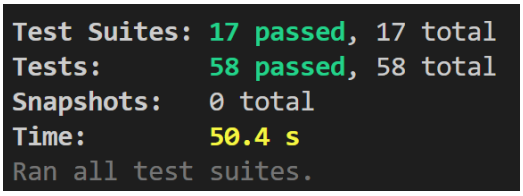
Darüber hinaus sollte geprüft werden, ob der Datenbankzugriff grundsätzlich auf das lokale Netzwerk beschränkt werden kann, um Risiken von aussen zu minimieren.

12.4 Empfehlung für zukünftige Versionen

Langfristig sollte der externe Zugriff auf die Datenbank durch ein Admin-Dashboard innerhalb des Discord-Bots ersetzt werden. Dieses könnte die wichtigsten administrativen Funktionen (z. B. Mitgliederverwaltung, Punktestand anpassen) über eine gesicherte Benutzeroberfläche bereitstellen, ohne dass direkter Datenbankzugriff erforderlich ist. Damit könnte die Angriffsfläche reduziert und die Benutzerfreundlichkeit für Vorstandsmitglieder verbessert werden.

Zusätzlich sollten alle Datenbankzugriffe protokolliert werden, um im Falle von Fehlfunktionen oder Sicherheitsvorfällen eine Nachverfolgbarkeit sicherzustellen.

13 Testreports

ID	**T1**
Datum	01.06.2025
Input	Die Befehl (`npm run test`) für die Durchführung des Unit-Tests wird gestartet.
Erwartetes Resultat	Alle Tests sind erfolgreich.
Tatsächliches Resultat	Alle Tests sind erfolgreich.  **Abb. 44:** Resultat der Unit-Tests Im Fokus der Tests stand die Sicherstellung, dass Fehler bei Befehlen den Discord-Bot nicht zum Absturz bringen. Folgende Befehle wurden getestet: • claimtask • completetask • createtask • donation • getdata • help • intro • join • leave • createleaderboard • ping • score • events • linkcalendar • synccalendar • syncroles • syncusers Folgender Befehl wurde nicht getestet: • rewards

Tab. 71: Test Report: Unit-Tests

ID	T2
Datum	20.05.2025
Input	Durchführung des Usability-Tests mit drei Zielgruppen: Nichtmitglied, Mitglied und Vorstand Die Tests wurden auf einem dedizierten Discord-Testserver durchgeführt. Jede Testsitzung dauerte je nach Zielgruppe ca. 30 Minuten bis maximal zwei Stunden. Der Test wurde nach dem Think-Aloud-Prinzip durchgeführt und mit Fragebögen ergänzt.
Erwartetes Resultat	Der Discord-Bot ist intuitiv bedienbar. Die getesteten Funktionen sind verständlich und einfach zu bedienen. Die wichtigsten Use Cases sind ohne grössere Probleme nutzbar.
Tatsächliches Resultat	Insgesamt wurde eine hohe Benutzerfreundlichkeit festgestellt, jedoch mit einzelnen Verbesserungspotenzialen: Folgende Punkte werden im Discord-Bot umgesetzt: • Die Sprache von Befehlen und Ausgaben wird komplett auf Deutsch umgestellt • Der Discord-System-Kanal wird nicht mehr für die Bot-Ausgabe verwendet • Strukturierte Vorstands-Kanal-Unterteilung (Bewerbungen, Belohnungen und Aufgaben) • Transaktionen bei getData korrekt anzeigen („+“ beim Hinzufügen oder „-“ bei Punkteabzug) • Die Währung bei Zahlungsmethoden mit „CHF“ kennzeichnen • Die Anzeigedauer der AGB verlängern • Dem austretenden Mitglied eine Privatnachricht mit einer Austritts-Bestätigung senden • Dem Vorstand eine Bestätigungsnachricht bei einem Vereinsaustritt senden • Das Lösch-Datum auf DD.MM.YYYY anpassen • Wiederholte Mitglieder-Selects bei Spenden verhindern • Ein Eingabefeld für die Punkte-Definition bei Spenden ergänzen • Der Modal-Titel bei Spenden anpassen • Eine einheitliche Event-Nachricht erstellen • Sichergestellen, dass die Rangliste Transaktionen korrekt nach Zeitraum aufsummiert • Punktegleichheit in der Rangliste berücksichtigen • Die Ranglisten-Darstellung optimieren • Die Rangliste auch ohne Einträge anzeigen • Den Beanspruchungs-Button deaktivieren, wenn die maximale Aufgabenteilnehmer erreicht • Den Beanspruchungs-Button entfernt, wenn die Aufgaben-Deadline erreicht ist • Die Aufgaben-Embed-Darstellung ohne ID im Titel anzeigen • Die Aufgaben-Deadline auf 23:55 Uhr setzen • Optionale Felder klarer anschreiben • Nur Verantwortliche sollen die Annahme / Ablehnungs-Buttons anklicken können • Die Intro- und Willkommens-Texte anpassen und Neuling-freundlich machen • Ein Benutzerhandbuch erstellen und den Link dazu im Intro eingefügen • Den Hilfe-Befehl je nach Rolle anzeigen (Vorstands-Befehle nur dem Vorstand zeigen) • Eine Liste aller Belohnungen als Übersicht ergänzen Folgende Punkte werden aus Zeitgründen und nach Priorisierung nicht umgesetzt: • Daten von anderen Usern aufrufen als Vorstand • Möglichkeit zur Bearbeitung eigener personenbezogener Daten • Neuer Button bei der Bewerbungsnachricht mit automatischer Zahlungserinnerung • Ein Sicherheits-Check vor Annahme einer Bewerbung (Hinweis auf Zahlungskontrolle) • Austrittsgrund angeben als Mitglied • showDonations: Alle erledigten Spenden anzeigen • Bearbeiten/Löschen von Ranglisten • showTasks: Alle offenen Aufgaben anzeigen Die Ergebnisse der Usability-Tests sind im Dokument `Usability Test - Formular.pdf` detailliert dargestellt.

Tab. 72: Test Report: Usability-Tests

ID	T3
Datum	22.05.2025
Input	Alle Befehle wurden manuell mit einer Stoppuhr drei mal durchgeführt. Das Resultat entspricht der Durchschnittsantwortzeit des jeweiligen Befehls.
Erwartetes Resultat	Alle Befehle geben im Durchschnitt innerhalb von 2 Sekunden eine Antwort.
Tatsächliches Resultat	Folgend eine Auflistung aller Befehle und deren durchschnittlicher Antwortzeit: • ping: 0.89s • linkcalendar: 1.08s • events: 1.68s • createtask: 0.96s • synccalendar: 4.25s • join: 1.11s • completetask: 1.13s • createleaderboard: 0.89s • score: 1.01s • rewards: 1.43s • syncroles: 1.25s • syncusers: 1.34s • getdata: 1.08s • donation: 1.01s • leave: 1.34s • help: 0.94s • intro: 1.15s Durchschnittliche Antwortzeit: 1.33s Von insgesamt 17 Befehlen lieferten 16 innerhalb von 2 Sekunden eine Antwort, was einer Erfolgsquote von 94.12% entspricht. Das angestrebte Ziel von 95% wurde damit knapp nicht erreicht. Dies liegt vor allem am `synccalendar`-Befehl, da dieser auf die Google Calendar API und auf die Discord Events zugreifen muss.

Tab. 73: Test Report: Integration-Tests

ID	T4
Datum	01.06.2025
Input	Alle funktionalen und nicht-funktionalen Requirements manuell überprüfen.
Erwartetes Resultat	Alle Requirements wurden erfolgreich umgesetzt.
Tatsächliches Resultat	Die Resultate der einzelnen Requirements können in Abschnitt 2.4 und Abschnitt 2.6 gefunden werden.

Tab. 74: Test Report: System-Tests

14 Deployment

Die Anwendung wird auf einem externen Linux-Server (VPS) betrieben und kann über manuelle Befehle deployed und verwaltet werden. Für den Produktivbetrieb wird ein Node.js-Prozess verwendet, der über Docker isoliert betrieben wird.

14.1 Build-Prozess

Vor der Auslieferung muss der TypeScript-Code `npm run build` kompiliert werden. Das erzeugt einen „dist/“-Ordner, der den ausführbaren Bot enthält.

14.2 Dockerisierung

Zur Containerisierung der Anwendung wird das bereitgestellte Dockerfile sowie eine docker-compose.yml verwendet. Damit können sowohl der Discord-Bot als auch die benötigte MongoDB-Datenbank als separate Container gestartet werden.

Die Umgebungsvariablen (z. B. Discord-Token, Discord-Guild) werden aus einer „.env“-Datei gelesen. Eine Beispielkonfiguration befindet sich im README des Projektes (vgl. Anhang `README.pdf`).

15 Screenshots Discord-Bot

Im Folgenden werden die Screenshots dokumentiert, welche die Nutzung des Discord-Bots innerhalb des Projektes illustrieren. Sie dienen der Veranschaulichung der Funktionalitäten und des Nutzerflusses.

15.1 Discord-Server Einstellungen

Damit der Bot die Rollenzuweisung für die Vorstandsebene vornehmen kann, muss er hierarchisch höher positioniert sein als die Vorstandsrolle. Andernfalls ist eine automatische Rollenzuweisung nicht möglich. Der Bot verfügt zudem über Administratorrechte, was sicherheitstechnisch kritisch bewertet werden kann und daher eine besondere Beachtung bei der Rechtevergabe erfordert.

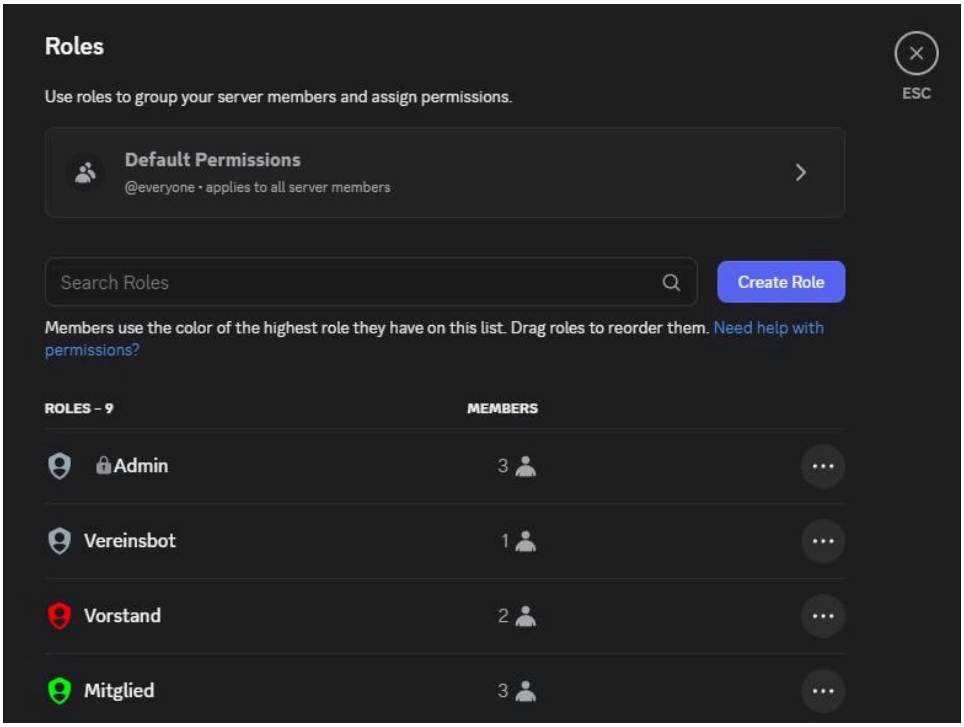


Abb. 45: Abbildung der Discord-Servereinstellungen

15.2 Einladung und Willkommensnachricht

Beim ersten Betreten des Discord-Servers wird das neue Mitglied zunächst in den Regel-Kanal geleitet und erhält eine private Willkommensnachricht. Diese informiert über die Regeln, Datenschutzrichtlinien und die grundlegende Nutzung des Bots. Im Bot-Kanal stehen alle Slash-Befehle zur Verfügung. Befehle, für die das Mitglied keine Berechtigung besitzt, werden automatisch verweigert. Mit dem `/help`-Befehl erhält das Mitglied eine Übersicht aller verfügbaren Befehle, wobei der Vorstand zusätzliche Admin- und Vorstandsfunktionen einsehen kann.

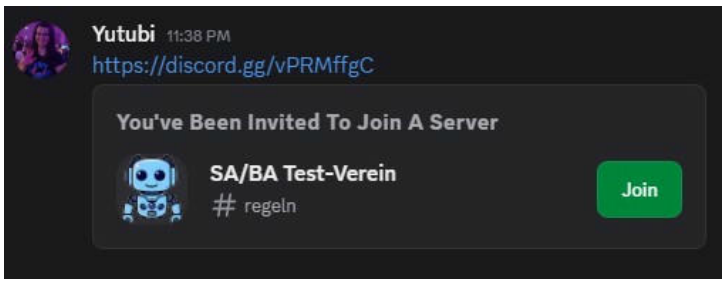


Abb. 46: Übersicht der Willkommensnachricht im privaten Chat des neuen Mitglieds

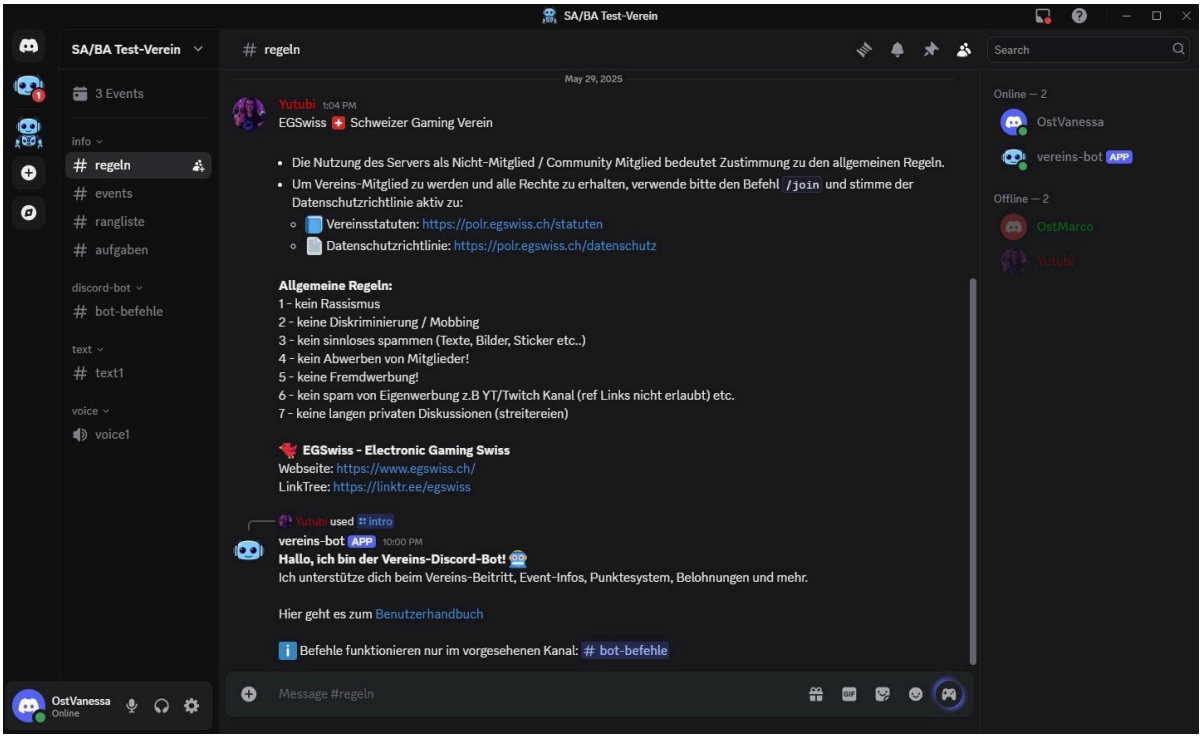


Abb. 47: Screenshot der Bot-Kanal-Übersicht mit Slash-Befehlen und Interaktionsmöglichkeiten

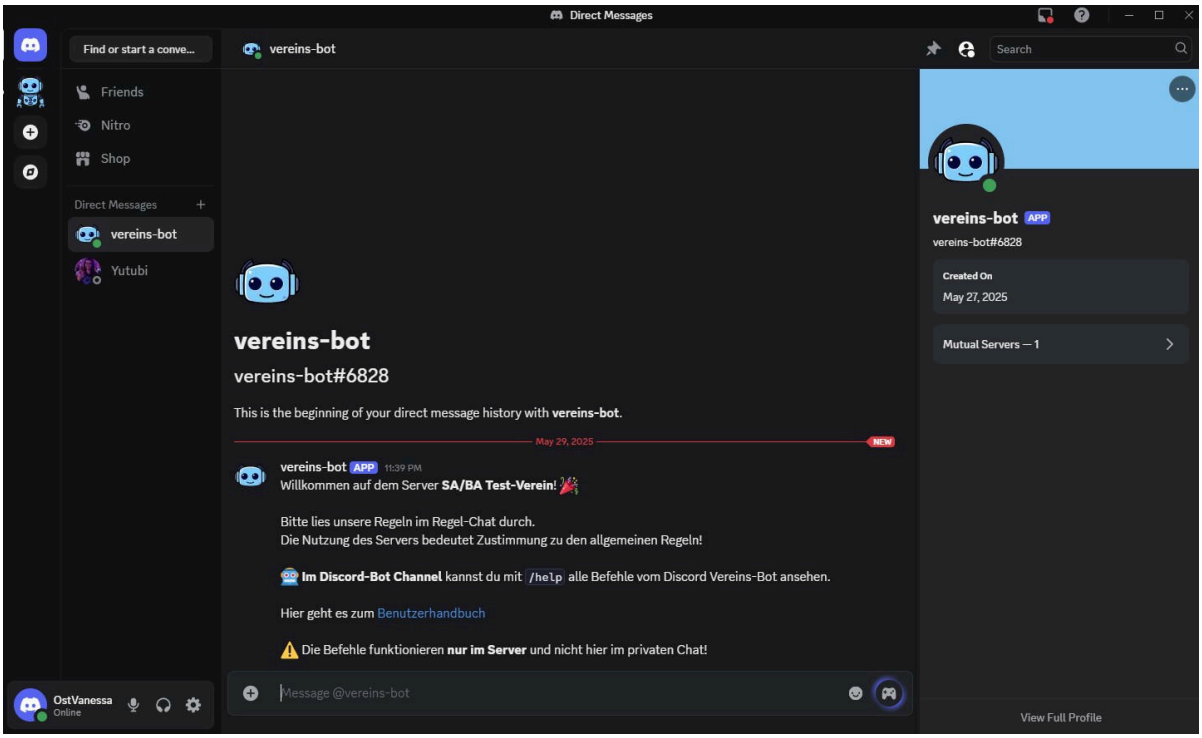


Abb. 48: Darstellung der interaktiven Willkommensnachricht mit Links zu den Datenschutzrichtlinien

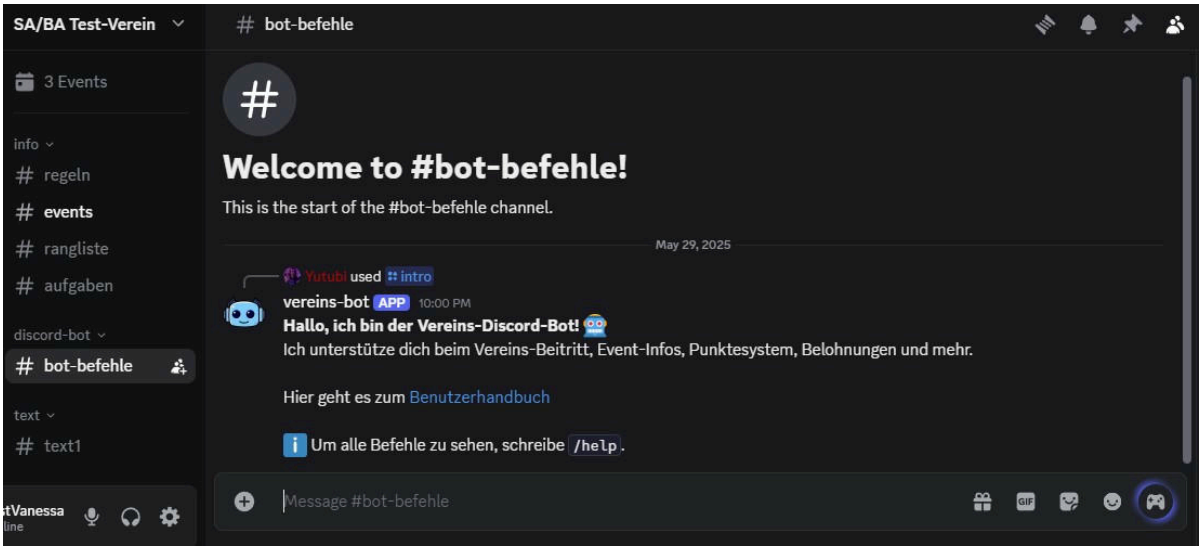


Abb. 49: Anzeige der ersten Willkommensnachricht für neue Mitglieder mit Hinweisen zur Bot-Benutzung

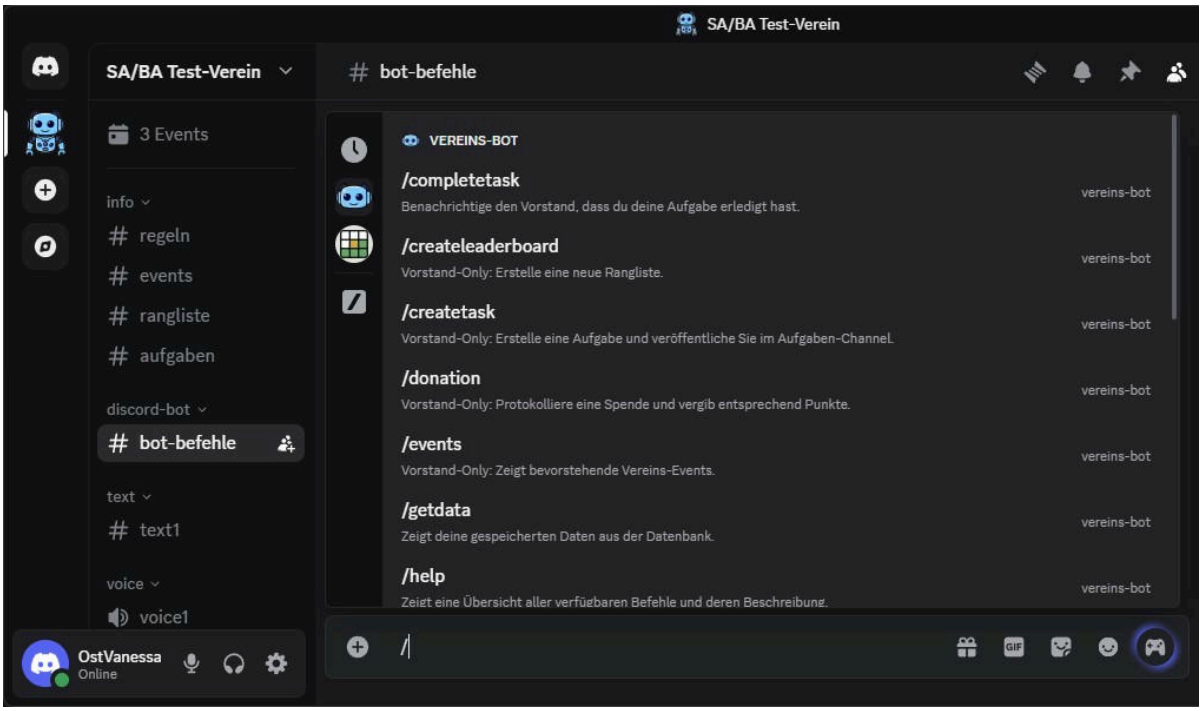


Abb. 50: Übersicht des `/help`-Befehls für alle Rollen mit verfügbaren Befehlen

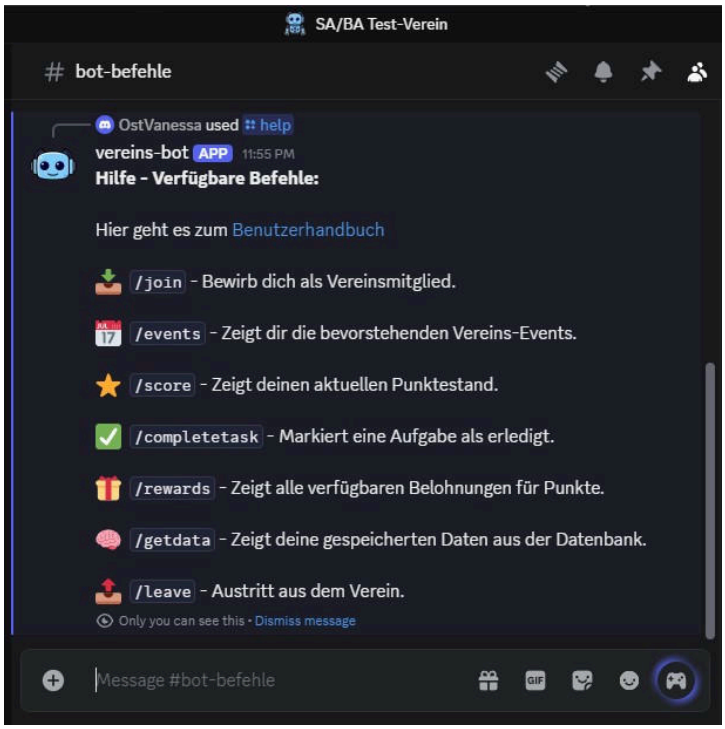


Abb. 51: Anzeige der erweiterten Befehlsliste für Vorstandsmitglieder

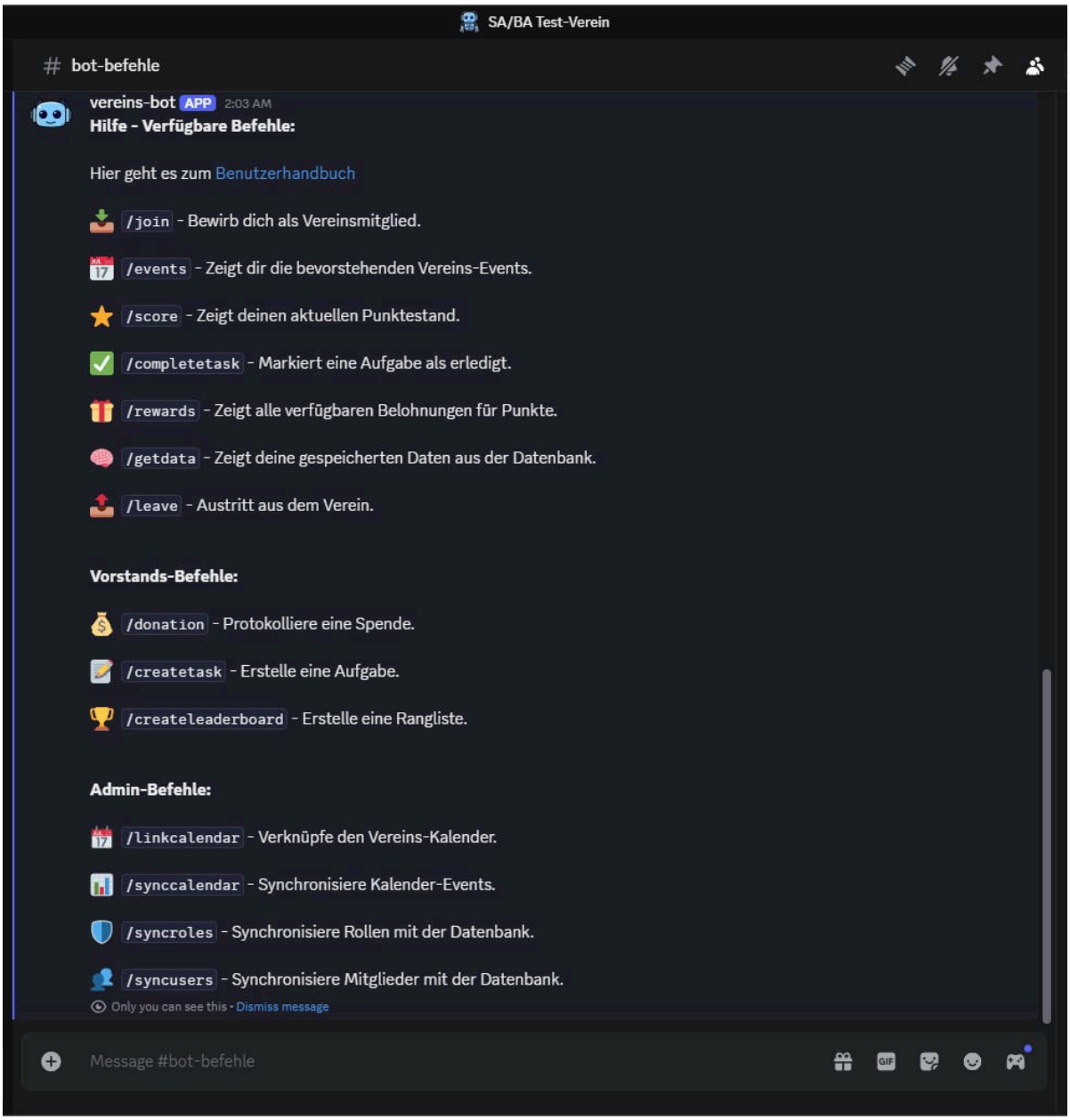


Abb. 52: Ausgabe der Admin- und Vorstandsbefehle für autorisierte Mitglieder

15.3 Synchronisierung mit der Datenbank

Die Synchronisation von Rollen und Nutzerdaten wird über die Slash-Befehle `/syncroles` und `/syncusers` initiiert. Diese Befehle prüfen die Konsistenz der Datenbankeinträge mit dem aktuellen Status im Discord-Server und führen bei Abweichungen entsprechende Updates durch.

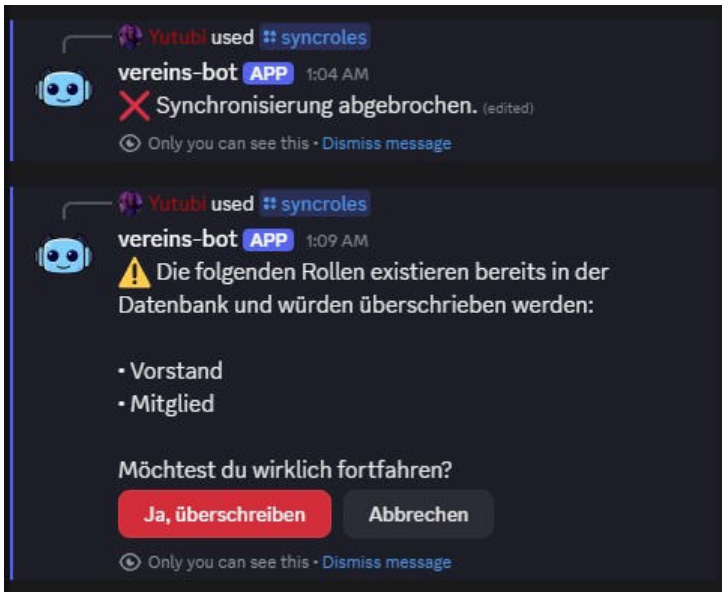


Abb. 53: Synchronisierung der Rollen aus Discord mit der Datenbank

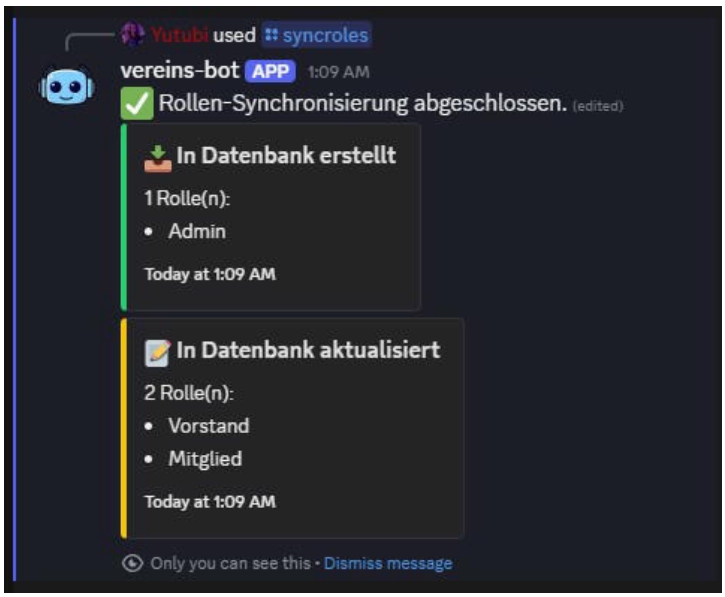


Abb. 54: Ergebnis der Rollensynchronisation mit Statusanzeige

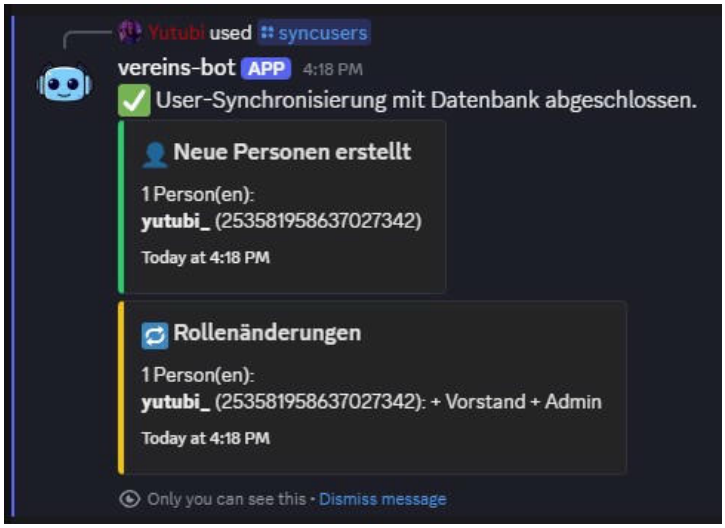


Abb. 55: Synchronisation der Benutzerdaten zwischen Discord und der Datenbank

15.4 Ein- und Austritt als Vereinsmitglied

Mit dem Befehl `/join` können Nicht-Mitglieder dem Verein beitreten. Der Prozess umfasst eine zweistufige Anmeldung mit Bestätigung der Datenschutzrichtlinien sowie die Übermittlung der Zahlungsinformationen. Nach Eingang des Mitgliedsbeitrags erfolgt eine Prüfung durch den Vorstand. Die Entscheidung (Annahme oder Ablehnung) wird der Person privat mitgeteilt. Der Befehl `/leave` ermöglicht den Austritt, wobei die Daten nach drei Monaten automatisch gelöscht werden, sofern kein Wiedereintritt erfolgt.

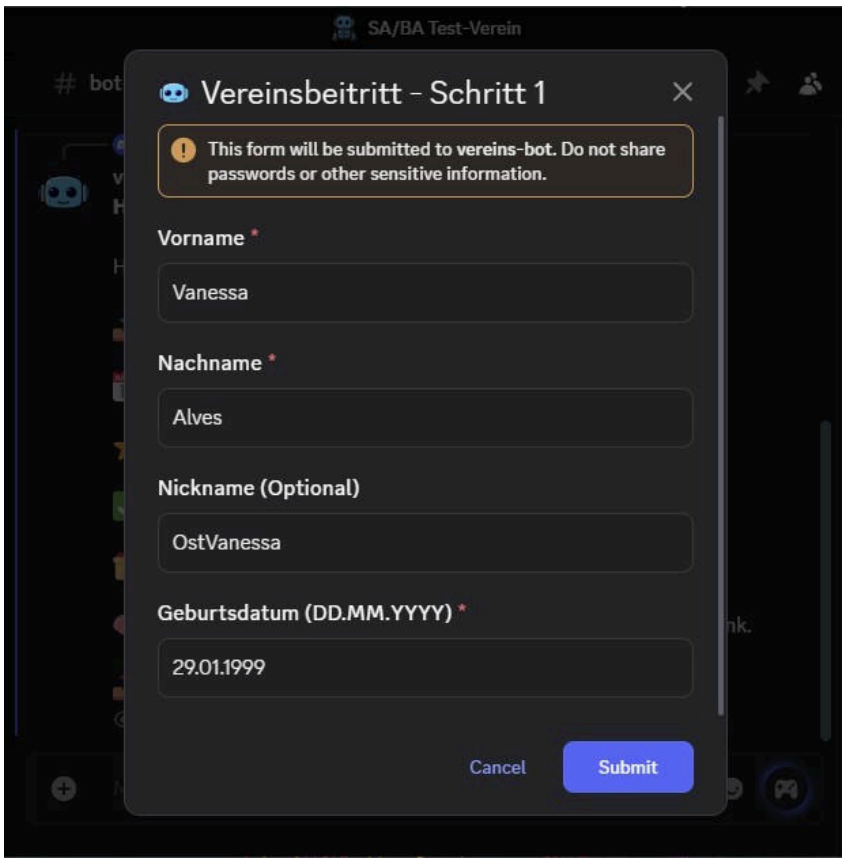


Abb. 56: Start des Beitrittsprozesses mit dem `/join`-Befehl

Abb. 57: Dateneingabe und Akzeptanz der Datenschutzrichtlinien

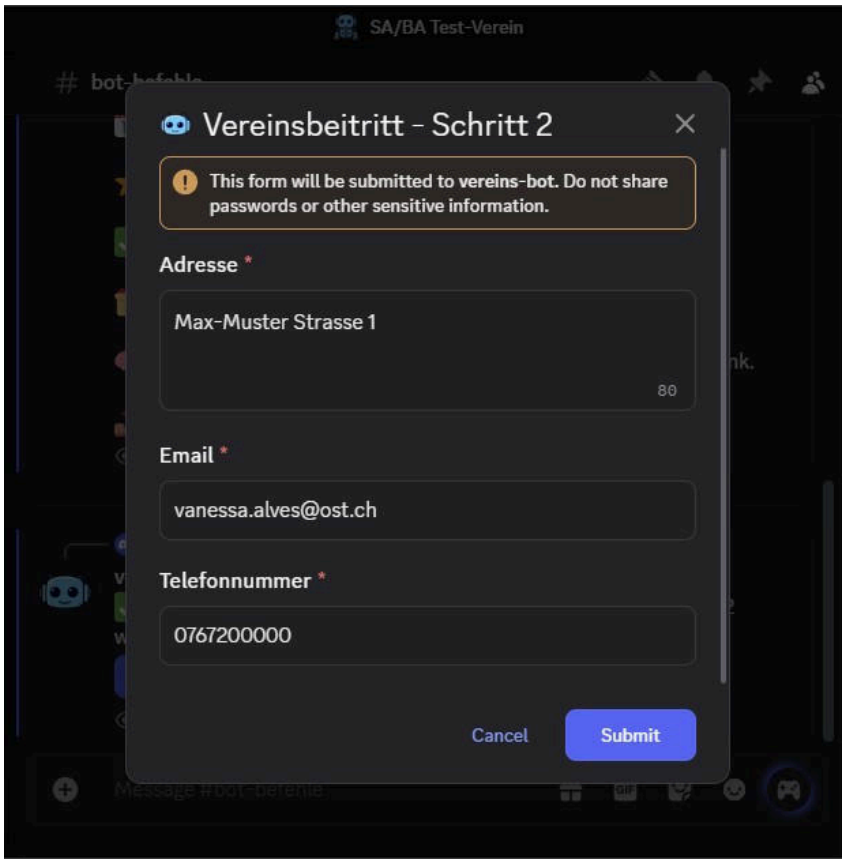


Abb. 58: Automatische private Nachricht mit Zahlungsinformationen

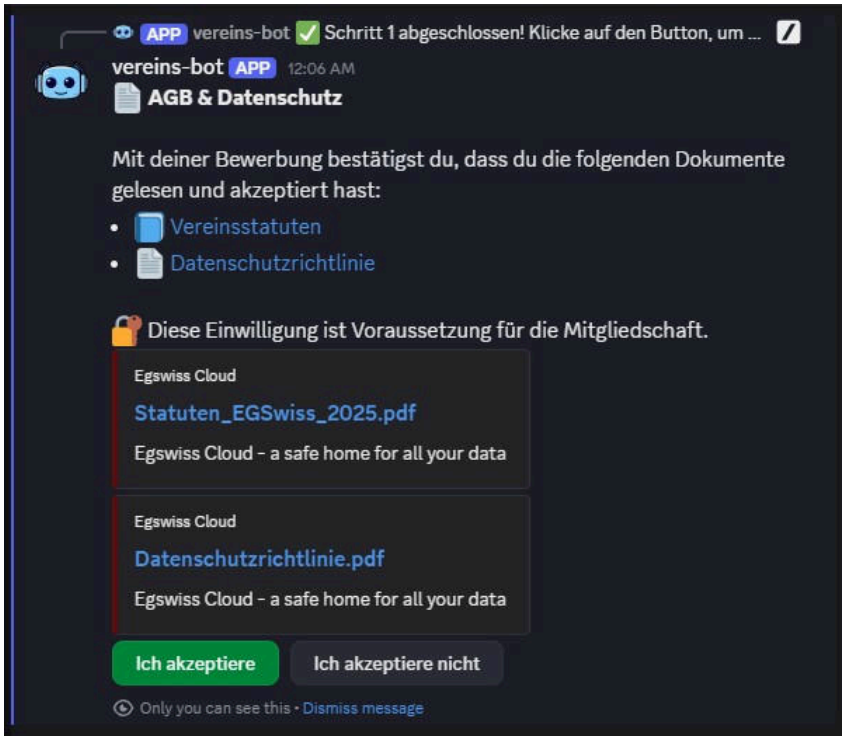


Abb. 59: Benachrichtigung des Vorstands im „bot-bewerbungen“-Kanal

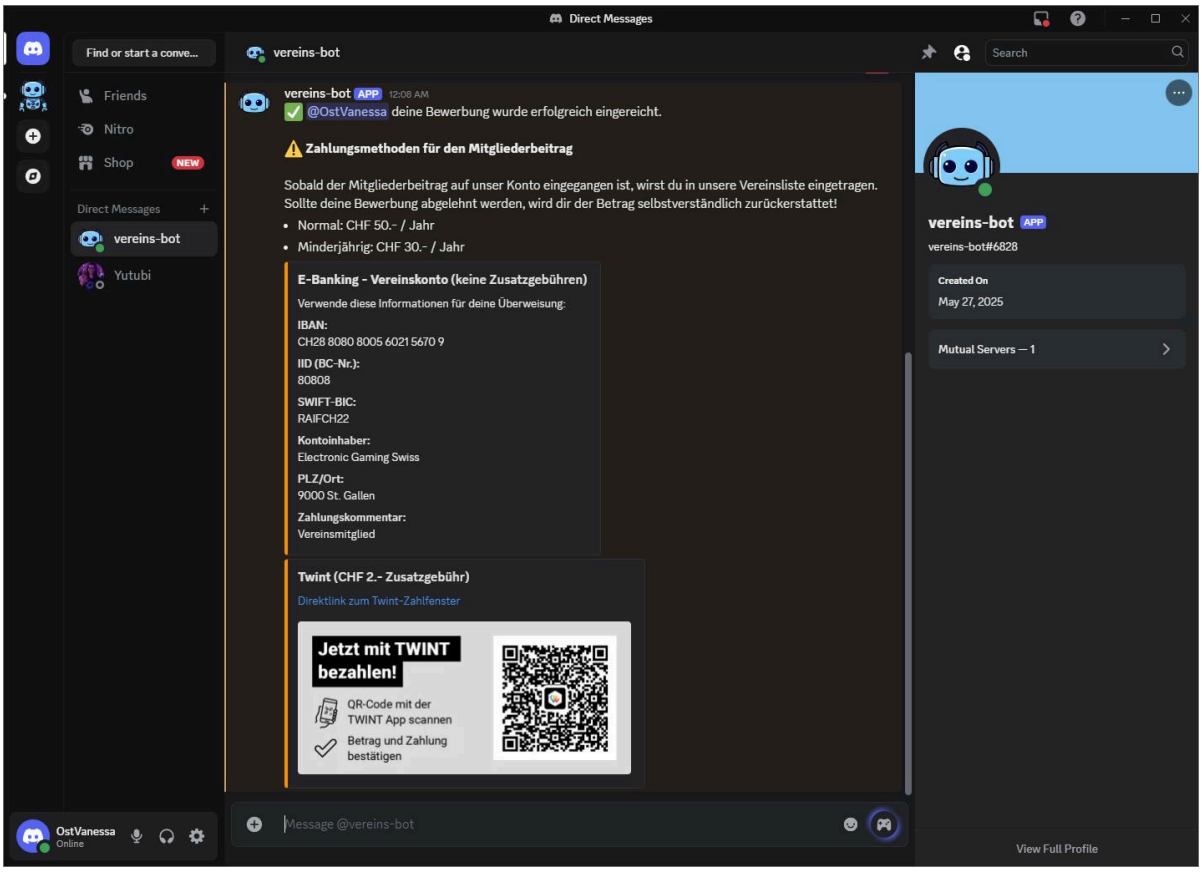


Abb. 60: Entscheidung über die Aufnahme durch die Vorstandsmitglieder

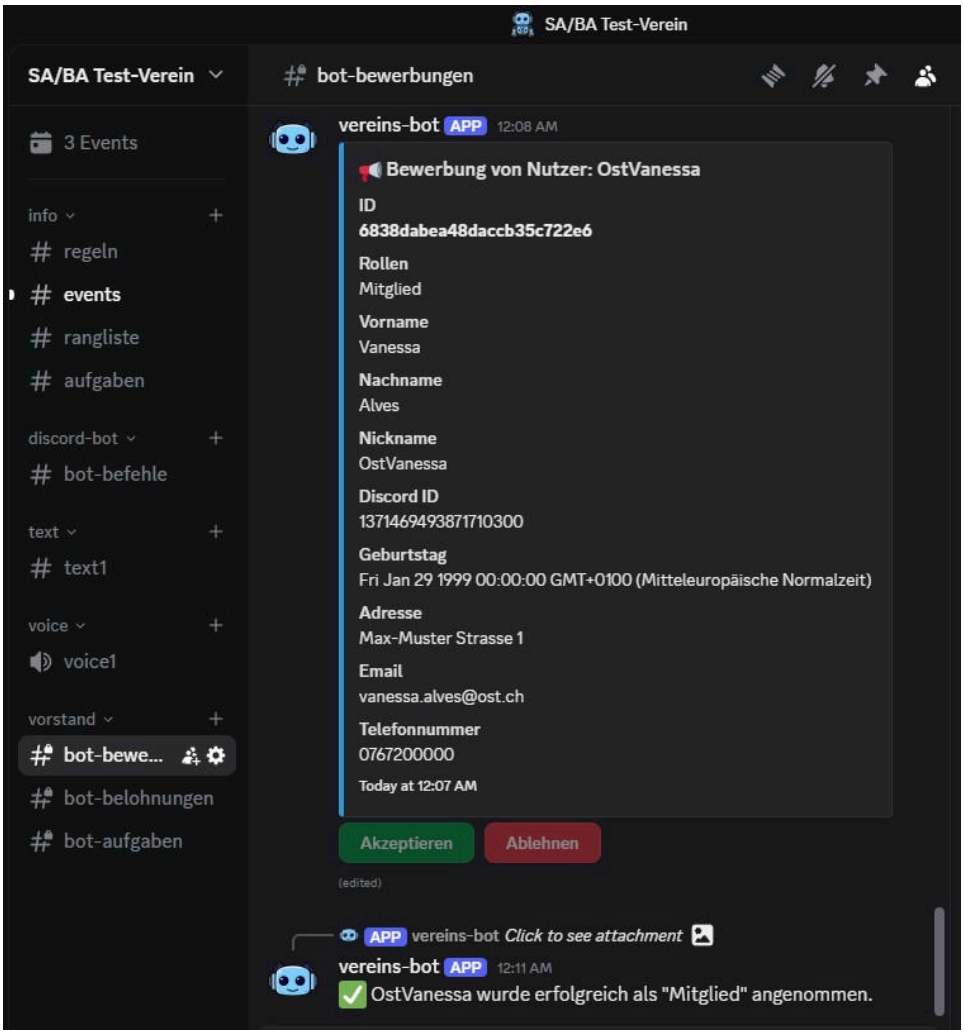


Abb. 61: Private Mitteilung des Aufnahmeentscheids an den Antragsteller

Abb. 62: Bestätigung der neuen Mitgliederrechte nach Annahme

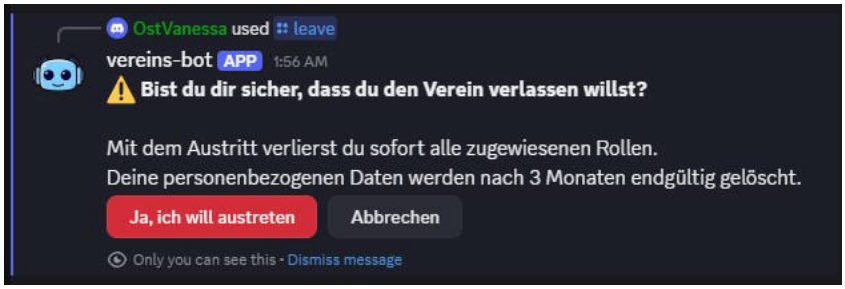


Abb. 63: Anzeige des Austrittsprozesses via `/leave`

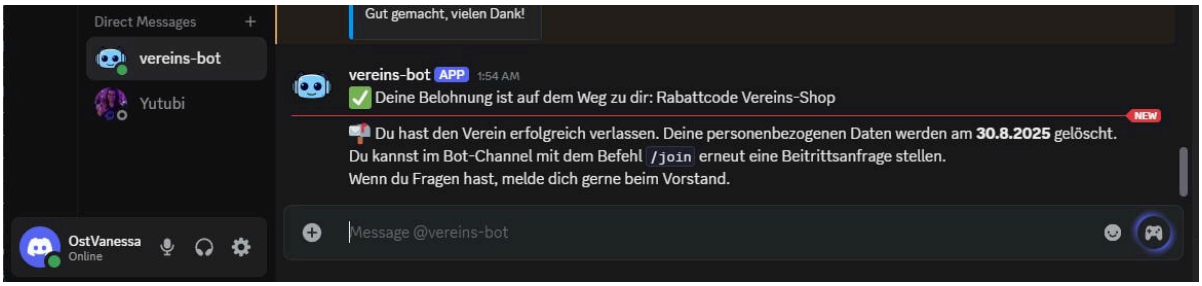


Abb. 64: Bestätigung der erfolgreichen Datenlöschung nach Austritt

15.5 Kalender und Events

Die Google-Kalender-Integration ermöglicht eine automatisierte Übertragung von Vereins-Events in den Discord-Server. Mitglieder werden über anstehende Veranstaltungen informiert und können ihre Teilnahme direkt über Discord bestätigen.

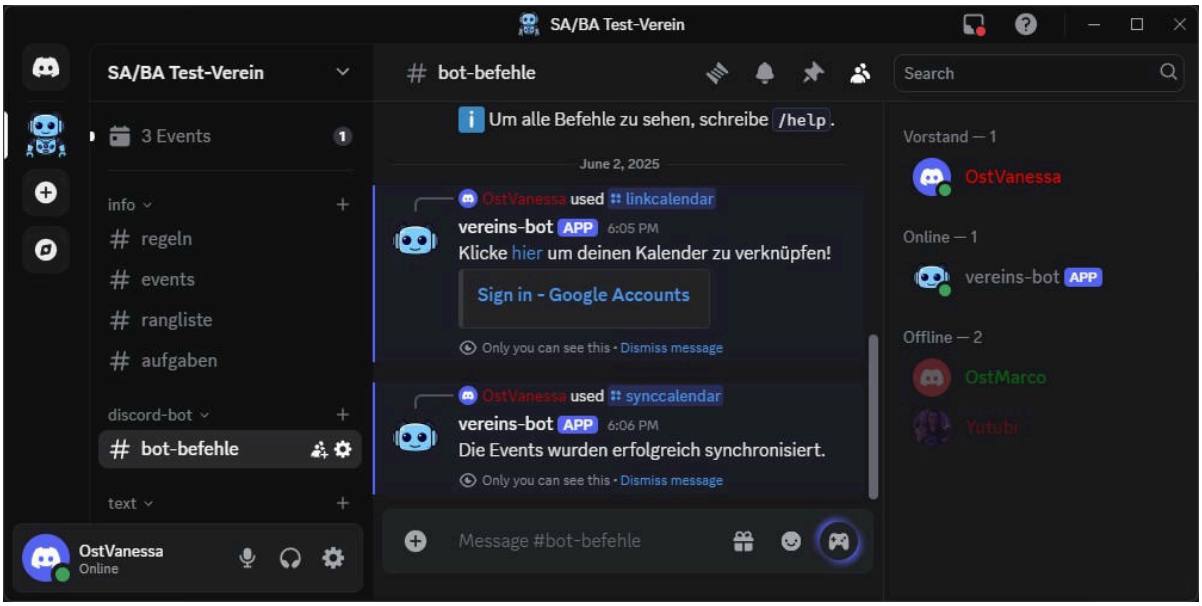


Abb. 65: Anbindung des Google-Kalenders an den Discord-Server

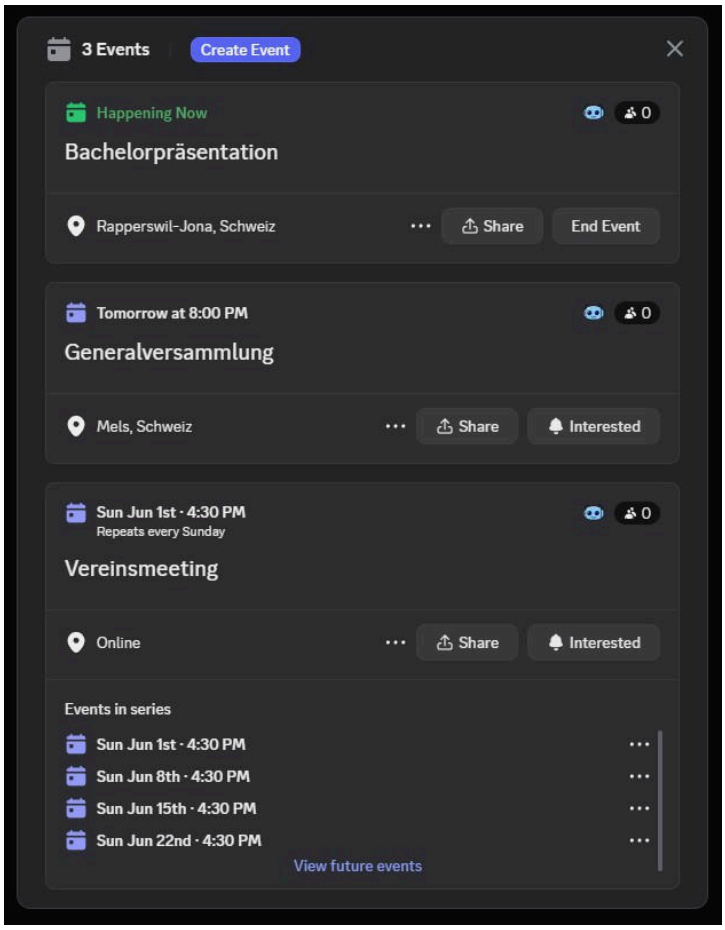


Abb. 66: Übersicht der anstehenden Events im Discord-Server

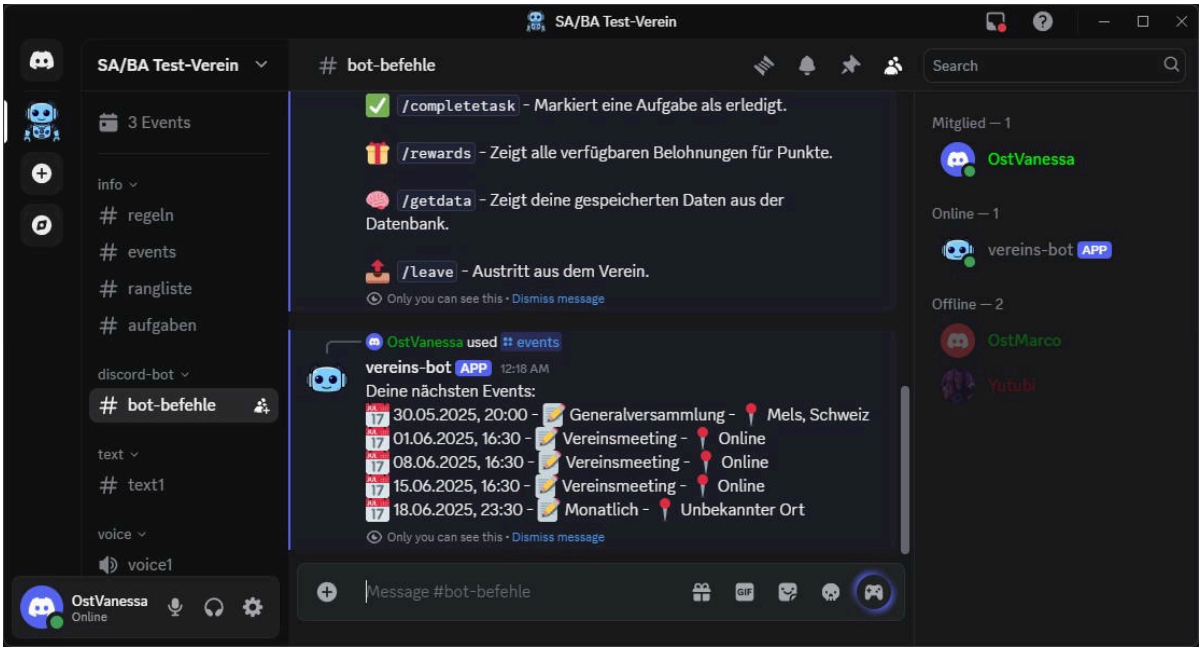


Abb. 67: Detailansicht einer Eventbenachrichtigung mit Teilnahmemöglichkeit

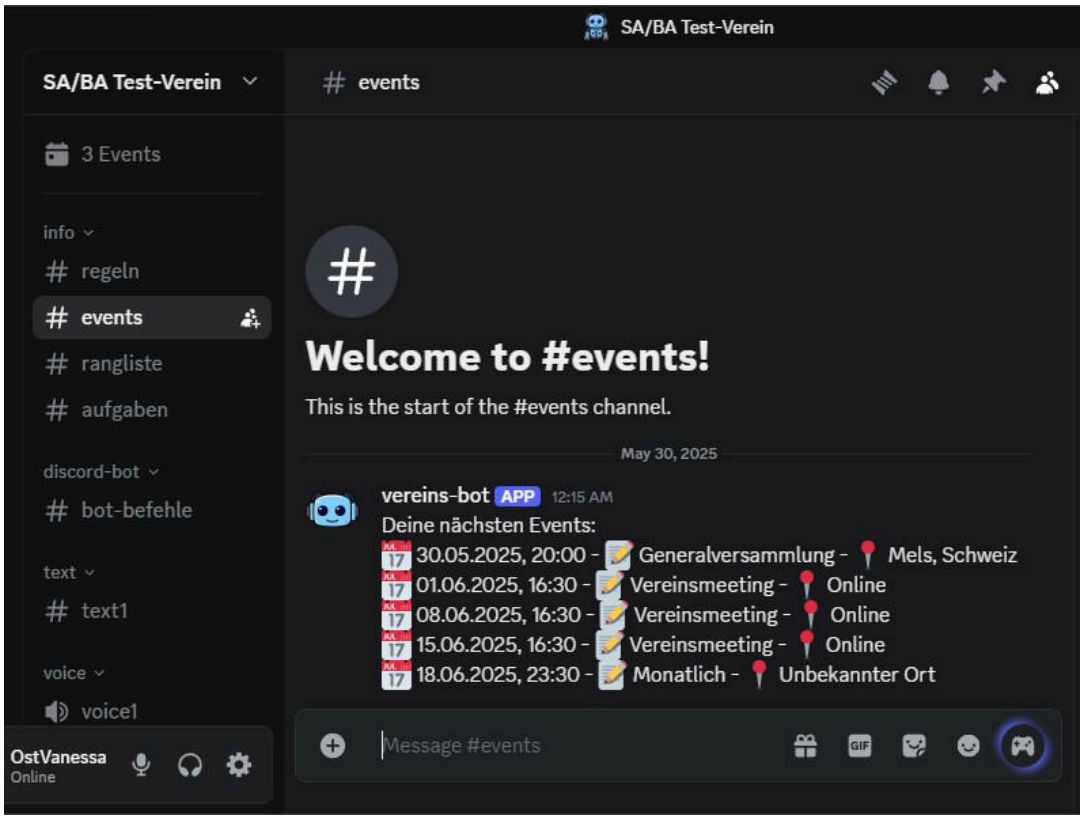


Abb. 68: Automatische Eventübersicht im „events“-Kanal

15.6 Aufgaben und Gamification

Das Gamification-System umfasst Aufgabenverwaltung, Spendenverwaltung und ein Punktesystem. Aufgaben können via `/createtask` erstellt werden, Mitglieder können diese beanspruchen, abschliessen und vom Vorstand verifizieren lassen. Über `/donation` können Spenden erfasst und mit Punkten belohnt werden. Mit `/getdata` werden alle personenbezogenen Daten angezeigt, `/score` liefert die Punktzahl und `/rewards` ermöglicht das Einlösen von Punkten für Belohnungen.

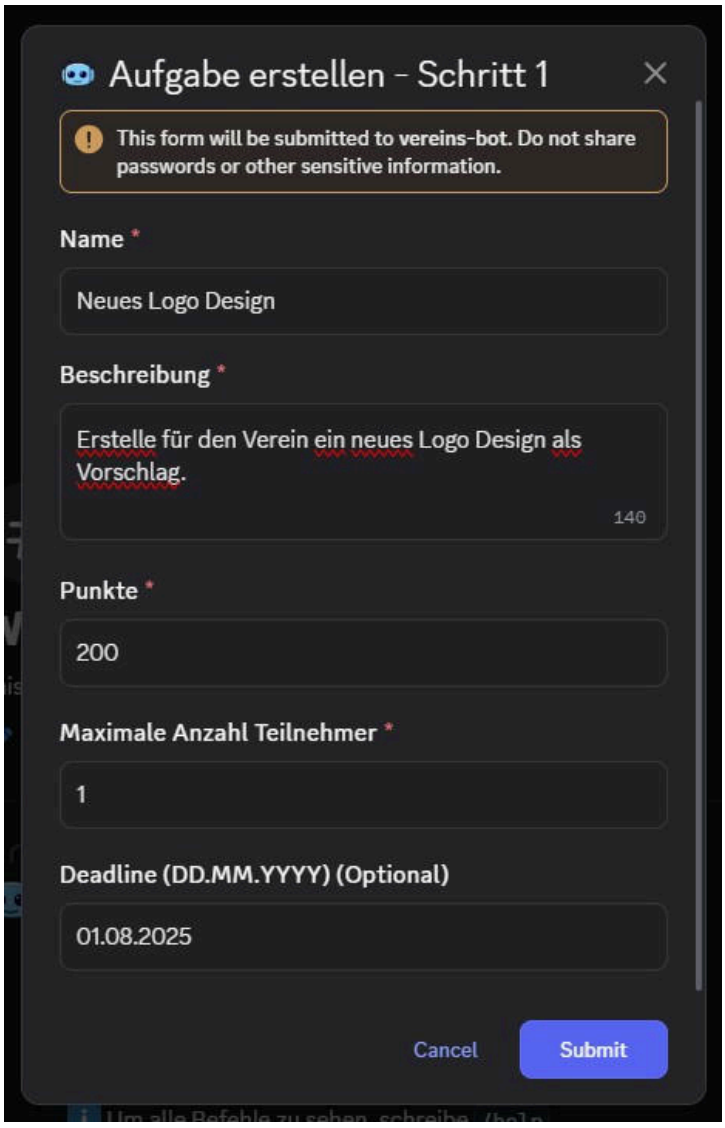


Abb. 69: Erstellung einer Aufgabe durch den Vorstand

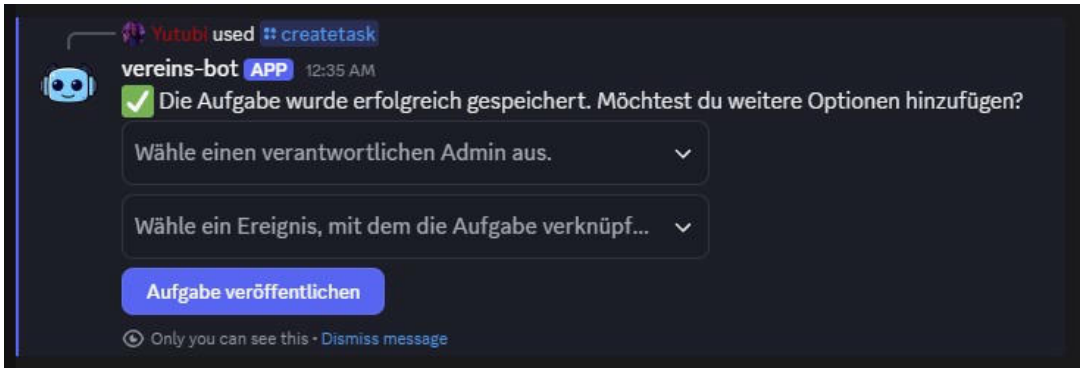


Abb. 70: Anzeige der Aufgabe im „aufgaben“-Kanal

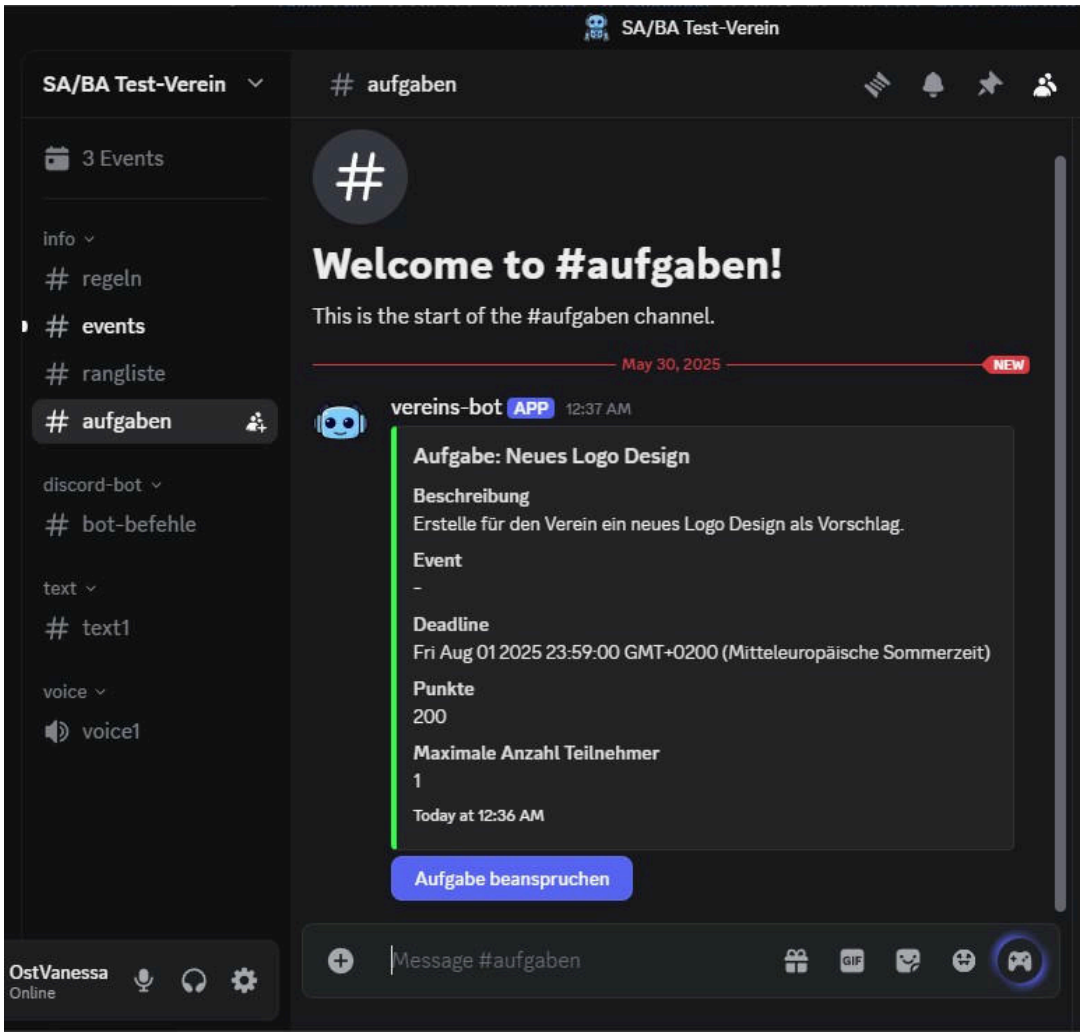


Abb. 71: Beantragung einer Aufgabe durch ein Mitglied

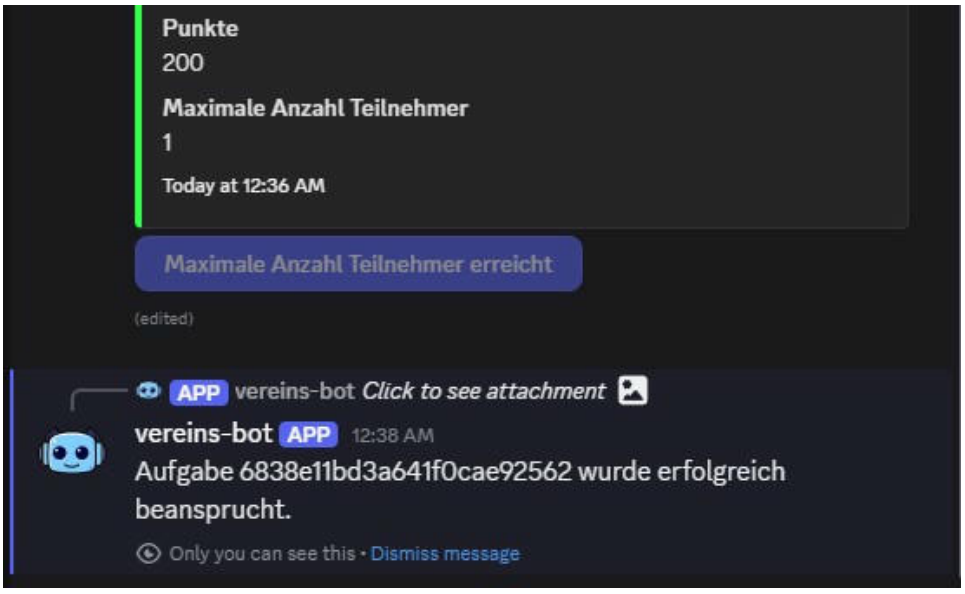


Abb. 72: Abschluss der Aufgabe durch das Mitglied

Abb. 73: Benachrichtigung an den Vorstand zur Prüfung der Aufgabe

Abb. 74: Feedback-Funktion nach Abschluss der Aufgabe

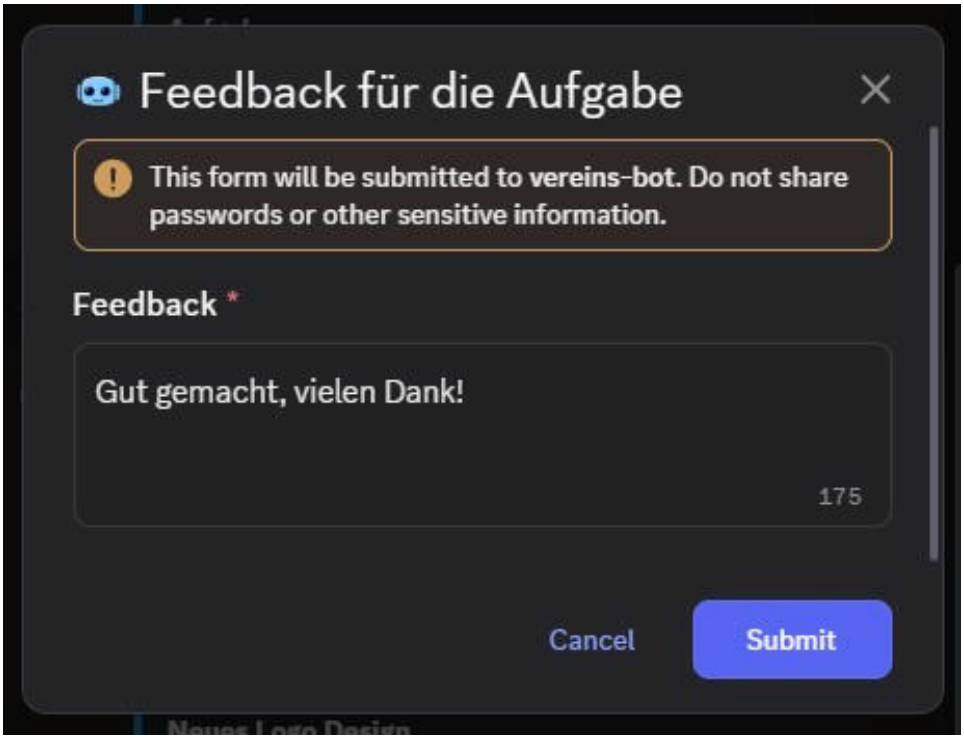


Abb. 75: Anzeige der Bestätigung oder Ablehnung einer Aufgabe durch den Vorstand

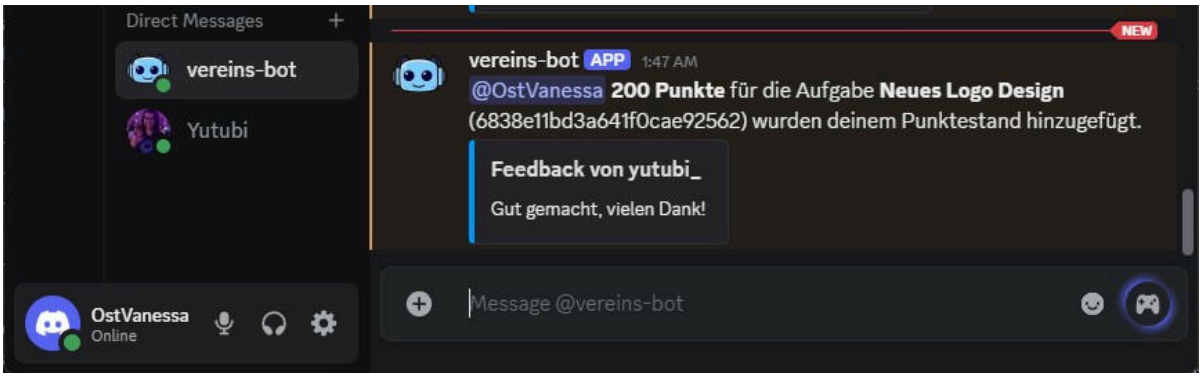


Abb. 76: Benachrichtigung an das Mitglied über die Entscheidung des Vorstands

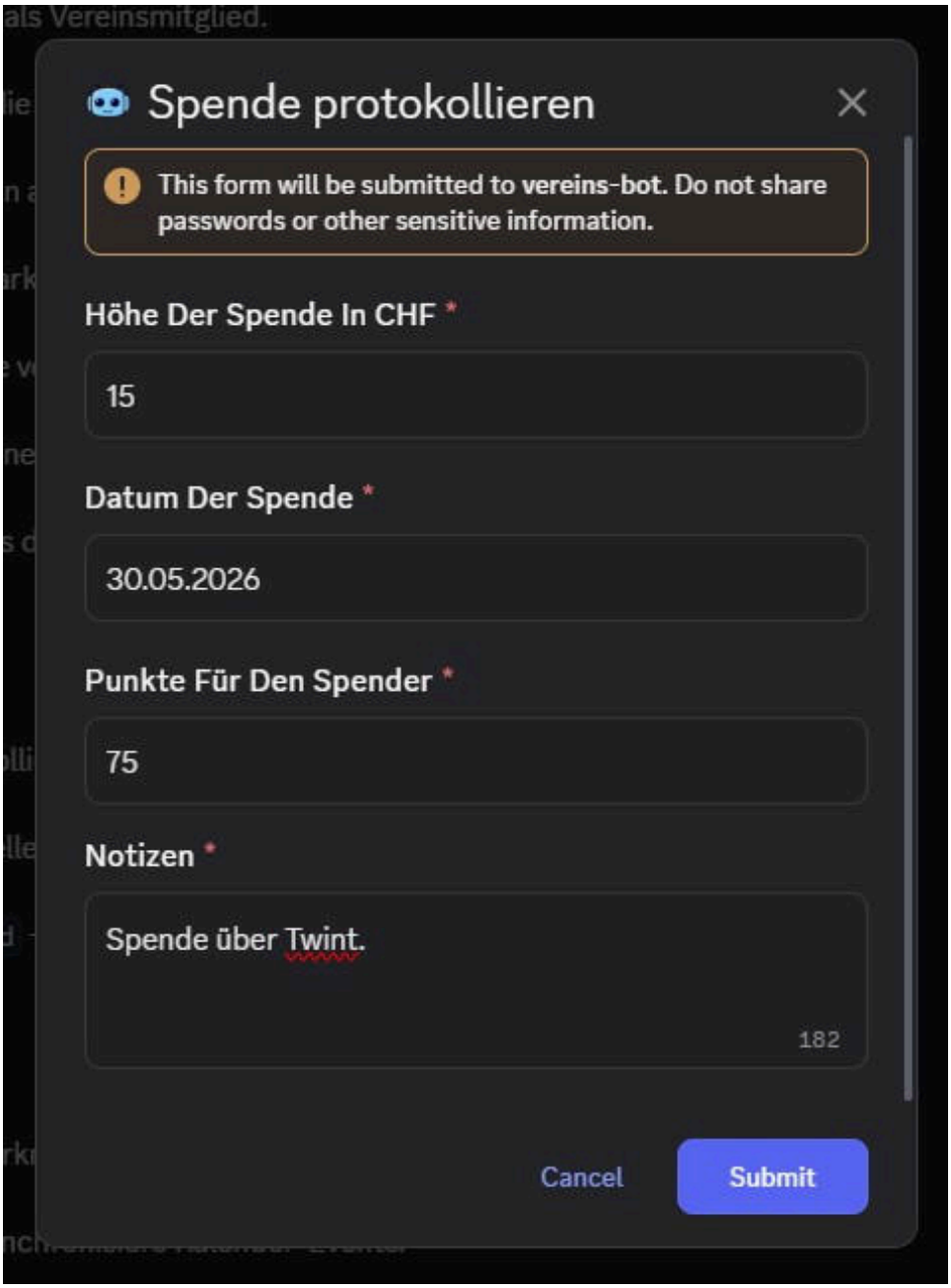


Abb. 77: Erfassung einer Spende und Punktevergabe durch den Vorstand

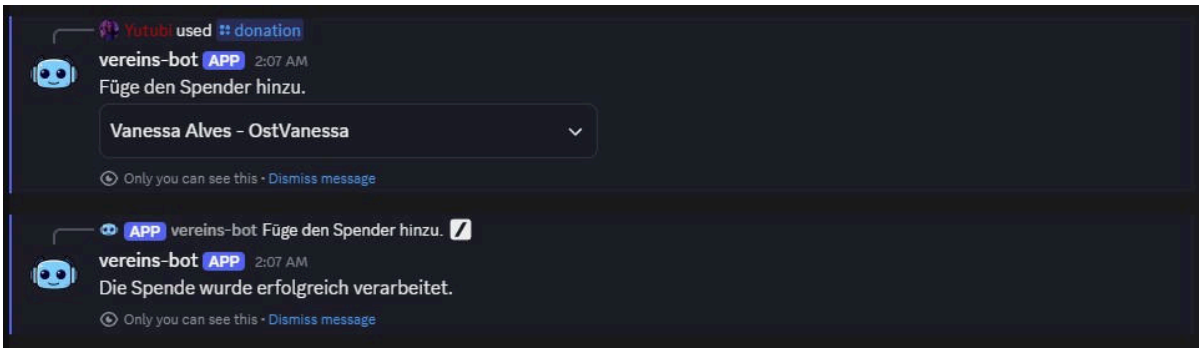


Abb. 78: Übersicht der Spendenhistorie im Bot

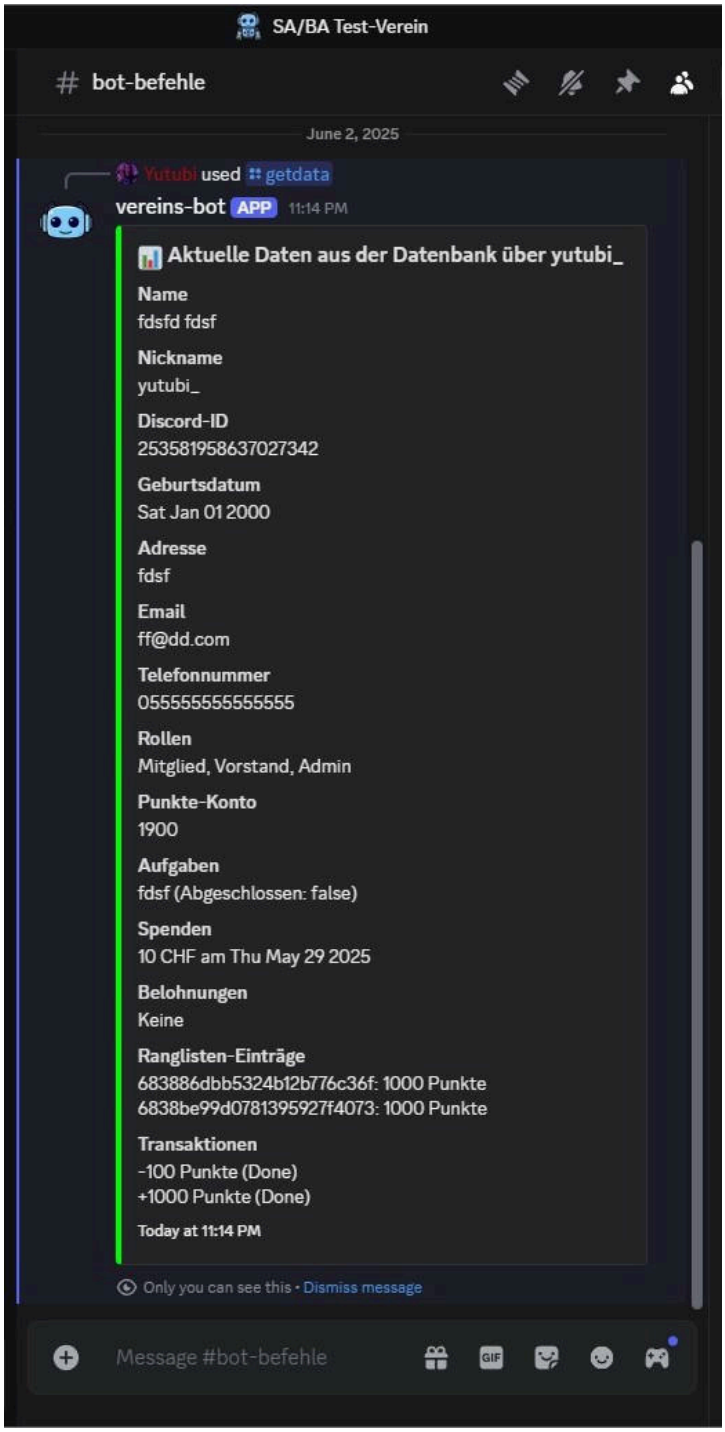


Abb. 79: Abruf aller gespeicherten Daten durch das Mitglied via `/getdata`

Abb. 80: Anzeige der aktuellen Punktzahl des Mitglieds

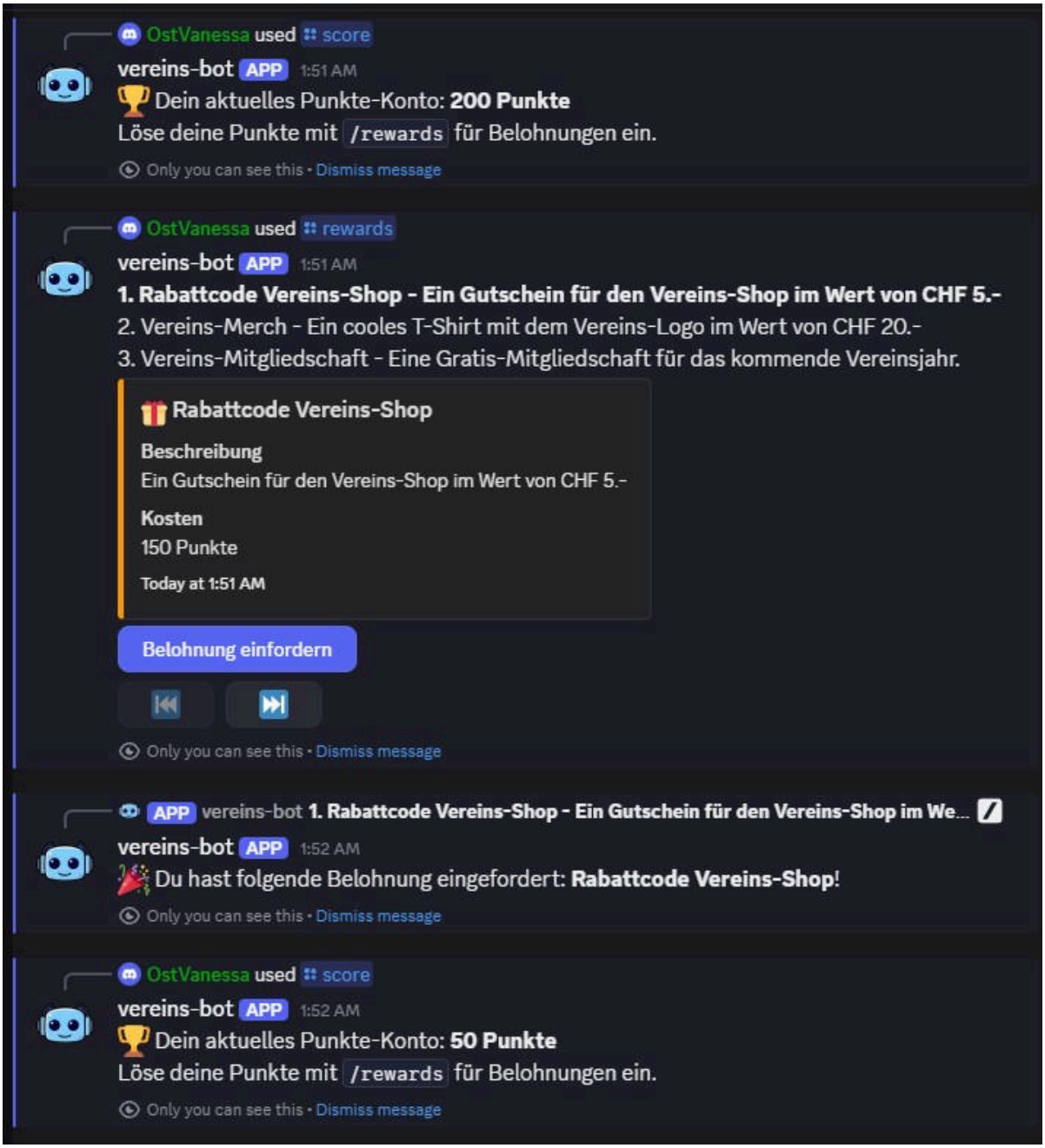


Abb. 81: Übersicht über verfügbare Belohnungen im Bot

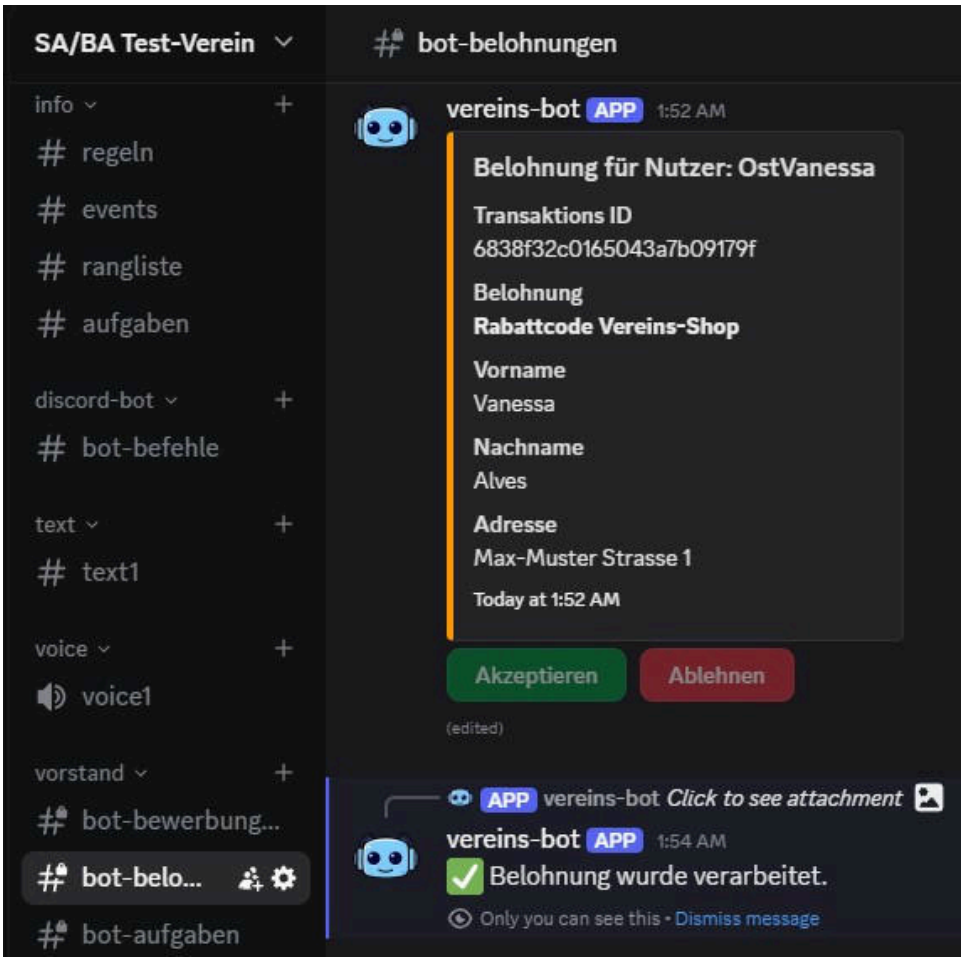


Abb. 82: Bestätigung der Belohnungsausgabe durch den Vorstand