OST
Ostschweizer
Fachhochschule

ClansCore

Weiterentwicklung des Discord-Bots für Vereine

Bachelor- und Studienarbeit im Herbst-Semester 2025

Ausdruck:

19.12.2025

Team / Autoren:

Vanessa Alves und Joel Thoma

Referent:

Prof. Dr. Frieder Loch

Koreferent:

Dr. Julia Czerniak-Wilmes

Gegenleser:

Severin Dellsperger

Abstract

Diese Arbeit beschreibt die konzeptionelle und technische Weiterentwicklung eines im Vorprojekt entstandenen Discord-Bots zu einer modularen und plattformunabhängigen Vereinsverwaltungs-Applikation mit dem Namen ClansCore. Während das Vorprojekt primär einen funktionsorientierten Discord-Bot zur Mitgliederverwaltung, Eventorganisation und zu ersten Gamification-Funktionen umfasste, liegt der Fokus dieser Arbeit auf der architektonischen und funktionalen Erweiterung sowie auf der klaren Abgrenzung der Systemverantwortlichkeiten. Ergänzend wird ein wissenschaftlich fundierter Ansatz zur Weiterentwicklung von Gamification- und Motivationsmechanismen verfolgt.

Zu Beginn wurden qualitative Interviews durchgeführt, um Anforderungen, Unterstützungspotenziale und motivationsfördernde Faktoren im Vereinskontext zu identifizieren. Aufbauend auf diesen Erkenntnissen sowie auf wissenschaftlicher Literatur wurden Konzepte zur Erweiterung von Gamification und Motivation erarbeitet. Diese Konzepte wurden im Rahmen der vorliegenden Arbeit bewusst nicht implementiert und dienen als Grundlage für zukünftige Weiterentwicklungen der Plattform.

Darauf aufbauend wurde der bestehende Discord-Bot zur modularen Vereinsplattform ClansCore weiterentwickelt. Das System wurde in drei zentrale Komponenten aufgeteilt: eine zentrale API als Anwendungskern, einen Discord-Bot als Interaktionsschnittstelle sowie ein webbasiertes Admin-Dashboard. Die API wurde mit TypeScript und Express.js umgesetzt, der Discord-Bot basiert auf Discord.js und das Admin-Dashboard wurde mit Angular realisiert. Die persistente Datenhaltung erfolgt mittels MongoDB. Mit Ausnahme des Dashboards wurden die eingesetzten Technologien aus dem Vorprojekt übernommen. Diese Architektur ermöglicht eine saubere Entkopplung von Geschäftslogik, Datenhaltung und Benutzerinteraktion und schafft die Grundlage für eine plattformunabhängige Weiterentwicklung über Discord hinaus.

Bestehende Funktionen des Vorprojekts, wie die Mitglieder- und Rollenverwaltung sowie die Synchronisation von Events mit Google Calendar, wurden integriert und erweitert. Der Discord-Bot unterstützt unter anderem die Mitgliedsbewerbung, Aufgaben- und Punktelogik, Ranglisten, Spenden sowie eine bidirektionale Event-Synchronisation inklusive automatisierter Erinnerungen. Das Admin-Dashboard ergänzt diese Funktionen durch eine zentrale administrative Sicht auf vereinsrelevante Daten und ermöglicht insbesondere die Konfiguration eines Punktesystems mit automatischen Punktevorschlägen.

Die Evaluation der entwickelten Plattform erfolgt anhand einer realitätsnahen Simulation des Vereinsalltags, in welcher typische Prozesse wie Mitgliederverwaltung, Eventteilnahmen, Aufgabenbearbeitung und Punktevergabe nachgestellt werden. Die Ergebnisse dieser Simulation werden nach Abgabe der vorliegenden Dokumentation erhoben und als Anhang ergänzt.

Insgesamt zeigt diese Arbeit, wie durch gezielte architektonische und konzeptionelle Weiterentwicklung aus einem bestehenden Discord-Bot eine erweiterbare Vereinsplattform entsteht, die administrative Transparenz schafft und eine fundierte Grundlage für zukünftige funktionale und motivationsbezogene Erweiterungen bietet.

Management Summary

Ausgangslage

Online-Vereine nutzen Plattformen wie Discord als zentrale Kommunikations- und Interaktionslösung. Neben der reinen Kommunikation spielen dabei auch Vereinsprozesse wie Mitgliederaufnahme, Eventankündigungen und die Einbindung der Mitglieder in das Vereinsleben eine zunehmend wichtige Rolle. Um diese Abläufe direkt innerhalb von Discord zu unterstützen, wurde im Vorprojekt ein Discord-Bot entwickelt, der vereinsrelevante Funktionen wie Mitgliederverwaltung, Eventorganisation und erste Gamification-Elemente integrierte.

Der Bot erfüllte das Ziel, Vereinsprozesse unmittelbar in der bestehenden Kommunikationsplattform abzubilden und so Medienbrüche zu vermeiden. Mit wachsendem Funktionsumfang zeigte sich jedoch, dass die monolithische Struktur und die enge Kopplung an Discord die Wartbarkeit, Erweiterbarkeit und Wiederverwendbarkeit der Lösung einschränkten. Zudem fehlte eine zentrale administrative Sicht ausserhalb von Discord sowie eine klare Trennung zwischen Geschäftslogik, Datenhaltung und Benutzerinteraktion.

Ziel dieser Arbeit war es daher, den bestehenden Discord-Bot konzeptionell und technisch weiterzuentwickeln und in eine strukturierte, erweiterbare und plattformunabhängige Vereinsverwaltungs-Applikation zu realisieren. Dabei sollten die bestehenden Stärken des Vorprojekts erhalten bleiben, gleichzeitig jedoch die Grundlage für zukünftige funktionale und motivationsbezogene Erweiterungen geschaffen werden.

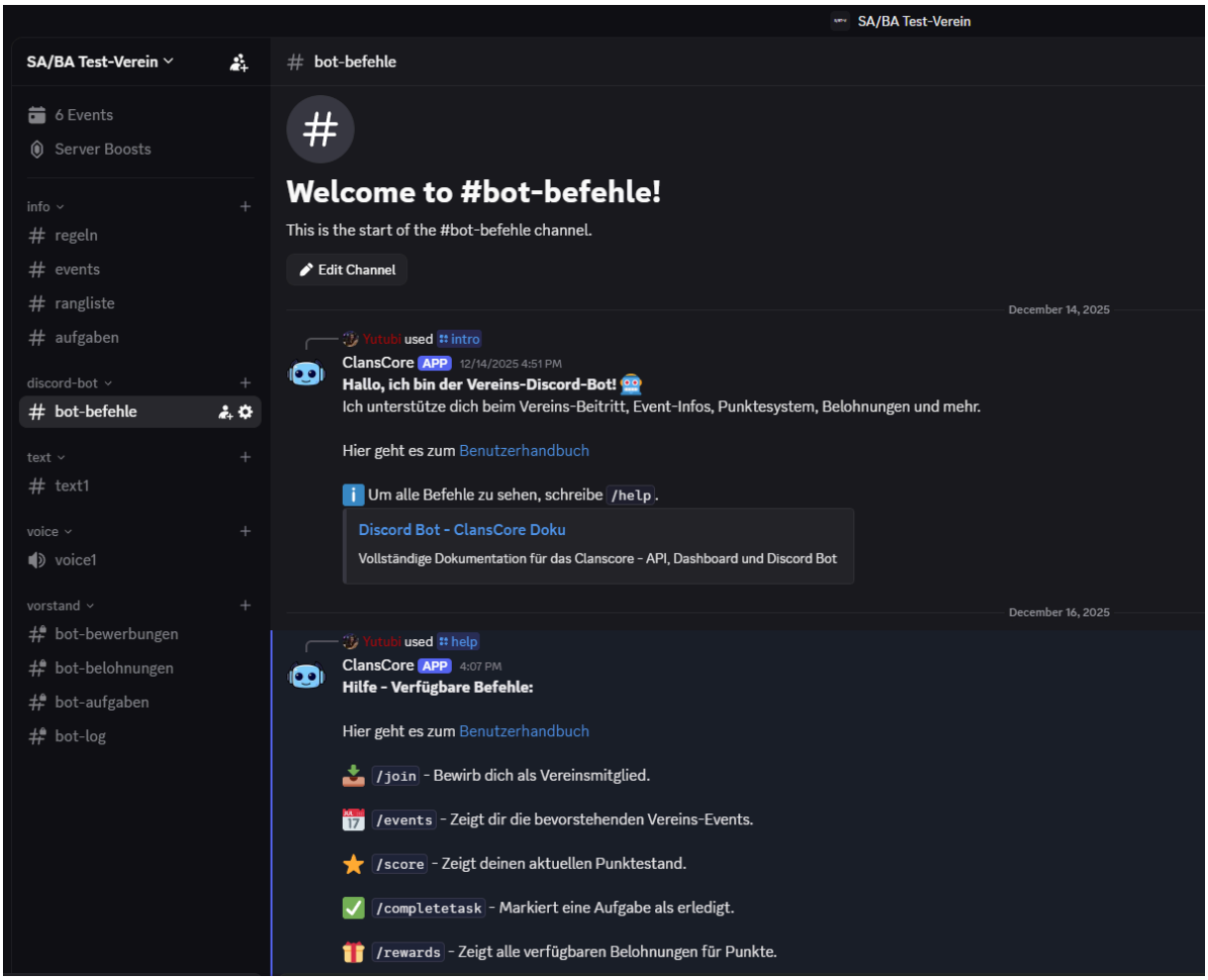


Abb. 1: Der Discord-Bot im Einsatz des Vereins-Server mit den Befehlen /intro und /help

Vorgehen und Technologien

Zu Beginn der Arbeit wurden qualitative Interviews mit Vereinspräsidenten verschiedener Schweizer Gaming-Vereine durchgeführt, um Anforderungen, Unterstützungsbedarfe und motivationsfördernde Faktoren im Vereinsalltag zu identifizieren. Auf dieser Basis sowie unter Einbezug wissenschaftlicher Literatur wurden Konzepte zur Erweiterung von Gamification- und Motivationsmechanismen erarbeitet. Diese Konzepte wurden bewusst nicht vollständig umgesetzt, sondern dienen als Grundlage für zukünftige Weiterentwicklungen.

Darauf aufbauend wurde der bestehende Discord-Bot zur modularen Vereinsverwaltungs-Applikation ClansCore weiterentwickelt. Das System wurde in drei getrennte Komponenten aufgeteilt:

- Eine zentrale API als Anwendungskern
- Einen Discord-Bot als primäre Interaktionsschnittstelle
- Ein webbasiertes Admin-Dashboard für administrative Aufgaben

Mit Ausnahme des Dashboards wurden die eingesetzten Technologien aus dem Vorprojekt übernommen. Durch diese Architektur konnte eine klare Trennung von Geschäftslogik, Datenhaltung und Benutzerinteraktion erreicht werden, wodurch die Plattform unabhängig von Discord weiterentwickelt werden kann.

Zur Bewertung der Lösung wurde eine realitätsnahe Simulation des Vereinsalltags konzipiert, welche typische Abläufe wie Mitgliederverwaltung, Eventteilnahmen, Aufgabenbearbeitung und Punktevergabe abbildet.

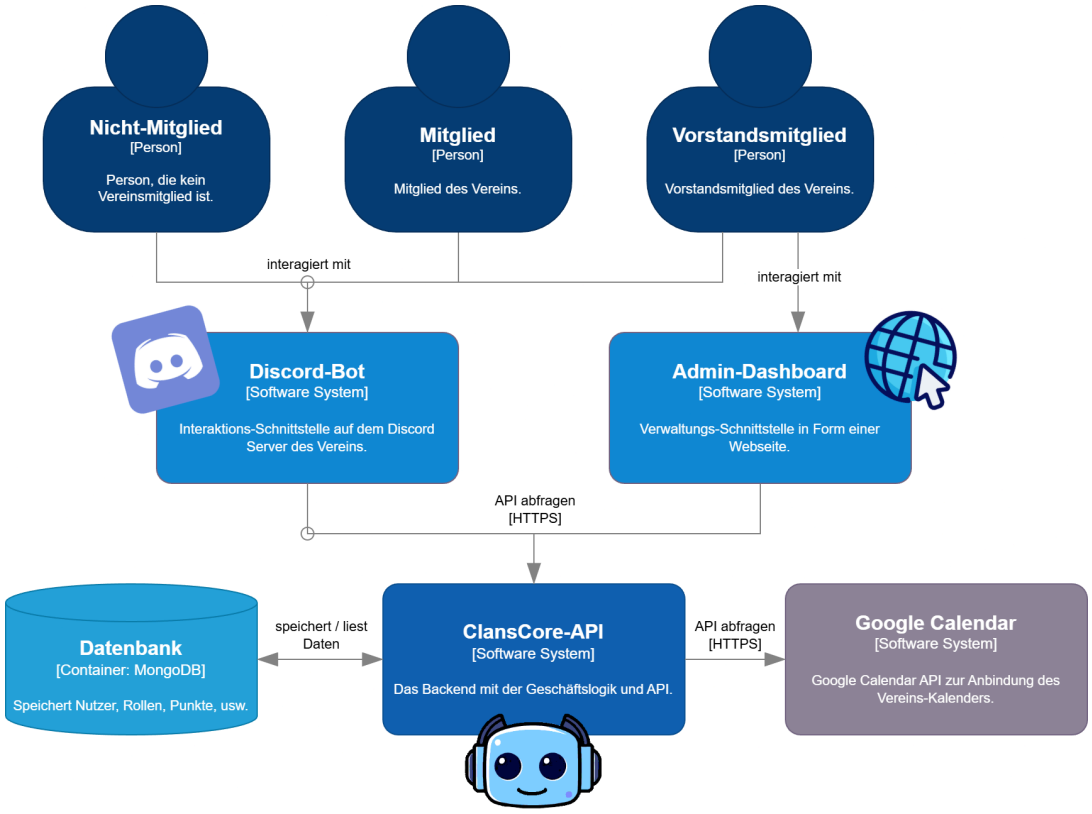


Abb. 2: Architektur-Übersicht des Gesamt-Systems Clanscore sowie der Interaktions-Schnittstellen.

Ergebnisse

Mit ClansCore entsteht eine funktionsfähige und erweiterbare Vereinsplattform, welche zentrale Vereinsprozesse automatisiert und transparenter gestaltet. Der Discord-Bot unterstützt unter anderem die Online-Mitgliedsbewerbung, die Verwaltung von Aufgaben und Punkten, Ranglisten, Spenden sowie die bidirektionale Synchronisation von Events mit Google Calendar inklusive automatisierter Erinnerungen.

Das Admin-Dashboard ergänzt den Bot um eine zentrale administrative Sicht auf vereinsrelevante Daten. Es ermöglicht die Verwaltung von Mitgliedern und Rollen sowie die Einsicht in Aufgaben, Belohnungen, Transaktionen und Ranglisten. Zudem kann ein Punktesystem zentral konfiguriert werden, um bei der Erstellung von Aufgaben durch automatische Punktevorschläge zu unterstützen. Dadurch können manuelle Hilfsmittel wie Tabellenkalkulationen ersetzt und der administrative Aufwand reduziert werden.

Die Evaluation der Plattform erfolgt anhand der entwickelten Simulation des Vereinsalltags. Die Ergebnisse dieser Simulation werden nach Abgabe der Arbeit erhoben und als Anhang ergänzt. Insgesamt zeigt die Arbeit, dass durch gezielte architektonische und konzeptionelle Weiterentwicklung aus einem bestehenden Discord-Bot eine strukturierte und plattformunabhängige Vereinsverwaltungs-Applikation entsteht, die sowohl administrative Transparenz schafft als auch eine fundierte Grundlage für zukünftige funktionale und motivationsbezogene Erweiterungen bietet.

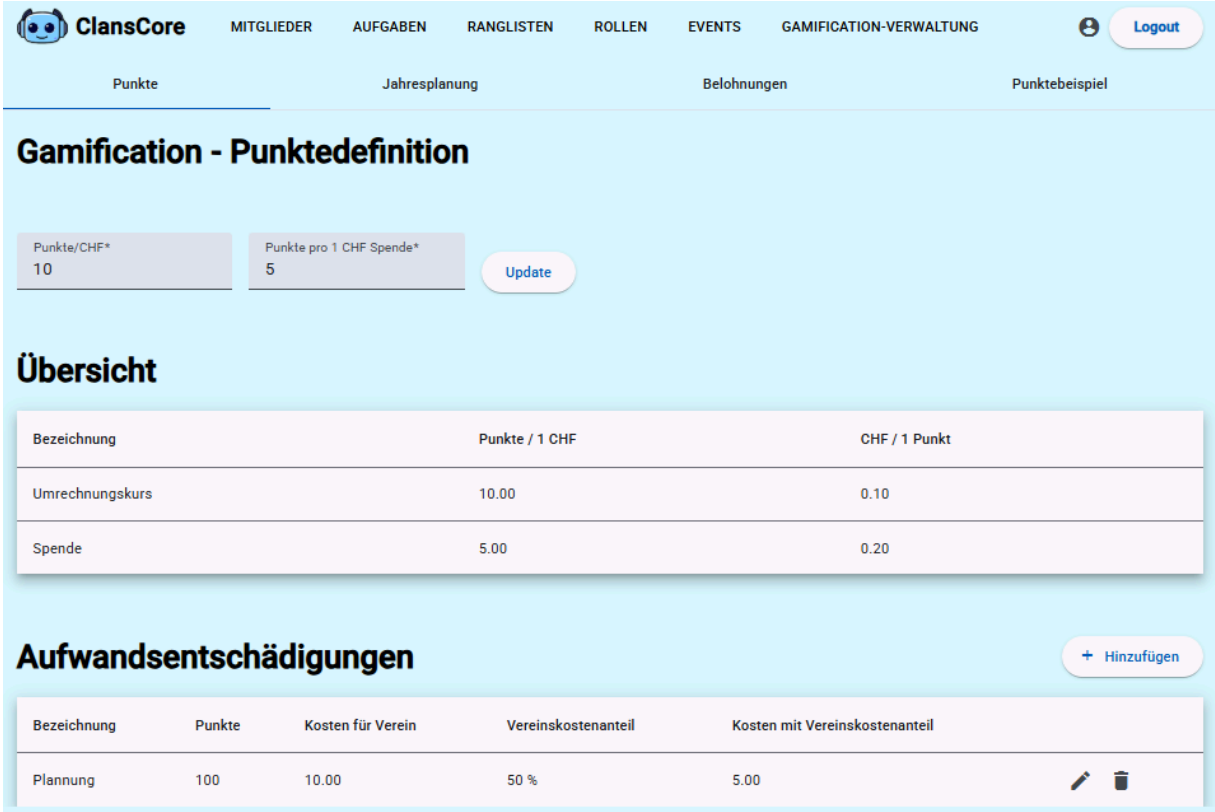


Abb. 3: Die Erstellungs-Seite des Punktesystems über das Admin-Dashboard

Inhaltsverzeichnis

Kapitel I
Produkt Dokumentation

1 Einleitung

Digitale Kommunikationsplattformen wie Discord haben sich in den vergangenen Jahren zunehmend als zentrale Werkzeuge für die Organisation und Interaktion innerhalb von Online-Communities und Vereinen etabliert. Insbesondere im Bereich der Freizeit- und Gaming-Vereine bieten sie die Möglichkeit, Kommunikation, Eventplanung und Mitgliederverwaltung zu bündeln. Trotz dieser Vorteile stossen bestehende Plattformfunktionen bei wachsender Mitgliederzahl, zunehmender organisatorischer Komplexität und steigenden Anforderungen an Datensicherheit und Automatisierung an ihre Grenzen.

Vor diesem Hintergrund befasst sich die vorliegende Bachelorarbeit mit der Weiterentwicklung eines im Vorprojekt erstellten Discord-Bots, der vereinsrelevante Prozesse automatisiert und Gamification-Mechanismen zur Steigerung des Engagements nutzt. Ziel der Arbeit ist es, das bestehende System, um ein plattformunabhängiges Admin-Dashboard zu erweitern, welches die sichere, benutzerfreundliche und rollenbasierte Verwaltung von Mitgliederdaten und Gamification-Statistiken ermöglicht. Damit wird ein Beitrag zur Entkopplung der Vereinsverwaltung von der Kommunikationsplattform Discord geleistet und die Grundlage für eine modulare, erweiterbare Systemarchitektur geschaffen.

1.1 Problemstellung

Das im Vorprojekt entwickelte System ermöglicht die Automatisierung einiger zentraler Vereinsprozesse, ist jedoch ausschliesslich innerhalb der Discord-Umgebung funktionsfähig. Dadurch ist die Nutzung stark Discord-zentriert und für Personen ohne Discord-Konto oder mit eingeschränkter technischer Erfahrung nur begrenzt zugänglich.

Die Bearbeitung vereinsrelevanter Daten erfolgt derzeit ausschliesslich über direkten Zugriff auf die MongoDB-Datenbank. Dies erfordert spezifische technische Kenntnisse, birgt Sicherheitsrisiken und verhindert eine kontrollierte, benutzerfreundliche Datenverwaltung. Zudem existiert keine zentrale Oberfläche, die eine visuelle und statistisch fundierte Auswertung des Vereinsengagements ermöglicht.

Darüber hinaus erschwert die aktuelle Architektur die Anbindung zusätzlicher Plattformen oder externer Systeme, etwa zur Erweiterung um Web- oder Mobile-Anwendungen. Diese Einschränkungen verdeutlichen den Bedarf nach einer modularen, skalierbaren und plattformunabhängigen Systemlösung, die sowohl technische als auch organisatorische Herausforderungen adressiert.

1.2 Vorarbeiten

Das Vorprojekt „Discord-Bot für Vereine“ (Alves & Schnider, 2025) bildete die Grundlage für die vorliegende Arbeit. Es verfolgte das Ziel, zentrale Vereinsprozesse wie Mitgliederverwaltung, Eventorganisation und Gamification in einem Discord-Bot zu integrieren. Der entwickelte Prototyp implementierte ein Rollen- und Berechtigungssystem, eine Anbindung an die Google Calendar API sowie ein auf wissenschaftlichen Erkenntnissen basierendes Gamification-Konzept.

Während die Machbarkeit und der Nutzen eines solchen Systems weitergehend bestätigt werden konnten, blieb dessen Erweiterbarkeit auf andere Plattformen sowie die administrative Steuerung offen. Das Admin-Dashboard wurde im Vorprojekt lediglich als potenzielle Weiterentwicklung beschrieben, jedoch nicht realisiert. Ebenso fehlten Mechanismen zur Messung und Visualisierung des Engagements sowie eine Entkopplung der Discord-spezifischen Logik von der Geschäftslogik.

Die vorliegende Arbeit schliesst an diese Ergebnisse an und adressiert die genannten Defizite durch eine Neugestaltung der Architektur, eine verbesserte Datenkonsistenz zwischen Discord, Dashboard und Datenbank sowie durch die Integration einer administrativen Benutzeroberfläche.

1.3 Ziel der Arbeit und Abgrenzung

Ziel dieser Bachelor- und Studienarbeit ist die Konzeption und Entwicklung einer plattformunabhängigen, administrativen Erweiterung des bestehenden Discord-Bots, die eine effiziente und sichere Verwaltung vereinsrelevanter Daten ermöglicht. Hierbei liegt der Fokus auf der funktionalen Umsetzung eines Admin-Dashboards sowie auf der architektonischen Modularisierung des Systems, um künftige Integrationen weiterer Plattformen leichter zu ermöglichen. Das neue Gesamt-System wird mit dem Namen ClansCore adressiert.

Im Einzelnen verfolgt die Arbeit folgende Teilziele:

- Refaktorisierung der Backend-Architektur zur Entkopplung von Discord-spezifischer Logik und zur Vorbereitung auf eine plattformübergreifende Erweiterung
- Entwurf und Implementierung eines funktionsfähigen Admin-Dashboards in ClansCore, zur Verwaltung von Mitgliedern, Rollen, Events und Gamification-Daten
- Entwicklung einer sicheren Schnittstellenkommunikation zwischen Bot, Dashboard und Datenbank zur Gewährleistung konsistenter Datenzustände innerhalb des gesamten ClansCore-Systems
- Konzeptionelle und technische Erweiterung des Gamification-Systems
- Evaluation von ClansCore durch eine realitätsnahem Simulation des Vereinsalltages

Diese Arbeit unterscheidet sich vom Vorprojekt durch ihren erweiterten technischen und analytischen Anspruch. Während das Vorprojekt primär die Funktionalität innerhalb von Discord validierte, liegt der Schwerpunkt nun auf der systematischen Generalisierung und Professionalisierung der Lösung als Softwarelösung namens ClansCore. Damit wird die Grundlage geschaffen, um das System langfristig als universelles Vereinsmanagement-Tool einsetzen zu können.

1.4 Rahmenbedingungen

Die Arbeit wurde im Rahmen der Bachelor- und Studienarbeit des Studiengangs Informatik an der Ostschweizer Fachhochschule (OST) durchgeführt.

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

Der Schwerpunkt liegt auf der Konzeption, Entwicklung und Evaluation eines funktionalen Software-Systems mit wissenschaftlich begründetem Gamification- und Motivations-Ansatz, welches den Namen ClansCore trägt. Die Arbeit vereint methodische Ansätze aus den Bereichen Software-Engineering und Motivationsforschung und ist als entwicklungsorientierte empirische Arbeit mit prototypischem Charakter einzuordnen.

1.5 Stand der Technik

Der Stand der Technik im Bereich digitaler Vereinsorganisation wurde im Vorprojekt umfassend untersucht und bildet die Grundlage der vorliegenden Arbeit. Die damalige Marktanalyse zeigte, dass bestehende Systeme entweder auf die Interaktion und Gamification innerhalb von Discord oder auf die administrative Vereinsverwaltung über klassische Softwarelösungen fokussieren, jedoch keine integrative Verbindung beider Ansätze bieten.

Seit Abschluss des Vorprojekts im Juli 2025 sind zwar Weiterentwicklungen einzelner Systeme erkennbar, eine Lösung, welche Gamification, Datenverwaltung und plattformübergreifende Nutzung in einem kohärenten Konzept vereint, wurde bisher jedoch nicht öffentlich dokumentiert.

Im Bereich der Discord-Bots zählt MEE6 zu den meistgenutzten Anwendungen und bietet über das Plugin „Statistics Channels“ grundlegende Auswertungen zur Serveraktivität sowie ein Levelsystem zur Förderung der Nutzerinteraktion (MEE6-Contributors, 2025) . Auch der Bot Arcane integriert vergleichbare Gamification-Funktionen in Form von Belohnungssystemen (Arcane-Contributors, 2025) . Beide Systeme bleiben jedoch auf die Discord-Plattform beschränkt und verfügen über keine dokumentierten Schnittstellen zu externen Administrations- oder Datenverwaltungssystemen.

Demgegenüber adressieren klassische Vereinsmanagement-Lösungen wie ClubDesk oder WildApricot die organisatorischen Anforderungen von Vereinen, insbesondere in den Bereichen Mitgliederverwaltung, Eventplanung und Buchhaltung (ClubDesk-Contributors, 2025; WildApricot-Contributors, 2025) . Motivierende oder interaktive Komponenten sind in diesen Systemen kein grosser Bestandteil. Diskussionen über eine mögliche Erweiterung durch Belohnungsmechanismen existieren lediglich in Anwenderforen, was die bislang geringe praktische Umsetzung solcher Konzepte verdeutlicht (WildApricot-Community, 2015) .

Die vorliegende Arbeit positioniert sich zwischen diesen beiden Ansätzen, indem sie den organisatorischen Verwaltungsfokus klassischer Systeme mit der motivierenden Wirkung von Gamification-Elementen in ClansCore kombiniert. Damit leistet die Arbeit einen Beitrag zur Integration von Motivation, Datenmanagement und Vereinsorganisation in einem einheitlichen, plattformunabhängigen Anwendungskonzept ClansCore.

1.6 Einführung in einer realen Vereinsumgebung

Durch die laufende Einführung des vorherigen Systems (Discord-Bot) im Verein EGSwiss über das Teammitglied Vanessa Alves während der Sommermonate 2025, konnten Rückmeldungen des Vorstandes gesammelt und praxisrelevante Anforderungen sowie Verbesserungsvorschläge identifiziert werden. Darunter finden sich die folgenden Anpassungs- und Erweiterungs-Wünsche:

- Vereinfachte Erstellung des Punktesystems für Gamification-Elemente
- Automatische Zahlungsbestätigung bei Mitgliederbeiträgen
- Möglichkeit Daten ohne direkten Datenbank-Zugriff anzupassen
- Verbesserte Darstellung der Rangliste und Events

Diese sind in der bestehenden Softwarelösung bislang unberücksichtigt geblieben und werden in dieser Arbeit berücksichtigt.

2 Anforderungen

Dieses Kapitel stellt die Anforderungen an ClansCore für den Verein systematisch dar. Die Anforderungen basieren auf klar definierten User Stories , welche die Bedürfnisse der Nutzenden abbilden. Aus diesen User Stories werden die Functional Requirements abgeleitet, welche die konkreten Funktionen des Bots und des Dashboards 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. Im Gegensatz zum Vorprojekt liegt der Schwerpunkt auf der Erweiterung des bestehenden Discord-Bots zu einem modularen und sicheren Verwaltungssystem mit Web-Dashboard (vgl. Abschnitt 1.3).

2.1 Vorgehen

Die Anforderungen wurden aus der Aufgabenstellung, dem Vorprojekt , den Praxiserfahrungen von Vanessa Alves in ihrer Funktion als Präsidentin des Vereins EGSwiss sowie einer ergänzenden Analyse der Domäne abgeleitet. Der Erhebungsprozess erfolgte in mehreren Schritten:

1. User Stories erfassen die Bedürfnisse und Erwartungen der verschiedenen Rollen.
2. Functional Requirements (FR) leiten daraus die technologie-neutralen Systemfunktionen ab.
3. Use Cases (UC) konkretisieren die Interaktionen zwischen Nutzern und System und validieren die FR.
4. Non-Functional Requirements (NFR) definieren überprüfbare Qualitäts- und Leistungsanforderungen.

Im Gegensatz zum Vorprojekt liegt der Schwerpunkt dieser Arbeit auf der Erweiterung des bestehenden Discord-Bots zu einem modularen und sicheren Verwaltungssystem mit Web-Dashboard (vgl. Abschnitt 1.3). Zwischen User Stories, Functional Requirements und Use Cases besteht kein 1:1-Mapping. Die Beziehungen sind häufig viele-zu-viele.

Die Umsetzung und Evaluierung konzentriert sich primär auf die Nutzung der Kernfunktionen über den Discord-Bot sowie auf ausgewählte prototypische Funktionen des Admin-Dashboards auf Desktop-Systemen.

2.2 Priorisierung der Anforderungen

Damit der Entwicklungsprozess auf die wichtigsten Features fokussiert bleibt, erhalten gewisse Anforderungen eine höhere Priorität als andere. Dadurch kann zügig ein Prototyp entwickelt, getestet und auf Basis von Nutzerfeedback iterativ verbessert werden. Die verpflichtenden 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	Diese 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. 1: Beschreibung der Prioritäts-Kategorien

2.3 Legende zum Erfüllungsgrad der Anforderungen

Die Resultate der Anforderungen werden in den Tabellen dokumentiert. Die farbliche Bewertung des Erfüllungsgrads erfolgt erst im Fazit, am Ende des Projekts, nicht während der Spezifikation.

Farbe	Status (wird am Projektende gesetzt)
	Ziel der Anforderung wurde erreicht.
	Ziel der Anforderung wurde zum Teil oder über einen anderen Weg erreicht.
	Ziel der Anforderung wurde nicht erreicht.

Tab. 2: Beschreibung der Resultats-Kategorien

2.4 User Stories

Die nachfolgenden User Stories beschreiben die Erwartungen und Bedürfnisse der Nutzenden an das aus dem Vorprojekt erweiterte System ClansCore. Sie werden aus der Sicht der Endanwendenden formuliert und bilden die Grundlage für die Ableitung der Functional Requirements . Die Analyse berücksichtigt die Rollen Vorstand, Mitglied, Nicht-Mitglied sowie die Systemperspektive Verein.

Als Verein...

- möchte ich ClansCore plattformunabhängig und modular betreiben, um auf neue Plattformen reagieren zu können.
- möchte ich Verlässlichkeit und Vertrauen durch Datenintegrität, Sicherheit und regelmässige Sicherungen.

Als Vorstandsmitglied...

- möchte ich eine zentrale Übersicht über Mitglieder, Rollen, Events, Aufgaben und Aktivitäten haben, um operative und strategische Entscheidungen zu treffen.
- möchte ich Aufgaben und Gamification erfassen und steuern sowie aggregierte Auswertungen zu Aktivität und Engagement erhalten.
- möchte ich Zahlungserinnerungen und Zahlungsbestätigungen automatisiert erhalten, um administrativen Aufwand zu reduzieren.
- möchte ich Nachvollziehbarkeit durch Änderungs- und Transaktionsprotokolle sowie Datenintegritätsprüfungen.

Als Mitglied...

- möchte ich Aufgaben, Punkte und Ranglisten plattformunabhängig einsehen, jedoch konsistent zu Discord.
- möchte ich transparente Punkteberechnung verstehen und mich für Aufgaben anmelden.
- möchte ich meine Datenschutzrechte (Auskunft, Löschung, Einwilligung) gemäss den definierten Prozessen ausüben können.

Als Nicht-Mitglied...

- möchte ich mich unabhängig von einer bestehenden Discord-Mitgliedschaft für den Verein bewerben können.

2.5 Functional Requirements

Die Functional Requirements leiten sich aus den User Stories ab und spezifizieren die funktionalen Anforderungen an ClansCore. Sie beschreiben, welche neuen oder erweiterten Funktionen im Rahmen dieser Arbeit umgesetzt oder konzeptionell vorbereitet werden, um die in der Einleitung definierten Ziele zu erreichen (vgl. Abschnitt 1.3).

Die Kernfunktionen umfassen Gamification, Vereins- und Eventverwaltung. Basisfunktionen sichern die erwarteten Standardprozesse ab, ohne den Fokus der Arbeit auf eine spezifische Benutzeroberfläche zu legen.

Im Gegensatz zum Vorprojekt liegt der Schwerpunkt auf Plattformunabhängigkeit, Sicherheit, Datenintegrität und Erweiterbarkeit. Funktionalitäten werden dabei primär über den Discord-Bot realisiert und sind konzeptionell für eine spätere Nutzung über ein webbasiertes Admin-Dashboard vorbereitet.

Das webbasierte Admin-Dashboard ist im Rahmen dieser Arbeit konzeptionell spezifiziert, jedoch nur in Teilen prototypisch umgesetzt. Zentrale Use Cases werden produktiv über den Discord-Bot realisiert.

ID	**FR1**
Name	Authentifizierung und Zugriffskontrolle
Beschreibung	• Der Zugriff auf administrative Funktionen erfolgt rollenbasiert über Discord-Authentifizierung. • Sensible Aktionen sind auf definierte Rollen (z. B. Vorstand) beschränkt. • Authentifizierungs- und Autorisierungsvorgänge werden nachvollziehbar protokolliert. • Für das Admin-Dashboard soll ein sicheres Login-System mit optionaler Multi-Faktor-Authentifizierung (MFA) umgesetzt werden.

Tab. 3: Beschreibung Functional Requirement 1

ID	**FR2**
Name	Online-Mitgliedsbewerbung
Beschreibung	• Nicht-Mitglieder können sich über ein strukturiertes Formular für den Verein bewerben. • Bewerbungen werden automatisch in der Datenbank gespeichert. • Der Vorstand wird über neue Bewerbungen im Discord informiert.

Tab. 4: Beschreibung Functional Requirement 2

ID	**FR3**
Name	Verwaltung von Mitgliedern, Rollen und Events
Beschreibung	• Mitgliederprofile, Rollen und Status können über administrative Funktionen verwaltet werden. • Events werden zentral über den angebundenen Kalender gepflegt und synchronisiert. • Änderungen werden konsistent zwischen Datenbank und Discord synchronisiert. • Eine grafische Verwaltung über ein Dashboard ist optional vorgesehen.

Tab. 5: Beschreibung Functional Requirement 3

ID	**FR4**
Name	Steuerung und Rechteverwaltung
Beschreibung	• Die Zugriffsrechte für Bot-Funktionen werden rollenbasiert gesteuert. • Es ist konfigurierbar, welche Rollen welche Aktionen ausführen dürfen. • Änderungen an Berechtigungen werden protokolliert.

Tab. 6: Beschreibung Functional Requirement 4

ID	FR5
Name	Gamification-Parameterverwaltung
Beschreibung	• Gamification-Parameter (Punkte, Regeln, Aufgabenarten) werden zentral in der Datenbank verwaltet. • Die Erstellung und Gewichtung von Aufgaben erfolgt über administrative Funktionen. • Eine automatisierte Punkteempfehlung während der Aufgabenstellung ist konzeptionell vorgesehen.

Tab. 7: Beschreibung Functional Requirement 5

ID	FR6
Name	Gamification-Datenanalyse
Beschreibung	• Gamification-relevante Ereignisse werden nachvollziehbar gespeichert und ausgewertet. • Aggregierte Kennzahlen (z. B. Ranglisten, Eventteilnahmen) werden regelmässig berechnet. • Die Auswertung erfolgt zeitnah im Rahmen der Systeminteraktionen und periodischer Berechnungen.

Tab. 8: Beschreibung Functional Requirement 6

ID	FR7
Name	Gamification-Ansicht für Mitglieder
Beschreibung	• Mitglieder können ihre Punkte, Ranglisten, Aufgaben und Belohnungen einsehen. • Die Regeln der Punktevergabe sind transparent kommuniziert. • Alle Werte stammen aus der zentralen Datenquelle und sind konsistent zur Discord-Darstellung.

Tab. 9: Beschreibung Functional Requirement 7

ID	FR8
Name	Aufgabenverwaltung und Anmeldung
Beschreibung	• Aufgaben werden durch den Vorstand über administrative Funktionen erstellt und verwaltet. • Mitglieder können sich für Aufgaben anmelden (z. B. über den Discord-Bot). • Der Abschluss von Aufgaben wird durch den Vorstand bestätigt.

Tab. 10: Beschreibung Functional Requirement 8

ID	FR9
Name	Zahlungserinnerung und Bestätigung
Beschreibung	• Das System unterstützt die Verwaltung von Mitgliederbeiträgen. • Bei ausstehenden Zahlungen können Erinnerungen versendet werden. • Erfasste Zahlungen werden nachvollziehbar dokumentiert.

Tab. 11: Beschreibung Functional Requirement 9

ID	FR10
Name	Event-Reminder-System
Beschreibung	• Mitglieder werden automatisch an anstehende Events erinnert. • Es können mehrere Erinnerungen pro Event definiert werden. • Erinnerungen erfolgen primär über Discord, während Direktnachrichten optional sind.

Tab. 12: Beschreibung Functional Requirement 10

ID	FR11
Name	Plattformunabhängigkeit und Erweiterbarkeit
Beschreibung	• Die Architektur trennt Kernlogik von plattformspezifischen Integrationen. • Zentrale Domänenfunktionen sind unabhängig von Discord implementiert. • Neue Plattformen können durch zusätzliche Adapter integriert werden.

Tab. 13: Beschreibung Functional Requirement 11

2.6 Use Cases

Die Use Cases beschreiben detailliert die Interaktionen zwischen Nutzern und ClansCore. Sie zeigen, wie die Functional Requirements umgesetzt werden. Darüber hinaus konkretisieren sie Anforderungen an Sicherheit, Skalierbarkeit und Benutzerfreundlichkeit.

ID	UC1
Name	Login und Authentifizierung (FR1)
Akteur	Vorstand (optional: Mitglied)
Beschreibung	Der Zugriff auf administrative Dashboard-Funktionen ist über eine Authentifizierung vorgesehen.
Voraussetzung	Ein gültiger Dashboard-Account besteht.
Standard-Verlauf	1. Die Person gibt Benutzername und Passwort ein. 2. ClansCore prüft die Zugangsdaten. 3. (Optional) Das System fordert den zweiten Faktor (MFA) an und validiert ihn. 4. Bei Erfolg wird die Person zur Startseite weitergeleitet.
Alternativ-Verlauf	• Ungültige Zugangsdaten führen zur Fehlermeldung. • Die Person hat das Passwort vergessen und kann es zurücksetzen.
Nachbedingung	Die Person ist eingeloggt und der Zugriff wird protokolliert.
Priorität	Hoch
Resultat	Vorstandsmitglieder können sich mit E-Mail und Passwort erfolgreich anmelden. Der Zugriff für Mitglieder und Zweifaktorauthorisierung wurden nicht implementiert. Über ein Admin-Login können Passwörter von Vorstandsmitglieder zurückgesetzt werden. Das Vorstandsmitglied kann mit dem vorherigen Passwort selbst ein neues setzen.

Tab. 14: Beschreibung Fully dressed Use Case 1

ID	UC2
Name	Online-Mitgliedsbewerbung (FR2)
Akteur	Nicht-Mitglied, Vorstand
Beschreibung	Nicht-Mitglieder bewerben sich ausserhalb von Discord. Der Vorstand wird kanal-neutral informiert (Dashboard-Hinweis / Discord-Nachricht / E-Mail). Die konkrete Ausprägung ist optional konfigurierbar.
Voraussetzung	Dashboard ist mit der Vereinsdatenbank verbunden.
Standard-Verlauf	1. Nicht-Mitglied füllt das Bewerbungsformular aus und stimmt der Datenverarbeitung zu. 2. ClansCore speichert die Bewerbung und informiert den Vorstand im Dashboard. 3. Der Vorstand prüft die Angaben und entscheidet (Annahme / Ablehnung). 4. Bei Annahme wird der Mitgliedsstatus gesetzt und die Synchronisation mit Discord ausgelöst. 5. Bewerbende erhalten eine Status-Benachrichtigung über den Entscheid.
Alternativ-Verlauf	• Unvollständige / fehlerhafte Angaben führen zu einem Validierungsfehler und Korrekturaufforderung. • Ablehnung führt zu einer Benachrichtigung mit Begründung (sofern vorgesehen).
Nachbedingung	Es wird im System dokumentiert und Rollen / Synchronisation sind konsistent.
Priorität	Mittel
Resultat	Die Online-Mitgliedsbewerbung wurde implementiert, ist in der aktuellen Version jedoch deaktiviert, da ein Workflow zur Annahme über ein Vorstandsmitglied innerhalb des Admin-Dashboards oder per E-Mail bewusst nicht umgesetzt wurde.

Tab. 15: Beschreibung Fully dressed Use Case 2

ID	UC3
Name	Verwaltung von Mitgliedern, Rollen, Events und Rechten (FR3)
Akteur	Vorstand
Beschreibung	Der Vorstand verwaltet Mitglieder, Rollen, Events und Berechtigungen zentral im Dashboard.
Voraussetzung	Dashboard ist mit der Vereinsdatenbank verbunden und das Vorstandsmitglied ist eingeloggt.
Standard-Verlauf	1. Vorstand ruft den Verwaltungsbereich auf. 2. Anzeigen / Bearbeiten von Mitgliedern, Rollen und Events. 3. Konfiguration von Zugriffsrechten und erlaubten Aktionen. 4. Änderungen werden gespeichert und synchronisiert.
Alternativ-Verlauf	• Ein Validierungsfehler führt zur Fehlermeldung und die Änderung wird nicht übernommen.
Nachbedingung	Änderungen sind konsistent persistiert und protokolliert.
Priorität	Hoch
Resultat	Die Verwaltung von Mitgliedern und Rollen wurde umgesetzt. Die Verwaltung von Zugriffsrechten wurde bewusst nicht implementiert, da die Interaktion von Mitgliedern mit ClansCore weiterhin ausschliesslich über Discord besteht und die Rechte dort direkt angepasst werden. Das Hinzufügen, Bearbeiten und Löschen von Events wurde implementiert, ist jedoch aktuell nicht freigeschaltet, da die Synchronisation mit dem Discord-Bot über ein Webhook vorbereitet, jedoch bewusst nicht umgesetzt wurde.

Tab. 16: Beschreibung Fully dressed Use Case 3

ID	UC4
Name	Gamification-Parameter verwalten (FR5)
Akteur	Vorstand
Beschreibung	Der Vorstand pflegt die Gewichtungen / Parameter, welche für Punktevorschläge verwendet werden.
Voraussetzung	Vorstandsmitglied ist eingeloggt.
Standard-Verlauf	1. Öffnen der Parameterverwaltung. 2. Anpassen der Parameter und Speichern.
Alternativ-Verlauf	Unvollständige oder fehlerhafte Eingaben ergeben einen Validierungsfehler.
Nachbedingung	Parameter sind aktuell und künftige Punktevorschläge berücksichtigen die Anpassungen.
Priorität	Hoch
Resultat	Die Gamification-Parameter (Punkte-pro-CHF, Aufgabetypen, Belohnungen usw.) können im Dashboard hinzugefügt und angepasst werden.

Tab. 17: Beschreibung Fully dressed Use Case 4

ID	UC5
Name	Aufgabenverwaltung mit Punktevorschlag (FR5 , FR8)
Akteur	Vorstand
Beschreibung	Der Vorstand erstellt und veröffentlicht Aufgaben. ClansCore schlägt basierend auf Parametern eine Punktezahl vor.
Voraussetzung	Das Dashboard ist verbunden und das Vorstandsmitglied eingeloggt. Die Gamification-Parameter sind gepflegt.
Standard-Verlauf	1. Vorstand erfasst eine Aufgabe (Metadaten, Kriterien). 2. System zeigt einen anhand Parameter berechneten Punktevorschlag. 3. Vorstand bestätigt / ändert und veröffentlicht die Aufgabe.
Alternativ-Verlauf	• Wenn keine Parameter vorhanden sind, gibt es keinen Vorschlag. Die Aufgabe kann dennoch mit manueller Punktezahl angelegt werden. • Ein Validierungsfehler führt zur Fehlermeldung.
Nachbedingung	Aufgabe ist veröffentlicht und im System gespeichert.
Priorität	Hoch
Resultat	Der Vorstand kann über den Discord-Bot Aufgaben erstellen und erhält dabei einen Vorschlag zur Punktevergabe. Neu gibt es zudem den Discord-Befehl `/statustasks` für die zentrale Übersicht aller Aufgaben und ihren jeweiligen Status. Die Erstellung und Bearbeitung von Aufgaben über das Dashboard wurde implementiert aber deaktiviert, da die Synchronisation mit dem Discord-Bot über ein Webhook vorbereitet, aber bewusst nicht implementiert wurde.

Tab. 18: Beschreibung Fully dressed Use Case 5

ID	UC6
Name	Ansicht Aufgaben, Ranglisten und Punkte (FR6 , FR7)
Akteur	Vorstand (optional: Mitglied)
Beschreibung	Nutzende sehen Aufgaben, Ranglisten, Punkte, Rewards, Historie und die Regeln der Punktevergabe. Der Vorstand erhält zusätzliche Analyseansichten.
Voraussetzung	Dashboard ist verbunden und die Person ist eingeloggt.
Standard-Verlauf	1. Die Person ruft die Gamification-Ansicht auf und sieht Aufgaben, Ranglisten sowie den eigenen Punktestand. 2. Vorstand ruft Analyse-Dashboard auf (Aggregationen / Trends).
Alternativ-Verlauf	• Wenn keine Aufgaben verfügbar sind, werden entsprechende Hinweise angezeigt.
Nachbedingung	Gamification-Daten werden konsistent und aktuell angezeigt.
Priorität	Hoch
Resultat	Vorstandsmitglieder können die Ranglisten, Aufgaben, Rewards, Punktestand und Historie aller Mitglieder sehen. Da sich Mitglieder nicht in das Dashboard einloggen können, haben sie abgesehen des `/getdata` Befehls über den Discord-Bot keine Möglichkeit ihre eigenen Daten (Punktestand, Aufgaben, Transaktionen usw.) anzusehen.

Tab. 19: Beschreibung Fully dressed Use Case 6

ID	UC7
Name	Anmeldung zu Aufgaben (FR7 , FR8)
Akteur	Mitglied, Vorstand
Beschreibung	Mitglieder melden sich über das Dashboard für Aufgaben an. Die Freigabe erfolgt durch den Vorstand.
Voraussetzung	Mitglied ist im Dashboard eingeloggt und die Aufgabe veröffentlicht.
Standard-Verlauf	1. Mitglied wählt eine Aufgabe und meldet sich an. 2. System speichert die Anmeldung. 3. Vorstand prüft Abschluss und gibt Punkte frei.
Alternativ-Verlauf	• Ein Fehler bei der Anmeldung führt zu einer Fehlermeldung und erneute Eingabe.
Nachbedingung	Anmeldung / Abschlussstatus sind korrekt gespeichert und Punkte entsprechend gutgeschrieben.
Priorität	Mittel
Resultat	Da sich Mitglieder nicht in das Dashboard einloggen können, wurde dieses Feature nicht implementiert.

Tab. 20: Beschreibung Fully dressed Use Case 7

ID	UC8
Name	Benachrichtigungen und Informationen zu Zahlungen (FR9)
Akteur	Mitglied, Vorstand, System
Beschreibung	Beiträge werden jährlich fällig. ClansCore automatisiert die Erinnerung und die revisionssichere Rückmeldung zu eingegangenen Zahlungen.
Voraussetzung	Ein Mitglied hat einen ausstehenden Mitgliederbeitrag. Zahlungsdaten sind im System hinterlegt.
Standard-Verlauf	1. Das System prüft periodisch Fälligkeiten und identifiziert ausstehende Zahlungen. 2. Das System versendet eine Zahlungserinnerung (Discord oder Dashboard-Benachrichtigung). 3. Die Zahlung wird erfasst (z. B. manuelle Verbuchung oder Import). 4. Das System speichert Datum und Referenz revisionssicher. 5. Der Vorstand erhält automatisch eine Zahlungsbestätigung im Dashboard.
Alternativ-Verlauf	• Zahlung kann nicht eindeutig zugeordnet werden: der Vorstand wird zur Klärung aufgefordert (offene Position im Dashboard). • Bei erneuter Nichtzahlung nach Erinnerung erscheint der Fall in der Liste „Überfällig“. Der Vorstand entscheidet das weitere Vorgehen.
Nachbedingung	Der Zahlungsstatus des Mitglieds ist aktuell und protokolliert.
Priorität	Mittel
Resultat	Die erforderlichen Anpassungen an der Datenbank wurden umgesetzt, das entsprechende Feature jedoch aufgrund von Planungsänderungen bewusst nicht implementiert.

Tab. 21: Beschreibung Fully dressed Use Case 8

ID	UC9
Name	Event-Erinnerung und Benachrichtigungssystem (FR10)
Akteur	Mitglied, Vorstand, System
Beschreibung	ClansCore erinnert Mitglieder automatisch an anstehende Events. Das System unterstützt mehrere Reminder-Stufen und versendet diese plattformunabhängig über Discord, Dashboard-Benachrichtigungen oder andere integrierte Kanäle.
Voraussetzung	Ein Event ist im System erfasst und besitzt konfigurierte Erinnerungstermine.
Standard-Verlauf	1. Das System prüft periodisch alle aktiven Events und identifiziert jene, für die Erinnerungen fällig sind. 2. Das System versendet die konfigurierte Event-Erinnerung (z. B. Discord-DM, Dashboard-Notifikation oder alternative Plattform). 3. Mitglieder erhalten die Benachrichtigung und sehen Details (Datum, Ort, Teilnahmehinweise). 4. Wenn mehrere Reminder konfiguriert sind, sendet das System automatisch weitere Nachrichten bis zum Eventbeginn.
Alternativ-Verlauf	• Ein Event wurde geändert oder abgesagt: Das System sendet eine aktualisierte Event-Mitteilung an alle betroffenen Mitglieder. • Der Erinnerungszeitpunkt ist ungültig oder liegt in der Vergangenheit: Das System protokolliert den Fehler und überspringt den fehlerhaften Reminder.
Nachbedingung	Alle fälligen Erinnerungen wurden zuverlässig ausgelöst und protokolliert. Mitglieder sind über anstehende Events informiert.
Priorität	Mittel
Resultat	Die Basis-Event-Erinnerung (24 Stunden vor Event) wurde umgesetzt. Die Konfiguration mehrerer Reminder-Stufen und alternative Kanäle wurden bewusst nicht implementiert, sind in der Datenbank jedoch vorbereitet für eine spätere Erweiterung.

Tab. 22: Beschreibung Fully dressed Use Case 9

ID	UC10
Name	Weitere Plattform-Integration (FR11)
Akteur	Nicht-Mitglied, Mitglied, System
Beschreibung	ClansCore stellt prototypisch Funktionen ausserhalb von Discord oder Dashboard bereit (Bewerbung, Datenabfrage) und synchronisiert zentrale Daten.
Voraussetzung	Neue Plattform ist mit der zentralen Datenbank verbunden.
Standard-Verlauf	1. Nicht-Mitglied reicht Bewerbung über die Plattform ein (UC2). 2. System persistiert Daten und informiert den Vorstand. 3. Vorstand entscheidet. Bei Annahme wird Mitgliedsstatus gesetzt und Daten synchronisiert. 4. Mitglied kann auf der neuen Plattform eigene Personendaten einsehen.
Alternativ-Verlauf	• Ablehnung führt zu keiner Aufnahme und die bewerbende Person wird informiert.
Nachbedingung	Zentrale Daten sind konsistent und Plattformunabhängigkeit ist prototypisch nachgewiesen.
Priorität	Niedrig
Resultat	Das Backend wurde so umstrukturiert, dass die Implementierung einer weiteren Plattform grundsätzlich möglich ist. Die Umsetzung dieser zusätzlichen Plattform erfolgte jedoch nicht.

Tab. 23: Beschreibung Fully dressed Use Case 10

2.7 Non-Functional Requirements

Die Non-Functional Requirements definieren die Qualitätsmerkmale von ClansCore. Sie betreffen Sicherheit, Wartbarkeit, Zuverlässigkeit, Kompatibilität und Benutzerfreundlichkeit. Jede Anforderung ist überprüfbar und wird im Projektverlauf gemessen oder getestet. Allgemeine Qualitätsziele wie Testabdeckung, CI/CD-Deployments oder Fehlertoleranz werden als etablierte Entwicklungsstandards umgesetzt und sind nicht Teil der expliziten Anforderungen.

ID	NFR1
Beschreibung	Die Codequalität des ClansCore-Systems erfüllt die im Projekt definierten Linting- und Formatierungsrichtlinien. Im CI-Prozess dürfen keine Fehler vom Typ „error“ auftreten.
Anforderungen	Maintainability - Code Quality
Priorität	Hoch
Messung	Automatische Prüfung durch ESLint in der CI-Pipeline. Ziel: 0 Fehler im Main-Branch.
Testen	Bei jedem Commit und Merge Request.
Resultat	Der Linter gibt keine Fehlermeldungen im Main-Branch aus.

Tab. 24: Beschreibung Non-Functional Requirement 1

ID	NFR2
Beschreibung	Das Admin-Dashboard bietet ein konsistentes, benutzerfreundliches Design mit klarer Navigation und unmittelbarem Feedback bei Interaktionen.
Anforderungen	Usability - Consistency
Priorität	Hoch
Messung	Usability-Test mit mindestens 3 Testpersonen. Erfolgsziel: 80 % der Aufgaben ohne Hilfe innerhalb von ≤ 2 Minuten lösbar.
Testen	Am Ende der Implementierungsphase.
Resultat	Die definierten Usability-Tests wurden für das Admin-Dashboard nicht durchgeführt. Stattdessen ist nach Abgabe dieser schriftlichen Arbeit eine zweiwöchige Nutzungssimulation mit qualitativer Befragung geplant (vgl. Abschnitt 6.1.5). Die Anforderung wird daher zum aktuellen Zeitpunkt nur teilweise erfüllt.

Tab. 25: Beschreibung Non-Functional Requirement 2

ID	NFR3
Beschreibung	Das Dashboard ist mit den aktuellen Versionen von Chrome und Firefox vollständig funktionsfähig. Layout und Interaktionen sind Desktop-first optimiert. Mobile Nutzung ist nicht Teil des Projektumfangs.
Anforderungen	Compatibility - Interoperability
Priorität	Hoch
Messung	Manueller Cross-Browser-Test. Erfolgsziel: 100 % Funktionsumfang und fehlerfreie Darstellung in beiden Browsern.
Testen	Am Ende der Implementierungsphase.
Resultat	Das Dashboard wurde erfolgreich mit Chrome und Firefox getestet.

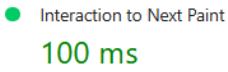
Tab. 26: Beschreibung Non-Functional Requirement 3

ID	NFR4
Beschreibung	Personenbezogene Daten werden nach dem Prinzip „Privacy by Design“ verarbeitet. Speicherung und Übertragung erfolgen ausschliesslich verschlüsselt (TLS, Datenbankverschlüsselung).
Anforderungen	Security - Confidentiality
Priorität	Hoch
Messung	Überprüfung der implementierten Sicherheitsmechanismen sowie der vorbereiteten TLS-Konfiguration auf Applikationsebene.
Testen	Vor Projektabschluss.
Resultat	Die Anwendung implementiert mehrere Sicherheitsschichten: • Passwörter werden mit bcrypt gehasht gespeichert. • Die API ist durch Helmet Security Headers geschützt. • Die MongoDB-Verbindung erfolgt authentifiziert. • Die Nginx-Konfiguration ist für eine spätere SSL-Terminierung vorbereitet (Reverse-Proxy-Architektur). • Die Aktivierung von HTTPS/TLS wurde im Rahmen des Projekts bewusst nicht umgesetzt und erfordert die Einrichtung von SSL-Zertifikaten auf Infrastrukturebene.

Tab. 27: Beschreibung Non-Functional Requirement 4

ID	NFR5
Beschreibung	Änderungen an Mitgliedern, Rollen und Gamification-Parametern müssen revisionssicher nachvollziehbar sein. Jede Änderung ist eindeutig einer Person oder einem Prozess zuordenbar.
Anforderungen	Security - Accountability
Priorität	Hoch
Messung	Überprüfung von 5 Stichproben im Änderungsprotokoll. Jede Aktion enthält Personen-ID, Zeitstempel und Aktionstyp.
Testen	Am Ende der Implementierungsphase.
Resultat	Es wurde ein Logging-Konzept implementiert, welches über Discord eingesehen werden kann (vgl. Abschnitt 7.5). Ein durchgängiges Logging auf Datenbankebene wurde jedoch nicht umgesetzt.

Tab. 28: Beschreibung Non-Functional Requirement 5

ID	NFR6
Beschreibung	Das Dashboard reagiert auf Benutzerinteraktionen innerhalb von 200 Millisekunden und erfüllt damit die empfohlene Reaktionszeitgrenze von Jeremy Wagner und Barry Pollard. (Wagner & Pollard, 2025)
Anforderungen	Performance - Response Time
Priorität	Mittel
Messung	Automatisierte Performance-Tests (z. B. Lighthouse, Playwright) für Login und Seitenwechsel. Ziel: < 200 ms INP (Interaction to Next Paint).
Testen	Am Ende der Implementierungsphase.
Resultat	Die Reaktionszeit des Dashboards wurde mithilfe von Lighthouse unter Verwendung des Richtwerts Interaction to Next Paint (INP) gemessen. Der gemessene INP-Wert betrug 100 ms und lag damit unter dem angestrebten Zielwert von 200 ms.  **Abb. 4:** INP Wert von Lighthouse für das Dashboard

Tab. 29: Beschreibung Non-Functional Requirement 6

3 Erweitertes Gamification- und Motivationskonzept

Bereits im Vorprojekt wurden erste Erkenntnisse zu Motivation, Hindernissen und Gamification im Vereinskontext erhoben. Diese basierten auf einer Nutzergruppenanalyse mit Nicht-Mitgliedern, Mitgliedern und Vorstand. Die Ergebnisse lieferten eine breite quantitative Datengrundlage, die allgemeine Tendenzen sichtbar machte. Beispielsweise, welche Aktivitäten besonders motivierend wirken oder welche Risiken bei der Einführung von Gamification-Systemen bestehen. (vgl. Vorprojekt, Kap. 4)

Während das Vorprojekt primär die Sicht der Mitglieder sowie die Erfahrungen von Vanessa Alves als Präsidentin von EGSwiss abbildete, verfolgt die vorliegende Arbeit eine andere Perspektive. Sie untersucht qualitativ, wie Vereinsvorstände Motivation aus organisatorischer Sicht verstehen, welche strukturellen Bedingungen sie fördern oder behindern und wie digitale Systeme, wie der Discord-Bot oder das geplante Admin-Dashboard, sie dabei unterstützen können.

Ziel ist es, die im Vorprojekt identifizierten Motivations-Muster kritisch zu hinterfragen und auf ihre langfristige Umsetzbarkeit im System ClansCore zu prüfen. Dafür wurde die Self-Determination Theory (SDT) als theoretische Grundlage gewählt. Sie beschreibt die psychologischen Bedingungen, die Motivation fördern oder hemmen, und diente als Raster zur Entwicklung und Auswertung der Interviews mit Vereinsvorständen.

Die Erkenntnisse aus diesen Interviews werden im Anschluss mithilfe des Octalysis Frameworks nach Yu-kai Chou in konkrete, technisch erweiterbare Gamification-Mechaniken überführt. Das erweiterte Konzept versteht Gamification somit nicht mehr als isoliertes Punktesystem, sondern als integralen Bestandteil eines digitalen Vereinsökosystems, das psychologische, organisatorische und soziale Faktoren verbindet.

3.1 Self-Determination Theory

Die Self-Determination Theory (SDT) nach Deci und Ryan beschreibt Bedingungen, die Motivation entweder unterstützen oder vermindern können. Im Mittelpunkt der Theorie stehen drei zentrale psychologische Grundbedürfnisse:

- **Autonomie**: Das Bedürfnis, selbstbestimmt handeln zu können und Entscheidungen eigenständig zu treffen.
- **Kompetenz**: Das Bedürfnis, sich fähig zu fühlen und Herausforderungen erfolgreich zu bewältigen.
- **Soziale Eingebundenheit**: Das Bedürfnis, sich mit anderen verbunden zu fühlen und Teil einer Gemeinschaft zu sein.

Werden diese Bedürfnisse erfüllt, erhöht sich die Wahrscheinlichkeit für intrinsisch motiviertes Verhalten. Dadurch können Engagement, Lernprozesse und psychisches Wohlbefinden positiv beeinflusst werden. (Weir, 2025)

Auch im Kontext digitaler Technologien lässt sich die SDT anwenden. Sie geht davon aus, dass Nutzende eine Technologie dann langfristig nutzen, wenn die Interaktion mit dem System zur Befriedigung ihrer psychologischen Grundbedürfnisse beiträgt. Das Interface- und Interaktionsdesign wirkt dabei unmittelbar auf das Erleben von Autonomie und Kompetenz. Eine unklare Bedienung oder mangelndes Feedback kann diese Bedürfnisse frustrieren und zur Ablehnung der Technologie führen. Die soziale Eingebundenheit ist besonders relevant, wenn das System Austausch und Anerkennung zwischen Nutzenden fördert. (Peters Dorian, 2018)

Da die SDT sowohl intrinsische als auch extrinsische Motivation berücksichtigt, bietet sie eine geeignete Grundlage, um zu untersuchen, wie Vereinsmitglieder und Vorstände durch digitale Systeme motiviert werden können. Aufbauend auf diesen theoretischen Annahmen wurden qualitative Interviews mit Vereinsvorständen durchgeführt, um die theoretischen Konzepte mit der Praxis abzugleichen.

Diese drei Bedürfnisse dienten als analytische Kategorien in der Auswertung der Interviews. Aussagen der Befragten wurden entlang dieser Dimensionen kodiert und hinsichtlich ihrer Bedeutung für Vereinsmotivation interpretiert.

3.2 Interviews zur Motivation von Vereinsmitgliedern

Zur Überprüfung der theoretischen Annahmen der Self-Determination Theory (SDT) wurden vier leitfadengestützte Interviews mit Präsidenten verschiedener Schweizer Gaming-Vereine durchgeführt. Ziel war es, zu verstehen, welche Faktoren in der Vereinsrealität Motivation fördern oder behindern, wie Vorstände diese gezielt unterstützen und welche Rolle digitale Systeme dabei spielen. Während das Vorprojekt vorwiegend die Perspektive der Mitglieder beleuchtete (vgl. Vorprojekt, Kap. 4.2), fokussiert diese Untersuchung die organisatorische Sicht der Vereinsleitung.

Die Auswertung erfolgte qualitativ-interpretativ entlang der drei SDT-Grundbedürfnisse Autonomie, Kompetenz und soziale Eingebundenheit. Sie diente dazu, wiederkehrende Muster und Einflussfaktoren auf die Vereinsmotivation zu identifizieren und im Kontext digitaler Vereinsarbeit zu interpretieren (vgl. Anhang Abschnitt 13).

Die Interviews zeigen, dass soziale Bindungen, klare Kommunikation und freiwillige Beteiligung die stärksten Treiber langfristiger Motivation darstellen. Freundschaften und gemeinsame Erlebnisse fördern die Verbundenheit, während Zwang oder übermässige Kontrolle als demotivierend empfunden werden. Symbolische Anerkennung (z. B. durch Danksagungen oder sichtbare Rollen) wirkt nachhaltiger als materielle Belohnungen. Fairnessmechanismen wie saisonale Resets werden als motivierend und ausgleichend wahrgenommen, da sie Einstiegshürden für neue Mitglieder senken und Gleichheit fördern.

3.2.1 SDT-basierte Analyse der Interviewergebnisse

Die nachfolgende Tabelle fasst die zentralen Ergebnisse zusammen und ordnet sie den drei Grundbedürfnissen der Self-Determination Theory zu:

SDT-Bedürfnis	Interview-Erkentnis
Autonomie	Freiwilligkeit, Wahlmöglichkeiten und flexible Beteiligungswege wurden als zentral hervorgehoben. Digitale Werkzeuge sollen unterstützen statt steuern. Transparente, optionale Pfade fördern Selbstbestimmung, während starre Vorgaben sie einschränken.
Kompetenz	Motivation steigt, wenn Fortschritt sichtbar ist (Feedback, Meilensteine, Statistiken) und Beiträge im Vereinskontext wertgeschätzt werden. Ranglisten fördern kurzfristige Aktivität, nachhaltiger sind Aufgaben- und Lernpfade mit klarer Rückmeldung.
Soziale Eingebundenheit (Relatedness)	Gemeinschaft entsteht durch gemeinsame Aktivitäten und sichtbare Anerkennung digitaler Beiträge. Online-Kanäle ergänzen, aber ersetzen keine realen Erlebnisse. Digitale Features sollen Beziehungen sichtbar und anschlussfähig machen (z. B. Team-Achievements, öffentliche Danksagungen).

Tab. 30: Mapping der Interviewergebnisse zu den SDT-Grundbedürfnissen

Insgesamt zeigt sich, dass soziale Eingebundenheit und Kompetenz gleichermassen zentrale Einflussfaktoren für langfristige Motivation darstellen. Autonomie spielt ebenfalls eine bedeutende, aber leicht geringere Rolle und wird insbesondere durch Transparenz, Feedback und flexible Beteiligungsmöglichkeiten gestärkt.

3.2.2 Querschnittliche Implikationen

Aus der Analyse ergeben sich drei zentrale Muster für die Gestaltung motivierender Vereinsstrukturen:

1) **Kommunikation als Schlüssel**: Mehrstufige Erinnerungen mit klaren Call-to-Actions erhöhen die Teilnahmebereitschaft.

2) **Fairnessmechaniken**: Zeitlich begrenzte Punkte und Resets senken Einstiegshürden und vermeiden dauerhafte Ungleichgewichte.

3) **Privatsphäre-wahrende Transparenz**: Aggregierte Kennzahlen fördern Orientierung und Motivation, ohne Privatsphäre zu verletzen.

Diese Erkenntnisse bestätigen die im Vorprojekt gewonnenen Einsichten zur Bedeutung sozialer Erlebnisse und liefern praxisnahe Anhaltspunkte für die Weiterentwicklung des Systems im Sinne einer motivierenden, fairen und nachhaltigen Vereinsführung.

3.3 Octalysis Framework

Das Octalysis Framework nach Yu-kai Chou (2015) ergänzt die psychologische Perspektive der Self-Determination-Theory (SDT) um ein praxisorientiertes Modell zur Analyse und Gestaltung von Motivation in digitalen Systemen. Es identifiziert acht sogenannte Core Drives, welche das Verhalten von Nutzern in Gamification-Umgebungen steuern. (Chou, 2015)

Die acht Core Drives sind:

1. **Epic Meaning & Calling**: das Gefühl, Teil von etwas Grösserem zu sein.
2. **Development & Accomplishment**: Motivation durch Fortschritt und Belohnung.
3. **Empowerment of Creativity & Feedback**: Motivation durch kreative Freiheit und Rückmeldung.
4. **Ownership & Possession**: Motivation durch Besitz oder Kontrolle.
5. **Social Influence & Relatedness**: Motivation durch Gemeinschaft, Anerkennung oder Wettbewerb.
6. **Scarcity & Impatience**: Motivation durch Verknappung oder Limitierung.
7. **Unpredictability & Curiosity**: Motivation durch Neugier und Überraschung.
8. **Loss & Avoidance**: Motivation durch die Vermeidung negativer Konsequenzen.

Während die SDT die Qualität der Motivation (intrinsisch vs. extrinsisch) erklärt, zeigt das Octalysis-Framework konkrete Design-Hebel auf, mit denen diese Motivation im System gefördert werden kann. Für ClansCore wurde das Modell genutzt, um die im Vorprojekt umgesetzten Gamification-Elemente mit einer Bewertungs-Skala von 1 (noch gar nicht umgesetzt) bis 10 (vollumfänglich integriert) einzuschätzen und die Balance zwischen intrinsischen und extrinsischen Anreizen zu bestimmen. Die Skala dient dabei ausschliesslich der qualitativen Einordnung und erhebt keinen Anspruch auf quantitative Messgenauigkeit.

Die Analyse der bestehenden Gamification-Mechaniken in Abb. 5 und Tab. 31 zeigt eine Schwerpunktsetzung auf Development & Accomplishment (CD2) sowie Social Influence & Relatedness (CD5). Diese Drives erzeugen eine messbare Belohnungsstruktur und fördern sozialen Vergleich innerhalb des Vereins. Intrinsische Anteile sind vorhanden, aber weniger stark systemisch verankert.

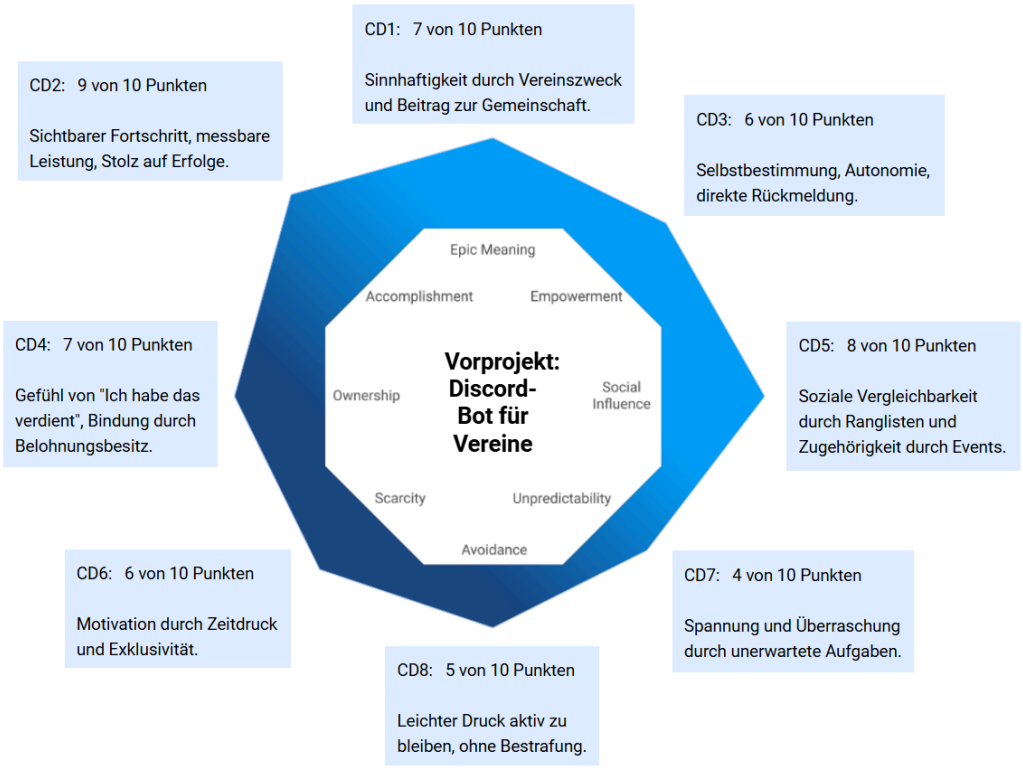


Abb. 5: Die Einordnung des Vorprojektes im Octalysis-Modell

Core Drive		Einschätzung (Stand Vorprojekt)
CD1	**Epic Meaning & Calling** (7 von 10 Punkten)	Durch Rollen wie Mitglied oder Vorstand sowie den klar kommunizierten Vereinszweck entsteht ein Sinnbezug und Gemeinschaftsgefühl.
CD2	**Development & Accomplishment** (9 von 10 Punkten)	Das Punktesystem und die Ranglisten (saisonal, jährlich, allzeit) sind stark ausgeprägt. Sie erzeugen sichtbaren Fortschritt und messbaren Erfolge, die Hauptquelle extrinsischer Motivation.
CD3	**Empowerment of Creativity & Feedback** (6 von 10 Punkten)	Aufgaben können selbstständig übernommen und abgeschlossen werden. Rückmeldungen erfolgen durch Annahme oder Ablehnung seitens des Vorstands. Dies unterstützt Autonomie und Selbstwirksamkeit.
CD4	**Ownership & Possession** (7 von 10 Punkten)	Punkte und Belohnungen (Merchandise, Rabatte, Gratis-Mitgliedschaft) erzeugen ein Gefühl von Besitz und Zugehörigkeit.
CD5	**Social Influence & Relatedness** (8 von 10 Punkten)	Öffentliche Ranglisten sowie gemeinsame Events fördern Anerkennung und Verbundenheit in der Community.
CD6	**Scarcity & Impatience** (6 von 10 Punkten)	Zeitlich begrenzte Saisons und manuell vergebene Belohnungen erzeugen moderate Verknappung und fördern Aktivität innerhalb klarer Zyklen.
CD7	**Unpredictability & Curiosity** (4 von 10 Punkten)	Neben gelegentlichen, kurzfristig ausgeschriebenen Aufgaben fehlen explizite Zufalls- / Überraschungsmechaniken. Potenzial für Mystery-Belohnungen oder Wochen-Boosts sind vorhanden.
CD8	**Loss & Avoidance** (5 von 10 Punkten)	Ranglisten-Resets motivieren zur regelmässigen Teilnahme. Die Punkte selbst bleiben bestehen. Der Anreiz entsteht durch periodisch erneuerte Wettbewerbschancen ohne starken Druck.

Tab. 31: Beschreibung der Einordnung vom Vorprojekt im Octalysis-Modell

Das System adressiert insbesondere die extrinsisch motivierenden Core Drives Development & Accomplishment (CD2) und Ownership & Possession (CD4), welche durch das Punktesystem, Ranglisten und Belohnungen stark ausgeprägt sind. In Kombination mit Social Influence & Relatedness (CD5) entsteht eine klare, faire sowie gemeinschaftsorientierte Wettbewerbsdynamik.

Intrinsische Faktoren wie Epic Meaning & Calling (CD1) sind zwar über den Vereinszweck vorhanden, werden im Systemdesign jedoch nur indirekt vermittelt. Empowerment of Creativity & Feedback (CD3) zeigt erste Ansätze durch selbstständige Aufgabenübernahme und Vorstands-Feedback, bleibt aber im Potenzial begrenzt. Explorative und überraschende Elemente (CD7, Curiosity) sowie dynamische Risikofaktoren (CD8, Loss & Avoidance) sind bislang schwächer ausgeprägt.

Insgesamt ist der Discord-Bot in dieser Form primär extrinsisch und kompetenzorientiert motivierend, was Einstieg und Aktivierung stark unterstützt. Für eine nachhaltige, qualitativ hochwertige Motivation im Sinne der SDT sollte die Gewichtung intrinsischer Drives (insbesondere Empowerment, Curiosity, Meaning) erhöht werden, um Selbstbestimmung, Kreativität und persönliche Identifikation mit dem Vereinsleben langfristig zu fördern.

3.4 Mögliche Erweiterungen, Chancen und Regeln

Auf Basis der Interview-Erkenntnisse, SDT und der Octalysis-Analyse des Vorprojekts ergeben sich konkrete Erweiterungen, Chancen sowie Gestaltungs- und Fairnessregeln für ClansCore. Ziel ist es, die aktuell dominanten extrinsischen Treiber (CD2, CD4, CD5) um intrinsische Faktoren (CD1, CD3, CD7) zu ergänzen, ohne Fairness, Transparenz und Datenschutz zu gefährden.

3.4.1 Designprinzipien

Aus den theoretischen und empirischen Erkenntnissen wurden zentrale Gestaltungsprinzipien für eine Weiterentwicklung von ClansCore abgeleitet. Sie stellen sicher, dass neue Funktionen Motivation, Fairness und Datenschutz im Sinne der SDT unterstützen.

Prinzip	SDT-Bezug	Octalysis-Hebel
Soziale Verbundenheit vor Wettbewerb	Relatedness: Anerkennung, Zugehörigkeit, Teamgefühl	• CD5 : Social Influence • CD1 : Meaning
Sichtbarer Fortschritt mit sinnvollem Kontext	Kompetenz: Feedback, Meilensteine, Beiträge „für den Verein“	• CD2 : Development • CD4 : Ownership
Selbstbestimmte Beteiligung statt Zwang	Autonomie: Wahlfreiheit, optionale Pfade	• CD3 : Empowerment
Fairness & Datenschutz-by-Design	Alle drei Grundbedürfnisse nicht untergraben	• CD6 : Scarcity • CD8 : Loss (transparent)

Tab. 32: Leitprinzipien für ClansCore

3.4.2 Zukünftige Erweiterungen und Regeln

Die folgenden Erweiterungen sind so konzipiert, dass sie Autonomie (SDT) stärken und die Octalysis-Drives CD1/ CD3/CD7 gezielt ausbalancieren. Die Wahl der Priorität ergibt sich aus der Wirkung und Umsetzbarkeit.

ID	Erweiterung	Motivationswirkung	Priorität
E1	Mehrstufiges Reminder-System für Events (Save-the-Date, Wochen-, Tages-Reminder, optional Direktnachricht). Optional: Export CSV / ICS im Dashboard (Kalendersubscribe).	• Beseitigt Kommunikations-Engpass • Steigert Teilnahme ohne Zwang (SDT: Kompetenz / Autonomie) • CD2 und CD5	Hoch
E2	Datensparsame Vereins-Analytics im Dashboard (nur Aggregat aktiver Mitglieder, Eventquote, Helferquote und Trendlinien, ohne personenbezogenes Dauertracking).	• Kompetenz der Vorstände (Steuerungsfähigkeit) (SDT: Kompetenz) • Keine Untergrabung der Autonomie / Privacy (SDT: Autonomie) • CD2	Hoch
E3	Punkte haben ein Ablaufdatum (z. B. jährlicher Reset oder schrittweiser Verfall zur neuen Saison).	• Fördert regelmässige Aktivität anstelle von kurzfristigem Punktesammeln • Verhindert „Horten“ und stärkt Fairness zwischen neuen und alten Mitgliedern • Bietet wiederkehrende Chancen auf Erfolg (SDT: Kompetenz / Autonomie) • CD8 in moderater, motivierender Form	Hoch
E4	Anerkennungssysteme in Form von speziellen Hervorhebungen (z. B. „Helfer des Monats“, Rollenfarben, öffentliche Danksagungen).	• Sichtbare Wertschätzung (SDT: Relatedness) • Ownership über sichtbare Symbole • CD1, CD4 und CD5	Mittel
E5	Kooperative Formate wie Team-Quests, Gemeinschaftsziele (z. B. „50 Teilnahmen in 30 Tagen“) mit serverweiter Belohnung (z. B. Punktevergabe).	• Verlagerung von Wettbewerb zu Kooperation • Stärkt Verbundenheit & Sinn (SDT: Relatedness) • CD1, CD3, CD5	Mittel
E6	Event-Recaps im Kanal (kurzer Auto-Post mit Teilnehmerzahl, Dank an Helfer, optional mit Bild).	• Verstärkt Anerkennung und Sinn (SDT: Relatedness) • Erhöht Sichtbarkeit realer Beiträge • CD1, CD2 und CD5	Mittel
E7	Leichte Curiosity-Elemente wie zum Beispiel „Wochen-Boost“ (+20 % für genau eine Aktivität, transparent angekündigt) rotierend per Saison (kein Zufall bei Belohnungen).	• Erhöht Abwechslung ohne Intransparenz • Schützt Fairness • CD6 und CD7	Niedrig

Tab. 33: Erweiterungen und Zielwirkung

Die geplanten Regeln stellen Fairness, Datenschutz und langfristige Motivation sicher. Es übernimmt unter anderem die in der Vorarbeit bereits beschriebenen Regeln (vgl. Vorprojekt, Kap. 4.4) und formuliert sie mit den neuen Erkenntnissen systemnah für ClansCore.

Die Regeln lauten wie folgt:

1. **Keine Macht- oder Stimmenvorteile durch Punkte**: Verhindert toxische Hierarchien; erhält demokratische Vereinsprozesse (SDT: Relatedness / Autonomie).
2. **Punkte sind nicht übertragbar**: Verhindert Pay-to-Win / Transferhandel, schützt Fairness und Ownership.
3. **Saisonale Ranglisten-Resets, wobei Punkte erhalten bleiben**: Ergibt niedrige Einstiegshürden für Neumitglieder, da der Wettbewerb periodisch neu geöffnet wird (CD6 und CD8).
4. **Transparente Punktequellen und Limits**: Jede vergütete Aktivität ist dokumentiert. Limits an Aufgaben (täglich / monatlich) vermeiden übertriebenes Sammeln ohne fertigstellen.
5. **Datensparsame Analytics (nur Aggregat)**: Steuerung ohne personenbezogenes Dauertracking sowie DSGVO- und Vereinskultur-konform.
6. **Freiwilligkeit**: Gamification ist optional und wird ohne Druck als zusätzliches Vereins-Angebot angesehen (SDT: Autonomie).
7. **Keine Überraschungs-Strafen**: Loss & Avoidance nur über Ranglisten-Resets, keine verdeckten Bestrafungen. Schützt intrinsische Motivation.
8. **Moderierte Curiosity**: Wochen-Boost oder Community-Quests sind angekündigt und fair verteilt. Keine Zufallsbelohnungen mit Ungleichheitseffekt.
9. **Angepasste Ästhetik und Sprache**: Gamification-Elemente sollen die Vereinsidentität widerspiegeln und nicht zu verspielt oder infantil wirken. Dadurch bleibt das System glaubwürdig und akzeptiert.

Bei der Weiterentwicklung von ClansCore ist darauf zu achten, dass Motivation nicht zu stark über Belohnungen gesteuert wird. Um eine Überbetonung extrinsischer Faktoren zu vermeiden, werden kooperative Formate und persönliche Zielpfade gezielt priorisiert. Ungleichheit durch Zufallsmechaniken wird ausgeschlossen, indem Curiosity-Elemente stets angekündigt und fair verteilt bleiben. Datenschutz wird durch strikte Aggregation der Kennzahlen gewährleistet.

3.4.3 Weiterentwicklungsperspektive und Evaluation

Die in diesem Kapitel beschriebenen Erweiterungen wurden bewusst nicht im Rahmen dieser Arbeit implementiert. Sie stellen eine konzeptionelle Weiterentwicklung dar, die auf den empirischen Erkenntnissen, den theoretischen Grundlagen (SDT, Octalysis) sowie der bestehenden Systemarchitektur von ClansCore aufbaut. Ziel ist es aufzuzeigen, wie das System zukünftig strukturiert und regelbasiert erweitert werden könnte, ohne Qualitätsanforderungen wie Fairness, Transparenz und Datenschutz zu untergraben. Die Erweiterungen sind daher nicht als Umsetzungsplan, sondern als technisch realistische und priorisierte Konzeptvorlage zu verstehen.

Aus konzeptioneller Sicht lassen sich die Erweiterungen hinsichtlich Nutzen und Systemnähe priorisieren. Besonders systemnah sind ein mehrstufiges Reminder-System (E1), ein datensparsames Dashboard mit aggregierten Kennzahlen (E2) sowie Punkte mit Ablaufdatum (E3). Anerkennungsfunktionen (E4), kooperative Formate (E5) und automatisierte Event-Recaps (E6) würden in einem weiteren Schritt insbesondere intrinsische Motivationsfaktoren stärken. Explorative Curiosity-Elemente (E7) sind bewusst als optionale Ergänzung vorgesehen.

Da diese Erweiterungen nicht Bestandteil der Umsetzung waren, konnte ihre Wirksamkeit nicht empirisch überprüft werden. Auf konzeptioneller Ebene lassen sich jedoch geeignete Indikatoren für eine zukünftige Evaluation definieren, darunter:

- Teilnahmequote an Events
- Anteil aktiv engagierter Mitglieder
- Anzahl und Vielfalt übernommener Aufgaben
- Sichtbarkeit von Anerkennungen pro Saison

Die Erhebung dieser Kennzahlen sollte ausschliesslich aggregiert erfolgen und keine personenbezogene Verlaufsanalyse beinhalten. So könnte geprüft werden, ob die konzipierten Mechaniken zu einer nachhaltigeren Beteiligung beitragen, ohne Autonomie oder Datenschutz zu beeinträchtigen.

4 Erweitertes Sicherheitskonzept

Das erweiterte Sicherheitskonzept definiert die konzeptionellen und organisatorischen Massnahmen, mit denen die Vertraulichkeit, Integrität und Verfügbarkeit der in ClansCore verarbeiteten Daten gewährleistet werden. Es baut auf den im Vorprojekt etablierten Grundlagen zu Datensicherheit, Rechteverwaltung und Datenschutz (vgl. Vorprojekt , Kap. 3 Sicherheitskonzept) auf, erweitert diese jedoch in Hinblick auf die neue Systemarchitektur mit einem webbasierten Dashboard und der synchronisierten Datenhaltung zwischen Discord, der Webplattform (Dashboard) und potenziell weiteren Plattformen (UC10 , NFR5).

Das Backend, welches bereits im Vorprojekt als eigenständige, modulare Anwendung konzipiert wurde, bildet weiterhin das zentrale Bindeglied zwischen allen Plattformen. Neu hinzugekommen sind die Anforderungen an eine sichere Kommunikation zwischen Backend und Angular-Frontend sowie an eine konsistente und konfliktfreie Synchronisation von Benutzerdaten, Rollen und Gamification-Informationen über mehrere Zugriffspunkte. Die Systemarchitektur ist somit nicht nur funktional, sondern auch sicherheitstechnisch stärker verteilt, was eine Erweiterung des bisherigen Sicherheitsmodells erforderlich macht. (UC3 , UC6)

4.1 Ziel und Geltungsbereich

Ziel des erweiterten Sicherheitskonzepts ist es, die personenbezogenen und vereinsinternen Daten vor unbefugtem Zugriff, Manipulation und Verlust zu schützen. Der Fokus liegt auf Vertraulichkeit, Integrität und Verfügbarkeit, welche die drei zentralen Schutzziele der Informationssicherheit bilden (NIST, 2020) (NFR4 , NFR5).

Das Konzept erstreckt sich auf alle sicherheitsrelevanten Komponenten des Systems:

- **Backend** (Node.js / Express): zentrale Applikationslogik und API-Kommunikation
- **Discord-Bot** (Discord.js): Interaktionsschnittstelle für Mitgliederaktionen
- **Admin-Dashboard** (Angular): browserbasierte Verwaltungs- und Interaktionsplattform
- **Datenbank** (MongoDB): persistente Speicherung von Personen-, Rollen- und Aktivitätsdaten
- **Infrastruktur und Serverumgebung** (Docker, Linux-Host, GitHub Actions): Containerisierung, CI/CD und physische bzw. virtuelle Serverabsicherung
- **Externe Dienste** (Google Calendar API): Synchronisation von Kalendereinträgen

Nicht berücksichtigt werden die nativen Sicherheitsmechanismen externer Plattformen (Discord, Google), da sie ausserhalb der direkten Kontrolle des Systems liegen.

4.2 Sicherheitsprinzipien

Das Sicherheitskonzept orientiert sich an vier zentrale Prinzipien der Informationssicherheit, welche als Fundament weiterer technischer und organisatorischer Sicherheitsmassnahmen dienen sowie in allen Systemschichten konsistent angewendet werden.

Prinzip	Beschreibung der Umsetzung
Least Privilege	Alle Komponenten und Benutzerrollen besitzen nur jene Rechte, die für ihre Funktion notwendig sind. Vorstandsmitglieder dürfen Daten auf Organisations-Ebene (Mitgliederverwaltung, Events, Punktesystem) bearbeiten, während Mitglieder ausschliesslich ihre eigenen Daten sehen und bearbeiten dürfen. Öffentliche Informationen wie Vereins-Events, Bewerbungsformular, Statuten und Datenschutzrichtlinie sind ohne Login zugänglich. (Bezug: UC3)
Defense in Depth	Sicherheit wird durch mehrere, voneinander unabhängige Schutzebenen gewährleistet, etwa durch Transportverschlüsselung, Container-Isolation, Zugriffsbeschränkungen und rollenbasierte Rechteprüfung. (Murugiah & Karen, 2017; NIST, 2020)
Accountability	Alle sicherheitsrelevanten Aktionen (Login-Versuche, Rollenänderungen, Punkteänderungen) werden protokolliert und sind für berechtigte Personen nachvollziehbar (Bezug: NFR5).
Privacy by Design	Es werden nur notwendige Daten gespeichert, analog zum im Vorprojekt definierten Datensparsamkeitsprinzip (vgl. Vorprojekt, Kap. 3.3 ff.). Die Datenschutzinformationen sind zusätzlich im Dashboard direkt einsehbar (Bezug: NFR4).

Tab. 34: Beschreibung der vier grundlegenden Prinzipien

Die Auswahl der Sicherheitsprinzipien erfolgte auf Basis der im Projekt definierten NFR (insbesondere NFR4 und NFR5), der erweiterten Systemarchitektur sowie etablierter Sicherheitsstandards. Als Referenz dienten insbesondere die Empfehlungen des NIST (SP 800-53) (NIST, 2020) , die OWASP Top 10 (OWASP-Contributors, 2021) sowie die im Vorprojekt identifizierten Datenschutz- und Sicherheitsanforderungen.

Die gewählten Prinzipien adressieren gezielt die zentralen Risiken eines verteilten Systems mit mehreren Zugriffspunkten und stellen sicher, dass Sicherheitsanforderungen konsistent über alle Systemkomponenten hinweg umgesetzt werden.

4.3 Authentifizierung und Autorisierung

Die Authentifizierung erfolgt über mehrere, voneinander getrennte Ebenen:

- **Discord- OAuth2** für Nutzerinteraktionen innerhalb der Plattform (unverändert aus dem Vorprojekt, vgl. Kap. 3.2.1)
- **Dashboard-Authentifizierung** für den Webzugriff
- **Infrastruktur-Authentifizierung** für administrative Server- und Datenbankzugriffe

Das Backend fungiert dabei als zentrale Autorisierungsinstanz und verwaltet sämtliche Rollen- und Rechteinformationen. Es prüft eingehende Anfragen unabhängig von der Plattform, um konsistente Zugriffsbeschränkungen sicherzustellen.

Die Authentifizierung unterscheidet zwischen Personen und automatisierten Dienstkonten. Menschliche Zugriffe erfolgen über eine interaktive Anmeldung und werden bei sensiblen Operationen durch eine Multi-Faktor-Authentifizierung ergänzt. Damit wird insbesondere der Zugriff auf Infrastruktur und Datenbanken gegen unautorisierte Übernahmen geschützt (UC1 , UC3 , NFR4).

4.3.1 Dienstkonten und automatisierte Zugriffe

Im Gegensatz zu Personen greifen automatisierte Systemkomponenten (z. B. der Discord-Bot, das Backend oder der CI/CD-Prozess) über sogenannte Dienstkonten auf Systemressourcen zu. Diese Konten sind speziell für maschinelle Abläufe vorgesehen, arbeiten mit sicheren Tokens oder Secrets und unterliegen restriktiven Rechten nach dem Prinzip Least Privilege. Aktionen werden nachvollziehbar protokolliert (Bezug: NFR5) und Secrets werden sicher in der Pipeline verwaltet.

Bereits im Vorprojekt war der Discord-Bot als solches Dienstkonto umgesetzt. Er lief in einem eigenen Container, authentifizierte sich über ein Bot-Token aus Umgebungsvariablen (.env) und kommunizierte über die Discord-API mit dem System. Da der Bot umfängliche Berechtigungen benötigt, wird er weiterhin Administratorrechte im Discord-Server erhalten. Diese Ausnahme wird bewusst akzeptiert, da sie für die technische Funktionsfähigkeit des Systems erforderlich ist.

Die Trennung zwischen menschlichen Admins (mit MFA-geschütztem Zugriff) und automatisierten Systemprozessen bleibt dennoch gewahrt. Dienstkonten werden systemweit durch Role-Based Access Control (nachfolgend RBAC genannt) begrenzt und besitzen nur die für ihren Anwendungsfall notwendigen Berechtigungen. Dadurch wird das Risiko einer Kompromittierung oder eines Missbrauchs erheblich reduziert.

4.3.2 Absicherung des Datenbankzugriffs

Die Datenbank (nachfolgend DB genannt) stellt das sicherheitskritischste Element des Systems dar, da sie sämtliche personenbezogenen und transaktionsbezogenen Daten des Vereins enthält. Entsprechend muss der Zugriff sowohl technisch als auch organisatorisch auf mehreren Ebenen geschützt werden. Die Absicherung umfasst einerseits systeminterne Verbindungen zwischen Backend, Bot und DB, andererseits den administrativen Zugang für Personen mit erweiterten Berechtigungen (Admins).

Im Rahmen der Konzeptentwicklung wurden drei Ansätze zur MFA des DB-Zugriffs untersucht. Hintergrund ist, dass MongoDB selbst keinen nativen Zwei-Faktor-Login unterstützt. Der zweite Faktor muss daher vorgelagert, also auf Netzwerkebene oder durch Besitzfaktoren wie Zertifikate, umgesetzt werden.

4.3.2.1 Bewertung und Entscheid

Die Analyse zeigt, dass Netzwerk-MFA den besten Kompromiss zwischen Sicherheit, Wartbarkeit und Implementierbarkeit bietet. Sie schützt den gesamten Zugriffspfad zur Datenbank und kann ohne strukturelle Änderungen an bestehenden Komponenten eingeführt werden. In der bestehenden OST-Serverumgebung (virtuelle Linux-Server mit SSH-Zugang) lässt sich diese Methode schnell, kostengünstig und mit geringem Risiko implementieren. Darüber hinaus ist sie vollständig kompatibel mit der bestehenden Container- und CI/CD-Umgebung (Docker, GitHub Actions).

Die detaillierte Evaluation und der technische Vergleich der einzelnen Varianten (Netzwerk-MFA, X.509-Zertifikate, SSO / OIDC) sind im Anhang dokumentiert. (vgl. Anhang Abschnitt 14.1)

4.3.2.2 Übertragbarkeit auf andere Vereine

Das vorgestellte Sicherheitskonzept berücksichtigt, dass ClansCore auch von Vereinen betrieben wird, die ihre Infrastruktur eigenständig hosten. Viele kleinere Organisationen verfügen über keinen dedizierten IT-Betrieb und müssen Sicherheit daher mit möglichst geringen Kosten und administrativem Aufwand gewährleisten. Für solche Vereine stellt das Modell der Netzwerk-MFA eine besonders praktikable Lösung dar. Es erfordert weder teure Identity-Management-Lösungen noch komplexe Zertifikatsverwaltungs-Systeme und kann mit frei verfügbaren Open-Source-Werkzeugen (z. B. OpenVPN[1]) umgesetzt werden. Dadurch bleibt das Sicherheitsniveau hoch, ohne dass der Betrieb zusätzliche finanzielle oder organisatorische Hürden verursacht. Diese Skalierbarkeit ist zentral, um den Einsatz von ClansCore auch in kleineren, ehrenamtlich geführten Vereinen langfristig sicherzustellen (UC10 , NFR4).

4.3.3 Dashboard-Login

Das Admin-Dashboard verfügt über eine eigenständige Authentifizierungsebene, die eine sichere Verwaltung von Vereinsdaten unabhängig von Discord ermöglicht. Die Benutzerverwaltung erfolgt zentral im Backend, welches Authentifizierung und Autorisierung des Web-Frontends übernimmt.

Nach erfolgreicher Anmeldung erhalten Nutzende eine kurzlebige Sitzungsberechtigung, die an ihre Rolle (Admin, Vorstand, Mitglied, Gast) gebunden ist. Rollenänderungen oder sicherheitsrelevante Ereignisse führen zum sofortigen Widerruf aktiver Sitzungen.

Die Berechtigungsprüfung erfolgt ausschliesslich serverseitig über rollenbasierte Zugriffskontrolle (RBAC). Alle Sitzungen und Aktionen werden im Audit-Log erfasst, um Nachvollziehbarkeit und Integrität sicherzustellen (NFR5). Für privilegierte Konten (z. B. Vorstand) ist eine optionale MFA vorgesehen (NFR4). Das Dashboard unterstützt zudem sichere Passwortverwaltung und Wiederherstellung. Technische Details zu den Sicherheitsmechanismen sind im Anhang dokumentiert. (vgl. Anhang Abschnitt 14.2)

Diese Umsetzung kombiniert Benutzerfreundlichkeit mit hoher Sicherheit und folgt den Prinzipien Least Privilege, Defense in Depth und Accountability.

4.4 Daten- und Schnittstellensicherheit

Die Kommunikation zwischen den Systemkomponenten erfolgt ausschliesslich über gesicherte Kanäle:

Verbindung	Technologie	Absicherung
Dashboard ↔ Backend	REST-API / HTTPS	TLS 1.3 + Token-basierte Auth (JWT-basierte Sitzung)
Backend ↔ Discord-Bot	WebSocket Event-Bridge	Bot-Token + Signaturprüfung
Backend ↔ MongoDB	TCP / TLS	SCRAM-SHA-256 + RBAC + Netzwerk-MFA
Backend ↔ Google API	OAuth2 Client-Secrets	Zugriffstoken mit Scopes

Tab. 35: Beschreibung der abgesicherten Kanäle in ClansCore

[1] https://openvpn.net/

4.4.1 Eingabevalidierung

Alle externen Eingaben (REST, WebSocket, Formular) werden serverseitig validiert und sanitisiert (Injection-Schutz). Fehler werden kontrolliert behandelt und geloggt, ohne sensitive Details preiszugeben (Bezug: NFR4 , NFR5).

4.4.2 Synchronisierung und Konfliktlösung

Da ClansCore sowohl über den Discord-Bot als auch über das Admin-Dashboard oder anderen potentiellen Plattformen genutzt wird, können Änderungen an denselben Datensätzen parallel erfolgen. Beispielsweise kann ein Vorstandsmitglied ein Mitgliederprofil im Dashboard anpassen, während der Bot gleichzeitig eine Punktebuchung durchführt. Um daraus entstehende Datenkonflikte zu vermeiden, wird im Backend eine optimistische Nebenläufigkeitskontrolle (Optimistic Concurrency Control, OCC) angewendet. Eine detaillierte Beschreibung des Transaktions- und Konfliktlösungsverfahrens findet sich im Anhang. (vgl. Anhang Abschnitt 14.3)

Dieses Verfahren stellt sicher, dass Datenintegrität und Nachvollziehbarkeit auch bei paralleler Nutzung durch mehrere Plattformen gewährleistet bleiben. Es basiert auf anerkannten Architekturmustern zur Transaktions- und Konsistenzsicherung in verteilten Systemen (Martin, 2003) .

4.5 Datenschutz und DSGVO / DSG-Umsetzung

Die im Vorprojekt formulierten Datenschutzrichtlinien (vgl. Vorprojekt, Kap. 3.3 - 3.5) gelten unverändert.

Neu können Nutzende direkt im Dashboard ihre gespeicherten Informationen einsehen, die Einwilligung zur Datenspeicherung widerrufen oder eine Datenlöschung anfordern. Das Hosting erfolgt auf Servern in der Schweiz ohne US-Rechtsbezug, wodurch die Datenhoheit gewahrt bleibt und der Zugriff durch ausländische Behörden (z. B. über den CLOUD Act) weitgehend ausgeschlossen werden kann (U.S.-Department-of-Justice, 2019) . Die Verarbeitung personenbezogener Daten richtet sich nach den Grundsätzen der DSGVO (Europäische-Union, 2016) , welche Privacy by Design and by Default voraussetzen, sowie dem schweizerischen Datenschutzgesetz (Schweizerische-Eidgenossenschaft, 2020) . (Bezug: NFR4)

4.6 Server- und Deployment-Sicherheit

Das Backend wird in einer containerisierten Umgebung betrieben, die nach dem Prinzip Defense in Depth aufgebaut ist. Ziel ist eine mehrschichtige Absicherung, bei der einzelne Schutzebenen unabhängig voneinander wirken. (NIST, 2020)

Weitere technische Details zur Container-Isolation, Namespaces, Capability-Drops und Netzwerksegmentierung sind im Anhang erläutert. (vgl. Anhang Abschnitt 14.4)

Durch die Kombination von Container-Isolation, Netzwerksegmentierung und Multi-Faktor-Authentifizierung entsteht eine robuste, mehrschichtige Sicherheitsarchitektur, welche die Vertraulichkeit, Integrität und grundlegende Verfügbarkeit des Systems sicherstellt. Eine umfassende Gewährleistung der Verfügbarkeit würde jedoch zusätzliche Massnahmen erfordern, wie beispielsweise eine Systemverteilung (Distribution), Lastverteilung (Load Balancing) oder automatisierte Failover-Mechanismen. Diese Aspekte gehen über den Rahmen der vorliegenden Arbeit hinaus und werden daher nicht weiter berücksichtigt. (Bezug: NFR4)

4.7 Manipulations- und Integritätsschutz

Wie in der Vorarbeit beschrieben (vgl. Vorprojekt, Kap. 3.3) werden alle Punkte- und Event-Transaktionen manipulationssicher in der Datenbank protokolliert. Neu ergänzt sind:

- **Transaktionale Speicherung** mittels atomarer Operationen wie MongoDB Sessions (MongoDB-Contributors, 2025a) .
- **Audit-Log** aller Änderungen, insbesondere von Gamification-Daten, sichtbar für den Vorstand.
- Optionaler **Hash-Check** für Integritätsprüfung einzelner Datensätze.

Damit bleibt jede Änderung rückverfolgbar und das Vertrauen in die Datenbasis langfristig erhalten. (Bezug: NFR5)

4.8 Risikoanalyse und Massnahmen

Die zentrale Sicherheitsrisiken umfassen unautorisierte Zugriffe, Datenlecks, Social-Engineering-Angriffe und Datenverlust. Für jedes Risiko existieren die folgenden präventiven und reaktiven Massnahmen.

Risiko	Auswirkungsgrad	Massnahme
Datenleck durch API	Hoch	HTTPS, Token-Auth, Input-Validation (NFR4 , NFR5)
Unautorisierter DB-Zugriff	Hoch	Netzwerk-MFA + SCRAM-SHA-256 + RBAC (NFR4 , NFR5)
Social Engineering	Mittel	Schulung, Prinzip der minimalen Rechte (UC3)
Datenverlust	Hoch	Regelmässige Backups + Restore-Tests

Tab. 36: Einschätzung der Risiken und deren Massnahmen

4.9 Monitoring und Wartung

Ein konsistentes Logging-Konzept dokumentiert sicherheitsrelevante Ereignisse (Auth-Flows, Rollenänderungen, fehlgeschlagene Zugriffe, Konfliktauflösungen). Für schwerwiegende Vorkommnisse sind Benachrichtigungen an einen Discord-Admin-Kanal und das Admin-Dashboard vorgesehen. Wiederkehrende Sicherheitsüberprüfungen orientieren sich an der OWASP-Top-10-Checkliste (OWASP-Contributors, 2021) . Konfigurations- und Rechteprüfungen erfolgen regelmässig. (Bezug: NFR5)

Diese Massnahmen schaffen einen kontinuierlichen Überblick über die Systemintegrität und ermöglichen ein rasches Eingreifen bei Sicherheitsvorfällen.

4.10 Abgrenzung des Sicherheitskonzeptes

Das vorliegende Sicherheitskonzept fokussiert auf zentrale sicherheitsrelevante Anforderungen, die im Rahmen dieser Arbeit spezifiziert, implementiert und überprüft werden können. Weitere Massnahmen, wie erweiterte Schutzmechanismen für API-Rate-Limiting, Security-Header, Key-Rotation oder detaillierte Frontend-Härtung, werden als Best Practices betrachtet, jedoch nicht explizit dokumentiert oder umgesetzt, da deren Implementierung stark von der jeweiligen Hosting- und Vereinsumgebung abhängt.

Diese Themen werden im Projekt dann adressiert, wenn sie technisch erforderlich sind oder sich aus zukünftigen Erweiterungen ergeben. Die bewusste Fokussierung auf Authentifizierung, Zugriffskontrolle, Datenintegrität und Datenschutz stellt sicher, dass die sicherheitskritischen Kernanforderungen (NFR4 , NFR5) erfüllt und überprüfbar bleiben, ohne die Arbeit mit Randaspekten zu überfrachten.

5 Systemanalyse, Architektur und Wireframes

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. Zum Schluss kommen die Wireframes mit den geplanten Ansichten im Dashboard.

5.1 Domain Analyse

Die Domain-Analyse baut auf dem im Vorprojekt entwickelten Vereins-Bot auf, der bereits zentrale Verwaltungs- und Gamification-Funktionen vereinte. Der Bot unterstützte die Mitgliederaufnahme, Rollenverwaltung sowie die Organisation von Aufgaben und Events. Gleichzeitig förderte er durch ein transparentes Punktesystem Motivation und Engagement im Vereinsalltag. (vgl. Vorprojekt, Kap. 5.1)

Im Zentrum steht weiterhin die Person, die als Vereinsmitglied mit unterschiedlichen Rollen, beispielsweise als Vorstandsmitglied, mit dem System interagiert. Bekannte Entitäten wie Aufgabe, Event, Transaktion, Belohnung und Rangliste bilden den Kern des Vereinsworkflows und des ursprünglichen Gamification-Systems. Mitglieder sammeln durch Aktivitäten Punkte, die in Belohnungen eingelöst werden können.

Mit ClansCore wird diese Grundlage zu einem plattformunabhängigen Vereinsmanagement-System weiterentwickelt. Ein erweitertes Account-Konzept ermöglicht die Zuordnung von Personen zu plattformspezifischen Accounts, ohne die Domänenlogik an eine konkrete Plattform zu binden. Ergänzend ermöglichen zeitlich begrenzte Punkte sowie Erinnerungsfunktionen für Events und periodische Zahlungen die Umsetzung von Erkenntnissen aus dem erweiterten Gamification- und Motivationskonzept (vgl. Abschnitt 3).

Die Domäne ist so gestaltet, dass Verwaltungsfunktionen, Gamification-Mechanismen und externe Integrationen klar voneinander getrennt sind. Dadurch entsteht ein konsistentes, erweiterbares Domänenmodell, das die Grundlage für eine transparente und nachhaltige Vereinsorganisation bildet.

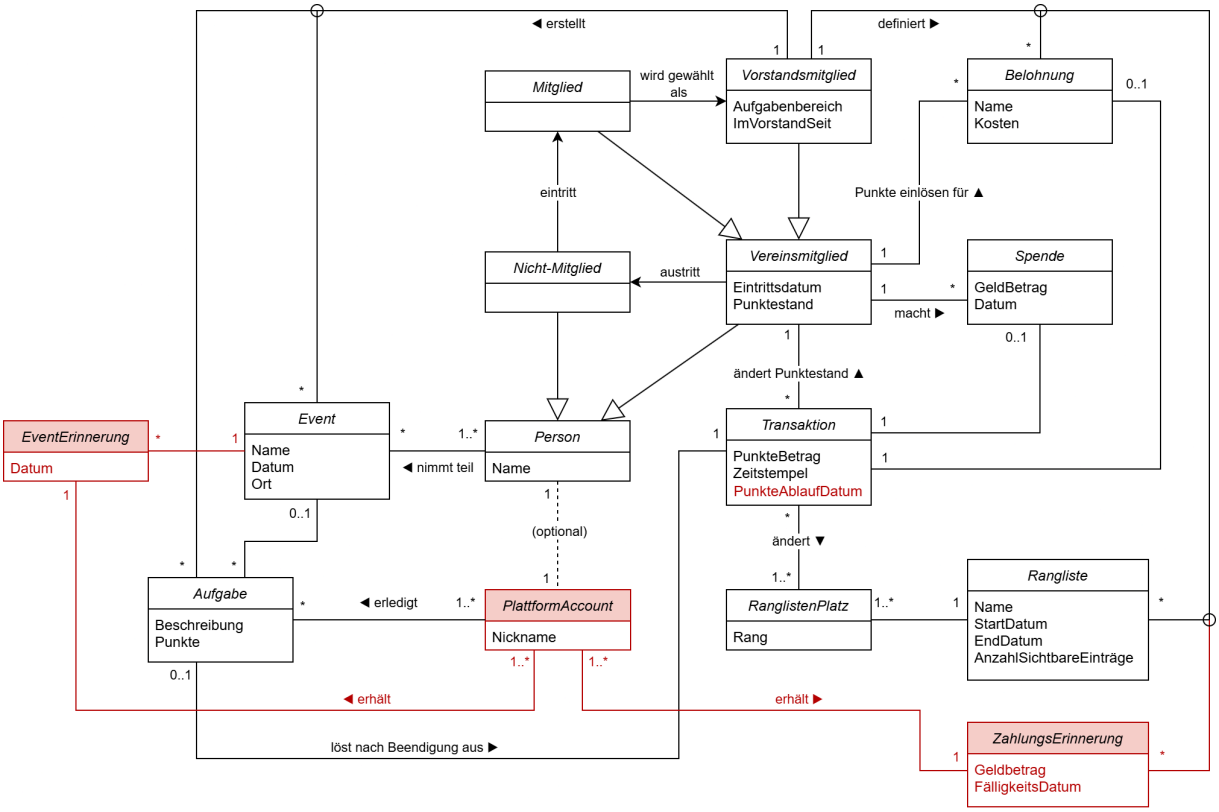


Abb. 6: Domain Model

5.2 C4-Modell

Die Architektur von ClansCore wurde anhand des C4-Modells beschrieben, um die Struktur des Systems auf unterschiedlichen Abstraktionsebenen nachvollziehbar darzustellen. Das Modell ermöglicht eine klare Trennung zwischen Systemkontext, internen Containern und deren Komponenten.

Bereits im Vorprojekt wurde ein modularer Discord-Bot entwickelt. Die konsequente Trennung der Verantwortlichkeiten sowie die Entkopplung der einzelnen Systemteile konnten dort jedoch noch nicht vollständig umgesetzt werden. Die vorliegende Arbeit greift diese Ansätze auf und führt sie zu einer klar strukturierten Zielarchitektur weiter, welche zukünftige Erweiterungen und zusätzliche Integrationen unterstützt.

5.2.1 System Context Diagramm

ClansCore stellt das zentrale Softwaresystem zur Verwaltung eines Gaming-Vereins dar. Es bündelt vereinsrelevante Funktionen wie Mitgliederverwaltung, Event-Organisation und Gamification und stellt diese über definierte Schnittstellen verschiedenen externen Akteuren und Plattformen zur Verfügung.

Die primären Nutzenden des Systems sind Vereinsmitglieder sowie Vorstandsmitglieder bzw. Admins. Vereinsmitglieder interagieren hauptsächlich über die Discord-Plattform mit dem System, während Vorstandsmitglieder zusätzlich über ein webbasiertes Dashboard administrative Aufgaben durchführen.

Neben den menschlichen Akteuren ist ClansCore an mehrere externe Systeme angebunden. Die Discord-Plattform dient als zentrale Kommunikationsschnittstelle für den Vereinsalltag. Ein Webbrowser ermöglicht den Zugriff auf das Admin-Dashboard. Zur Verwaltung und Synchronisation von Vereinsterminen ist zudem ein externer Kalenderdienst angebunden, welcher im aktuellen System durch Google Calendar realisiert ist.

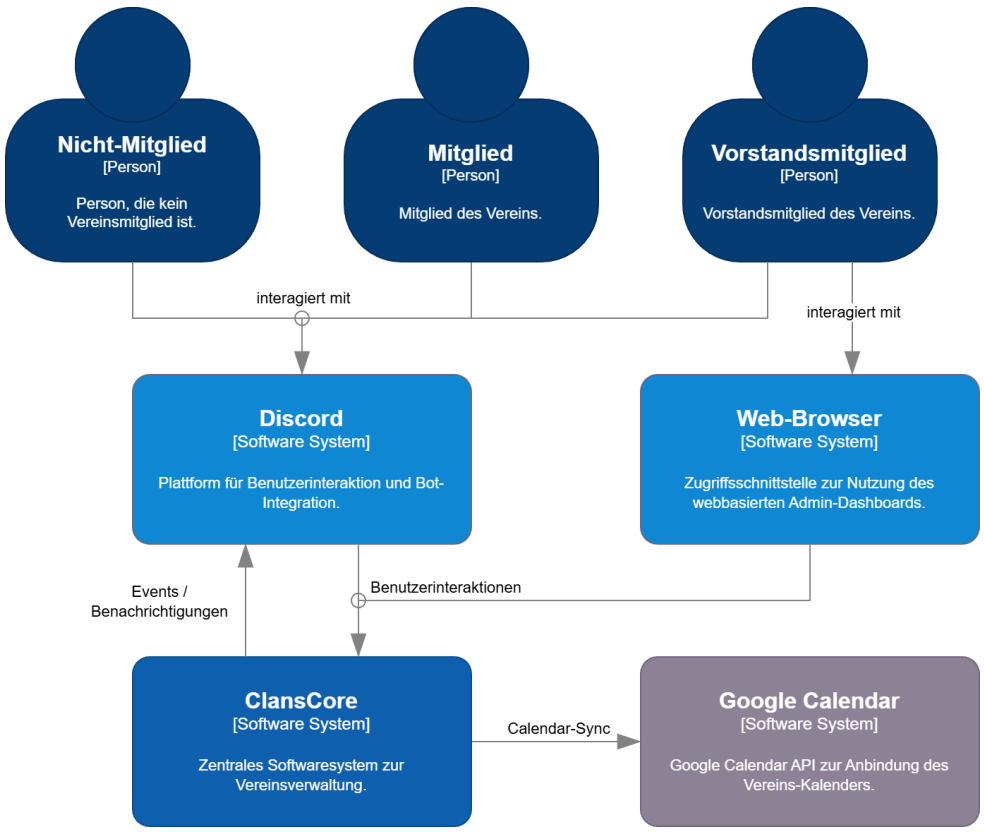


Abb. 7: System Context Diagramm von ClansCore

5.2.2 Container Diagramm

Das Container Diagramm beschreibt die Aufteilung des ClansCore-Systems in eigenständige Softwareeinheiten und deren Kommunikationsbeziehungen. Auf dieser Ebene werden die zentralen technischen Container des Systems dargestellt.

ClansCore besteht aus einer zentralen Backend-API, einem Discord-Bot, einem webbasierten Admin-Dashboard sowie einer MongoDB-Datenbank.

Die ClansCore-API bildet das Kernsystem und kapselt die Geschäftslogik. Sie stellt eine REST-basierte Schnittstelle für den Discord-Bot und das Admin-Dashboard bereit, übernimmt die Authentifizierung und Autorisierung von Anfragen und fungiert als Integrationspunkt für externe Dienste, insbesondere dem Kalenderdienst.

Der Discord-Bot realisiert die Interaktion mit der Discord-Plattform und stellt benutzerseitige Funktionen wie Slash-Commands und automatisierte Abläufe bereit. Die Kommunikation mit der API erfolgt über gesicherte REST-Schnittstellen. Ereignisbasierte Änderungen werden zusätzlich über Webhooks empfangen.

Das Admin-Dashboard dient als grafische Benutzeroberfläche für administrative Aufgaben und greift über HTTPS auf die ClansCore-API zu. Eigene Geschäftslogik ist darin nicht enthalten.

Die MongoDB-Datenbank übernimmt die persistente Speicherung aller systemrelevanten Daten und ist über die API angebunden, wodurch eine klare Trennung zwischen Applikationslogik und Datenhaltung gewährleistet wird.

Externe Dienste wie der Google Calendar sind zentral mit der API verbunden, was eine lose Kopplung der übrigen Container ermöglicht und die Erweiterbarkeit des Systems unterstützt.

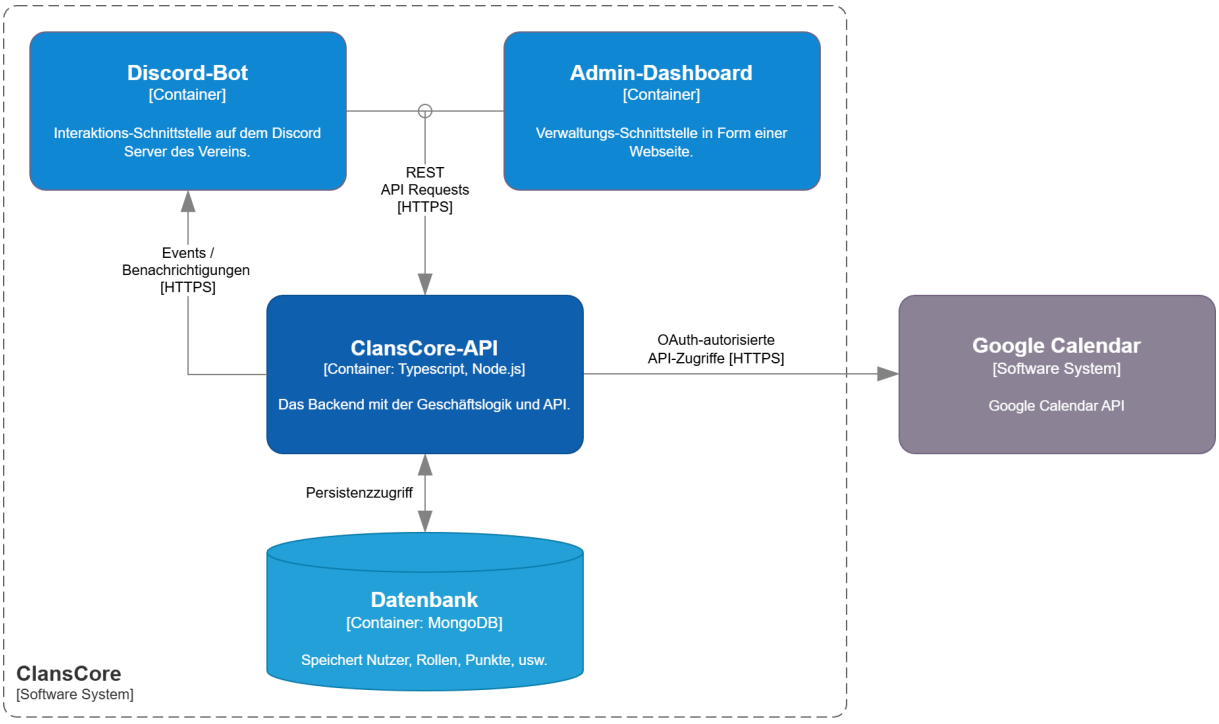


Abb. 8: Container Diagramm von ClansCore

5.2.3 Component Diagramm

Das Component Diagramm beschreibt die interne Struktur des Containers ClansCore-API und zeigt dessen zentrale Softwarekomponenten sowie deren Abhängigkeiten. Im Gegensatz zum Container Diagramm liegt der Fokus hier auf der Aufteilung der API in fachlich und technisch abgegrenzte Bausteine.

Im Vorprojekt waren Discord-spezifische Funktionen direkt in die Backend-Applikation integriert. Der Discord-Client und die zugehörige Command-Logik liefen im selben Prozess wie der Webserver und griffen ohne klar definierte Schnittstellen auf die Geschäftslogik zu (vgl. Vorprojekt, Kap. 5.2.5). Diese enge Kopplung vereinfachte die initiale Entwicklung, führte jedoch zu Einschränkungen hinsichtlich Wartbarkeit, Testbarkeit und Erweiterbarkeit.

In der überarbeiteten Architektur ist die ClansCore-API klar strukturiert und von plattformspezifischer Logik getrennt. Der Webserver fungiert als zentraler Einstiegspunkt und übernimmt die HTTP-basierte Kommunikation über eine REST-Schnittstelle, die Authentifizierung sowie die Weiterleitung von Anfragen an die fachlichen Komponenten.

Die fachliche Logik ist in die Komponenten User, Event und Gamification aufgeteilt, welche jeweils klar abgegrenzte Verantwortlichkeiten besitzen. Externe Integrationen, insbesondere die Synchronisation mit Kalenderdiensten, sind in der Calendar-Komponente gekapselt. Ereignisbasierte Benachrichtigungen werden über eine dedizierte Notification-Komponente verarbeitet, welche unter anderem Webhooks an den Discord-Bot auslöst.

Der Zugriff auf persistente Daten erfolgt ausschliesslich über spezialisierte Database-Services, welche die Datenhaltung abstrahieren und die Trennung zwischen Geschäftslogik und Infrastruktur sicherstellen. Die eigentliche MongoDB-Datenbank ist nicht Teil des Containers und wird lediglich als externer Persistenzdienst genutzt.

Gemeinsam genutzte Typdefinitionen, DTOs und Validierungslogik sind in einem separaten Shared Package zusammengefasst, welches von mehreren Komponenten verwendet wird, jedoch keine Laufzeitabhängigkeiten erzeugt.

Durch diese klare komponentenbasierte Struktur bleibt die ClansCore-API übersichtlich strukturiert, gut testbar und unabhängig von konkreten Client-Plattformen wie Discord oder dem Admin-Dashboard.

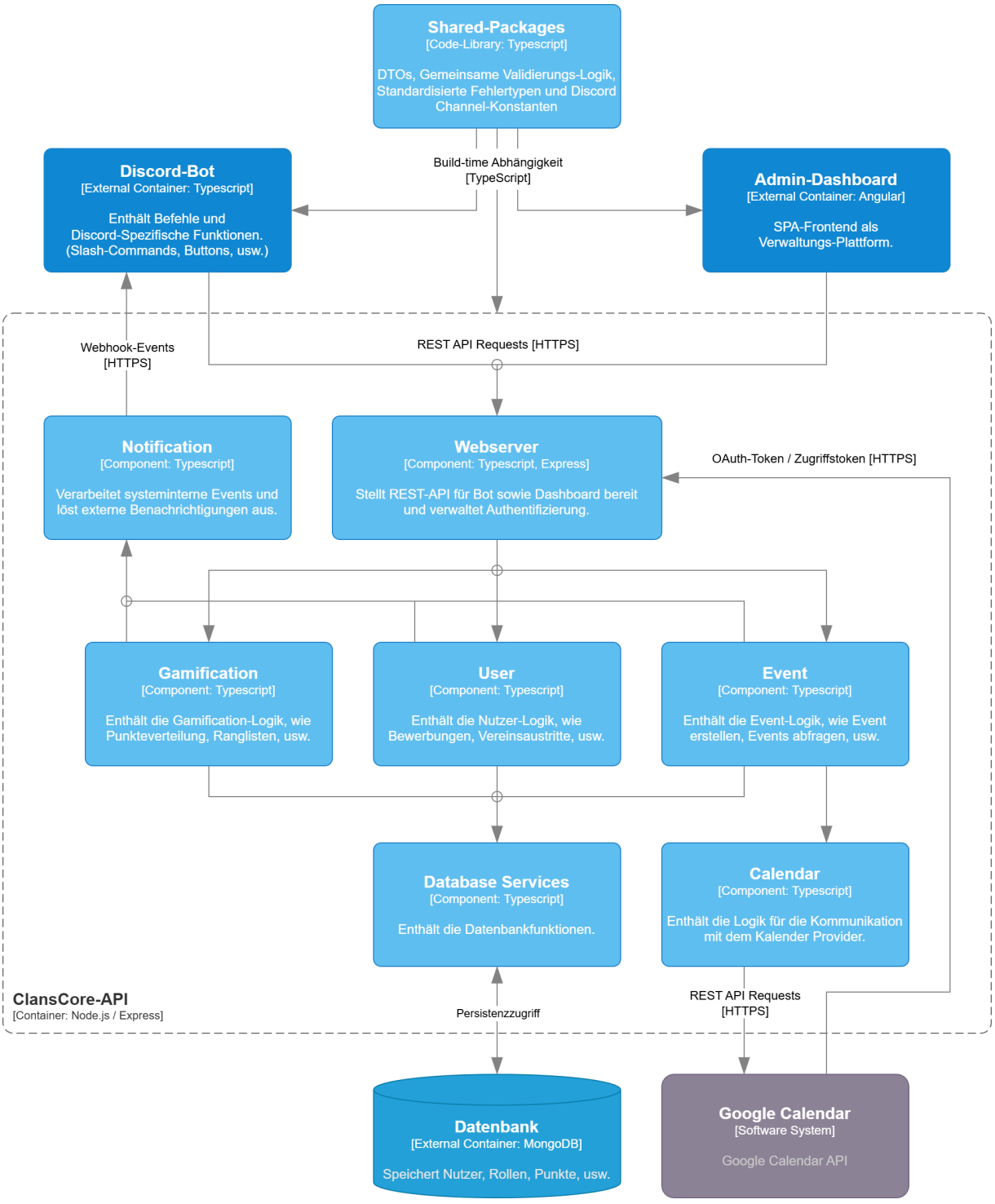


Abb. 9: Component Diagramm der ClansCore-API

5.3 Wireframes

Zur Skizzierung des Aufbaus und der grundlegenden Struktur des webbasierten Dashboards wurde das Wireframe-Tool Balsamiq[2] verwendet. Die Wireframes wurden unter Bezug auf bestehende Angular-Webseiten (madewithangular-Contributors, 2025) entworfen und mit Blick auf den Einsatz von Angular-Material-Komponenten konzipiert.

Die dargestellten Wireframes repräsentieren sowohl die aktuell implementierten Ansichten des Admin-Dashboards als auch konzeptionell vorgesehene Erweiterungen für zukünftige Nutzergruppen. In der vorliegenden Version von ClansCore ist das Dashboard ausschliesslich für administrative Nutzer (Vorstand) vorgesehen. Ansichten für Mitglieder oder Nicht-Mitglieder dienen der konzeptionellen Einordnung und wurden im Rahmen dieser Arbeit nicht vollständig implementiert.

Nicht für alle Seiten wurden Wireframes erstellt, da diese primär dazu dienen, einen strukturellen Überblick über Navigation, Funktionalität und Rollenabgrenzung des Dashboards zu vermitteln.

5.3.1 Vereins-Verwaltung

Die nachfolgenden Wireframes zeigen sowohl aktuell umgesetzte Administrationsansichten als auch konzeptionelle Benutzeransichten für zukünftige Erweiterungen. In der aktuellen Implementierung sind ausschliesslich die Vorstandsfunktionen Bestandteil des Dashboards.

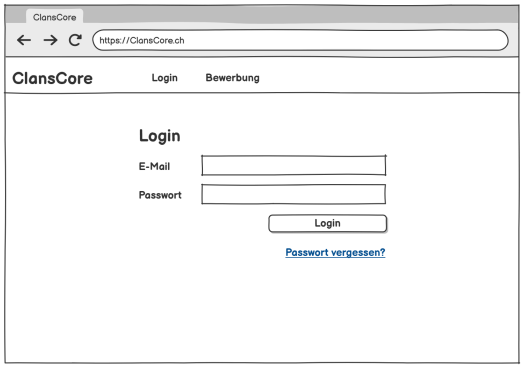


Abb. 10: Login

Login

Um die Kernfunktionen von ClansCore nutzen zu können, ist eine Authentifizierung erforderlich. Diese erfolgt auf der Login-Seite durch die Eingabe der E-Mail-Adresse sowie des zugehörigen Passworts (UC1). Für den Fall, dass eine Person ihr Passwort vergessen hat, besteht über einen entsprechenden Link die Möglichkeit, den Passwort-Zurücksetzungsprozess zu starten und ein neues Passwort anzufordern.

Nicht-Mitglied

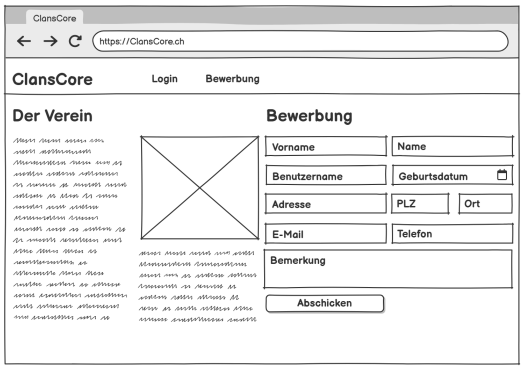


Abb. 11: Bewerbungsformular

Bewerbung

Die Bewerbungsseite ist zweispaltig aufgebaut. Auf der linken Seite werden allgemeine Informationen über den Verein präsentiert, während auf der rechten Seite ein Formular zur Einreichung einer Beitrittsbewerbung zur Verfügung steht. (UC2)

[2] https://balsamiq.com/

Mitglied und Vorstand

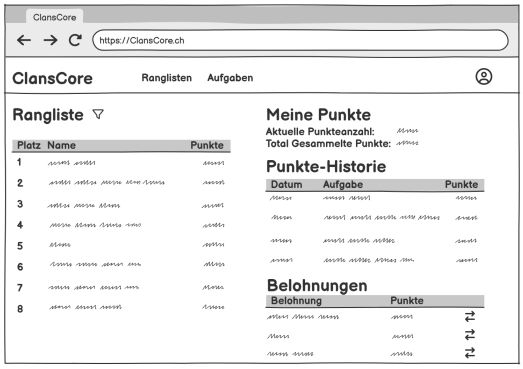


Abb. 12: Ranglisten-Seite

Ranglisten

Auf der Ranglisten-Seite befindet sich auf der linken Seite eine filterbare Rangliste (Saisonal/Gesamt), die einen Überblick über die Platzierungen der Nutzenden bietet. Auf der rechten Seite werden die eigenen Punkte, deren Historie und aktuelle Belohnungen angezeigt. (UC6)

Abb. 13: Aufgaben-Seite

Aufgaben

Auf der Aufgaben-Seite werden alle verfügbaren Aufgaben angezeigt (UC6). Diese lassen sich nach verschiedenen Kriterien filtern (Anmeldestatus, Datum, Punkteanzahl). Solange eine Aufgabe noch nicht abgeschlossen ist, besteht die Möglichkeit, sich dafür anzumelden. (UC7) Vorstandsmitglieder haben zusätzlich die Berechtigung, Aufgaben zu bearbeiten und hinzuzufügen. (UC5)

Vorstand

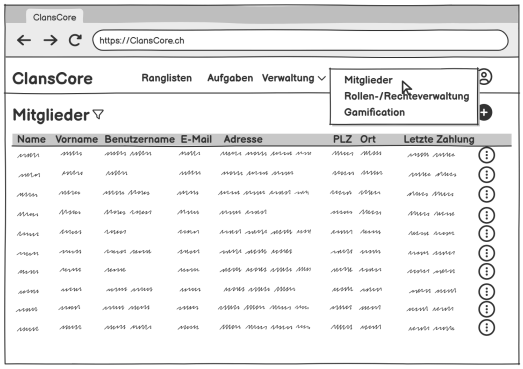


Abb. 14: Mitglieder-Seite

Mitglieder

Zugriff auf die Mitglieder-Seite haben ausschliesslich Vorstandsmitglieder und Admins. Es können bestehende Mitglieder gefiltert (Name, Vorname, Benutzername), bearbeitet, hinzugefügt oder gelöscht werden. (UC3)

5.3.2 Gamification-Verwaltung

In der Gamification-Verwaltung werden alle Parameter welche für die Punkteberechnung von Aufgaben benötigt werden erfasst. Da die Gamification-Verwaltung ein bestehendes Excel-Tool aus dem Vorprojekt (vgl. Vorprojekt, Kap. 4.8) ablöst, wurde das Design gezielt an dessen Layout und Bedienlogik angelehnt. Dadurch soll der Übergang zur neuen Anwendung für die Nutzenden erleichtert werden.

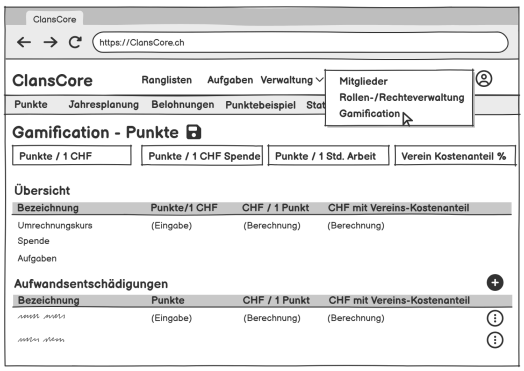


Abb. 15: Punktedefinitions-Seite

Punktedefinition
Auf der Punkte-Seite werden die zentralen Parameter des Gamification-Systems vom Vorstand definiert. Dazu gehören unter anderem der monetäre Wert eines Punktes, die Punktebewertung einer Arbeitsstunde sowie der Punktewert einer Spende. Zusätzlich kann der Kostenanteil des Vereins festgelegt werden. Darüber hinaus besteht die Möglichkeit, neue Aufwandsentschädigungen anzulegen, bestehende zu bearbeiten oder zu löschen. (UC4)

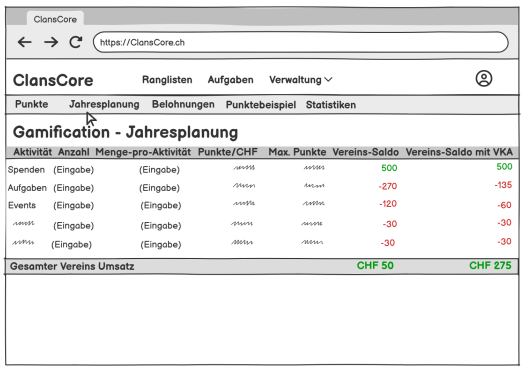


Abb. 16: Jahresplanung-Seite

Jahresplanung
Auf der Jahresplanung-Seite können die jährlichen Kosten für den Verein simuliert werden. Zur Berechnung werden Parameter von der Punkte-Seite benötigt. Dadurch lässt sich abschätzen, wie sich unterschiedliche Szenarien auf den jährlichen Gesamtumsatz des Vereins auswirken. (UC4)

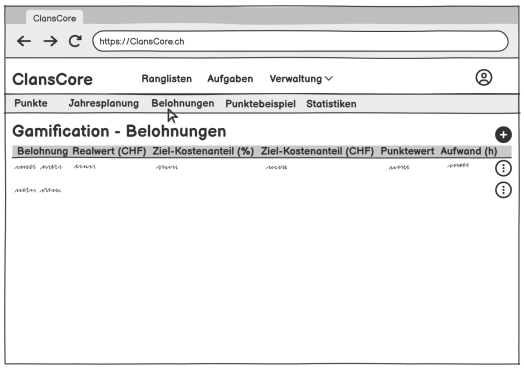


Abb. 17: Belohnung-Seite

Belohnungen
Es können Belohnungen bearbeitet, hinzugefügt oder gelöscht werden. Zur Berechnung des Punktewertes und dem Aufwand werden Parameter von der Punkte-Seite benötigt. (UC4)

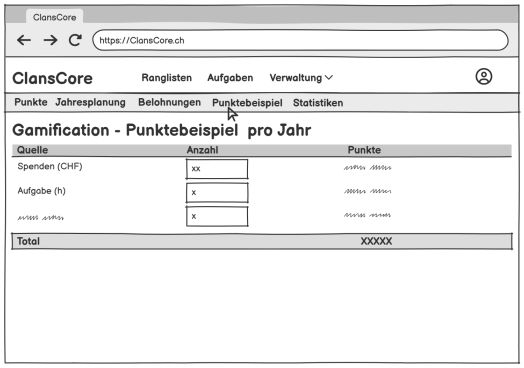


Abb. 18: Punktebeispiel-Seite

Punktebeispiel
Auf der Punktebeispiel-Seite kann die Punktevergabe für ein Mitglied simuliert werden. Dabei werden verschiedene Aktivitäten, wie Spenden, Event-Teilnahmen oder Aufgaben, berücksichtigt, um die resultierende Gesamtpunktzahl zu berechnen und deren Auswirkungen im Punktesystem zu veranschaulichen. (UC4)

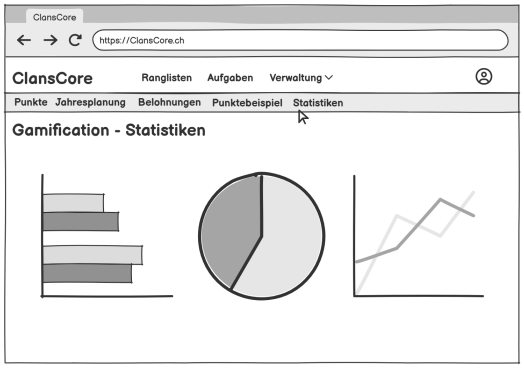


Abb. 19: Statistiken-Seite

Statistiken
Auf der Statistik-Seite können Auswertungen zu Events und Mitgliedern angezeigt werden. Diese bieten einen Überblick über relevante Kennzahlen und zukünftige Trends. Diese unterstützen den Vorstand bei der Analyse der Vereinsaktivitäten und dienen als Planungshilfe.

5.4 Konkrete Umsetzung der neuen Architektur

6 Qualitätssicherung

Die Qualitätssicherung stellt sicher, dass ClansCore stabil, sicher und nachvollziehbar betrieben werden kann. Dieses Kapitel beschreibt das Testkonzept sowie die eingesetzten Massnahmen zur Sicherstellung der funktionalen und nicht-funktionalen Anforderungen. Ziel ist es, durch strukturierte Tests, Reviews und qualitative Evaluationen eine angemessene Softwarequalität im Rahmen dieser Arbeit zu gewährleisten.

6.1 Testkonzept

Das Testkonzept definiert die Vorgehensweise zur Qualitätssicherung von ClansCore. Es legt fest, welche Testarten durchgeführt werden, wie diese organisiert sind und in welcher Form funktionale sowie nicht-funktionale Anforderungen überprüft werden. Dabei wird zwischen automatisierten Tests, manuellen Systemtests und qualitativen Evaluationsmethoden unterschieden.

6.1.1 Vorgehensweise

Um eine strukturierte und nachvollziehbare Qualitätssicherung sicherzustellen, wird folgendes Vorgehen angewendet:

1. Ableitung der Testziele aus den Functional und Non-Functional Requirements.
2. Festlegung geeigneter Testarten (automatisiert, manuell, qualitativ).
3. Durchführung der Tests gemäss Testart.
4. Dokumentation von Ergebnissen, Abweichungen und Erkenntnissen.
5. Behebung identifizierter Fehler, sofern diese im Rahmen der Arbeit adressiert werden.
6. Wiederholung relevanter Tests nach Anpassungen.

6.1.2 Testziele

Ziel der Qualitätssicherung ist die Überprüfung der folgenden Aspekte:

- Umsetzung aller Use Cases
- Absicherung zentraler Non-Functional Requirements , insbesondere:
  - Usability
  - Maintainability
  - Security
  - Reliability

Nicht alle nicht-funktionalen Anforderungen werden quantitativ getestet, da sich einige Qualitätsmerkmale (z. B. Usability oder Maintainability) primär qualitativ bewerten lassen. Je nach Anforderung erfolgt die Überprüfung entweder durch automatisierte Tests, manuelle Systemtests oder qualitative Evaluationen.

6.1.3 Testumgebung

Die Tests werden in einer produktionsnahen Umgebung durchgeführt.

Client:

Betriebssystem	Windows 11
Browser	Google Chrome, Mozilla Firefox
Plattform	Discord

Tab. 37: Testumgebung Client

Server:

Betriebssystem	Linux Ubuntu 24.04.3 LTS
TypeScript	Version 5.9.3
Node.js	Version 20.19.6
Datenbank	MongoDB 8.2.2

Tab. 38: Testumgebung Server

6.1.4 Unit- und Integrationtests

Die Unit- und Integrationtests des Admin-Dashboards werden mit dem Jasmine-Testing-Framework umgesetzt, welches standardmässig in Angular-Projekten integriert ist. Als Test Runner kommt Karma zum Einsatz, der die automatisierte Ausführung der Tests in einer browserbasierten Testumgebung ermöglicht. (Angular-Contributors, 2025a)

Die Tests werden mithilfe der Angular Testing Utilities (TestBed) implementiert. Dadurch können neben isolierten Unit-Tests auch Integrationstests durchgeführt werden, bei denen das Zusammenspiel einzelner Komponenten überprüft wird.

Für den Discord-Bot werden bestehende Unit- und Integrationtests mit dem Jest-Framework weiterverwendet und an die neue Systemarchitektur angepasst. Dabei lag der Fokus auf der Sicherstellung der korrekten Verarbeitung von Commands sowie der Interaktion mit externen Schnittstellen. Eine Erweiterung des Testumfangs erfolgte im Rahmen dieser Arbeit nicht.

6.1.5 Evaluation durch Simulation

Im Rahmen dieser Arbeit wurden keine klassischen Usability-Tests durchgeführt. Stattdessen ist eine zweiwöchige Simulation eines Vereinsbetriebs vorgesehen, welche nach Abgabe der schriftlichen Arbeit durchgeführt wird. Die Simulation dient insbesondere der qualitativen Überprüfung der Usability- und Reliability-Anforderungen.

Die Simulation dient der praxisnahen Evaluation von ClansCore im Vereinsalltag. Dabei soll untersucht werden, wie das System im realitätsnahen Einsatz genutzt wird und welchen Beitrag es zur Organisation, Interaktion und Motivation innerhalb eines Vereins leisten kann.

Die grobe Planung ist im Anhang zu finden (vgl. „ClansCore-Simulation - Plan.pdf“). Durchführung und Auswertung der Simulation sind nicht Bestandteil dieser Arbeit und werden in einem separaten Dokument im Anhang detailliert beschrieben.

6.2 Ablaufplan

Der Ablaufplan beschreibt die eingesetzten Testarten sowie deren Relevanz in Bezug auf funktionale und nichtfunktionale Anforderungen.

ID	**T1**
Name	Unit-Tests
Beschreibung	Funktionen im Backend sowie die Logik einzelner Angular-Komponenten werden mit Unit-Tests überprüft. Die Tests werden automatisiert über GitHub Actions nach jedem Commit ausgeführt.
Methode	Automatisch
Relevant für	Use Cases, Functional Requirements
Testumgebung	Client und Server
Intervall	Bei jedem Commit

Tab. 39: Ablauf Unit-Tests

ID	T2
Name	Integration-Tests
Beschreibung	Überprüfung des Zusammenspiels zwischen Angular, Backend und Datenbank sowie externer Schnittstellen. Die Tests werden automatisiert nach jedem Commit ausgeführt.
Methode	Automatisch
Relevant für	Use Cases, Reliability
Testumgebung	Client und Server
Intervall	Bei jedem Commit

Tab. 40: Ablauf Integration-Tests

ID	T3
Name	Simulation Vereinsbetrieb
Beschreibung	Die Evaluation des Systems erfolgt im Rahmen einer zweiwöchigen Simulation eines Vereinsbetriebs. Die Planung und Durchführung der Simulation sind im Anhang dokumentiert.
Methode	Simulation
Relevant für	Usability, Reliability
Testumgebung	Produktionsnahe Discord- und Web-Umgebung
Intervall	Nach Abgabe der schriftlichen Arbeit

Tab. 41: Ablauf Simulation Vereinsbetrieb

ID	T4
Name	System-Tests
Beschreibung	Manuelle Überprüfung, ob alle Use Cases sowie ausgewählte Non-Functional Requirements korrekt umgesetzt wurden.
Methode	Manuell
Relevant für	Use Cases, Security, Maintainability
Testumgebung	Client und Server
Intervall	Am Ende

Tab. 42: Ablauf System-Tests

6.3 Code-Qualität

Zur Sicherstellung der Code-Qualität wird das Vier-Augen-Prinzip angewendet. Änderungen am Code werden vor dem Merge in den Main-Branch durch mindestens eine weitere Person überprüft. Weitere Informationen zu Code-Qualität und GitHub können im Abschnitt 11 gefunden werden.

Zusätzlich wird ein Linter eingesetzt, um eine einheitliche Code-Struktur und die Einhaltung definierter Code-Guidelines sicherzustellen. Diese Massnahmen unterstützen insbesondere die Maintainability-Anforderungen des Systems. Weitere Details sind im Code Setup (vgl. Abschnitt 11.5.2) beschrieben.

6.4 Definition of Done

Ein Use Case gilt als abgeschlossen, wenn folgende Kriterien erfüllt sind:

- Die im Use Case beschriebenen Funktionen wurden vollständig umgesetzt.
- Relevante Tests wurden durchgeführt und dokumentiert.
- Zentrale Non-Functional Requirements wurden berücksichtigt oder bewusst begründet ausgeschlossen.
- Die Code Guidelines wurden eingehalten (vgl. Abschnitt 11.5.1).

7 Implementation

Dieses Kapitel beschreibt ausgewählte technische Implementierungsaspekte von ClansCore . Der Fokus liegt auf der konkreten Umsetzung architektonischer Konzepte, der Integration externer Systeme sowie auf sicherheits-, wartbarkeits- und betrieblich relevanten Mechanismen. Konzeptionelle Architekturentscheide wurden in Abschnitt 5 behandelt.

7.1 Datenbank

In diesem Abschnitt werden die zentralen Erweiterungen der Datenbankstruktur sowie ausgewählte technische Aspekte der Datenpersistenz beschrieben, welche für die neuen Funktionalitäten von ClansCore erforderlich waren.

Die Datenbank von ClansCore basiert auf einem im Rahmen des Vorprojektes entwickelten Datenbankmodell, welches für das aktuelle Projekt erweitert und angepasst wurde, um die neuen funktionalen und technischen Anforderungen zu erfüllen (vgl. Vorprojekt, Kap. 7.1).

7.1.1 Erweiterung des Datenbankmodells

Bei der Entität `Person` wurde das Attribut „WebPW“ hinzugefügt, um die Authentifizierung auf dem Dashboard zu ermöglichen (vgl. UC1). Das Passwort wird gehasht und gesalzen gespeichert. Die Entität `Person` ist mit der neu eingeführten Entität `BeitragsZahlung` verknüpft, welche den Zeitpunkt speichert, wann eine Person ihren Mitgliederbeitrag bezahlt hat. Dieser wird in der Entität `MitgliederBeitrag` verwaltet. Zudem besteht eine Verbindung zur Entität `Erinnerung`, in der festgelegt wird, an welchem Datum und für welche Events oder Mitgliederbeiträge eine Erinnerung versendet werden soll (vgl. UC8).

Ein zentraler Bestandteil des Gamification-Systems sind die in der Entität `GamificationParameter` definierten Attribute „PunkteProCHF“ und „PunkteProSpende“. Diese dienen als Grundlage für die Punkteberechnung und Punkteplanung. In der Entität `Transaktion` wurden die neuen Attribute „BetragVerbraucht“ und „PunkteLaufenabAm“ ergänzt, um die geplante Gamification-Erweiterung E4 zu realisieren.

Die Entität `Aufgabentyp`, welche mit der Entität `Aufgabe` verbunden ist, beschreibt die verschiedenen Aufgabentypen und dient der Punkteberechnung (vgl. UC4 , UC5). Zur Differenzierung der Art der Punktevergabe verfügt die Entität `Aufgabentyp` über das Attribut „Entschädigung“. Dieses ist als Enumeration modelliert und kennzeichnet, ob eine Entschädigung stundenbasiert oder einmalig vergeben wird. Die Entität `Jahresplanung` steht in Beziehung zur Entität `Aufgabentyp` und dient der Unterstützung der vereinsinternen Jahresplanung sowie als Kontrollinstrument des Punktesystems. Dabei wird definiert, welche Aufgabentypen in welchem Umfang und mit welcher Anzahl an Teilnehmenden geplant sind.

In orange sind die im Vergleich zum Vorprojekt modifizierten Entitäten eingefärbt.

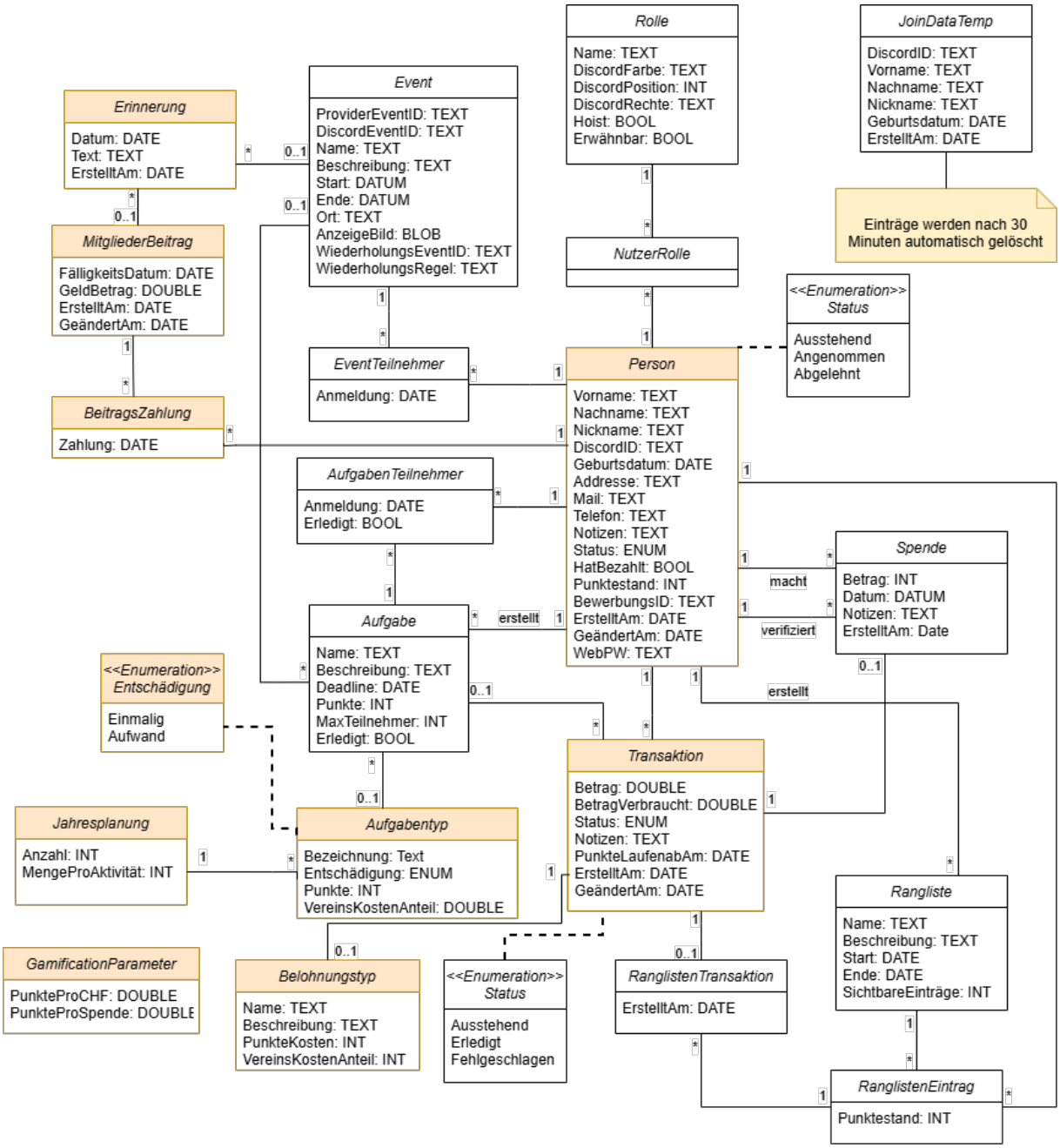


Abb. 20: Das erweiterte logische Datenbankmodell

7.1.2 Datenbank-Transaktionen

Um Datenbankaktionen atomar auszuführen, wurde die Implementierung von Datenbanktransaktionen untersucht. Da Transaktionen in MongoDB ausschliesslich innerhalb eines Replica Sets unterstützt werden, wurde die zuvor verwendete Standalone-Instanz auf eine Replica-Set-Instanz umgestellt.

Damit mehrere Datenbankaktionen konsistent und koordiniert ausgeführt werden können, muss vor deren Ausführung eine Session initialisiert werden. Diese Session wird bei jeder Datenbankoperation übergeben, die Teil der Transaktion ist. Können alle vorgesehenen Operationen erfolgreich durchgeführt werden, werden die Änderungen mittels Commit dauerhaft übernommen. Andernfalls wird die Transaktion abgebrochen und alle Änderungen werden zurückgerollt.

```
const session = await mongoose.startSession();
      session.startTransaction();
      try {
      await Promise.all([
          deleteTaskParticipantsByPerson(person._id, session),
          deleteTransactionsByPerson(person._id, session),
          deleteDonationsByPerson(person._id, session),
          deleteRewardsByPerson(person._id, session),
          removePerson(person._id.toString(), session),
      ]);
      await session.commitTransaction();
      }
      catch (error){
          await session.abortTransaction();
          ...
      }
      finally{
          session.endSession();
      }
```

Listing 1: Ausschnitt aus der Implementierung einer Datenbank-Transaktion beim Löschen einer Person

7.2 Technische Umsetzung des architekturellen Refactorings

Dieses Kapitel fokussiert bewusst auf die technische Umsetzung der im Architekturkapitel definierten Zielarchitektur (vgl. Abschnitt 5.2.3). Die nachfolgende Darstellung der Ausgangslage dient der Einordnung konkreter Refactoring-Massnahmen und ersetzt keine vollständige Architekturbetrachtung.

Das Vorprojekt basierte auf einem Discord-Bot, der bereits erste Ansätze einer Schichtenarchitektur (models, application, infrastructure, platform) aufwies, ohne jedoch eine klare Trennung dieser Schichten durchgängig umzusetzen. Obwohl Komponenten wie das `UserModel` oder `GamificationModel` eine gewisse Abstraktion gegenüber Mongoose ermöglichten, fehlten wesentliche Elemente, die für eine skalierbare, testbare und langfristig erweiterbare Plattform erforderlich gewesen wären. Dazu gehören insbesondere eine eigenständige API-Schicht, eine konsequente Domänenabgrenzung sowie eine wiederverwendbare Shared-Library für verschiedene Applikationen.

Da das Zielsystem aus mehreren logisch getrennten Anwendungen bestehen sollte (Backend / API, Discord-Bot, Admin-Dashboard, Shared-Package), war ein tiefgreifendes Refactoring notwendig. In diesem Kapitel wird die Ausgangslage des Vorprojekts beschrieben, die architektonischen Schwächen analysiert und die daraus motivierte Neugestaltung erläutert.

7.2.1 Analyse der Ausgangsarchitektur im Vorprojekt

Die Architektur des Vorprojekts befand sich in einer Übergangsphase. Es existierte eine Grundstruktur nach fachlichen Domänen (user, gamification, events) sowie erste technische Layer, jedoch fehlten zentrale Eigenschaften einer vollständig modularen Softwarearchitektur.

Merkmal	Beschreibung
Teilweise Schichtenarchitektur	Die Ordnerstruktur unterschied zwischen `application/`, `infrastructure/`, `models/` und `platform/discord/`. Die Trennung war jedoch nicht durchgängig umgesetzt: • Commands griffen direkt auf Application-Models zu • Application-Models fungierten hauptsächlich als Wrapper um Datenbankservices • Präsentations- und Geschäftslogik waren nicht klar getrennt
Indirekte, aber enge Datenbankkopplung	Commands nutzten nicht direkt Mongoose, jedoch Application-Models wie `UserModel` oder `GamificationModel`. Diese stellten: • lediglich leichte Abstraktionen dar • keine eigenständige Businesslogik bereit • eine enge Kopplung an die Datenbankstruktur her Eine dedizierte Businesslogik-Schicht war nicht vorhanden.
Keine Wiederverwendbarkeit über Projektgrenzen	Obwohl Typen, Modelle und Validierungen vorhanden waren, fehlte eine Shared-Library zur gemeinsamen Nutzung in mehreren Applikationen (API, Bot, Dashboard). Dies führte zu: • mehrfach definierten Typen • fehlenden DTO-Strukturen • uneinheitlicher Validierungslogik • redundanten Fehlertypen
Fehlende API-Schicht	Der Bot kommunizierte nicht über HTTP mit dem Backend, sondern griff direkt über Application-Models und DB-Services auf Daten zu. Der einzige vorhandene Endpunkt diente dem Google- OAuth -Callback für die Kalenderanbindung. Dadurch fehlten: • eine Sicherheits- und Validierungsschicht • eine klare Trennung zwischen Bot und Backend • technische Voraussetzungen für ein Admin-Dashboard
Unvollständige Domänenabgrenzung	Obwohl Domänenordner existierten, waren diese nicht isoliert: • Commands importierten mehrere Domänen direkt • keine Aggregates im Sinne von Domain-Driven Design • keine Bounded Contexts • Geschäftsregeln waren auf Commands und DB-Services verteilt

Tab. 43: Beschreibung der architektonischen Merkmale des Vorprojekts

7.2.2 Neue Zielarchitektur

Die neue Architektur folgt Domain-Driven Design (DDD) und Modularitätsprinzipien. Sie besteht aus vier logisch getrennten, aber klar integriert arbeitenden Komponenten:

```
apps/
  clanscore-api/       > Zentrale Backend-API
  dashboard/           > Admin-Dashboard (Angular)
  discord-bot/         > Entkoppelter Discord-Bot (API-Client)
shared/                > DTOs, Validierungen, Fehlertypen, Utilities
```

Diese Struktur ermöglicht erstmals eine saubere Trennung aller fachlichen und technischen Verantwortlichkeiten.

Änderungs-Schritt	Nutzen
Entkopplung des Discord-Bots	Der Discord-Bot kommuniziert nun ausschliesslich über die Backend-API: • keine direkten Datenbankzugriffe • Commands enthalten keine Geschäftslogik • die Kommunikation erfolgt über einen dedizierten API-Client • gemeinsame DTOs gewährleisten Typensicherheit Die Entkopplung ermöglicht separate Deployments und verbessert Sicherheit, Validierung und Wartbarkeit erheblich.
Einführung einer vollständigen Schichtenarchitektur	Das Backend wurde in klar getrennte Schichten überführt: • **Domain Layer**: Geschäftsobjekte und Regeln • **Application Layer**: Orchestrierung von Use Cases • **Infrastructure Layer**: Datenbankzugriffe, externe Dienste • **Presentation Layer**: REST-Endpunkte, Routing, Validierung Diese Struktur schafft klare Verantwortlichkeiten und erlaubt testbare, erweiterbare Use Cases.
API-first Ansatz	Für alle Funktionen existieren nun REST-Endpunkte: • Mitgliederverwaltung • Rollenverwaltung • Events • Kalender • Gamification (Tasks, Rewards, Leaderboards) Dies ermöglicht: • vollständige Dashboard-Anbindung • stabile Bot-Integration • zentrale Validierung, Authentifizierung und Fehlerbehandlung
Einführung der Shared-Library	Die Shared-Library fasst sämtliche gemeinsam genutzten Strukturen zusammen: • DTOs • Validierungen • Fehlerdefinitionen • Domänenkonstanten • Utility-Funktionen Dadurch werden konsistente Typen und Schnittstellen für alle Apps bereitgestellt.

Tab. 44: Beschreibung der zentralen Änderungen der neuen Architektur

7.2.3 Umsetzung und Ergebnis des Refactorings

Die technische Umsetzung des Refactorings erfolgte in mehreren aufeinander abgestimmten Schritten. Zunächst wurde die alte Bot-Architektur vollständig dekonstruiert, indem direkte Modellzugriffe entfernt, eng gekoppelte Commands aufgelöst und über 25 Kommandodateien auf eine ausschliessliche Nutzung der neuen API migriert wurden. Parallel dazu wurde die gesamte API-Schicht neu aufgebaut. Es entstanden zahlreiche dedizierte Controller (u. a. für User, Tasks, Events, Gamification, Rollen und Kalender) sowie eine zentrale Validierungs- und Fehler-Middleware und eine konsistente REST-Struktur, welche alle Domänen einheitlich abbildet.

Darauf aufbauend wurde der Application Layer neu gestaltet und die Geschäftslogik in klar definierte Use Cases überführt. Gleichzeitig wurden zahlreiche Datenbank-Mapper (zum Zeitpunkt der Umsetzung 19) für zentrale Entitäten wie Events, Tasks, Rewards, Leaderboards, Personen und Rollen entwickelt, um Datenflüsse sauber zwischen Datenbank-, Domain- und DTO-Ebene zu trennen. Auch die DB-Service-Schicht wurde umfassend überarbeitet. Exemplarisch umfasst der neugestaltete `user.db.service.ts` inzwischen deutlich über 500 Zeilen und kapselt sämtliche Benutzerinteraktionen zentral. Die Domain-Modelle wurden erweitert, unter anderem um `GamificationParameter`, `Reminder`, `MembershipFee`, wodurch eine modularere Fachlogik und eine klarere Domänenstruktur erreicht wurden.

Der Discord-Bot wurde daraufhin vollständig neu implementiert und kommuniziert nun ausschliesslich über einen dedizierten API-Client mit dem Backend, welcher sämtliche HTTP-Zugriffe kapselt und damit eine klare Trennung von Präsentations- und Geschäftslogik sicherstellt.

Im Ergebnis resultierte eine saubere, modulare Gesamtarchitektur mit klar abgegrenzten Domänen, einer vollständig entkoppelten Struktur zwischen Bot, Dashboard und Backend sowie einer API-first Plattform. Die neu gestalteten Use Cases sind testbar und erweiterbar, ein gemeinsames typisiertes Datenmodell stellt konsistente Datenflüsse sicher und die Systembasis bildet eine robuste Grundlage für Gamification, Rollen-Synchronisation, Event-Management sowie zukünftige Erweiterungen. Trotz des beträchtlichen Aufwands stellt die neue Architektur eine langfristig wartbare und skalierbare Plattform dar, die den Anforderungen der geplanten Weiterentwicklung entspricht.

7.3 Admin-Dashboard

Im Vorprojekt mussten verschiedene administrative Daten direkt in der Datenbank angepasst werden, was sowohl fehleranfällig als auch für Vorstandsmitglieder ohne technische Kenntnisse ungeeignet war (vgl. Abschnitt 1.2). Zur Vereinfachung dieser Verwaltungsaufgaben wurde im Rahmen dieses Projekts ein webbasiertes Admin-Dashboard implementiert. Das Dashboard dient als zentrale Administrationsoberfläche für Vereinsdaten und ermöglicht eine sichere, strukturierte und nachvollziehbare Verwaltung der relevanten Domänen. Die Umsetzung erfolgte mit dem Angular-Framework.

Das Admin-Dashboard ist vollständig in die API-first-Architektur von ClansCore integriert und greift ausschliesslich über definierte REST-Endpunkte auf die Backend-API zu. Direkte Datenbankzugriffe sind nicht vorgesehen. Dadurch bleibt die Trennung zwischen Präsentationsschicht und Geschäftslogik konsequent gewahrt.

7.3.1 State-Management mit NgRx

Zur Verwaltung des Anwendungszustands im Admin-Dashboard wurde NgRx eingesetzt. Die Wahl dieses State-Management-Ansatzes erlaubt eine zentrale, vorhersagbare Zustandsverwaltung und unterstützt eine klare Trennung zwischen UI-Komponenten, Datenhaltung und Seiteneffekten. Dies ist insbesondere für ein administratives Dashboard mit mehreren Datenansichten, Formularen und asynchronen API-Aufrufen von Vorteil.

Der Datenfluss im Dashboard folgt einem einheitlichen Muster. Benutzeraktionen oder Lifecycle-Ereignisse in Komponenten lösen Actions aus, welche den Zustand nicht direkt verändern, sondern über den Store verarbeitet werden. Asynchrone Operationen, wie API-Aufrufe, sind in Effects gekapselt und dadurch vollständig von der Präsentationslogik entkoppelt. Der globale Zustand wird ausschliesslich durch Reducer verändert, welche als reine Funktionen implementiert sind und keine Seiteneffekte enthalten.

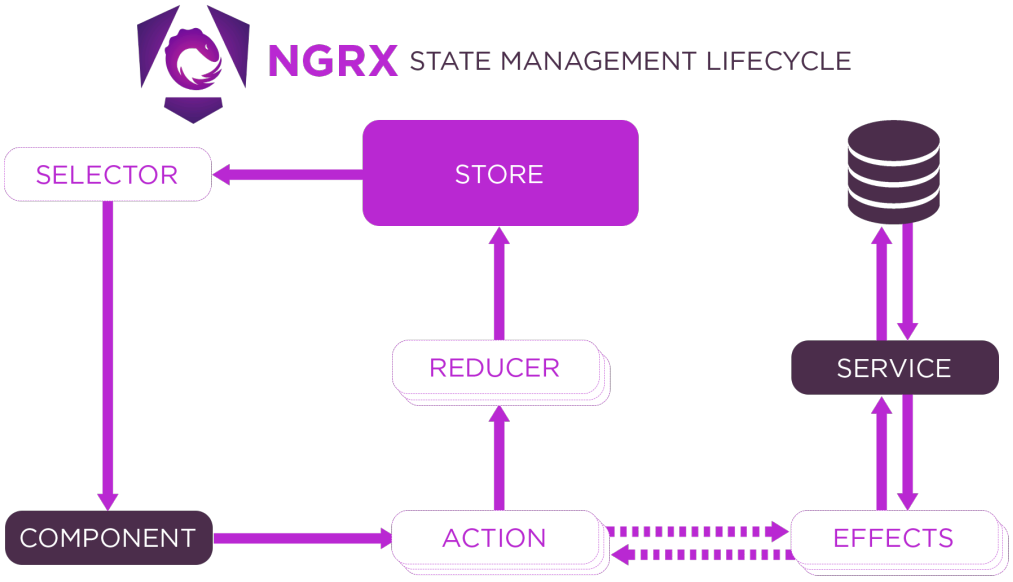


Abb. 21: Standardisierter Ablauf des State-Managements mit NgRx (NgRx-Contributors, 2025)

Dieses Architekturkonzept stellt sicher, dass Zustandsänderungen im Dashboard jederzeit nachvollziehbar bleiben und erleichtert sowohl Wartung als auch Erweiterung der Anwendung.

7.3.2 Umsetzung des Datenflusses

Für jede fachliche Domäne des Dashboards werden spezifische Actions definiert, welche Ereignisse wie das Laden, Erstellen oder Aktualisieren von Daten repräsentieren. Diese Actions dienen als einzige Möglichkeit, den Zustand des Stores zu beeinflussen.

```
export const getEvents = createAction('[Event] GetEvents');
export const getEventsSuccess = createAction(
  '[Event] GetEvents Success',
  props<{ events: GuildEvent[] }>()
);

export const getEventsFailure = createAction(
  '[Event] GetEvents Failure',
  props<{ error: string }>()
);
```

Listing 2: Ausschnitt aus der Implementierung einer Action zur Event-Verwaltung

Die Actions werden in den Komponenten ausgelöst, etwa beim Initialisieren einer Seite oder durch Benutzerinteraktionen.

```
ngOnInit() { this.store.dispatch(EventActions.getEvents()); }
```

Listing 3: Ausschnitt aus der Implementierung des Auslösens einer Action

Die Verarbeitung der Actions erfolgt in Effects, welche für die Durchführung von API-Aufrufen zuständig sind. Die Ergebnisse dieser Aufrufe werden anschliessend als Success- oder Failure-Actions an den Store zurückgegeben.

```
this.getEvents$ = createEffect(() =>
  this.actions$.pipe(
    ofType(EventActions.getEvents),
    mergeMap(() =>
      this.eventService.getAllEvents().pipe(
        map(events => EventActions.getEventsSuccess({ events })),
        catchError(error =>
          of(EventActions.getEventsFailure({
            error: error.message || 'Failed to load events'
          }))
) ) ) ) );
```

Listing 4: Ausschnitt aus der Implementierung eines Effects zur Kapselung eines API-Aufrufs

Reducer verarbeiten diese Actions und erzeugen daraus einen neuen, unveränderlichen Zustand.

```
export interface EventState {
  events: GuildEvent[];
  loading: boolean;
  error: string | null;
}

export const initialState: EventState = {
  events: [],
  loading: false,
  error: null,
};
```

Listing 5: Ausschnitt aus der Implementierung Zustandsdefinition eines Reducers

Zur Nutzung der Daten im UI werden Selektoren eingesetzt, welche gezielt Teilbereiche des Zustands extrahieren. Im Dashboard werden diese Selektoren über Angular Signals konsumiert, wodurch UI-Komponenten automatisch auf Zustandsänderungen reagieren.

```
export const selectEvents =
  createSelector(selectEventState, (state) => state.events);

events = this.store.selectSignal(EventSelectors.selectEvents);
```

Listing 6: Ausschnitt aus der Implementierung eines Selektors zur Anbindung des Stores an die UI

Durch diese Architektur bleibt die Komponentenlogik schlank, während Datenzugriffe, Fehlerbehandlung und Zustandsänderungen zentral und konsistent im Store verwaltet werden.

7.3.3 Authentifizierung und Zugriffskontrolle

Der Zugriff auf das Admin-Dashboard ist ausschliesslich Vorstandsmitgliedern vorbehalten. Die Authentifizierung erfolgt über eine klassische Login-Maske mit E-Mail-Adresse und Passwort. Passwörter werden niemals im Klartext gespeichert, sondern mittels bcrypt gehasht. Für jedes Passwort wird automatisch ein individueller Salt erzeugt, wodurch gängige Angriffe wie Rainbow-Table-Attacken verhindert werden.

Zur administrativen Verwaltung von Zugängen wurde ein separater Admin-Login implementiert. Über diesen können neue Personen angelegt, bestehende Personen bearbeitet sowie Passwörter für Vorstandsmitglieder zurückgesetzt werden. Nach der Anmeldung sind Nutzende in der Lage, eigenständig ihr Passwort zu ändern, durch erneute Eingabe des bisherigen Passworts.

Die Autorisierung basiert auf den zentralen Rollen „Vorstand“ und „Mitglied“, welche sowohl für die Zugriffskontrolle im Discord-Bot als auch für das Admin-Dashboard verwendet werden. Diese Rollen sind systemweit fest definiert und dürfen nicht verändert werden. Die Authentifizierung für das Dashboard erfolgt unabhängig von Discord, während die Discord-Rollen ausschliesslich zur Autorisierung von Bot-Befehlen sowie zur Zugriffskontrolle auf Dashboard-Funktionen herangezogen werden.

7.4 Erweiterung des Discord-Bots durch Webhooks

Dieses Kapitel baut auf den im Abschnitt 5.2.3 beschriebenen architekturellen Entkopplungsmechanismen auf und beschreibt deren konkrete technische Umsetzung. Zur Erweiterung der bestehenden Funktionalität des Discord-Bots wurde eine webhook-basierte Schnittstelle zwischen der ClansCore-API und dem Discord-Bot implementiert. Ziel dieser Erweiterung war es, systemübergreifende Änderungen aus dem Backend zuverlässig und entkoppelt im Discord-Server auszuführen, ohne direkte Abhängigkeiten zwischen API und Discord-spezifischer Logik zu erzeugen.

Die Kommunikation erfolgt unidirektional von der API zum Discord-Bot über HTTP-Webhooks. Auf Seiten der API übernimmt ein zentraler NotificationService die Orchestrierung aller Benachrichtigungen. Dieser Service nutzt ein Adapter-Pattern, wodurch die Discord-Integration vollständig im DiscordAdapter gekapselt ist. Dadurch bleibt die API plattformunabhängig und kann bei Bedarf um weitere Zielsysteme erweitert werden.

Der Discord-Bot betreibt hierfür einen separaten HTTP-Server auf Basis von Express. Diese Trennung vom eigentlichen Discord-Gateway erlaubt eine klare Verantwortungsabgrenzung zwischen Ereignisempfang (Webhook) und Ausführung von Discord-Aktionen (Rollen, Nachrichten, Logs). Alle Webhook-Endpunkte sind thematisch gebündelt und verarbeiten ausschliesslich klar definierte Ereignistypen wie Statusänderungen, Rollenänderungen oder Synchronisationsaufrufe.

```
await notificationService.notifyApplicationAccepted({
  userId: person._id.toString(),
  platformUserId: person.discordId,
  username,
  roleName,
}).catch(() => { ... });
```

Listing 7: Ausschnitt aus der Implementierung des NotificationService

Nach erfolgreichem Abschluss des Use Cases wird ein plattformneutrales Ereignis über den NotificationService ausgelöst. Die eigentliche Zustellung an Discord erfolgt asynchron über einen Adapter und beeinflusst den Abschluss des Use Cases nicht. Der DiscordAdapter kapselt sämtliche Discord-spezifischen Integrationsdetails. Die API kennt weder Endpunkte noch Datenstrukturen des Discord-Bots und bleibt dadurch vollständig plattformunabhängig.

```
async onApplicationAccepted(event: ApplicationAcceptedEvent): Promise<void> {
  if (!this.isEnabled()) return;

  await fetch(`${this.baseUrl}/api/notifications/user-status`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "x-webhook-token": this.secret,
    },
    body: JSON.stringify({
      discordId: event.platformUserId,
      status: "Accepted",
      roleName: event.roleName,
    }),
  }).catch(() => { ... });
}
```

Listing 8: Ausschnitt aus der Implementierung des DiscordAdapters

7.4.1 Validierung und Sicherheit

Alle eingehenden Webhook-Requests werden strikt validiert. Die Payloads werden mittels Zod-Schemas geprüft, wodurch fehlerhafte oder unvollständige Requests frühzeitig abgefangen werden. Zusätzlich ist die Schnittstelle durch einen gemeinsamen Secret-Token abgesichert, der im HTTP-Header übertragen wird und ausschliesslich über Umgebungsvariablen konfiguriert ist.

Fehler bei der Webhook-Verarbeitung führen nicht zum Abbruch der auslösenden Geschäftslogik in der API. Stattdessen werden sie strukturiert geloggt (vgl. Abschnitt 7.5). Auf Bot-Seite werden Validierungs-, Authentifizierungs- und Ressourcenfehler klar voneinander getrennt und mit passenden HTTP-Statuscodes beantwortet. Nicht zustellbare Direktnachrichten werden protokolliert, ohne den gesamten Webhook als fehlgeschlagen zu markieren.

```
if (!verifySecret(req)) {
  return res.status(401).json({ error: "Unauthorized" });
}

const body = UserStatusSchema.parse(req.body);
```

Listing 9: Ausschnitt aus der Implementierung der Webhook-Route

7.4.2 Automatisierte Benutzer-Synchronisation

Die bestehende Benutzer-Synchronisation wurde so umgesetzt, dass sie sowohl manuell über einen Discord-Command als auch automatisiert über einen Webhook ausgelöst werden kann. Ein Cron-Job in der API triggert diesen Webhook periodisch. Implementierungsseitig greifen beide Varianten auf dieselbe Anwendungslogik zurück, wodurch redundanter Code vermieden und konsistentes Verhalten sichergestellt wird. Synchronisiert werden ausschliesslich Personen mit den zentralen Rollen „Mitglied“ und „Vorstand“.

7.5 Logging und Fehlerbehandlung

Für den Betrieb, die Wartbarkeit sowie die Nachvollziehbarkeit administrativer Aktionen wurde ein Logging-Konzept implementiert. Ziel dieses Konzeptes ist es, relevante Systemereignisse transparent zu protokollieren und gleichzeitig die Benutzerinteraktion in Discord übersichtlich zu halten.

Das Logging unterscheidet dabei zwischen technischem Logging, Ereignis-Logging und benutzerspezifischem Feedback.

7.5.1 Zentrales Ereignis-Logging über Discord

Kern des Loggings ist ein dedizierter Discord-Textkanal namens `bot-log`, welcher ausschliesslich für Vorstandsmitglieder sichtbar ist. Dieser Channel dient als Audit-ähnliches Protokoll für alle vereinsrelevanten Änderungen, die entweder durch Discord-Befehle, automatisierte Prozesse oder externe Systeme ausgelöst werden.

Geloggte Ereignisse umfassen:
- Rollen- und Statusänderungen von Mitgliedern
- Synchronisationen von Nutzern und Rollen zwischen Discord und Datenbank
- Erstellung von Gamification-Elementen (Aufgaben und Ranglisten)
- Spenden-Transaktionen mit Punktevergabe
- Manuelle Kalender-Synchronisationen und Kalender-Verlinkung

Die Logeinträge werden als strukturierte Discord-Embeds umgesetzt. Diese enthalten Titel, Metadaten (z. B. ausführende Person oder Systemprozess), Detailinformationen sowie einen Zeitstempel. Durch die Verwendung von Farben können unterschiedliche Ereignistypen visuell unterschieden werden.

```
const embed = new EmbedBuilder()
    .setTitle("🔄 Rollenänderung (Dashboard)")
    .setColor(0x3498db)
    .addFields(
        { name: "Benutzer", value: `<@${body.discordId}>`, inline: true },
        { name: "Geändert von", value: body.changedBy, inline: true },
    )
    .setTimestamp();

await logChannel.send({ embeds: [embed] }).catch( ... );
```

Listing 10: Implementierungs-Ausschnitt des Sendens eines Log-Embeds in den `bot-log` Channel

Der `bot-log` Channel fungiert damit als zentrale, chronologische Änderungsübersicht und stellt eine pragmatische Alternative zu einem datenbankbasierten Audit-Logging dar. (Für Bilder zum Discord-Bot, vgl. Abschnitt 17)

7.5.2 Trennung von Logging und Benutzerfeedback

Ein zentrales Designprinzip ist die strikte Trennung zwischen Logging und Nutzerkommunikation. Während administrative und systemrelevante Ereignisse im `bot-log` protokolliert werden, erhalten ausführende Personen ihr Feedback ausschliesslich über Ephemeral Messages.

Ephemeral Messages werden für Erfolgsbestätigungen, Statusanzeigen sowie Fehlermeldungen verwendet und sind nur für die jeweilige Person sichtbar. Dadurch wird verhindert, dass öffentliche Channels mit technischen oder personenbezogenen Informationen überladen werden, während gleichzeitig eine direkte und verständliche Rückmeldung gewährleistet wird.

```ts
export async function execute(interaction: CommandInteraction) {
  await interaction.deferReply({ flags: MessageFlags.Ephemeral });

  const guild = interaction.guild as Guild;
  const syncResult = await performSyncUsers(guild, `<@${interaction.user.id}>`);

  if (!syncResult.ok) {
    return interaction.editReply(
      getErrorMessage(syncResult.error));
  }
  const { changes } = syncResult.value;

  let content = `✅ User-Synchronisierung mit Datenbank abgeschlossen.`;
  const embeds = [...];

  await interaction.editReply({ content, embeds });
}
```

Listing 11: Implementierungs-Ausschnitt eines Ephemeral Benutzerfeedbacks bei Befehlen

7.5.3 Technisches Logging und Fehlerbehandlung

Zusätzlich zum Discord-basierten Logging wird klassisches Console-Logging eingesetzt. Dieses dient primär technischen Zwecken und umfasst:

- Start- und Initialisierungsinformationen der Services
- Statusmeldungen zu Datenbank-Verbindungen
- Ausführung von Cron-Jobs
- Fehlerausgaben bei unerwarteten Exceptions

Fehler werden im Projekt über ein zentrales Error-Handling-Modell verarbeitet. Fehlerobjekte enthalten einen Typ sowie strukturierte Detailinformationen. Abhängig vom Kontext werden Fehler als Ephemeral Message an die Person zurückgegeben, im Console-Log ausgegeben oder bei systemrelevanten Fehlern zusätzlich implizit im `bot-log` reflektiert (z. B. bei fehlgeschlagenen Synchronisationen).

7.5.4 Systemübergreifendes Logging mittels Webhooks

Da ClansCore aus mehreren gekoppelten Anwendungen besteht, erfolgt ein Teil des Loggings systemübergreifend. Die API informiert den Discord-Bot über relevante Ereignisse (z. B. Rollenänderungen im Dashboard oder automatische User-Synchronisationen) mittels authentifizierter Webhooks.

```ts
await fetch(`${this.baseUrl}/api/notifications/role-changed`, {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
        "x-webhook-token": this.secret,
    },
    body: JSON.stringify({
        discordId: event.platformUserId,
        ...
    }),
});
```

Listing 12: Implementierungs-Ausschnitt einer Webhook-basierter Log-Auslösung aus der API

Der Discord-Bot übernimmt dabei ausschliesslich die Darstellung und Protokollierung im `bot-log` Channel. Die Logik verbleibt in der API, wodurch eine klare Trennung der Verantwortlichkeiten und eine lose Kopplung der Systeme erreicht wird. Diese Architektur erlaubt eine schrittweise Erweiterung auf weitere Ereignistypen, ohne strukturelle Änderungen am Gesamtsystem vorzunehmen.

7.5.5 Dashboard-bezogene Logging-Aspekte

Das Admin-Dashboard verfügt über kein eigenes Logging-System im Sinne eines eigenständigen Audit- oder Ereignis-Logs. Alle relevanten Aktionen werden über HTTP-Requests an die API ausgelöst und dort verarbeitet. Erst auf API-Ebene erfolgt die Auslösung von fachlich vorgesehen Logging-Ereignissen mittels Webhooks an den Discord-Bot.

Entsprechend werden Dashboard-Aktionen indirekt im `bot-log` Channel protokolliert, sofern diese als vereins- oder sicherheitsrelevant eingestuft sind. Zu den aktuell geloggten Dashboard-Aktionen zählen Rollenänderungen an Benutzern (Hinzufügen oder Entfernen von Rollen) sowie Aktualisierungen bestehender Rollen. Diese Aktionen werden mit strukturierten Embeds dokumentiert, inklusive Angabe des betroffenen Benutzers bzw. der Rolle sowie des ausführenden Dashboard-Administrators.

Andere Dashboard-Aktionen (das Erstellen, Bearbeiten oder Löschen von Benutzern, Aufgaben, Events, Ranglisten, Spenden oder Belohnungen) werden bewusst nicht im Discord-Logging erfasst, sondern einzig in der Datenbank persistiert.

Das Nutzer-Feedback im Dashboard erfolgt über lokale UI-Mechanismen (Angular Material Snackbar) und wird nicht geloggt. Diese Rückmeldungen sind ausschliesslich für die aktuell angemeldete Person sichtbar und dienen der unmittelbaren Interaktion, nicht der Nachverfolgung.

7.6 Event-Synchronisierung

Die Event-Synchronisierung verbindet Google Calendar, die interne Datenbank und Discord Events zu einem konsistenten Gesamtsystem. Die technische Umsetzung folgt einer klaren Prioritätshierarchie, um konkurrierende Änderungen eindeutig aufzulösen und inkonsistente Zustände zu vermeiden.

Google Calendar fungiert als primäre Quelle für Eventdaten. Änderungen aus dem Kalender haben grundsätzlich Vorrang und werden in die Datenbank übernommen. Die Datenbank übernimmt eine vermittelnde Rolle und dient als persistenter Integrationslayer zwischen den externen Systemen. Discord wird ausschliesslich aus der Datenbank heraus synchronisiert und dient lediglich als Darstellungs- und Interaktionsplattform. Eine direkte Synchronisation zwischen Google Calendar und Discord ist nicht vorgesehen.

Die Synchronisation zwischen Google Calendar und Datenbank erfolgt bidirektional. Für die Konfliktauflösung wird ein Zeitstempel-Vergleich (`updatedAt`) eingesetzt, wobei stets die aktuellere Version übernommen wird. Neue Events werden in der Datenbank angelegt, bestehende bei Abweichungen aktualisiert. Umgekehrt werden Datenbank-Events nach Google Calendar zurückgeschrieben, sofern diese neuer sind. Diese Rücksynchronisation wird jedoch gezielt übersprungen, wenn parallele Änderungen aus Discord erkannt werden, um Benutzeraktionen nicht unbeabsichtigt zu überschreiben.

Zur Begrenzung von Laufzeit und Datenmenge werden nur Events innerhalb eines konfigurierbaren Zeitfensters verarbeitet. Vergangene sowie weit in der Zukunft liegende Events werden automatisch aus der Datenbank entfernt. Dadurch bleibt die Synchronisation auch bei langfristigem Betrieb performant und wartbar.

Die Synchronisation von der Datenbank nach Discord erfolgt in einem separaten Verarbeitungsschritt. Bestehende Discord Events werden anhand gespeicherter Event-IDs abgeglichen und bei Bedarf aktualisiert oder gelöscht. Falls keine direkte Zuordnung möglich ist, wird eine Zuordnung über Name und Startzeit verwendet. Laufende oder explizit abgesagte Events werden dabei geschützt und weder überschrieben noch neu erstellt.

Abb. 22: Prioritätshierarchie sowie Datenflüsse zwischen Google Calendar, Datenbank und Discord.

Ein besonderer Implementierungsaspekt betrifft wiederkehrende Events. Da Discord nur eine eingeschränkte Unterstützung für Wiederholungsregeln bietet (Discord-Contributors, 2025) , werden wiederkehrende Events aus Google Calendar abhängig vom jeweiligen Wiederholungsmuster unterschiedlich verarbeitet. Unterstützte Muster werden als einzelnes Discord-Event mit begrenzter Wiederholungsregel synchronisiert, während nicht unterstützte Muster vollständig aufgelöst und als einzelne Events übertragen werden. Die maximale Anzahl angezeigter Wiederholungen ist konfigurierbar.

Die Event-Synchronisierung kann sowohl periodisch als auch ereignisbasiert ausgelöst werden. Unabhängig vom Auslöser wird stets ein vollständiger Synchronisationslauf durchgeführt, um einen konsistenten Endzustand sicherzustellen.

Zur Absicherung des Betriebs wurden mehrere Schutzmechanismen implementiert. Kurzfristige, durch den Bot selbst ausgelöste Änderungen werden temporär ignoriert, um Endlosschleifen zu verhindern. Zusätzlich werden kürzlich geänderte Discord Events vor unmittelbarem Überschreiben geschützt. Diese Mechanismen stellen insbesondere bei parallelen Änderungen die Stabilität des Systems sicher.

Fehler während der Synchronisation werden zentral protokolliert, unterbrechen den Ablauf jedoch nicht vollständig. Einzelne fehlerhafte Events werden übersprungen, sodass die Synchronisation der übrigen Events fortgesetzt werden kann und der Betrieb auch bei partiellen Fehlern gewährleistet bleibt.

7.7 Umsetzung des Punktesystems

Das Punktesystem von ClansCore basiert auf dem im Vorprojekt entwickelten Gamification-Konzept. Im Vorprojekt wurde eine Excel-Vorlage zur Quantifizierung der Punktevergabe eingesetzt, welche eine transparente und konsistente Planung ermöglichte sowie die Simulation unterschiedlicher Szenarien hinsichtlich Spenden, Aufwänden und Eventteilnahmen unterstützte (vgl. Vorprojekt, Kap. 4.8).

Die ausschliessliche Verwendung einer Excel-Vorlage erwies sich jedoch als unpraktisch, da Aktualität, gemeinsame Nutzung und Koordination zwischen mehreren Vorstandsmitgliedern nur eingeschränkt gewährleistet waren. Aus diesem Grund wurde das Punktesystem vollständig in das Admin-Dashboard überführt, welches eine zentrale, jederzeit aktuelle und rollenbasierte Verwaltung ermöglicht.

Zur Unterstützung des Übergangs orientiert sich die Struktur des Dashboards an der bestehenden Excel-Vorlage. Unter dem Bereich Gamification-Verwaltung stehen vier funktionale Teilbereiche zur Verfügung: Die Punktedefinition dient der Verwaltung globaler Gamification-Parameter (z. B. Punkte pro CHF und pro CHF Spende) sowie der Definition von Aufgabentypen. Die Jahresplanung ermöglicht eine aggregierte Übersicht über geplante Spenden und erwartete Punktevergaben. Ergänzend können Belohnungen verwaltet sowie mittels Punktebeispiel exemplarische Jahresplanungen für einzelne Mitglieder simuliert werden.

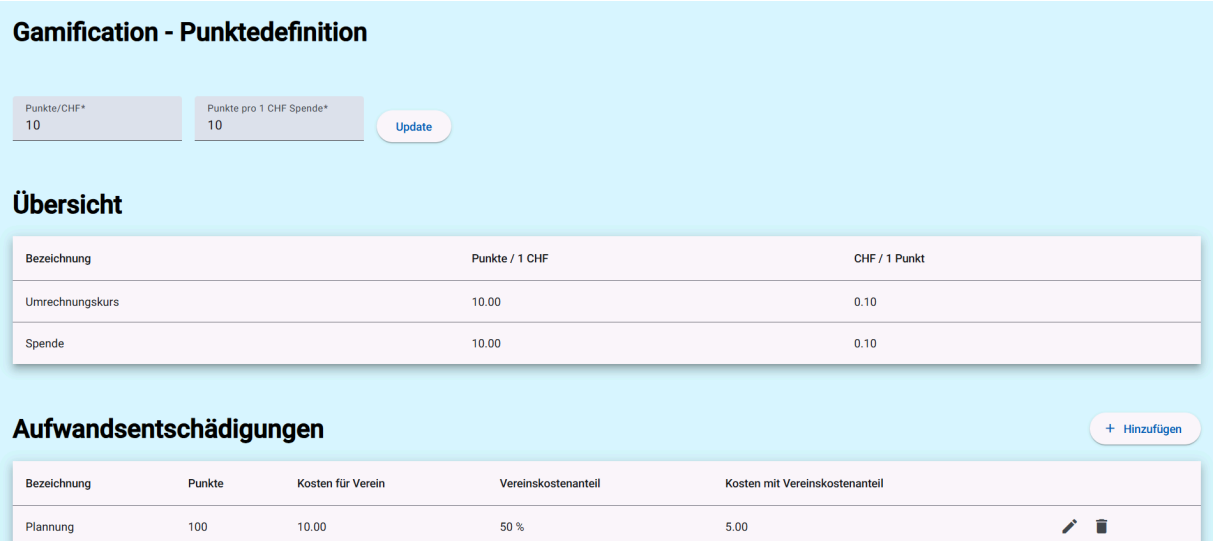


Abb. 23: Dashboard: Gamification Punktedefinition

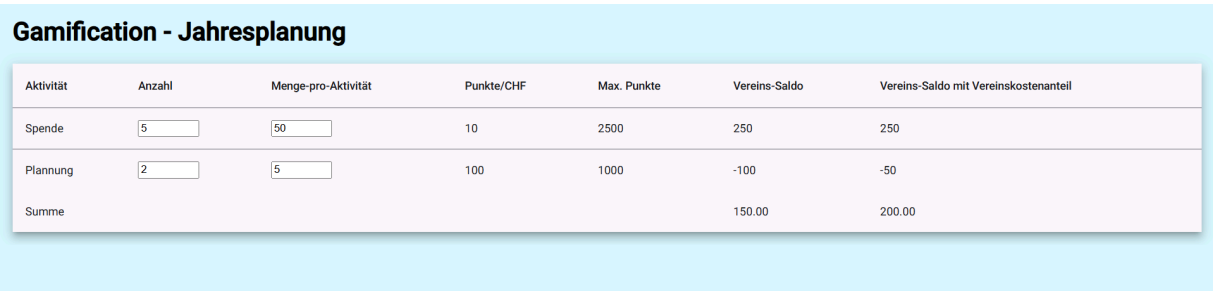


Abb. 24: Dashboard: Gamification Jahresplanung

Die operative Nutzung des Punktesystems erfolgt primär über den Discord-Bot. Aufgaben werden über den Befehl `/createtask` erstellt. Für eine automatische Punkteschätzung werden Aufgabentypen verwendet, die entweder als Einmalig oder nach Aufwand klassifiziert sind. Einmalige Aufgaben erhalten eine feste Punktzahl, während aufwandsbasierte Aufgaben anhand der definierten Punkte pro Stunde bewertet werden. Wird kein Aufgabentyp ausgewählt, erfolgt keine Punkteschätzung. Die berechneten Werte dienen als Vorschlag und unterstützen Vorstandsmitglieder bei der konsistenten und nachvollziehbaren Punktevergabe.

7.8 Bewertung der Sicherheitsumsetzung

Dieser Abschnitt bewertet die Umsetzung des im erweiterten Sicherheitskonzept (vgl. Abschnitt 4) beschriebenen Zielzustands in der aktuellen Version von ClansCore. Ziel ist es, den Grad der Umsetzung sicherheitsrelevanter Anforderungen transparent darzustellen und Abweichungen zwischen Konzept und Implementierung nachvollziehbar einzuordnen.

Das Sicherheitskonzept beschreibt eine erweiterte Zielarchitektur, die über die im Projekt minimal erforderlichen Sicherheitsmechanismen hinausgeht. Die Bewertung erfolgt vor diesem Hintergrund als reflektierte Einordnung des tatsächlich realisierten Sicherheitsniveaus.

Bereich	Sicherheit-Konzept	Tatsächliche Umsetzung
Authentifizierung	JWT + optionale MFA	JWT ohne MFA
Autorisierung	RBAC + Least Privilege	RBAC umgesetzt
Transportverschlüsselung	TLS/HTTPS	Nicht umgesetzt
Eingabevalidierung	Vorgesehen	Zod-Schemas
Audit & Logging	DB-Audit-Log	Discord-Logging
Datenintegrität	OCC + Transaktionen	Teilweise Transaktionen
Infrastruktur	Container + Backups	Docker + GPG-Backups

Tab. 45: Vergleich Sicherheitskonzept und Implementierung

Zentrale Sicherheitsmechanismen wie JWT-basierte Authentifizierung, rollenbasierte Zugriffskontrolle, Eingabevalidierung sowie grundlegende Infrastrukturabsicherung wurden umgesetzt. Das Prinzip Least Privilege wird sowohl im Admin-Dashboard als auch im Discord-Bot konsequent angewendet.

Weitere im Sicherheitskonzept definierte Massnahmen, insbesondere Transportverschlüsselung (TLS), Multi-Faktor-Authentifizierung, Optimistic Concurrency Control sowie ein vollständiges Audit-Log auf Datenbankebene, wurden im Rahmen des Projektumfangs nicht implementiert. Diese Massnahmen erfordern zusätzliche infrastrukturelle und betriebliche Voraussetzungen und wurden im Rahmen dieser Arbeit bewusst als erweiterte Sicherheitsziele definiert.

Die bestehende Architektur (Reverse Proxy, containerisierte Services, zentrale Autorisierung) erlaubt eine spätere Nachrüstung dieser Massnahmen ohne strukturelle Änderungen am Anwendungscode.

8 Fazit

Dieses Kapitel fasst die Resultate der Arbeit zusammen, reflektiert die getroffenen technischen und konzeptionellen Entscheidungen kritisch und gibt einen Ausblick auf mögliche Weiterentwicklungen von ClansCore. Dabei wird der Grad der Zielerreichung anhand der definierten Anforderungen beurteilt und in einen übergeordneten Projektkontext eingeordnet.

8.1 Resultate

Ziel dieser Arbeit war die Konzeption und Implementierung eines Discord-basierten Systems zur Unterstützung vereinsrelevanter Prozesse mit besonderem Fokus auf Mitgliederverwaltung, Eventorganisation und Gamification. Die Umsetzung orientierte sich konsequent an den in Kapitel Anforderungen definierten Use Cases sowie den nichtfunktionalen Anforderungen.

Die definierten Use Cases für die Nutzergruppen Verein, Vorstand, Mitglied und Nicht-Mitglied (vgl. UC1 , UC2 , UC3 , UC4 , UC5 , UC6) konnten im vorgesehenen Umfang umgesetzt oder bewusst abgegrenzt werden. Die Online-Mitgliedsbewerbung wurde technisch umgesetzt, jedoch aufgrund eines nicht realisierten Annahme-Workflows bewusst deaktiviert (vgl. UC2). Administrative Kernprozesse wie die Mitglieder- und Rollenverwaltung, ausgewählte Aspekte der Eventverwaltung, die Aufgaben- und Gamification-Steuerung sowie die Anzeige von Ranglisten beziehungsweise Aktivitätsdaten wurden produktiv realisiert.

Zentrale Interaktionen im Vereinsalltag, wie die Verwaltung von Aufgaben inklusive Punktevorschlägen (vgl. UC5), die Einsicht in Gamification-Daten (vgl. UC6) sowie grundlegende Event-Erinnerungen mit einer einzelnen Reminder-Stufe (vgl. UC9), wurden erfolgreich umgesetzt und technisch validiert.

Eine qualitative Evaluation der Nutzung und der wahrgenommenen Motivationseffekte ist nach Abgabe der schriftlichen Arbeit vorgesehen und stellt einen zentralen Bestandteil der weiteren Projektarbeit dar.

Mehrere Use Cases wurden bewusst nur teilweise oder nicht umgesetzt. Dazu zählen unter anderem die Anmeldung von Mitgliedern zu Aufgaben über das Dashboard (vgl. UC7), automatisierte Zahlungserinnerungen und Zahlungsbestätigungen (vgl. UC8) sowie weiterführende Plattform-Integrationen ausserhalb von Discord (vgl. UC10). Diese Abgrenzungen wurden vorgenommen, um den Fokus auf ein stabiles und funktionales Minimum Viable Product zu legen. (vgl. Abschnitt 9.6)

Aus zeitlichen Gründen wurde zudem auf eine flächendeckende Implementierung atomarer Datenbanktransaktionen verzichtet. Die bestehenden Implementierungen gewährleisten im vorgesehenen Nutzungsszenario eine ausreichende Datenkonsistenz und bilden eine tragfähige Grundlage für zukünftige Erweiterungen.

8.2 Schlussfolgerung

Die Arbeit zeigt, dass Discord als Plattform nicht nur für Kommunikation, sondern auch als technische Basis für vereinsrelevante Verwaltungsprozesse geeignet ist. Durch die Kombination eines Discord-Bots mit einer zentralen Datenhaltung und einem webbasierten Admin-Dashboard konnte ein System geschaffen werden, das organisatorische Abläufe vereinfacht und gleichzeitig die Mitgliederaktivität fördert.

Ein wesentlicher Erfolgsfaktor war die notwendige strukturelle Neuordnung der bestehenden Codebasis, um der gewachsenen fachlichen und technischen Komplexität von ClansCore gerecht zu werden. Die Architektur wurde schrittweise weiterentwickelt und entlang klarer Verantwortlichkeiten neu ausgerichtet, insbesondere durch die Trennung von fachlicher Logik, technischer Infrastruktur und plattformspezifischen Integrationen.

Dabei kamen grundlegende Architekturprinzipien wie Modularisierung, Separation of Concerns und lose Kopplung zum Einsatz. Diese wurden pragmatisch und projektbezogen angewendet, ohne einem formalen Architektur- oder Methodikansatz wie Domain-Driven Design (DDD) strikt zu folgen. Das Refactoring stellte keinen Selbstzweck dar, sondern war Voraussetzung für die stabile Umsetzung der definierten Use Cases und die langfristige Erweiterbarkeit des Systems.

Diese Strukturierungsprinzipien reduzieren den Wartungsaufwand und unterstützen die langfristige Erweiterbarkeit des Systems, erfordern jedoch einen erhöhten initialen Analyse- und Refactoring-Aufwand.

Trotz der architektonischen Entkopplung der Kernlogik verbleibt eine minimale technische Kopplung an die Discord-Plattform. Diese zeigt sich beispielsweise darin, dass Discord-Benutzer-IDs als externe Referenzen für

Personen gespeichert werden und Discord-Rollen als Quelle für Authentifizierungs- und Autorisierungsentscheidungen dienen. Die fachliche Logik von ClansCore ist jedoch nicht an Discord gebunden. Konzepte wie Personen, Rollen, Aufgaben oder Punkte existieren unabhängig von der Plattform und werden lediglich über Adapter mit Discord synchronisiert. Die verbleibende technische Kopplung wurde bewusst akzeptiert, da sie eine konsistente Personenverwaltung ermöglicht und den Integrations- sowie Wartungsaufwand im Projektkontext reduziert.

Neben den funktionalen Interaktionen konnten auch zentrale nichtfunktionale Anforderungen erfüllt werden. Dies betrifft insbesondere die Sicherheit und den Schutz personenbezogener Daten (vgl. NFR4), die Wartbarkeit und Codequalität des Systems (vgl. NFR1) sowie die Kompatibilität und Performance des Admin-Dashboards (vgl. NFR3 , NFR6). Anforderungen an Revisionssicherheit und Nachvollziehbarkeit wurden durch ein praxisnahes Logging-Konzept teilweise erfüllt (vgl. NFR5).

Einschränkungen im Admin-Dashboard, wie der vergleichsweise hohe Implementierungsaufwand durch NgRx sowie aktuell fehlende Filter- und Analysefunktionen, wurden bewusst in Kauf genommen, um den Fokus auf funktionale Kernprozesse und die architektonische Stabilität des Systems zu legen.

Damit konnte das in der Aufgabenstellung definierte Ziel einer modularen, erweiterbaren und praxistauglichen Weiterentwicklung eines Discord-basierten Vereinsverwaltungssystems mit Web-Dashboard erreicht werden.

8.3 Ausblick

ClansCore bildet eine stabile Grundlage für weiterführende Entwicklungen über den Rahmen dieser Arbeit hinaus. Ein zentrales Erweiterungspotenzial liegt in der Weiterentwicklung des Gamification- und Motivationskonzepts. Aufbauend auf den im Projekt identifizierten Erweiterungen könnten künftig unter anderem mehrstufige Event-Reminder (vgl. E1), datensparsame Vereins-Analytics (vgl. E2) sowie saisonale Punkteverfälle zur Förderung kontinuierlicher Aktivität (vgl. E3) umgesetzt werden.

Ergänzend bieten Anerkennungssysteme wie öffentliche Danksagungen oder Rollen-Hervorhebungen (vgl. E4) sowie kooperative Formate wie Team-Quests oder Gemeinschaftsziele (vgl. E5) Potenzial, Motivation stärker über Verbundenheit und Sinn zu fördern. Weitere niedrig priorisierte Elemente wie Event-Recaps (vgl. E6) oder transparente Curiosity-Mechaniken (vgl. E7) könnten das System langfristig ergänzen.

Auf technischer Ebene bieten sich weitere Optimierungen an. Die aktuell fixen Rollenbezeichnungen für Mitglied und Vorstand könnten durch ein dynamisches Rollenmodell ersetzt werden. Zudem könnten Webhooks zwischen Dashboard und Bot erweitert werden, um zusätzliche Use Cases wie Aufgaben- oder Event-Synchronisationen vollständig abzudecken.

Auch das Admin-Dashboard bietet weiteres Ausbaupotenzial, insbesondere durch Filter-, Such- und Analysefunktionen. Die Einführung eines zentralen, plattformunabhängigen Logging-Systems sowie automatisierter CI/CD-Pipelines würde den Betrieb in produktiven Umgebungen zusätzlich erleichtern.

Zusammenfassend zeigt diese Arbeit, dass ein Discord-basierter Ansatz zur Vereinsverwaltung technisch realisierbar, organisatorisch sinnvoll und eine hohe Akzeptanz erwarten lässt. ClansCore erfüllt die definierten Use Cases des MVP sowie die wesentlichen nichtfunktionalen Anforderungen und stellt eine tragfähige Basis für eine langfristige Weiterentwicklung dar.

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.

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:

- **Erweiterung Discord-Bot**: Ziel ist es, den bestehenden Discord-Bot basierend auf dem während des Einführungsprozesses für den Verein EGSwiss erhaltenen Feedback gezielt weiterzuentwickeln. Dabei sollen Funktionen neu erstellt, ergänzt und verbessert werden.
- **Erweiterung Gamification**: Ziel ist es, das bestehende Gamification-System auszubauen und zu untersuchen, welche Erweiterungen die Motivation und das Engagement im Verein steigern können.
- **Plattform-Unabhängigkeit**: Ein Admin-Dashboard soll erstellt werden, welches dem Vorstand Anpassungen an bereits erfassten Daten ermöglicht, ohne direkten Zugriff auf die Datenbank. Das Admin-Dashboard soll unabhängig von Discord nutzbar sein und somit die Applikation plattformunabhängig machen.

9.4 Long-Term Plan

Der Long-Term Plan wurde in 5 verschiedene Phasen aufgeteilt. Diese werden in der folgenden Tabelle genauer beschrieben.

Phase	Woche	Datum	Ziele	Meilenstein
Inception	1	15.09 - 21.09.	Einrichtung, MVP definieren	
	2	22.09. - 28.09.	Projektplan, Risikomanagement, Long-Term-Plan	M1
Elaboration	3	29.09. - 05.10.	Requirements, Einleitung, Stand der Technik	
	4	06.10. - 12.10.	Interviews andere Vereine, Sicherheitskonzept	
	5	13.10. - 19.10.	Gamificationkonzept, Architektur, Wireframes	
	6	20.10. - 26.10.	Testkonzept, Prototyp Admin-Dashboard (Proof of Concept)	M2
Construction (Version 2.0)	7	27.10. - 02.11.	Analyse Architektur Vorprojekt, Vollständiges Refactoring starten, 1.1, 1.2, 5.1	
	8	03.11. - 09.11.	API Grundgerüst, Dashboard aufsetzen, 3.1, 3.2	
	9	10.11. - 16.11.	API-Schicht, Shared-Library, erste Controller, 3.3, 3.4, 3.5	
	10	17.11. - 23.11.	Discord-Bot auf API umstellen, 2.1, 2.2, 2.3, 2.4, 3.6, 3.7, 3.8	M3
	11	24.11. - 30.11.	2.5, 2.6, 3.9, 3.10, 3.11	
	12	01.12. - 07.12.	2.7, 4, 5.2, 5.3, 5.4, Implementation beenden	M4
Transition	13	08.12. - 14.12.	Simulation vorbereiten, Dokumentation, Abschluss	
	14	15.12. - 19.12.	Fertigstellung, Abgabe Abstract und Dokumentation	M5
Presentation	15	20.12. - 28.12.	Simulation durchführen	
	16	29.12.2025 - 04.01.2026		
	17	05.01. - 09.01.	Ergebnisse auswerten, Präsentation und A0-Poster fertigstellen	M6

Tab. 46: Long-Term Plan mit angepasster Construction-Phase (Version 2.0)

Die angepassten Komponenten und Funktionen:

1. Datenbank
- **1.1**: Schema anpassen
- **1.2**: DB-Transactions / Rollbacks (Theorie angesehen + Beispiel-Umsetzung)

2. Discord-Bot
- **2.1**: Mitglied-Bewerbung (/join, /leave)
- **2.2**: Event-Ansicht (/events)
- **2.3**: Darstellung Events verbessern
- **2.4**: Kalender-Anbindung (/linkcalendar, /synccalendar)
- **2.5**: Gamification-Elemente (/score, /createtask, /completetask, /createleaderboard, /rewards, /donation)
- **2.6**: Daten-Ansicht (/getdata)
- **2.7**: Daten-Synchronisation (/syncroles, /syncusers)

3. Admin-Dashboard
- **3.1**: Login erstellen
- **3.2**: Mitgliederverwaltung
- **3.3**: Rollenverwaltung
- **3.4**: Transaktions-Historie (nur eigenes)
- **3.5**: Aufgabenverwaltung (ohne claim)
- **3.6**: Event-Ansicht (ohne Verwaltung)
- **3.7**: Belohnungs-Ansicht (ohne Verwaltung)
- **3.8**: Ranglisten-Ansicht (nur All-Time)
- **3.9**: Transaktionen anzeigen (Auswahl aller User, read-only)
- **3.10**: Ranglisten ansehen (Auswahl aller Ranglisten, read-only)
- **3.11**: Erstellung Punkte-System

4. Unit-Tests: Discord-Bot Tests refaktorisieren

5. Pipelines
- **5.1**: Dokumentation
- **5.2**: Backend
- **5.3**: Discord-Bot
- **5.4**: Dashboard

Optionale Features, falls Zeit vorhanden:

- **Backend:**
  - Bidirektionales Event-Sync
  - Gamification-Logik (Punkteablauf, Erinnerung Jahresbeitrag / Aufgaben, Hervorhebungen aktiver Mitglieder)
  - Logging
- **Discord-Bot:** Logs einsehen (/logs)
- **Admin-Dashboard:**
  - Vollständige Event-Verwaltung
  - Vollständige Belohnungs-Verwaltung
  - Vollständige Ranglisten-Verwaltung
  - Vereins-Analytics
  - E-Mail-Funktion (bei Vergessen des Passwortes)
  - Logs einsehen
  - Login und Darstellung der Daten für Mitglieder
- **Weitere Tests**
- **Vollständiges 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 wird verwendet, um erste Systeme zu testen, auf denen dann später aufgebaut wird.

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 Vanessa Alves, welche die Bachelorarbeit durchführt, an der Präsentation und am Poster alleine.

9.4.6 Meilensteine

Die folgenden Meilensteine dienen als Hilfe, um die definierten Ziele des Projektes zu erreichen. Sie gelten als erfüllt, sobald die einzelnen Komponenten durch das Team in Jira als erledigt gekennzeichnet wurden.

M1: Projekt Initiierung

- Der Projektplan ist definiert.
- Die Risiken sind definiert.
- Das MVP ist definiert.
- Der Long-Term-Plan ist definiert.

M2: Prototyp

- Die Requirements sind definiert.
- Die Architektur ist definiert.
- Mockups für das Admin-Dashboard sind erstellt.
- Das Gamificationkonzept, Datenschutzkonzept und Testkonzept ist erstellt.
- Der erste Prototyp für das Admin-Dashboard ist erstellt.

M3: Architektur stabilisiert

- Refactoring abgeschlossen
- Backend API, Discord-Bot und Admin-Dashboard laufen

M4: MVP Implementation abgeschlossen

- Gamification-Basis funktioniert mit der neuen Architektur
- Bot-Kommandos und Dashboard-Views sind umgesetzt
- Das MVP wird erreicht.

M5: Abgabe

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

M6: Präsentation

- Die Simulation wurde durchgeführt und ausgewertet.
- 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	R9, R11		
2 - Signifikant			R4, R8, R13	R1, R12	
1 - Gering		R5		R6, R14	

Tab. 47: Risiko Matrix

ID	R1
Risiko	Lernkurve
Kommentar	Der Einsatz neuer Technologien (z. B. Frameworks, CI/CD, Web-Technologien) führt zu einer erhöhten Einarbeitungszeit.
Prävention	Frühzeitige Prototypen reduzieren die Einarbeitungszeit. Wissen wird im Team dokumentiert und geteilt.
Korrektur	Komplexe Features werden in kleinere Arbeitspakete unterteilt. Bei Problemen erfolgt gegenseitige Unterstützung im Team.

Tab. 48: Beschreibung Risiko 1

ID	R2
Risiko	Probleme bei Discord oder der neuen Plattformen
Kommentar	Discord oder alternative Plattformen (z. B. MS Teams, Matrix) können temporär nicht verfügbar sein.
Prävention	Evaluierung alternativer Plattformen und Absicherung kritischer Funktionen über die Web-Applikation.
Korrektur	Risiko muss weitgehend akzeptiert werden. Bei längeren Ausfällen wird die Nutzung der Web-Applikation verstärkt.

Tab. 49: Beschreibung Risiko 2

ID	R3
Risiko	Ressourcen limitiert
Kommentar	Ein Teammitglied fällt temporär oder dauerhaft aus.
Prävention	Alle Arbeitsschritte und Entscheidungen werden systematisch dokumentiert, um Wissenstransfer sicherzustellen.
Korrektur	Aufgaben werden durch verbleibende Teammitglieder übernommen. Fokus liegt auf priorisierten Features, um Projektziele trotz reduzierter Kapazität zu erreichen.

Tab. 50: Beschreibung Risiko 3

ID	R4
Risiko	Scope Creep
Kommentar	Anforderungen werden unklar definiert oder nachträglich erweitert, was den Projektumfang vergrössert.
Prävention	Detaillierte Anforderungsdefinition und Scope-Management im Rahmen der Meetings.
Korrektur	Nachträglich eingebrachte oder optionale Anforderungen mit niedriger Priorität werden in spätere Phasen verschoben oder gestrichen.

Tab. 51: Beschreibung Risiko 4

ID	R5
Risiko	Kompatibilitätsprobleme
Kommentar	Unterschiedliche Versionen von Tools, Bibliotheken oder Frameworks können zu Integrationsproblemen führen.
Prävention	Einheitliche Dokumentation der eingesetzten Versionen und Verwendung von Lockfiles (z. B. package-lock.json).
Korrektur	Versionskonflikte werden durch Abstimmung im Team gelöst, eine einheitliche Version wird verbindlich definiert.

Tab. 52: Beschreibung Risiko 5

ID	R6
Risiko	Zeiteinschätzung
Kommentar	Arbeitsaufwände werden zu optimistisch oder pessimistisch eingeschätzt.
Prävention	Iterative Abschätzung in den wöchentlichen Meetings sowie Nutzung von Erfahrungswerten und Vergleich mit vorherigen Projekten.
Korrektur	Optional geplante Features werden verschoben oder gestrichen, um Kernziele nicht zu gefährden.

Tab. 53: Beschreibung Risiko 6

ID	R7
Risiko	Änderungen in der Discord API oder andere Technologien
Kommentar	Fundamentale Änderungen in der Discord API oder anderen zentralen Technologien (z. B. Datenbank, Framework).
Prävention	Keine Verwendung von Funktionen, die als „deprecated“ markiert sind und Monitoring von Release Notes.
Korrektur	Anpassung des Codes und Refactoring bei API-Änderungen, auch wenn zusätzlicher Aufwand entsteht.

Tab. 54: Beschreibung Risiko 7

ID	R8
Risiko	Der bestehende Code ist stark an Discord gekoppelt, was die Entwicklung einer Web-App erschwert.
Kommentar	Der bisherige Code ist nicht komplett von Discord entkoppelt.
Prävention	Frühzeitiges Refactoring und Einführung einer modularen Architektur.
Korrektur	Falls Aufwand höher als geplant, Anpassung des Zeitplans und Reduktion optionaler Features.

Tab. 55: Beschreibung Risiko 8

ID	R9
Risiko	Dashboard ist nicht benutzerfreundlich
Kommentar	Dashboard erfüllen nicht die Erwartungen der Mitglieder/Vorstände hinsichtlich Usability.
Prävention	Entwicklung mit Usability-Tests und Einbindung von Stakeholdern.
Korrektur	Anpassung der Benutzeroberfläche und Interaktionskonzepte anhand von Feedback.

Tab. 56: Beschreibung Risiko 9

ID	R10
Risiko	Manipulation Gamification-System
Kommentar	Mitglieder könnten das Punktesystem missbrauchen (z. B. durch Mehrfachaktionen).
Prävention	Definition klarer Regeln, technische Prüfungen gegen Mehrfacheinträge, Monitoring verdächtiger Aktivitäten durch den Vorstand.
Korrektur	Rücksetzen oder Anpassung von Punkten bei Missbrauch und Anpassung der Regeln.

Tab. 57: Beschreibung Risiko 10

ID	R11
Risiko	Datenschutz
Kommentar	Potenzielle Risiken im Umgang mit personenbezogenen Daten.
Prävention	Erstellung eines Datenschutzkonzepts in Anlehnung an DSGVO und Schweizer Datenschutzgesetz sowie die Minimierung der erhobenen Daten.
Korrektur	Einholung von Einverständniserklärungen.

Tab. 58: Beschreibung Risiko 11

ID	R12
Risiko	Stakeholder-Anforderungen nicht berücksichtigt
Kommentar	Feedback von Mitgliedern oder Vorstand wird nicht ausreichend eingebunden.
Prävention	Priorisierung der Anforderungen, Usability-Tests mit realen Mitgliedern.
Korrektur	Anpassungen anhand von Feedback, Verschiebung von Features niedriger Priorität.

Tab. 59: Beschreibung Risiko 12

ID	R13
Risiko	Plattform-Integration komplexer als erwartet
Kommentar	Die geplante Entkopplung auf Dashboard oder anderen Plattformen verursacht höheren Aufwand oder technische Probleme.
Prävention	Proof-of-Concept frühzeitig erstellen, modulare Architektur.
Korrektur	Fokus auf Admin-Dashboard, weitere Plattformintegration in spätere Phase verschieben.

Tab. 60: Beschreibung Risiko 13

ID	R14
Risiko	Hosting oder CI/CD Ausfall
Kommentar	Server oder CI/CD-Pipeline fallen aus und blockieren Entwicklung oder Betrieb.
Prävention	Monitoring und automatisierte Backups, klare Verantwortlichkeiten im Team.
Korrektur	Deployment auf alternative Cloud-Umgebung oder manuelles Deployment als Notfalllösung.

Tab. 61: Beschreibung Risiko 14

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.

9.6 Auswirkungen des Refactorings auf die Planung

Der ursprüngliche Long-Term-Plan (vgl. Abschnitt 9.4) ging davon aus, dass das Vorprojekt eine weitgehend tragfähige Grundlage für die Weiterentwicklung bilden würde. Insbesondere wurde angenommen, dass die bestehenden Komponenten Mitgliederverwaltung, Kalender-Synchronisation, Discord-Bot sowie Gamification nur erweitert und nicht grundlegend neu strukturiert werden müssten. Auf dieser Basis wurden die Meilensteine M3 (Erreichung des MVP) und M4 (Fertigstellung der Implementation) zeitlich ambitioniert geplant, da nur ein moderater technischer Ausbau erwartet wurde (vgl. Abschnitt 9.4.6).

Wie im Kapitel Architekturelles Refactoring (vgl. Abschnitt 7.2) ausführlich erläutert, zeigte sich zu Beginn der Construction-Phase, dass das Vorprojekt trotz erster Strukturierungsansätze wesentliche architektonische Defizite aufwies. Besonders problematisch war das Fehlen einer eigenständigen API-Schicht, was dazu führte, dass der Discord-Bot direkt mit Datenbankmodellen interagierte und keinerlei saubere Trennung zwischen Präsentations-, Anwendungs- und Infrastrukturschicht bestand. Zudem fehlte eine gemeinsame Shared-Library, wodurch Typen, Validierungen und Fehlermeldungen mehrfach und inkonsistent implementiert waren. Die Kopplung zwischen Bot, Backend und Datenbank war so eng, dass eine flexible Erweiterung oder die Einführung einer zweiten Plattform, wie beispielsweise das Admin-Dashboard, nicht möglich war.

Aufgrund dieser strukturellen Einschränkungen war die ursprünglich geplante Multi-App-Architektur aus Backend, Discord-Bot, Dashboard und Shared-Package auf Basis des bestehenden Codes nicht realisierbar. Ein umfassendes Refactoring war daher unvermeidlich, um eine saubere technische Grundlage zu schaffen und die geplanten Funktionen nachhaltig und wartbar implementieren zu können.

9.6.1 Teamverteilung und Einfluss auf den Projektverlauf

Die Aufgaben wurden im Team strategisch verteilt. Das Teammitglied Vanessa übernahm das Refactoring des Backends und des Bots, während Joel gleichzeitig das Admin-Dashboard entwickelte. Diese Aufteilung war bewusst gewählt, da Vanessa das Vorprojekt erstellt hatte und sich somit besonders gut mit den Bestandsstrukturen auskannte. Es wurde erwartet, dass sie das Refactoring durch ihr Vorwissen effizient durchführen könnte, während Joel unabhängig davon das Dashboard neu aufbauen kann.

In der Praxis zeigte sich jedoch, dass der tatsächliche Aufwand deutlich höher war als prognostiziert. Die Entkopplung des Bots, der vollständige Neubau des API-Layers, die Migration aller Commands und der Neuaufbau der Domain-Logik waren komplexer und zeitintensiver als erwartet. Dennoch war die Aufgabenaufteilung effizient. Während Vanessa die technische Basis modernisierte, konnte Joel das Dashboard parallel vorantreiben, sodass trotz der Herausforderungen kontinuierlicher Fortschritt erzielt wurde.

Dieser Verlauf entspricht den im Risikomanagement identifizierten Risiken R6 (Zeiteinschätzung) , R8 (Kopplung des alten Systems) und R13 (komplexere Plattform-Integration) , die zwar als moderat eingeschätzt wurden, sich jedoch als planungsrelevant herausstellten.

9.6.2 Auswirkungen auf Zeitplan und Meilensteine

Der hohe technische Aufwand führte dazu, dass ein Grossteil der Wochen 7-10 für strukturelle Arbeiten verwendet werden musste. Die Funktionsentwicklung, wie sie im ursprünglichen Long-Term-Plan (vgl. Anhang Abschnitt 12) und den Meilensteinen (vgl. Anhang Abschnitt 12.0.1) für diese Phase vorgesehen war, konnte daher nicht parallel erfolgen. Erst gegen Ende von Woche 10 erreichte das System wieder einen stabilen Zustand, der die weitere Implementierung erlaubte.

Trotz dieser Verzögerung konnten wesentliche Kernkomponenten wie Bot-Basiskommandos, Mitglieder- und Rollenverwaltung, Dashboard-Login und erste Gamification-Ansichten fertiggestellt werden. Die Verschiebung betraf allerdings direkt die Meilensteine M3 und M4, da der ursprünglich geplante Funktionsumfang in diesem Zeitraum nicht vollständig erreicht werden konnte.

Die nachfolgende Tabelle zeigt den Stand der in Woche 10 umgesetzten und noch nicht erledigten Funktionen. Diese Abweichung machte eine umfassende Anpassung des Long-Term-Plans notwendig.

Umgesetzt	• Discord-Bot: /join, /leave, /events, /linkcalendar, /synccalendar • Dashboard: Login, Mitgliederverwaltung, Rollenverwaltung, Basisansichten für Aufgaben, Events, Belohnungen und Ranglisten • Backend: vollständige neue Architektur mit Calendar-Sync, Domain-Modellen, DB-Services und API-Layer
Nicht umgesetzt	• Gamification-Logik • Rollen- und Benutzersynchronisation • Leaderboards und Rewards • erweiterte Dashboard-Verwaltung • CI/CD, Tests, E-Mail-Funktionalität

Tab. 62: Beschreibung der umgesetzten bzw. noch nicht umgesetzten Funktionen in Woche 10

Um die Meilensteine trotz der architektonischen Notwendigkeiten zu erreichen, wurde eine realistische Priorisierung vorgenommen. Der Fokus liegt nun ausschliesslich auf MVP-relevanten Funktionen :

- **Discord-Bot**: Punktevergabe, Aufgabenabschluss, Sync-Befehle, Ranglisten-Erstellung und das Speichern von Spenden
- **Admin-Dashboard**: Anzeige aller Transaktionen, Ranglisten sowie das Erstellen des Punktesystems
- **Backend API**: Gamification-Minimalversion sowie Rollen- / User-Sync
- **Weiteres**: einfache CI und grundlegende Tests

Die nachfolgenden Funktionen werden bewusst verschoben oder gestrichen, da sie nicht MVP-relevant sind:

- Event-, Belohnungs- und Ranglistenverwaltung im Dashboard
- Vereins-Analytics
- Logging der Aktionen
- E-Mail Passwort-Reset
- Gamification Elemente (Punkteablauf, Jahresbeitrags-Reminder)
- Bidirektionaler Event-Sync
- Automatisierte Zahlungsbestätigung von Mitgliederbeiträgen
- Vollständige CI/CD-Pipeline
- Mitglieder-Frontend
- Erweiterte Testabdeckung

Diese Reduktion entspricht dem im Risikomanagement definierten Umgang mit Scope Creep (R4) und stellt sicher, dass das Projekt weiterhin zielgerichtet in Richtung MVP voranschreiten kann.

Durch die gezielte Fokussierung auf MVP-relevante Funktionen, die klare Priorisierung und die effiziente Teamaufteilung bleibt der Projektverlauf trotz der Herausforderungen planbar. Die überarbeiteten Planungsentscheidungen stellen sicher, dass die zentralen Meilensteine termingerecht erreicht werden können und die Simulation in der Transition-Phase erfolgreich vorbereitet ist.

10 Arbeitszeitnachweis

In diesem Kapitel wird eine Übersicht des zeitlichen Aufwands gegeben, der während der Umsetzung des Projekts angefallen ist. Zudem wird aufgezeigt, wie viel Zeit pro Projektphase aufgewendet wurde.

10.1 Zeit-pro-Person

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

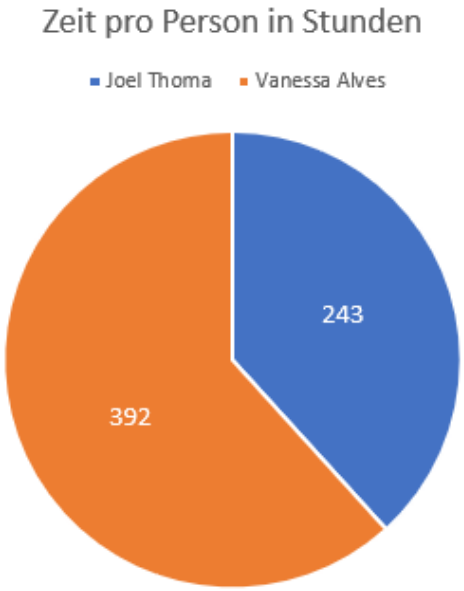


Abb. 25: Zeit pro Person

10.2 Zeit pro Phase

Das folgende Diagramm zeigt die aufgewendete Zeit pro Projektphase sowie für die Meetings. ClansCore basiert auf einer Elaboration-Phase, in der viel Zeit investiert wurde, um ein durchdachtes Projekt umzusetzen. Die Construction-Phase ist die längste Phase und nahm insgesamt den grössten Zeitaufwand in Anspruch.

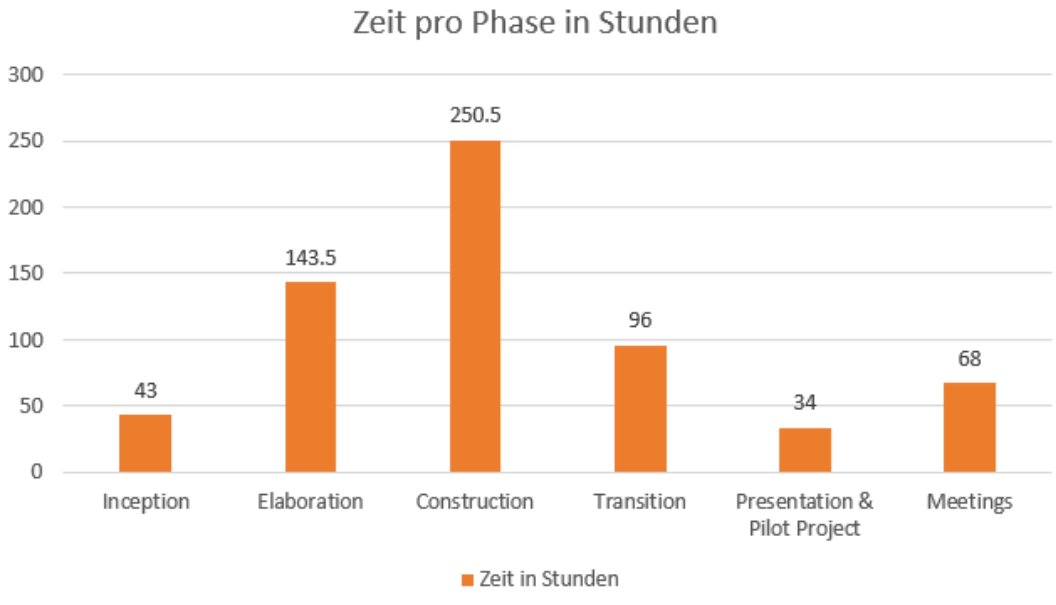


Abb. 26: Zeit pro Phase

11 Tooling und Technologie-Entscheidungen

Dieses Kapitel beschreibt die im Rahmen der Bachelorarbeit eingesetzten Werkzeuge, Technologien und Entwicklungsprozesse. Es begründet die wesentlichen Entscheidungen hinsichtlich Projektorganisation, Dokumentation, Code-Qualität und Deployment. Grundlage bilden die Erfahrungen aus dem Vorprojekt , welche in dieser Arbeit systematisch weiterentwickelt werden.

11.1 Jira

Die Zusammenarbeit und Aufgabenverteilung wird mit Jira[3] 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).
- **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.

11.2 Teams

Für die interne Kommunikation, spontane Absprachen und die Planung von ausserordentlichen Meetings wird Microsoft Teams[4] verwendet. Es dient als zentraler Kommunikationskanal und Speicherort für geteilte Dokumente. Damit wird ein kontinuierlicher Informationsfluss sichergestellt, was insbesondere bei verteilter Arbeit im Rahmen der Bachelorarbeit entscheidend ist.

11.3 GitHub

Das Vorprojekt wurde zu Beginn der Arbeit von GitLab[5] auf GitHub[6] migriert, um die Entwicklungs- und Veröffentlichungsprozesse langfristig unabhängig von der OST-internen Infrastruktur zu gestalten. Die zuvor auf der OST-GitLab gehostete Umgebung war nur innerhalb der Hochschule zugänglich, wodurch externe Zusammenarbeit und eine spätere Öffnung des Projekts erschwert wurden. GitHub bietet im Gegensatz dazu eine öffentlich zugängliche, weit verbreitete Plattform, die sich in der Open-Source-Community etabliert hat. Durch die Nutzung von GitHub Organisations und integrierten CI/CD-Workflows (GitHub Actions) kann die Projektstruktur klar gegliedert und zukünftige Erweiterungen effizient verwaltet werden. Neben diesen Vorteilen diente der Wechsel ausserdem dem Erkenntnisgewinn für das Team.

[3] https://www.atlassian.com/software/jira
[4] https://www.microsoft.com/de-ch/microsoft-teams/log-in?market=ch
[5] https://gitlab.ost.ch/
[6] https://github.com/

Für die Projektstruktur wurde eine eigene Organisation namens „ClansCore“[7] erstellt. Innerhalb dieser Organisation befinden sich die nachfolgenden Repositories:

Repository	Beschreibung
clanscore[8]	Der Quellcode der Applikation.
clanscore-dok[9]	Die Projekt-Dokumentation.
clanscore-dok-pub[10]	Die automatisch generierten PDF-Versionen der Dokumentation, Sitzungs-Protokolle sowie Online-Doku von ClansCore, welche über GitHub Pages veröffentlicht werden (vgl. Abschnitt 11.4.2).

Tab. 63: Übersicht der Repositories innerhalb der GitHub-Organisation ClansCore

Während der Bearbeitung dieser Arbeit sind die Haupt-Repositories privat gestellt, um den Entwicklungsprozess geschützt zu halten. Die Organisation selbst ist jedoch öffentlich, wodurch Transparenz, Auffindbarkeit und spätere Zusammenarbeit gewährleistet bleiben.

11.3.1 GitHub Guidelines

1. Auf dem Dokumentations-Repository hat jedes Teammitglied seinen eigenen Branch im Format „dev-Initialen“ (z.B. dev-jt)
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 Dokumentation

Die Projektdokumentation wird vollständig mit Typst[11] erstellt. Die Entscheidung für Typst erfolgte aufgrund der einfacheren Syntax im Vergleich zu LaTeX[12], der nativen Git-Integration und der bereits im Vorprojekt begonnenen Typst-Vorlage.

11.4.1 Dokumentation Guidelines

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 snake_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.

[7] https://github.com/ClansCore
[8] https://github.com/ClansCore/clanscore
[9] https://github.com/ClansCore/clanscore-doc
[10] https://github.com/ClansCore/clanscore-doc-pub
[11] https://typst.app/
[12] https://www.latex-project.org/

11.4.2 Dokumentation Deployment

Im Rahmen dieser Arbeit wurde eine automatisierte Dokumentations-Pipeline implementiert, welche die kontinuierliche Erstellung und Veröffentlichung der Projektdokumentation gewährleistet. Ziel war es, den zuvor im Vorprojekt auf GitLab basierenden CI/CD-Prozess auf die GitHub-Infrastruktur zu migrieren und an die Anforderungen des neuen Projektsetups anzupassen (vgl. Abschnitt 11.3).

Die Pipeline dient der automatischen Kompilierung und Veröffentlichung der Typst-Dokumentation, bestehend aus dem Hauptbericht und den Sitzungsprotokollen. Dabei werden sämtliche Schritte, vom Erstellen der PDF-Dateien bis zur Bereitstellung auf GitHub Pages, automatisiert ausgeführt. Da GitHub Pages für private Repositories in der kostenfreien Version nicht verfügbar ist, wurde eine Split-Repository-Strategie umgesetzt. Das private Repository (clanscore-doc) enthält den vollständigen Typst-Quellcode, während ein separates öffentliches Ziel-Repository (clanscore-doc-pub) ausschliesslich die generierten Ausgabedateien beherbergt.

Die Pipeline wurde mittels GitHub Actions realisiert und befindet sich im privaten Repository. Der Workflow wird bei jedem Push auf den Main-Branch ausgelöst. Für die Authentifizierung und das Deployment wurde eine eigene GitHub App innerhalb der Organisation ClansCore erstellt. Diese App besitzt ausschliesslich die für das Deployment benötigten Berechtigungen auf das Ziel-Repository. Das App-Zugriffstoken wird zur Laufzeit über eine Action generiert und anschliessend verwendet, um den Inhalt des public/-Ordners in den Main-Branch des öffentlichen Repositories zu pushen.

Die GitHub Pages des Ziel-Repositories wurden so konfiguriert, dass der Main-Branch automatisch als statische Website veröffentlicht wird. Damit wird bei jeder Änderung im privaten Repository die Dokumentation neu erzeugt und unmittelbar online verfügbar gemacht.

Dieser Ansatz stellt eine klare und sichere CI/CD-Lösung dar. Der Quellcode und interne Inhalte bleiben privat, während die erzeugten Enddokumente öffentlich zugänglich sind. Durch die Verwendung einer GitHub App wird zudem auf persönliche Zugriffstokens verzichtet, was eine nachhaltige und wartungsarme Authentifizierung gewährleistet.

11.5 Code und Entwicklung

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

11.5.1 Code Guidelines

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 nur wo es nötig ist, der Code sollte selbsterklärend geschrieben werden.
11. Verwende für gleiche Konzepte die gleichen Wörter.
12. Überprüfe Änderungen vor dem pushen mit Prettier und ESLint.

11.5.2 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.

11.5.3 Hosting

Um ClansCore dauerhaft online zu halten und beispielsweise auf Discord-Ereignisse zu reagieren oder zuverlässigen Zugriff auf das Admin-Dashboard zu gewährleisten, ist ein geeignetes Hosting erforderlich. Während dem Vorprojekt wurde der Discord-Bot auf einem Linux-Server des Vereins EGSwiss gehostet, durch manuelle Deployments über den Vereinsinternen Serveradmin. Da dieser Serveradmin zum Zeitpunkt der Arbeit anderweitig beschäftigt war sowie die Teammitglieder keinen direkten Zugriff auf den Server hatten, fiel die Entscheidung auf einen temporären virtuellen Server der Fachhochschule OST. Dieser läuft in einer stabilen Umgebung, ist kostenfrei und garantiert dem Team vollen Zugriff für die Einführung eines automatischen Deployments (vgl. Abschnitt 11.5.4).

11.5.4 Code Deployment

Im Vorprojekt wurde das Deployment mit Docker manuell durch eine Drittperson auf einem dedizierten Linux-Server durchgeführt. Da ein manuelles Deployment fehleranfällig, langsam und ineffizient ist, soll ein automatisiertes Deployment umgesetzt werden (AutoMQ-Contributors, 2025) .

Mit GitHub Actions kann das Deployment automatisiert und sichergestellt werden, dass bei einem Push auf den Main-Branch die Änderungen in die Produktivumgebung übernommen werden. Docker wird für das Containermanagement verwendet, da es eine zuverlässige und plattformunabhängige Bereitstellung der Anwendung ermöglicht (Docker-Contributors, 2025) und das Team bereits Erfahrung mit Docker hat.

Die tatsächliche Umsetzung des Deployments von ClansCore wird im Anhang beschrieben. (vgl. Abschnitt 15)

11.6 Datenbank

Im Vorprojekt wurde eine MongoDB-Datenbank verwendet. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank , die Daten in JSON-ähnlichen Dokumenten speichert (MongoDB-Contributors, 2025b) . Eine zentrale Funktion von MongoDB ist die horizontale Skalierbarkeit, was insbesondere bei einer steigenden Nutzung des Bots einen entscheidenden Vorteil darstellt. Die Erweiterungen, welche für ClansCore umgesetzt wurden, erforderten keine fundamentalen Änderungen an der Datenbank, sodass kein Anlass bestand, eine alternative Datenbanklösung in Erwägung zu ziehen.

11.7 Programmiersprachen und Frameworks

ClansCore baut auf einem bereits im Vorprojekt entwickelten Discord-Bot auf. Dieser wurde in TypeScript programmiert und verwendet die Bibliothek Discord.js, welche als die am weitesten verbreitete und aktiv gepflegte Schnittstelle zur Discord API gilt (Discord.js-Contributors, 2025a) . Die Bibliothek abstrahiert die Kommunikation über REST- und WebSocket-Schnittstellen und ermöglicht dadurch eine effiziente und stabile Implementierung der Bot-Funktionalitäten.

Discord.js wird fortlaufend weiterentwickelt und erhält regelmässig sicherheits- und funktionsbezogene Updates (Discord.js-Contributors, 2025b) . Durch die positiven Erfahrungen aus dem Vorprojekt, den aktiven Community-Support und die hohe Kompatibilität mit TypeScript, wird Discord.js weiterhin als technologische Basis verwendet. Diese Entscheidung stellt sicher, dass bestehende Codebestandteile übernommen und nachhaltig weiterentwickelt werden können, ohne eine kostspielige Migration auf eine alternative Bibliothek durchführen zu müssen.

Für die Entwicklung des Admin-Dashboards standen verschiedene moderne Web-Technologien zur Auswahl. Da ein Framework den Entwicklungsprozess strukturiert, die Wiederverwendbarkeit erhöht sowie die Wartung erleichtert, wurde ein komponentenbasiertes Framework eingesetzt. Die Auswahl erfolgte anhand von Kriterien wie Verbreitung, Funktionsumfang, Zukunftssicherheit und vorhandene Teamerfahrung.

Web-Technologie	Beschreibung
React	Eine weit verbreitete JavaScript-Bibliothek für User Interfaces mit Millionen von Anwendern weltweit. React ist komponentenbasiert, kombiniert Logik und Markup in derselben Datei und profitiert von einer grossen Entwickler-Community. Die Flexibilität der Bibliothek ermöglicht unterschiedliche Architekturansätze, erfordert jedoch zusätzliche Tools für Routing, State-Management und Build-Prozesse. (React-Contributors, 2025)
Angular	Ein umfassendes Web-Framework, das eine vollständige Entwicklungsumgebung für skalierbare Web-Applikationen bietet. Angular wird von Google gepflegt, unterstützt TypeScript nativ und beinhaltet bereits Funktionen wie Routing, Dependency Injection und Formularverwaltung. Es ermöglicht eine klare Strukturierung und eine nachhaltige Code-Architektur. (Angular-Contributors, 2025b)
Vue.js	Ein progressives JavaScript-Framework zur Entwicklung von User Interfaces. Vue kombiniert einfache Syntax mit hoher Flexibilität und eignet sich sowohl für kleine als auch komplexe Anwendungen. Es besitzt eine engagierte Community, wird jedoch primär von Einzelentwicklern und kleineren Teams gepflegt. (Vue.js-Contributors, 2025)

Tab. 64: Beschreibung der evaluierten Web-Technologien

Nach einer systematischen Bewertung wurde entschieden, das Admin-Dashboard mit Angular zu entwickeln. Diese Entscheidung beruht auf mehreren Faktoren:

- **TypeScript-Integration**: Angular ist vollständig in TypeScript implementiert, was eine einheitliche Codebasis zwischen Frontend und Backend ermöglicht. Dadurch können Typdefinitionen und Schnittstellen gemeinsam genutzt werden, was die Konsistenz und Fehlersicherheit erhöht.
- **Vollständige Framework-Struktur**: Im Gegensatz zu React, das primär eine UI-Bibliothek darstellt, bietet Angular ein integriertes Framework mit klaren Architekturvorgaben, standardisierten Projektstrukturen und eingebautem Tooling (z. B. Testing, Routing, Dependency Injection). Dies reduziert den Integrationsaufwand externer Bibliotheken und erhöht die Wartbarkeit.
- **Skalierbarkeit und Nachhaltigkeit**: Angular ist auf gross angelegte, modular aufgebaute Anwendungen ausgelegt und gewährleistet damit eine langfristig stabile Codebasis.
- **Teamkompetenz**: Beide Teammitglieder verfügten bereits über Erfahrung im Umgang mit Angular, was den Einarbeitungsaufwand erheblich reduzierte und einen produktiven Entwicklungsstart ermöglichte. Mit React und Vue.js liegen hingegen keine oder nur geringe praktische Erfahrungen vor.

Zusammenfassend wurde Angular aufgrund der klaren Struktur, nativen TypeScript-Unterstützung, hohen Skalierbarkeit und der bestehenden Teamexpertise als geeignetstes Framework für die Umsetzung des Admin-Dashboards bewertet. Diese Wahl stellt sicher, dass das Dashboard langfristig wartbar bleibt und sich konsistent in die bestehende ClansCore-Systemarchitektur integriert.

11.8 UI-Bibliothek

Bibliothekenname	Beschreibung
Angular Material	Angular Material ist eine von der Angular-Entwicklergemeinschaft bereitgestellte Komponentenbibliothek, die auf den Richtlinien des Material Designs basiert. Sie bietet qualitativ hochwertige, internationalisierte und barrierefreie UI-Komponenten, die durch umfangreiche Tests auf Leistung und Zuverlässigkeit geprüft sind. (Material-Contributors, 2025)
PrimeNG	PrimeNG ist eine umfassende UI-Komponentenbibliothek für Angular, die eine Vielzahl an individuell anpassbaren und funktionsreichen Komponenten bereitstellt. Mit über 80 verfügbaren UI-Elementen unterstützt sie Entwicklerinnen und Entwickler bei der effizienten Umsetzung moderner Webanwendungen. (PrimeNg-Contributors, 2025)
NG-Zorro	NG-Zorro ist eine auf dem Ant Design basierende UI-Komponentenbibliothek für Angular, die speziell für den Unternehmenseinsatz entwickelt wurde. Sie stellt über 70 hochwertige, in TypeScript implementierte Komponenten bereit. (NG-ZORRO-Contributors, 2025)

Tab. 65: Bibliotheken für das User Interface

Im Unterschied zu PrimeNG und NG-Zorro fiel die Wahl auf Angular Material als UI-Komponentenbibliothek, da es eine besonders enge Integration in das Angular-Framework bietet und direkt vom Angular-Entwicklungsteam gepflegt wird. Dadurch ist eine hohe Kompatibilität mit aktuellen Angular-Versionen sowie eine langfristige Wartung und Stabilität gewährleistet.

Zudem überzeugt Angular Material durch seine klare Ausrichtung auf die Material-Design-Richtlinien, wodurch ein konsistentes und modernes Benutzererlebnis sichergestellt wird. Die Komponenten sind qualitativ hochwertig, barrierefrei und umfassend getestet, was insbesondere für produktive und langlebige Anwendungen von Vorteil ist.

11.9 KI-Tools

KI-Tools wurden 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, primär für technische Beschreibungen, Kapitelüberschriften und sprachliche Korrektheit.

Die KI wurde als Assistenzwerkzeug eingesetzt und diente nicht zur automatisierten Generierung vollständiger Inhalte. Alle durch die KI erzeugten Vorschläge wurden kritisch geprüft und bei Bedarf angepasst.

Folgende KI-Tools wurden verwendet:

- OpenAI ChatGPT[13]
- Cursor[14]
- Google Gemini[15]
- GitHub Copilot[16]

[13] https://chatgpt.com/
[14] https://cursor.com/features
[15] https://gemini.google.com
[16] https://github.com/features/copilot

Kapitel III
Glossar und Abkürzungsverzeichnis

ClansCore	ClansCore ist das Gesamt-System und der Produktname für die Erweiterung des Discord-Bots aus dem Vorprojekt . Es ist das End-Produkt dieser Arbeit und verbindet sämtliche Teil-Komponenten wie Backend, Datenbank, Discord-Bot, Admin-Dashboard und weitere Plattformen.
EGSwiss	Ein Schweizer Gaming Verein, wobei das Teammitglied Vanessa Alves Präsidentin ist und daher die nötige Expertise für die Vorarbeit und aktuelle Weiterentwicklung mitbringt sowie Erfahrungen aus dem Vereinsalltag einfliessen lässt.
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.
Representational State Transfer (REST)	Representational State Transfer (REST) ist eine Architekturform für webbasierte Schnittstellen, bei der Daten über standardisierte HTTP-Methoden (z. B. GET, POST, PATCH, DELETE) ausgetauscht werden. REST-APIs sind zustandslos, d. h. jede Anfrage enthält alle notwendigen Informationen, um sie unabhängig von früheren Interaktionen zu verarbeiten.
WebSocket-Schnittstelle	WebSockets sind ein bidirektionales Kommunikationsprotokoll (RFC 6455), welches eine dauerhafte Verbindung zwischen Client und Server ermöglicht. Im Gegensatz zu REST, das auf einzelnen HTTP-Anfragen basiert, erlaubt WebSocket eine kontinuierliche Übertragung von Daten in Echtzeit.
Role-Based Access Control (RBAC)	Role-Based Access Control (RBAC) ist ein Sicherheitskonzept, bei dem der Zugriff auf Ressourcen anhand von Rollen vergeben wird. Nutzende erhalten dabei bestimmte Rollen, die jeweils unterschiedliche Berechtigungen für Funktionen und Daten gewähren. So können Admins granular steuern, wer auf welche Teile eines Systems zugreifen darf.
Zwei- / Multi-Faktor-Authentifizierung (2FA/MFA)	Zwei- oder Multi-Faktor-Authentifizierung (2FA/MFA) 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 beziehungsweise MFA erhöht die Sicherheit erheblich, da ein Angreifer beide Faktoren kennen oder besitzen müsste, um sich erfolgreich anzumelden.

SCRAM-SHA-256	Authentifizierungsverfahren gemäss RFC 7677[17], das nach dem Prinzip des Salted Challenge Response Authentication Mechanism (SCRAM) arbeitet. Passwörter werden dabei nicht im Klartext gespeichert oder übertragen, sondern mit einem zufälligen Salt und mehreren Iterationen nach dem Hash-Algorithmus SHA-256 gesichert. Während des Logins prüft der Server eine kryptografische Antwort (Challenge-Response), wodurch das Verfahren gegenüber Replay- und Wörterbuchangriffen geschützt ist. MongoDB verwendet SCRAM-SHA-256 als Standardmechanismus für Benutzeranmeldungen, um die Vertraulichkeit und Integrität der Authentifizierung zu gewährleisten.
Reverse Proxy	Vermittlungsinstanz zwischen Clients und internen Serverdiensten, die eingehende Anfragen entgegennimmt und an den entsprechenden Backend-Service weiterleitet. Dadurch werden interne Systeme nach aussen abgeschirmt, Lasten verteilt und zusätzliche Sicherheitsfunktionen wie TLS-Verschlüsselung, Zugriffskontrolle und Caching bereitgestellt. Reverse Proxys sind ein zentrales Element moderner Webarchitekturen und dienen der Trennung von öffentlicher und privater Netzwerkebene.
Virtual Private Network (VPN)	Ein Virtual Private Network (VPN) ist eine Technologie, die eine verschlüsselte Verbindung über ein öffentliches oder unsicheres Netzwerk ermöglicht. Durch die Nutzung eines VPNs wird der Datenverkehr zwischen Client und Server gesichert, wodurch Vertraulichkeit und Integrität der übertragenen Daten gewährleistet werden.
Systemhärtung (Hardening)	Massnahmen zur Reduktion der Angriffsfläche eines Systems durch Entfernen unnötiger Dienste, Einschränkung von Benutzerrechten und konsequente Sicherheitskonfiguration. Ziel ist es, die Widerstandsfähigkeit gegen Angriffe zu erhöhen und unautorisierte Zugriffe zu verhindern.
Bastion Host	Ein gehärteter Server , der als sicherer Zugangspunkt zu einem internen Netzwerk dient. Er vermittelt administrative Zugriffe über einen kontrollierten, MFA -geschützten Kanal und verhindert direkten Zugriff auf interne Systeme.
Time-based One-Time Password (TOTP)	Ein zeitbasiertes Einmalpasswort ist ein Verfahren zur Zwei-Faktor-Authentifizierung (2FA) , bei dem temporäre, nur wenige Sekunden gültige Passwörter generiert werden. Diese werden auf einem Authenticator-Gerät oder in einer App erstellt. Die Synchronisation erfolgt über eine gemeinsame geheime Basis und die aktuelle Zeit, was eine zusätzliche Sicherheitsebene beim Login-Prozess bietet.
Phishing	Phishing bezeichnet den Versuch, über gefälschte E-Mails, Websites oder Nachrichten vertrauliche Informationen wie Passwörter, Kreditkartendaten oder Zugangsdaten zu erlangen. Angreifer nutzen hierbei Social-Engineering-Methoden, um das Vertrauen von Nutzenden zu erschleichen. Ziel ist meist der unbefugte Zugriff auf Konten oder Systeme. Schulungen und Sensibilisierung helfen, Phishing-Angriffe zu erkennen und zu vermeiden.

[17] https://www.rfc-editor.org/rfc/rfc7677.html

CLOUD Act	Der Clarifying Lawful Overseas Use of Data Act (CLOUD Act) ist ein US-amerikanisches Gesetz von 2018, das US-Behörden den Zugriff auf Daten ermöglicht, die von US-Unternehmen gespeichert werden, auch wenn sich diese Daten physisch ausserhalb der USA befinden. Für in der Schweiz gehostete Systeme bedeutet dies, dass bei Nutzung von US-Cloud-Anbietern theoretisch ein ausländischer Datenzugriff möglich ist. Daher wird häufig auf Hosting in der Schweiz gesetzt, um die nationale Datenhoheit zu gewährleisten.
Zod-Schemas	Zod ist eine TypeScript-first Schema-Validierungsbibliothek, die zur Laufzeit und zur Compile-Zeit Typen validiert. Schemas definieren die erwartete Struktur und den Typ von Daten und ermöglichen eine sichere Validierung von Eingaben, beispielsweise bei API-Requests oder Webhook-Payloads.
Cron-Job	Ein Cron-Job ist ein zeitgesteuerter Prozess, der bestimmte Aufgaben zu festgelegten Zeiten oder Intervallen automatisch ausführt. Die Ausführung erfolgt unabhängig von Benutzerinteraktionen und ermöglicht die Automatisierung wiederkehrender Prozesse.
Domain-Driven Design (DDD)	Domain-Driven Design ist ein Softwareentwicklungsansatz, der den Fokus auf die Modellierung der Geschäftsdomäne legt. Die Architektur wird dabei in klar abgegrenzte Domänen (Bounded Contexts) unterteilt, wobei jede Domäne ihre eigenen Entitäten, Wertobjekte und Geschäftsregeln definiert. DDD unterstützt die Entwicklung komplexer Systeme durch eine gemeinsame Sprache (Ubiquitous Language) zwischen Entwicklern und Fachexperten.
NoSQL	NoSQL (Not Only SQL) bezeichnet Datenbanken, die nicht dem relationalen Datenmodell folgen. Im Gegensatz zu SQL-Datenbanken speichern NoSQL-Datenbanken Daten in verschiedenen Formaten wie Dokumenten, Key-Value-Paaren oder Graphen. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank, die Daten in JSON-ähnlichen Dokumenten speichert und eine hohe horizontale Skalierbarkeit bietet.
OpenID Connect (OIDC) / OAuth 2.0	OAuth 2.0 ist ein Autorisierungsprotokoll, das Anwendungen einen sicheren Zugriff auf Ressourcen ermöglicht, ohne Passwörter preiszugeben. OpenID Connect (OIDC) erweitert OAuth 2.0 um eine Authentifizierungsschicht und ermöglicht es, die Identität von Nutzenden zu verifizieren. Beide Protokolle werden häufig für Single-Sign-On (SSO) und sichere API-Zugriffe verwendet.

Kapitel IV
Literaturverzeichnis

Alves, V., & Schnider, M. (2025). *Discord-Bot für Vereine* [Studien- und Bachelorarbeit].

Angular-Contributors. (2025a,). *Testing*. https://angular.dev/guide/testing

Angular-Contributors. (2025b,). *What is Angular?*. https://angular.dev/overview

Arcane-Contributors. (2025,). *Arcane Discord Bot*. https://arcane.bot/

AutoMQ-Contributors. (2025,). *Common Issues with Manual Deployment*. https://www.automq.com/blog/fully-automated-deployment-vs-manual-deployment-comparison#common-issues-with-manual-deployment

Chou, Y.-k. (2015). *Actionable Gamification: Beyond Points, Badges and Leaderboards*. CreateSpace Independent Publishing Platform.

CISA. (2022). *Implementing Multi-Factor Authentication (MFA) for Remote Access and VPNs* [Technical report]. https://www.cisa.gov/sites/default/files/publications/CISA_MFA_Guidelines_Remote_Access_2022.pdf

ClubDesk-Contributors. (2025,). *Vereinssoftware für die einfache Vereinsverwaltung*. https://www.clubdesk.de/de/startseite

Discord-Contributors. (2025,). *Guild Scheduled Event*. https://discord.com/developers/docs/resources/guild-scheduled-event

Discord.js-Contributors. (2025b,). *Commits*. https://github.com/discordjs/discord.js/commits/main/

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

Docker-Contributors. (2025,). *Developers bring their ideas to life with Docker*. https://www.docker.com/why-docker/

Europäische-Union. (2016,). *VERORDNUNG (EU) 2016/679 DES EUROPÄISCHEN PARLAMENTS UND DES RATES*. https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32016R0679

madewithangular-Contributors. (2025,). *A showcase of web apps made with Angular*. https://www.madewithangular.com/sites

Martin, F. (2003,). *Optimistic Offline Lock*. https://martinfowler.com/eaaCatalog/optimisticOfflineLock.html

Material-Contributors, A. (2025,). *Angular Material - Material Design components for Angular*. https://material.angular.dev/

MEE6-Contributors. (2025,). *Statistics Channels Plugin*. https://wiki.mee6.xyz/en/plugins/statistics-channels

Microsoft-Contributors. (2025b,). *Planning for mandatory multifactor authentication for Azure and other admin portals*. https://learn.microsoft.com/en-us/entra/identity/authentication/concept-mandatory-multifactor-authentication?tabs=dotnet

Microsoft-Contributors. (2025a,). *Security at your organization: Multifactor authentication statistics*. https://learn.microsoft.com/en-us/partner-center/security/security-at-your-organization

MongoDB-Contributors. (2025d,). *Authentication and Authorization with OIDC/OAuth 2.0*. https://www.mongodb.com/docs/manual/core/oidc/security-oidc/

MongoDB-Contributors. (2025b,). *Introduction to MongoDB*. https://www.mongodb.com/docs/manual/introduction/

MongoDB-Contributors. (2025f,). *Role-Based Access Control in Self-Managed Deployments*. https://www.mongodb.com/docs/manual/core/authorization/

MongoDB-Contributors. (2025e,). *SCRAM*. https://www.mongodb.com/docs/manual/core/security-scram/

MongoDB-Contributors. (2025a,). *Transactions*. https://www.mongodb.com/docs/manual/core/transactions/

MongoDB-Contributors. (2025c,). *Use X.509 Certificates to Authenticate Clients on Self-Managed Deployments*. https://www.mongodb.com/docs/manual/tutorial/configure-x509-client-authentication/

Murugiah, S., & Karen, S. (2017). *Application Container Security Guide (NIST SP 800-190)* [Technical report]. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

NG-ZORRO-Contributors. (2025,). *Angular UI Component Library based on Ant Design*. https://github.com/NG-ZORRO/ng-zorro-antd

NgRx-Contributors. (2025,). *@ngrx/store*. https://ngrx.io/guide/store

NIST. (2020). *Security and Privacy Controls for Information Systems and Organizations (SP 800-53 Rev. 5)* [Technical report]. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

OWASP-Cheat-Sheets-Series-Contributors. (2025,). *Session Management Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

OWASP-Contributors. (2021,). *OWASP Top 10*. https://owasp.org/Top10/

Peters Dorian, R. R. M., Calvo Rafael A. (2018). Designing for Motivation, Engagement and Wellbeing in Digital Experience. *Frontiers in Psychology*, *9*. https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2018.00797

PrimeNg-Contributors. (2025,). *The Next-Gen UI Suite for Angular*. https://primeng.org/

React-Contributors. (2025,). *React*. https://react.dev/

Schweizerische-Eidgenossenschaft. (2020,). *Bundesgesetz über den Datenschutz (Datenschutzgesetz, DSG)*. https://www.fedlex.admin.ch/eli/cc/2022/491/de

U.S.-Department-of-Justice. (2019,). *The Purpose and Impact of the CLOUD Act - FAQs*. https://www.justice.gov/criminal/media/999616/dl?inline

Vue.js-Contributors. (2025,). *What is Vue?*. https://vuejs.org/guide/introduction

Wagner, J., & Pollard, B. (2025,). *Interaction to Next Paint (INP)*. https://web.dev/articles/inp?utm_source=lighthouse&utm_medium=devtools&hl=de

Weir, K. (2025,). *Self-determination theory: A quarter century of human motivation research*. https://www.apa.org/research-practice/conduct-research/self-determination-theory

WildApricot-Community. (2015,). *Gamify Member's Site Participation*. https://forums.wildapricot.com/forums/308932-wishlist/suggestions/10321731-gamify-member-s-site-participation

WildApricot-Contributors. (2025,). *Membership Management Software*. https://www.wildapricot.com/

Kapitel V
Abbildungsverzeichnis

Kapitel VI
Tabellenverzeichnis

Kapitel VII
Code-Listings

Kapitel VIII
Anhang

12 Long-Term Plan (Version 1.0 - Ursprüngliche Planung)

Der ursprüngliche Long-Term Plan wurde in 5 verschiedene Phasen aufgeteilt. Diese werden in der folgenden Tabelle genauer beschrieben. Optionale Aufgaben stehen in Klammern.

Phase	Woche	Datum	Ziele	Meilenstein
Inception	1	15.09 - 21.09.	Einrichtung, MVP definieren	
	2	22.09. - 28.09.	Projektplan, Risikomanagement, Long-Term-Plan	M1
Elaboration	3	29.09. - 05.10.	Requirements, Einleitung, Stand der Technik	
	4	06.10. - 12.10.	Interviews andere Vereine, Sicherheitskonzept	
	5	13.10. - 19.10.	Gamificationkonzept, Architektur, Wireframes	
	6	20.10. - 26.10.	Testkonzept, Prototyp Admin-Dashboard (Proof of Concept)	M2
Construction (Version 1.0)	7	27.10. - 02.11.	2.1, 2.2, 2.3, 7.1	
	8	03.11. - 09.11.	Refactoring, 1.1, (1.2)	
	9	10.11. - 16.11.	3.1, 3.2, 4.1, 4.3	
	10	17.11. - 23.11.	4.2, 4.4, 4.5, (5.1), (5.2)	M3
	11	24.11. - 30.11.	3.3, 4.6, 6.1, 6.2, 6.3, (6.4), Reminder-System, Vereins-Analytics	
	12	01.12. - 07.12.	7.2, 7.3, 7.4, (7.5), (8.1), (8.2), (8.3), Implementation beenden	M4
Transition	13	08.12. - 14.12.	Simulation vorbereiten, Dokumentation, Abschluss	
	14	15.12. - 19.12.	Fertigstellung, Abgabe Abstract und Dokumentation	M5
Presentation	15	20.12. - 28.12.	Simulation durchführen	
	16	29.12.2025 - 04.01.2026		
	17	05.01. - 09.01.	Ergebnisse auswerten, Präsentation und A0-Poster	M6

Tab. 66: Long-Term Plan mit ursprünglicher Construction-Phase (Version 1.0)

Die einzelnen Komponenten und Funktionen sind wie folgt definiert:

1. Backend
- **1.1**: Features ergänzen
- **1.2**: Auto. Zahlungsbestätigung über Bank

2. Datenbank
- **2.1**: Schema anpassen
- **2.2**: DB-Login verbessern (2FA)
- **2.3**: DB-Transactions / Rollbacks

3. Discord-Bot
- **3.1**: Features ergänzen
- **3.2**: Darstellung Events verbessern
- **3.3**: Logs einsehen

4. Admin-Dashboard
- **4.1**: Login erstellen
- **4.2**: Darstellung / Bearbeitung DB-Daten (Mitglieder, Rollen, Events)
- **4.3**: Darstellung / Bearbeitung / Anmeldung von Aufgaben
- **4.4**: Darstellung / Bearbeitung von Ranglisten
- **4.5**: Erstellung des Punktesystems (Punkte, Belohnungen, Jahresplanung)
- **4.6**: Logs einsehen

5. Neue Plattform (optional)
- **5.1**: Wahl der Plattform
- **5.2**: Implementierung gleicher Features wie Discord-Bot

6. Unit-Tests
- **6.1**: Backend
- **6.2**: Discord-Bot
- **6.3**: Dashboard
- **6.4**: Neue Plattform

7. Pipelines
- **7.1**: Dokumentation
- **7.2**: Backend
- **7.3**: Discord-Bot
- **7.4**: Dashboard
- **7.5**: Neue Plattform

8. Usability Tests (optional)
- **8.1**: Discord-Bot
- **8.2**: Dashboard
- **8.3**: Neue Plattform

12.0.1 Meilensteine (Version 1.0 - Ursprüngliche Planung)

Ursprünglich waren die Meilensteine folgendermassen geplant:

M1: Projekt Initiierung
- Der Projektplan ist definiert.
- Die Risiken sind definiert.
- Das MVP ist definiert.
- Der Long-Term-Plan ist definiert.

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

M3: Erreichung des MVP
- Die Discord-Bot Erweiterungen sind umgesetzt.
- Das Admin-Dashboard ist elementarisch umgesetzt.
- Das MVP wird erreicht.

M4: Fertigstellung der Implementation

- Alle geplanten, nicht-optionalen Features sind umgesetzt.
- Alle Funktionen sind erfolgreich getestet.

M5: Abgabe

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

M6: Präsentation

- Die Simulation wurde durchgeführt und ausgewertet.
- Die Präsentation ist fertiggestellt.
- Das Poster ist fertiggestellt.

13 Interviews zur Vereins-Motivation

Die Interviews wurden als halbstrukturierte, qualitative Gespräche mit vier anonymisierten Präsidenten Schweizer Gaming-Vereine geführt (durchschnittlich ungefähr 40 Minuten, leitfadengestützt). Die Auswahl erfolgte gezielt, um organisatorische und strategische Perspektiven abzubilden.

Der Leitfaden war entlang der drei Grundbedürfnisse der Self-Determination Theory (SDT: Autonomie, Kompetenz, soziale Eingebundenheit) strukturiert und wurde um praxisbezogene Blöcke (Kommunikation, digitale Tools, Fairness / Regeln, Kennzahlen) ergänzt. Die Auswertung erfolgte qualitativ-interpretativ in Form einer abgeleiteten Kodierung nach SDT, ergänzt um darauf aufbauende Subcodes (z. B. „Kommunikationsengpass“, „Anerkennung sichtbar machen“, „Saisonalität / Resets“, „Datensparsamkeit“).

Zur Sicherstellung eines klaren Aufbaus wurde der Leitfaden in sieben thematische Blöcke gegliedert:

1. **Allgemeine Fragen**: Kontext zu Rolle, Vereinsgrösse und Gründungsjahr zur Vergleichbarkeit der Antworten.
2. **Vereinsmotivation**: Aktuelle Beitrittsgründe, Bindungsfaktoren und Herausforderungen.
3. **Aktivitäten und Engagement**: Motivierende Angebote, Unterschiede zwischen neuen und erfahrenen Mitgliedern, Umgang mit Passivität.
4. **Digitale Tools und Gamification**: Nutzung digitaler Plattformen, Erfahrungen mit spielerischen Elementen, Chancen und Risiken.
5. **Zukunft und Anpassungen**: Wünsche nach Belohnungen und nachhaltigen Fördermassnahmen.
6. **Digitale Brücke**: Hypothetische Fragen zu einem digitalen Tool, Kennzahlen und gewünschten Features.
7. **Abschluss**: Offene Ergänzungen und Reflexionen.

Um unbeeinflusste Antworten zu gewährleisten, wurden digitale Lösungsansätze erst in den abschliessenden Blöcken thematisiert. Die vollständigen Leitfadenfragen sowie die anonymisierten Rohantworten und die Kodiermatrix sind als separate Dateien im Anhang abgelegt (vgl. „Gamification Interviews - Plan.pdf“ und „Gamification Interviews - Antworten.xlsx“).

Eine zusammenfassende Analyse der Ergebnisse erfolgt in Abschnitt 3.2 des erweiterten Gamification-Konzeptes.

14 Technische Ergänzungen zum Sicherheitskonzept

Dieses Kapitel enthält technische Detailinformationen zum erweiterten Sicherheitskonzept (vgl. Abschnitt 4). Die hier dargestellten Inhalte dienen der Nachvollziehbarkeit der sicherheitsrelevanten Implementierungsaspekte und unterstützen die im Hauptteil beschriebenen konzeptionellen Massnahmen.

14.1 Vergleich der DB-MFA-Varianten

In der folgenden Tabelle sind die untersuchten MFA-Varianten für die in diesem Projekt verwendete MongoDB dargestellt:

Variante	Beschreibung	Stärken (+) und Schwächen (-)
Netzwerk-MFA (VPN / SSH-Bastion)	Zugriff auf die Datenbank ist nur über ein privates, nicht öffentlich erreichbares Netzwerk möglich. Der Zugang zu diesem Netzwerk wird über eine Bastion oder einen VPN-Gateway geregelt, der eine Multi-Faktor-Authentifizierung (z. B. TOTP oder Hardware-Token) verlangt. Damit wird der gesamte Zugriffspfad geschützt, ohne dass Backend oder Bot angepasst werden müssen. (Bezug: NFR4 , NFR5)	+ geringer Implementierungsaufwand, hohe Kompatibilität und transparente Integration in bestehende Infrastruktur. - VPN oder Bastion werden zu zentralen Kontrollpunkten, die gezieltes Monitoring und Backup-Strategien erfordern.
X.509-Clientzertifikate (mutual TLS)	Die Authentifizierung erfolgt über ein digitales Zertifikat als Besitzfaktor (MongoDB-Contributors, 2025c) (Bezug: NFR4). Diese Methode bietet einen hohen Schutz vor Phishing und erlaubt eine präzise Nachverfolgbarkeit einzelner Zugriffe.	+ stark gegen Phishing und gute Auditierbarkeit. - aufwändige Zertifikatsverwaltung, zusätzlicher Administrationsaufwand.
SSO (Single-Sign-On) / Identity-Provider-basierte MFA	Authentifizierung erfolgt über einen externen Identity Provider (z. B. Microsoft Entra ID[18]) mittels OpenID Connect (nachfolgend OIDC genannt). MongoDB unterstützt seit Version 7.0 die direkte Integration solcher Anbieter, wodurch sich SSO und MFA zentral verwalten lassen (MongoDB-Contributors, 2025d) (Bezug: NFR4).	+ zentrale Personen- und Richtlinienverwaltung über Systeme hinweg. - erfordert zusätzliche Infrastruktur, Lizenzkosten und laufende Wartung.

Tab. 67: Varianten zur Absicherung des DB-Zugangs

Die Alternative über X.509-Zertifikate als besonders sicher eingestuft, ist aber aufgrund des erhöhten Verwaltungsaufwands und der nötigen Zertifikatsinfrastruktur für den Anwendungsfall eines Vereins-Tools nur bedingt sinnvoll.

Eine SSO- / IdP-basierte Lösung bietet theoretisch die beste Richtliniensteuerung und zentrale Benutzerverwaltung, würde aber zusätzliche Infrastruktur, Lizenzen und Wartungskosten verursachen. Da ClansCore bewusst als Open-Source-System für Vereine konzipiert ist, stehen Kosteneffizienz und einfache Wartbarkeit im Vordergrund. Die Einführung eines kommerziellen Identity-Providers würde diesen Grundsatz unterlaufen und die Einstiegshürde für kleinere Vereine erheblich erhöhen.

Aus diesen Gründen wird für ClansCore die Lösung Netzwerk-MFA über VPN beziehungsweise SSH-Bastion gewählt. Diese Variante bietet eine robuste Absicherung gegen unautorisierte Zugriffe, ist wirtschaftlich umsetzbar und unterstützt das Prinzip der klaren Trennung zwischen menschlichen und maschinellen Zugängen. Menschliche Admins authentifizieren sich über MFA auf der Netzwerkebene und systeminterne Dienste (z. B. Bot, Backend) kommunizieren ausschliesslich innerhalb des privaten Docker-Netzwerks über dedizierte Dienstkonten mit minimalen Rechten.

[18] https://www.microsoft.com/de-ch/security/business/identity-access/microsoft-entra-id

Die Wirksamkeit dieser Architektur wird durch externe Quellen untermauert. Laut Microsoft verhindern aktivierte MFA-Mechanismen ca. 99.2% bis 99.9% der üblichen Kontoübernahmen und werden deshalb in Zukunft potenziell zwingend eingeführt für VPN / Bastion (Microsoft-Contributors, 2025a; 2025b) . Ausserdem empfiehlt die Cybersecurity and Infrastructure Security Agency (CISA) den MFA-Einsatz, insbesondere für VPN- und Bastion-Zugänge (CISA, 2022) . Damit erfüllt die gewählte Lösung die anerkannten Best Practices moderner IT-Sicherheitsarchitekturen und bleibt zugleich ressourcenschonend, was den spezifischen Anforderungen einer Vereinssoftware gerecht wird.

Zusätzlich nutzt die Datenbank das standardisierte Authentifizierungsverfahren SCRAM-SHA-256 (MongoDB-Contributors, 2025e) in Kombination mit rollenbasierter Zugriffskontrolle (MongoDB-Contributors, 2025f) . Für zukünftige Ausbaustufen kann eine ergänzende Zertifikatsauthentifizierung für besonders privilegierte Konten in Betracht gezogen werden.

14.2 Dashboard-Sicherheit und Passwortmanagement

Nach erfolgreicher Anmeldung werden kurzlebige Zugriffstokens und rotierende Auffrischungstokens ausgestellt. Tokens werden als HttpOnly- und Secure-Cookies gespeichert und zustandsverändernde Anfragen erfordern ein CSRF-Token. Token-Lebenszyklen und Session-Invalidierung entsprechen den Empfehlungen des OWASP zur sicheren Sitzungsverwaltung (OWASP-Cheat-Sheets-Series-Contributors, 2025) .

Sämtliche Kommunikation zwischen Frontend und Backend erfolgt über TLS-verschlüsselte HTTPS-Verbindungen. Rollen- und Berechtigungsprüfungen erfolgen serverseitig über RBAC.

Vergessene Passwörter können über einen zeitlich begrenzten Reset-Link (Gültigkeit ≤ 30 Minuten) zurückgesetzt werden. Reset-Links werden ausschliesslich an die hinterlegte E-Mail-Adresse gesendet, ohne offenzulegen, ob ein Konto existiert. Die Passwort-Policy fordert mindestens 12 Zeichen, Kombination aus Gross- / Kleinbuchstaben, Ziffern und Sonderzeichen. Passwörter werden mit bcrypt (≥ 12 Rounds) oder einem gleichwertigen KDF sicher gehasht gespeichert. Fehlgeschlagene Anmeldeversuche führen zu progressiven Delays (Rate-Limiting). Erfolgreiche und fehlerhafte Änderungen werden im Audit-Log protokolliert.

Diese technischen Massnahmen dienen der Umsetzung der Anforderungen NFR4 und NFR5 und stellen sicher, dass Authentifizierungsvorgänge den Prinzipien von Privacy by Design und Defense in Depth entsprechen.

14.3 Nebenläufigkeitskontrolle und Transaktionen

Jeder Datensatz in der Datenbank enthält eine Versionskennung sowie einen Zeitstempel. Bei einem Update-Vorgang überprüft das Backend, ob die vom Client übermittelte Version noch mit der aktuellen Version in der Datenbank übereinstimmt. Wurde der Datensatz in der Zwischenzeit verändert, wird der Schreibvorgang abgewiesen und als Konflikt im Audit-Log vermerkt. Das Dashboard fordert daraufhin den aktuellen Datensatz an und kann die Änderungen manuell oder automatisch zusammenführen.

Für besonders sensible Vorgänge, wie Punktebuchungen, Rollenänderungen oder Event-Updates, werden zusätzlich atomare Transaktionen eingesetzt. Dadurch ist gewährleistet, dass zusammenhängende Operationen entweder vollständig oder gar nicht ausgeführt werden, was Inkonsistenzen in der Datenbank verhindert.

Alle Konflikte sowie deren Auflösung werden im Audit-Log dokumentiert.

14.4 Container- und Deployment-Details

Die Architektur besteht aus separaten Containern für Discord-Bot, Backend-Service, Admin-Dashboard, Reverse-Proxy und MongoDB. Jeder Container besitzt ein eigenes Dateisystem, eigene Prozessräume und dedizierte Netzwerk-Namespaces. Die Kommunikation erfolgt ausschliesslich über interne Docker-Netzwerke. Externe Zugriffe laufen ausschliesslich über den Reverse-Proxy mit TLS 1.3-Verschlüsselung. Die Datenbank ist nicht öffentlich erreichbar (vgl. Abschnitt 4.3.2).

Container-Isolation basiert auf Linux-Namespaces, Cgroups und Capability-Drops, die laut NIST als Kernmechanismen moderner Container-Sicherheit gelten sowie die Angriffsfläche erheblich reduzieren. (Murugiah & Karen, 2017)

Zugriff auf den Host erfolgt ausschliesslich über SSH-Key-Authentifizierung mit MFA über VPN / Bastion-Ebene (CISA, 2022) . Das Deployment erfolgt automatisiert über GitHub Actions, wobei sämtliche Secrets (z. B. Tokens, Zugangsdaten) in verschlüsselten GitHub-Vaults verwaltet werden. Backups sind versioniert, verschlüsselt und nur für Admins zugänglich. (Bezug: NFR4 , NFR5)

15 Umsetzung des Deployments

Dieses Kapitel beschreibt das umgesetzte Deployment-Konzept des Systems ClansCore. Es ergänzt die Architektur- und Implementationskapitel um betriebliche Aspekte. Ziel ist die Beschreibung einer reproduzierbaren, plattformunabhängigen Bereitstellung des Systems für Entwicklungs- und kleinere Produktivumgebungen.

15.1 Deployment-Architektur

Das System wird als Multi-Container-Anwendung betrieben. Jede Hauptkomponente läuft in einem eigenen Docker-Container und ist über ein internes Bridge-Netzwerk verbunden. Dadurch werden Abhängigkeiten klar getrennt und die interne Kommunikation standardisiert.

Die Laufzeitumgebung umfasst:
- REST-API zur Geschäftslogik
- Discord-Bot zur Nutzerinteraktion
- Webbasiertes Admin-Dashboard
- MongoDB-Datenbank zur persistenten Datenspeicherung

Externe Zugriffe sind auf die erforderlichen Schnittstellen beschränkt. Die interne Service-Kommunikation erfolgt ausschliesslich über Service-Namen innerhalb des Docker-Netzwerks. Ein Überblick über die logische Container-Struktur ist in Abschnitt 5.2.2 dargestellt.

15.2 Containerisierung, Konfiguration und Betrieb

Die Containerisierung erfolgt mit Docker, die Orchestrierung mit Docker Compose. Alle Services werden deklarativ beschrieben und gemeinsam gestartet. Zur Trennung von Build- und Laufzeitumgebungen sowie zur Reduktion der Image-Grösse kommen durchgängig Multi-Stage Builds zum Einsatz.

Die Systemkonfiguration erfolgt vollständig über Umgebungsvariablen. Sensible Informationen wie Tokens, Zugangsdaten oder API-Schlüssel werden nicht im Code hinterlegt, sondern zur Laufzeit injiziert. Dadurch wird eine klare Trennung von Code und Konfiguration sowie die Unterstützung unterschiedlicher Umgebungen ermöglicht.

Der Build- und Startprozess ist automatisiert und reproduzierbar. Persistente Daten werden über Docker Volumes gespeichert. Für zentrale Services sind Restart-Policies und Health Checks konfiguriert, sodass das System nach Fehlern oder Neustarts automatisch wieder hochfährt. Backups der Datenbank sind vorgesehen und können regelmässig durchgeführt werden.

15.3 Deployment-spezifische Einschränkungen

Das Deployment ist für einen Single-Host-Betrieb ausgelegt und nutzt Docker Compose als Orchestrierungswerkzeug. Für den vorgesehenen Projektumfang ist diese Lösung ausreichend.

Nicht umgesetzt wurden weiterführende betriebliche Funktionen wie automatische Skalierung, zentrale Log-Aggregation oder eine vollständige CI/CD-Integration. Diese Punkte stellen bewusste Abgrenzungen dar und werden in der Schlussfolgerung der Arbeit reflektiert (vgl. Abschnitt 8).

16 Test-Ergebnisse

ID	**T1**
Datum	16.12.2025
Input	Es wird ein Commit erstellt.
Erwartetes Resultat	Alle Unit-Tests schliessen erfolgreich ab.

Tatsächliches Resultat	Die Tests werden nach jedem Commit automatisiert ausgeführt und liefen zum Zeitpunkt der Dokumentation erfolgreich durch.
	
	Discord-Bot:
	
	`Test Suites: 17 passed, 17 total`
	`Tests: 56 passed, 56 total`
	`Snapshots: 0 total`
	`Time: 22.456 s`
	`Ran all test suites.`
	
	Abb. 27: Resultat der Discord-Bot Unit Tests
	
	Folgende Elemente des Discord-Bots wurden getestet:
	• claimtask
	• completetask
	• createTask
	• donation
	• events
	• getData
	• help
	• intro
	• join
	• leaderboard
	• leave
	• linkcalendar
	• ping
	• score
	• synccalendar
	• syncroles
	• syncusers
	
	Dashboard:
	
	`Chrome Headless 143.0.0.0 (Linux 0.0.0): Executed 30 of 30 SUCCESS (1.131 secs / 1.018 secs)`
	`TOTAL: 30 SUCCESS`
	`✓ Browser application bundle generation complete.`
	`✓ Browser application bundle generation complete.`
	
	Abb. 28: Resultat der Dashboard Unit Tests
	
	In den Unit-Tests für das Dashboard wird getestet ob:
	• die Komponente sich instanziieren lässt
	• keine Import-/Dependency-Injection-Fehler vorhanden sind
	• das Template ohne Fehler kompiliert

Tab. 68: Test Report: Unit-Tests

ID	T2
Datum	16.12.2025
Tatsächliches Resultat	Auf dedizierte, separat implementierte Integrationstests wurde verzichtet. Die korrekte Integration der Systemkomponenten (Backend, Datenbank, externe APIs) wurde im Rahmen der Unit-Tests sowie durch manuelle Systemtests überprüft.

Tab. 69: Test Report: Integration-Tests

ID	T3
Datum	16.12.2025
Tatsächliches Resultat	Im Rahmen dieser Arbeit wurden keine klassischen Usability-Tests durchgeführt. Stattdessen ist nach Abschluss der Arbeit eine zweiwöchige Simulation eines Vereinsbetriebs zur qualitativen Evaluation der Usability vorgesehen (vgl. Anhang „ClansCore-Simulation - Plan.pdf“). Die Resultate der Simulation werden ebenfalls im separaten Anhang beigelegt.

Tab. 70: Test Report: Usability-Tests

ID	T4
Datum	16.12.2025
Input	Alle funktionalen und nicht-funktionalen Requirements manuell überprüfen.
Erwartetes Resultat	Alle Requirements sind überprüft und dokumentiert.
Tatsächliches Resultat	Alle relevanten funktionalen sowie ausgewählte nicht-funktionale Anforderungen wurden manuell überprüft und im Abschnitt Use Cases sowie Non-Functional Requirements dokumentiert.

Tab. 71: Test Report: System-Tests

17 Screenshots Discord-Bot

In diesem Kapitel werden ausgewählte Funktionalitäten des entwickelten Discord-Bots anhand von Screenshots dargestellt. Der Fokus liegt dabei auf den zentralen Benutzerinteraktionen sowie auf den wesentlichen Erweiterungen gegenüber dem Vorprojekt.

Die Abbildungen zeigen sowohl grundlegende Funktionen wie den Beitritt zum Verein, die Anzeige von Events und Ranglisten als auch administrative Prozesse, beispielsweise die Synchronisation von Benutzern, Rollen und Kalender-Events. Besonderes Augenmerk wird auf jene Funktionen gelegt, die im Rahmen dieser Arbeit neu konzipiert oder architektonisch überarbeitet wurden.

Die Darstellung der Screenshots dient primär der Veranschaulichung der Benutzerführung und der Integration der entwickelten Logik in Discord. Eine vollständige und detaillierte Beschreibung aller verfügbaren Befehle, Konfigurationsoptionen sowie technischer Details des Discord-Bots ist nicht Bestandteil dieses Kapitels.

Für eine vollständige Einsicht in den gesamten Funktionsumfang sowie die Einrichtungs- und Administrationsmöglichkeiten des ursprünglichen Systems wird auf den Anhang des Vorprojekts verwiesen (vgl. Vorprojekt, Kap. 17).

17.1 Grundlegende Benutzerinteraktionen

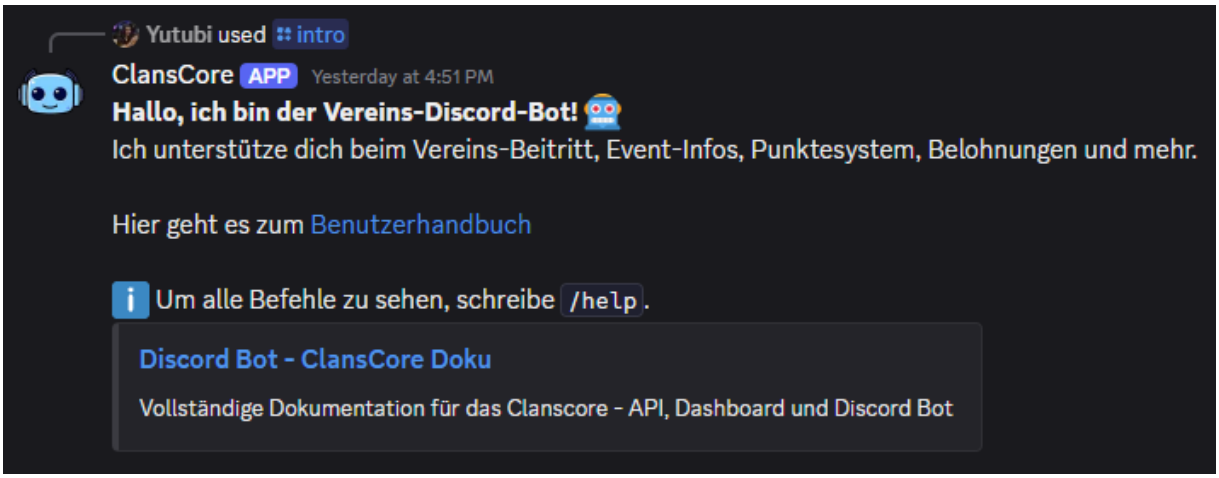


Abb. 29: Discord-Bot: Intro-Nachricht des Bots im Discord Server

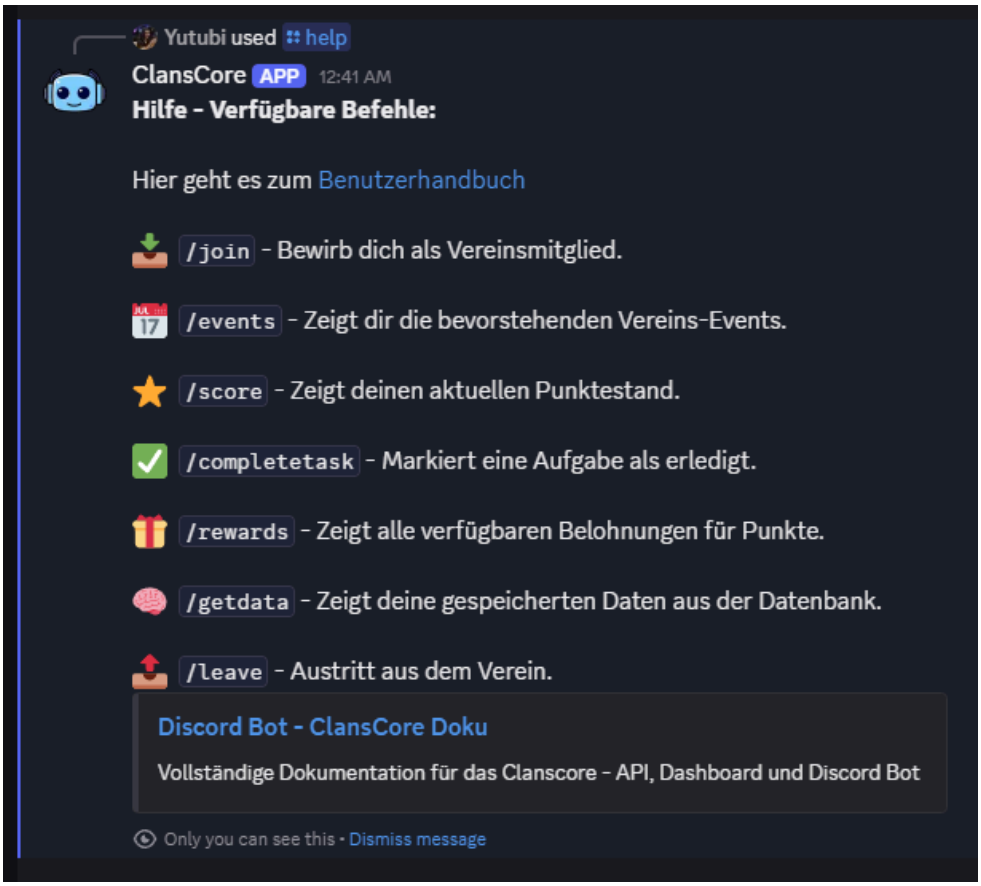


Abb. 30: Discord-Bot: Befehl `/help` aus der Sicht eines Nicht- bzw. Mitgliedes

Abb. 31: Discord-Bot: Erweiterte Ansicht des Befehls `/help` bei Vorstandmitglieder

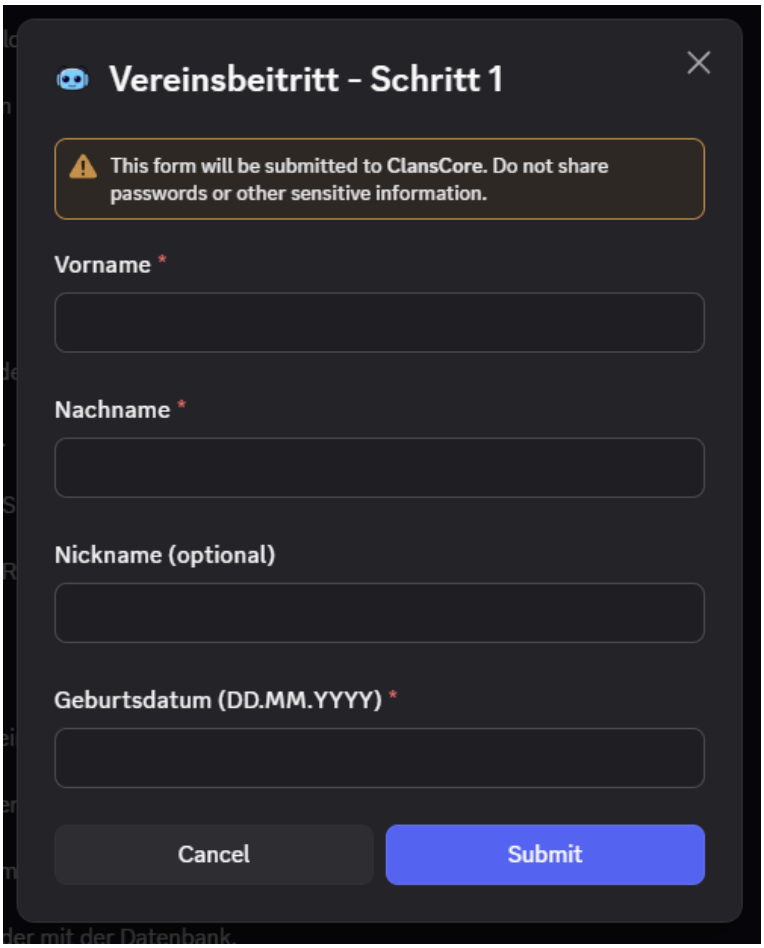


Abb. 32: Discord-Bot: Vereins-Beitritt eines neuen Mitglieds über den Befehl `/join`, Schritt 1

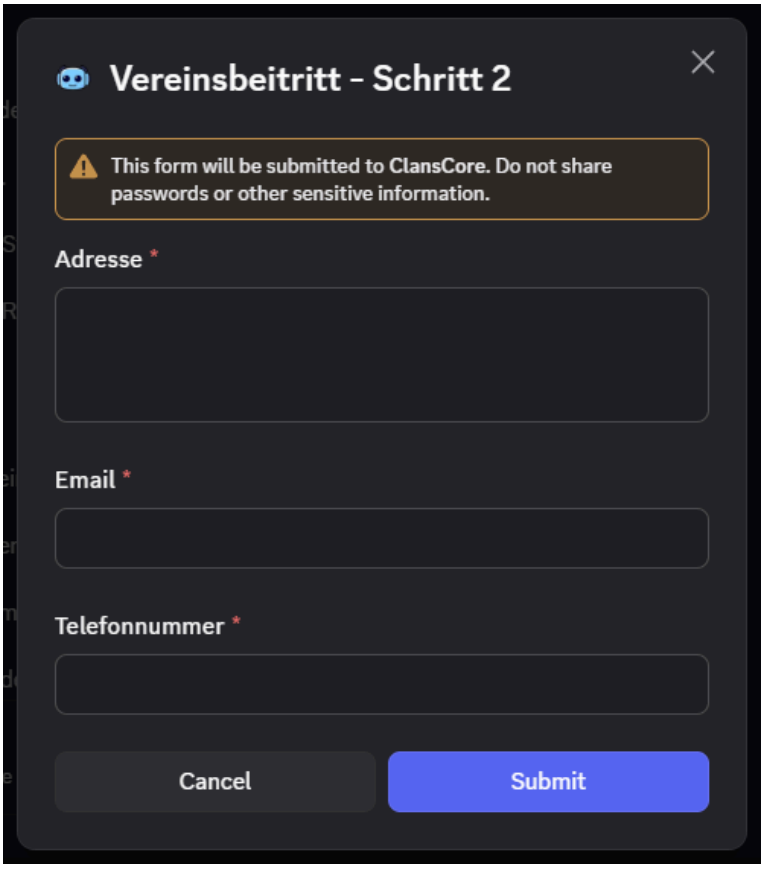


Abb. 33: Discord-Bot: Vereins-Beitritt eines neuen Mitglieds über den Befehl `/join`, Schritt 2

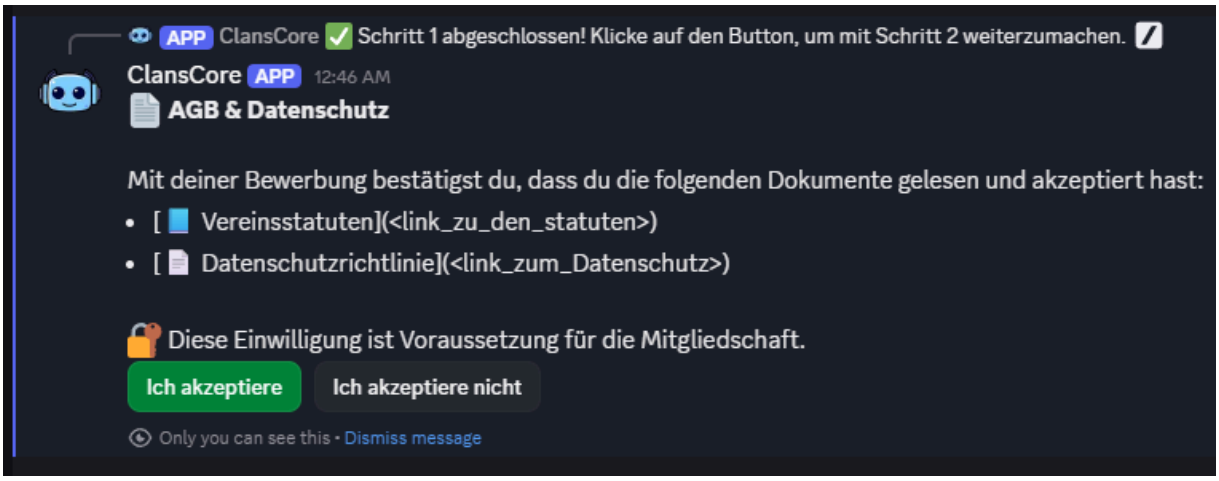


Abb. 34: Discord-Bot: Bestätigungs-Aufforderung der Nutzungsbedingungen und Vereins-Statuten

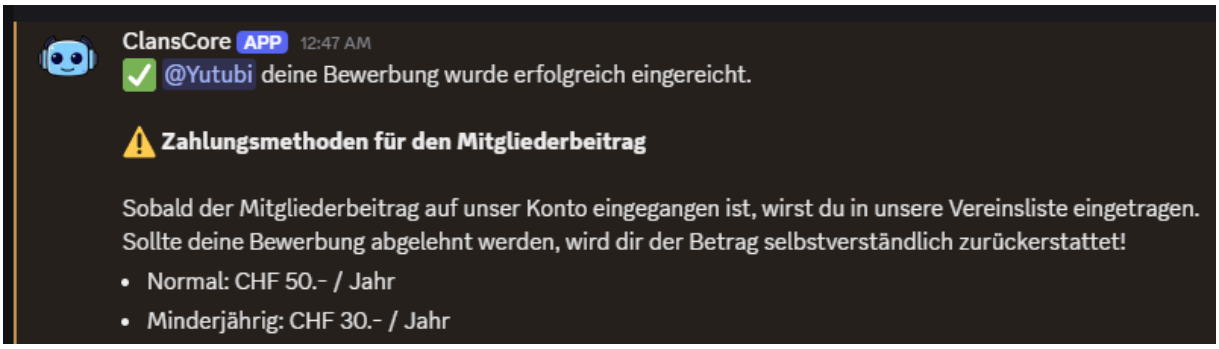


Abb. 35: Discord-Bot: Direktnachricht und Zahlungsaufforderung des MitgliederBeitrages

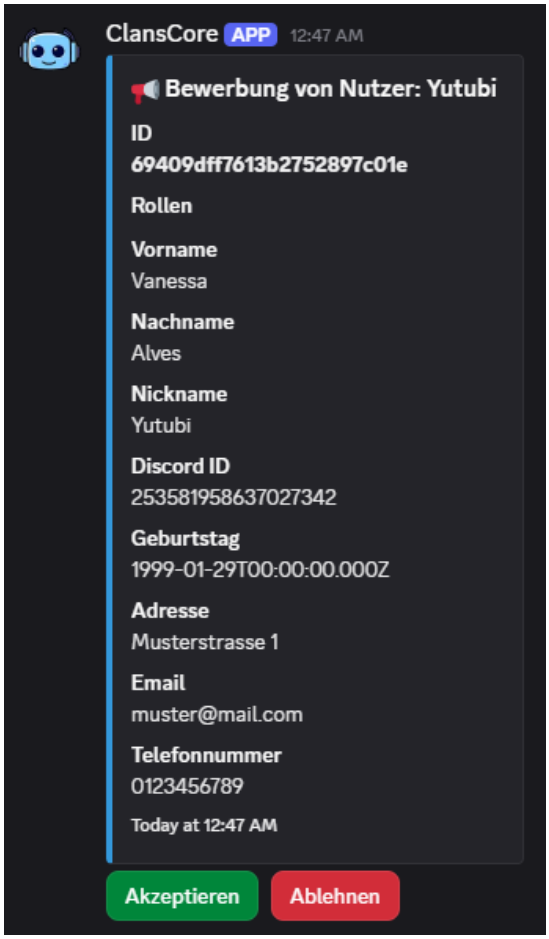


Abb. 36: Discord-Bot: Entscheidungs-Nachricht im Vorstands-Channel `bot-bewerbungen`

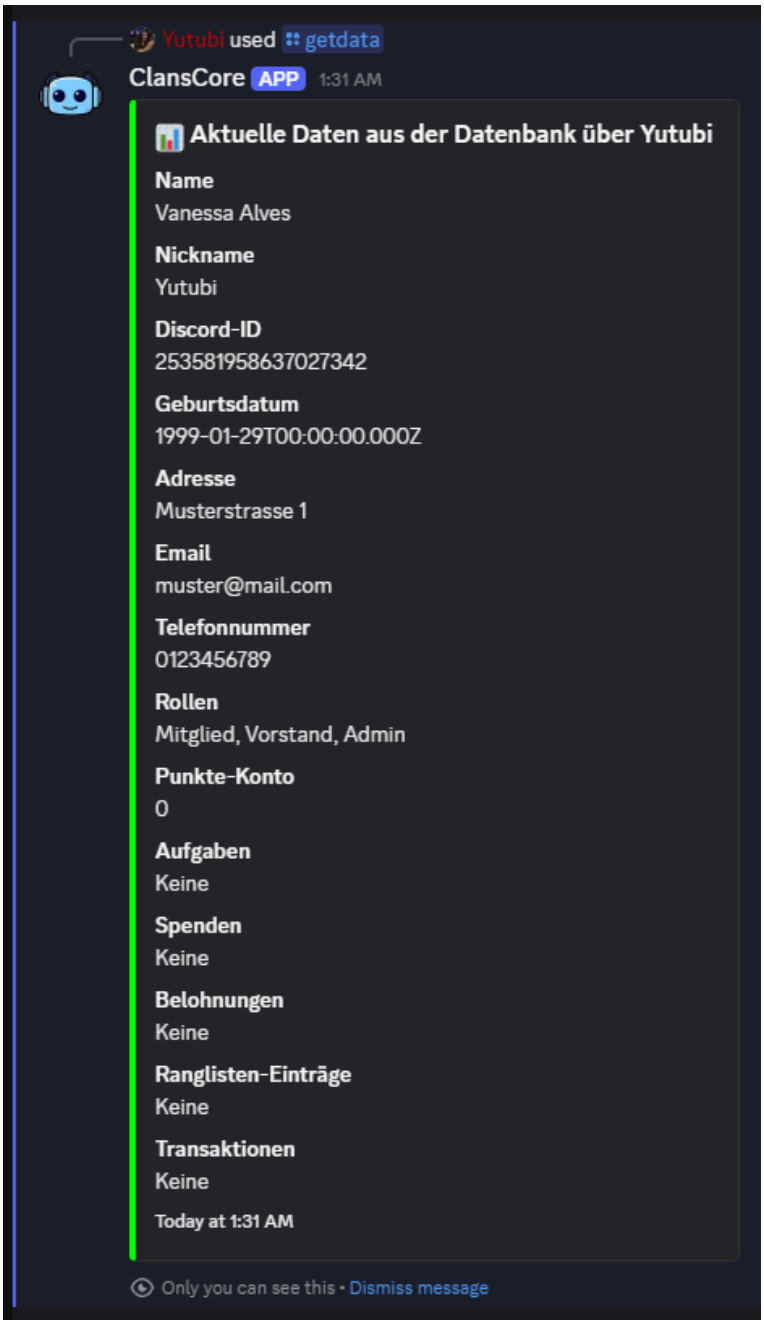


Abb. 37: Discord-Bot: Befehl `/getdata` für die eigenständige Daten-Einsicht

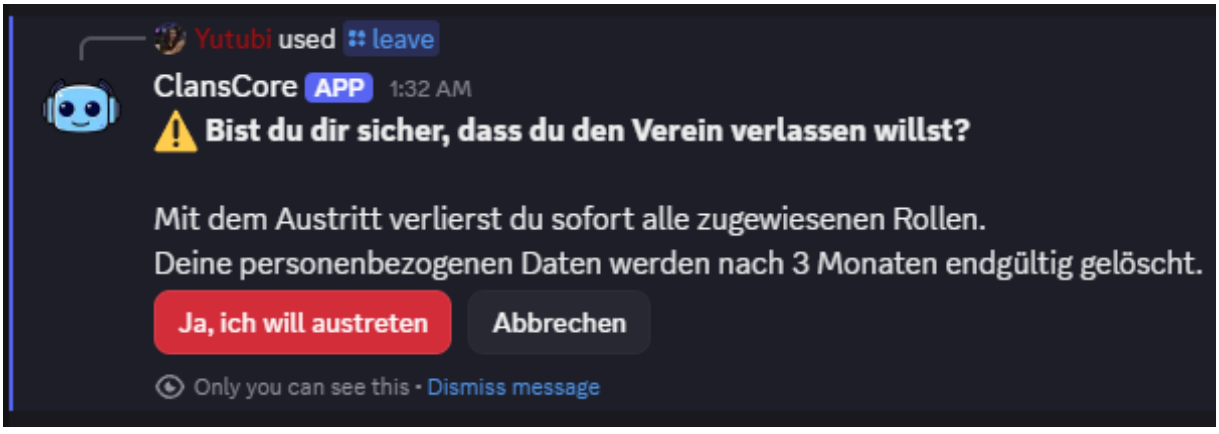


Abb. 38: Discord-Bot: Befehl `/leave` für den eigenständigen Vereins-Austritt

Abb. 39: Discord-Bot: Benachrichtigung zum Vereins-Austritt inkl. Löschdatum der Daten

17.2 Administrative Funktionen im Discord-Bot

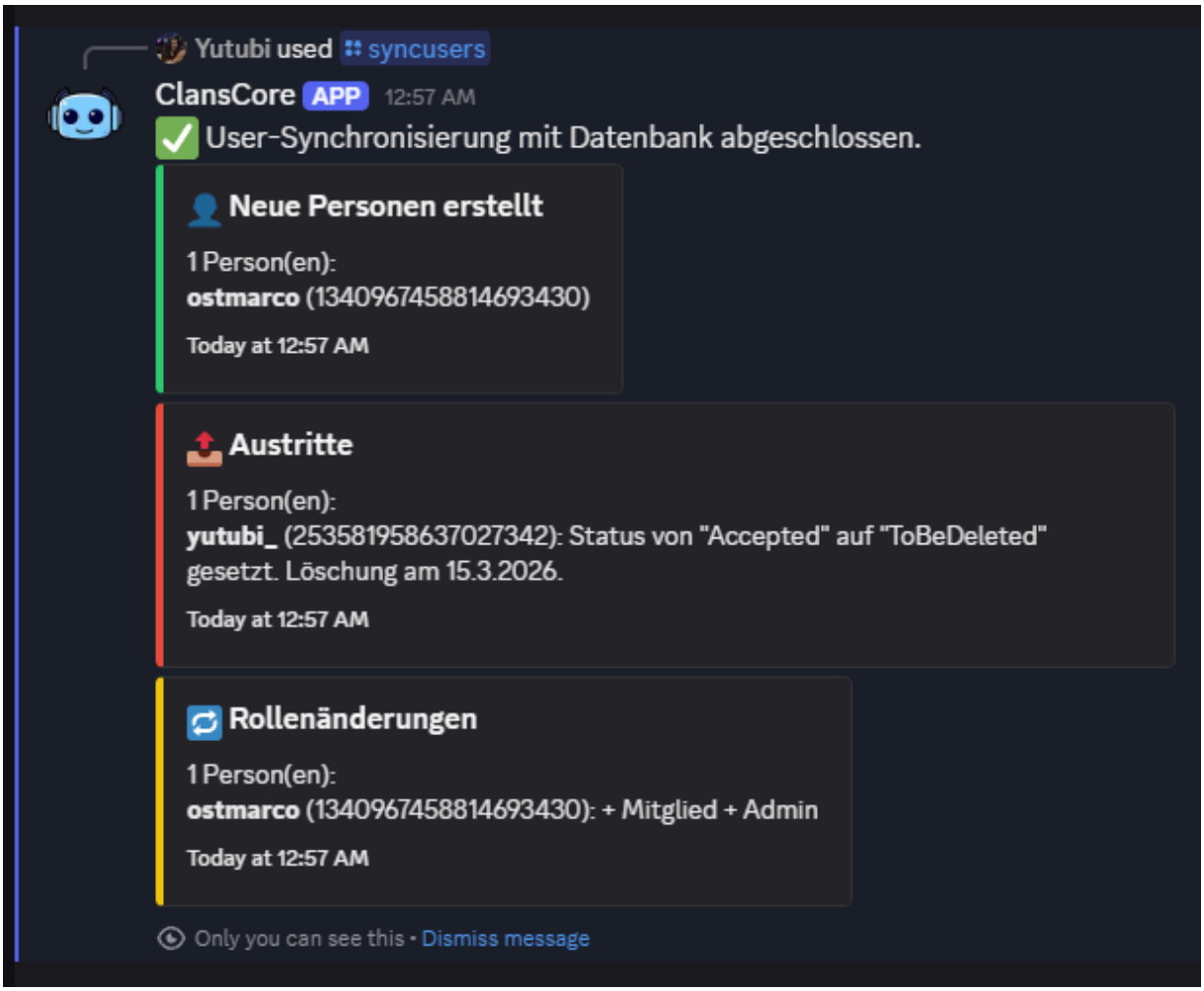


Abb. 40: Discord-Bot: Synchronisation von Discord-Benutzern und Vereinsmitgliedern mit der zentralen Datenbank über /syncusers

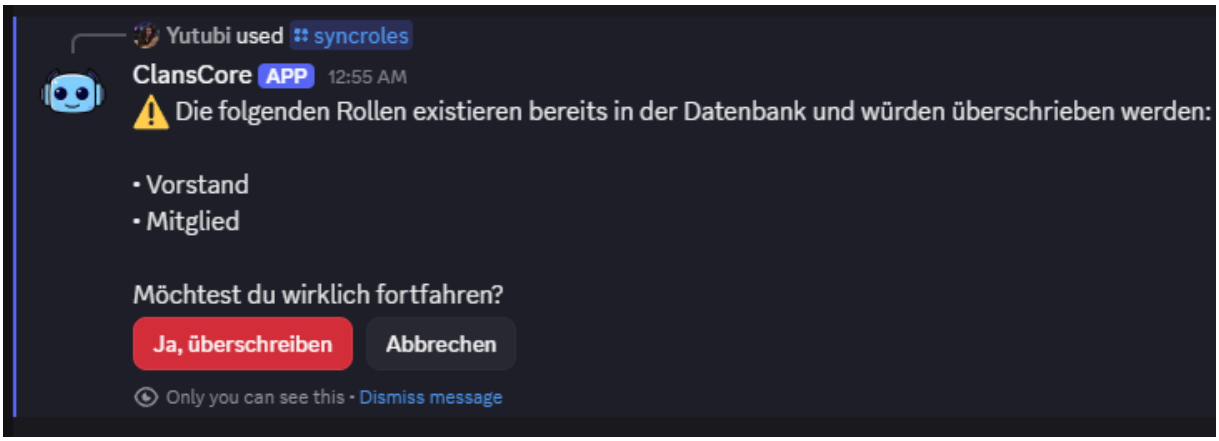


Abb. 41: Discord-Bot: Abgleich und Aktualisierung zentraler Vereinsrollen zwischen Discord und Datenbank mittels /syncroles

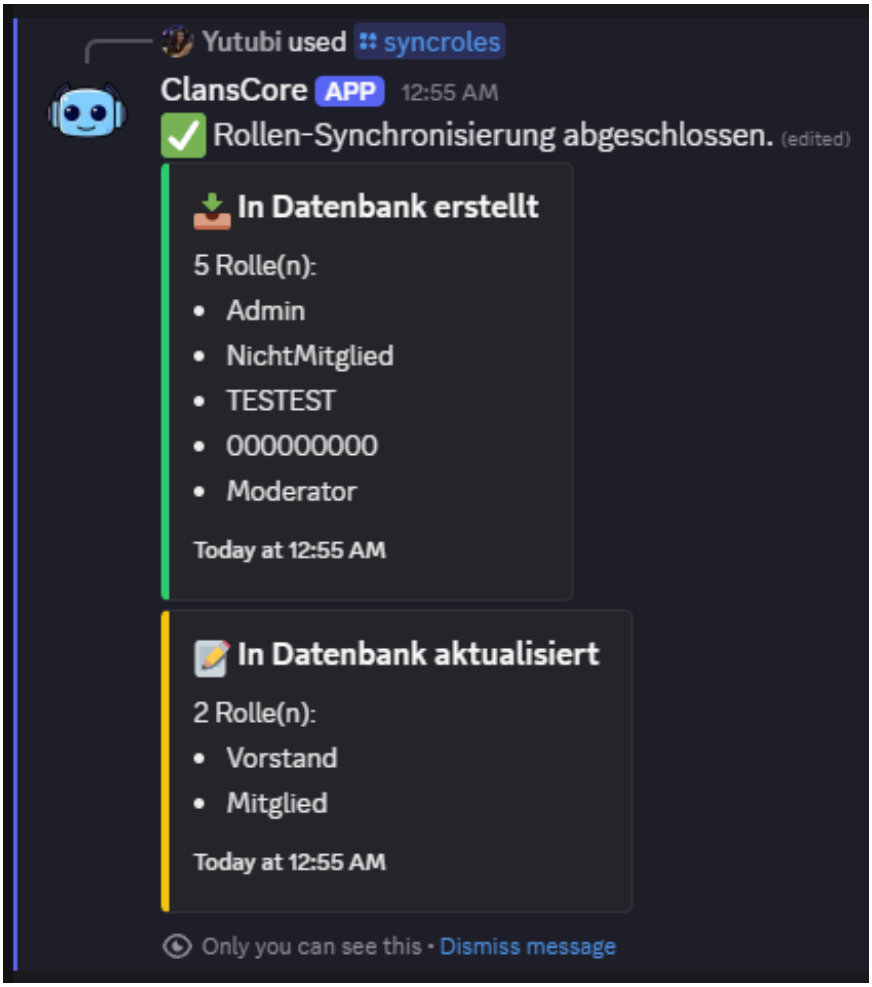


Abb. 42: Discord-Bot: Bestätigung der Rollen-Änderungen in Discord und der Datenbank

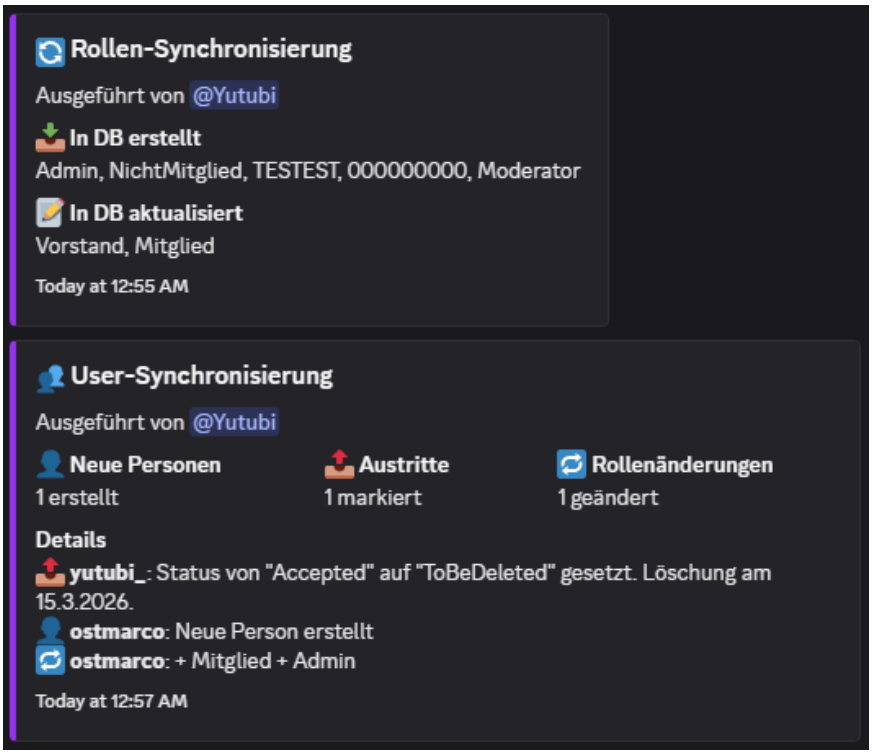


Abb. 43: Discord-Bot: Log-Nachricht der User- und Rollen-Synchronisation im Vorstands-Channel `bot-log`

17.3 Gamification- und Engagement-Funktionen

Abb. 44: Discord-Bot: Abfrage des aktuellen Punktestandes eines Mitglieds über den Befehl `/score`

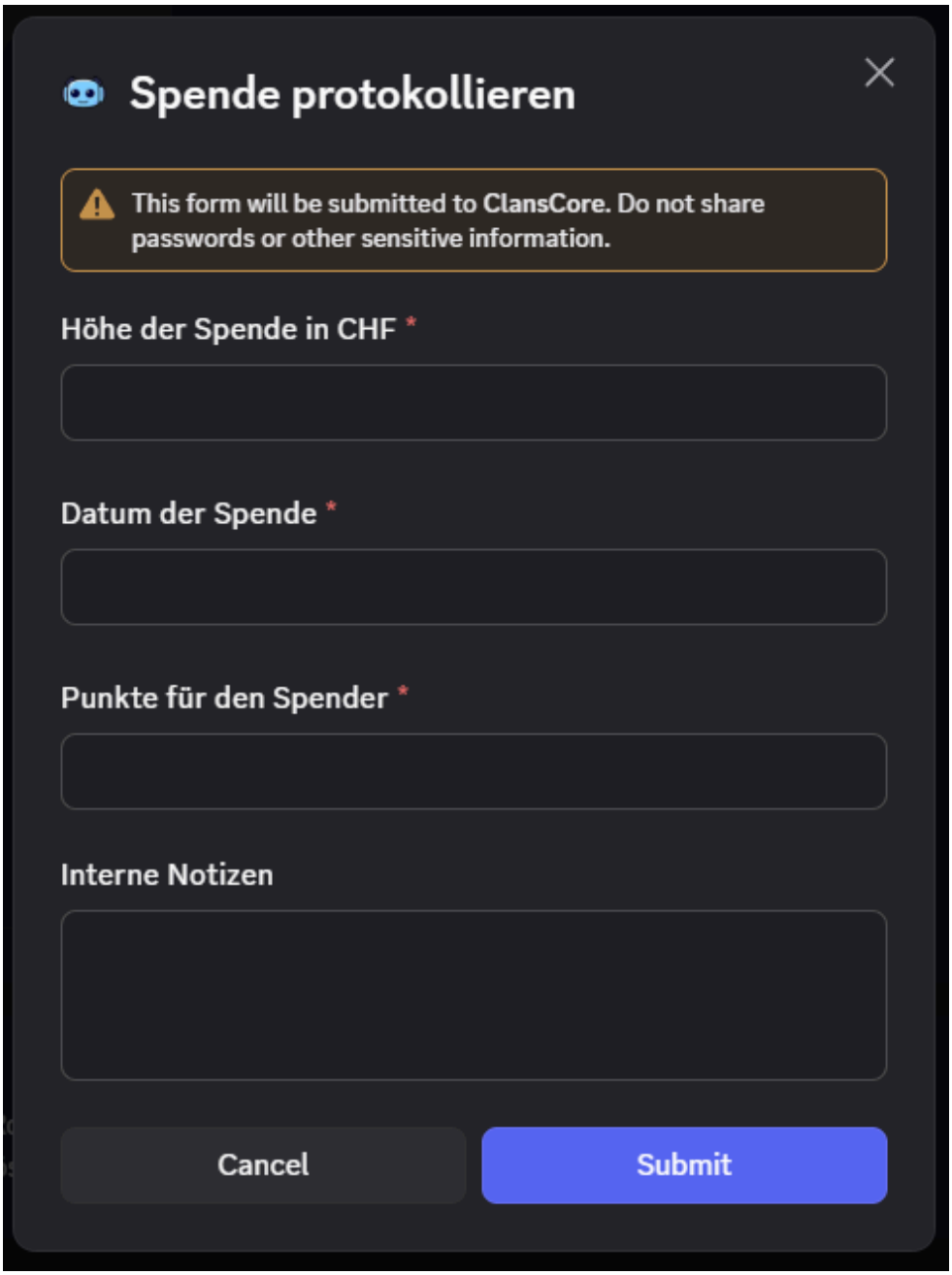


Abb. 45: Discord-Bot: Vorstands-Befehl `/donation` zur protokollierung von Spenden und vergabe von Punkten

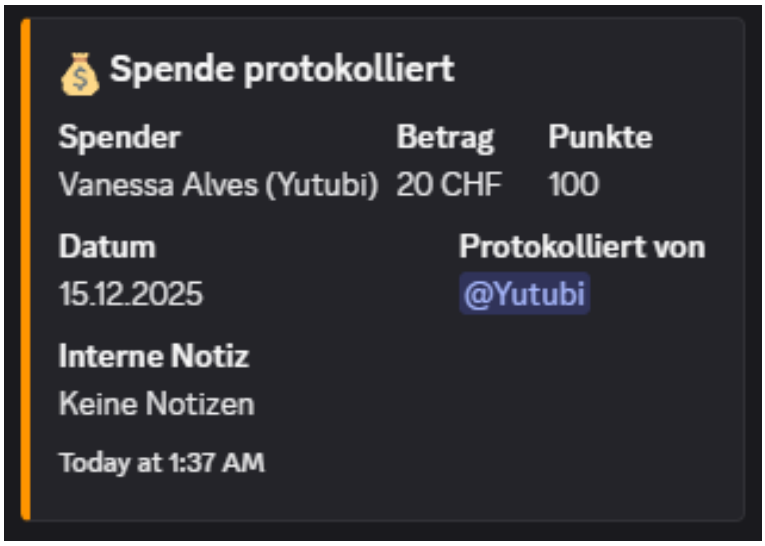


Abb. 46: Discord-Bot: Log-Nachricht der protokollierte Spende im Vorstands-Channel `bot-log`

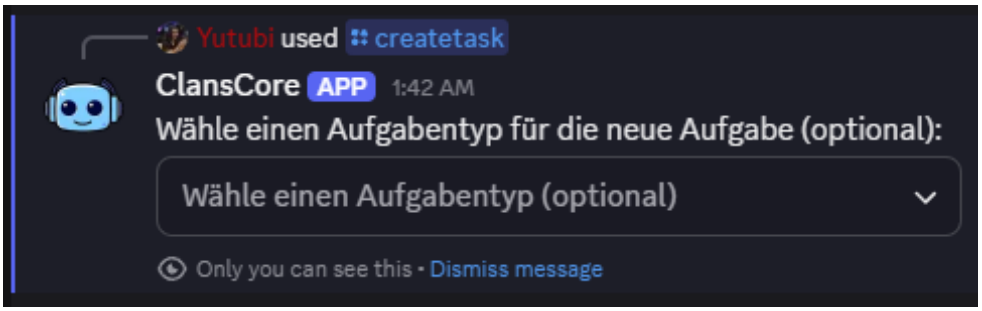


Abb. 47: Discord-Bot: Vorstands-Befehl `/createtask` zur Erstellung neuer Aufgaben inkl. Auswahl vordefinierter Aufgabentypen

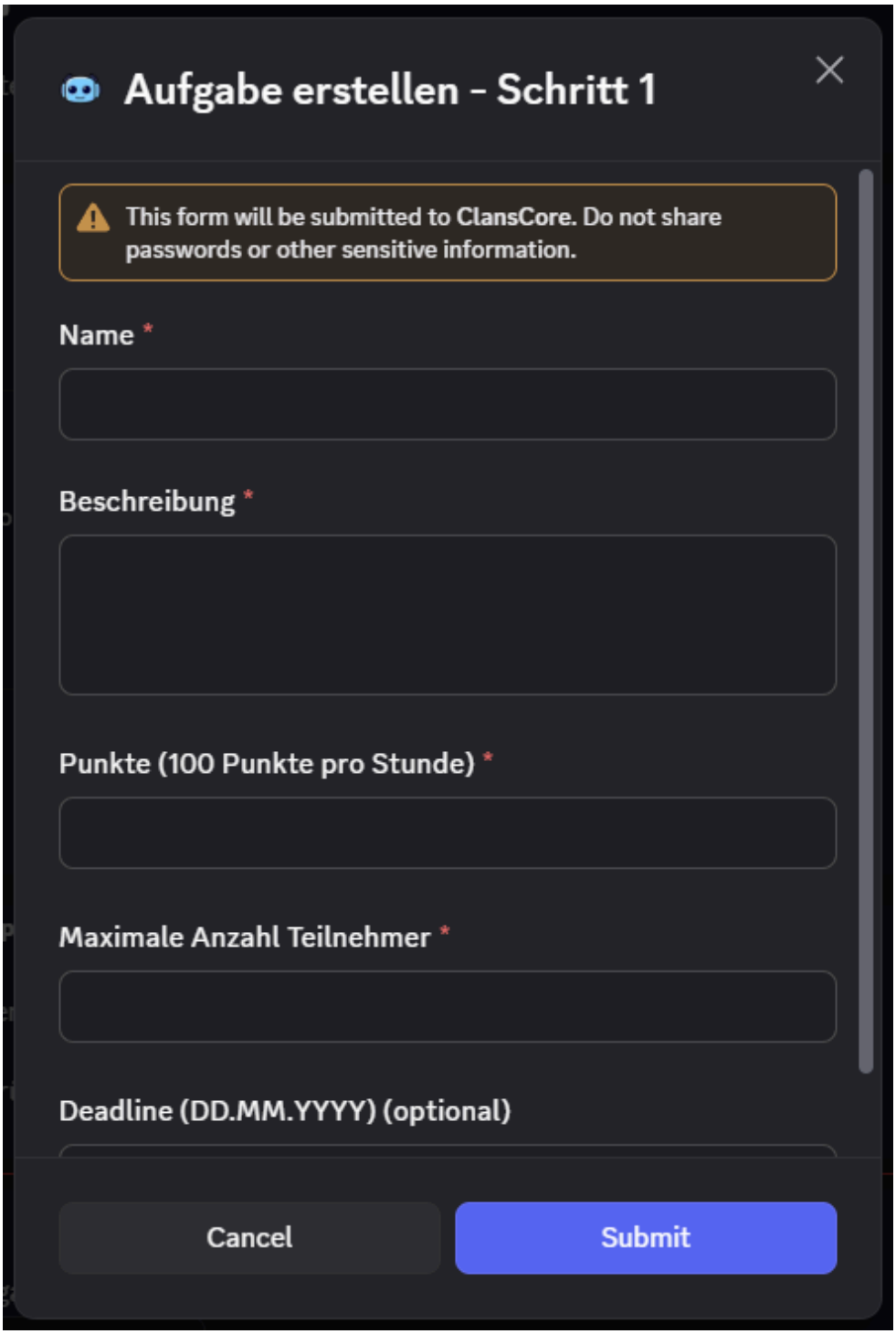


Abb. 48: Discord-Bot: Erstellung neuer Aufgaben inkl. automatischem Punkte-Vorschlag gemäss Aufgabentyp

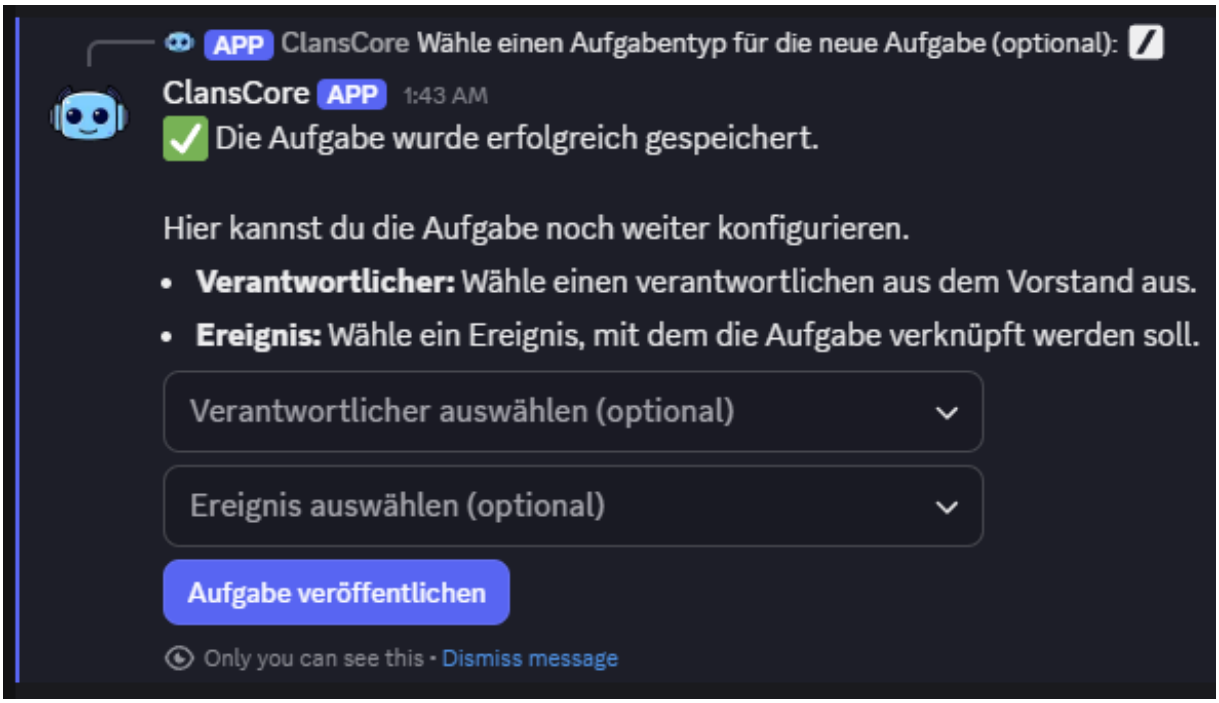


Abb. 49: Discord-Bot: Bestätigung der Erstellung mit optionalen Konfigurationen (Verantwortlicher und Event)

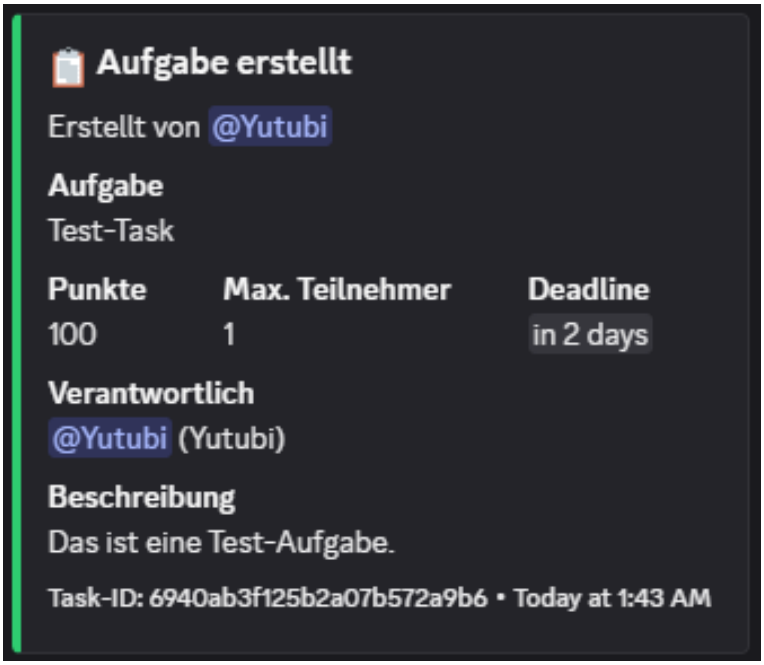


Abb. 50: Discord-Bot: Log-Nachricht der erstellten Aufgabe im Vorstands-Channel `bot-log`

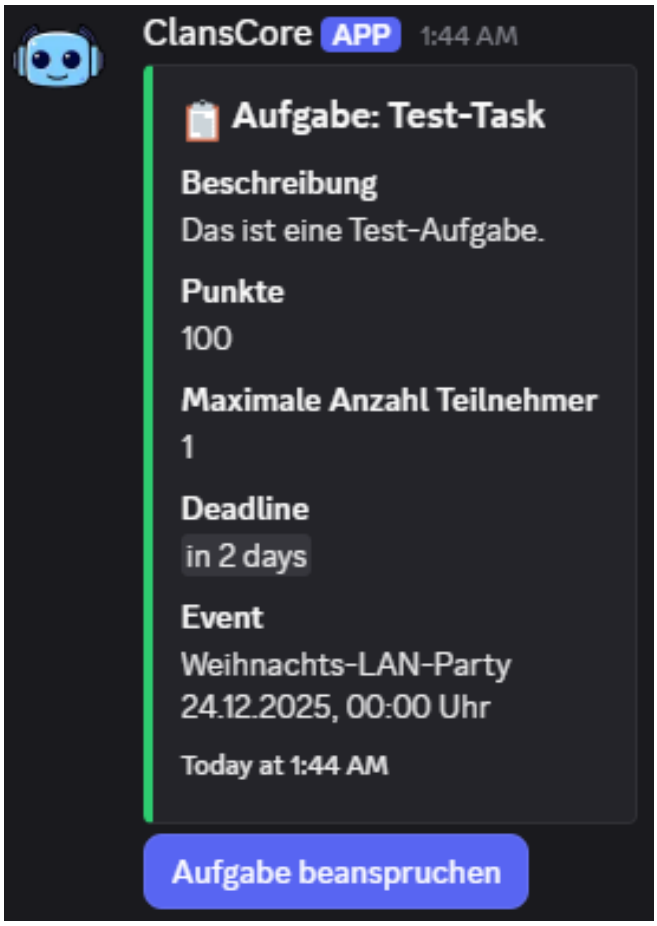


Abb. 51: Discord-Bot: Beanspruchen einer Aufgabe im Channel `aufgaben`

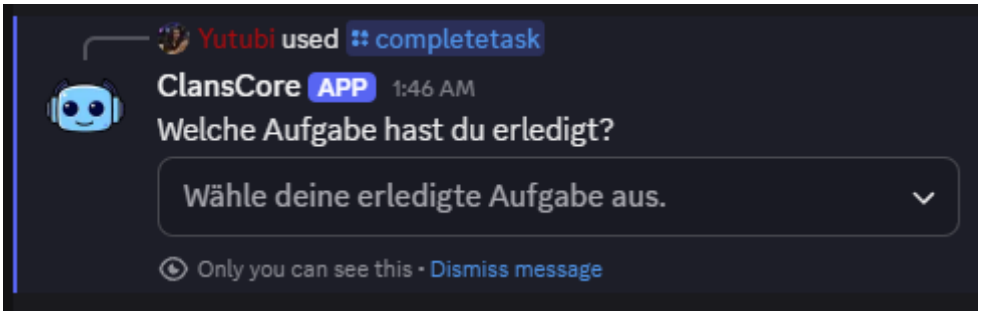


Abb. 52: Discord-Bot: Abschluss einer Aufgabe mittels `/completetask`

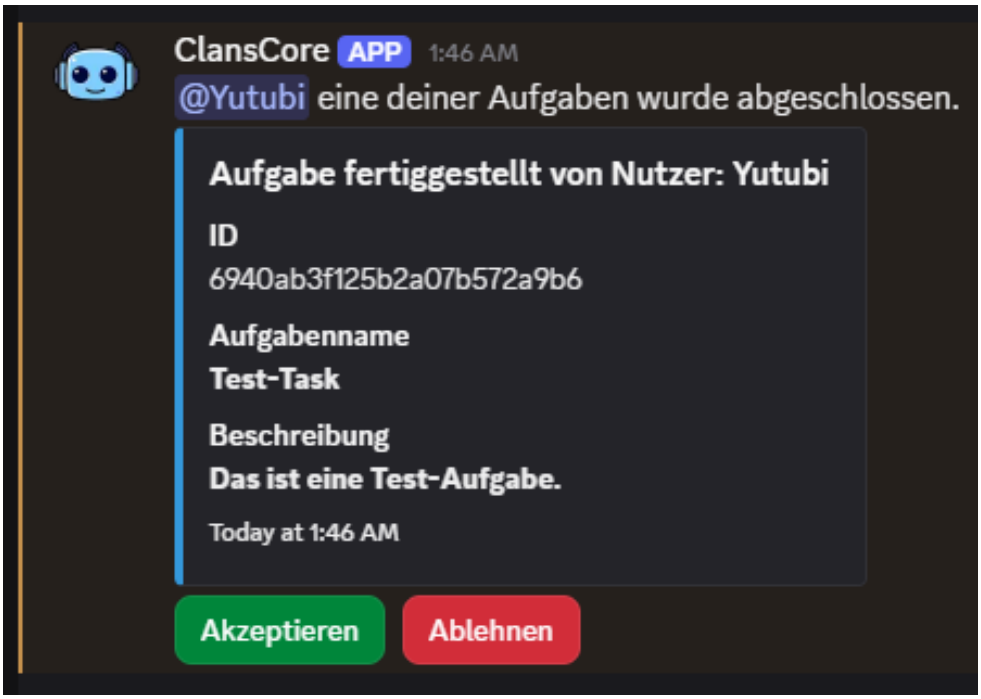


Abb. 53: Discord-Bot: Entscheidungs-Nachricht der fertiggestellten Aufgabe im Vorstands-Channel `bot-aufgaben`

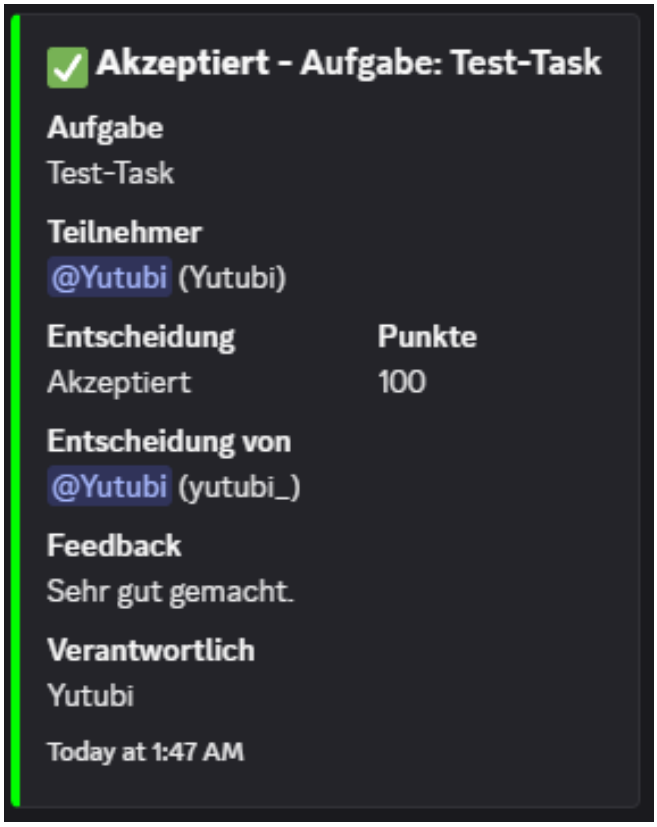


Abb. 54: Discord-Bot: Log-Nachricht der fertiggestellten Aufgabe im Vorstands-Channel `bot-log`

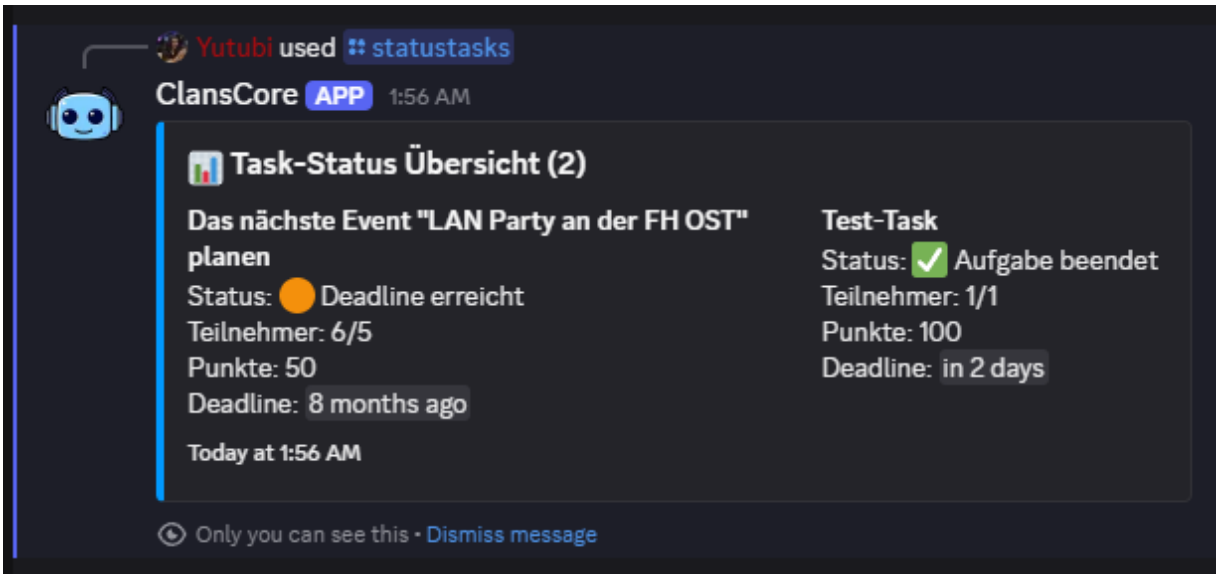


Abb. 55: Discord-Bot: Neuer Vorstands-Befehl `/statustasks` für die Darstellung aller Aufgaben und ihren Status

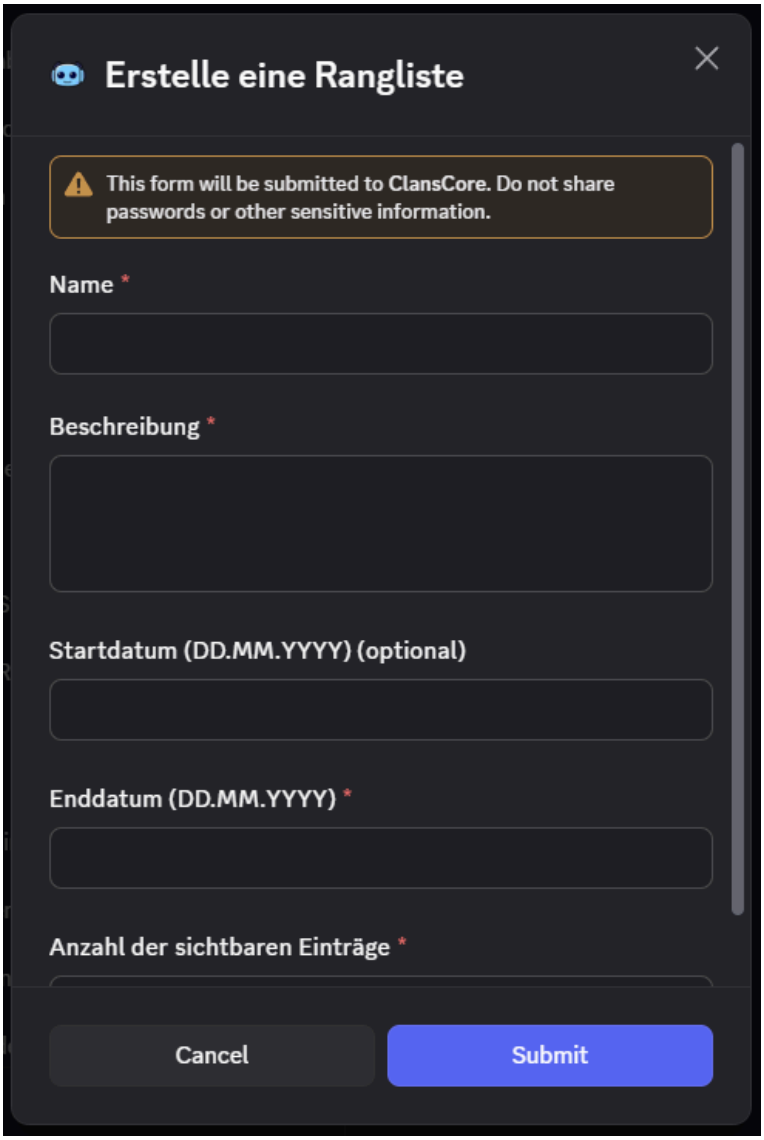


Abb. 56: Discord-Bot: Vorstands-Befehl `/createleaderboard` zur Erstellung neuer Ranglisten

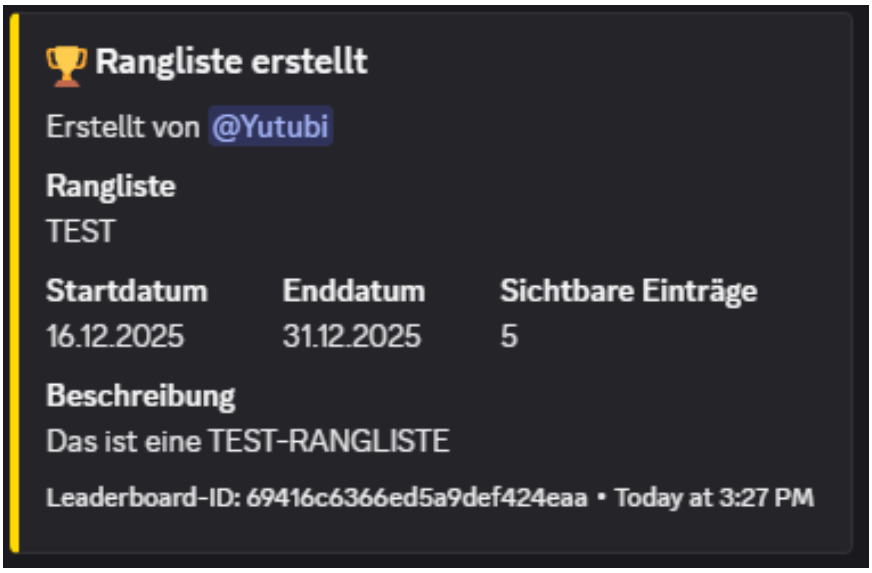


Abb. 57: Discord-Bot: Log-Nachricht der erstellten Rangliste im Vorstands-Channel `bot-log`

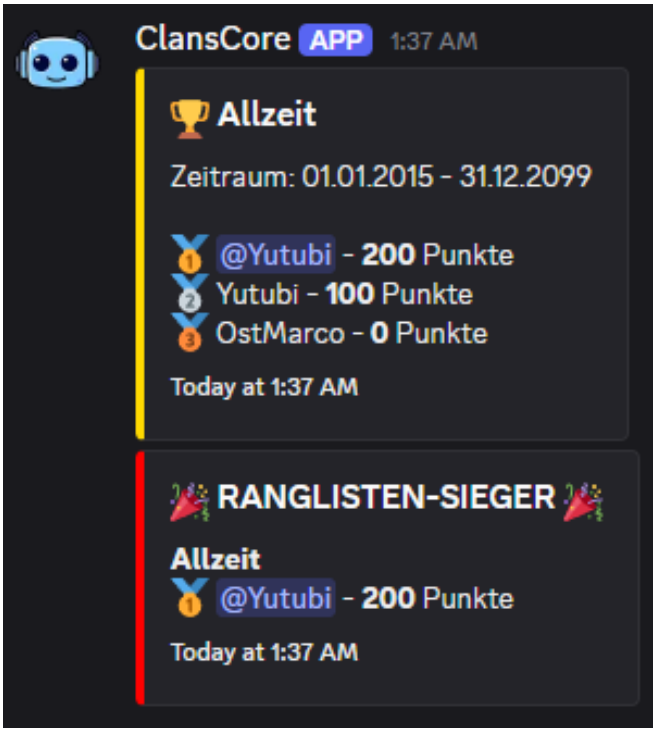


Abb. 58: Discord-Bot: Darstellung der Ranglisten mit den jeweiligen Punktesständen im Channel `rangliste`

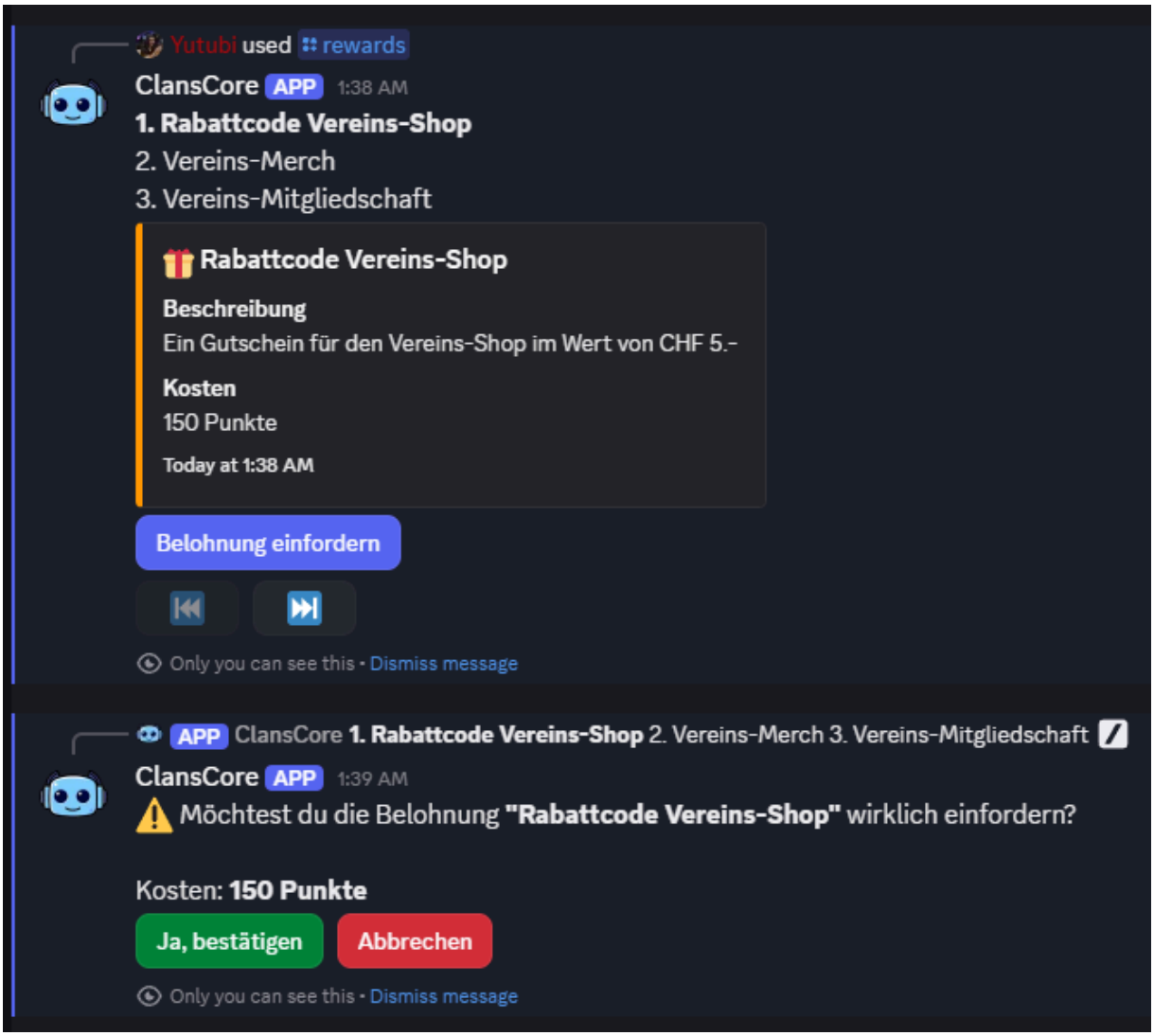


Abb. 59: Discord-Bot: Befehl `/rewards` zur Einlösung der Punkte für Belohnungen

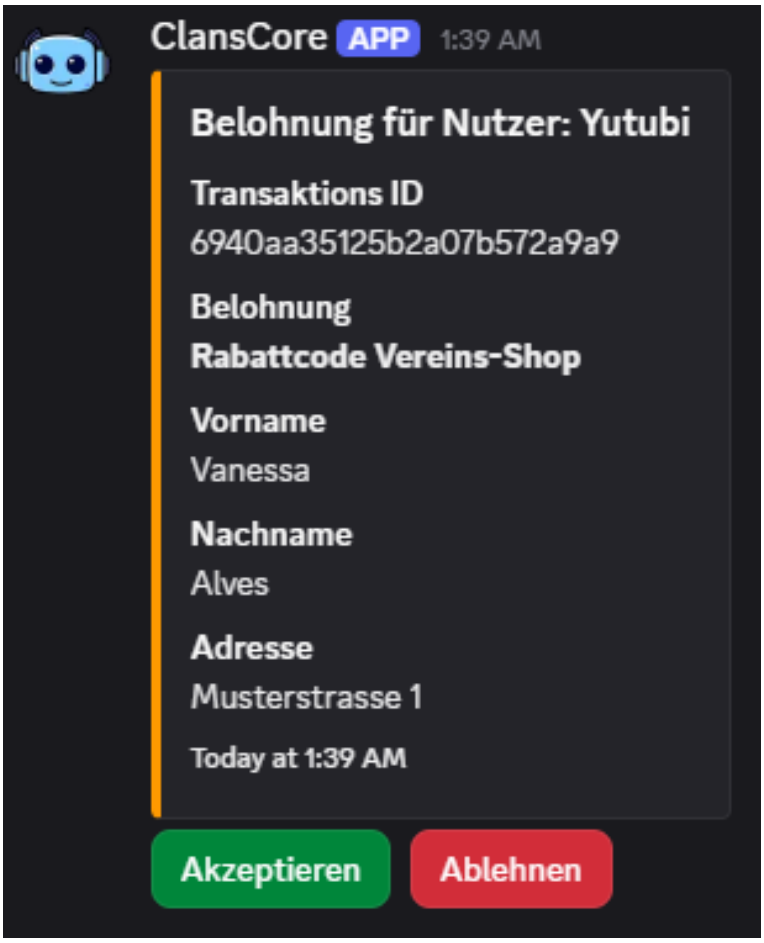


Abb. 60: Discord-Bot: Entscheidungs-Nachricht im Vorstands-Channel `bot-belohnungen`

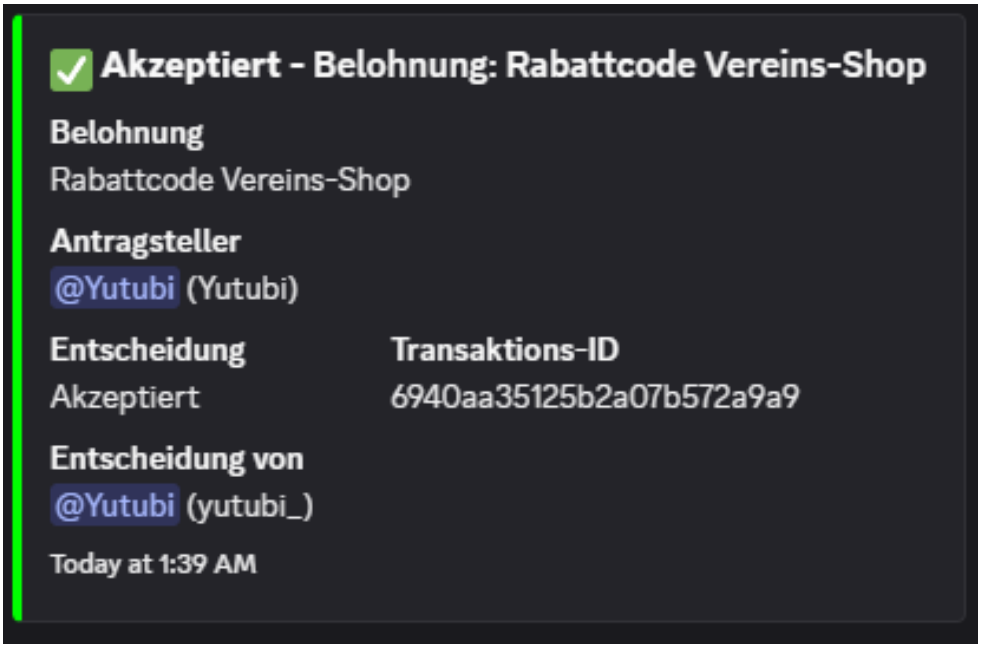


Abb. 61: Discord-Bot: Log-Nachricht der akzeptierten Belohnungs-Einforderung im Vorstands-Channel `bot-log`

17.4 Event- und Kalender-Synchronisation

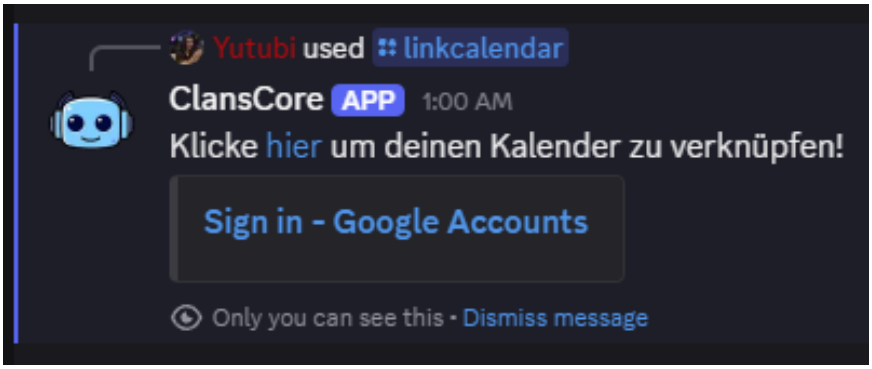


Abb. 62: Discord-Bot: Vorstands-Befehl `/linkcalendar` zur Verknüpfung eines externen Google-Kalenders

Abb. 63: Discord-Bot: Log-Nachricht der Event-Verlinkung und Synchronisierung im Vorstands-Channel `bot-log`

Abb. 64: Discord-Bot: Anzeige der bevorstehenden Vereins-Events im Channel `events` und über den Befehl `/events`

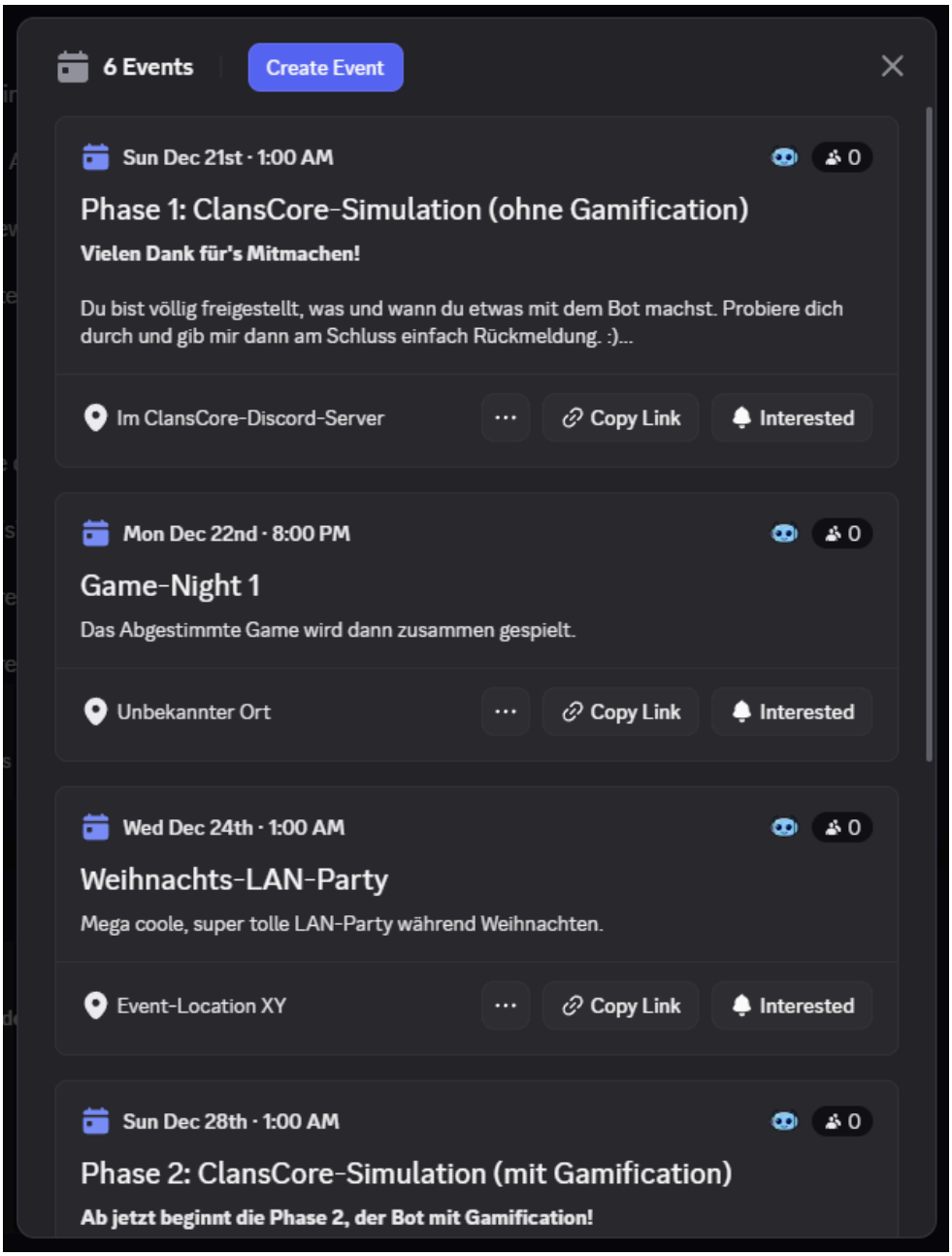


Abb. 65: Discord-Bot: Vereins-Kalender aus der Sicht von Discord

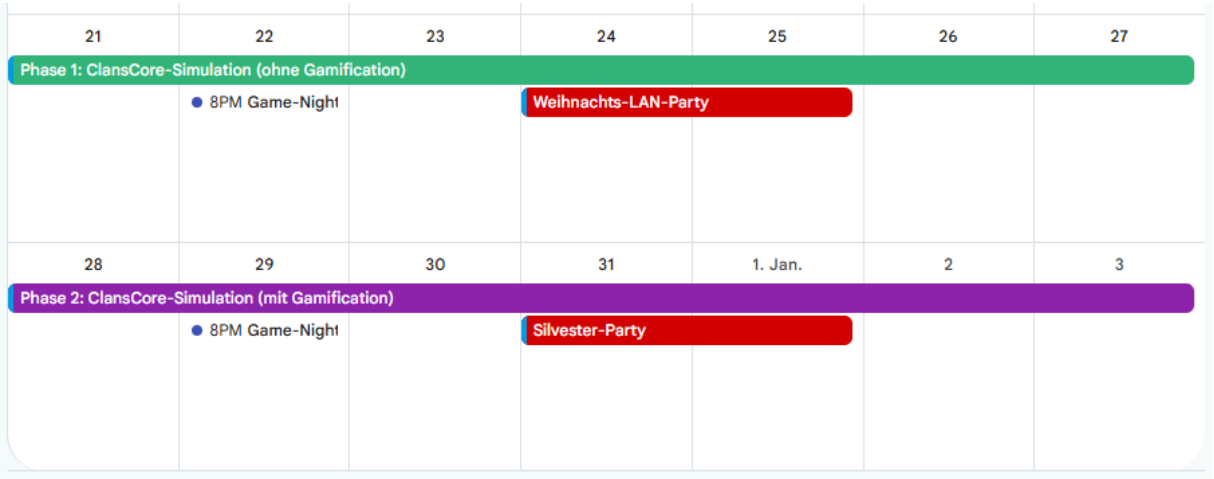


Abb. 66: Discord-Bot: Vereins-Kalender aus der Sicht von Google-Kalender

18 Screenshots Admin-Dashboard

In diesem Kapitel wird ein Überblick über das Dashboard anhand ausgewählter Screenshots gegeben. Dabei werden sämtliche Seiten sowie alle relevanten Masken des Dashboards dargestellt. Die Screenshots dienen ausschliesslich der Veranschaulichung und werden nicht im Detail beschrieben.

18.1 Vereinsverwaltung

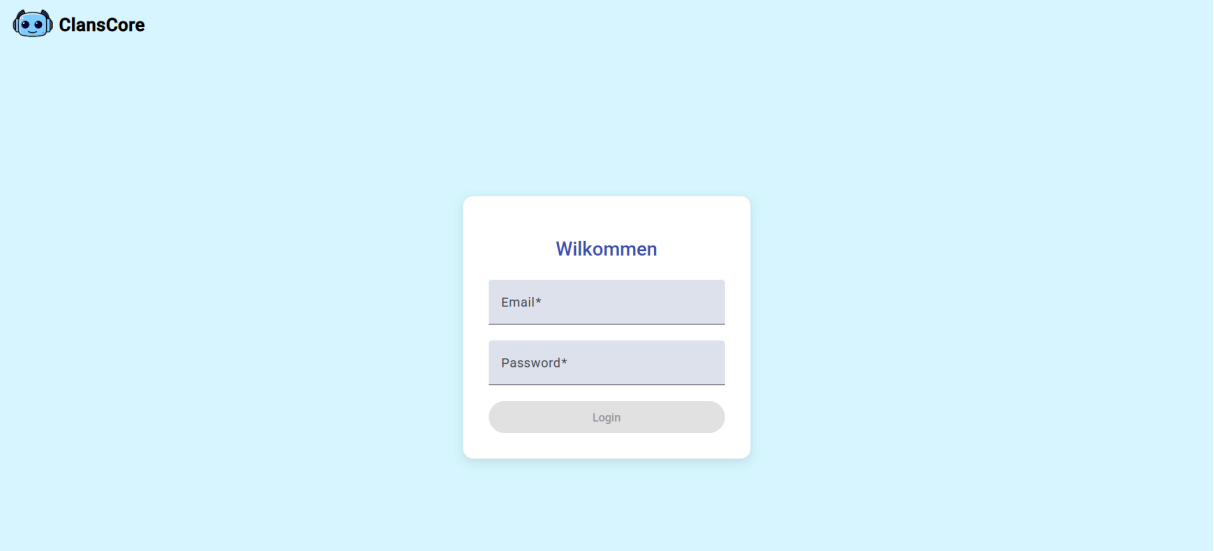


Abb. 67: Admin-Dashboard: Login-Seite

Abb. 68: Admin-Dashboard: Mitgliederverwaltung

Abb. 69: Admin-Dashboard: Modal zu Erfassung neuer Mitglieder

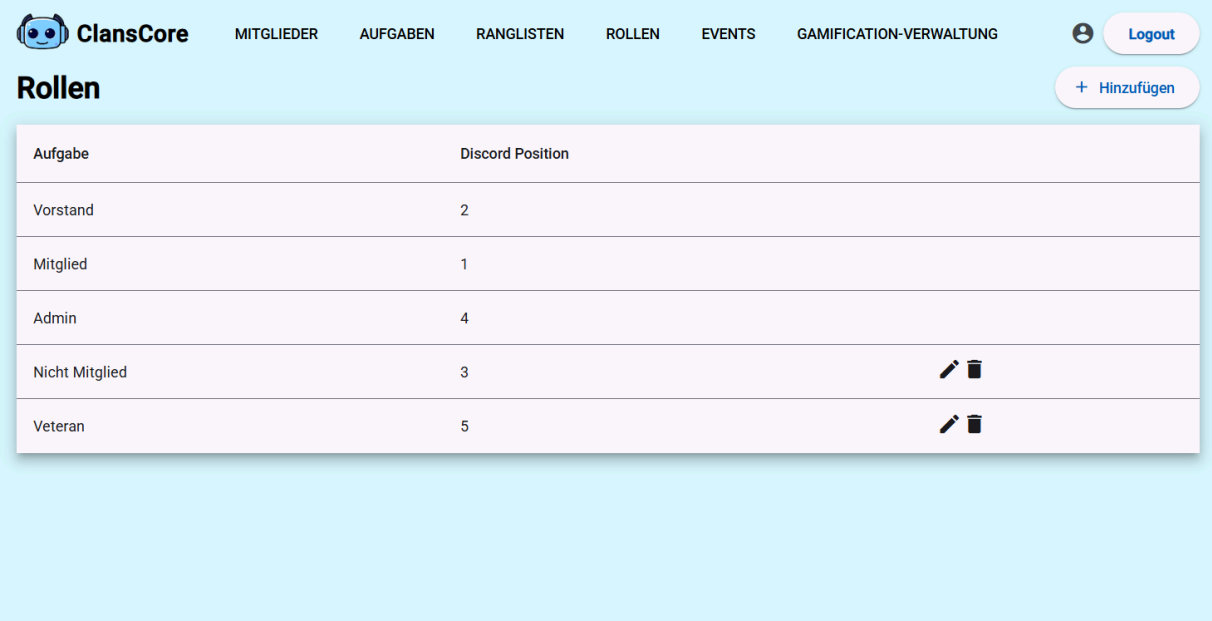


Abb. 70: Admin-Dashboard: Rollenverwaltung

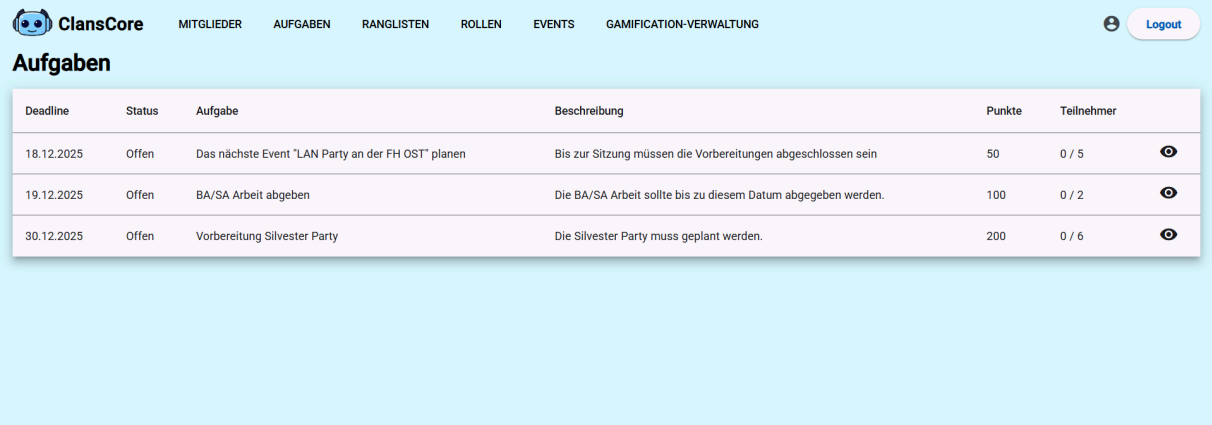


Abb. 71: Admin-Dashboard: Aufgaben-Ansicht

Abb. 72: Admin-Dashboard: Modal für die Detail-Ansicht von Aufgaben

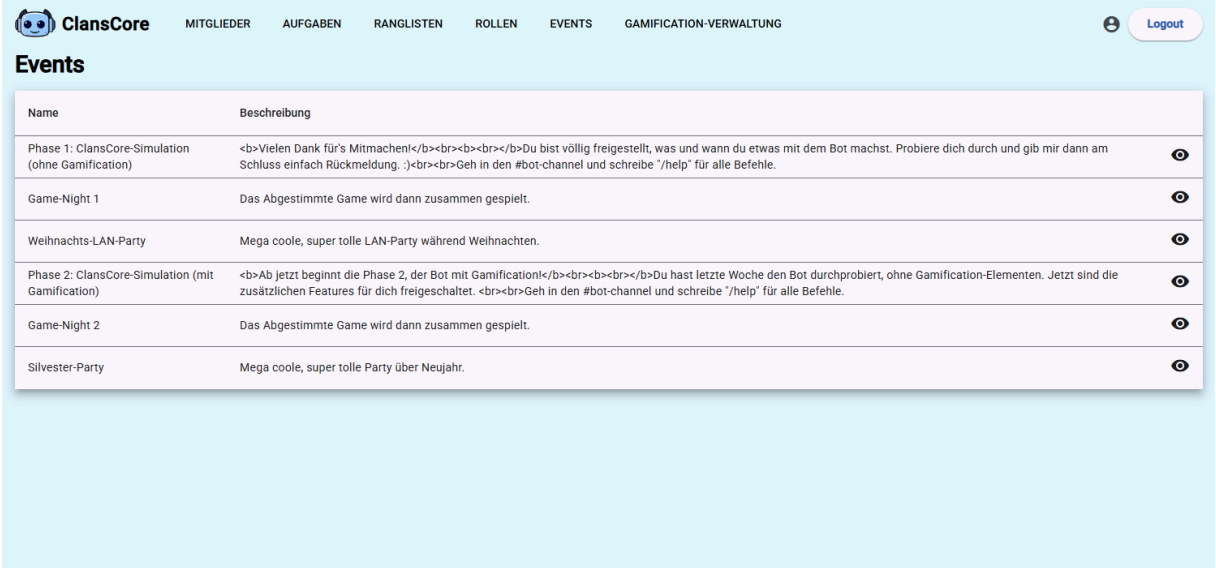


Abb. 73: Admin-Dashboard: Auflistung aller Events

Abb. 74: Admin-Dashboard: Modal für die Detail-Ansicht von Events

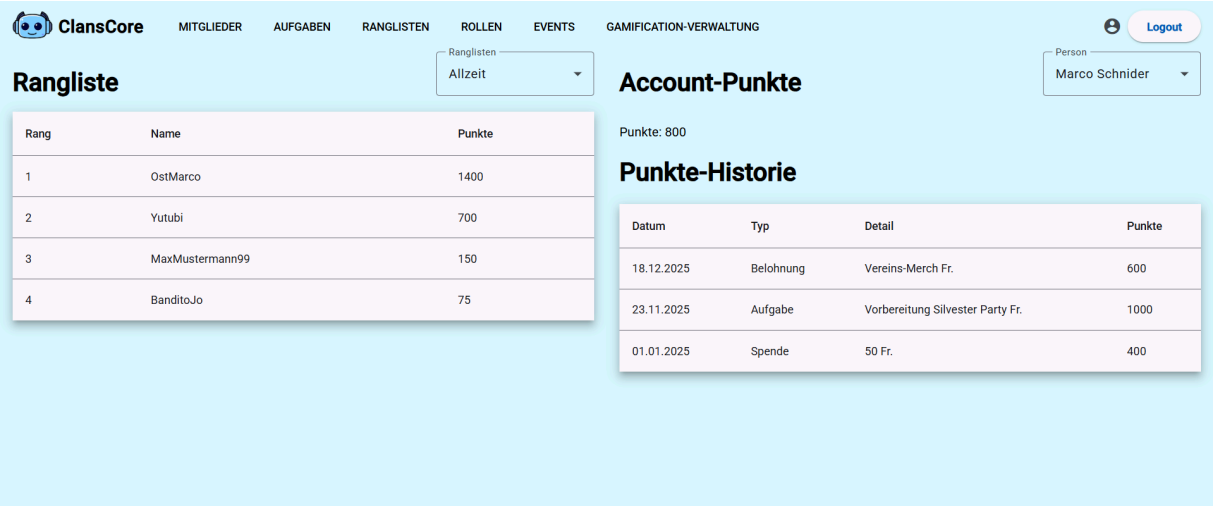


Abb. 75: Admin-Dashboard: Darstellung der Ranglisten und Punktehistorie aller Mitglieder

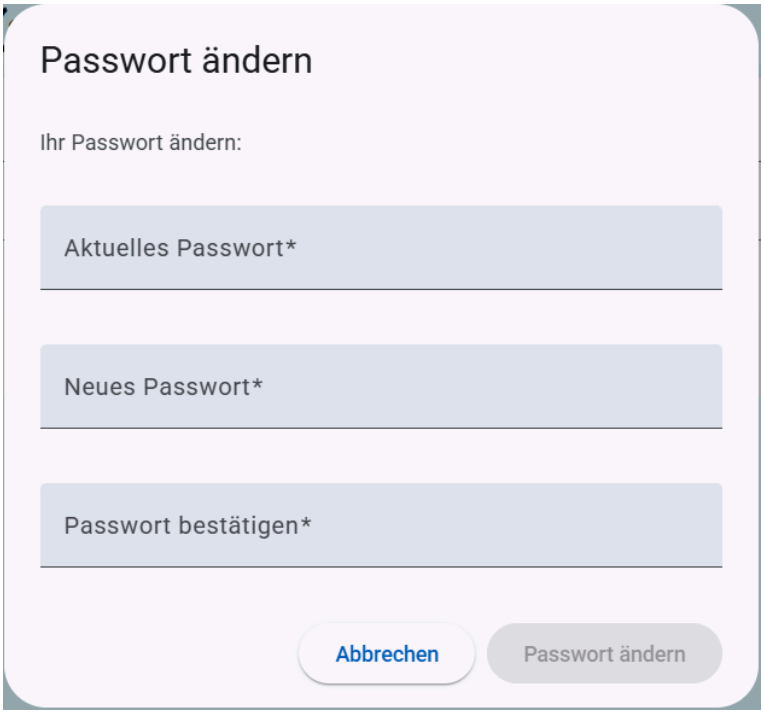


Abb. 76: Admin-Dashboard: Modal für die Passwortänderung

18.2 Admin-Seite

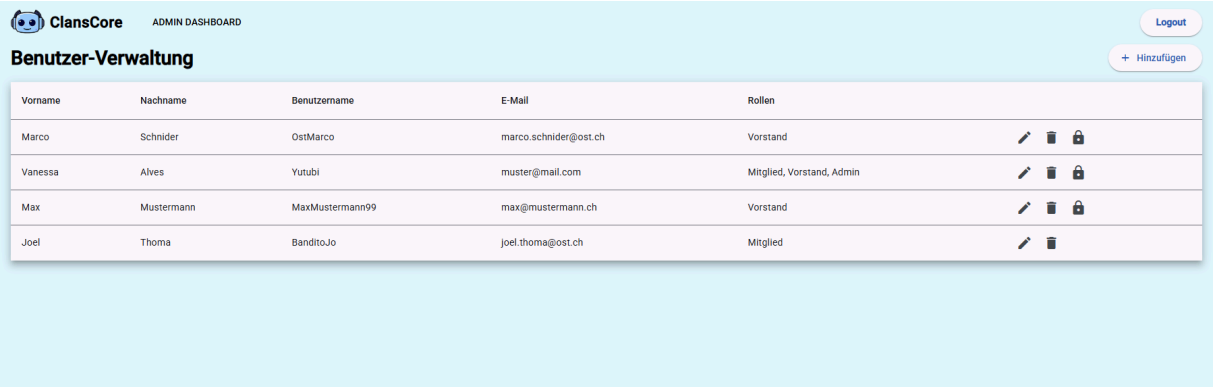


Abb. 77: Admin-Dashboard: Admin-Ansicht für Mitgliederverwaltung und Passwort-Reset

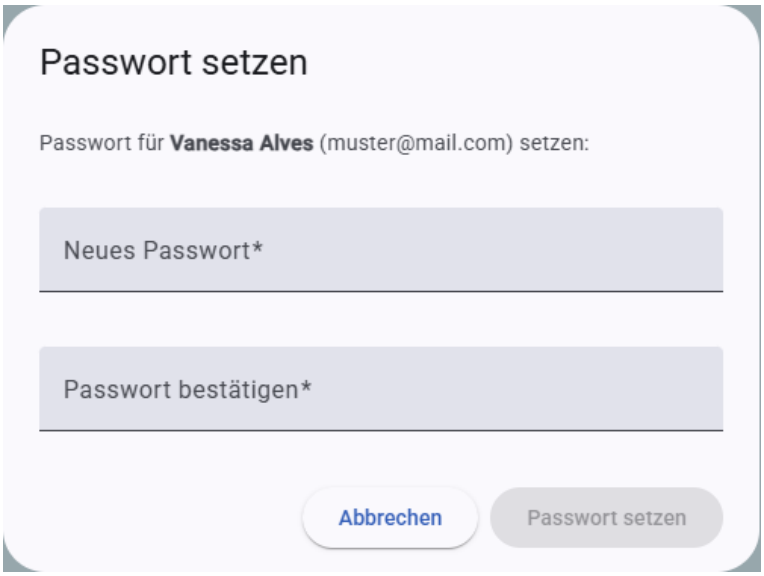


Abb. 78: Admin-Dashboard: Modal für den Passwort-Reset

18.3 Gamification-Verwaltung

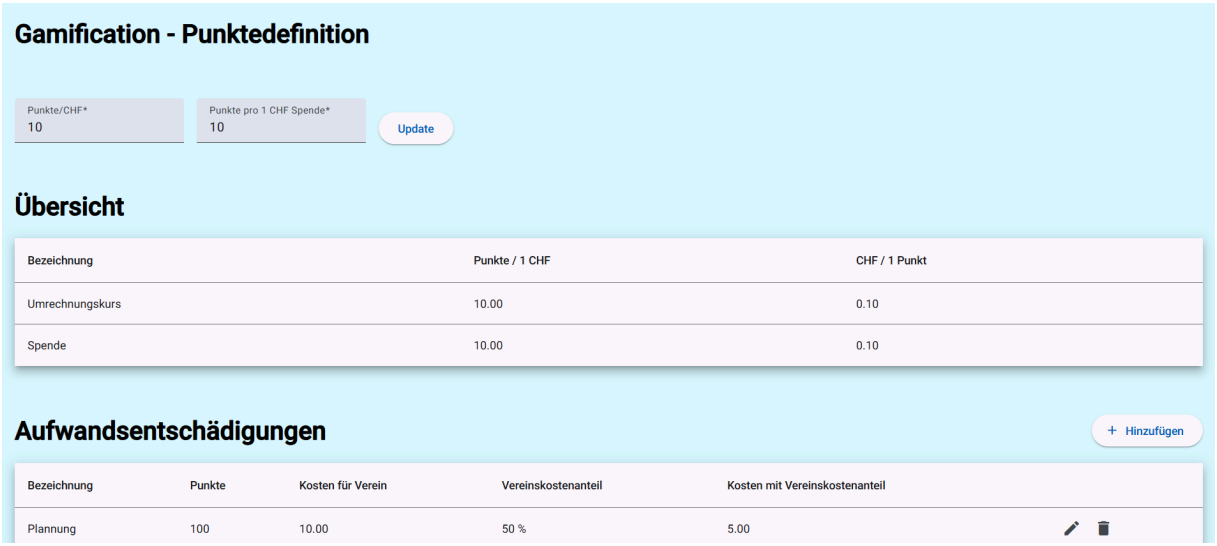


Abb. 79: Admin-Dashboard: Punktedefinitions-Seite zur Verwaltung der Gamification-Parameter und Aufgabentypen

Abb. 80: Admin-Dashboard: Modal für die Verwaltung der Aufgabentypen

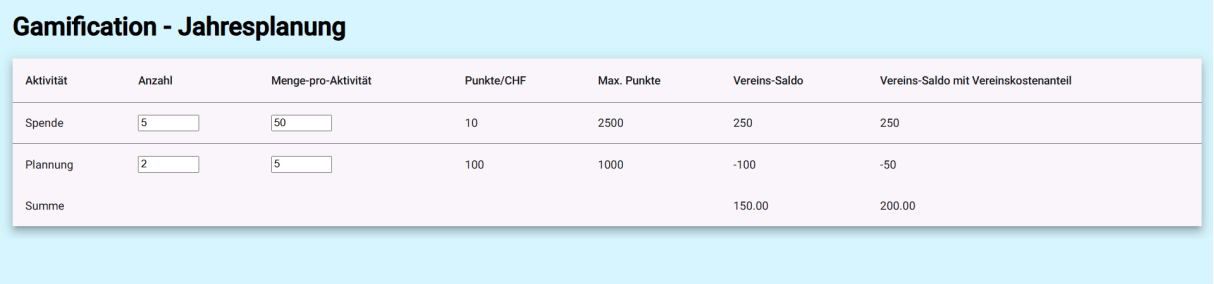


Abb. 81: Admin-Dashboard: Jahresplanung-Seite für strategische Entscheide

Abb. 82: Admin-Dashboard: Belohnungsverwaltung

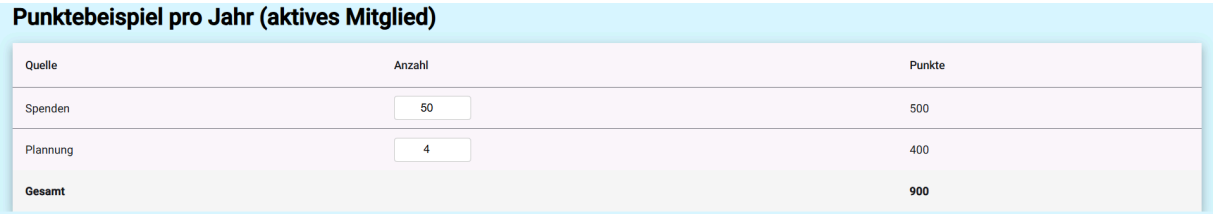


Abb. 83: Admin-Dashboard: Punktebeispiel-Seite für strategische Entscheide