Diplomarbeit

im Fach

Informatiker/in HF Fachrichtung Applikation

Social-Media für DevOps und Software-Engineer

Innovative Ansätze zur Automatisierung und Skalierung von DevOps-Prozessen auf einer Social-Media-Plattform für IT-Profis

Diplomand	Markus Bürgi
Schule	TEKO Schweizerische Fachschule AG
Klasse	O-TIN-21-S/H-a
Ausbildung	Dipl. Informatiker/in HF Fachrichtung Applikation
Jahr	2024

1 INHALTSVERZEICHNIS

1 Inhaltsverzeichnis 2
2 Management Summary 10
3 Kurzer beruflicher Lebenslauf 11
4 Qualifikationsprofil 13
5 Projektinitialisierung 14
5.1 Pflichtenheft 14
5.1.1 Einleitung 14
5.1.2 Fachexperte 14
5.1.3 Richtziel 14
5.1.4 Ziele (Endergebnisse / Erfolgskriterien) 15
5.1.5 Technische Anforderungen 17
5.1.5.1 Backend 17
5.1.5.2 Frontend 17
5.1.5.3 DevOps und Infrastruktur 17
5.1.5.4 Sicherheitsaspekte 17
5.1.5.5 Nachrichtensystem 17
5.1.6 Schematische Skizzen 18
5.1.6.1 Systemarchitektur 18
5.1.6.2 Datenbankdesign 19
5.1.6.3 CI/CD Pipeline 20
5.1.7 Meilensteine und Zeitplan 21
5.1.8 Entwicklungsmethodik 23
5.1.8.1 Arbeitsorganisation 23
5.1.8.2 Aufgabenmanagement 23

5.1.8.3 Entwicklungspraktiken 23
5.1.8.4 Fortschrittsverfolgung und Verbesserung 23
5.1.9 Qualitätssicherung 24
5.1.9.1 Implementierung von automatisierten Tests 24
5.1.9.2 Regelmässige manuelle Tests und Überprüfungen 24
5.1.9.3 Verwendung von Codequalitäts-Tools 24
5.1.9.4 Sicherheitsüberprüfungen 24
5.1.9.5 Performance-Tests 24
5.1.9.6 Risiken und Herausforderungen 24
5.1.10 Abnahmekriterien 25
5.2 Zielscheibe 26
6 Projektplanung 28
6.1 Vorgehensmodell 28
6.1.1 Projektinitialisierung und Planung (1 Woche: 09. - 15. September 2024) 29
6.1.2 Architektur und Design (1 Woche: 16.09.2024 - 20.09.2024) 29
6.1.3 Bewertungsanalyse (1 Tag: 21.09.2024) 29
6.1.4 Implementierung Backend (2 Wochen: 23.09.2024 - 06.10.2024) 29
6.1.5 Implementierung Frontend (1 Woche: 07.10.2024 - 13.10.2024) 29
6.1.6 DevOps und Infrastruktur (1 Woche: 14.10.2024 - 20.10.2024) 30
6.1.7 Integration und Systemtests (1 Woche: 21.10.2024 - 27.10.2024) 30
6.1.8 Dokumentation und Abschluss (1 Woche: 28.10.2024 - 03.11.2024) 30
6.1.9 Abgabe der Diplomarbeit: 04.11.2024 30
6.1.10 Vorbereitung der Präsentation (1,5 Wochen: 05.11.2024 - 14.11.2024) 30
6.1.11 Präsentation der Diplomarbeit: 15.11.2024 30
6.2 Kanban Projektplan für die Social Media Plattform 31

6.3 Projektstrukturplanung 33
6.4 Projektablaufplanung.....34
6.5 Risikoanalyse 35
6.5.1 Risikomatrix 36
6.6 SWOT-Analyse.....37
6.7 UX / UI Konzept.....38
6.7.1 Analyse.....38
6.7.1.1 Zielgruppenanalyse und Segmentierung 38
6.7.1.2 Bevorzugte Social-Media-Plattformen 39
6.7.1.3 Nutzungsverhalten der Social-Media-Plattformen: 40
6.7.1.4 Wichtige Erkenntnisse: 40
6.7.1.5 Kernstrategien für ein erfolgreiches Social Media Marketing.....41
6.7.1.6 Best Practice Beispiel einer Marketingstrategie.....42
6.7.1.7 Erfolgsfaktoren der Entwickelnden 42
6.7.2 Informationsarchitektur 43
6.7.3 Visuelle Gestaltung.....44
6.7.4 Farbkonzept der Benutzeroberfläche.....47
7 Projektrealisierung 50
7.1 Kreativitätsmethode.....50
7.1.1 Umsetzung der adaptierten Mindmapping-Methode 50
7.2 Variantenbildung 52
7.2.1 Variante «Microservices skalierbare Anwendungen» 52
7.2.1.1 Vorteile der «Microservices skalierbare Anwendungen»: 52
7.2.1.2 Nachteile der «Microservices skalierbare Anwendungen»: 53
7.2.2 Variante «Monolithische Architektur für integrierte Anwendungen».....54

7.2.2.1 Vorteile der «Monolithische Architektur für integrierte Anwendungen»: 54

7.2.2.2 Nachteile «Monolithische Architektur für integrierte Anwendungen»: ...54

7.2.3 Variante «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen» 55

7.2.3.1 Vorteile «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen»: 55

7.2.3.2 Nachteile «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen»: 55

7.3 Priorisierungsmethode 56

7.3.1 Begründung Priorisierungsmatrix: 56

7.3.2 Vorgehen Priorisierungsmatrix: 56

7.4 Argumentation Microservices-Architektur 58

7.4.1 Kostenfaktor: 58

7.4.2 Kontrolle über die Infrastruktur: 58

7.4.3 Langfristige Flexibilität und Anpassungsfähigkeit: 58

7.4.4 Unabhängigkeit von Cloud-Anbietenden: 59

7.4.5 Lokalität des Betriebs: 59

7.4.6 Entwicklerproduktivität und lokale Entwicklung: 59

7.5 Bewertungsanalyse 61

7.6 Argumentation Backend und Frontend 68

7.6.1 Backend: «Go» 68

7.6.2 Frontend: «SvelteKit» 68

7.7 CI/CD-Pipeline 70

7.7.1 «Jenkins» 70

7.7.2 «ArgoCD» 70

7.7.3 «Kubernetes» 70

7.7.4 «Traefik» als Ingress Controller 70

7.8 UML....72
7.8.1 Sequenzendiagramm....72
7.8.2 Aktivitätsdiagramm....73
7.9 Vorbereitung....74
7.9.1 Voraussetzungen....74
7.9.2 Übersicht von «Prometheus», «Grafana», «Helm» und «K3s»....74
7.10 Datenbank....76
7.10.1 Entity-Relationship-Diagramm....76
7.10.2 Datenbank-Implementierung....77
7.10.3 Datenbank-Beziehungen und Fremdschlüssel....78
7.10.4 Evaluierung der Datenbank....79
7.11 Backend-Implementierung....80
7.11.1 Modell....82
7.11.1.1 Ein Beispiel für eine Struktur mit mehreren Modellen:....82
7.11.1.2 Beispiel: Mehrere Modelle in «Go»....83
7.11.2 Repository....84
7.11.2.1 Beschreibung der Komponenten des Repository-Musters....84
7.11.2.2 Zusammenfassung des Repository-Musters:....85
7.11.3 Service....86
7.11.3.1 Beschreibung der Geschäftslogik im Service:....86
7.11.3.2 Service-Architektur....87
7.11.4 Handler....88
7.11.4.1 Hauptaufgaben des Handlers:....88
7.11.4.2 Vorteile einer klaren Handler-Architektur:....90
7.11.5 RabbitMQ Producer....90

7.11.5.1 Beschreibung der Komponenten:.....91
7.11.6 RabbitMQ Consumer 91
7.11.6.1 Beschreibung der Komponenten eines RabbitMQ-Consumers:.....92
7.11.6.2 Generische Beschreibung eines RabbitMQ Consumers: 93
7.11.7 Redis.....94
7.11.7.1 Beschreibung des Redis: 94
7.11.8 PostgreSQL-Datenbankverbindung 95
7.11.8.1 Beschreibung der PostgreSQL-Datenbankverbindung: 96
7.11.9 Middleware.....96
7.11.9.1 Middleware-Komponente: RequestLogger.....97
7.11.9.2 Middleware-Komponente: Recover Middleware.....98
7.11.9.3 Middleware-Komponente: CORS (Cross-Origin Resource Sharing) mit Config 98
7.11.9.4 Middleware-Komponente: Gzip.....98
7.11.9.5 Middleware-Komponente: Rate Limiter.....99
7.11.9.6 Middleware-Komponente: Secure Middleware.....99
7.11.10 Tests und Validierungen des Backends 100
7.11.10.1 «API-Tests» mit Postman Dashboard 100
7.11.10.2 Überwachung von PostgreSQL, RabbitMQ und Redis 104
7.11.11 Sicherheitsaspekte des Backends 110
7.11.11.1 JWT-Authentifizierung in GO 110
7.11.11.2 Erklärung des Codes: 111
7.11.11.3 Sicherheitsaspekte: 112
7.12 Frontend-Implementierung 113
7.12.1 Startseite.....113

7.12.2 Registrierung 114
7.12.3 Login-Seite 115
7.12.4 JSON Web Tokens 116
7.12.5 Profil 117
7.12.6 Neuer Beitrag 118
7.13 CI/CD-Pipeline Implementierung 119
7.13.1 Jenkins 119
7.13.1.1 Wichtige Funktionen: 119
7.13.1.2 Netzwerk und Aufbau «Jenkins» 120
7.13.2 SonarQube 130
7.13.2.1 Anwendung «SonarQube» 130
7.13.3 «K3s» (Kubernetes) und «K9s»-Managementtool 136
7.13.4 «ArgoCD» 138
7.13.4.1 Installation «ArgoCD» 139
7.13.4.2 Erstellung der «ArgoCD-Anwendung» 141
7.13.4.3 Synchronisation 143
7.13.5 Traefik 145
7.13.5.1 Load Balancer 146
7.13.5.2 API-Gateway 146
7.13.5.3 Ingress-Routing 147
7.13.5.4 Sicherheit und Zertifikate 148
8 Projektabschluss 149
8.1 Projektüberwachung 149
8.2 Evaluation der Zielerreichung 150
8.3 Reflexion Weg zum Ziel 150

8.4 Lessons learnt 150
8.5 Ausblicke 152
9 Eigenständigkeitserklärung 153
10 Verzeichnisse 154
10.1 Abkürzungsverzeichnis 154
10.2 Abbildungsverzeichnis 155
10.3 Tabellenverzeichnis 157
10.4 Diagrammverzeichnis 158
10.5 Programmierverzeichnis 158
10.6 Literatur- und Quellenverzeichnis 161
11 Anhang 164
11.1 Projektstatusberichte 164
11.2 Meetingprotokolle 168

2 MANAGEMENT SUMMARY

Die vorliegende Arbeit befasst sich mit der Entwicklung einer Social-Media-Plattform für DevOps-Experten:innen und Softwareentwickelnde, die den Wissensaustausch und die Zusammenarbeit in der IT-Community fördern soll. Ziel des Projekts ist es, eine robuste und skalierbare Plattform bereitzustellen, die moderne DevOps- und Softwareentwicklungspraktiken unterstützt.

Ausgangslage

Angesichts der wachsenden Bedeutung von DevOps und der zunehmenden Nachfrage nach speziellen Plattformen zur Förderung eines Wissensaustauschs beziehungsweise der Zusammenarbeit wurde diese Plattform konzipiert. Die Problemstellung liegt darin, eine Umgebung zu schaffen, die technische Herausforderungen von IT-Experter adressiert und dabei hilft, den Wissensaustausch zu fördern und Kompetenzen zu bündeln. Das Hauptziel lautet die Entwicklung einer zentralen Anlaufstelle zu erstellen, die sowohl funktional als auch skalierbar ist und zukünftige Anforderungen der IT-Community abdeckt.

Vorgehen

Im Verlauf des Projekts wurden wesentliche Schritte wie die Projektinitialisierung, Backend- und Frontend-Entwicklung, CI/CD-Integration und die Bereitstellung in einer Kubernetes-Umgebung durchgeführt. Für die Entwicklung kamen unter anderem «Golang», «SvelteKit», «Docker», «Jenkins», «Treafik» und «ArgoCD» zum Einsatz. Durch eine strukturierte Arbeitsweise und die Anwendung von agilen Prinzipien wurde eine effiziente Umsetzung sichergestellt. Die Methoden und Werkzeuge wurden sorgfältig ausgewählt und kontinuierlich auf ihre Eignung überprüft, um die hohen Anforderungen an Performance und Skalierbarkeit zu erfüllen.

Ergebnisse

Als Hauptresultate wurden ein voll funktionsfähiges Backend, ein interaktives Frontend, sowie eine stabile CI/CD-Pipeline realisiert, die automatisierte Tests und Deployments ermöglicht. Die Plattform bietet Benutzerprofile mit Authentifizierung und Autorisierung, ein Echtzeit-Kommunikationssystem, ein Projektmanagement-Tool sowie ein Message Broker zur sicheren Verarbeitung von Nachrichten. Die definierten Ziele wurden überwiegend erreicht, wobei die Plattform ein Produkt darstellt, welches für zukünftige Erweiterungen vorbereitet ist.

Ausblick

Für die Weiterentwicklung der Plattform bieten sich mehrere Alternativen an, darunter die Implementierung zusätzlicher Kommunikations- und Kollaborationsfunktionen sowie die Optimierung der Infrastruktur für eine grössere Benutzeranzahl. Weiterhin ist die Erhöhung der Testabdeckung geplant, um die langfristige Zuverlässigkeit und Stabilität der Plattform sicherzustellen. Langfristig könnte die Plattform als Open-Source-Projekt zur Förderung der IT-Community bereitgestellt werden.

3 KURZER BERUFLICHER LEBENSLAUF

Persönliche Daten:

- **Name:** Markus Bürgi
- **Geburtsdatum:** 4. Dezember 1986
- **Familienstand:** Verheiratet
- **Kinder:** 2 Töchter
- **Sprachkenntnisse:**
  - **Deutsch:** Sehr gut
  - **Englisch:** Gut
  - **Japanisch:** Gut
  - **Deutschschweizerische Gebärdensprache:** Sehr gut

Beruflicher Werdegang:

- **Software-Engineer, Linguala (unbezahlte, Remote)**
  Seit Oktober 2023
  Tätigkeiten: Fullstack-Entwicklung mit TypeScript, SvelteKit, MongoDB und GraphDB. Mitarbeit an der Entwicklung einer Plattform zur Vermittlung von Übersetzern. Involviert in die gesamte Entwicklungs-Pipeline, von der Backend-Architektur bis zur Frontend-Implementierung.
- **IT-Leiter, Schweizerischer Gehörlosenbund, Zürich**
  Juni 2019 – Februar 2023
  Tätigkeiten: Leitung der IT-Abteilung, Planung und Durchführung von IT-Projekten, Verwaltung der Netzwerkinfrastruktur, Zusammenarbeit mit verschiedenen Abteilungen zur Sicherstellung eines effizienten IT-Betriebs.
- **Junior SAP ABAP-Entwickler, Alpiq AG, Olten**
  Juni 2017 – Februar 2018
  Tätigkeiten: Entwicklung und Wartung von SAP-ABAP-Anwendungen, Durchführung von Systemanalysen und Performance-Optimierungen.

Ausbildung:

- **Dipl. Techniker HF Informatik (Applikation), TEKO Schweizerische Fachschule Olten**
  Oktober 2021 – Oktober 2024
  Vertiefung in Applikationsentwicklung und Software-Engineering.
- **Professional Software-Engineering, Wilhelm Büchner Hochschule, Darmstadt**
  Juli 2019 – April 2021
  Fernstudium mit Fokus auf moderne Software-Entwicklungsprozesse.
- **Geprüfter Informatiker, Fernstudium SGD, Darmstadt**
  November 2017 – Juli 2019
  Schwerpunkte: Grundlagen der Informatik, Mengenlehre, Aussagenlogik, Softwareentwicklung mit C#, Datenbankentwicklung mit PHP/MySQL, Netzwerkbetrieb, Software-Engineering, Java-Programmierung, Robotik mit Lego, Künstliche Intelligenz.

4 QUALIFIKATIONSPROFIL

Handlungsfeld / Fachbereich: ICT-Projekte planen und umsetzen (A3)

Handlung: Planung und Durchführung von ICT-Projekten unter Berücksichtigung von Ressourcen, Zeitrahmen und Qualitätsstandards (A3.2)

Arbeitsergebnisse: Strukturierte Umsetzung von ICT-Projekten unter Einhaltung der Vorgaben

Qualität: Hohe Präzision und Zielorientierung in der Projektdurchführung

Umfeld: Planung und Steuerung von Projekten im ICT-Bereich

Unterrichtsfach / Grundlagenwissen: Projektmanagement

Handlungsfeld / Fachbereich: IT-Sicherheit sicherstellen (B12)

Handlung: Sicherheitsstandards für IT-Infrastrukturen definieren und umsetzen (B12.1)

Arbeitsergebnisse: Konzeption und Umsetzung von Sicherheitsrichtlinien zur Minimierung der Risiken

Qualität: Erfüllung hoher Sicherheitsstandards und Schutz sensibler Daten

Umfeld: Sicherheitsmanagement und Datenschutz in der IT

Unterrichtsfach / Grundlagenwissen: IT-Sicherheit

Handlungsfeld / Fachbereich: System- und Netzwerkadministration (B10)

Handlung: Administration und Betrieb von Netzwerken und Systemen sicherstellen (B10.3)

Arbeitsergebnisse: Gewährleistung der Systemstabilität und Netzwerkverfügbarkeit

Qualität: Hohe Zuverlässigkeit und Effizienz im Betrieb der Infrastruktur

Umfeld: Administration und Wartung von Netzwerken und Systemen

Unterrichtsfach / Grundlagenwissen: Netzwerkbetriebssysteme

Handlungsfeld / Fachbereich: Software-Entwicklung und -Wartung (B6)

Handlung: Anwendung von Entwicklungsmethoden und -werkzeugen zur Softwareerstellung (B6.4)

Arbeitsergebnisse: Implementierung und Optimierung von Softwarelösungen

Qualität: Sicherstellung von Effizienz und Benutzerfreundlichkeit durch moderne Entwicklungsmethoden

Umfeld: Softwareentwicklung und kontinuierliche Wartung zur Sicherung von Stabilität und Qualität

Unterrichtsfach / Grundlagenwissen: Web-Engineering

5 PROJEKTINITIALISIERUNG

5.1 PFLICHTENHEFT

5.1.1 Einleitung

Derzeit bin ich noch auf Arbeitssuche und möchte diese Zeit gerne nutzen, um meine beruflichen Fähigkeiten durch ein persönliches Projekt weiterzuentwickeln. Die Idee für dieses Projekt entsprang anhand meines Wunsches, eine eigene Karriere im Bereich DevOps und Softwareentwicklung zu verfolgen. Ich hoffe, anhand der Umsetzung dieser Diplomarbeit meine Kenntnisse zu vertiefen und gleichzeitig ein Produkt zu erarbeiten, welches einen Mehrwert für die DevOps-Community bietet.

5.1.2 Fachexperte

Wie bereits erwähnt, bin ich derzeit auf Arbeitssuche und kann Ihnen deshalb leider keinen festen Fachexperten zur Verfügung stellen. Ich möchte Ihnen jedoch eine andere Möglichkeit aufzeigen: Die Rolle könnte auch durch einen Mentor oder Berater besetzt werden, der über fundiertes Expertenwissen im Bereich Softwareentwicklung und DevOps verfügt. Eine solche Person könnte beispielsweise über Verbindungen zur TEKO in Olten gefunden werden, da diese Institution gut mit der IT-Branche vernetzt ist. Ein solcher Experte könnte mich in technischen Fragen unterstützen und dazu beitragen, dass die Arbeit den branchenüblichen Standards entspricht.

5.1.3 Richtziel

Entwicklung einer robusten und skalierbaren Social Media Plattform für DevOps und Softwareentwickler, die Zusammenarbeit und Wissensaustausch fördert. Da ich dieses Projekt allein umsetze, liegt der Fokus darauf, meine technischen Fähigkeiten und mein Projektmanagement eigenständig zu entwickeln, um ein funktionsfähiges und professionelles Produkt zu schaffen.

5.1.4 Ziele (Endergebnisse / Erfolgskriterien)

- Implementierung von Benutzerprofilen mit Authentifizierung und Autorisierung.
- Entwicklung eines Echtzeit-Kommunikationssystems (Chat und Foren).
- Integration eines Projektmanagement-Tools mit Aufgabenverwaltung.
- Implementierung einer Code-Sharing-Funktionalität mit Versionskontrolle.
- Einrichtung einer CI/CD-Pipeline für automatisierte Tests und Deployments.
- Entwicklung von Monitoring-Tools für Projekte und Systeme.
- Bereitstellung der Plattform in einer Kubernetes-Umgebung für Skalierbarkeit.
- Implementierung eines Message Brokers (RabbitMQ) zur zuverlässigen Verarbeitung von Nachrichten und Ereignissen.

Aspekt	Ziel und Zweck	Endergebnisse	Erfolgskriterien	Resultat
Hauptziel	Entwicklung einer skalierbaren Plattform für DevOps und Softwareentwickler zur Förderung von Zusammenarbeit und Wissensaustausch			
Benutzerprofile		Sichere Profile mit OAuth 2.0 Authentifizierung und Autorisierung		
Echtzeit-Kommunikation		Chat- und Forensystem für sofortige Kommunikation		
Projektmanagement		Integriertes Tool zur Verwaltung von Projekten und Aufgaben		
Code-Sharing		Funktionalität zum Teilen und Versionieren von Code		
CI/CD-Pipeline		Automatisierte Pipeline für Tests und Deployments		
Monitoring		Werkzeuge zur Überwachung der Systemleistung		
Kubernetes-Umgebung		Bereitstellung für Skalierbarkeit und Leistungsfähigkeit		
Message Broker (RabbitMQ)		System zur Verwaltung von Nachrichten und Ereignissen		
Funktionalität			Alle Module und Tools funktionieren wie vorgesehen	
Skalierbarkeit			Plattform bewältigt zusätzliche Benutzer und erhöhte Last	
Sicherheit			Sichere Authentifizierung, Autorisierung und Kommunikation	
Stabilität und Zuverlässigkei			Minimale Ausfallzeiten, zuverlässige Funktionen	
Benutzerakzeptanz			Plattform wird als nützlich und benutzerfreundlich wahrgenommen	
Automatisierung			Erfolgreiche automatisierte Tests und Deployments	
Endprodukt				Funktionsfähige Plattform für effiziente Entwicklerzusammenarbeit
Technologieeinsatz				Einsatz moderner Technologien für Leistung und Skalierbarkeit

Tabelle 1 – Ziele

5.1.5 Technische Anforderungen

Die technische Umsetzung der Social-Media-Plattform setzt eine hochmoderne Architektur und ausgewählte Technologien voraus, um den Anforderungen an Skalierbarkeit, Performance und Sicherheit gerecht zu werden. Die wichtigsten technischen Anforderungen umfassen:

5.1.5.1 Backend

- **Programmiersprache:** Golang
- **Datenbank:** PostgreSQL (primär), optional Redis zur Performance-Optimierung

5.1.5.2 Frontend

- **Framework:** SvelteKit

5.1.5.3 DevOps und Infrastruktur

- **Containerisierung:** Docker
- **Orchestrierung:** K3s (Kubernetes)
- **CI/CD:** Jenkins und ArgoCD
- **Monitoring:** Prometheus und Grafana
- **Load Balancing und API-Gateway:** NGINX

5.1.5.4 Sicherheitsaspekte

- **Authentifizierung:** JWT
- **Autorisierung:** Role-Based Access Control (RBAC)
- **Sichere Kommunikation:** HTTPS/TLS

5.1.5.5 Nachrichtensystem

- **Message Broker:** RabbitMQ

5.1.6 Schematische Skizzen

Um die Architektur und Struktur der Social-Media-Plattform übersichtlich darzustellen und das Zusammenspiel der einzelnen Komponenten zu veranschaulichen, werden die wesentlichen Systembestandteile durch schematische Skizzen abgebildet.

5.1.6.1 Systemarchitektur

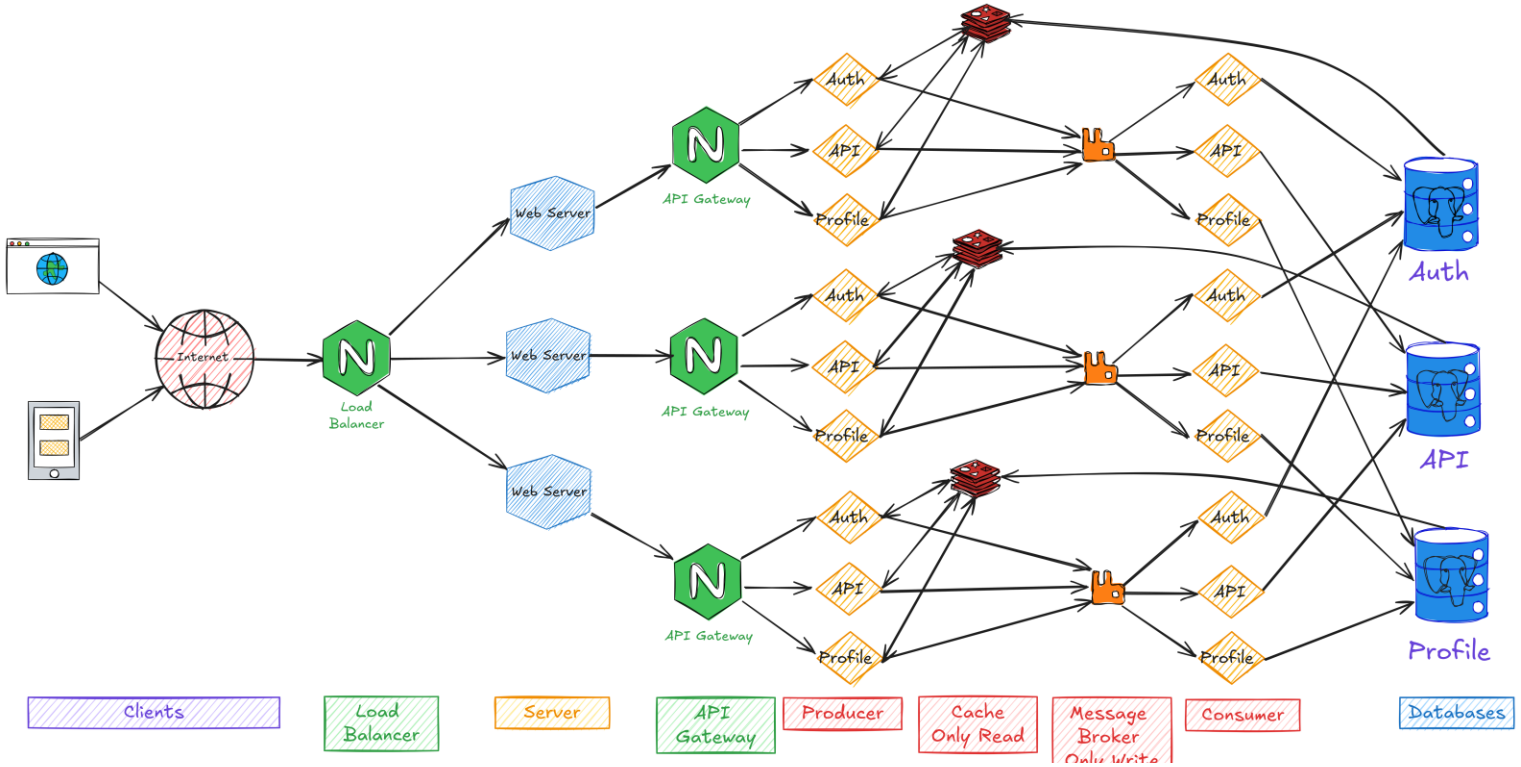


Diagramm 1 – Systemarchitektur

5.1.6.2 Datenbankdesign

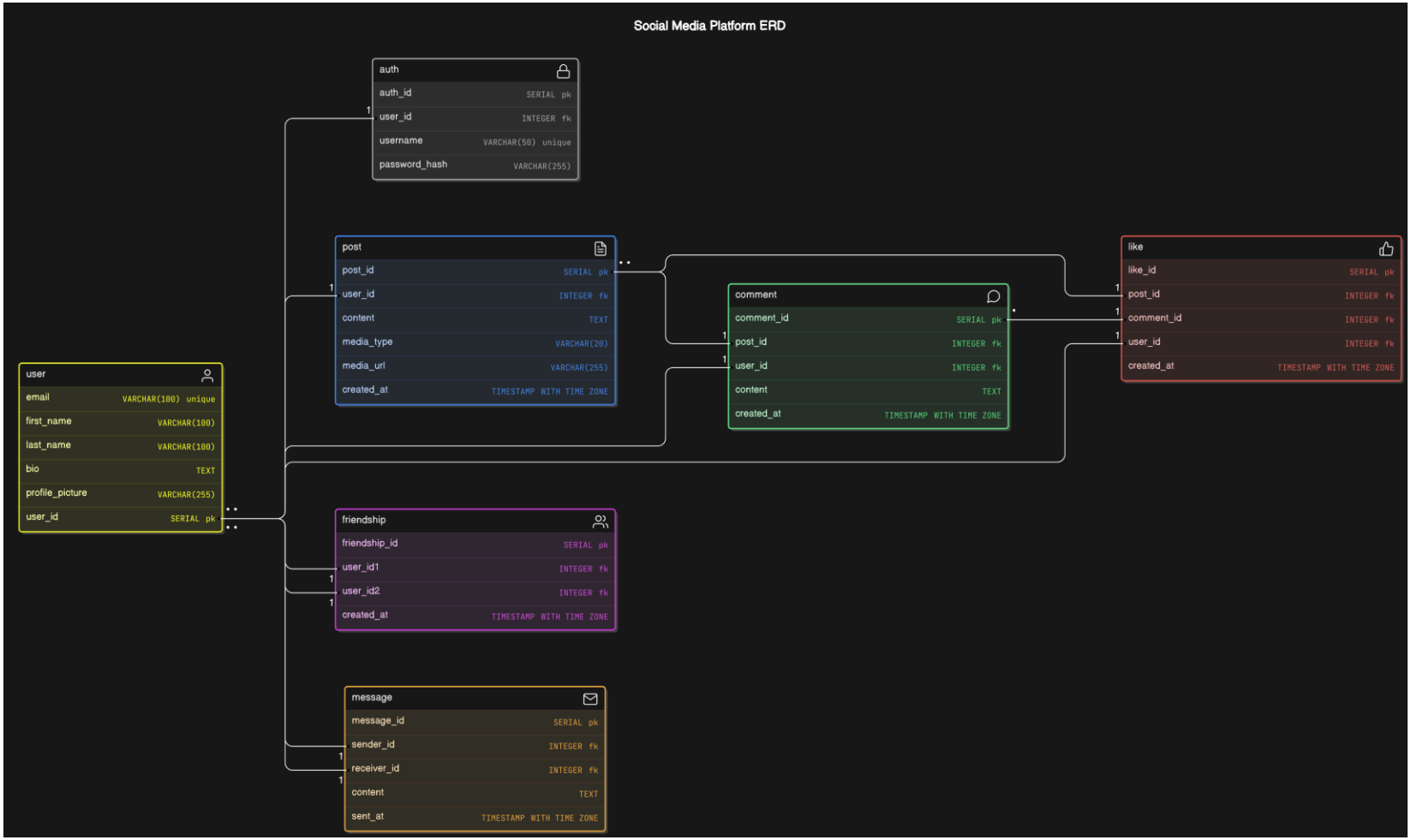


Diagramm 2 – Datenbankdesign

5.1.6.3 CI/CD Pipeline

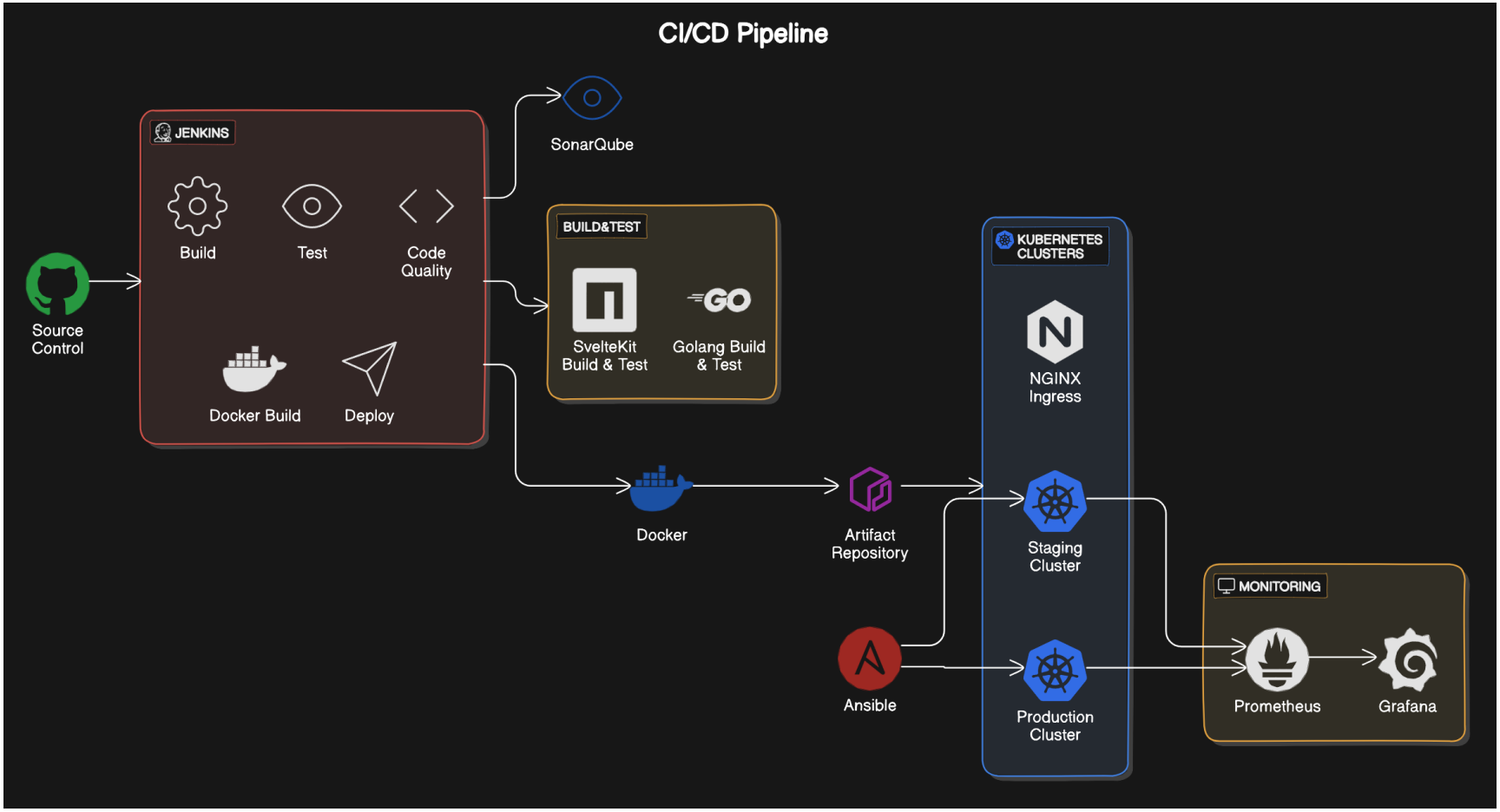


Diagramm 3 – CI/CD-Pipeline des Designs

5.1.7 Meilensteine und Zeitplan

Um einen Überblick dieses Projekts zu erhalten, werden die Meilensteine zeitlich geordnet aufgeführt.

Gesamtprojektdauer:	8 Wochen
Start der Diplomarbeit:	Dienstag, 10. September 2024
Abgabe der Diplomarbeit:	Dienstag, 05. November 2024
Präsentationen:	15. November 2024

Woche 1 (10. - 15. September 2024): Projektinitialisierung und Grundlagen

- Detaillierte Projektplanung und Aufgabenaufschlüsselung
- Grundlegende Projektmanagement-Funktionalitäten
- Einrichtung der Entwicklungsumgebung
- Erstellung der grundlegenden Projektstruktur für Backend und Frontend
- Entwurf der Systemarchitektur

Meilenstein: Projektgrundlagen stehen, Architektur ist definiert

Woche 2-3 (16. - 29. September 2024): Backend-Entwicklung Kernfunktionen

- Implementierung der Basis-Microservices-Architektur
- Entwicklung der Benutzerauthentifizierung und -autorisierung
- Grundlegende Datenbankstruktur und API-Endpunkte
- Integration von RabbitMQ
- Einrichtung von Redis für Caching

Meilenstein: Funktionierendes Backend mit Kernfunktionalitäten

Woche 4-5 (30. September - 13. Oktober 2024): Frontend-Entwicklung Basiskomponenten

- Erstellung des UI-Grunddesigns
- Implementierung der Hauptkomponenten (Benutzerprofile, Dashboards)
- Integration der Authentifizierung im Frontend

Meilenstein: Interaktives Frontend mit Basiskomponenten

Woche 6 (14. - 20. Oktober 2024): Integration und Basis-Features

- Zusammenführung von Backend und Frontend
- Implementierung des Echtzeit-Kommunikationssystems
- Einführung von Docker

Meilenstein: Integration von Backend und Frontend mit funktionsfähigen Basis-Features

Woche 7 (21. - 27. Oktober 2024): DevOps und Erweiterungen

- Einrichtung der CI/CD-Pipeline
- Basis-Konfiguration des Kubernetes-Clusters
- Integration von Nginx als Load Balancer
- Implementierung von Code-Sharing und Versionskontrolle

Meilenstein: Integrierte Anwendung mit CI/CD-Pipeline und Basis-DevOps-Funktionen

Woche 8 (28. Oktober - 05. November 2024): Feinschliff, Tests und Dokumentation

- Durchführung von Integrations- und Systemtests
- Performance-Optimierungen
- Erstellung der Projektdokumentation
- Vorbereitung der finalen Präsentation für den 15. November 2024

Meilenstein: Vollständig getestete und dokumentierte Anwendung, bereit zur Abgabe

5.1.8 Entwicklungsmethodik

Für die Umsetzung der Social-Media-Plattform wurde eine agile Entwicklungsmethodik gewählt, um Flexibilität und schnelle Anpassungen an Projektanforderungen zu ermöglichen. Die Methode zielt darauf ab, durch kontinuierliche Iterationen und klare Strukturierung der Arbeitsprozesse eine effiziente Entwicklung und hohe Qualität des Endprodukts sicherzustellen.

5.1.8.1 Arbeitsorganisation

- Nutzung eines digitalen Kanban-Boards (z.B. Trello, GitHub Projects)
- Visualisierung des Arbeitsflusses mit Spalten: Backlog, To Do, In Progress, Testing, Done
- Begrenzung des Work in Progress (WIP) auf maximal 2 Aufgaben gleichzeitig

5.1.8.2 Aufgabenmanagement

- Aufteilung der Arbeit in kleine, überschaubare Einheiten
- Priorisierung der Aufgaben im Backlog
- Regelmässige Verfeinerung des Backlogs (wöchentlich)

5.1.8.3 Entwicklungspraktiken

- Test-Driven Development (TDD) für Backend-Komponenten
- Event-Driven Development (EDD) für Backend-Event
- Domain-Driven Design (DDD) für Backend-Geschäftslogik
- Continuous Integration und Continuous Deployment (CI/CD)
- Regelmässige Code-Reviews durch Selbstreflexion und statische Codeanalyse-Tools
- Refactoring

5.1.8.4 Fortschrittsverfolgung und Verbesserung

- Tägliche kurze Selbst-Standups zur Überprüfung des Fortschritts
- Wöchentliche Retrospektive zur Prozessverbesserung
- Führen eines Entwicklungstagebuchs für Reflexion und Dokumentation

5.1.9 Qualitätssicherung

Um die Zuverlässigkeit und Funktionalität der Plattform sicherzustellen, wurde ein umfassender Ansatz zur Qualitätssicherung implementiert. Dieser umfasst automatisierte Tests, regelmässige manuelle Überprüfungen und den Einsatz von Code-Analyse-Tools, um eine gleichbleibend hohe Qualität in allen Entwicklungsphasen zu gewährleisten.

5.1.9.1 Implementierung von automatisierten Tests

Unit-Tests, Integrationstests, End-to-End-Tests

5.1.9.2 Regelmässige manuelle Tests und Überprüfungen

Überprüfung der Benutzeroberfläche

5.1.9.3 Verwendung von Codequalitäts-Tools

SonarQube, GoLint für Go, typescript-eslint für TypeScript

5.1.9.4 Sicherheitsüberprüfungen

Penetrationstests und regelmässige Sicherheitsaudits

5.1.9.5 Performance-Tests

Durchführung von Lasttests und Optimierungen

5.1.9.6 Risiken und Herausforderungen

- **Technische Komplexität:**
  - Die Integration verschiedener Technologien könnte herausfordernd sein.
    - **Lösung:**
      - Gründliche Planung und inkrementelle Entwicklung mit regelmässigen Integrationstests.
- **Zeitmanagement:**
      - Als Einzelentwickler könnte es schwierig sein, alle Aspekte des Projekts rechtzeitig zu bewältigen.
    - **Lösung:**
      - Strikte Priorisierung und Fokus auf MVP-Funktionalitäten (Minimum Viable Product).

- **Performance und Skalierbarkeit:**
    - Sicherstellen, dass die Plattform auch bei wachsender Nutzerzahl performant bleibt.
    - **Lösung:**
        - Frühzeitige Performance-Tests und Optimierungen, Nutzung von Caching-Strategien.
- **Sicherheitsrisiken:**
    - Als soziale Plattform muss besonderer Wert auf Datenschutz und Sicherheit gelegt werden.
    - **Lösung:**
        - Implementierung von Best Practices für Sicherheit, regelmässige Sicherheitsaudits.

5.1.10 Abnahmekriterien

Die Abnahmekriterien definieren die erforderlichen Bedingungen, die erfüllt sein müssen, damit die Plattform als erfolgreich abgeschlossen gilt. Sie umfassen funktionale und technische Anforderungen, die sicherstellen, dass das Projektziel erreicht und alle Qualitätsstandards eingehalten werden.

- Alle in «Tabelle 1 – Ziele» definierten Ziele sind erfüllt und funktionsfähig.
- Die Plattform ist in einer Kubernetes-Umgebung deployed und skalierbar.
- Alle automatisierten Tests (Unit, Integration, E2E) sind erfolgreich.
- Die Plattform erfüllt definierte Performance-Benchmarks.
- Eine vollständige Dokumentation des Projekts einschliesslich Architektur und Codebasis liegt vor.
- Ein erfolgreicher Penetrationstest wurde durchgeführt und alle kritischen Sicherheitslücken wurden geschlossen.
- Die Benutzeroberfläche ist responsiv und funktioniert auf gängigen Browsern und Geräten.

5.2 ZIELSCHEIBE

Richtziel

Das Ziel ist die Entwicklung einer innovativen, skalierbaren Social-Media-Plattform, die als zentraler Knotenpunkt für DevOps-Experten und Softwareentwickler fungiert. Diese Plattform soll als Katalysator für effiziente Zusammenarbeit und fortschrittlichen Wissensaustausch dienen, um die kollektive Expertise der Community zu fördern und zu nutzen.	Die Plattform richtet sich primär an: 1. **DevOps-Ingenieure** – Austausch über Best Practices und Erfahrungen. 2. **Softwareentwickler aller Stufen** – Zugang zu praxisnahem Fachwissen. 3. **IT-Unternehmen und Start-ups** – Plattform zur Talentsuche und Technologiepräsentation.
Wozu soll das Endergebnis dienen?	Für wen tun wir das?

Welches Ergebnis soll erreicht werden?	Wie wird das Ergebnis gemessen?	
1. Produktionsreife Plattform: • Erfolgreiches Deployment in einer stabilen Produktionsumgebung • Intuitive, benutzerfreundliche Schnittstellen für optimale User Experience • Vollständige Integration aller geplanten Kernfunktionen **2. Technologische Exzellenz:** • Effiziente Implementierung und Nutzung modernster Technologien (Golang, SvelteKit, Docker, Kubernetes, CI/CD, RabbitMQ) • Demonstration von Best Practices in Softwarearchitektur und DevOps **3. Gemeinschaftsbildung:** • Schaffung einer aktiven, engagierten Community von DevOps-Experten und Entwicklern • Etablierung eines Ökosystems für Wissensaustausch und berufliche Weiterentwicklung	**1. Funktionale Vollständigkeit** • Alle Kernmodule implementiert • 80% Testabdeckung • Dokumentierte Codebasis **2. Technische Umsetzung** • Microservices-Architektur • CI/CD-Pipeline eingerichtet • Containerisierung realisiert **3. Sicherheit** • HTTPS implementiert • Authentifizierung umgesetzt • Grundlegende Sicherheitstests	**4. Benutzerfreundlichkeit** • Responsive Oberfläche • Intuitive Navigation • Grundlegende Usability-Tests **5. DevOps-Praktiken** • Automatisierte Tests • Monitoring eingerichtet • Dokumentierte Deployment-Prozesse

6 PROJEKTPLANUNG

Die Projektplanung umfasst alle wesentlichen Schritte zur Strukturierung, Zeitplanung und Risikobewertung, um eine reibungslose und effiziente Umsetzung des Projekts sicherzustellen. Durch eine sorgfältige Planung werden alle Aktivitäten klar definiert und priorisiert, sodass die Projektziele termingerecht und in hoher Qualität erreicht werden können.

6.1 VORGEHENSMODELL

Das Vorgehensmodell beschreibt die strukturierten Schritte und Methoden, die während der Projektdurchführung eingesetzt wurden, um eine systematische und zielgerichtete Entwicklung der Plattform zu gewährleisten. Die Darstellung des Wasserfall-Modells erfolgt unter Verwendung von Excalidraw mit Mermaid, um eine visuelle Nachvollziehbarkeit der einzelnen Phasen zu ermöglichen. Detaillierte Erläuterungen zu den einzelnen Schritten sind in den Unterabschnitten 6.1.1 bis 6.1.11 zu entnehmen.

Diagramm 4 – Vorgehensmodell

6.1.1 Projektinitialisierung und Planung (1 Woche: 09. - 15. September 2024)

- Festlegung der Projektziele und des Umfangs
- Erstellung eines detaillierten Projektstrukturplans
- Erstellung eines detaillierten Projektablaufplans
- Ausarbeitung einer Risikoanalyse und einer Risikomatrix
- Ausarbeitung einer SWOT-Analyse
- Erstellung einer UX / UI Konzept

6.1.2 Architektur und Design (1 Woche: 16.09.2024 - 20.09.2024)

- Entwurf der Microservices-Architektur
- Design des Datenbankschemas
- Konzeption der Benutzeroberfläche
- Entwurf der CI / CD

6.1.3 Bewertungsanalyse (1 Tag: 21.09.2024)

- Präferenzmatrix erstellt
- Nutzwertanalyse durchgeführt
- Sensitivitätsanalyse abgeschlossen

6.1.4 Implementierung Backend (2 Wochen: 23.09.2024 - 06.10.2024)

- Entwicklung der Golang
- Implementierung der PostgreSQL
- Einrichtung der Docker für Backend-Services und RabbitMQ
- Implementierung der API-Endpunkte
- Implementierung der Redis (Optimal)

6.1.5 Implementierung Frontend (1 Woche: 07.10.2024 - 13.10.2024)

- Entwicklung der Benutzeroberfläche mit Svelte
- Integration der Frontend-Komponenten mit dem Backend
- Implementierung der Benutzerauthentifizierung

- Einrichtung der Docker

6.1.6 DevOps und Infrastruktur (1 Woche: 14.10.2024 - 20.10.2024)

- Konfiguration und Optimierung der CI/CD-Pipeline
- Aufbau des Load Balancing und des Gateways für die Nignx
- Einrichtung des Kubernetes-Clusters
- Implementierung von Monitoring-Lösungen
- Konfiguration von Logging und Tracing

6.1.7 Integration und Systemtests (1 Woche: 21.10.2024 - 27.10.2024)

- Integration aller Komponenten
- Durchführung von End-to-End-Tests
- Leistungsoptimierung und Skalierbarkeitstests
- Sicherheitsüberprüfungen

6.1.8 Dokumentation und Abschluss (1 Woche: 28.10.2024 - 03.11.2024)

- Erstellung der technischen Dokumentation
- Verfassen der Projektdokumentation für die Diplomarbeit
- Abschliessende Überprüfung und Feinabstimmung des Systems

6.1.9 Abgabe der Diplomarbeit: 04.11.2024

6.1.10 Vorbereitung der Präsentation (1,5 Wochen: 05.11.2024 - 14.11.2024)

- Erstellung der Präsentationsunterlagen
- Vorbereitung der Live-Demonstration
- Proben der Präsentation (13.11.2024)

6.1.11 Präsentation der Diplomarbeit: 15.11.2024

6.2 KANBAN PROJEKTPLAN FÜR DIE SOCIAL MEDIA PLATTFORM

Der Kanban-Projektplan dient zur Visualisierung und Steuerung der einzelnen Aufgaben und Arbeitsfortschritte während der Entwicklung der Social Media Plattform. Mithilfe eines digitalen Kanban-Boards werden Aufgaben in verschiedenen Phasen des Workflows abgebildet und kontinuierlich angepasst, um den Projektverlauf transparent und effizient zu gestalten.

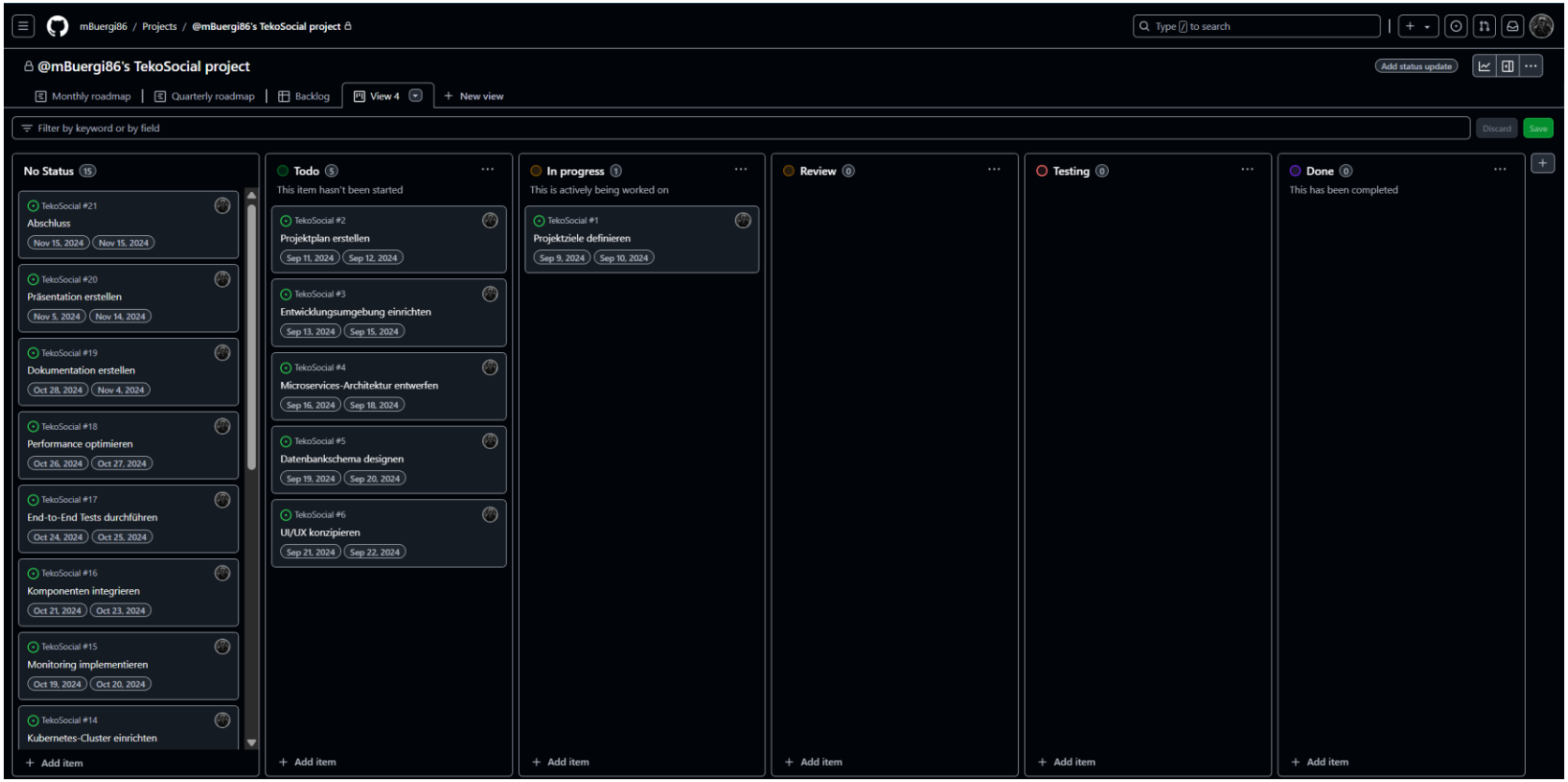


Abbildung 1 – Kanban des GitHub-Projekts

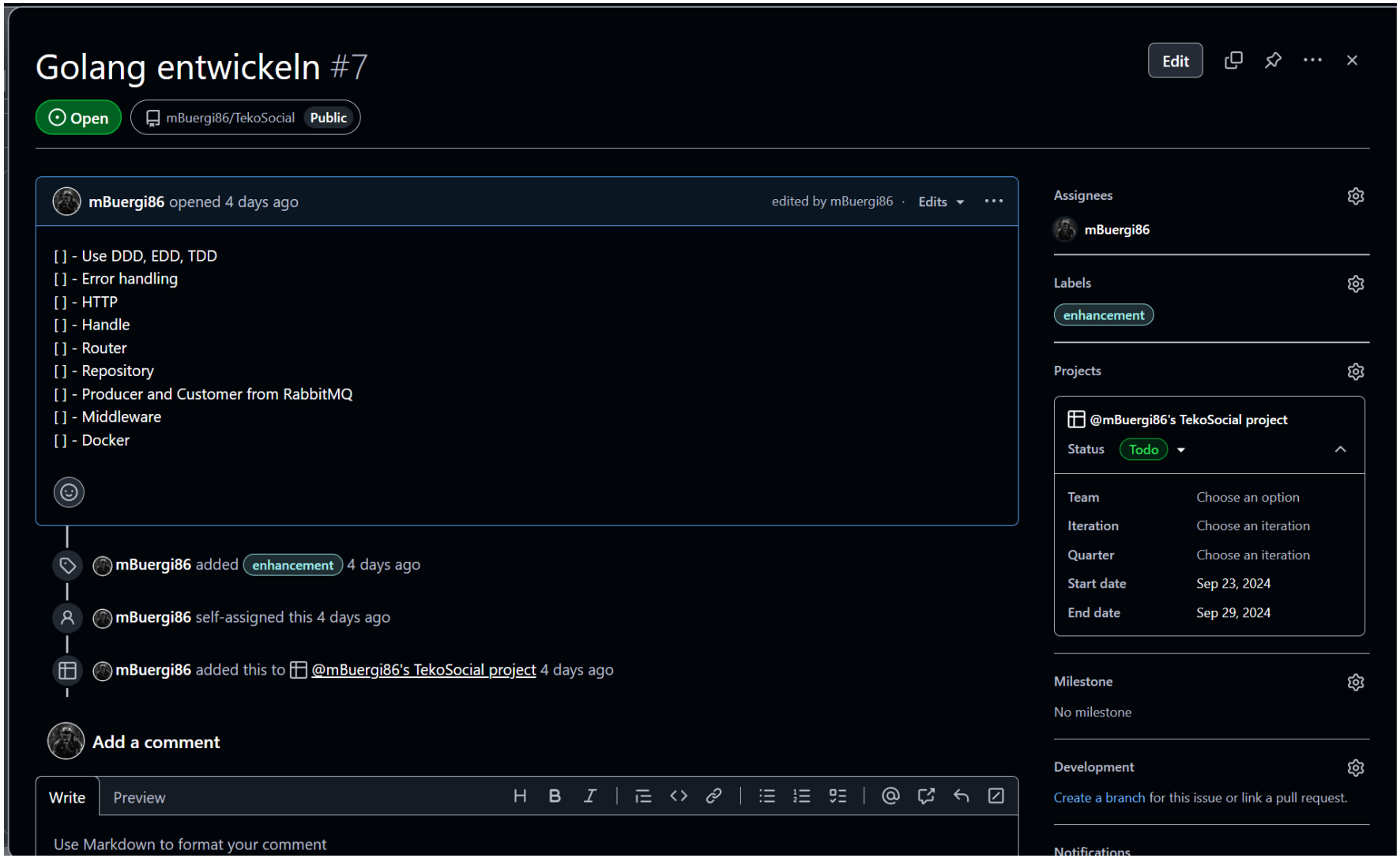


Abbildung 2 – Kanban des GitHub-Projekts

6.3 PROJEKTSTRUKTURPLANUNG

Die Projektstrukturplanung gliedert alle wesentlichen Aufgaben und Teilprojekte hierarchisch, um Verantwortlichkeiten und Abhängigkeiten klar darzustellen.

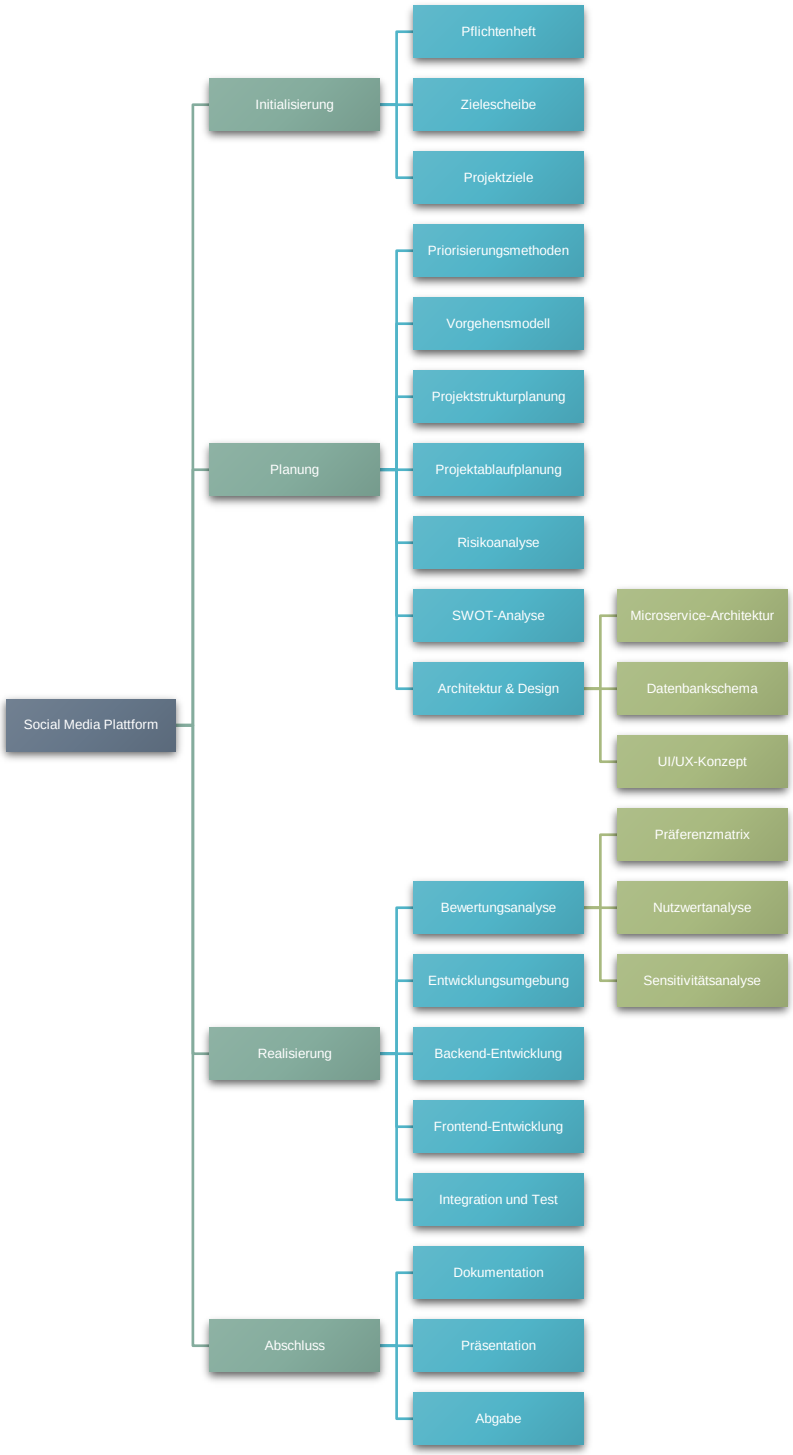


Diagramm 5 – Projektstrukturplanung

6.4 PROJEKTABLAUFPLANUNG

Die Projektablaufplanung legt die zeitliche Reihenfolge aller Projektaktivitäten fest, um einen strukturierten und reibungslosen Projektverlauf zu gewährleisten.

Projekt: Social Media für DevOps und Software-Engineer

Monat	9			10					11	
Kalenderwoche	37	38	39	40	41	42	43	44	45	46
Initialisierung										
Pflichtenheft erstellen	■									
Zielscheibe definieren	■									
Projektziele definieren	■									
Planung										
Vorgehensmodell erstellen	■									
Projektstrukturplanung erstellen	■									
Projektablaufplanung erstellen	■									
Risikoanalyse erstellen	■	■								
SWOT-Analyse erstellen		■								
Microservices-Architektur entwerfen		■								
Datenbankschema designen		■								
UI/UX konzipieren		■								
Realisierung										
Bewertungsanalyse		■					■			
Backend-Entwicklung			■	■			■			
Frontend-Entwicklung				■	■					
Entwicklungsumgebung einrichten						■				
Integration und Test						■	■			
Abschluss										
Auswertung Zielerreichung							■			
Fazit								■		
Präsentation										■
Abgabe									■	

Tabelle 2 – Projektablaufplanung

6.5 RISIKOANALYSE

Die Risikoanalyse identifiziert potenzielle Projektrisiken und bewertet deren Einfluss, um frühzeitig geeignete Gegenmassnahmen planen zu können.

Risikoanalyse								Social Media für DevOps und Software-Engineer 10. September 2024
Nr.	Risikobeschrieb	Ursache	Eintretens-wahrscheinlichkeit	Schadensausmass	Effekt	Massnahmen	Priorität	Bereich
1	Zeitverzögerungen bei der Implementierung	Komplexität der Technologien, ausreichend erfahrene Entwickler	Hoch	Hoch	Verspätete Abgabe der Diplomarbeit	Detaillierte Zeitplanung - Priorisierung der Kernfunktionen - Regelmässige Fortschrittskontrollen	Hoch	Projektmanagement
2	Integrationsprobleme zwischen Technologien	Inkompatibilitäten, fehlende Erfahrung mit Stack	Mittel	Hoch	Funktionale Einschränkungen, erhöhter Zeitaufwand	Frühzeitige Proof-of-Concept-Tests - Nutzung von Integrationsframeworks - Konsultation von Experten/Dozenten	Hoch	Entwicklung
3	Sicherheitslücken in der Anwendung	Unzureichende Sicherheitsmassnahmen, Übersehen von Schwachstellen	Mittel	hoch	Datenverlust, Reputationsschaden	Implementierung von Best Practices für Sicherheit - Regelmässige Sicherheitsaudits - Penetrationstests	Hoch	Sicherheit
4	Performanceprobleme	Ineffiziente Datenbankabfragen, unoptimierter Code	Mittel	Hoch	Langsame Anwendung, schlechte Benutzererfahrung	Frühzeitige Performancetests - Optimierung von Datenbankabfragen - Implementierung von Caching	Mittel	Backend
5	Schwierigkeiten mit Kubernetes-Konfiguration	Komplexität von Kubernetes, mangelnde Erfahrung	Hoch	Mittel	Probleme beim Deployment, ineffiziente Ressourcennutzung	Intensives Kubernetes-Training - Nutzung von Kubernetes-Management-Tools - Vereinfachte Konfigurationen für den Anfang	Mittel	DevOps
6	Unzureichende Dokumentation	Zeitdruck, Unterschätzung der Wichtigkeit	Mittel	Mittel	Erschwerter Support, Probleme bei der Weiterentwicklung	Kontinuierliche Dokumentation als Teil des Entwicklungsprozesses - Verwendung von Dokumentationstools - Regelmässige Dokumentationsreviews	Mittel	Dokumentation
7	Datenbankdesign unterstützt nicht alle Anforderungen	Unvollständige Anforderungsanalyse, mangelnde Voraussicht	Niedrig	Hoch	Einschränkungen in der Funktionalität, aufwändige Umstrukturierungen	Gründliche Anforderungsanalyse - Flexibles Datenbankschema - Regelmässige Überprüfung des Datenbankdesigns	Mittel	Backend
8	Probleme bei der Implementierung von DevOps-Praktiken	Komplexität von DevOps, Zeitmangel	Mittel	Mittel	Ineffiziente Entwicklungs- und Deploymentprozesse	Klare Definition von DevOps-Zielen - Schrittweise Einführung von DevOps-Praktiken - Fokus auf automatisierte Tests und Deployments	Mittel	DevOps
9	Mangelnde Benutzerakzeptanz	Komplexe Benutzeroberfläche, fehlende Usability-Tests	Mittel	Hoch	Geringe Nutzung der Plattform, negative Bewertung	Frühzeitige und regelmässige Usability-Tests - Iteratives UI/UX-Design - Fokus auf Intuitivität und Benutzerfreundlichkeit	Hoch	UI/UX
10	Überschreitung des Projektumfangs	Feature Creep, unklare Abgrenzung	Hoch	Mittel	Zeitverzögerungen, unvollständige Implementierung	Klare Definition und Dokumentation des Projektumfangs - Strenge Änderungskontrolle - Regelmässige Überprüfung der Projektziele	Hoch	Projektmanagement
11	Probleme mit der Skalierbarkeit der Anwendung	Unzureichende Planung für Wachstum, ineffiziente Architektur	Mittel	Hoch	Leistungseinbrüche bei steigender Nutzerzahl	Entwurf einer skalierbaren Architektur - Implementierung von Lastverteilung - Regelmässige Lasttests	Hoch	Architektur
12	Schwierigkeiten bei der Implementierung von Echtzeit-Funktionen	Komplexität von Websockets, mangelnde Erfahrung	Mittel	Mittel	Eingeschränkte Interaktivität der Plattform	Gründliche Recherche zu Echtzeit-Technologien - Prototyping von Echtzeit-Features - Nutzung bewährter Bibliotheken	Mittel	Entwicklung
13	Kompatibilitätsprobleme mit verschiedenen Browsern	Unterschiedliche Browser-Standards, fehlende Tests	Niedrig	Mittel	Inkonsistente Benutzererfahrung	Cross-Browser-Testplan - Verwendung von Polyfills - Progressive Enhancement Strategie	Niedrig	Frontend
14	Datenschutzprobleme	Unzureichende Berücksichtigung von DSGVO-Anforderungen	Mittel	Hoch	Rechtliche Konsequenzen, Vertrauensverlust	Frühzeitige DSGVO-Konformitätsprüfung - Implementierung von Privacy by Design - Regelmässige Datenschutz-Audits	Hoch	Sicherheit
15	Schwierigkeiten bei der Implementierung von CI/CD	Komplexität der Werkzeuge, fehlende Erfahrung	Hoch	Mittel	Verzögerungen im Entwicklungsprozess, manuelle Fehler	Schrittweise Einführung von CI/CD-Praktiken - Schulung in CI/CD-Tools - Automatisierung von Build- und Testprozessen	Hoch	DevOps
16	Probleme mit der Datenintegrität	Fehlerhafte Transaktionslogik, Race Conditions	Niedrig	Hoch	Inkonsistente oder korrupte Daten	Implementierung robuster Transaktionsmechanismen - Gründliche Tests von Datenoperationen - Regelmässige Datenbank-Konsistenzprüfungen	Mittel	Backend
17	Schwierigkeiten bei der Implementierung von Microservices	Komplexität der Architektur, Overhead in der Kommunikation	Hoch	Mittel	Erhöhte Entwicklungszeit, Performanceeinbussen	Gründliche Planung der Microservices-Architektur - Implementierung effizienter Service-Discovery - Nutzung von Microservices-Frameworks	Hoch	Architektur
18	Unzureichende Testabdeckung	Zeitdruck, Unterschätzung der Wichtigkeit von Tests	Hoch	Hoch	Erhöhte Fehleranfälligkeit, schwierige Wartbarkeit	Implementierung einer Test-First-Strategie - Automatisierung von Tests - Regelmässige Code-Coverage-Analysen	Hoch	Test
19	Probleme mit der Versionierung von APIs	Inkompatible Änderungen, fehlende Strategie	Mittel	Mittel	Brüche in der Clientkompatibilität	Implementierung einer klaren API-Versionierungsstrategie - Nutzung von API-Gateways - Gründliche Dokumentation von API-Änderungen	Mittel	Backend
20	Schwierigkeiten bei der Integration	API-Änderungen, Ausfälle externer Dienste	Mittel	Mittel	Funktionsausfälle, Abhängigkeiten von externen Faktoren	Implementierung von Fallback-Mechanismen - Regelmässige Überprüfung von API-Dokumentationen - Entwicklung von Mock-Services für Tests	Mittel	Entwicklung
21	Geringfügige UI-Inkonsistenzen	Unterschiedliche Entwickler, fehlende Design-Guidelines	Hoch	Niedrig	Leicht inkonsistentes Erscheinungsbild	Erstellung und Durchsetzung von UI-Richtlinien - Regelmässige UI-Reviews - Verwendung von UI-Komponenten-Bibliotheken	Niedrig	UI/UX
22	Verzögerungen bei der Bereitstellung von Entwicklungsressourcen	Administrative Verzögerungen, begrenzte Ressourcen	Mittel	Niedrig	Leichte Verzögerungen im Entwicklungsprozess	Frühzeitige Ressourcenanforderung - Nutzung von Cloud-Entwicklungsumgebungen - Vorbereitung von Alternativplänen	Niedrig	DevOps
23	Kleinere Fehler in der Dokumentation	Zeitdruck, unzureichende Prüfung	Hoch	Niedrig	Leichte Verwirrung bei der Nutzung der Dokumentation	Implementierung eines Peer-Review-Prozesses für Dokumentation - Regelmässige Dokumentations-Audits - Nutzung von Dokumentations-Tools mit Rechtschreibprüfung	Niedrig	Dokumentation
24	Geringe Beteiligung an internen Code-Reviews	Zeitdruck, fehlende Kultur des Peer-Reviews	Mittel	Mittel	Möglicherweise übersehene kleinere Codeprobleme	Einführung von obligatorischen Code-Review-Richtlinien - Schulungen zur Bedeutung von Code-Reviews - Gamification des Review-Prozesses	Mittel	Entwicklung
25	Leichte Überschreitung des Speicherplatzbedarfs	Ungenauigkeit in der Ressourcenplanung	Mittel	Niedrig	Geringe zusätzliche Kosten für Speicherressourcen	Regelmässige Überwachung der Ressourcennutzung - Implementierung von Datenbereinigungsroutinen - Planung von Speichererweiterungen	Niedrig	DevOps
26	Geringfügige Abweichungen in der Farbpalette der UI	Unterschiedliche Bildschirmkalibrierungen, leichte Variationen in CSS-Definitionen	Niedrig	Niedrig	Minimal sichtbare Farbunterschiede in der Benutzeroberfläche	Verwendung eines standardisierten Farbmanagement-Systems - Regelmässige visuelle Überprüfungen auf verschiedenen Geräten - Implementierung eines zentralen CSS-Variablen-Systems	Niedrig	UI/UX

Tabelle 3 – Risikoanalyse

6.5.1 Risikomatrix

Die Risikomatrix bewertet potenzielle Projektrisiken nach Eintrittswahrscheinlichkeit und Einfluss, um gezielte Gegenmassnahmen zu planen.

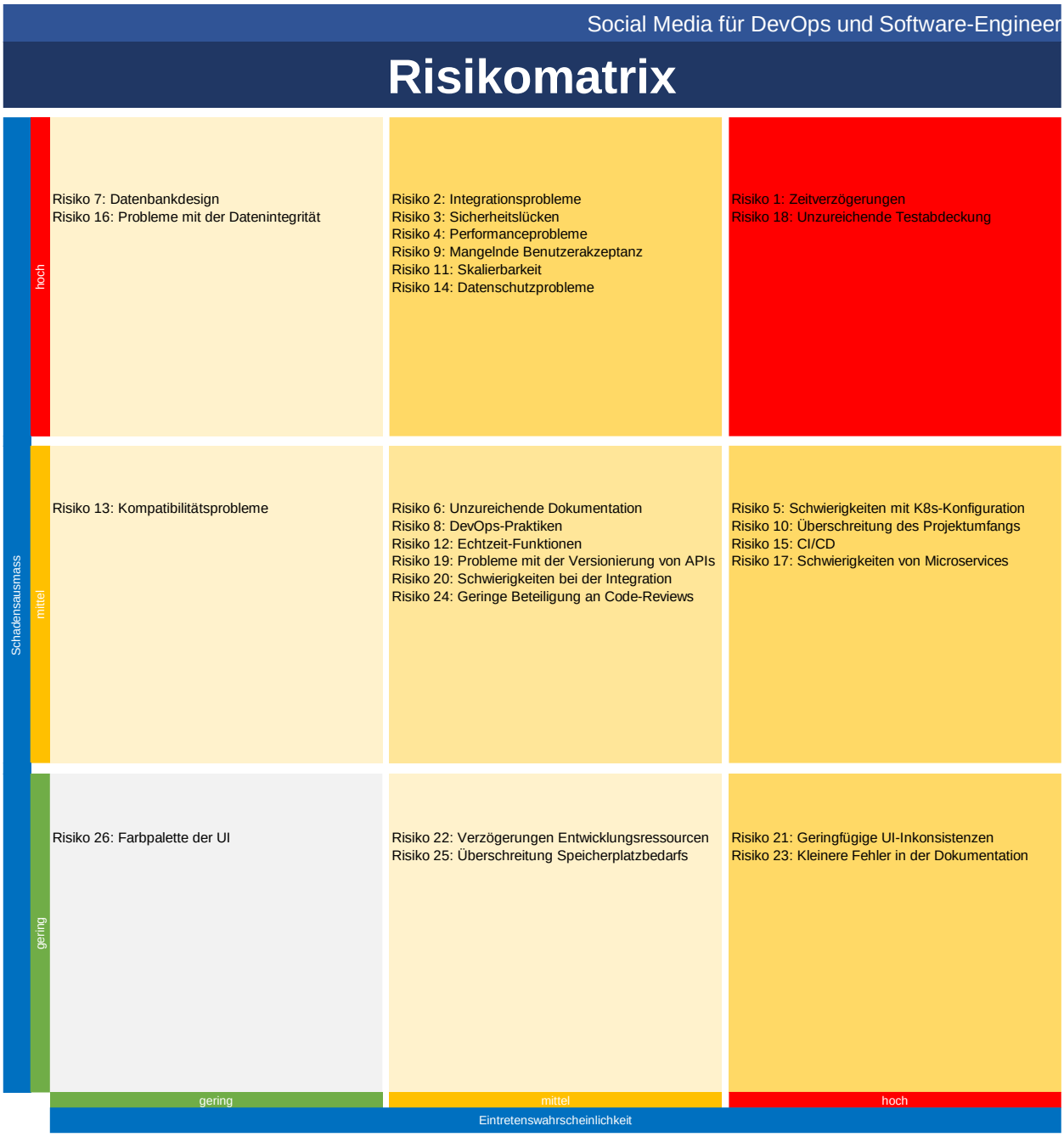


Tabelle 4 – Risikomatrix

6.6 SWOT-Analyse

Die SWOT-Analyse untersucht die Stärken, Schwächen, Chancen und Risiken des Projekts, um strategische Vorteile zu nutzen und potenzielle Herausforderungen zu bewältigen.

SWOT-Analyse Durchgeführt durch: Markus Bürgi Datum: 10. September 2024	**Externe Faktoren**	
	Chancen (Opportunities)	**Risiken (Threats)**
	O1: Wachsender Markt für DevOps und Softwareentwickler-Tools	T1: Starke Konkurrenz durch etablierte Plattformen
	O2: Förderung der Zusammenarbeit zwischen DevOps und Softwareentwicklern	T2: Schnelle technologische Veränderungen in der Branche
	O3: Showcase für moderne Technologien und Praktiken	T3: Sicherheitsrisiken durch komplexe Architektur
	O4: Potenzial für Erweiterungen und Integrationen	T4: Mögliche Verzögerungen im Entwicklungsprozess
	O5:Networking-Plattform für IT-Fachleute	T5: Herausforderungen bei Datenschutz und Compliance
Interne Faktoren	**Stärken (Strengths)**	**SO-Strategien**	**ST-Strategien**
	S1: Spezialisierte Plattform für DevOps und Softwareentwickler	SO1: Positionierung als führende Kollaborationsplattform für DevOps und Softwareentwickler	ST1: Einsatz der Microservices-Architektur für schnelle Anpassungen an Marktveränderungen
	S2: Moderner Technologiestack (Golang, Svelte, PostgreSQL, RabbitMQ, Docker, Kubernetes)	SO2: Showcase der Plattform als Best-Practice für moderne IT-Architektur	ST2: Nutzung des strukturierten Entwicklungsprozesses für robuste Sicherheitsimplementierung
	S3: Microservices-Architektur für Skalierbarkeit	SO3: Nutzung der integrierten Tools zur Förderung von DevOps-Kultur	ST3: Hervorhebung der spezialisierten Features zur Differenzierung von Konkurrenz
	S4: Integrierte DevOps-Tools und Entwicklungsumgebungen	SO4: Entwicklung von Features, die Synergien zwischen DevOps und Softwareentwicklung nutzen	ST4: Implementierung strenger Datenschutzmassnahmen basierend auf der gründlichen Planung
	S5: Strukturierter Entwicklungsprozess		
	Schwächen (Weaknesses)	**WO-Strategien**	**WT-Strategien**
	W1: Begrenzte Flexibilität bei Änderungen	WO1: Nutzung des Marktwachstums zur Gewinnung früher Benutzer für Feedback	WT1: Priorisierung von Sicherheit und Datenschutz in jeder Entwicklungsphase
	W2: Komplexe Integration verschiedener Technologien	WO2: Fokussierung auf Kernfunktionen, die beide Zielgruppen ansprechen	WT2: Entwicklung einer klaren Differenzierungsstrategie gegenüber etablierten Plattformen
	W3: Ressourcenbeschränkungen einer Diplomarbeit	WO3: Entwicklung einer klaren Roadmap für zukünftige Flexibilität und Erweiterungen	WT3: Implementierung von Mechanismen für regelmässiges Nutzerfeedback trotz Wasserfallmodell
	W4: Fehlendes frühes Benutzerfeedback	WO4: Aufbau einer aktiven Community zur Überwindung von Ressourcenbeschränkungen	WT4: Fokus auf essentielle Features zur Minimierung von Verzögerungsrisiken
	W5: Herausforderung, beide Zielgruppen gleichermassen anzusprechen		

Tabelle 5 – SWOT-Analyse

6.7 UX / UI KONZEPT

Bei der Entwicklung des UX/UI-Konzepts für DevSeConnect wurde ein kollaborativer Ansatz gewählt, der meine eigenen Ideen mit KI-gestützten Vorschlägen kombiniert. Als Entwickler ohne umfassende UX/UI-Erfahrung habe ich mich entschieden, die Expertise einer KI zu nutzen, um ein fundiertes und nutzerfreundliches Konzept zu erstellen. Dieser Prozess ermöglichte es mir, von fortschrittlichen Design-Prinzipien zu lernen und gleichzeitig meine eigene Vision für die Plattform umzusetzen.

6.7.1 Analyse

Die Analyse des UX / UI Konzepts erfasst die Anforderungen und Nutzerbedürfnisse, um eine benutzerfreundliche und zielgerichtete Gestaltung der Plattform sicherzustellen.

6.7.1.1 Zielgruppenanalyse und Segmentierung

Es ist darauf hinzuweisen, dass die Entwickelnden keine homogene Gruppe abbilden. Sie zeigen unterschiedliche Interessen, Rollenfunktionen auf und verwenden verschiedene Tools, Sprachen und Hardwares. Die Entwickelnden lassen sich somit in vier Hauptkategorien unterteilen:

- Ingenieure:innen (z.B. Fullstack-Entwicklelnde, Webentwickelnde)
- Architekturschaffende (z.B. Softwarearchitekten:innen, Datenbankadministratorende)
- Data Scientists (z.B. Data Engineers, Data Analysts)
- DevOps (z.B. DevOps-Ingenieure:innen, SysOps-Administratorende)

6.7.1.2 Bevorzugte Social-Media-Plattformen

In diesem Abschnitt werden die Social-Media-Plattformen identifiziert, die von der Zielgruppe bevorzugt genutzt werden. Diese Erkenntnisse sind entscheidend, um die Plattform optimal an die Bedürfnisse und Gewohnheiten der Nutzer anzupassen.

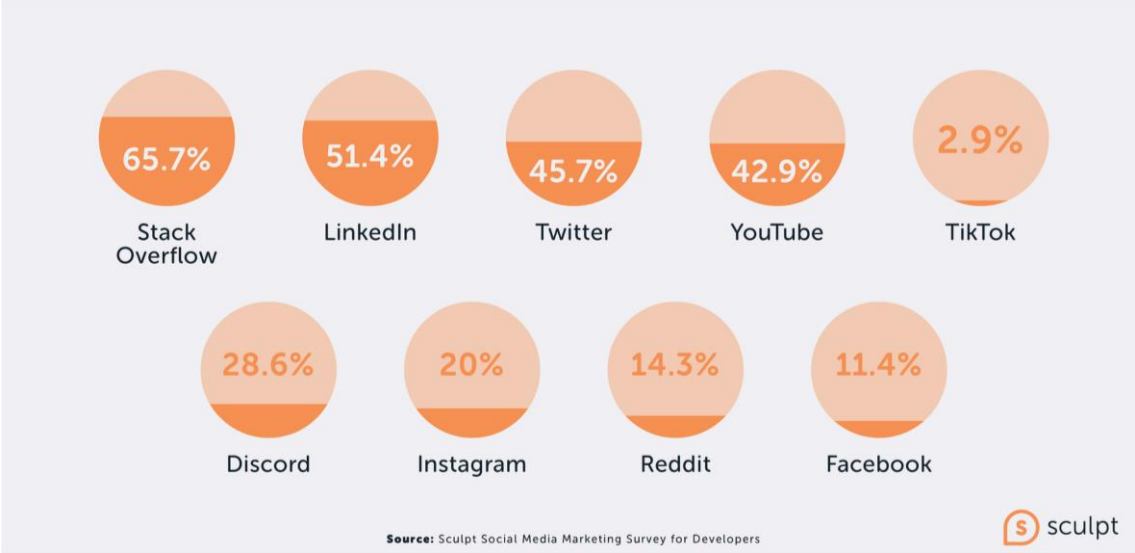


Abbildung 3 – Social Media Marketing (Quelle: wearesculpt.com)

- **Entwicklungsplattformen:**
  - GitHub (93%), ursprüngliche Stack Overflow Survey
  - Stack Overflow (65,7%)
- **Professionelle Netzwerke und Informationsquellen:**
  - LinkedIn (51,4%)
  - Twitter (45,7%)
  - YouTube (42,9%)
- **Community und Diskussionsplattformen:**
  - Discord (28,6%)
  - Reddit (14,3%)
- **Allgemeine Social Media:**
  - Instagram (20%)
  - Facebook (11,4%)
  - TikTok (2,9%)

6.7.1.3 Nutzungsverhalten der Social-Media-Plattformen:

- 28,6% der Entwickelnden verbringen 1-2 Stunden täglich in den sozialen Medien.
- Bis zu 20% der Entwickelnden nutzen die sozialen Medien täglich zwischen 5-7 Stunden.
- Entwickelnde nutzen Twitter, Stack Overflow, Slack, Discord und Reddit, um sich über neue Lösungen zu informieren.
- GitHub verzeichnet ein starkes Nutzungs-Wachstum in Indien (über 9 Millionen Entwickelndeler), China und den USA, somit sind regionale Unterschiede zu erkennen.

6.7.1.4 Wichtige Erkenntnisse:

- GitHub dominiert als Plattform für eine Projektzusammenarbeit und das Code-Sharing.
- Stack Overflow bleibt eine wichtige Ressource für technische Fragen und Antworten.
- Professionelle Netzwerke wie LinkedIn und Twitter spielen eine grosse Rolle für Karriere und Branchentrends.
- YouTube ist eine beliebte Plattform für Tutorial-Videos und Tech-Talks.
- Spezialisierte Community-Plattformen wie Discord gewinnen an Bedeutung.

Diese Erkenntnisse zeigen ein umfassenderes Bild der Plattformnutzung unter Entwickelnden auf. Sie zeigt, dass neben den traditionellen sozialen Medien auch entwicklungsspezifische Plattformen wie GitHub und Stack Overflow eine zentrale Rolle spielen. Für ein effektives Social Media Marketing ist es daher wichtig, diese technischen Plattformen in die Strategie einzubeziehen und plattformspezifische Ansätze zu entwickeln.

6.7.1.5 Kernstrategien für ein erfolgreiches Social Media Marketing

Um eine nachhaltige und wirkungsvolle Social-Media-Präsenz aufzubauen, ist es wichtig, einige Kernstrategien zu beachten. Die folgenden Punkte zeigen, wie man gezielt Zielgruppen anspricht und das Engagement in sozialen Netzwerken fördert.

- **Zielgruppenorientierung:**
  - Fokus auf spezifische Bedürfnisse der verschiedenen Entwickelnden-Gruppen
  - Anpassung der Inhalte an jeweilige Interessen und Herausforderungen
- **Community-Engagement:**
  - Aktive Teilnahme in Online-Communities für spezifische Programmiersprachen/Technologien
  - Nutzung von Plattformen wie Slack (von über 80% der Fortune-100-Unternehmen genutzt)
- **Content-Strategie:**
  - Bereitstellung relevanter, zuordenbarer Inhalte zu Entwickelnden-Herausforderungen
  - Hochwertiger Dokumentationen
  - Mischung aus nützlichen und unterhaltsamen Inhalten, auch produktunabhängig
- **Influencer-Kooperationen:**
  - Zusammenarbeit mit angesehenen Entwickelnden und Tech-Experten:innen
- **Authentizität:**
  - Vermeidung von Spam-Taktiken
  - Förderung echter, wertvoller Interaktionen
- **Technologie-Diskurs:**
  - Aktive Beteiligung an Gesprächen über Innovationen und Trends
- **Multi-Channel-Ansatz:**
  - Nutzung verschiedener Plattformen für breite Zielgruppenansprache
- **Innovative Formate:**
  - Nutzung verschiedener Social-Media-Funktionen (z.B. Videos, Livestreams, Umfragen)

6.7.1.6 Best Practice Beispiel einer Marketingstrategie

Die FreeCodeCamp

Strategie ist eine erfolgreiche Marketingstrategie, die wie folgt umgesetzt wird:

- Dedizierter YouTube-Kanal für Tech-Talks (über 37.000 Follower)
- Kostenlose Coding-Ressourcen
- Aufbau einer starken Community

6.7.1.7 Erfolgsfaktoren der Entwickelnden

Die Entwickelnden brauchen ein tiefes Verständnis der Zielgruppe und die Fähigkeit sich an neue Trends und Technologien anzupassen, um eine ansprechende Form für das Klientel zu publizieren. Zusätzlich sind relevante und hochwertige Inhalte zu präsentieren von grosser Bedeutung.

Eine authentische Interaktion zu zeigen und konstante Präsenz auf relevanten Plattformen vorzunehmen, gehören zudem zu den Erfolgsfaktoren.

6.7.2 Informationsarchitektur

Die Informationsarchitektur beschreibt die Struktur und Organisation der Inhalte der Plattform, um eine intuitive Navigation und effiziente Nutzerinteraktion zu gewährleisten.

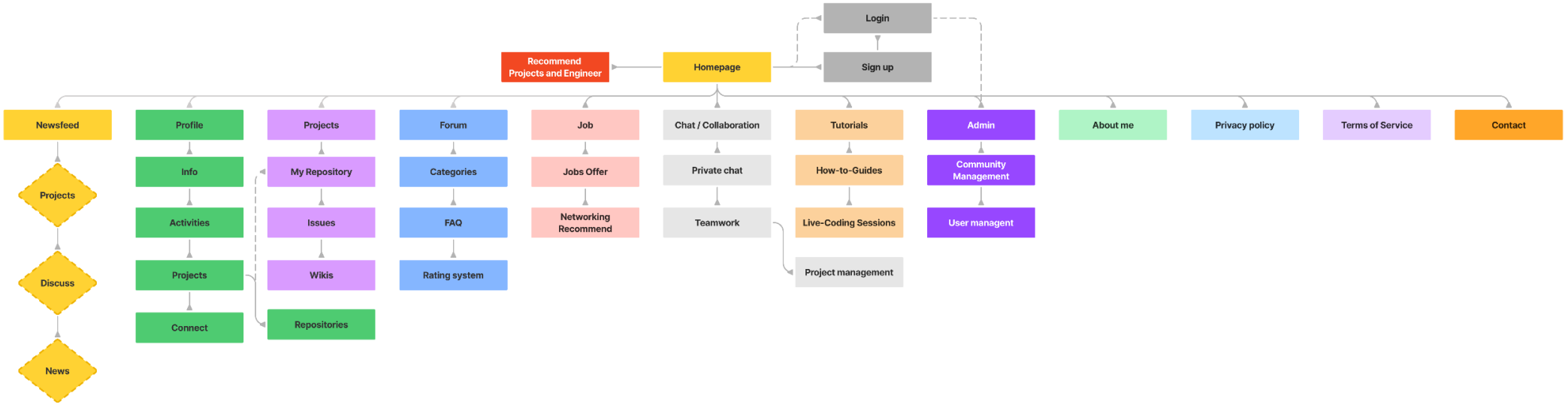


Diagramm 6 - Informationsarchitektur

6.7.3 Visuelle Gestaltung

Die visuelle Gestaltung legt das Design der Benutzeroberfläche fest, um eine ansprechende und funktionale Nutzungserfahrung zu schaffen, die das Nutzungserlebnis optimiert.

Startseite ohne Login:

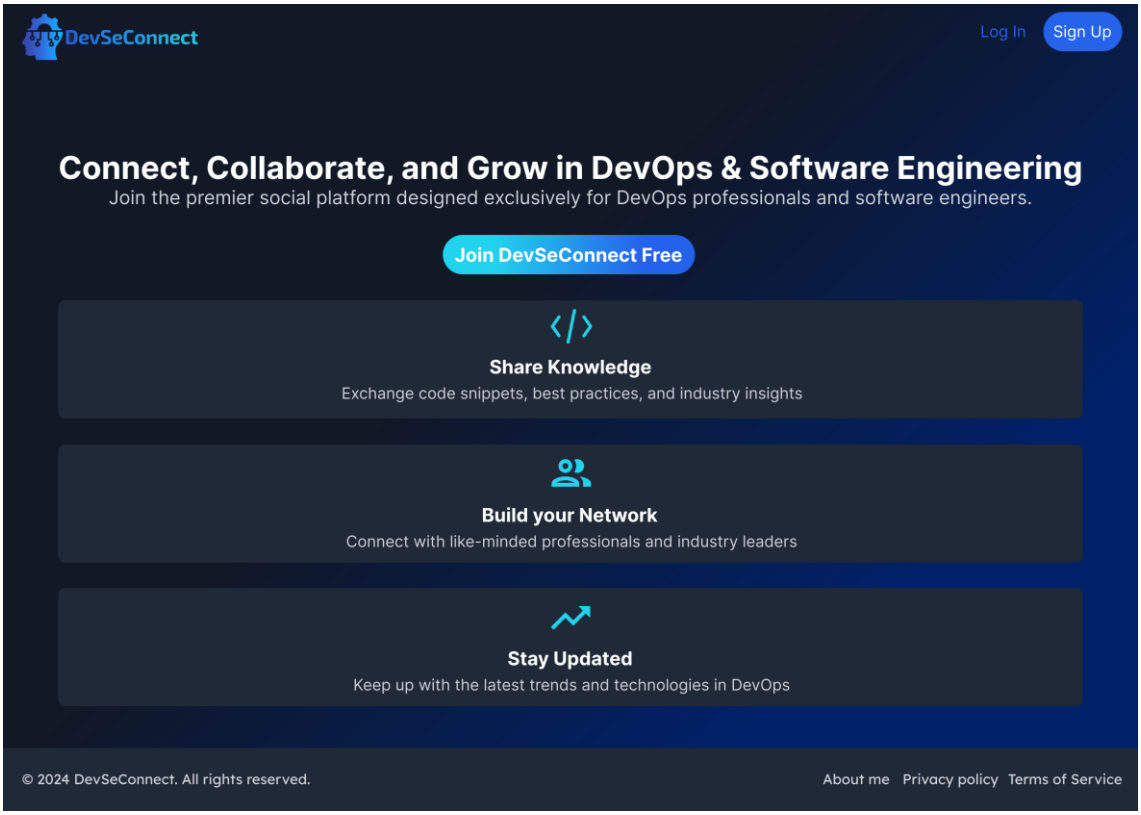


Abbildung 4 – Design der Startseite

Dashboard:

Das Dashboard bietet eine zentrale Übersicht über alle wichtigen Informationen und Funktionen der Plattform, sodass Benutzer schnell auf relevante Inhalte und Tools zugreifen können.

Dark-Mode:

Der Dark-Mode sorgt für eine augenschonende Darstellung der Benutzeroberfläche in dunklen Farbtönen und bietet den Nutzern eine alternative visuelle Option, die besonders in lichtarmen Umgebungen angenehm ist.

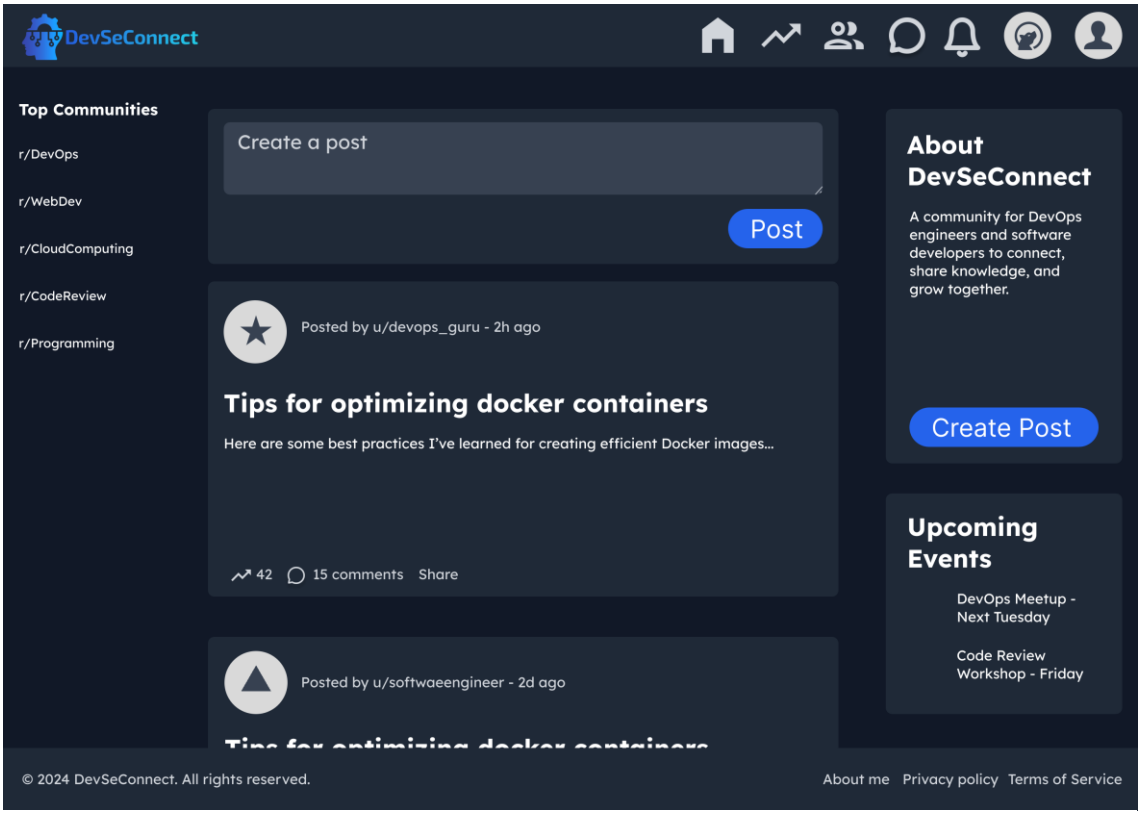


Abbildung 5 – Design des Dashboards (Dark-Mode)

Light-Mode:

Der Light-Mode stellt die Benutzeroberfläche in hellen Farbtönen dar und bietet eine klare und freundliche Ansicht, die besonders in gut beleuchteten Umgebungen angenehm ist.

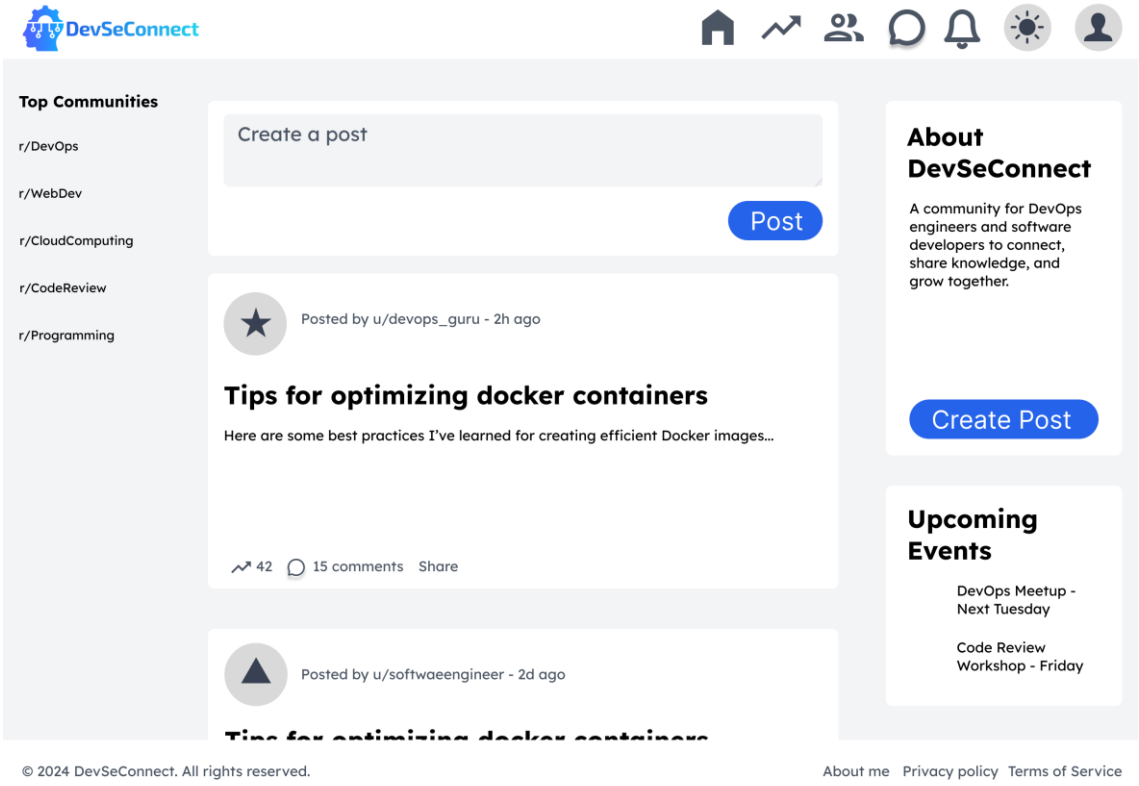


Abbildung 6 – Design des Dashboards (Light-Mode)

6.7.4 Farbkonzept der Benutzeroberfläche

Die folgenden Abbildungen zeigen die Farbcodes für Dark Mode, Light Mode, Akzentfarben und Funktionsfarben, um eine konsistente und ansprechende Gestaltung der Benutzeroberfläche zu gewährleisten.

Dark Mode:

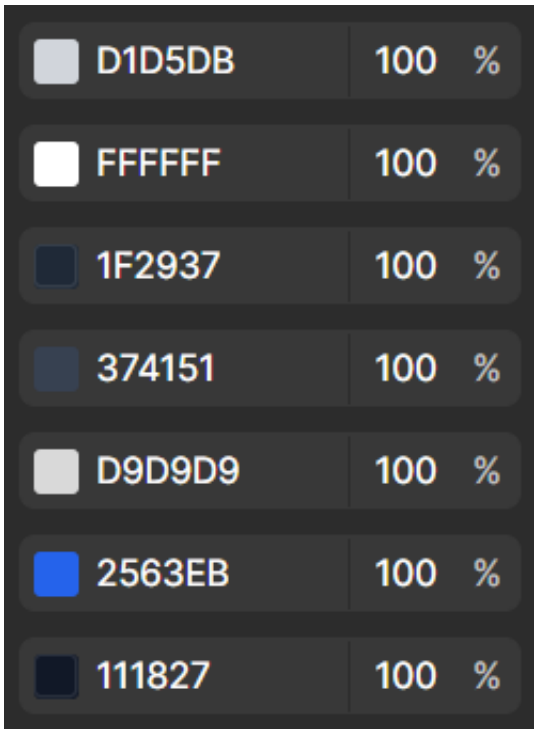


Abbildung 7 – Farbkonzept für den Dark-Mode

Die Farben im Dark Mode sind in dunklen Tönen gehalten und bieten eine augenschonende Darstellung, die besonders in lichtarmen Umgebungen angenehm ist. Die Palette umfasst verschiedene Grautöne für den Hintergrund sowie Blau- und Schwarztöne, die für Kontraste und Hervorhebungen sorgen.

Light Mode:

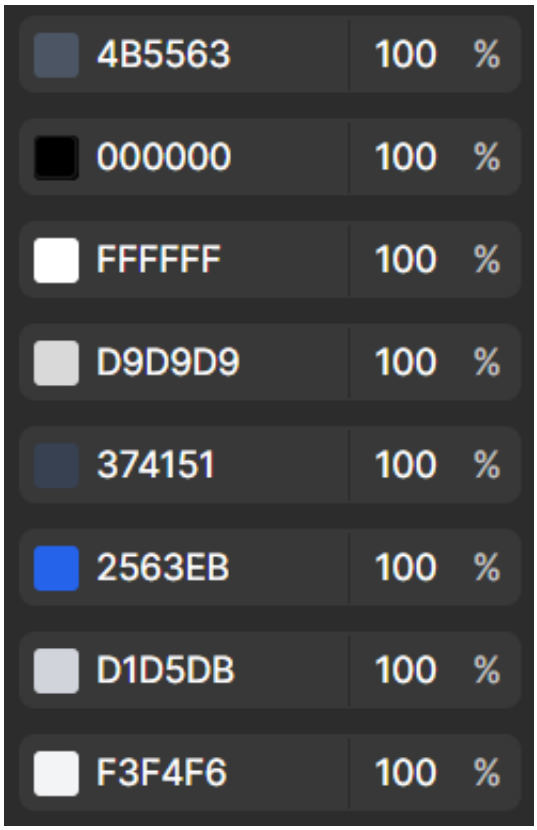


Abbildung 8 – Farbkonzept für den Light-Mode

Die Farben im Light Mode bestehen aus helleren Tönen, die eine klare und freundliche Ansicht ermöglichen. Die Palette enthält helle Grautöne und Weiss für den Hintergrund sowie Blau- und Schwarztöne für Akzente und Texte, was in gut beleuchteten Umgebungen besonders angenehm ist.

Accent Colors:

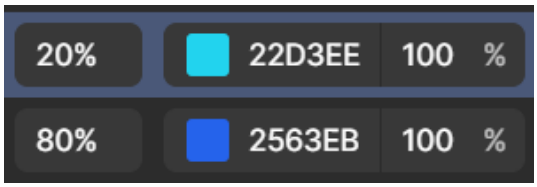


Abbildung 9 – Farbkonzept für den Akzentfarben

Die Akzentfarben setzen visuelle Highlights auf der Benutzeroberfläche. Sie umfassen kräftige Blau- und Türkistöne, die für wichtige Elemente und Interaktionspunkte verwendet werden, um die Aufmerksamkeit der Nutzer zu lenken und eine visuelle Hierarchie zu schaffen.

Functional Colors:

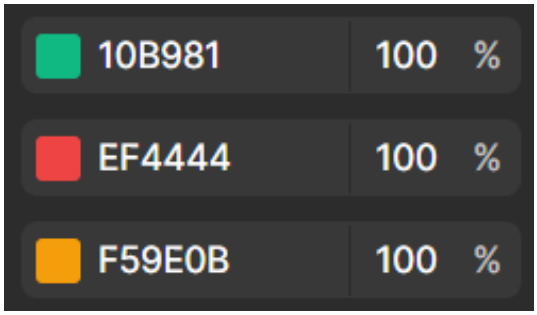


Abbildung 10 – Farbkonzept für den Funktionsfarben

Die Funktionsfarben sind speziell für die Darstellung von Statusmeldungen und Aktionen vorgesehen. Grün zeigt positive Zustände (z.B. erfolgreich), rot signalisiert Warnungen oder Fehler, und gelb wird für Hinweise verwendet. Diese Farben sorgen für eine intuitive und schnelle Erkennung wichtiger Informationen.

7 PROJEKTREALISIERUNG

7.1 KREATIVITÄTSMETHODE

Für die Ideensammlung des Projekts wurde die adaptierte Mindmapping-Methode gewählt, unter anderem da sie sich bei einer Einzelarbeit eignet. Zudem ist sie die passende Methode für dieses Projekt aus folgenden Gründen:

- Die Methode ermöglicht eine visuelle Darstellung von Ideen und deren Zusammenhängen.
- Sie ist flexibel und iterativ, was sich insbesondere bei komplexen Projekten wie DevSeConnect als vorteilhaft erweist.
- Sie fördert das kreative Denken und bietet gleichzeitig eine Strukturierungshilfe.

7.1.1 Umsetzung der adaptierten Mindmapping-Methode

Die konkrete Umsetzung der Ideensammlung sah wie folgt aus:

- Die Platzierung von «DevSeConnect» erfolgte in der Mitte eines grossformatigen Blattes bzw. digitalen Whiteboards.
- Es wurden Hauptäste mit den zentralen Aspekten erstellt und die anknüpfenden Ideen zu diesen Aspekten ergänzt.
- Im Anschluss wurde ein Zeitlimit von 30 Minuten definiert und der Mindmapping Prozess kontinuierlich fortgesetzt
- Zum Schluss der Mindmappings-Erstellung und Reflexion des Mindmappings wurden Verbindungen zwischen verwandten Ideen hinzugefügt, somit die Struktur verfeinert und das Verständnis vertieft.

Anhand der nachfolgenden Abbildung ist das erstellte Mindmap dieses Projekts ersichtlich.

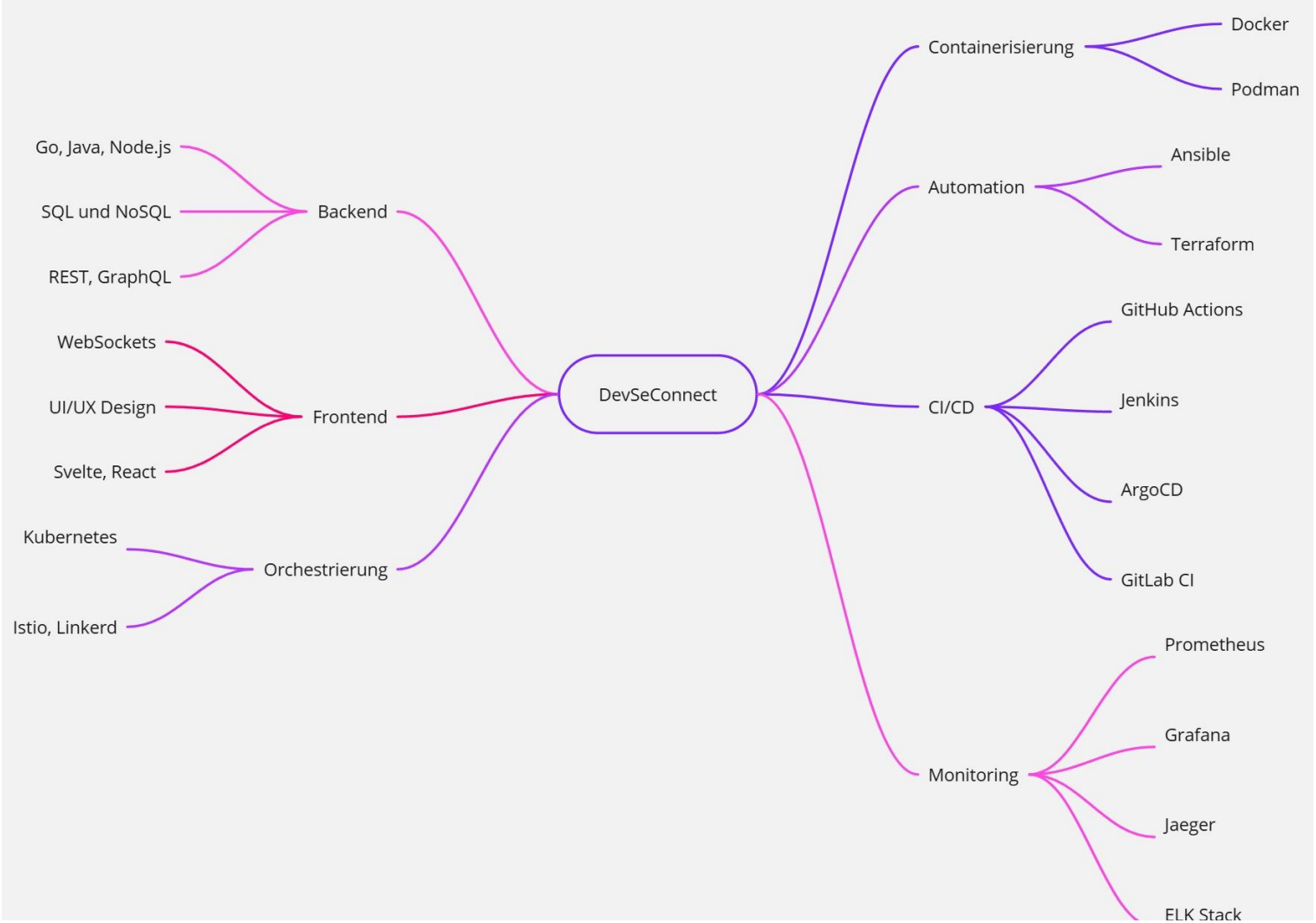


Abbildung 11 – MindMap

7.2 VARIANTENBILDUNG

Basierend auf den priorisierten Ideen wurden drei Hauptvarianten für DevSeConnect ausgearbeitet:

Abbildung 12 – Monoliths vs Microservices vs Serverless (Quelle: community.ops.io)

7.2.1 Variante «Microservices skalierbare Anwendungen»

- Fokus auf der Microservices-Architektur, die hohe Skalierbarkeit und Flexibilität ermöglicht.
- Ziel: Modularität durch unabhängige Services, die separat entwickelt, bereitgestellt und skaliert werden können.

7.2.1.1 Vorteile der «Microservices skalierbare Anwendungen»:

- **Hohe Skalierbarkeit:**
  - Einzelne Microservices können unabhängig voneinander skaliert werden, was eine flexible Anpassung an unterschiedliche Anforderungen ermöglicht.
- **Flexibilität bei der Technologieauswahl:**
  - Verschiedene Microservices können mit unterschiedlichen Technologien entwickelt werden, somit wird die technologische Vielfalt erhöht.

- **Unabhängige Entwicklungsteams:**
  - Teams können an verschiedenen Microservices parallel arbeiten, dadurch kann die Entwicklungsgeschwindigkeit erhöht sein.
- **Fehlerisolierung:**
  - Die Zuverlässigkeit ist erhöht, da mögliche Fehler in einem Microservice nicht das gesamte System beeinflussen.

7.2.1.2 Nachteile der «Microservices skalierbare Anwendungen»:

- **Komplexe Architektur**: Die Verwaltung vieler einzelner Services erfordert ausgeklügelte Orchestrierung und Monitoring, was den Verwaltungsaufwand erhöht.
- **Hoher Kommunikationsaufwand**: Da Microservices häufig über Netzwerke kommunizieren, können Latenzen und Fehlerquellen in der Kommunikation entstehen.
- **Herausforderungen bei der Datenkonsistenz**: Die Verteilung der Daten auf verschiedene Services macht das Sicherstellen der Konsistenz schwieriger.

Eignung «Microservices skalierbare Anwendungen» für DevSeConnect:

Diese Variante eignet sich hervorragend für DevSeConnect, wenn hohe Skalierbarkeit und Flexibilität Priorität haben. Sie ermöglicht eine unabhängige Entwicklung und Bereitstellung von Funktionen, die schnelles Wachstum und kontinuierliche Erweiterung für eine wachsende Entwickler-Community unterstützt. Insbesondere bei einer dynamischen Nutzerbasis, bei der einzelne Dienste unabhängig voneinander aktualisiert und skaliert werden müssen, bietet die Microservices-Architektur eine zukunftssichere Lösung.

7.2.2 Variante «Monolithische Architektur für integrierte Anwendungen»

- Fokus auf der Monolithischen Architektur, bei der die gesamte Anwendung als eine Einheit entwickelt und bereitgestellt wird.
- Ziel: Einfachheit in der Entwicklung und Verwaltung, aber begrenzte Skalierbarkeit.

7.2.2.1 Vorteile der «Monolithische Architektur für integrierte Anwendungen»:

- **Einfachere Entwicklung und Verwaltung**: Die gesamte Anwendung wird als eine Einheit entwickelt, was die Entwicklung weniger komplex macht.
- **Geringer initialer Aufwand**: Weniger Infrastruktur und Setup im Vergleich zu Microservices, da alles in einer Anwendung gebündelt wird.
- **Einfache Integration und Tests**: Da die Komponenten eng miteinander verbunden sind, können Funktionen einfacher integriert und getestet werden.

7.2.2.2 Nachteile «Monolithische Architektur für integrierte Anwendungen»:

- **Begrenzte Skalierbarkeit**: Die Skalierung des gesamten Systems kann schwieriger sein, da nicht einzelne Komponenten unabhängig skaliert werden können.
- **Weniger Flexibilität bei der Technologieauswahl**: Ein Monolith muss meist mit einer einzigen Technologieplattform entwickelt werden.
- **Schwierigere Weiterentwicklung**: Wenn die Anwendung wächst, kann der Monolith schwer zu warten und anzupassen sein, was die Weiterentwicklung verlangsamen kann.

Eignung «Monolithische Architektur für integrierte Anwendungen» für DevSeConnect:

Die monolithische Architektur könnte für DevSeConnect geeignet sein, wenn der Fokus auf einer einfachen Entwicklung und schnellen Bereitstellung liegt. Sie bietet sich besonders für kleinere Projekte an oder für Fälle, in denen die Nutzerbasis und die Funktionalität in den Anfangsphasen noch überschaubar sind. Wenn hohe Skalierbarkeit und Flexibilität keine Priorität haben und ein schnelles MVP (Minimum Viable Product) wichtig ist, kann diese Architektur effizient sein.

7.2.3 Variante «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen»

- Fokus auf der Serverless-Architektur, bei der die Infrastruktur dynamisch je nach Bedarf skaliert wird, ohne Serververwaltung durch den Entwickelnden.
- Ziel: Kosteneffizienz und automatische Skalierung für unvorhersehbare Lasten.

7.2.3.1 Vorteile «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen»:

- **Automatische Skalierung**: Die Anwendung skaliert automatisch je nach Bedarf, ohne dass dedizierte Server verwaltet werden müssen.
- **Kostenoptimierung**: Bezahlt wird nur für die tatsächliche Nutzung, was besonders kosteneffizient bei schwankenden oder geringeren Lasten ist.
- **Kein Infrastrukturmanagement**: Entwickelnde können sich auf den Code konzentrieren, ohne sich um die zugrunde liegende Serverinfrastruktur zu kümmern.

7.2.3.2 Nachteile «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen»:

- **Vendor Lock-in**: Abhängigkeit vom jeweiligen Cloud-Anbieter, was zukünftige Migrationen erschweren kann.
- **Eingeschränkte Kontrolle**: Weniger Kontrolle über die zugrunde liegende Infrastruktur kann zu Einschränkungen bei speziellen Anforderungen führen.
- **Potenzielle Latenz**: Funktionen, die selten aufgerufen werden, können beim ersten Aufruf verzögerte Reaktionszeiten haben.

Eignung «Serverless-Architektur für dynamische, bedarfsgerechte Anwendungen» für DevSeConnect:

Die Serverless-Architektur eignet sich ideal, wenn Kostenoptimierung und automatische Skalierung im Vordergrund stehen. Besonders für ein Projekt wie DevSeConnect, das möglicherweise mit einer kleineren Nutzerbasis startet, aber schnell wachsen kann, bietet Serverless eine flexible und kosteneffiziente Lösung. Sie ermöglicht, Ressourcen nur dann zu nutzen und zu bezahlen, wenn sie benötigt werden, was für ein Projekt mit unvorhersehbarem Nutzerverhalten vorteilhaft ist.

Die Schlussfolgerung lautet, dass die Microservices-Architektur die zukunftssicherste Lösung für das Projekt darstellt. Dies begründet sich in der hohen Flexibilität, Skalierbarkeit und Unabhängigkeit von einem Cloud-Anbieter.

7.3 PRIORISIERUNGSMETHODE

Die Priorisierungsmethode legt fest, welche Aufgaben und Anforderungen im Projekt vorrangig behandelt werden, um eine effiziente Ressourcennutzung und Zielerreichung sicherzustellen.

7.3.1 Begründung Priorisierungsmatrix:

Eine modifizierte Entscheidungsmatrix ist für eine Einzelarbeit geeignet, da sie folgendes aufweist:

- Eine objektive Bewertung durch vorher festgelegte Kriterien wird ermöglicht.
- Sie regt zur Selbstreflexion über die Bedeutung verschiedener Aspekte an.
- Besonders für technische Projekte eignet sie sich ist und erlaubt detaillierte Vergleiche.

7.3.2 Vorgehen Priorisierungsmatrix:

1. Fünf bis sieben Kernkriterien wurden definiert (z.B. Skalierbarkeit, Entwicklungszeit, Innovationsgrad, Umsetzbarkeit).
2. Die Kriterien wurden anschliessend in einer Skala von priorisiert bzw. zwischen eins bis fünf gewichtet, basierend jeweils auf deren Bedeutung für DevSeConnect.
3. Die Hauptideen wurden aus dem Mindmapping aufgelistet.
4. Jede Idee wurde anhand der Kriterien mit einer Skala von eins bis zehn bewertet.
5. Die Bewertungen wurden mit den Gewichtungen multipliziert und die Ergebnisse summiert.
6. Zum Schluss wurde eine Rangfolge der Ideen basierend auf den Gesamtpunktzahlen erstellt

Priorisierungsmatrix	Bereich: Architektur	Datum: 21.09.2024
		Bearbeiter: Markus Bürgi

	Bewertungskriterien	Gewichtung in %	Serverless		Microservices		Monolith	
			Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte
1	Skalierbarkeit	25%	9	**2.25**	10	**2.5**	5	**1.25**
2	Entwicklungszeit	20%	8	**1.6**	7	**1.4**	9	**1.8**
3	Innovationsgrad	10%	9	**0.9**	8	**0.8**	5	**0.5**
4	Umsetzbarkeit	10%	7	**0.7**	6	**0.6**	8	**0.8**
5	Flexilität	10%	10	**1**	9	**0.9**	4	**0.4**
6	Performance	15%	8	**1.2**	9	**1.35**	7	**1.05**
7	Wartbarkeit	10%	7	**0.7**	8	**0.8**	6	**0.6**
	Summe	**100%**		**8.35**		**8.35**		**6.4**

Tabelle 6 – Priorisierungsmatrix für Serverarchitektur

7.4 ARGUMENTATION MICROSERVICES-ARCHITEKTUR

Obwohl die Serverless-Architektur in der Priorisierungsmatrix ähnlich hohe Punktwerte wie die Microservices-Architektur erreicht hatte, gab es mehrere entscheidende Faktoren, die die Microservices-Architektur zur besseren Wahl der DevSeConnect machten.

7.4.1 Kostenfaktor:

- Serverless-Lösungen sind zwar anfänglich attraktiv, da sie keine dedizierte Infrastruktur benötigen, jedoch entstehen bei steigendem Nutzeraufkommen laufende Kosten, die sich schnell summieren können. Cloud-Anbieter berechnen die Nutzung auf Basis von Anfragen, Speicher, Rechenleistung und weiteren Faktoren.
- Microservices hingegen können lokal auf eigenen Servern oder internen Systemen ausgeführt werden, was langfristig kosteneffizienter ist, besonders wenn bereits über die nötige Hardware verfügt wird. Durch die Nutzung der bestehenden Infrastruktur entstehen keine zusätzlichen Cloud-Kosten, was im Betriebszustand von grossem Vorteil ist.

7.4.2 Kontrolle über die Infrastruktur:

- Mit einer Microservices-Architektur besteht die volle Kontrolle über die Infrastruktur, da nicht auf externe Cloud-Dienste zugegriffen wird. Dies bietet die Möglichkeit, die Ressourcen, die Leistung und die Sicherheit direkt zu steuern und zu optimieren.
- Serverless schränkt diese Kontrolle ein, da die Skalierung und Verwaltung der Server vollständig vom Cloud-Anbieter übernommen wird. Dadurch ist der Einfluss auf die Performance-Optimierung und die Nutzung der Ressourcen, was zu einer Abhängigkeit (Vendor Lock-in) führt, geringer.

7.4.3 Langfristige Flexibilität und Anpassungsfähigkeit:

- Die Microservices-Architektur ermöglicht das System schrittweise zu erweitern, ohne die gesamte Architektur zu überarbeiten. Bestehende Services können weiter optimiert und neue Services nach Bedarf hinzugefügt werden, ohne an die spezifischen Einschränkungen eines Cloud-Anbieters gebunden zu sein.

- Im Gegensatz dazu kann die Nutzung von Serverless-Funktionen teuer und kompliziert werden, wenn der Funktionsumfang des Systems wächst und die Nachfrage steigt. Da die Preisstruktur und Verfügbarkeit von Serverless-Diensten vollständig vom Anbietenden abhängig ist, kann sich dies langfristig problematisch auswirken.

7.4.4 Unabhängigkeit von Cloud-Anbietenden:

- Durch den Einsatz von Microservices bleibt die Unabhängigkeit von spezifischen Cloud-Anbietenden und deren Preisgestaltung bestehen. Aus technischer wie auch aus betrieblicher Perspektive entsteht eine gewisse Flexibilität.
- Serverless-Lösungen sind stark mit dem jeweiligen Cloud-Anbietenden (z.B. AWS, GCP und Azure) verbunden, was bei späteren Anpassungen oder Migrationen zu hohen Umstellungskosten führen kann.

7.4.5 Lokalität des Betriebs:

- Da DevSeConnect lokal auf bestehenden Computern läuft, ist die Microservices-Architektur optimal geeignet, um die vorhandene Hardware-Infrastruktur zu nutzen. Dies reduziert die Betriebskosten erheblich im Vergleich zu Serverless, bei der kontinuierliche Cloud-Kosten anfallen würden, selbst wenn die Anwendung nur selten genutzt wird.
- Microservices bieten hier die Flexibilität, lokal und ohne externe Abhängigkeiten performant zu arbeiten.

7.4.6 Entwicklerproduktivität und lokale Entwicklung:

- Bei einer Microservices-Architektur können Entwickelnde einzelne Services lokal entwickeln, testen und debuggen, ohne die gesamte Anwendung starten zu müssen. Dies fördert die Produktivität und ermöglicht schnellere Iterationen im Entwicklungsprozess.
- Serverless Entwicklung kann hingegen herausfordernd sein, da es schwieriger ist, die Umgebung lokal zu replizieren. Entwickelnde müssen oft mit emulierten Umgebungen arbeiten oder direkt in der Cloud testen, was den Entwicklungsprozess verlangsamen und verkomplizieren kann.

- Für DevSeConnect bedeutet dies, dass mit Microservices eine agilere Entwicklung umgesetzt werden kann und neue Features oder Verbesserungen schneller implementiert und getestet werden können. Diesen Vorteil ist besonders während der Anfangsphase des Projekts hilfreich.
- Zudem erleichtert die Microservices-Architektur die Zusammenarbeit im Entwicklungsteam, da verschiedene Teammitglieder unabhängig voneinander an verschiedenen Services arbeiten können, ohne sich gegenseitig zu behindern.

7.5 BEWERTUNGSANALYSE

Die Bewertungsanalyse dient dazu, die Projektziele und -ergebnisse systematisch zu prüfen und anhand festgelegter Kriterien auf ihre Wirksamkeit und Qualität zu bewerten.

Präferenzmatrix	Bereich: Backend	Datum: 21.09.2024
		Bearbeiter: Markus Bürgi

		Bewertungskriterien												
		GO	Java	Python	C#	Node.js	PHP	Ruby	C++	Rust	Gleam	Punktesumme je Kriterium	Gewichtungsfaktor in %	Rang
	Bewertungskriterien	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**			
1	GO		1	1	1	1	1	1	1	1	1	9	20%	1
2	Java	0		1	1	1	1	1	1	0	0	6	13%	4
3	Python	0	0		0	0	1	1	1	0	0	3	7%	7
4	C#	0	0	1		1	1	1	1	0	0	5	11%	5
5	Node.js	0	0	1	0		1	1	1	0	0	4	9%	6
6	PHP	0	0	0	0	0		1	1	0	0	2	4%	8
7	Ruby	0	0	0	0	0	0		0	0	0	0	0%	10
8	C++	0	0	0	0	0	0	1		0	0	1	2%	9
9	Rust	0	1	1	1	1	1	1	1		1	8	18%	2
10	Gleam	0	1	1	1	1	1	1	1	0		7	16%	3
										Prüfsumme			**100%**	

Tabelle 7 – Präferenzmatrix für Backend-Programmiersprachen

Die Präferenzanalyse ergab für die Programmiersprache «Go» die meisten Prozentwerte, wobei auch «Rust» und «Gleam» ein gutes Resultat erzielten. Der Autor hat bereits Erfahrung mit «Go» und schätzt es insbesondere aufgrund der freundlichen Lernbarkeit, einfachen Programmierung sowie Effizienz und Ressourcenarmut. Da «Go» noch nicht abgeschlossen war, stand als nächstes Analysen die Nutzwert- und Sensitivitätsanalysen an. Im Anschluss daran wurde die Präferenzanalyse für das Framework als Frontend durchgeführt.

Präferenzmatrix	Bereich: Frontend	Datum: 21.09.2024
		Bearbeiter: Markus Bürgi

		Bewertungskriterien												
		Sveltekit	React	Next.js	Vue.js	Angular	HTMX	Nuxt.js	Astro	Flask	Blazor	Punktesumme je Kriterium	Gewichtungsfaktor in %	Rang
Bewertungskriterien		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**			
1	Sveltekit		1	1	1	1	0	1	1	1	1	8	19%	1
2	React	0		0	0	1	0	0	0	1	1	3	7%	4
3	Next.js	0	1		1	1	0	1	0	1	1	6	14%	2
4	Vue.js	0	0	0		1	0	0	0	1	1	3	7%	4
5	Angular	0	0	0	0		0	0	0	1	1	2	5%	5
6	HTMX	1	1	0	1	1		0	0	1	1	6	14%	2
7	Nuxt.js	0	0	0	1	1	1		0	1	1	5	12%	3
8	Astro	0	1	1	1	1	1	1		1	1	8	19%	1
9	Flask	0	0	0	0	0	0	0	0		0	0	0%	7
10	Blazor	0	0	0	0	0	0	0	0	1		1	2%	6
										Prüfsumme			**100%**	

Tabelle 8 – Präferenzmatrix für Frontend-Programmiersprachen

Bei dieser Präferenzanalyse zeigte sich, dass die Frontend-Frameworks «Sveltekit» und «Astro» sowie «Next.js» und «HTMX» eine gute Wahl darstellen. Allerdings hat der Autor bereits Erfahrungen mit «Sveltekit» gesammelt, somit feststellen können, dass «Sveltekit» eine ähnliche Leistung, Ressourcenarmut und Lernfreundlichkeit aufzeigt wie «Astro».

In der nachfolgenden Analyse kann das Backend und das Frontend von der Nutzwert- und Sensitivitätsanalyse betrachtet werden.

Nutzwertanalyse	Bereich: Backend		Datum: 21.09.2024
			Bearbeiter: Markus Bürgi

			Bewertungskriterien																			
				GO		Java		Python		C#		Node.js		PHP		Ruby		C++		Rust		Gleam	
	Bewertungskriterien	Gewichtung in %	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte
1	Leistung	20%	3	**0.6**	3	**0.6**	2	**0.4**	3	**0.6**	2	**0.4**	2	**0.4**	2	**0.4**	3	**0.6**	3	**0.6**	2	**0.4**
2	Skalierbarkeit	10%	3	**0.3**	3	**0.3**	2	**0.2**	3	**0.3**	2	**0.2**	2	**0.2**	2	**0.2**	3	**0.3**	3	**0.3**	2	**0.2**
3	DevOps-Integration	5%	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**	2	**0.1**	2	**0.1**
4	Ökosystem	10%	2	**0.2**	3	**0.3**	3	**0.3**	3	**0.3**	3	**0.3**	3	**0.3**	2	**0.2**	2	**0.2**	2	**0.2**	1	**0.1**
5	Lernkurve	30%	3	**0.9**	1	**0.3**	3	**0.9**	2	**0.6**	3	**0.9**	3	**0.9**	3	**0.9**	1	**0.3**	1	**0.3**	2	**0.6**
6	Zukunftssicherheit	20%	3	**0.6**	3	**0.6**	3	**0.6**	3	**0.6**	3	**0.6**	2	**0.4**	2	**0.4**	3	**0.6**	3	**0.6**	2	**0.4**
7	Community-Support	5%	2	**0.1**	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**	1	**0.05**
	Summe	**100%**		**2.85**		**2.4**		**2.7**		**2.7**		**2.7**		**2.45**		**2.3**		**2.2**		**2.2**		**1.85**
	Rang			**1**		**4**		**2**		**2**		**2**		**3**		**5**		**6**		**6**		**7**

Tabelle 9 – Nutzeranalyse für Backend-Programmiersprachen

Im Rahmen der vorliegenden Untersuchung wurden sieben Bewertungskriterien definiert, die für die weitere Analyse von entscheidender Bedeutung waren. Der Autor möchte zudem darauf hinweisen, dass die Gewichtung bereits bei der Präferenzanalyse berücksichtigt wurde und somit eine mögliche Verrechnung des Punktestands sein könnte. Es ist trotzdem anhand der Auswertung ersichtlich, dass «Python», «C#» und «Node.js» besonders gut abgeschnitten haben und «Go» die

besten Ergebnisse erzielten. In der Folge werden drei Sensitivitätsanalysen mit den groben dynamischen Punktzahlen präsentiert, deren Resultate jedoch als unrealistisch zu bewerten sind.

Die erste Szenerie demonstriert ein signifikantes Interesse an der Programmiersprache «GO», was sich in der höchsten Punktzahl wiederspiegelt.

			GO	Java	Python	C#	Node.js	PHP	Ruby	C++	Rust	Gleam
Szenerie 1	**Summe**	**100%**	**2.85**	**2.4**	**2.7**	**2.7**	**2.7**	**2.45**	**2.3**	**2.2**	**2.2**	**1.85**
	Präferenzmatrix		**20%**	**13%**	**7%**	**11%**	**9%**	**4%**	**0%**	**2%**	**18%**	**16%**
	Gesamt		**0.57**	**0.31**	**0.19**	**0.3**	**0.24**	**0.1**	**0**	**0.04**	**0.4**	**0.3**

Tabelle 10 – Sensitivitätsanalyse für Szenario 1 zur Auswahl der Backend-Programmiersprache

Die Die zweite Szenerie resultierte, dass «Gleam» die beste Wahl sei.

			GO	Java	Python	C#	Node.js	PHP	Ruby	C++	Rust	Gleam
Szenerie 2	**Summe**	**100%**	**2.15**	**2.3**	**2.3**	**2.3**	**2.3**	**2.55**	**2.7**	**2.5**	**2.5**	**3**
	Präferenzmatrix		**20%**	**13%**	**7%**	**11%**	**9%**	**4%**	**0%**	**2%**	**18%**	**16%**
	Gesamt		**0.43**	**0.3**	**0.16**	**0.25**	**0.21**	**0.1**	**0**	**0.05**	**0.45**	**0.48**

Tabelle 11 – Sensitivitätsanalyse für Szenario 2 zur Auswahl der Backend-Programmiersprache

Die dritte Szenerie zeigte auf, dass auch «Gleam» ein gutes Resultat erzielte.

			GO	Java	Python	C#	Node.js	PHP	Ruby	C++	Rust	Gleam
Szenerie 3	**Summe**	**100%**	**1.3**	**1.3**	**1.6**	**2**	**1.4**	**1.9**	**2.2**	**2.1**	**2.1**	**2.85**
	Präferenzmatrix		**20%**	**13%**	**7%**	**11%**	**9%**	**4%**	**0%**	**2%**	**18%**	**16%**
	Gesamt		**0.26**	**0.17**	**0.11**	**0.22**	**0.13**	**0.08**	**0**	**0.04**	**0.38**	**0.46**

Tabelle 12 – Sensitivitätsanalyse für Szenario 3 zur Auswahl der Backend-Programmiersprache

Die Auswertung ergab, dass «Gleam» zwei Mal das höchste Resultat erzielte. Allerdings verfügte der Autor über keine Erfahrungen mit dieser Programmierung und konnte für die Diplomarbeit diese nicht umsetzen. Er griff auf «Go» zurück, da dies ebenso gute Ergebnisse liefert und in diesem Kontext eine vielversprechende Programmiersprache ermöglichte.

TEKO Schweizerische Fachschule

Nutzwertanalyse	Bereich: Frontend		Datum: 21.09.2024
			Bearbeiter: Markus Bürgi

			Bewertungskriterien																					
		Gewichtung in %	Sveltekit		React		Next.js		Vue.js		Angular		HTMX		Nuxt.js		Astro		Flask		Blazor	
	Bewertungskriterien		Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte
1	Leistung	20%	3	**0.6**	2	**0.4**	3	**0.6**	2	**0.4**	2	**0.4**	2	**0.4**	3	**0.6**	3	**0.6**	2	**0.4**	2	**0.4**
2	Entwicklungstempo	10%	3	**0.3**	2	**0.2**	2	**0.2**	3	**0.3**	2	**0.2**	3	**0.3**	2	**0.2**	3	**0.3**	3	**0.3**	2	**0.2**
3	DevOps-Integration	5%	2	**0.1**	3	**0.15**	3	**0.15**	2	**0.1**	3	**0.15**	1	**0.05**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**
4	Ökosystem	5%	2	**0.1**	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	1	**0.05**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**
5	Lernkurve	20%	3	**0.6**	2	**0.4**	2	**0.4**	3	**0.6**	1	**0.2**	3	**0.6**	2	**0.4**	3	**0.6**	2	**0.4**	1	**0.2**
6	SEO-Freundlichkeit	5%	2	**0.1**	2	**0.1**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**	3	**0.15**	3	**0.15**	1	**0.05**	2	**0.1**
7	TypeScript-Support	30%	2	**0.6**	3	**0.9**	3	**0.9**	2	**0.6**	3	**0.9**	1	**0.3**	2	**0.6**	2	**0.6**	1	**0.3**	1	**0.3**
8	Community-Support	5%	2	**0.1**	3	**0.15**	3	**0.15**	3	**0.15**	3	**0.15**	1	**0.05**	3	**0.15**	2	**0.1**	2	**0.1**	2	**0.1**
	Summe	**100%**		**2.5**		**2.45**		**2.7**		**2.4**		**2.25**		**1.85**		**2.4**		**2.55**		**1.75**		**1.5**
	Rang			**3**		**4**		**1**		**5**		**6**		**7**		**5**		**2**		**8**		**9**

Tabelle 13 – Nutzeranalyse für Frontend-Programmiersprachen

Die Bewertungskriterien sind im Wesentlichen identisch mit denen des Backends, allerdings stellte die Aufteilung der Prozentwerte eine grosse Herausforderung dar. «Next.js» erreichte den fünften Rang und «Astro» den Zweiten. Dies ist darauf zurück zu führen, dass «Next.js» eine entwicklungsstarke Legacy-Anwendung ist. Zudem ist darauf hinzuweisen, dass die genannten Frameworks «Sveltekit» und «Astro» während der Projektarbeit zum Einsatz kamen, obwohl «Next.js», das auf React basiert, zusätzlich zu SSR auch SSG nutzt. Allerdings ist diese Vorgehensweise für Lernende weniger geeignet. Daher entschied sich der Autor mit «Sveltekit» für dieses Projekt zu arbeiten.

In der ersten Szenerie lässt sich ein positives Resultat für «Sveltekit» und «Astro» verzeichnen. Allerdings ist die Sensitivitätsanalyse lediglich eine unrealistische Bewertung. Insgesamt kann jedoch von einem guten Ergebnis gesprochen werden.

			Sveltekit	React	Next.js	Vue.js	Angular	HTMX	Nuxt.js	Astro	Flask	Blazor
Szenerie 1	**Summe**	**100%**	**2.5**	**2.45**	**2.7**	**2.4**	**2.25**	**1.85**	**2.4**	**2.55**	**1.75**	**1.5**
	Präferenzmatrix		**19%**	**7%**	**14%**	**7%**	**5%**	**14%**	**12%**	**19%**	**0%**	**2%**
	Gesamt		**0.48**	**0.17**	**0.38**	**0.17**	**0.11**	**0.26**	**0.29**	**0.48**	**0**	**0.03**

Tabelle 14 – Sensitivitätsanalyse für Szenario 1 zur Auswahl der Frontend-Programmiersprache

In der zweiten Szenerie kann «Sveltekit» schliesslich das optimale Resultat erzielen, muss sich jedoch knapp mit einer Differenz von 0,01 % gegen «Astro» geschlagen geben.

			Sveltekit	React	Next.js	Vue.js	Angular	HTMX	Nuxt.js	Astro	Flask	Blazor
Szenerie 2	**Summe**	**100%**	**2.5**	**2.55**	**2.3**	**2.6**	**2.35**	**2.25**	**2.6**	**2.45**	**1.75**	**2.5**
	Präferenzmatrix		**19%**	**7%**	**14%**	**7%**	**5%**	**14%**	**12%**	**19%**	**0%**	**2%**
	Gesamt		**0.48**	**0.18**	**0.32**	**0.18**	**0.12**	**0.32**	**0.31**	**0.47**	**0**	**0.05**

Tabelle 15 – Sensitivitätsanalyse für Szenario 2 zur Auswahl der Frontend-Programmiersprache

In der dritten Szenerie kann erneut ein gutes Resultat für «Sveltekit» verzeichnet werden. Die Abweichung beträgt lediglich 0,02 % und führt dazu, dass «Sveltekit» gegen «Astro» erneut sich geschlagen geben muss.

			Sveltekit	React	Next.js	Vue.js	Angular	HTMX	Nuxt.js	Astro	Flask	Blazor
Szenerie 3	**Summe**	**100%**	**2**	**2.1**	**1.6**	**2.2**	**1.9**	**1.95**	**2.2**	**1.9**	**2.45**	**2.5**
	Präferenzmatrix		**19%**	**7%**	**14%**	**7%**	**5%**	**14%**	**12%**	**19%**	**0%**	**2%**
	Gesamt		**0.38**	**0.15**	**0.22**	**0.15**	**0.1**	**0.27**	**0.26**	**0.36**	**0**	**0.05**

Tabelle 16 – Sensitivitätsanalyse für Szenario 3 zur Auswahl der Frontend-Programmiersprache

7.6 ARGUMENTATION BACKEND UND FRONTEND

Die Architektur des Backends und Frontends stellte eine wesentliche Grundlage für die Stabilität und Effizienz der Anwendung dar. Im Folgenden wird erläutert, welche spezifischen Überlegungen und Technologien zur Optimierung beider Bereiche getroffen wurden, um eine reibungslose Funktionalität und ein optimales Nutzungserlebnis sicherzustellen.

7.6.1 Backend: «Go»

- Die Präferenzanalyse hatte ergeben, dass «Go» die geeignetste Programmiersprache für das Backend war. Es bot eine exzellente Ausgewogenheit zwischen Leistungsfähigkeit, Ressourceneffizienz und Einfachheit in der Entwicklung.
- Die bereits vorhandene Erfahrung des Autors mit der Programmiersprache «Go», die schnelle Implementierung sowie die effiziente und skalierbare Backend-Architektur stellten wesentliche Vorteile dar.
- Die Programmiersprache «Go» zeichnete sich ebenso durch eine hohe Effizienz aus, was sich positiv auf die Ressourcenschonung und Performance auswirkte. Der geringe «Runtime Overhead» ermöglichte eine optimierte Ressourcennutzung, welche insbesondere für rechenintensive Anwendungen von Vorteil war.
- Als mögliche Alternativen zu «Go» sind «Rust» und «Gleam» zu nennen, die ebenfalls gute Ergebnisse erzielten. Da jedoch keine praktischen Erfahrungen des Autors mit diesen Sprachen vorlag, konnten sie nicht als optimale Wahl betrachtet werden. «Go» stellte zudem eine stabile und zukunftssichere Grundlage dar.

7.6.2 Frontend: «SvelteKit»

- Die Präferenzanalyse hatte ergeben, dass «SvelteKit» die höchste Punktzahl erreichte und somit die beste Wahl für das Frontend war. Es stellte eine leichte und performante Lösung für moderne Webanwendungen, insbesondere mit serverseitigem Rendering (SSR), bereit.
- Die bisherige Erfahrung des Autors mit «SvelteKit» stellte ein entscheidendes Kriterium bei der Evaluierung dar. Aufgrund der ressourcenschonenden und

lernfreundlichen Eigenschaften, ähnlich denen von «Astro», wurde ein flüssiger Entwicklungsprozess gewährleistet und das Risiko von Komplikationen minimiert.

- Als potenzielle Alternativen wurden zudem die Frameworks «Astro», «Next.js» und «HTMX» evaluiert. Dabei hatte sich gezeigt, dass «SvelteKit» insbesondere durch seine Kombination aus Performance, Flexibilität und einfacher Integration herausstach.
- Die langfristige Skalierbarkeit stellte ein weiteres entscheidendes Kriterium dar. «SvelteKit» überzeugte in diesem Kontext durch eine hohe Performance sowie eine exzellente Anpassungsfähigkeit, wodurch zukünftige Erweiterungen und Optimierungen erheblich erleichtert werden.

Die Analyse der Bewertungen hatte ergeben, dass eine detailliertere Untersuchung des Automatisierungssystems für CI/CD und der Containerisierung nicht erforderlich war. Dies war in erster Linie darauf zurückzuführen, dass die Auswahl an alternativen Plattformen und Technologien in diesem Bereich begrenzt war und die bereits getroffene Entscheidung auf fundierten, praxisorientierten Erfahrungen basierte. In vorangehenden Projekten manifestierte sich eine deutliche Tendenz, welche Plattformen und Technologien in tatsächlichen Szenarien die höchste Zuverlässigkeit und Effizienz aufweisen.

Ein weiterer wesentlicher Aspekt war der bereits beträchtliche Aufwand, der in die Einarbeitung und Implementierung neuer Frameworks und Technologien investiert wurde. Eine Fortführung der Testphase wäre nicht nur mit einem unnötigen Ressourcenaufwand verbunden gewesen, sondern hätte auch zu einer erheblichen Verzögerung des Projektfortschritts führen können. In Anbetracht des gesetzten Meilensteins, der höchste Priorität genoss, sowie der potenziellen Risiken weiterer Änderungen wäre es unklug gewesen, zusätzliche Technologien zu prüfen, die das Projekt in seiner Komplexität noch weiter vergrössert hätten.

Aufgrund der langjährigen Erfahrung des Autors bezüglich den gewählten Technologien erarbeitete er einen Plan, der sowohl in Bezug auf Effizienz, Skalierbarkeit als auch Nachhaltigkeit optimiert war. Die genannten Erfahrungen erlaubten dem Autor, die aktuelle Lösung so zu optimieren, dass sie den Anforderungen von DevSeConnect gerecht wurde, ohne unnötige Komplexität oder Risiken einzuführen. Der vom Autor gewählte Ansatz basierte auf bewährten Grundlagen und gewährleistete, dass das Risiko für Verzögerungen und unerwartete Komplikationen minimiert wurde.

Zusammenfassend lässt sich konstatieren, dass die aktuelle Strategie bereits eine solide, zukunftssichere Basis bietet und es keine Notwendigkeit gab, alternative Ansätze weiterzuverfolgen. Der Fokus lag daher auf der konsequenten Umsetzung und der termingerechten Fertigstellung, um den Meilenstein zu erreichen. Diesbezüglich waren keine weiteren Massnahmen erforderlich, um das Projekt erfolgreich abzuschliessen.

7.7 CI/CD-PIPELINE

Die Entscheidung für die CI/CD-Pipeline basierte auf der bereits bewährten Nutzung von «Jenkins» und «ArgoCD» in vorangegangenen Projekten. Eine umfassende Bewertungsanalyse alternativer CI/CD-Tools wurde nicht durchgeführt, da die Auswahl an Plattformen in diesem Bereich sehr eingeschränkt ist und bereits ausreichend fundierte Erfahrungen mit den ausgewählten Technologien vorlagen.

7.7.1 «Jenkins»

«Jenkins» stellt aufgrund seiner Open-Source-Natur und der vollständigen Kostenfreiheit die bevorzugte Lösung für die Continuous Integration dar. Obwohl Alternativen wie «GitLab CI» evaluiert wurden, fiel die Wahl auf «Jenkins», da «GitLab» in der Regel auf cloudbasierte Lösungen ausgerichtet ist, die eine langfristige Bindung an eine spezifische Domain und Infrastruktur erfordert. Dies hätte zu Einschränkungen in der Flexibilität und möglicherweise höheren Kosten geführt, was in diesem Projekt vermieden werden sollte. «Jenkins» bietet eine stabile, langfristige und vollständig selbst hostbare Lösung.

7.7.2 «ArgoCD»

Für die Continuous Deployment-Prozesse wurde «ArgoCD» gewählt. «ArgoCD» überzeugte durch seine einfache Handhabung und die native Integration mit «Kubernetes». Es bat die Möglichkeit, Deployments in Kubernetes-Clustern zu automatisieren und zu überwachen. Aufgrund dieser praktischen und benutzerfreundlichen Eigenschaften war keine tiefere Evaluierung anderer Tools notwendig, da «ArgoCD» die Anforderungen des Projekts bereits vollumfänglich abdeckte.

7.7.3 «Kubernetes»

Kubernetes wurde in der Form von K3s eingesetzt, einer leichtgewichtigen und für lokale Umgebungen optimierten Version. K3s bietet eine einfache Bereitstellung und Verwaltung von Containern in lokalen Umgebungen, was es ideal für Entwicklungsprojekte macht, die ohne umfangreiche Cloud- oder Cluster-Ressourcen auskommen möchten.

7.7.4 «Traefik» als Ingress Controller

Die Entscheidung, den Ingress Controller "Traefik" zu implementieren, basierte auf dessen nativer Integration in K3s. Aufgrund der Integration von "Traefik" als Standard-Ingress-Controller in K3s war keine zusätzliche Konfiguration erforderlich, wodurch sich der Wartungsaufwand im Vergleich zu externen Lösungen wie NGINX erheblich reduzierte. Des Weiteren integrierte "Traefik" die Funktionalitäten eines API-Gateways und eines Load Balancers, wodurch eine zentrale Lösung für das Routing und den Datenverkehr im Cluster bereitgestellt wurde. Die ressourcenschonende Architektur des Systems erwies sich insbesondere in lokalen Entwicklungsumgebungen als vorteilhaft, da sie zu einer Einsparung von Ressourcen in kleineren Entwicklungsprojekten führte.

Begründung der Entscheidung

Die Entscheidung, keine weitergehende Bewertungsanalyse für die CI/CD-Tools durchzuführen, basierte auf der Erkenntnis, dass die oben genannten Tools bereits umfangreich getestet und bewährt waren. Eine Fortführung der Analyse oder der Testphase mit zusätzlichen Technologien hätte einen unnötigen Ressourcenaufwand zur Folge gehabt und möglicherweise zu Verzögerungen im Projektfortschritt geführt. Des Weiteren konnte in vorangehenden Projekten festgestellt werden, dass die genannten Technologien in der Praxis eine hohe Zuverlässigkeit und Effizienz aufweisen.

Die Kombination aus «Jenkins» für CI, «ArgoCD» für CD, «K3s» für die lokale Kubernetes-Umgebung und «Traefik» als Ingress Controller erwies sich als eine stabile, effiziente und zukunftssichere Grundlage, die den Anforderungen von DevSeConnect gerecht wurde. Die Verwendung dieser bewährten Technologien minimierte das Risiko unvorhergesehener Komplikationen und ermöglichte einen reibungslosen Projektverlauf bis zum gesetzten Meilenstein.

7.8 UML

Die UML-Diagramme dienen der Visualisierung der Struktur und des Ablaufs der Systemkomponenten. Dadurch wird ein gemeinsames Verständnis der Architektur geschaffen.

7.8.1 Sequenzendiagramm

Das Sequenzdiagramm präsentiert die zeitliche Abfolge der Interaktionen zwischen den Systemkomponenten und veranschaulicht den Nachrichtenfluss.

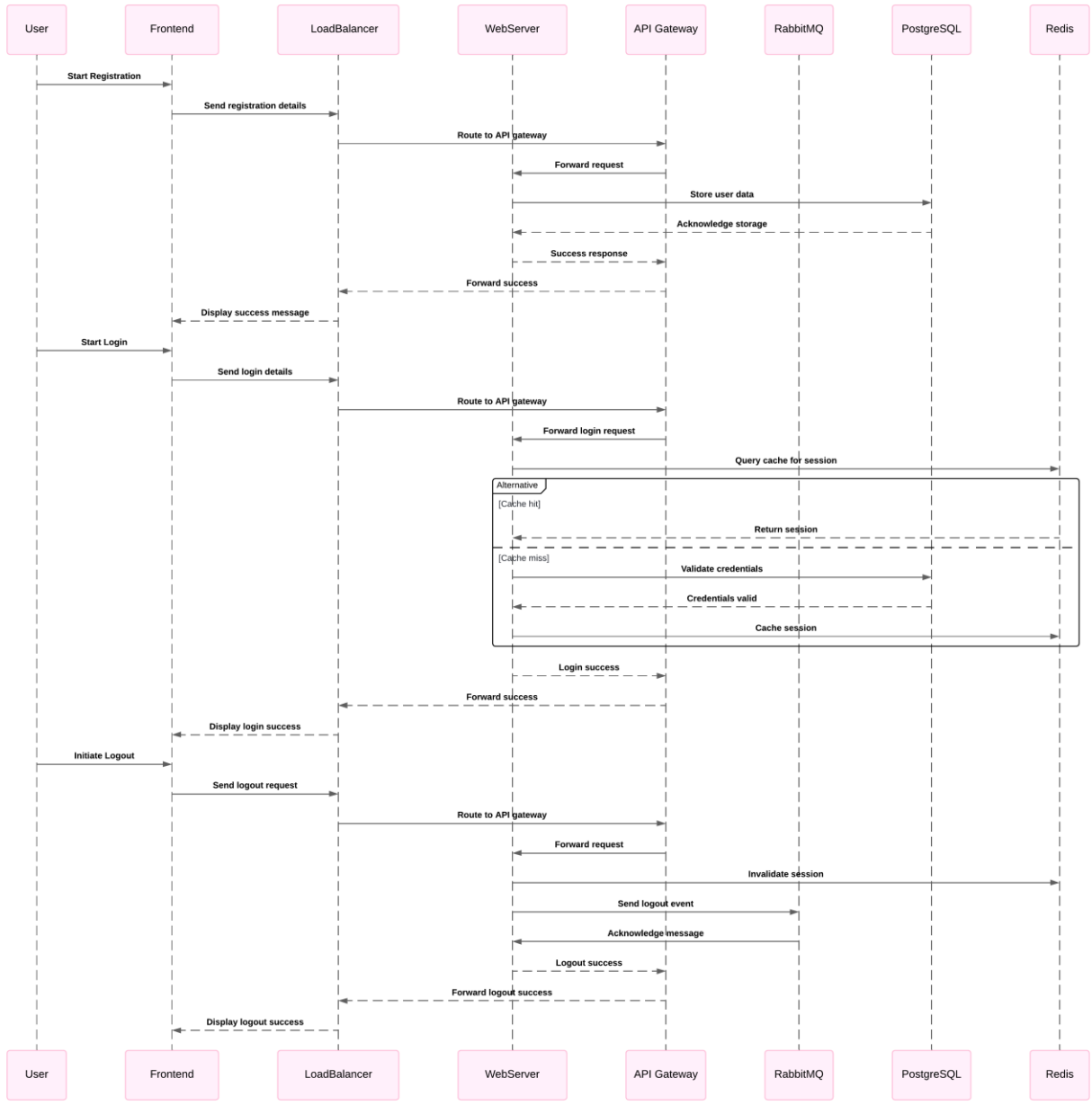


Diagramm 7 – Sequenzendiagramm

7.8.2 Aktivitätsdiagramm

Das Aktivitätsdiagramm visualisiert die logischen Abläufe und Entscheidungen im System und dient somit der Analyse der Prozessschritte.

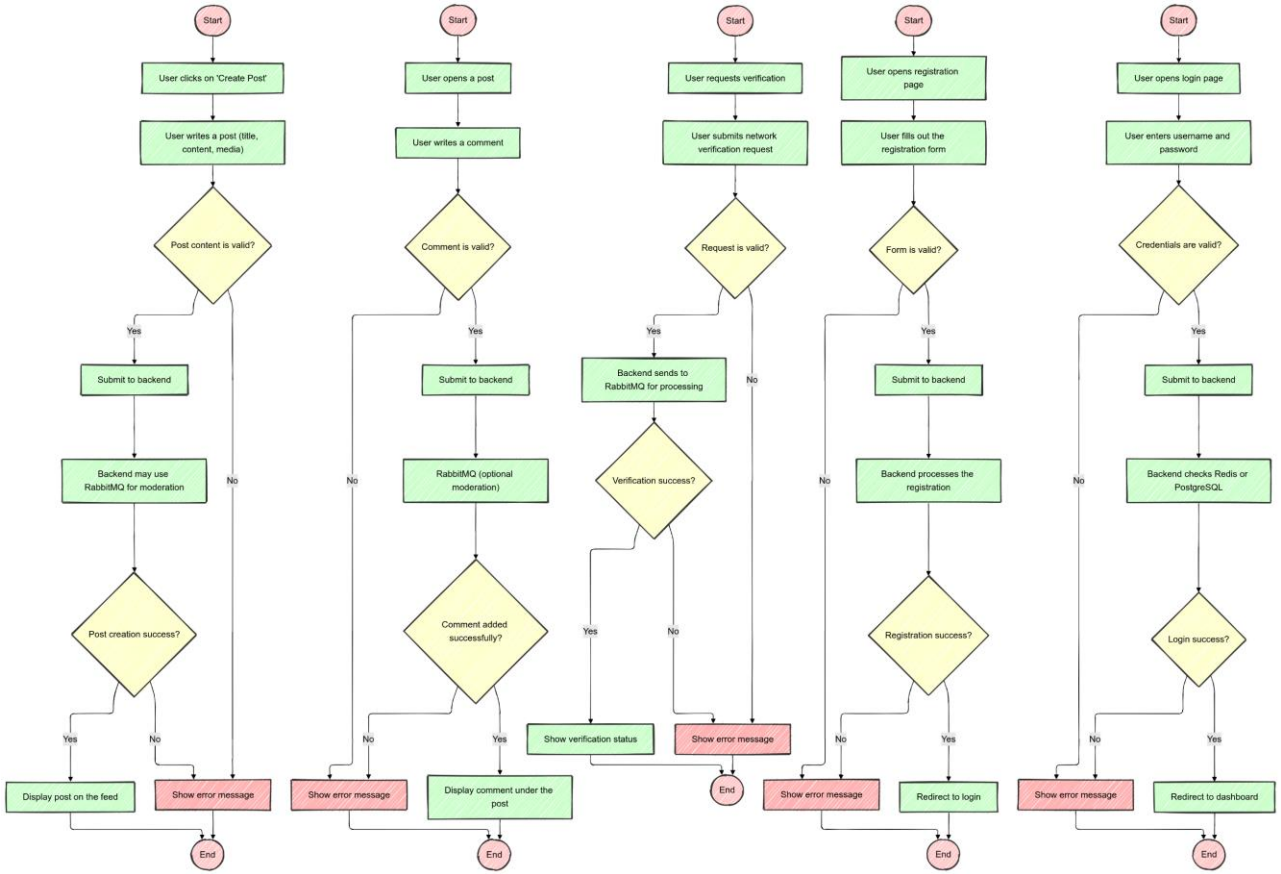


Diagramm 8 – Aktivitätsdiagramm

7.9 VORBEREITUNG

Zu Beginn meiner Überlegungen ging ich davon aus, dass das Monitoring in der Containerisierung aufgebaut werden sollte, um eine einfache Einbindung mit PostgreSQL, Backend und Frontend zu gewährleisten. Daher erfolgte zuerst die Einrichtung von «Prometheus» und «Grafana». Es wäre sinnvoll gewesen, jedoch sollte der Meilenstein nicht in starrem Rahmen definiert werden, da die agile Arbeitsweise für den Autor von grosser Bedeutung war.

7.9.1 Voraussetzungen

Bevor mit der Einrichtung begonnen wurde, musste die folgenden Komponenten installiert und konfiguriert sein:

- Docker
- Kubernetes-Cluster
- K3s
- Helm (Kubernetes-Paketmanager)

7.9.2 Übersicht von «Prometheus», «Grafana», «Helm» und «K3s»

- **Prometheus:** Ein Open-Source-Überwachungs- und Alarmierungs-Toolkit, das für «Kubernetes» entwickelt wurde. Es zeichnet sich insbesondere durch die Fähigkeit zur Erfassung und Abfrage von Echtzeit-Metriken aus.
- **Grafana:** Eine leistungsstarke Analyse- und Überwachungsplattform, die sich nahtlos in «Prometheus» integriert und auf Basis von Visualisierungen die Erstellung von Dashboards ermöglicht.
- **Helm:** Der «Kubernetes-Paketmanager», der die Bereitstellung und Verwaltung von Anwendungen durch Diagramme vereinfacht.
- **K3s:** Eine leichtgewichtige «Kubernetes-Distribution», die speziell für Edge- und IoT-Umgebungen entwickelt wurde. «K3s» vereinfacht den Betrieb von «Kubernetes», indem es die Systemanforderungen reduziert und dennoch volle Kompatibilität mit den Standard-Kubernetes-Funktionalitäten bietet. Es ist besonders für Entwicklungs- und Produktionsumgebungen geeignet, da es schneller und ressourcenschonender als herkömmliche Kubernetes-Installationen ist.

Die Installation des «Helm» Charts erfolgte auf einer vorbereiteten Plattform. Anschliessend wurde durch «Kubernetes» der Diensttyp von **ClusterIP** auf **NodePort** geändert, um eine externe IP-Adresse und einen statischen Port für «K3s» bereitzustellen.

Die grafische Benutzeroberfläche «Grafana Dashboard für PostgreSQL» ermöglichte einen ausgezeichneten Überblick über die relevanten Daten.

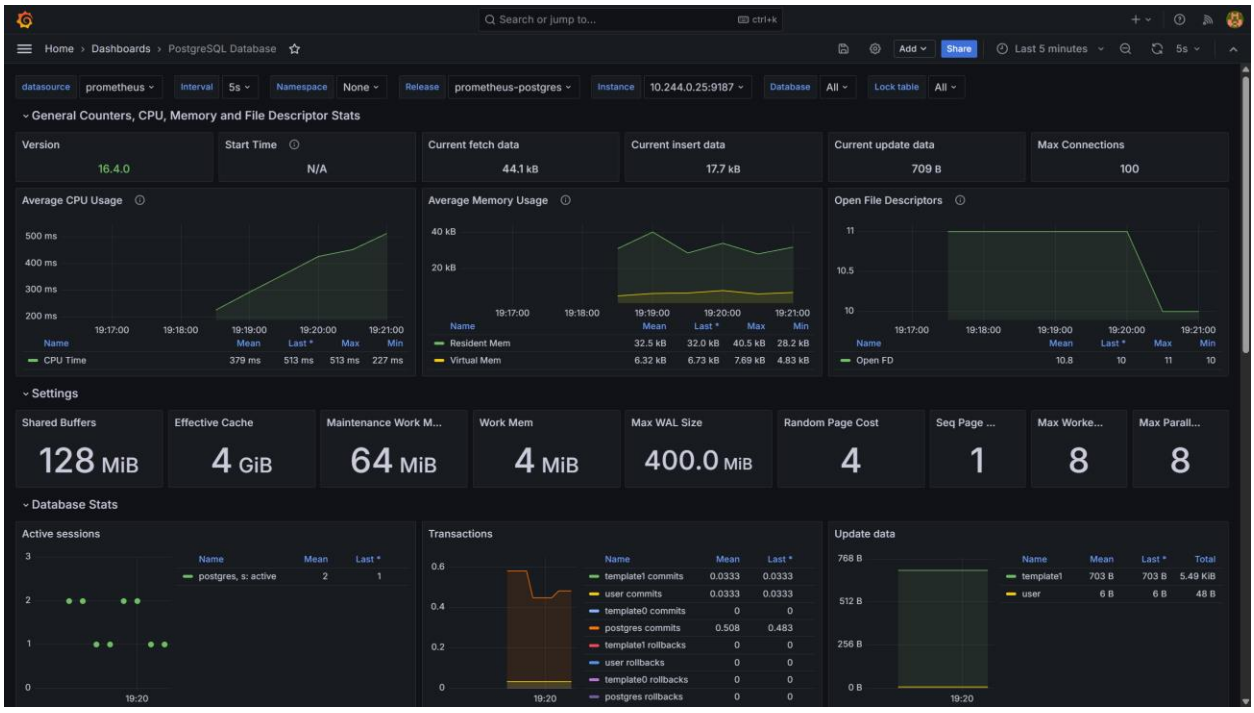


Abbildung 13 – Monitoring von PostgreSQL in Grafana

Die Vorbereitung von Monitoring und Datenbank war abgeschlossen. Künftig wurde jedoch noch das Backend und Frontend auf dem «Grafana» Dashboard erzeugt.

7.10 DATENBANK

Die Datenbank fungierte als zentrales Element im Hinblick auf die Speicherung und Verwaltung von Daten auf der Plattform. Sie sicherte den strukturierten und effizienten Zugang zu Informationen.

7.10.1 Entity-Relationship-Diagramm

Die Datenstruktur wurde durch ein Entity-Relationship-Diagramm veranschaulicht, welches die Beziehungen zwischen den wichtigsten Entitäten der Datenbank darstellte.

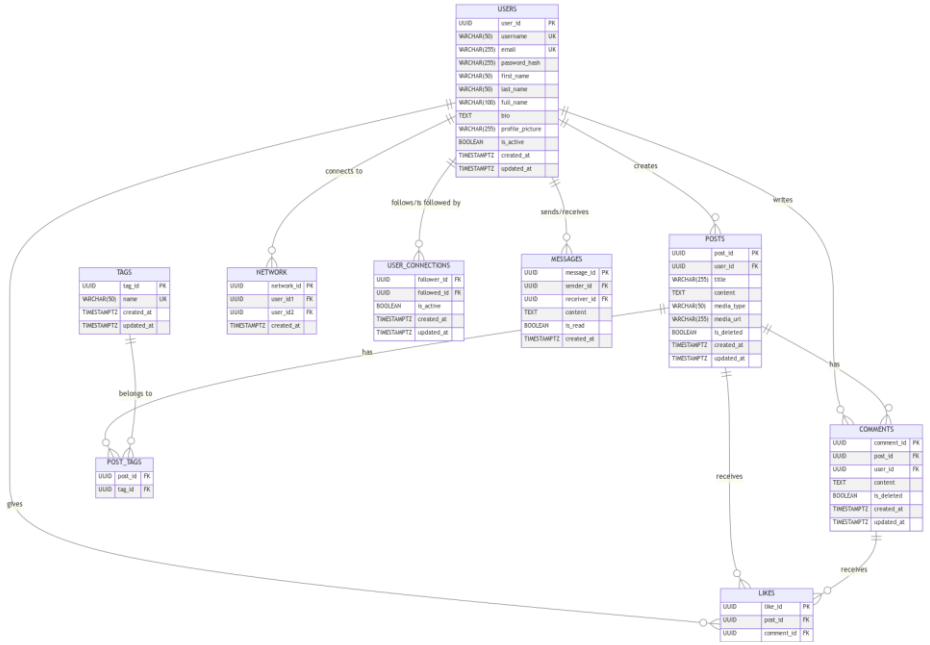


Diagramm 9 – ERD (Entity-Relationship-Diagramm)

7.10.2 Datenbank-Implementierung

Die Implementierung der Datenbank erfolgte ausschliesslich unter Verwendung von «PostgreSQL». Das Tool «pgAdmin» diente dabei als grafisches Verwaltungsinstrument, welches die Verwaltung der Datenbankstruktur sowie die Durchführung von Abfragen in einer effizienten Weise ermöglichte. Die Auswahl von «PostgreSQL» basierte auf dessen Stabilität, Open-Source-Natur sowie der Fähigkeit, grosse Datenmengen effizient zu verarbeiten.

Die Datenbank basierte auf dem «Entity-Relationship-Modell (ERD)», das die logischen Beziehungen zwischen den Entitäten der Plattform definieret:

- Benutzende (User): Diese Entität enthält Informationen zu den registrierten Nutzenden, wie user_id, username, email und password_hash. Die Felder created_at und updated_at speichern die Zeitstempel der Kontoerstellung und -aktualisierung.
- Posts (Beiträge): Diese Entität speichert die von den Benutzenden erstellten Beiträge. Die Felder umfassen post_id, title, content, created_at und den Fremdschlüssel user_id, der den Beitrag mit der jeweiligen benutzenden Person verknüpft. Ein:e Benutzende kann mehrere Beiträge erstellen, was durch eine 1

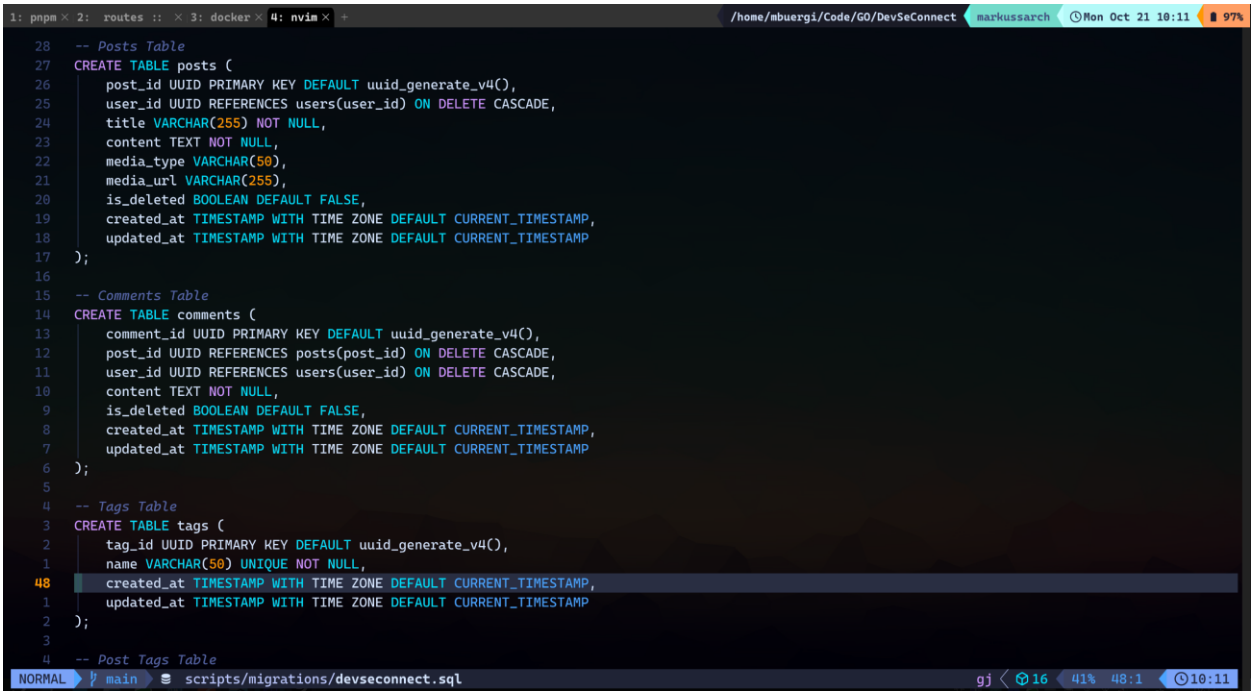
zwischen Benutzern und Beiträgen modelliert ist.

- Kommentare (Comments): Die Kommentare sind mit den Posts und Benutzenden verknüpft. Die Felder umfassen comment_id, content, post_id, user_id und created_at. Es besteht eine 1

zwischen Posts und Kommentaren, da ein Post mehrere Kommentare haben kann.

7.10.3 Datenbank-Beziehungen und Fremdschlüssel

Die Beziehungen zwischen den Tabellen wurden über Fremdschlüssel definiert, um die referenzielle Integrität zu gewährleisten. Dies stellte sicher, dass jeder Kommentar und jeder Post eindeutig einem:r Benutzenden zugeordnet sind. Dabei wurde auf Mechanismen wie «ON DELETE CASCADE» verzichtet, da die manuelle Verwaltung der Löschvorgänge in dieser Implementierung bevorzugt wurde.



Programmierung 1 – SQL-Skripte zur Tabellenerstellung

7.10.4 Evaluierung der Datenbank

Im Rahmen der Implementierung wurden umfassende Datenbanktests durchgeführt, um die Konsistenz und Integrität des Datenmodells zu validieren. Diese Tests prüften:

- Fremdschlüssel-Validierung:

  Es wurde überprüft, dass jeder Datensatz in der Kommentar- und Post-Tabelle gültige Verknüpfungen zu einem:r existierenden Benutzenden aufweist.

- Integrität bei CRUD-Operationen:

  Es wurden Tests für das Erstellen, Aktualisieren und Löschen von Benutzenden, Posts und Kommentaren durchgeführt, um sicherzustellen, dass die referenzielle Integrität in allen Fällen gewahrt bleibt.

- Datenabfragen:

  Die Performance der Datenbank bei häufigen Abfragen wurde evaluiert, insbesondere bei der Verwendung von Joins zwischen den Benutzenden-, Post- und Kommentar-Tabellen.

Query | Query History

```sql
-- 1. Alle Benutzer auflisten
SELECT username, full_name, email FROM users;

-- 2. Alle Posts mit Autorennamen
SELECT p.title, p.content, u.full_name AS author
FROM posts p
JOIN users u ON p.user_id = u.user_id;

-- 3. Anzahl der Posts pro Benutzer
SELECT u.username, COUNT(p.post_id) AS post_count
FROM users u
LEFT JOIN posts p ON u.user_id = p.user_id
GROUP BY u.username
ORDER BY post_count DESC;
```

Programmierung 2 – SQL-Skript für Abfragen

Die Tests wurden erfolgreich absolviert. Zu Beginn wurde die Implementierung mit «Golang» vorgenommen.

7.11 BACKEND-IMPLEMENTIERUNG

Bei umfangreicheren Anwendungen kann die Strukturierung des Projekts nach den Grundsätzen des **Domain-Driven Design (DDD)** vorteilhaft gewesen sein. Dieser Ansatz beinhaltet die Erstellung eines eigenen Verzeichnisses für jeden Bereich.

Abbildung 14 – Verzeichnisstruktur des Domain-Driven Designs

Die Entwicklung erfolgte unter Verwendung des Web-Frameworks «Echo». Dieses zeichnet sich durch eine hohe Leistungsfähigkeit und Vielseitigkeit aus und eignet sich insbesondere für die Erstellung von Web-Applikationen, die eine hohe Skalierung und eine optimale Leistung erfordern.

Die wesentlichen Merkmale des «Echos» lassen sich wie folgt zusammenfassen:

- Weiterleitung
- Middleware
- Kontextbasierte Anfrageverarbeitung
- Leistungsstarkes Template-Rendering
- Validierung und Bindung
- Erweiterbarkeit
- Gemeinschaft und Ökosystem

Die initialen Tests des Servers waren erfolgreich, sofern die Webseite "Hello, World!" angezeigt wurde.

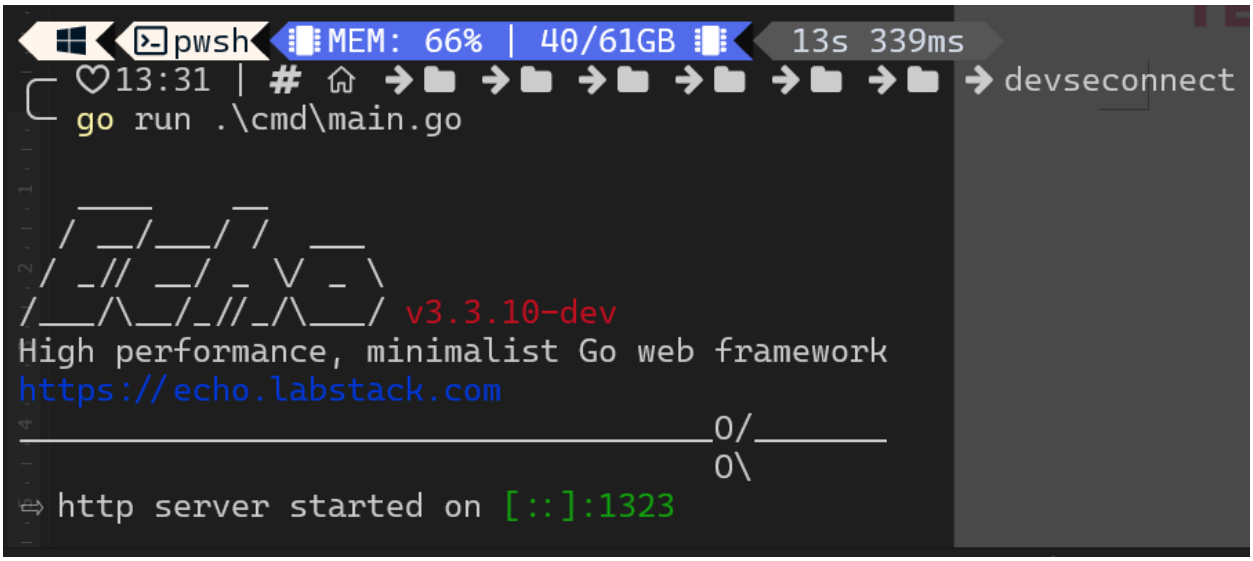


Abbildung 15 – GO gibt Echo-Print für Serverstart im Terminal aus

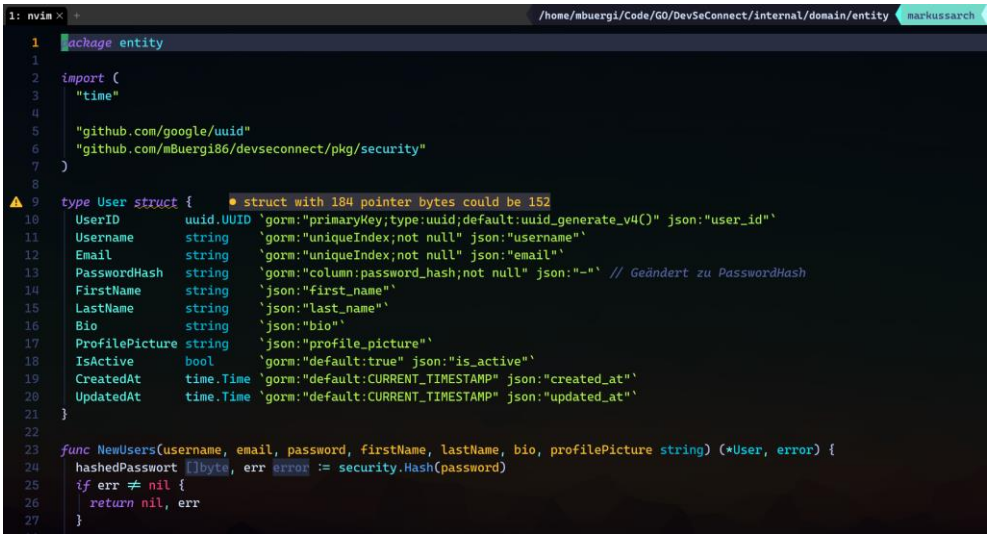
Die initialen Implementierungsschritte des Webservers konnten erfolgreich abgeschlossen werden, sodass dieser nun einwandfrei funktioniert. Im nächsten Schritt erfolgte die Umsetzung der Funktionalität für CRUD-Operationen (Create, Read, Update, Delete). In diesem Zusammenhang war es von entscheidender Bedeutung, Regelungen für die verschiedenen Anwendungsfälle festzulegen. Als Beispiel für eine derartige Regelung war die zeitliche Einschränkung, dass ein Beitrag maximal 15 Minuten nach seiner Erstellung bearbeitet werden durfte.

Die Erstellung eines Aktivitätsdiagramms empfahl sich, um die Abläufe und Prozesse der CRUD-Operationen übersichtlich und strukturiert darzustellen. Das Diagramm diente der anschaulichen Visualisierung des Prozessverlaufs der Benutzerinteraktionen – von der Erstellung bis hin zur Löschung von Beiträgen – und verdeutlichte mögliche Abhängigkeiten sowie zeitliche Beschränkungen.

Für jede Entität in der Datenbank folgte der Autor einer klar strukturierten Schichtarchitektur, die die Modularität und Wartbarkeit des Systems sicherstellte. Dieser Prozess bestand aus den folgenden Komponenten:

7.11.1 Modell

Das Modell repräsentierte die Datenstruktur jeder Entität. Es bildete die Datenbanktabelle in Form einer «Go-Struct» ab, wobei jede Spalte der Tabelle einem Field der Struct entspricht. Das Modell wurde zur Serialisierung und Deserialisierung von Daten verwendet und bildete das Kernstück der Kommunikation zwischen den anderen Schichten.



Programmierung 3 – Backend-Modell für Entität erstellen

7.11.1.1 Ein Beispiel für eine Struktur mit mehreren Modellen:

- User-Modell:
  - Repräsentiert eine:n Benutzenden mit Eigenschaften wie Username, Email, PasswordHash, FirstName, LastName und weiteren benutzerspezifischen Informationen.
- Post-Modell:
  - Repräsentiert einen Beitrag (z. B. in einem Blog) und enthält Felder wie Title, Content, PostID (eine Referenz auf die Benutzenden) und CreatedAt.
- Comment-Modell:
  - Repräsentiert einen Kommentar zu einem Beitrag und hat Felder wie Content, PostID (eine Referenz auf den Beitrag), UserID (eine Referenz auf die Benutzenden, die den Kommentar erstellt hatten) und CreatedAt.

7.11.1.2 Beispiel: Mehrere Modelle in «Go»

```go
// User model
type User struct {
    ID           uuid.UUID `gorm:"type:uuid;primary_key"`
    Username     string    `json:"username"`
    Email        string    `json:"email"`
    PasswordHash string    `json:"-"`
    Posts        []Post    `gorm:"foreignKey:AuthorID"`
    CreatedAt    time.Time `gorm:"default:CURRENT_TIMESTAMP"`
    UpdatedAt    time.Time `gorm:"default:CURRENT_TIMESTAMP"`
}

// Post model
type Post struct {
    ID        uuid.UUID `gorm:"type:uuid;primary_key"`
    Title     string    `json:"title"`
    Content   string    `json:"content"`
    UserID    uuid.UUID `json:"author_id"`
    Author    User      `gorm:"foreignKey:AuthorID"`
    Comments  []Comment `gorm:"foreignKey:PostID"`
    CreatedAt time.Time `gorm:"default:CURRENT_TIMESTAMP"`
    UpdatedAt time.Time `gorm:"default:CURRENT_TIMESTAMP"`
}

// Comment model
type Comment struct {
    ID        uuid.UUID `gorm:"type:uuid;primary_key"`
    Content   string    `json:"content"`
    PostID    uuid.UUID `json:"post_id"`
    Post      Post      `gorm:"foreignKey:PostID"`
    UserID    uuid.UUID `json:"user_id"`
```

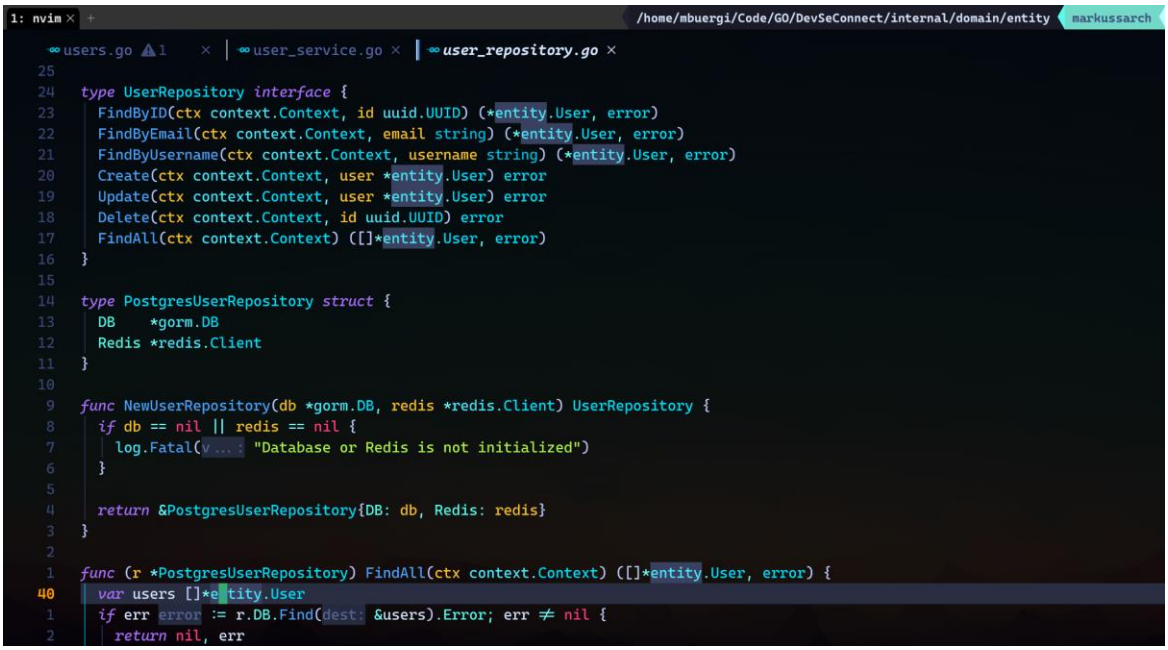
Programmierung 4 - Beispiel für ein Backend-Modell einer Entität

Die Beziehungen zwischen den Modellen sind wie folgt zu erschliessen:

- **User** und **Post** haben eine **1**: Ein:e Benutzer:in kann mehrere Beiträge haben (ein User hat viele Posts).
- **Post** und **Comment** haben ebenfalls eine **1**: Ein Beitrag kann mehrere Kommentare haben.
- **User** und **Comment** haben ebenfalls eine **1**: Ein:e Benutzer:in kann mehrere Kommentare schreiben.

7.11.2 Repository

Das Repository stellte die Datenzugriffsschicht dar und war für die Interaktion mit der Datenbank verantwortlich. Hier wurden CRUD-Operationen (Create, Read, Update, Delete) definiert, um auf die spezifische Entität zuzugreifen und Daten zu verwalten. Es stellte sicher, dass alle Datenbankoperationen isoliert und testbar waren.



Programmierung 5 – Backend-Implementierung für das Repository

7.11.2.1 Beschreibung der Komponenten des Repository-Musters

- **Repository Interface:**
  - Das Repository Interface definiert eine Reihe von Methoden zur Interaktion mit Daten in der Anwendung. Diese Methoden sind abstrahiert, um die spezifischen Datenmodelle (z.B. Benutzende, Posts, Kommentare) zu verbergen und die Datenbankoperationen allgemeingültig zu machen. Zu den typischen Methoden gehören:
    - FindByID, FindByEmail, FindByUsername: Methoden, die das Abrufen von Daten auf Basis verschiedener Kriterien wie einer eindeutigen ID, E-Mail oder einen Benutzernamen ermöglichen.
    - Create, Update, Delete: Methoden für das Erstellen, Aktualisieren und Löschen von Datensätzen in der Datenbank.
    - FindAll: Gibt eine Liste aller vorhandenen Datensätze (z.B. alle Benutzer, Posts oder Kommentare) zurück.

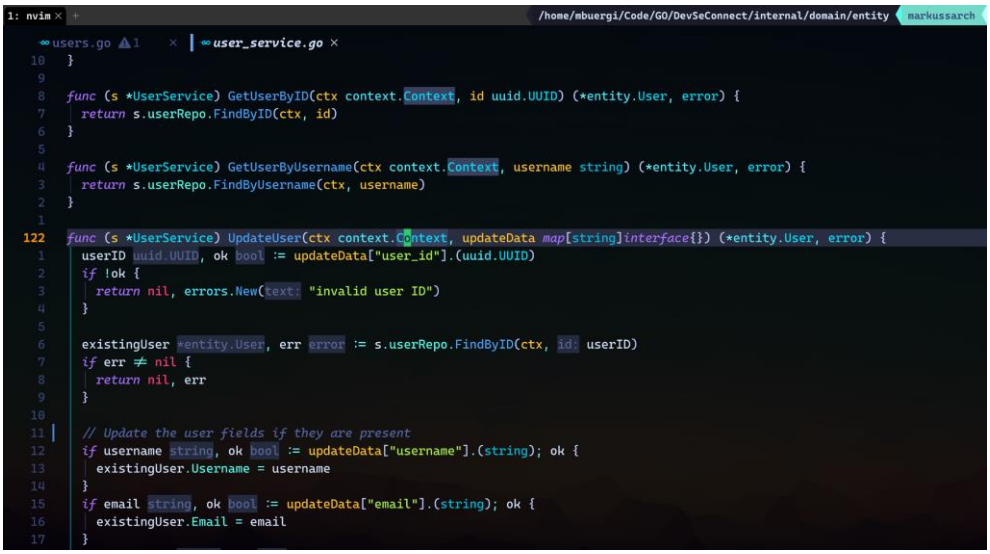
- **PostgresRepository Struct:**
  - Das PostgresRepository enthält die Datenbankverbindung zur PostgreSQL-Datenbank (durch die Verwendung des gorm.DB-Objekts) sowie einen Redis-Client. Diese Struktur ermöglicht die Nutzung von PostgreSQL als primäre Datenbank für die dauerhafte Datenspeicherung und Redis für schnelles Caching oder schnellen Zugriff auf häufig genutzte Informationen.
  - Redis kann in diesem Kontext verwendet werden, um häufig abgerufene Daten wie Benutzerprofile oder Beitragsinformationen zwischenzuspeichern und somit die Last auf die relationale Datenbank zu reduzieren.
- **NewRepository Funktion:**
  - Die NewRepository-Funktion initialisiert ein neues PostgresRepository mit einer Verbindung zu einer PostgreSQL-Datenbank und einem Redis-Client. Falls eine der beiden Verbindungen (Datenbank oder Redis) nicht ordnungsgemäss initialisiert werden kann, wird ein Fehler geloggt und das Programm möglicherweise beendet, um sicherzustellen, dass keine Operationen auf nicht verbundenen Diensten durchgeführt werden.
- **FindAll Methode:**
  - Die FindAll-Methode durchsucht die PostgreSQL-Datenbank nach allen Datensätzen (z.B. Benutzende, Posts, Kommentare). Sollte beim Abrufen der Daten ein Fehler auftreten, wird dieser an den Aufrufer zurückgegeben. Falls kein Fehler auftritt, wird die Liste der Datensätze zurückgegeben, die von der Datenbank abgerufen wurden. Dies ermöglicht eine flexible Handhabung und Verarbeitung grosser Datenmengen.

7.11.2.2 Zusammenfassung des Repository-Musters:

Das Repository-Muster stellt eine abstrakte Schnittstelle zur Verfügung, um verschiedene Datenmodelle zu verwalten. Es kapselt die Datenzugriffsschicht der Anwendung, indem es mit einer Kombination aus PostgreSQL (für langfristige, persistente Speicherung) und Redis (für schnelles Caching) arbeitet. Diese Struktur ermöglichte eine effiziente und flexible Verwaltung von Daten und unterstützte die Trennung der Geschäftslogik von der zugrunde liegenden Datenbanktechnologie. Dies führte zu einer besseren Wartbarkeit, Skalierbarkeit und Testbarkeit der Anwendung.

7.11.3 Service

Die Verantwortung für die Umsetzung der Geschäftslogik oblag dem Service. Der Service gewährleistete die korrekte und effiziente Ausführung von CRUD-Operationen (Create, Read, Update, Delete) für unterschiedliche Entitäten, darunter Benutzende, Beiträge und Kommentare. Ansonsten übernahm der Service die Kommunikation mit dem Repository, führte Validierungen durch und integrierte ein asynchrones Nachrichtensystem (z. B. RabbitMQ), um Ereignisse wie die Registrierung neuer Benutzenden zu verarbeiten.



Programmierung 6 – Backend-Implementierung des Service

7.11.3.1 Beschreibung der Geschäftslogik im Service:

- **Registrierung**
  - Der Service übergibt die Benutzerdaten an das Repository, das die neuen Benutzenden in der Datenbank speichert.
  - Nach erfolgreicher Registrierung wird ein Ereignis vom Typ «user_registered» an ein Nachrichtensystem gesendet, damit andere Dienste darüber informiert werden.
  - Fehler bei der Veröffentlichung des Ereignisses beeinträchtigen nicht die Registrierung selbst, werden jedoch zur späteren Überprüfung geloggt.

- **Login**
  - Der Service authentifiziert den Benutzer durch Abgleich von Benutzername und Passwort. Das Passwort wird mit dem gehashten Passwort in der Datenbank verglichen.
  - Bei einer erfolgreichen Authentifizierung wird ein JWT (JSON Web Token) generiert, das zur Authentifizierung des Benutzenden bei nachfolgenden Anfragen dient.
  - Das JWT enthält Informationen wie Benutzer-ID und Ablaufzeit, wodurch die Authentifizierung für eine festgelegte Zeit gültig bleibt.
- **Verwaltung von Beiträgen und Kommentaren**
  - **Erstellen eines Beitrags**: Der Service validiert die Eingaben und übergibt sie an das Post-Repository zur Speicherung.
  - **Kommentarerstellung**: Kommentare werden validiert und dem entsprechenden Beitrag zugeordnet.
  - **Abrufen, Aktualisieren und Löschen**: Der Service koordiniert das Abrufen, Aktualisieren und Löschen von Beiträgen und Kommentaren. Dabei werden die Benutzerrechte überprüft, um unbefugte Aktionen zu verhindern.

7.11.3.2 Service-Architektur

Repository-Kommunikation

Der Service verwendet das Repository, um Datenbankoperationen durchzuführen. Diese Trennung sorgte dafür, dass die Geschäftslogik unabhängig von der Datenbank bleibt. Änderungen in der Datenbank oder im Repository beeinflussen die Geschäftslogik nicht direkt.

Validierung und Fehlerbehandlung

Der Service prüft alle Daten vor ihrer Verarbeitung. Passwörter werden unter anderem vor der Speicherung validiert und gehasht, und nur gültige Daten werden akzeptiert. Bei fehlerhaften Eingaben gibt der Service eine klare Fehlermeldung an den API-Handler zurück, damit die Benutzenden informiert werden.

JWT-Authentifizierung

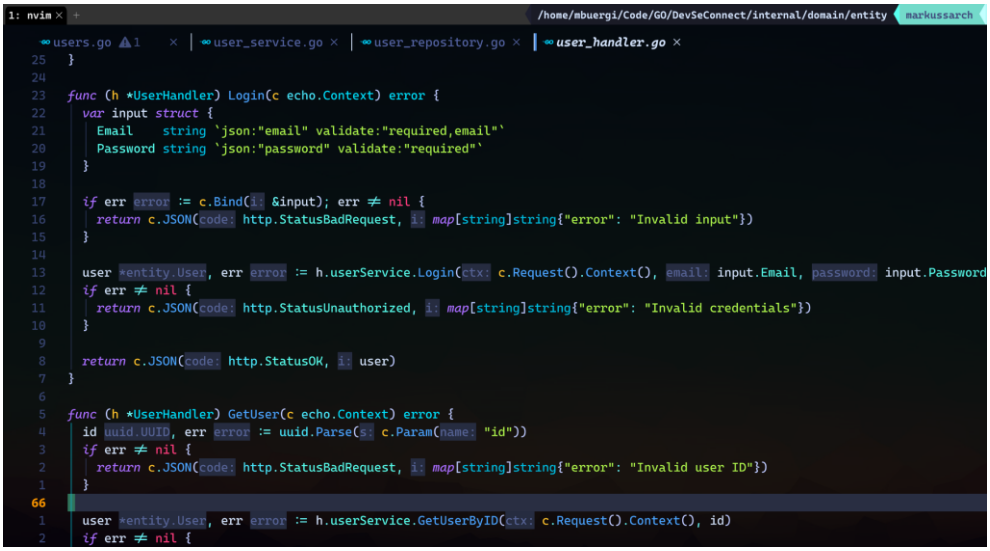
Der Service stellt sicher, dass alle geschützten Endpunkte nur von authentifizierten Benutzern aufgerufen werden können. Dazu wird bei jeder Anfrage ein JWT überprüft. Bei einem erfolgreichen Login wird das JWT vom Service erstellt und enthält wichtige Informationen wie Benutzer-ID und Berechtigungen. Dies ermöglicht eine sichere und skalierbare Authentifizierung.

Ereignisverarbeitung

Nach bestimmten Aktionen, wie der Erstellung eines Beitrags oder der Registrierung der Benutzenden, veröffentlicht der Service ein Ereignis an ein Nachrichtensystem wie RabbitMQ. Diese asynchrone Verarbeitung ermöglicht es anderen Diensten, auf Ereignisse zu reagieren, ohne dass die direkte Kopplung zwischen Systemen erforderlich ist.

7.11.4 Handler

Der Handler ist ein zentrales Element der API-Schicht in einer Webanwendung. Er fungiert als Schnittstelle zwischen den HTTP-Anfragen, die vom Router verarbeitet werden, und den Geschäftslogikdiensten, die die Kernfunktionalitäten der Anwendung bereitstellen.



Programmierung 7 - Backend-Implementierung des Handlers

7.11.4.1 Hauptaufgaben des Handlers:

- **Entgegennahme von HTTP-Anfragen**:

  Der Handler empfängt die HTTP-Anfragen, die durch den Router weitergeleitet werden. Dabei wird die Anfrage (z.B. GET, POST, PUT, DELETE) entsprechend ihrer Route an den richtigen Handler übergeben. Jede Route wird also einem spezifischen Handler zugeordnet.

- **Datenvalidierung und Parsing**:

  Der Handler extrahiert und validiert die in der HTTP-Anfrage enthaltenen Daten. Dies beinhaltet:

  - Das Parsen von JSON-Payloads in strukturierte Objekte (z.B. für die Erstellung oder Aktualisierung von Datensätzen).

  - Die Überprüfung auf korrekte Datentypen, fehlende Parameter oder falsche Werte. Bei ungültigen Daten gibt der Handler eine entsprechende Fehlermeldung zurück (z.B. HTTP 400 Bad Request).

- **Aufruf der Geschäftslogik (Services)**:

  Nach erfolgreicher Validierung der Eingaben übergibt der Handler die Daten an den entsprechenden Service. Der Service führt dann die eigentlichen Geschäftsoperationen durch, wie zum Beispiel:

  - Erstellung, Abruf, Aktualisierung oder Löschung von Daten (CRUD-Operationen).
  - Verarbeitung von spezifischen Geschäftsregeln.

Hier ist der Handler entscheidend für die Trennung der Zuständigkeiten. Der Handler selbst enthält keine Geschäftslogik, sondern delegiert diese vollständig an den Service.

- **Handler als Zuständigkeits-Trenner:**

  Der Handler stellte eine entscheidende Komponente für die Trennung der Zuständigkeiten dar. Er enthielt keine Geschäftslogik, sondern delegierte diese vollständig an den Service.

- **Verarbeitung der Service-Antwort**:

  Nachdem der Service die Anfrage bearbeitet hat (z.B. durch den Zugriff auf die Datenbank), erhält der Handler eine Antwort (z.B. den erstellten Datensatz oder eine Erfolgsmeldung). Der Handler formatiert diese Antwort und sendet sie als HTTP-Antwort (z.B. JSON-Format) an den Client zurück.

- **Fehlerbehandlung**:

  Der Handler ist auch dafür zuständig, Fehler, die während der Datenverarbeitung auftreten, zu erfassen und dem Client in geeigneter Weise mitzuteilen. Dies umfasst Fehler wie:

  - **Ungültige Eingaben** (z.B. fehlende Pflichtfelder).
  - **Datenbankfehler oder fehlende Berechtigungen** (HTTP 403, 404).
  - **Serverfehler** (HTTP 500).

Eine saubere Fehlerbehandlung ist entscheidend, um dem Client klare Rückmeldungen zu geben und die Integrität des Systems zu gewährleisten.

7.11.4.2 Vorteile einer klaren Handler-Architektur:

- **Trennung der Verantwortlichkeiten**:

  Der Handler ist nur für die Entgegennahme und Weiterleitung von Anfragen zuständig. Er enthält keine Geschäftslogik, was die Wartbarkeit und Testbarkeit des Codes deutlich verbessert.

- **Skalierbarkeit**:

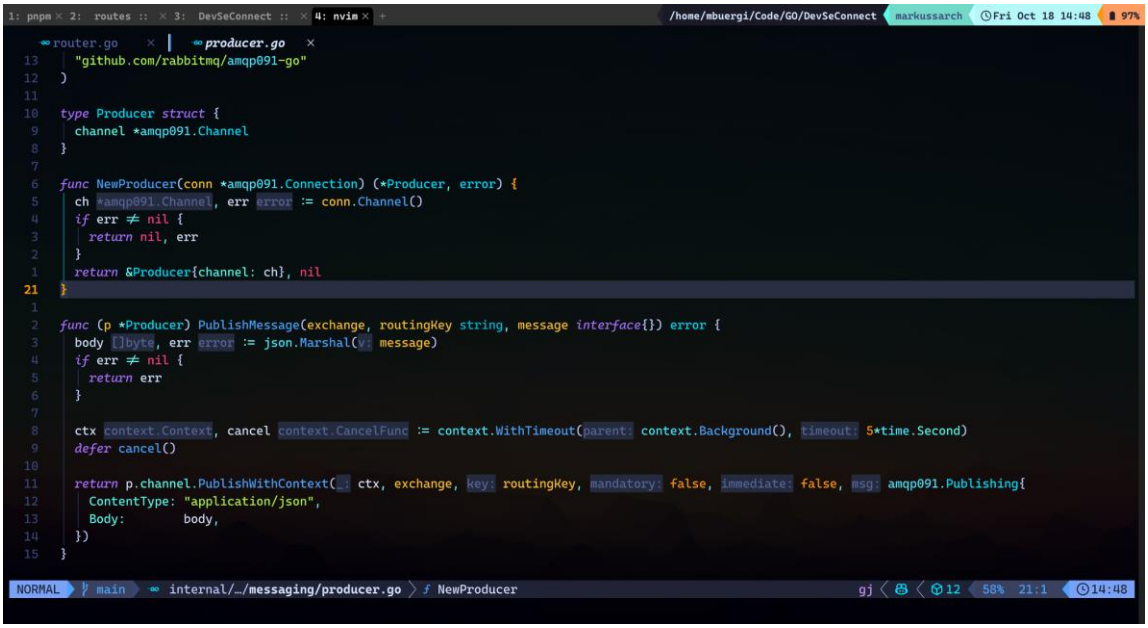
  Durch die klare Aufteilung zwischen Handler und Service kann die API leicht erweitert werden. Neue Routen und Endpunkte können durch Hinzufügen weiterer Handler leicht implementiert werden, ohne den Kern der Geschäftslogik zu beeinflussen.

- **Sicherheit**:

  Handler können Sicherheitsvorkehrungen wie Authentifizierung und Autorisierung umsetzen, bevor die eigentlichen Datenoperationen im Service aufgerufen werden.

7.11.5 RabbitMQ Producer

Der Producer ist die Komponente, die Nachrichten in das RabbitMQ des Nachrichtensystems sendet. Er stellt die Verbindung zu einem RabbitMQ Channel her und verwendet diesen, um Nachrichten an eine definierte Exchange mit einem bestimmten Routing Key zu senden.



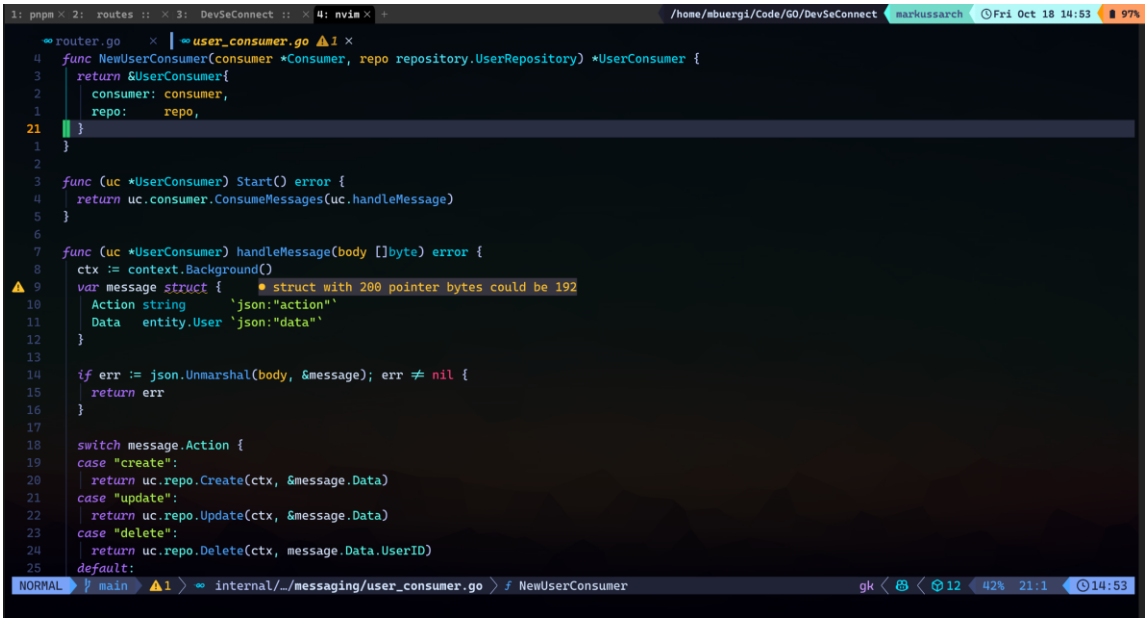
Programmierung 8 – Backend-Implementierung des Producers für RabbitMQ

7.11.5.1 Beschreibung der Komponenten:

- **NewProducer:**
  - Diese Funktion erstellt einen neuen Producer, indem sie eine Verbindung zu RabbitMQ nutzt, um einen Channel zu öffnen. Der Channel ist notwendig, um Nachrichten zu publizieren. Falls es Probleme beim Öffnen des Channels gibt, wird ein Fehler zurückgegeben.
- **PublishMessage:**
  - Diese Methode sendet eine Nachricht an RabbitMQ. Sie wandelt die Nachricht in ein JSON-Format um und veröffentlicht sie unter Verwendung einer spezifischen Exchange und Routing Keys.
  - Es wird ein Kontext mit Timeout erstellt, um sicherzustellen, dass die Nachrichtensendung innerhalb einer bestimmten Zeit abgeschlossen ist.

7.11.6 RabbitMQ Consumer

Der RabbitMQ Consumer ermöglicht eine asynchrone Verarbeitung von Nachrichten, indem er die Nachrichten verarbeitet, die von verschiedenen Services oder Komponenten gesendet werden. Statt sich auf Benutzerdaten zu beschränken, kann der Consumer jede Art von Daten empfangen und verarbeiten. Dies umfasst das Erstellen, Aktualisieren oder Löschen von Datensätzen basierend auf dem Nachrichtentyp und der Geschäftslogik. Der Consumer ist also nicht nur eine Komponente für Benutzeroperationen, sondern kann für jedes Datenmodell angepasst werden, das in der Anwendung verwendet wird.



Programmierung 9 - Backend-Implementierung des Consumers für User in RabbitMQ

7.11.6.1 Beschreibung der Komponenten eines RabbitMQ-Consumers:

- **NewConsumer:**
  - Diese Funktion erstellt eine neue Instanz des Consumers, indem sie eine Verbindung zu RabbitMQ sowie eine Referenz auf ein Repository (oder einen anderen Service) entgegennimmt. Das Repository ist verantwortlich für die Verarbeitung oder Speicherung der Daten, die über die Nachrichten übermittelt werden.
  - Ziel: Den Consumer so zu konfigurieren, dass er Nachrichten aus einer bestimmten Warteschlange (Queue) konsumiert und mit der nötigen Infrastruktur (wie Datenbank oder Services) verbunden ist.
- **Start:**
  - Die Start-Methode initiiert den eigentlichen Konsumprozess. Hier wird der Consumer angewiesen, kontinuierlich Nachrichten aus der Warteschlange zu lesen.
  - Der Aufruf einer Methode wie ConsumeMessages enthält die Logik für das Verarbeiten der Nachrichten, indem der Consumer in eine Schleife eintritt und eingehende Nachrichten entgegennimmt.
  - Ziel: Den Prozess starten, damit der Consumer auf neue Nachrichten wartet und diese verarbeiten kann.
- **handleMessage:**
  - Die handleMessage Methode ist der Hauptprozessor jeder empfangenen Nachricht. Diese Funktion nimmt die empfangene Nachricht (im JSON oder einem anderen Format) und deserialisiert sie in eine Struktur, die den notwendigen Inhalt (wie Aktionstypen und Daten) enthält.
  - Der Inhalt der Nachricht bestimmt, welche Operation ausgeführt wird. Typischerweise gibt es mehrere Aktionen wie:
    1. create: Erstellen eines neuen Eintrags (z. B. Benutzende, Post, Kommentar, etc.).
    2. update: Aktualisieren eines bestehenden Eintrags.
    3. delete: Löschen eines Eintrags.

- Ziel: Die Nachricht interpretieren und auf Grundlage ihres Inhalts die entsprechenden Operationen auf dem Repository oder dem entsprechenden Service durchzuführen.

7.11.6.2 Generische Beschreibung eines RabbitMQ Consumers:

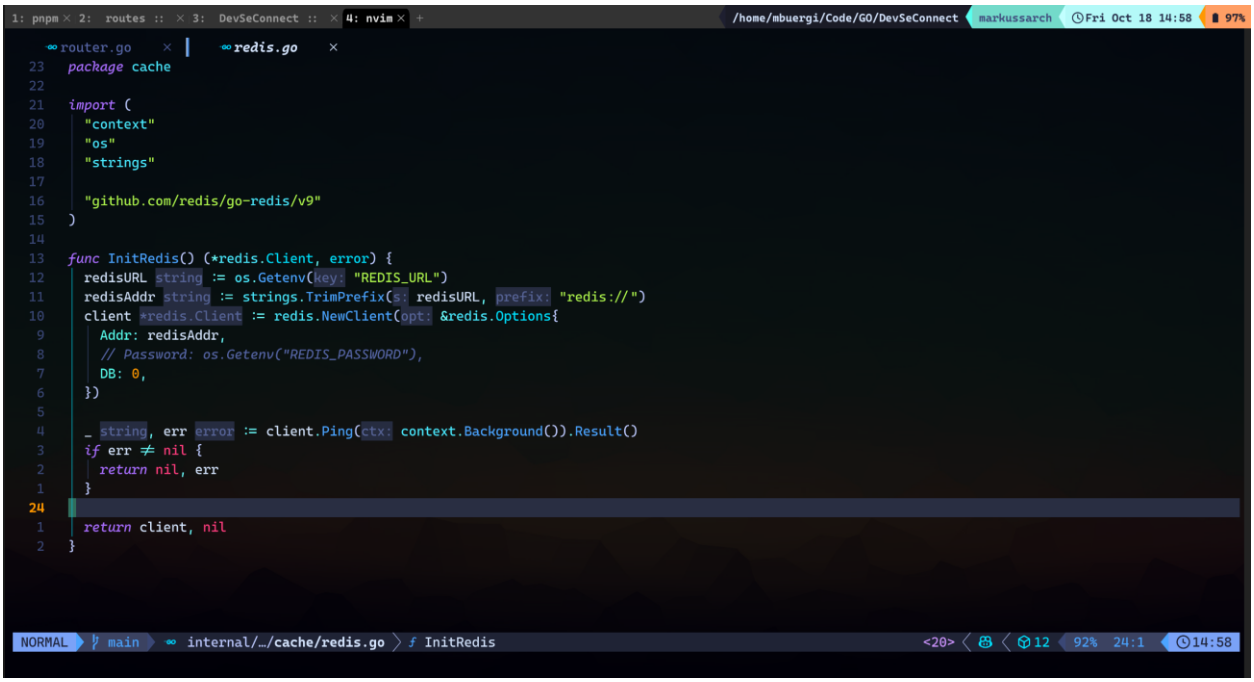
Ein RabbitMQ Consumer dient als Schnittstelle zwischen einem Messaging-System und der Geschäftslogik. Er verarbeitet Nachrichten, die aus der RabbitMQ Warteschlange konsumiert werden, und führt entsprechende Aktionen aus. Diese Nachrichten können verschiedene Datenarten enthalten, wie z. B. Benutzende, Posts, Kommentare, Transaktionen oder andere Domänenobjekte.

Der Consumer:

- Entkoppelt die Datenverarbeitung von der Logik, indem er Nachrichten in asynchronen Prozessen empfängt.
- Skaliert die Verarbeitung von Daten, indem er mehrere Nachrichten parallel bearbeitet.
- Verarbeitet jede Nachricht auf Basis einer vordefinierten Aktion, die in der Nachricht enthalten ist, und delegiert die Geschäftslogik an ein entsprechendes Repository oder eine Service-Schicht.

7.11.7 Redis

Der Codeausschnitt zeigt die Initialisierung eines «Redis» Clients in «Go» mithilfe der «Redis» Bibliothek «go-redis».



Programmierung 10 – Backend-Implementierung für Redis

7.11.7.1 Beschreibung des Redis:

- **Lesen der Redis URL:**
  - Die redisURL wird aus den Umgebungsvariablen mit dem Schlüssel REDIS_URL geladen. Diese URL enthält die Verbindungsinformationen für den Redis Server.
  - Die redisAddr wird durch das Entfernen des redis://-Präfixes aus der URL bereinigt, damit nur die eigentliche Adresse verwendet wird.
- **Erstellen des Redis Clients:**
  - Ein neuer Redis Client wird mit den Verbindungsoptionen erstellt, die die bereinigte Adresse (redisAddr) und optional das Passwort (das hier auskommentiert ist) enthalten.
  - Die Redis Datenbank wird durch DB: 0 festgelegt, was die Standard-Datenbank von Redis angibt (Redis unterstützt mehrere nummerierte Datenbanken).

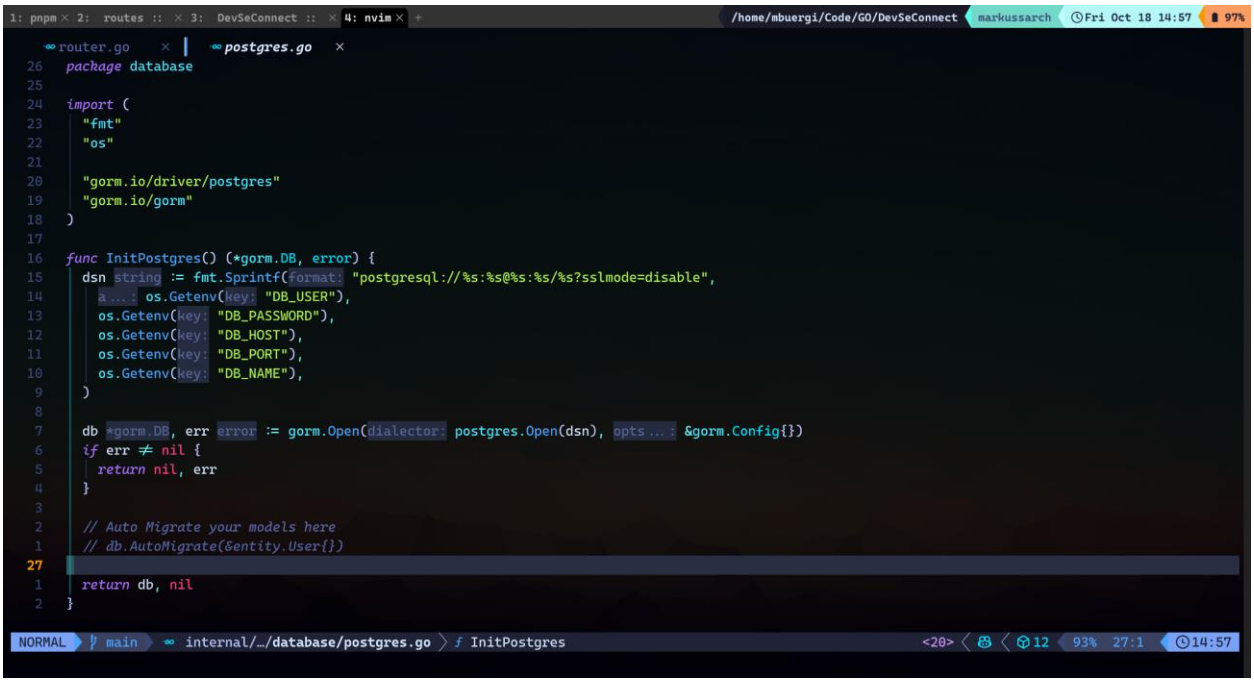
- **Verbindung testen:**
  - Die Ping()-Methode wird verwendet, um sicherzustellen, dass die Verbindung zum Redis-Server erfolgreich hergestellt wurde. Falls ein Fehler auftritt, wird dieser zurückgegeben.
- **Rückgabe des Redis Clients:**
  - Wenn die Verbindung erfolgreich ist, wird der Redis Client zurückgegeben. Dieser Client wird verwendet, um Daten in Redis zu speichern, abzurufen oder zu manipulieren.

Zusammenfassung:

Dieser Code zeigt die Einrichtung einer Verbindung in «Go» zu einem «Redis» Server. Der «Redis» Client wird mit den Verbindungsinformationen erstellt, und es wird ein Test durchgeführt, um sicherzustellen, dass die Verbindung funktioniert. «Redis» wird häufig als Cache oder schnelles Datenspeichersystem verwendet.

7.11.8 PostgreSQL-Datenbankverbindung

Der Codeausschnitt zeigt die Initialisierung einer PostgreSQL-Datenbankverbindung in einer «Go-Anwendung» unter Verwendung des ORM Frameworks «GORM».



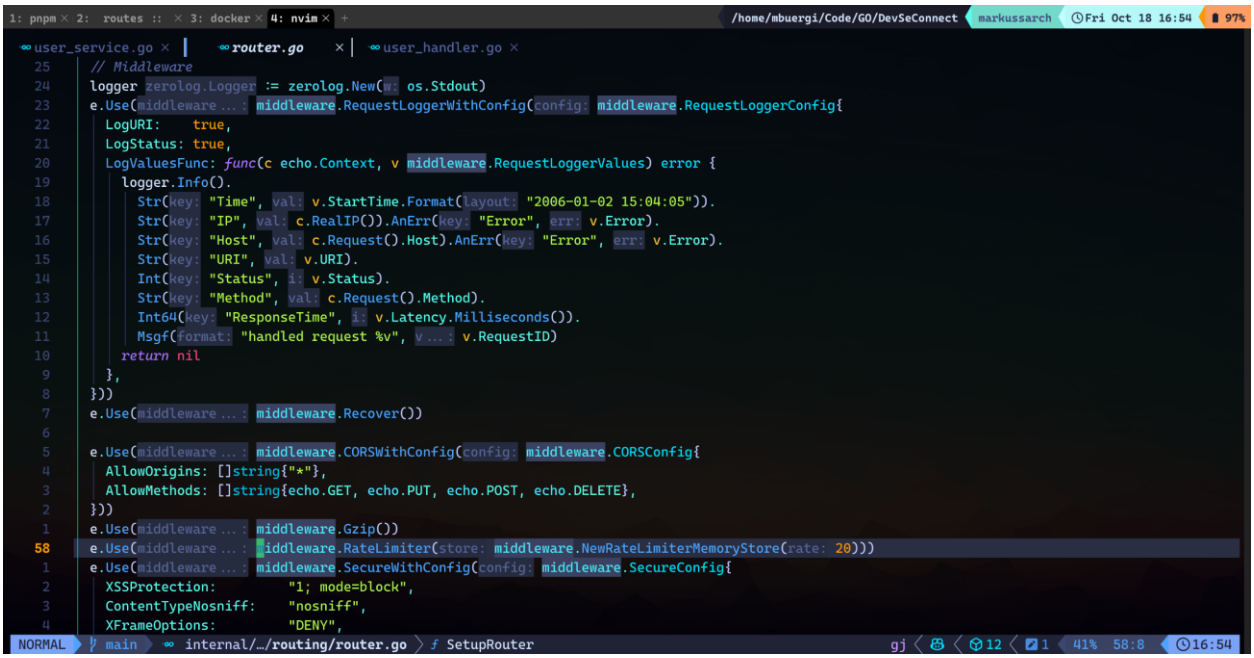
Programmierung 11 – Backend-Implementierung für die PostgreSQL-Datenbankanbindung

7.11.8.1 Beschreibung der PostgreSQL-Datenbankverbindung:

- **Datenbankverbindungszeichenfolge (DSN**):
  - Die Verbindung zu PostgreSQL wird durch die Erstellung einer DSN (Data Source Name) hergestellt, die aus Umgebungsvariablen (DB_USER, DB_PASSWORD, DB_HOST, DB_PORT, DB_NAME) generiert wird. Diese Umgebungsvariablen enthalten die Zugangsdaten und Konfigurationsdetails für die PostgreSQL-Datenbank.
  - sslmode=disable gibt an, dass keine SSL-Verschlüsselung für die Verbindung verwendet wird.
- **Öffnen der Datenbankverbindung:**
  - Die Methode gorm.Open() wird verwendet, um die Verbindung zur Datenbank herzustellen. Falls ein Fehler auftritt, wird dieser zurückgegeben.
- **Auto Migrate (optional):**
  - Die auskommentierte Zeile db.AutoMigrate(&entity.User{}) weist darauf hin, dass «GORM» die Tabellenstrukturen für das Modell User automatisch migrieren könnte, um sicherzustellen, dass die Datenbank das korrekte Schema für das Modell verwendet.
- **Rückgabe der Datenbankverbindung:**
  - Wenn die Verbindung erfolgreich hergestellt wurde, gibt die Funktion eine Referenz auf die gorm.DB-Instanz zurück.

7.11.9 Middleware

Die Middleware stellt eine wesentliche Komponente innerhalb der Softwarearchitektur dar, welche die Verarbeitung von Anfragen sowie die Anwendung zusätzlicher Sicherheits- oder Optimierungsmassnahmen übernimmt. Im Folgenden erfolgt eine Beschreibung der verwendeten Middleware-Komponenten sowie eine Darlegung ihrer Vorteile.



Programmierung 12 – Backend-Implementierung des Routers

7.11.9.1 Middleware-Komponente: RequestLogger

- Diese Middleware zeichnet wichtige Informationen jeder eingehenden Anfrage auf, darunter:
    1. Zeitstempel, IP-Adresse, Host, URI, Statuscode der Antwort, HTTP-Methode und die Antwortzeit.
- Diese Informationen sind hilfreich, um Anfragen zu überwachen, Fehler zu erkennen und die Performance der Anwendung zu analysieren.

```
// Middleware
logger zerolog.Logger := zerolog.New(w: os.Stdout)
e.Use(middleware ... : middleware.RequestLoggerWithConfig(config: middleware.RequestLoggerConfig{
  LogURI:    true,
  LogStatus: true,
  LogValuesFunc: func(c echo.Context, v middleware.RequestLoggerValues) error {
    logger.Info().
      Str(key: "Time", val: v.StartTime.Format(layout: "2006-01-02 15:04:05")).
      Str(key: "IP", val: c.RealIP()).AnErr(key: "Error", err: v.Error).
      Str(key: "Host", val: c.Request().Host).AnErr(key: "Error", err: v.Error).
      Str(key: "URI", val: v.URI).
      Int(key: "Status", i: v.Status).
      Str(key: "Method", val: c.Request().Method).
      Int64(key: "ResponseTime", i: v.Latency.Milliseconds()).
      Msgf(format: "handled request %v", v ... : v.RequestID)
    return nil
  },
}))
```

Programmierung 13 – Backend-Implementierung der Middleware für Logging mit „zerolog“

- **Nutzen**: Detaillierte Logging-Daten, die zur Fehlerbehebung und Analyse von Performance-Engpässen verwendet werden können.

7.11.9.2 Middleware-Komponente: Recover Middleware

- Diese Middleware fängt Panics im Code ab und sorgt dafür, dass die Anwendung nicht abstürzt. Sie liefert stattdessen eine sinnvolle Antwort, wenn ein interner Fehler auftritt.
- **Nutzen**: Verhindert den Absturz der Anwendung und sorgt für eine stabilere Laufzeitumgebung, selbst bei unerwarteten Fehlern.

7.11.9.3 Middleware-Komponente: CORS (Cross-Origin Resource Sharing) mit Config

- Diese Middleware regelt den Zugriff auf die API von anderen Domänen aus. Hier wird beispielsweise festgelegt, welche HTTP-Methoden und Ursprünge erlaubt sind.

```
e.Use(middleware ... : middleware.CORSWithConfig(config: middleware.CORSConfig{
  AllowOrigins: []string{"*"},
  AllowMethods: []string{echo.GET, echo.PUT, echo.POST, echo.DELETE},
}))
```

Programmierung 14 – Backend-Implementierung der Middleware für CORS zur Steuerung von Cross-Origin-Anfragen

- **Nutzen**: Erlaubt den sicheren Zugriff auf die API von verschiedenen Quellen aus und schützt vor unbefugtem Zugriff durch eingeschränkte Cross-Origin-Requests.

7.11.9.4 Middleware-Komponente: Gzip

- Diese Middleware komprimiert die HTTP-Antworten, bevor sie an den Client gesendet werden, um Bandbreite zu sparen und die Ladezeiten zu verbessern.
- **Nutzen**: Reduziert die Grösse der übertragenen Daten und verbessert dadurch die Performance, insbesondere bei langsamen Netzwerken.

7.11.9.5 Middleware-Komponente: Rate Limiter

- Diese Middleware begrenzt die Anzahl der Anfragen, die ein Client in einer bestimmten Zeit senden darf. Hier ist ein Rate-Limit von 20 Anfragen konfiguriert.

```
e.Use(middleware ... : middleware.RateLimiter(store: middleware.NewRateLimiterMemoryStore(rate: 20)))
```

Programmierung 15 – Backend-Implementierung der Middleware für Rate Limiting mit Memory Store

- Nutzen: Schützt die API vor DDoS-Angriffen und übermässigen Anfragen, indem die Rate der Anfragen pro Client eingeschränkt wird.

7.11.9.6 Middleware-Komponente: Secure Middleware

- Diese Middleware setzt Sicherheits-Header wie XSS-Schutz (Cross-Site Scripting Protection), Content-Type-Options, HSTS (HTTP Strict Transport Security), und mehr.

```
1  e.Use(middleware ... : middleware.SecureWithConfig(config: middleware.SecureConfig{
2    XSSProtection:         "1; mode=block",
3    ContentTypeNosniff:    "nosniff",
4    XFrameOptions:         "DENY",
5    HSTSMaxAge:            3600,
6    HSTSExcludeSubdomains: true,
7    ContentSecurityPolicy: "default-src 'self'",
8  }))
```

Programmierung 16 – Backend-Implementierung der Middleware für Sicherheitsmassnahmen in verschiedenen Bereichen

- **Nutzen**: Stellt sicher, dass die API gegen verschiedene Angriffe wie XSS, Clickjacking und man-in-the-middle-Angriffe geschützt ist.

Diese erwähnten Middleware-Komponenten sind essenziell für die Sicherstellung von Logging, Sicherheit, Performance und Stabilität in der Anwendung.

7.11.10 Tests und Validierungen des Backends

In diesem Abschnitt wird der Testprozess des Backends beschrieben, um sicherzustellen, dass die Anwendung korrekt funktioniert und den Anforderungen entspricht. Die Tests umfassen «API-Tests» mit «Postman» sowie die Überwachung der Backend-Dienste mit «Grafana».

7.11.10.1 «API-Tests» mit Postman Dashboard

Die API-Tests spielen eine zentrale Rolle bei der Sicherstellung der Funktionalität der Schnittstellen des Backends. Mithilfe des «Postman» Dashboards werden die CRUD-Operationen (GET, POST, PUT, DELETE) getestet, ohne dass ein kostenpflichtiger Account oder Newman erforderlich ist.

Schritte zur Durchführung von API-Tests:

1. Erstellen der API-Anfragen:

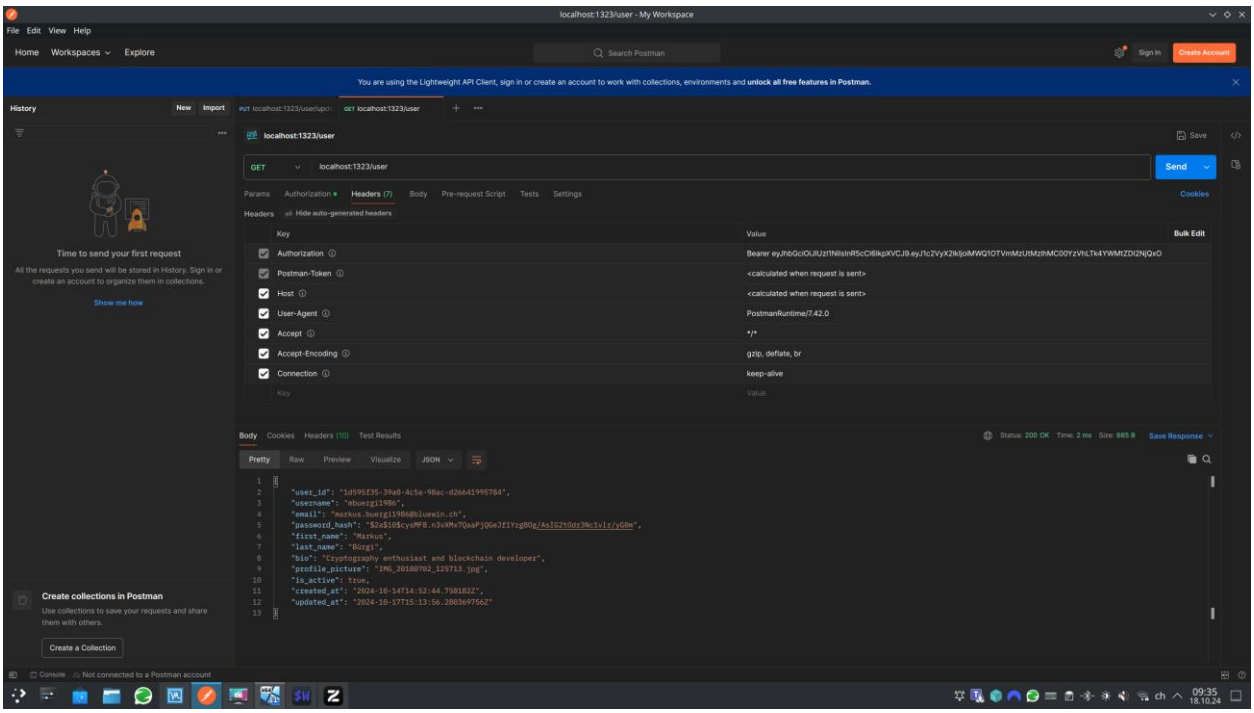


Abbildung 16 – Testen von API-Endpunkten mit Postman

Die wichtigen Schritte zur Erstellung von API-Anfragen in diesem Kontext wären:

1. **Manuelles Erstellen der Anfragen**: In Postman wird der Endpunkt für die API aufgerufen, in diesem Fall `localhost:1323/user`. Dies kann durch verschiedene Methoden wie GET, POST, PUT oder DELETE erfolgen.

2. **Konfiguration der Header und Parameter**: In der Anfrage sind verschiedene Header zu sehen, darunter Authorization, Host, User-Agent usw. Diese werden konfiguriert, um die Anfrage zu authentifizieren und die benötigten Informationen zu übermitteln.
3. **Überprüfung der Antwort (Response)**: Die Antwort der API-Anfrage wird in einem JSON-Format angezeigt, das Informationen über den Benutzer enthält, wie z.B. `user_id`, `username`, `email`, `password_hash`, `first_name`, `last_name`, und andere Attribute wie `created_at` und `updated_at`.
4. **Ergebnisse manuell überprüfen**: Sobald die Anfrage gesendet wurde, überprüft der Tester die Antwort (`Status 200 OK`, sowie den Inhalt der Antwort), um sicherzustellen, dass die API ordnungsgemäss funktioniert.

2. Validierung der Antworten:

Es wird überprüft, ob die HTTP-Statuscodes und die Rückmeldungen der API korrekt sind. Dabei werden sowohl Erfolgs- als auch Fehlerfälle abgedeckt.

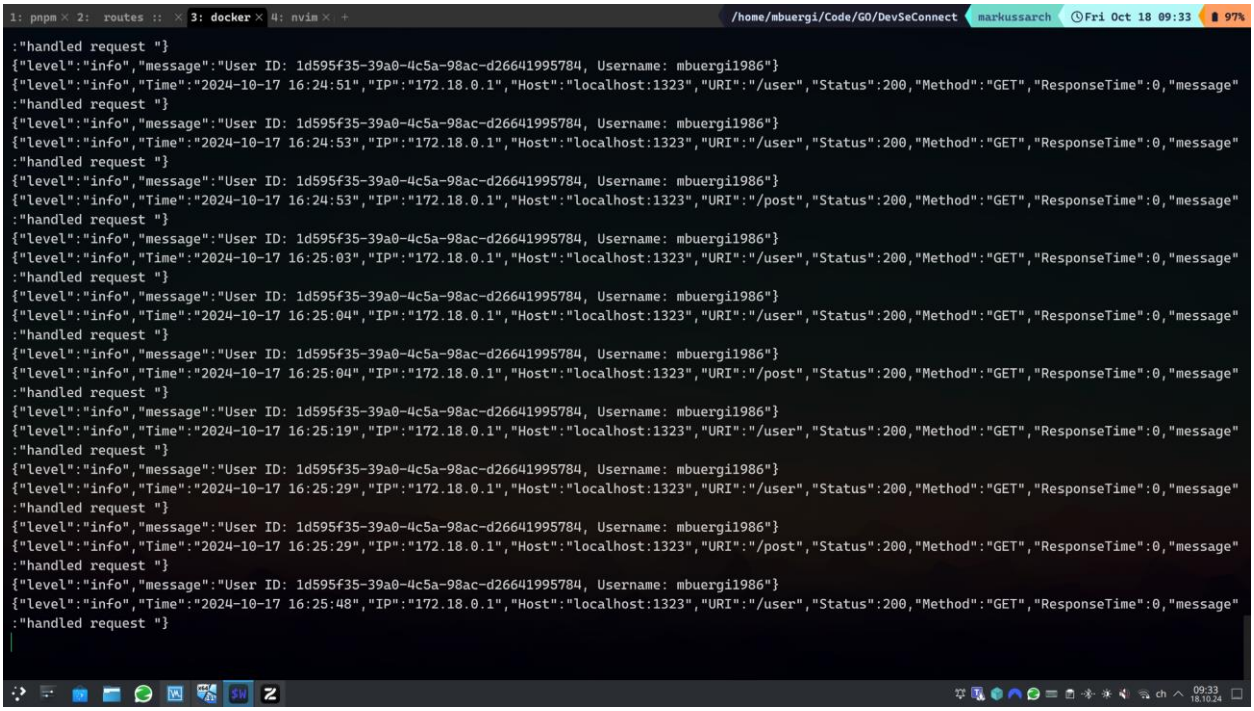


Abbildung 17 – Anzeige von Fehlermeldungen und Erfolgsnachrichten im Terminal

Das Bild zeigt eine Konsolenausgabe, die die Protokollierung mehrerer API-Anfragen dokumentiert. Die Protokolle enthalten Informationen über die Anfragen an den Endpunkt `/user` und `/post` unter `localhost:1323`. Die Struktur der Logs umfasst:

1. **Zeitstempel**: Jede Anfrage enthält einen Zeitstempel, um festzuhalten, wann die Anfrage bearbeitet wurde (z.B. `"Time":"2024-10-17 16:24:51"`).

2. **Benutzerinformationen**: Die Protokolle dokumentieren die `User ID` und den `Username` des Benutzers, der die Anfrage gestellt hat (z.B. `"User ID: 1d595f35-39a0-4c5a-98ac-d26641995784", Username: mbuergi1986`).

3. **Anfragemethode und URI**: Es wird die HTTP-Methode (z.B. `"Method":"GET"`) und der angefragte URI (z.B. `"URI":"/user"`) aufgezeichnet.

4. **Statuscode**: Der Statuscode (z.B. `"Status":200`) gibt an, ob die Anfrage erfolgreich bearbeitet wurde.

5. **Response Time**: Die Antwortzeit (z.B. `"ResponseTime":0`) zeigt, wie lange es gedauert hat, um die Anfrage zu bearbeiten.

Hier eine kurze Beschreibung der **wichtigsten HTTP-Statuscodes**, die in diesem Zusammenhang relevant sind:

200 OK:

Die Anfrage war erfolgreich und das Ergebnis wird zurückgegeben. Dies ist der häufigste Erfolgsstatus für GET- und POST-Anfragen, bei denen keine Ressource erstellt wird.

201 Created:

Die Anfrage wurde erfolgreich verarbeitet und führte zur Erstellung einer neuen Ressource. Typischerweise wird dieser Statuscode bei erfolgreichen POST-Anfragen zurückgegeben, wenn eine neue Ressource (z. B. ein Datensatz) erstellt wurde.

204 No Content:

Die Anfrage wurde erfolgreich verarbeitet, aber es wird kein Inhalt zurückgegeben. Dieser Statuscode wird häufig verwendet, wenn eine Anfrage eine Aktion auslöst (z. B. das Löschen einer Ressource), der Server aber keine Antwortdaten liefert.

401 Unauthorized:

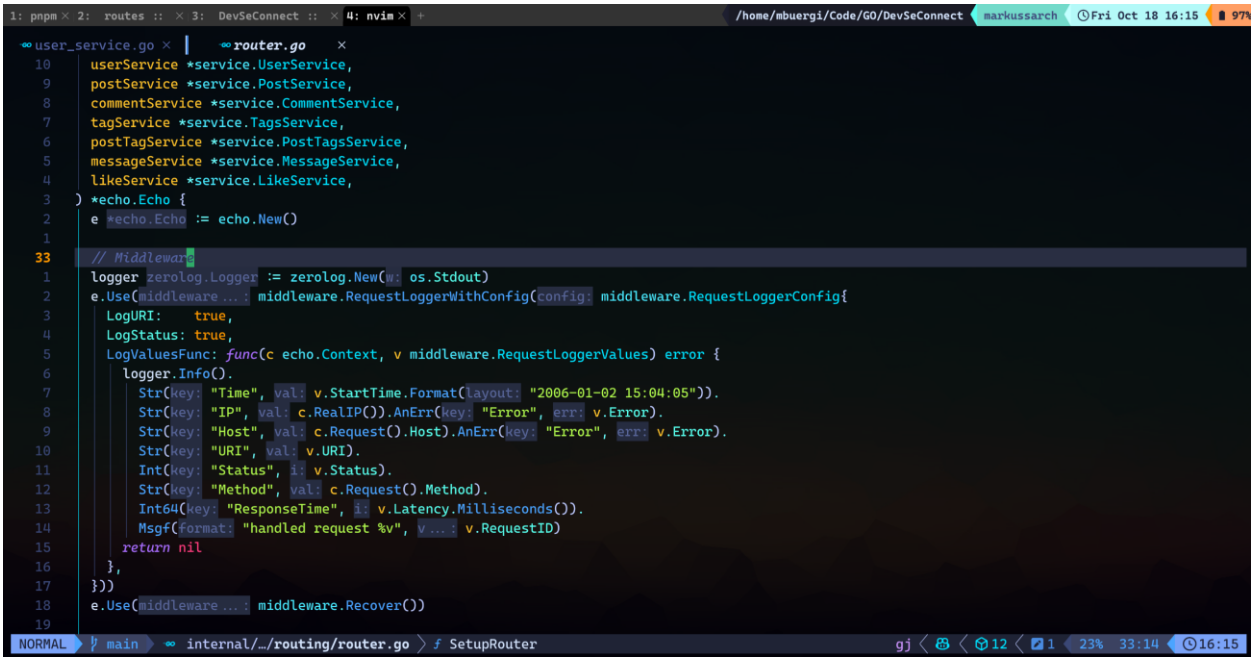
Die Anfrage erfordert eine Authentifizierung, aber es wurde keine gültige Authentifizierung übergeben oder die Authentifizierung ist ungültig. Dieser Fehler tritt häufig auf, wenn das Zugriffstoken fehlt oder abgelaufen ist.

404 Not Found:

Der angeforderte Ressourcen-Endpunkt wurde nicht gefunden. Dieser Fehler tritt auf, wenn die URL falsch ist oder die Ressource, auf die zugegriffen werden soll, nicht existiert.

500 Internal Server Error:

Ein allgemeiner Fehler auf der Serverseite. Dieser Statuscode zeigt an, dass auf dem Server ein unerwarteter Fehler aufgetreten ist, der die Bearbeitung der Anfrage verhindert hat. Dies kann z.B. durch einen Programmfehler oder einen unerwarteten Zustand verursacht werden.



Programmierung 17 – Backend-Implementierung des Routers mit Logging von Meldungen

7.11.10.2 Überwachung von PostgreSQL, RabbitMQ und Redis

Neben den Tests der API-Endpunkte ist die Überwachung der Backend-Dienste von zentraler Bedeutung, um ihre Stabilität und Performance sicherzustellen. Hierbei werden PostgreSQL**,** RabbitMQ und Redis mit «Grafana» überwacht.

Schritte zur Überwachung mit «Grafana»:

1. **Metrik-Sammlung mit «Prometheus»**:

   «Prometheus» wird verwendet, um Metriken aus PostgreSQL, RabbitMQ und Redis zu sammeln. Diese Metriken bieten Einblicke in die Systemleistung und -nutzung.

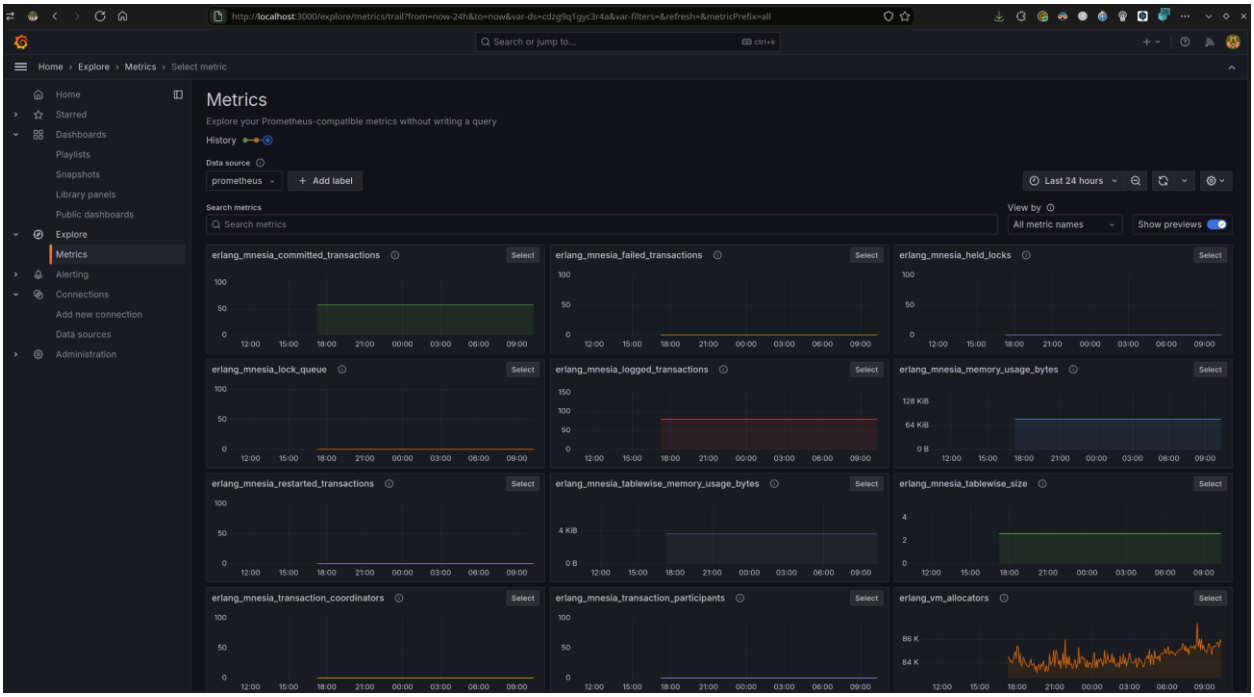


Abbildung 18 – Darstellung von Prometheus-Metriken in Grafana

Das Open-Source-Monitoring-System «Prometheus» wurde speziell für die Überwachung von Systemen und Anwendungen entwickelt. Die Funktionalität des Systems umfasst die Sammlung von Metriken von konfigurierten Zielsystemen, deren Speicherung als Zeitreihendaten sowie die Bereitstellung zur Analyse und Überwachung.

Der Begriff «Metrik» bezeichnet einen messbaren Wert, welcher Informationen über den Zustand und die Leistung eines Systems liefert. Als Beispiele für Metriken können die CPU-Auslastung, der Speicherverbrauch oder die Anzahl von erfolgreichen bzw. fehlgeschlagenen Anfragen genannt werden. In dem vorliegenden Bild werden verschiedene Metriken dargestellt, welche spezifische Werte eines Systems visualisieren. Diese werden von «Prometheus» erfasst und von «Grafana» grafisch dargestellt.

Das Ziel besteht darin, potenzielle Probleme frühzeitig zu erkennen und die Leistung zu optimieren, indem die Metriken überwacht werden.

2. **Visualisierung mit «Grafana»**:

   Die gesammelten Metriken werden in **«Grafana»** visualisiert. Dashboards zeigen relevante Daten wie die Datenbankleistung, Nachrichtenwarteschlangen in RabbitMQ und die Speichernutzung von Redis in Echtzeit.

3. **Erkennen von Engpässen**:

   «Grafana» ermöglicht die frühzeitige Erkennung von Leistungsproblemen, indem es Alarme für bestimmte Schwellenwerte konfiguriert, z.B. eine hohe CPU-Auslastung oder Speicherengpässe in PostgreSQL.

4. **Stetige Überwachung**:

   Diese Dashboards werden kontinuierlich überwacht, um sicherzustellen, dass das Backend unter verschiedenen Lastbedingungen stabil und performant bleibt.

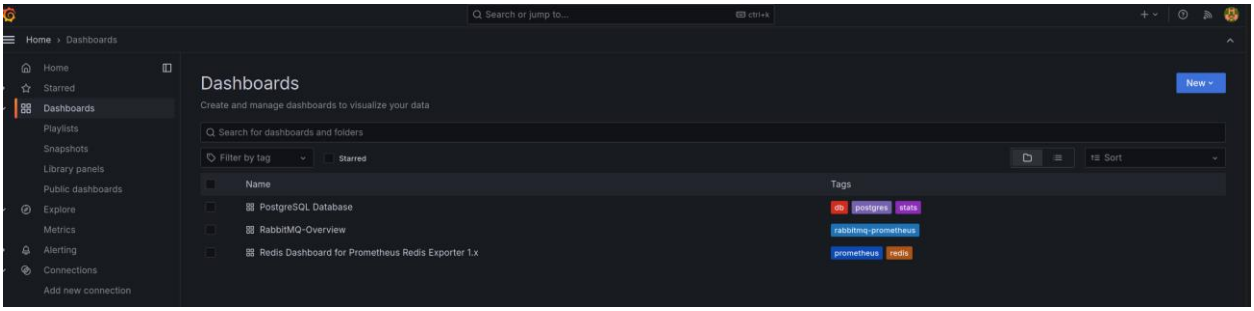


Abbildung 19 – Monitoring des Dashboard-Menüs in Grafana

Das Bild zeigt das **Dashboard-Menü von «Grafana»**, einem Tool, das zur Visualisierung von Metriken genutzt wird. In diesem Menü sind drei Dashboards zu sehen:

- **PostgreSQL Database:**

  Ein Dashboard zur Überwachung einer PostgreSQL-Datenbank, wahrscheinlich mit Metriken wie Abfragen, Speicher- und CPU-Auslastung.

- **RabbitMQ-Overview:**

  Ein Dashboard, das die Metriken des RabbitMQ-Messaging-Brokers anzeigt, wie z. B. Nachrichtenflüsse, Warteschlangenstatus und Performance.

- **Redis Dashboard for Prometheus Redis Exporter 1.x:**

  Ein Dashboard, das die Metriken von Redis überwacht, wahrscheinlich mit Informationen über Speicherverbrauch, Schlüsseloperationen und Performance.

Die Dashboards sind mit Tags versehen, die sie den entsprechenden Technologien zuordnen, wie postgres, rabbitmq, redis, und prometheus. Mit diesen Dashboards können die Systemadministratoren die Leistung und Gesundheit der jeweiligen Dienste überwachen.

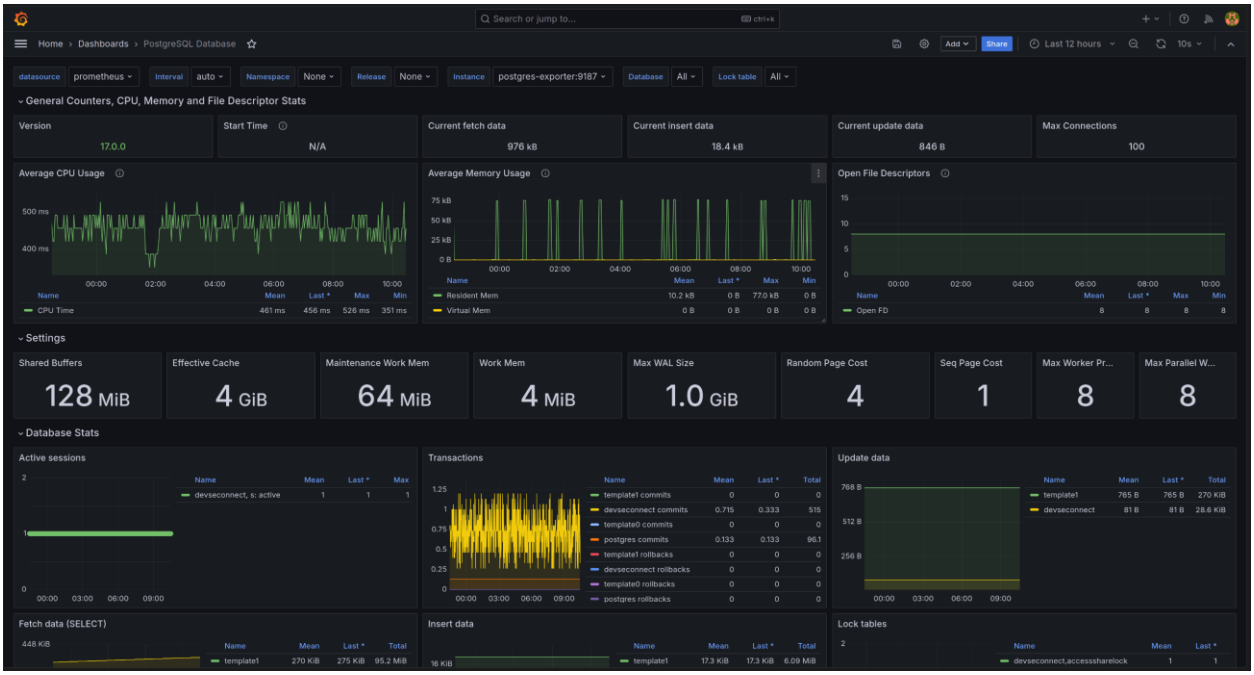


Abbildung 20 – Monitoring von PostgreSQL in Grafana

Das Bild zeigt ein **«Grafana-Dashboard»,** das für die **Überwachung einer PostgreSQL** genutzt wird. In diesem Fall werden verschiedene Metriken visualisiert, die zur Überwachung der Leistung und Stabilität der Datenbank dienen. Diese Metriken bieten Einblicke in die CPU-Auslastung, Speichernutzung, die Anzahl der Verbindungen, Transaktionen und Datenoperationen.

Wichtige Punkte der PostgreSQL-Überwachung:

- **CPU und Speicher:**

  Die durchschnittliche CPU-Zeit und die Speicherauslastung werden überwachten, um sicherzustellen, dass die Datenbank effizient arbeitet.

- **Datenverkehr:**

  Die Metriken zeigen die Menge an Daten, die abgerufen, eingefügt und aktualisiert werden, sowie die maximale Anzahl an Verbindungen zur Datenbank.

- **Datenbankeinstellungen:**

  Parameter wie Shared Buffers und Effective Cache sind sichtbar, die wichtige Einstellungen für die Leistung der Datenbank beeinflussen.

- **Transaktionen:**

  Die Anzahl der erfolgreichen und fehlgeschlagenen Transaktionen wird überwacht, um Probleme zu erkennen und die Datenbankkonsistenz sicherzustellen.

- **Sitzungen und Sperren:**

  Aktive Sitzungen und Sperren, die auf bestimmte Datenbankoperationen angewendet werden, werden ebenfalls angezeigt.

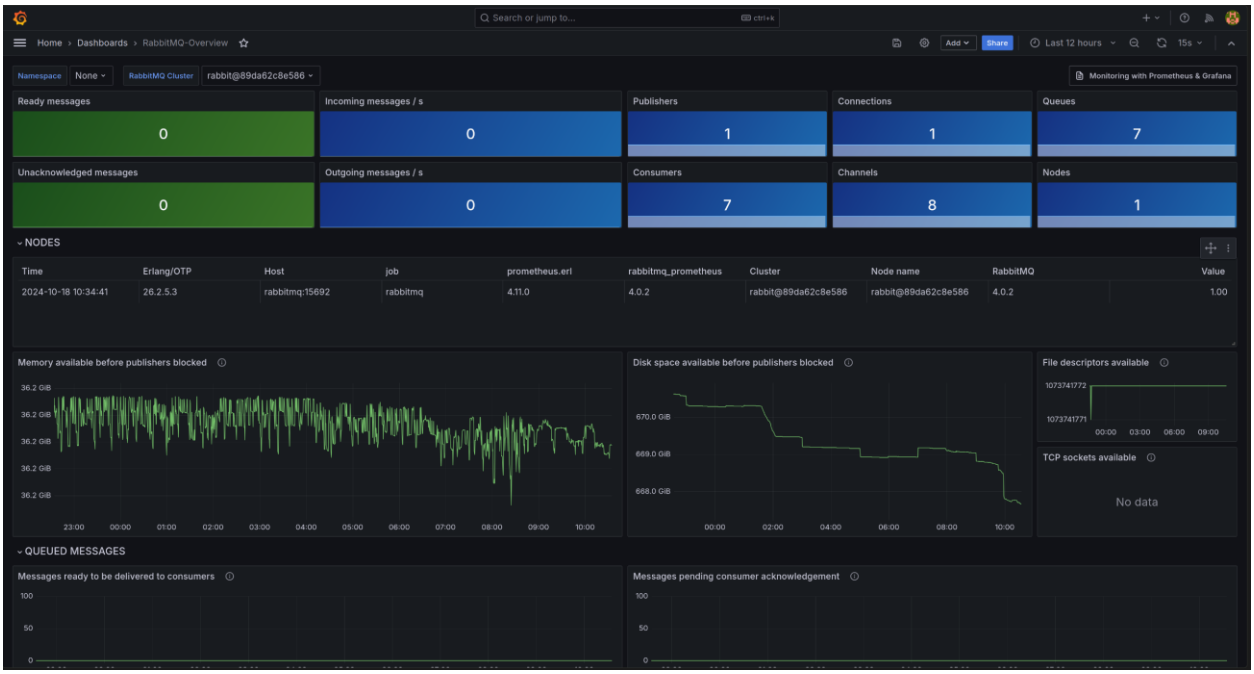


Abbildung 21 – Monitoring von RabbitMQ in Grafana

Das Bild zeigt ein **Grafana-Dashboard**, das die **RabbitMQ der Überwachung** darstellt. RabbitMQ ist eine Open-Source der Nachrichtenwarteschlange, die für Messaging und die Kommunikation zwischen verschiedenen Anwendungen oder Diensten verwendet wird.

Wichtige Punkte der RabbitMQ-Überwachung:

- **Nachrichtenstatus:**
  - Ready messages: Nachrichten, die bereit sind, an Verbraucher gesendet zu werden.
  - Unacknowledged messages: Nachrichten, die von den Verbrauchern empfangen, aber noch nicht bestätigt wurden.

- Incoming/Outgoing messages per second: Anzahl der eingehenden und ausgehenden Nachrichten pro Sekunde.

- **Verbindungen und Warteschlangen:**
  - Publishers: Anzahl der Anwendungen, die Nachrichten an RabbitMQ senden.
  - Consumers: Anzahl der Anwendungen, die Nachrichten von RabbitMQ empfangen.
  - Connections, Queues, Channels, Nodes: Zeigt die Anzahl der aktiven Verbindungen, Warteschlangen, Kanäle und Knoten im RabbitMQ-Cluster.

- **Systemressourcen:**
  - Memory: Verfügbarer Speicher für RabbitMQ, bevor Publisher blockiert werden.
  - Disk space: Verfügbarer Speicherplatz auf der Festplatte, bevor Publisher blockiert werden.
  - File descriptors & TCP sockets: Verfügbare Dateideskriptoren und TCP-Sockets, die für die Verbindungen benötigt werden.

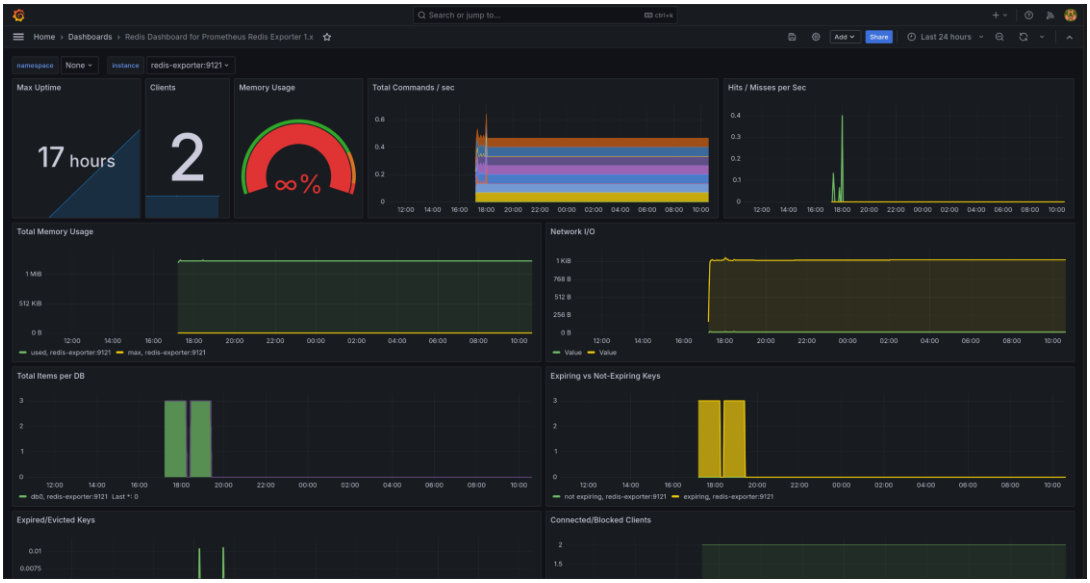


Abbildung 22 – Monitoring von Redis in Grafana

Das Bild zeigt ein **Grafana-Dashboard**, das speziell für die **Überwachung von Redis** eingerichtet ist. Redis ist eine In-Memory-Datenbank, die häufig für Caching und schnelles Speichern von Daten verwendet wird. Die Metriken in diesem Dashboard bieten einen Überblick über den Status und die Leistung des Redis-Servers.

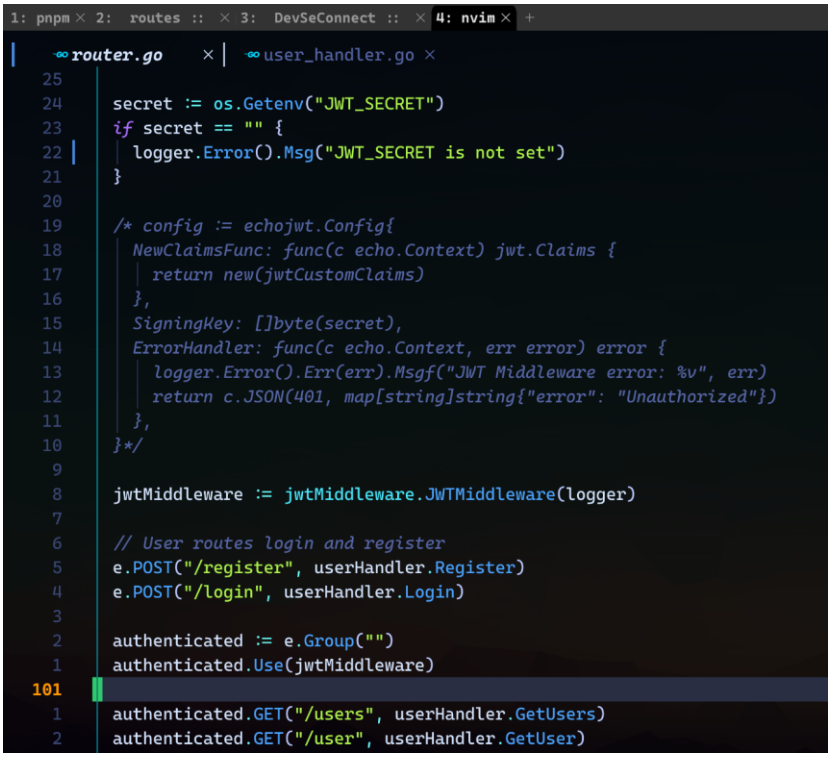
Wichtige Punkte:

- **Max Uptime:**
  - Zeigt an, wie lange der Redis-Server bereits läuft (in diesem Fall 17 Stunden).
- **Clients:**
  - Gibt die Anzahl der aktiven Clients an, die derzeit mit dem Redis-Server verbunden sind (zwei Clients).
- **Memory Usage:**
  - Veranschaulicht den Speicherverbrauch des Redis-Servers, wobei hier ein ungewöhnlich hoher Speicherverbrauch angezeigt wird (∞%).
- **Total Commands per Second:**
  - Zeigt die Anzahl der Redis-Befehle, die pro Sekunde ausgeführt werden.
- **Hits / Misses per Second:**
  - Zeigt den Erfolg von Cache-Abfragen (Hits) und fehlgeschlagene Anfragen (Misses), was auf eine effiziente oder ineffiziente Nutzung des Caches hinweisen kann.
- **Total Memory Usage:**
  - Ein Diagramm, das die tatsächliche Speicherverwendung im Vergleich zum verfügbaren Speicher zeigt.
- **Network I/O:**
  - Veranschaulicht den Datenverkehr, der durch den Redis-Server läuft.
- **Expiring vs. Not-Expiring Keys:**
  - Zeigt die Anzahl der Schlüssel, die im Redis-Cache ablaufen, und die Schlüssel, die nicht ablaufen.
- **Expired/Evicted Keys:**
  - Zeigt die Anzahl der abgelaufenen oder entfernten Schlüssel an, was darauf hinweisen kann, dass der Cache voll ist oder veraltete Daten entfernt werden.

7.11.11 Sicherheitsaspekte des Backends

Die Gewährleistung der Sicherheit des Backends stellte einen entscheidenden Faktor dar, um die Unversehrtheit, Vertraulichkeit und Verfügbarkeit der Plattform zu sichern.

7.11.11.1 JWT-Authentifizierung in GO



Programmierung 18 – Backend-Implementierung der Authentifizierung mit JWT

Das Bild zeigt einen Codeausschnitt in «Go» (Golang), der **JWT (JSON Web Token) Authentifizierung** verwendet. Hier wird geprüft, ob ein JWT-Secret gesetzt ist, und ein Middleware-Mechanismus implementiert wird, um den Zugriff auf geschützte Routen zu sichern.

7.11.11.2 Erklärung des Codes:

1. Geheim-Überprüfung:

```
secret := os.Getenv("JWT_SECRET")
if secret == "" {
    logger.Error().Msg("JWT_SECRET is not set")
}
```

Programmierung 19 – Backend-Implementierung der Environment

Hier wird das «JWT_SECRET» aus den Umgebungsvariablen gelesen. Wenn kein «JWT_SECRET» gesetzt ist, wird ein Fehler geloggt und eine Meldung angezeigt.

2. JWT-Middleware:

```
jwtMiddleware := jwtMiddleware.JWTMiddleware(logger)
```

Programmierung 20 – Backend-Implementierung der Middleware für JWT

Die JWT-Middleware wird verwendet, um zu prüfen, ob eingehende Anfragen ein gültiges JWT-Token haben. Dies stellt sicher, dass nur authentifizierte Benutzende Zugriff auf geschützte Routen haben.

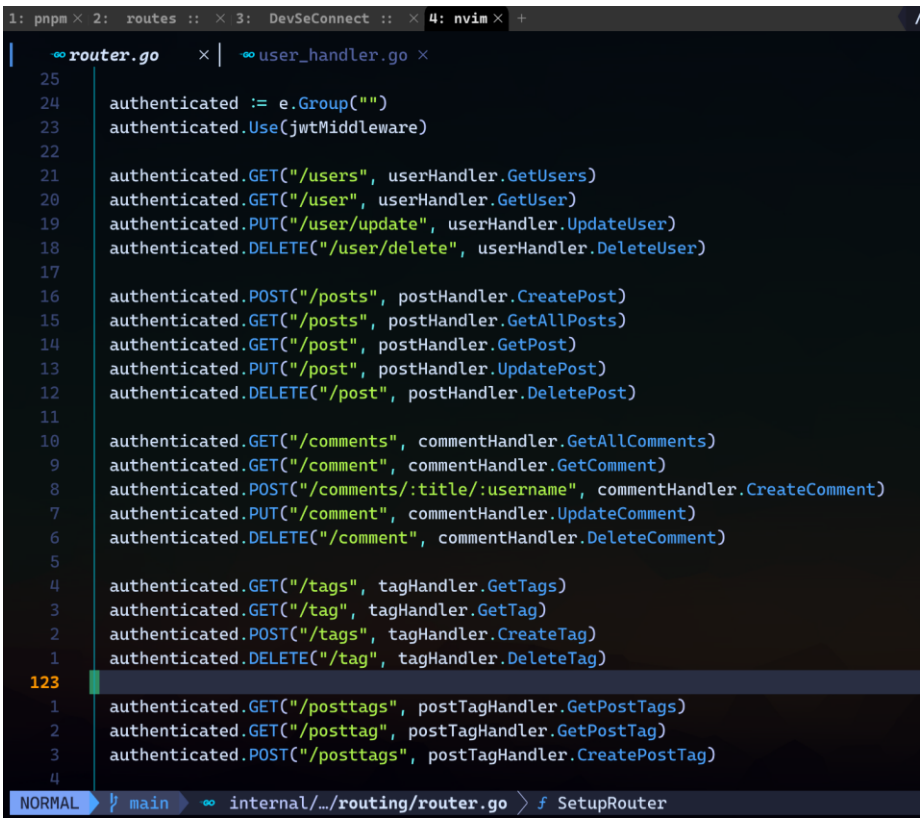
3. Routen für die Registrierung das und Login:

```
// User routes login and register
e.POST("/register", userHandler.Register)
e.POST("/login", userHandler.Login)
```

Programmierung 21 – Backend-Implementierung des Routers mit Handler-Verknüpfung

Hier werden zwei öffentliche Endpunkte für die Registrierung und das Login definiert. Diese Endpunkte erfordern keine Authentifizierung.

4. Authentifizierte Routen:



Programmierung 22 - Backend-Implementierung des Routers mit Handler-Verknüpfung und Authentifizierung

Eine Route-Gruppe wird erstellt, auf die nur authentifizierte Benutzende zugreifen können. Die Middleware stellt sicher, dass ein gültiges JWT vorliegt. Hier werden z. B. Routen definiert, um Benutzerinformationen der Benutzenden abzurufen.

7.11.11.3 Sicherheitsaspekte:

- **JWT-Sicherheit:** Durch die Verwendung von JWT-Token wird sichergestellt, dass nur autorisierte Benutzende auf bestimmte Routen zugreifen können. Das Geheim wird verwendet, um Token zu signieren und die Authentizität zu überprüfen.
- **Fehlerprotokollierung:** Falls das «JWT_SECRET» nicht gesetzt ist, wird dies sofort als kritischer Fehler behandelt und geloggt, um sicherzustellen, dass keine ungesicherten Verbindungen bestehen.
- **Middleware:** Die Middleware sorgt dafür, dass unbefugte Anfragen abgefangen und als «Unauthorized» zurückgewiesen werden, was eine weitere Sicherheitsebene darstellt.

7.12 FRONTEND-IMPLEMENTIERUNG

Im Rahmen dieser Arbeit wird «SvelteKit» als Entwicklungswerkzeug herangezogen, da es sich als ressourcenschonende und optimierte Wahl für das System auszeichnet. Die kürzlich erschienene Version 5 von «SvelteKit» beinhaltet eine Vielzahl an Verbesserungen, die darauf abzielen, den Entwicklungsprozess effizienter zu gestalten. Die Verwendung von «SvelteKit» in Kombination mit «TypeScript» ist weit verbreitet, da «TypeScript» im Vergleich zu «JavaScript» eine strukturiertere und strengere Typendeklaration bietet. Dies führt zu einer verbesserten Code-Qualität und erleichtert das Auffinden und Beheben von Fehlern im Entwicklungsprozess.

7.12.1 Startseite

Die Startseite erweckt den Eindruck, dass die Disziplinen des User-Experience- und User-Interface-Designs nur geringfügige Unterschiede aufweisen. Dennoch lässt sich eine signifikant hohe Nutzerzufriedenheit feststellen.

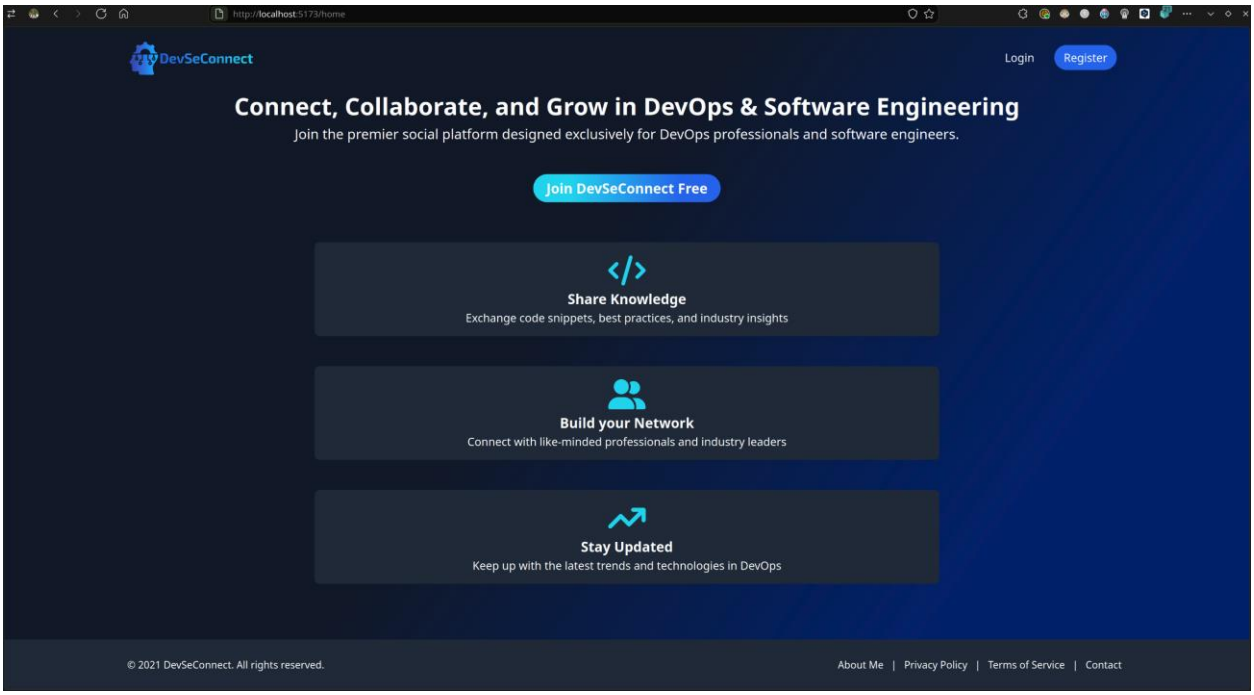


Abbildung 23 – Startseite des Internets

7.12.2 Registrierung

Im Rahmen der Registrierung werden die folgenden Daten erhoben: Name, Benutzername, E-Mail-Adresse und Passwort. Die Richtlinie verlangt die doppelte Eingabe des Passworts, wobei identische Eingaben ausgeschlossen sind, um das Risiko von Sicherheitsverletzungen zu minimieren. Bei Abweichungen zwischen dem registrierten und dem bestätigten Passwort wird eine Fehlermeldung angezeigt, welche die Nutzenden darauf hinweist, dass das Formular korrekt auszufüllen ist. Die erhobenen Daten werden unmittelbar an den Webserver übermittelt, von wo aus sie entweder gespeichert oder abgefragt werden. Zu diesem Zweck wird eine sogenannte Application Programming Interface (API) an die Datenbank weitergeleitet.

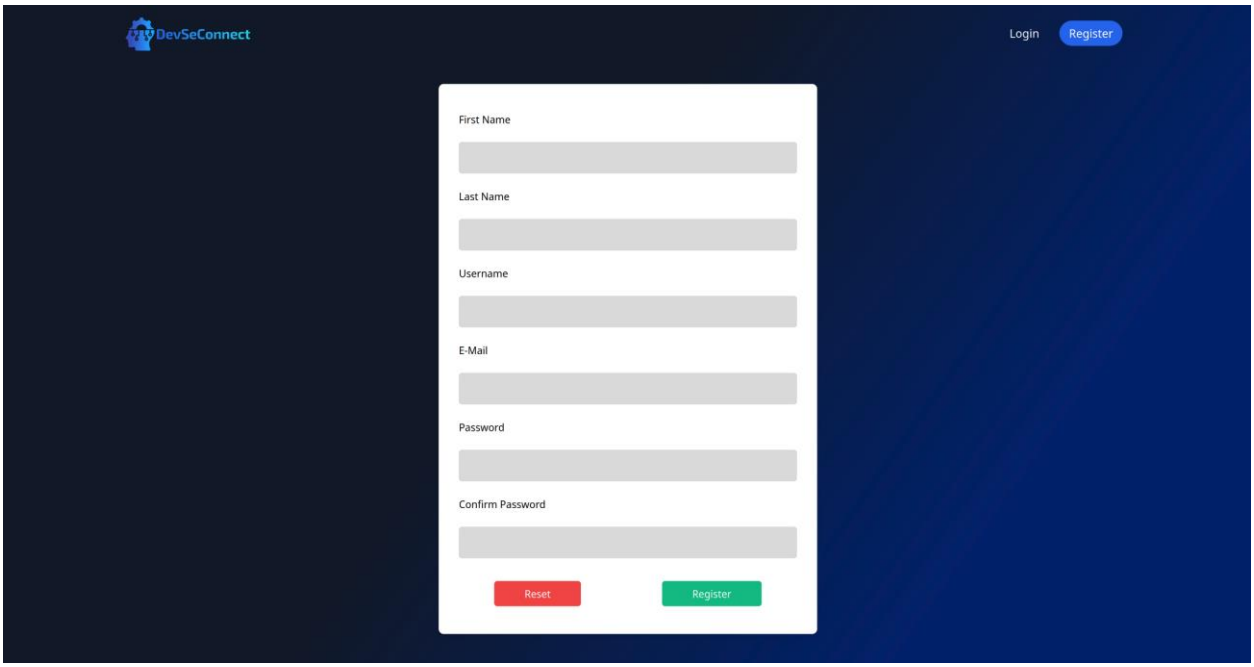


Abbildung 24 – Registrierung des Internets

7.12.3 Login-Seite

Die Gestaltung der Login-Seite ist geprägt durch eine übersichtliche und auf das Wesentliche reduzierte Form. Im Rahmen des Authentifizierungsprozesses erfolgt zunächst eine Anfrage an die Datenbank, anschliessend wird der Webserver konsultiert. Unter der Voraussetzung der korrekten Eingabe von Benutzername und Passwort generiert das System ein neues JSON Web Token (JWT), welches den Nutzenden den Zugang ermöglicht und gleichzeitig als Autorisierung für zukünftige Interaktionen dient.

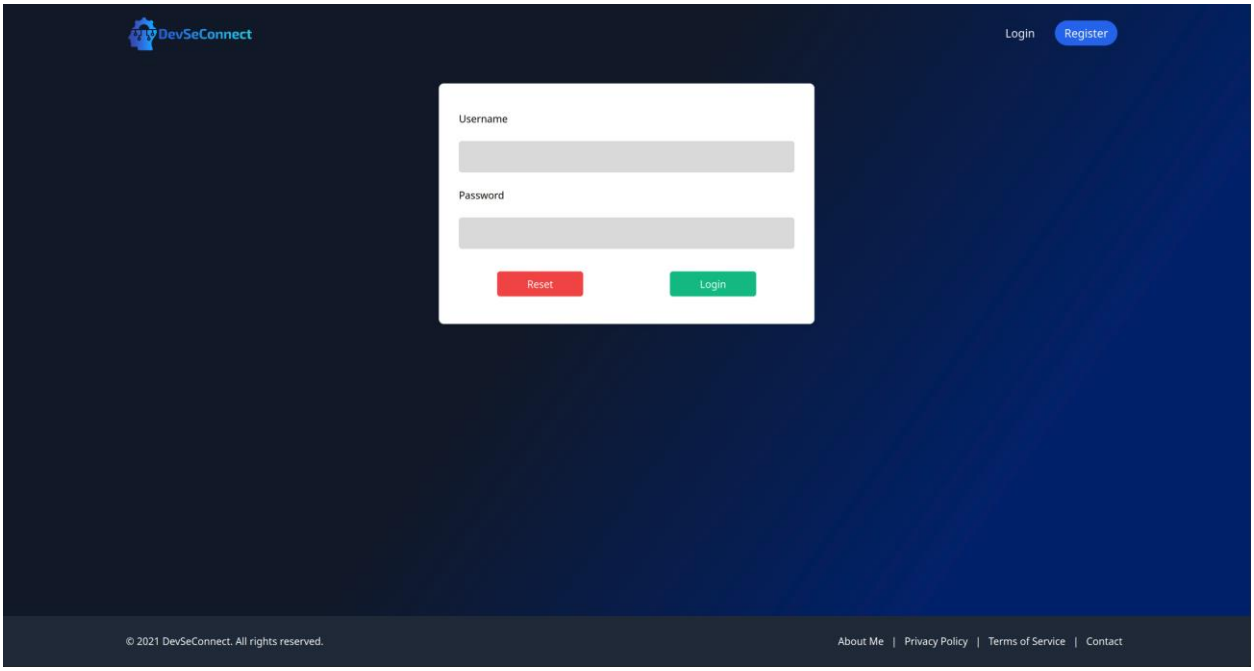


Abbildung 25 – Login des Internets

7.12.4 JSON Web Tokens

Nach erfolgreicher Authentifizierung des Nutzers erfolgt die Generierung eines neuen JSON Web Tokens (JWT), welches den Zugang zum Dashboard ermöglicht. Das Token fungiert somit als Autorisierung für den Zugriff auf das Dashboard sowie als Gewährleistung der Unversehrtheit nachfolgender Interaktionen mit dem System.

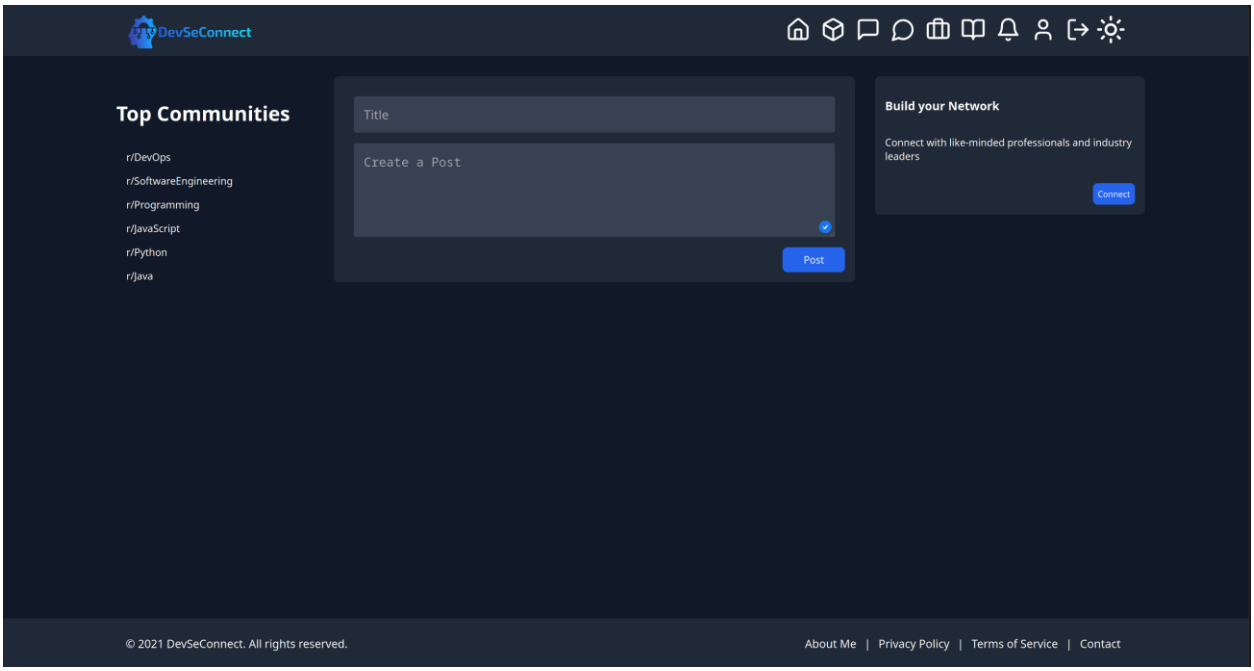


Abbildung 26 – Dashboard des Internets

An dieser Stelle wird das Token in gelber Markierung angezeigt. Sollte den Nutzer:innen das Token verlieren oder es manuell löschen wollen, wird automatisch eine Abmeldung aus dem Dashboard initiiert, gefolgt von einer Rückkehr zur Login-Seite.

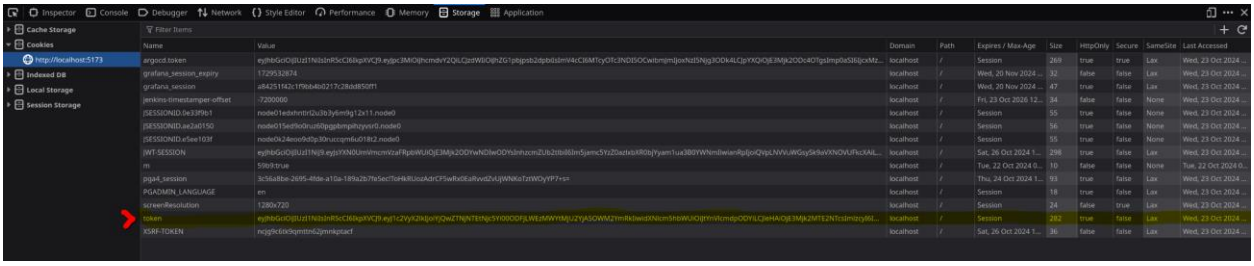


Abbildung 27 – Speicherung des JWT-Tokens in der Webanwendung

7.12.5 Profil

Das Profil präsentiert sich in einer vergleichsweise einfachen und noch unvollständigen Struktur, was auf die Notwendigkeit weiterer Anpassungen bei der Datenabfrage und -darstellung hinweist. Aufgrund von Herausforderungen im Zeitmanagement wurden bisher keine umfassenden Änderungen durchgeführt. Perspektivisch besteht jedoch die Möglichkeit, das Profil und weitere Seiten weiter zu optimieren und gezielt zu überarbeiten.

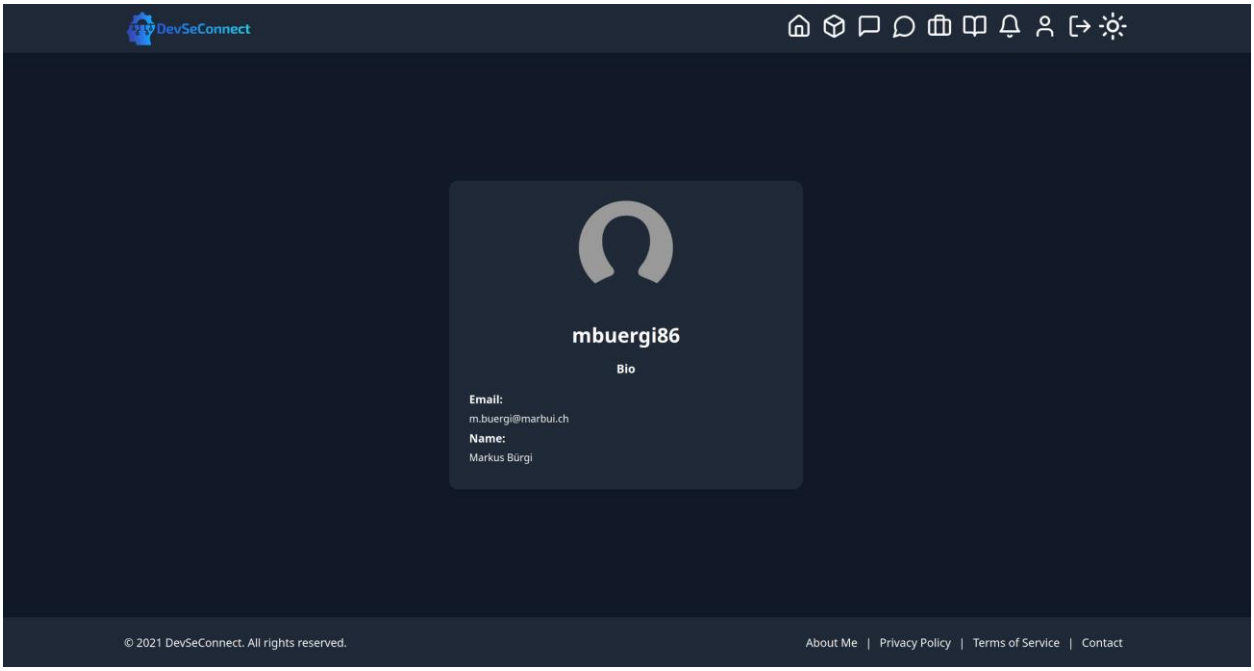


Abbildung 28 – Profil des Internets

7.12.6 Neuer Beitrag

Der Prozess zur Erfassung eines neuen Beitrags in der Datenbank verläuft reibungslos, wobei die Seite nach der erfolgreichen Speicherung automatisch aktualisiert wird.

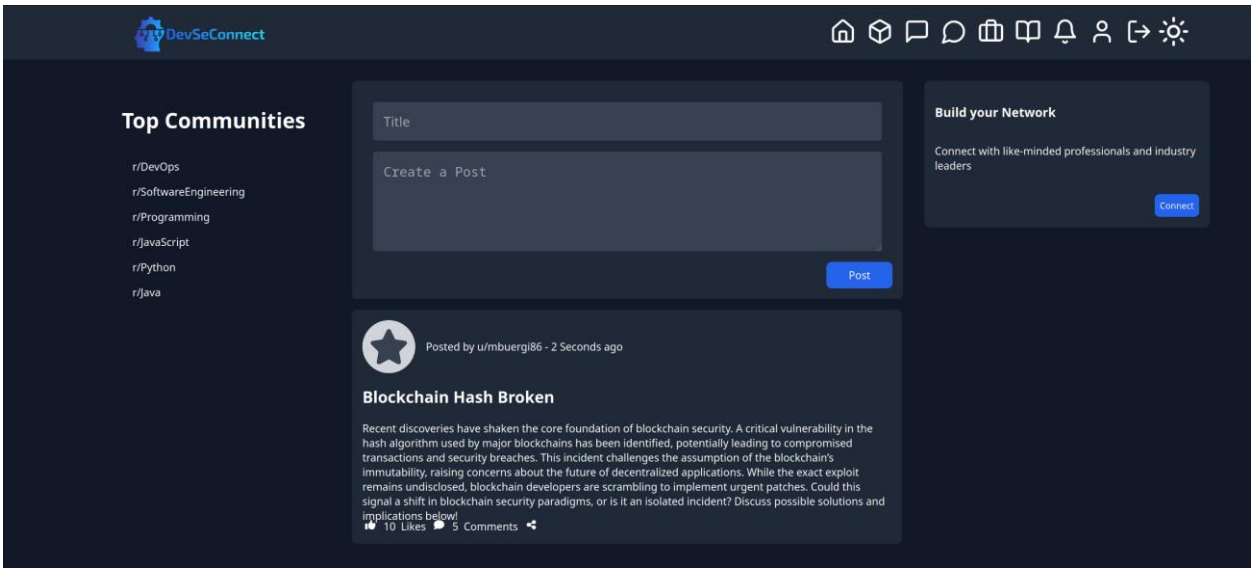


Abbildung 29 – Dashboard des Internets

7.13 CI/CD-Pipeline Implementierung

7.13.1 Jenkins

«Jenkins» ist ein Open-Source-Tool zur Automatisierung von Softwareentwicklungsprozessen, besonders für Continuous Integration (CI) und Continuous Deployment (CD). Es ermöglicht Entwickelnden, die Software automatisch zu bauen, testen und deployen. «Jenkins» läuft auf «Java» und bietet Flexibilität durch eine Vielzahl von Plugins.

7.13.1.1 Wichtige Funktionen:

- **CI/CD-Pipeline:**
  - Automatisiert den Entwicklungsprozess in Schritten wie Code-Testen, Bauen, und Bereitstellen.
- **Continuous Integration:**
  - Jenkins testet und integriert neuen Code automatisch ins gemeinsame Repository.
- **Continuous Deployment:**
  - Automatisiert das Deployment von Anwendungen in verschiedene Umgebungen.

Durch seine Flexibilität und Automatisierung spart «Jenkins» Zeit und hilft die Software schneller und zuverlässiger zu veröffentlichen.

7.13.1.2 Netzwerk und Aufbau «Jenkins»

Nach der Installation von «Jenkins» in Docker erfolgt die Ausführung über den Befehl "docker-compose.yml". Das folgende Bild zeigt eine beispielhafte Darstellung der Datei "docker-compose.yml".

```
12  jenkins:
11    image: jenkins/jenkins:lts
10    container_name: jenkins
 9    restart: always
 8    ports:
 7      - "8082:8080"
 6      - "50000:50000"
 5    volumes:
 4      - jenkins_home:/var/jenkins_home
 3    environment:
 2      - TZ=Europe/Zurich
 1    networks:
119      - jenkins_network
```

Programmierung 23 – Implementierung der Docker-Compose YAML-Datei für Jenkins

Die Weltzeit lautet «Europa/Zürich». Es erfolgt eine Isolierung des Jenkins-Servers innerhalb des Netzwerkes, wobei eine Kommunikation mit anderen Instanzen des Netzwerkes unterbunden wird.

Im Docker Log wird ein Initialpasswort angezeigt, welches kopiert und anschliessend eingefügt werden kann.

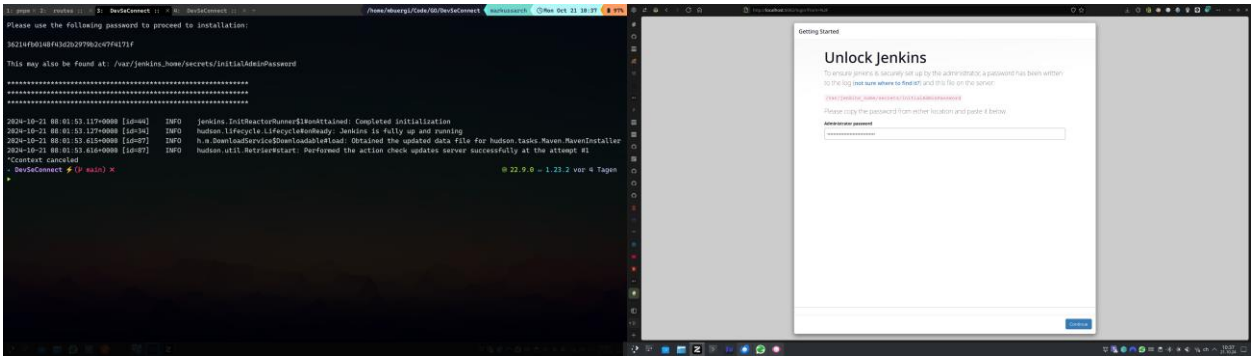


Abbildung 30 – Initialpasswort für Jenkins

Die Installation erfolgt automatisch durch «Jenkins». Ein manuelles Vorgehen war nicht länger erforderlich.

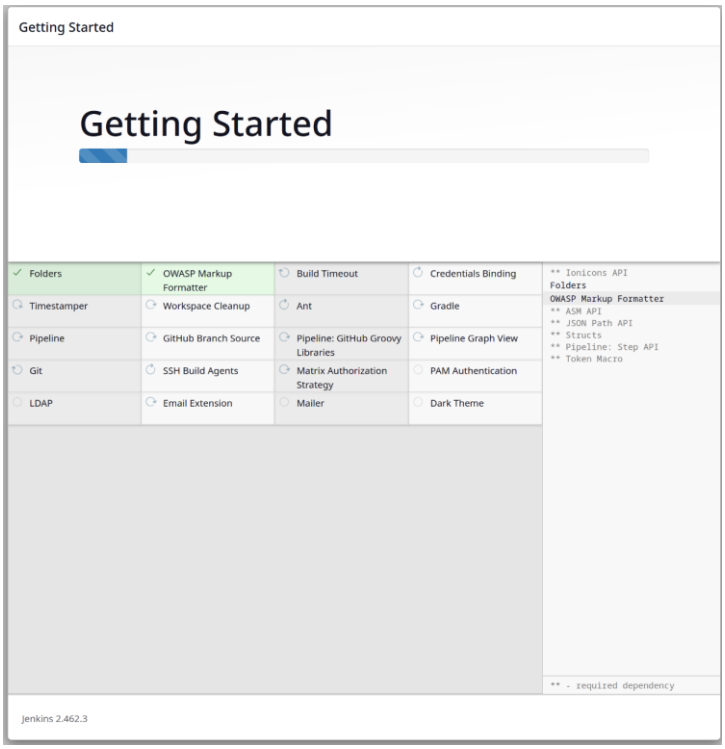


Abbildung 31 – Installation von Jenkins

Die Rolle des Administrators für das eigene «Jenkins Management» ist zu besetzen.

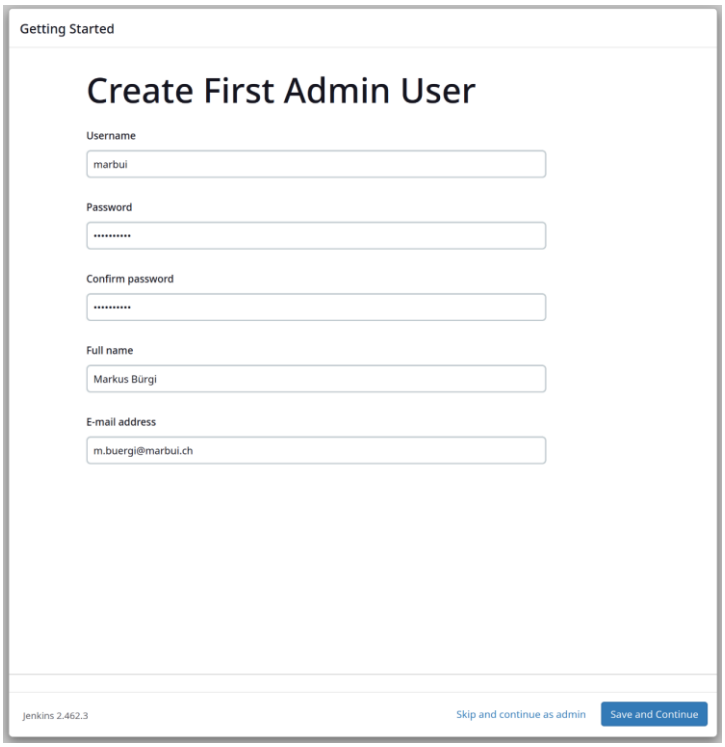


Abbildung 32 – Einrichtung von Jenkins

Es besteht die Möglichkeit, die anpassende URL von der Adressleiste oben in die Adressleiste des lokalen Netzwerks zu übertragen. Da jedoch der lokale Netzwerkbetrieb keine Domäne bereitstellt, ist es erforderlich, localhost:8082 einzugeben, um den Zugriff auf «Jenkins Management» zu ermöglichen.

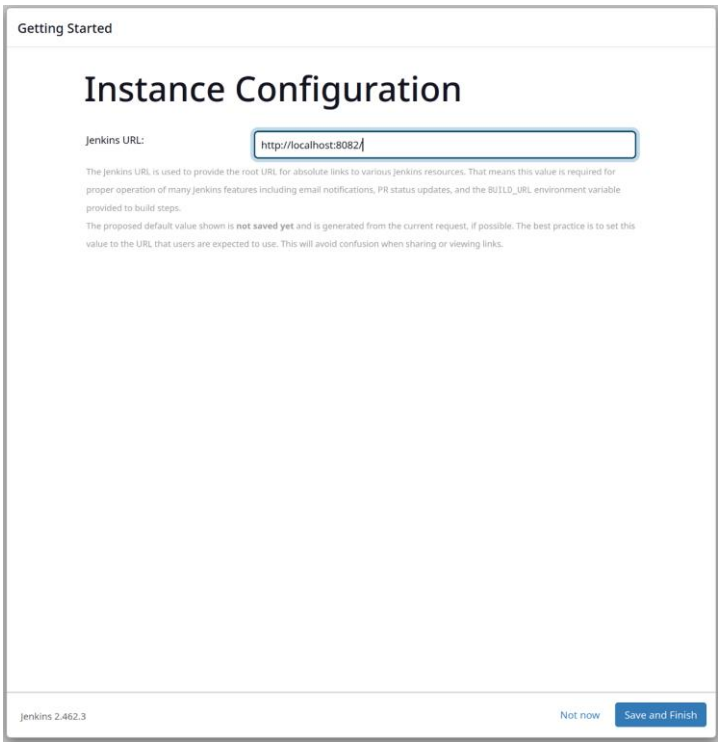


Abbildung 33 – Konfiguration von Jenkins

Für den Zugriff auf die genannten Plattformen ist die Eingabe von Zugangsdaten erforderlich, wobei es sich nicht nur um «GitHub» handelt, sondern auch um «GitLab» und andere. Zudem ist der Einsatz von «SSH-Zugangsdaten» notwendig.

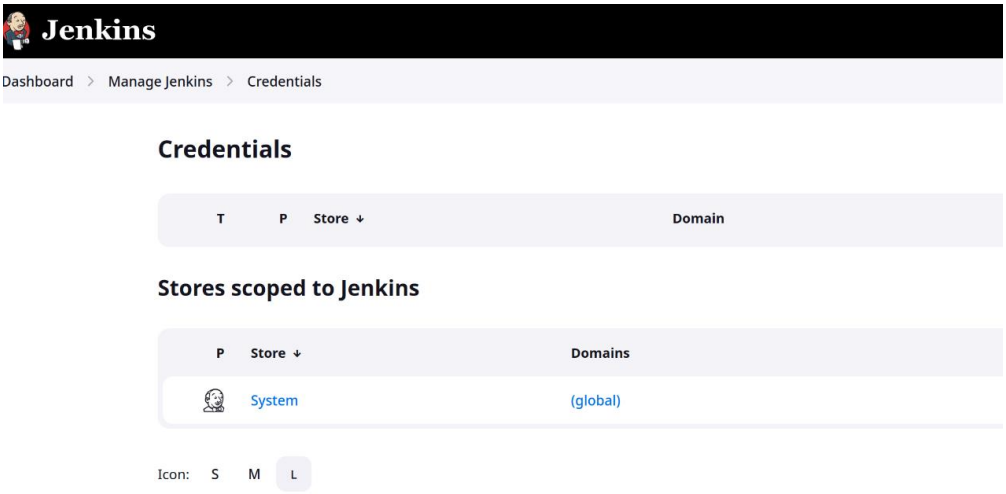


Abbildung 34 – Credentials in Jenkins

Der Unterschied zwischen «Global» und «System» liegen darin, dass «Global» ausserhalb «Jenkins» den Zugang z.B. «GitHub», «GitLab», «SSH» regelt und «System» innerhalb ««Jenkins des Zugangs.

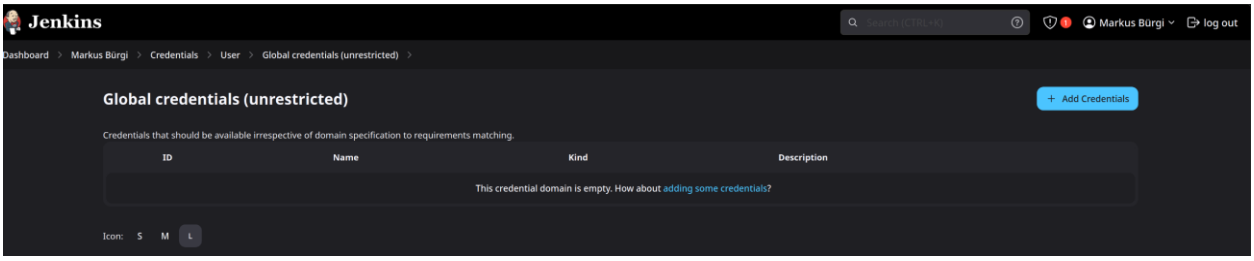


Abbildung 35 – Globale Credentials in Jenkins

Nach den vorliegenden Informationen bietet «Jenkins» einen globalen Zugang für «GitHub» an.

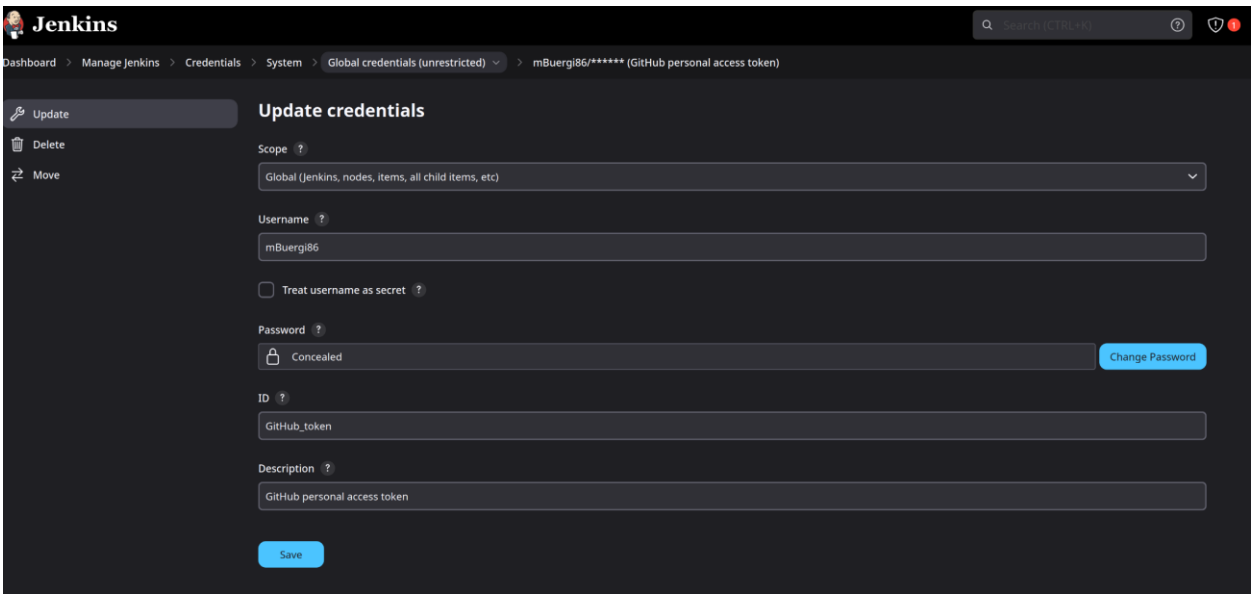


Abbildung 36 – Neue Credential-Einrichtung in Jenkins

Die erfassten «Credentials» repräsentieren einen neuen Zugang zu «GitHub». Es besteht die Möglichkeit, diesen mit einem «GitHub-Repository» zu verbinden.

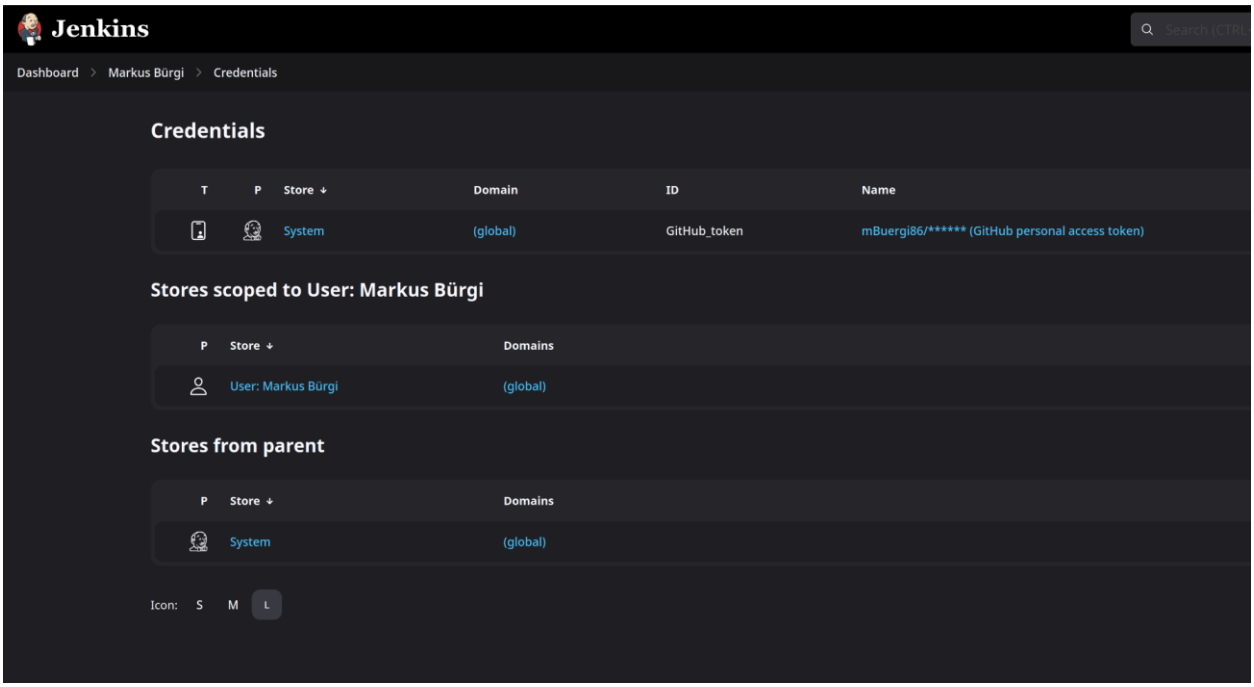


Abbildung 37 – Überblick über Credentials in Jenkins

Im nächsten Schritt ist die Erstellung einer Pipeline erforderlich, wobei anschliessend die Option «Multibranch Pipeline» ausgewählt werden muss. «Multibranch Pipeline» dient der Zusammenführung mehrerer Branchen, beispielsweise «Merge», «Checkpoint» und anderer Branche, wie etwa «Test» oder «API».

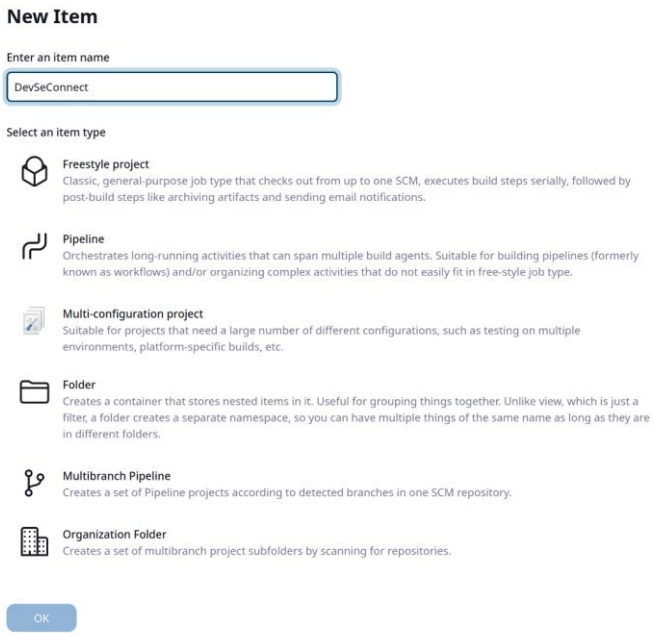


Abbildung 38 – Neuer Eintrag in Jenkins

In einem vorherigen Schritt hat das «Credentials» bereits einen globalen Zugang erstellt, um die Quelle von «GitHub» zu identifizieren und das entsprechende Repository einzugeben.

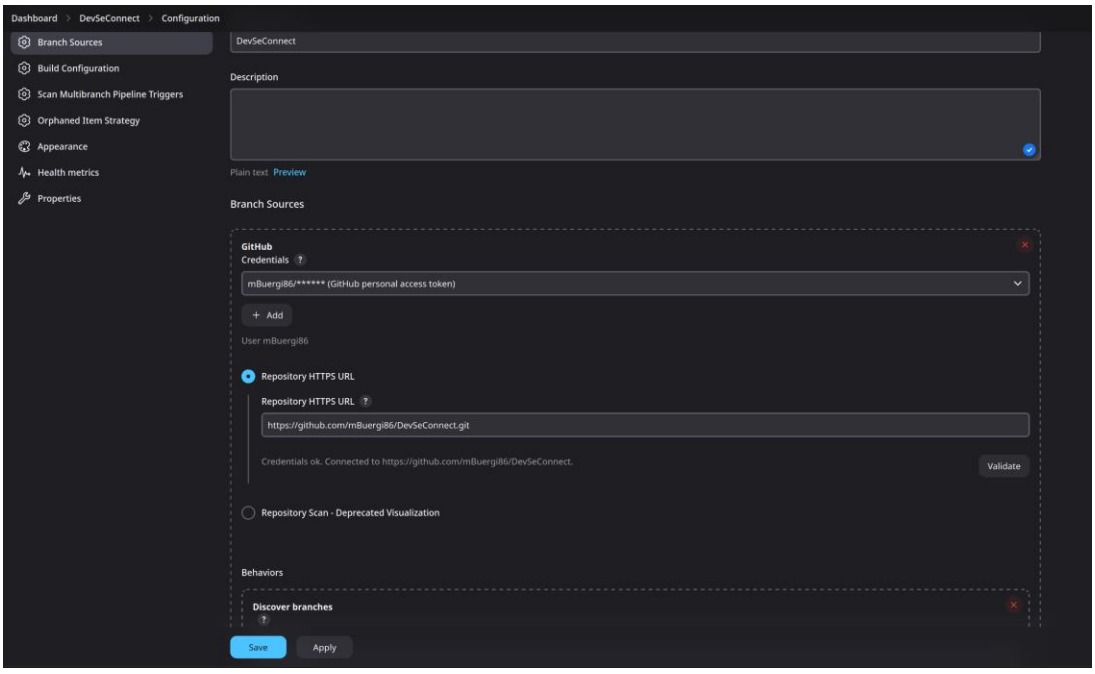


Abbildung 39 – Konfiguration von Jenkins

Es wird empfohlen, zunächst den Repository Log auszuführen und zu testen, ob das Aufrufen und Verbinden des «GitHub Repository» korrekt erfolgt. Allerdings ist bislang noch keine «Jenkinsfile» vorhanden, sodass deren Erstellung zu einem späteren Zeitpunkt erfolgen kann. Dennoch kann ein Erfolg verzeichnet werden. Dies bedeutet, dass die einzelnen Schritte in korrekter Abfolge zum Erfolg führen.

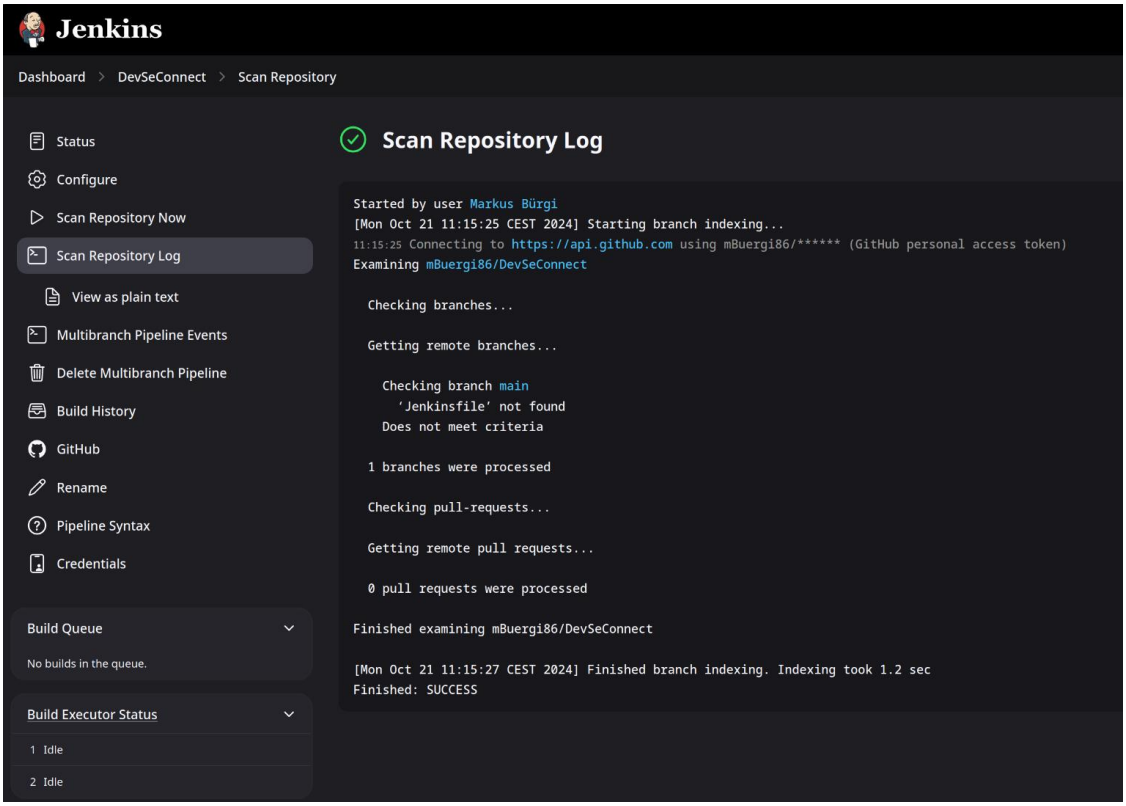


Abbildung 40 – Scan Repository Log in Jenkins

Es könnte eine «Jenkinsfile» erstellt werden, um erstmal mit dem Test zu beobachten, wie die Konsolenanwendung über «Jenkins» aussehen könnte.

Programmierung 24 – Kurze Implementierung eines Jenkinsfiles

- **Pipeline:**
  - Die Pipeline bildet den Kern des Workflows in Jenkins und definiert die Abfolge verschiedener Schritte wie Bauen, Testen und Deployen.
- **Agent:**
  - Der Agent definiert denjenigen Rechner oder Container, auf dem die Pipeline ausgeführt wird.
- **Stage:**
  - Eine Pipeline setzt sich aus mehreren Stufen (sogenannten Stages) zusammen, welche verschiedene logische Schritte definieren. Beispielhaft seien hier die Stages "Build" und "Test" genannt.
- **Step:**
  - Jeder Step ist ein einzelner Befehl oder eine Aktion innerhalb einer Stage, wie z. B. das Kompilieren des Codes oder das Starten eines Tests.
- **Echo:**
  - Die Echo-Funktion dient der Ausgabe von Nachrichten in der Pipeline und wird oft zum Debuggen oder zur Information über den aktuellen Status verwendet.

Die Ausführung von Build 01 kann als erfolgreich bezeichnet werden.

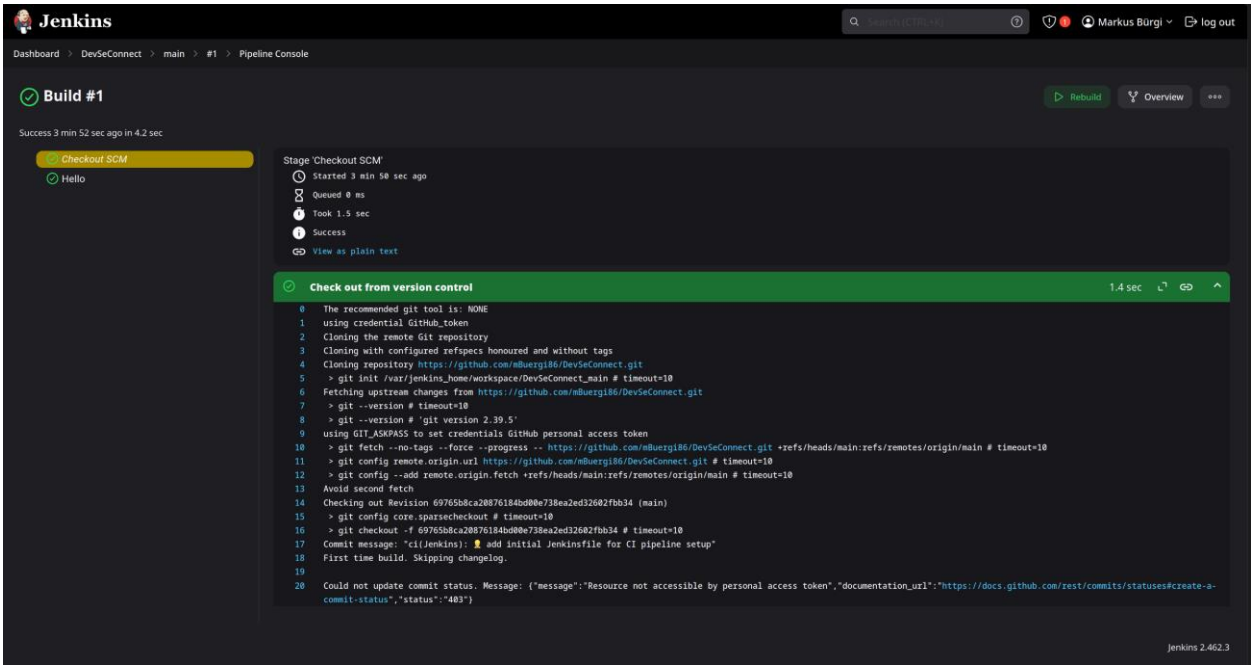


Abbildung 41 – Erfolgreicher Pipeline-Aufbau in Jenkins

Die Ausführung präsentiert die Ausgabe des Textes "Hello World".

Abbildung 42 – Erfolgreicher Pipeline-Aufbau in Jenkins

«Jenkinsfile» liegt in Führung bei «Stages», benötigt für «Build», «Test», «Deploy» auch «SonarQube» dazu.

7.13.2 SonarQube

«SonarQube» stellt eine Open-Source-Plattform zur statischen Code-Analyse und Bewertung der technischen Qualität von Quellcode dar. Die Analyse umfasst verschiedene Qualitätsaspekte, darunter doppelten Code, Komplexität, potenzielle Fehler sowie Testabdeckung. Die Ergebnisse der Analyse werden über eine Weboberfläche dargestellt. «SonarQube» unterstützt eine Vielzahl an Programmiersprachen und verfügt über Plugins, die eine Erweiterung der Funktionen ermöglichen. Es findet häufig Anwendung in CI/CD-Pipelines, um die Codequalität während der Entwicklung und Bereitstellung zu überprüfen.

7.13.2.1 Anwendung «SonarQube»

Nach der Installation von «SonarQube» in Docker erfolgt die Ausführung über den Befehl "docker-compose.yml". Das folgende Bild zeigt eine beispielhafte Darstellung der Datei "docker-compose.yml".

```
137  sonarqube:
  1    image: sonarqube:lts-community
  2    command: >
  3      -Dsonar.ce.javaOpts=-Xmx1192m
  4      -Dsonar.web.javaOpts=-Xmx1192m
  5    restart: unless-stopped
  6    container_name: sonarqube
  7    depends_on:
  8      - sonarqube_db
  9    environment:
 10      - SONARQUBE_JDBC_URL=jdbc:postgresql://sonarqube_db:5432/sonar
 11      - SONARQUBE_JDBC_USERNAME=sonar
 12      - SONARQUBE_JDBC_PASSWORD=sonar
 13      - SONAR_ES_JAVA_OPTS=-Xms512m -Xmx1g
 14    ports:
 15      - 9000:9000
 16    volumes:
 17      - sonar-data:/opt/sonarqube/data
 18      - sonar-extensions:/opt/sonarqube/extensions
 19      - sonar-logs:/opt/sonarqube/logs
 20    networks:
 21      - jenkins_network
```

Programmierung 25 - Implementierung der Docker-Compose YAML-Datei für SonarQube

Zu Beginn ist eine Anmeldung mit dem Benutzernamen «admin» und demselben Passwort erforderlich. Im Anschluss ist ein neues Passwort zu vergeben. Es erfolgt eine Meldung über den erfolgreichen Wechsel zum Dashboard des betreffenden Projektes.

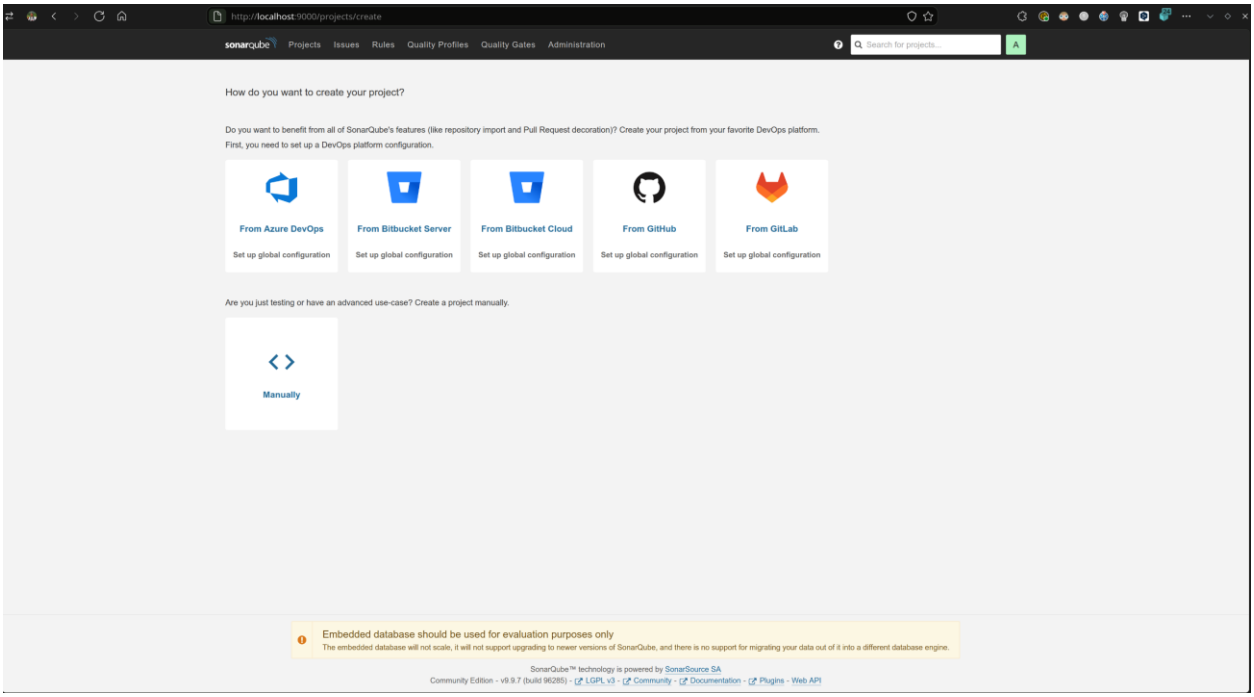


Abbildung 43 – Dashboard in SonarQube

Nachdem «SonarQube» erfolgreich mit «Jenkins» verbunden wurde, kann die Option «Manual» ausgewählt werden. Anschliessend werden der Projektname und der «branch-name» eingegeben, und der Prozess kann über die Schaltfläche «Setup» fortgesetzt werden.

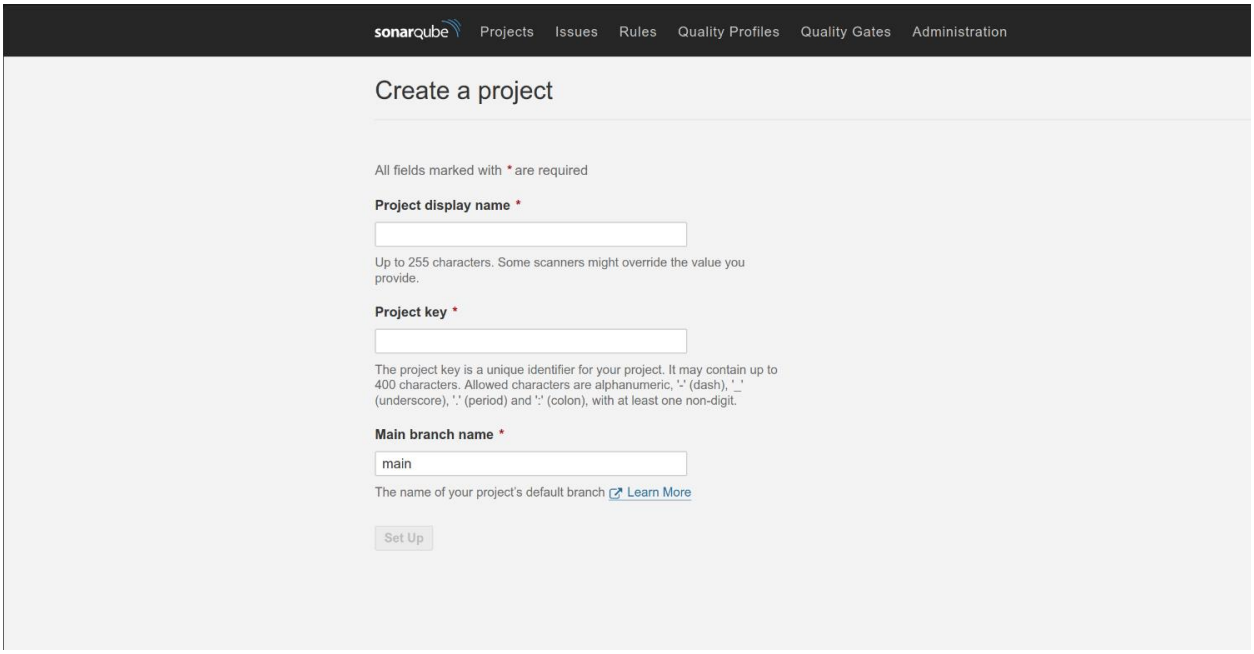


Abbildung 44 – Neue Projekteinrichtung in SonarQube

Wenn die Option «With Jenkins» ausgewählt ist, kann der Prozess konsequent mit «weiter» fortgeführt werden, ohne dass Änderungen erforderlich sind.

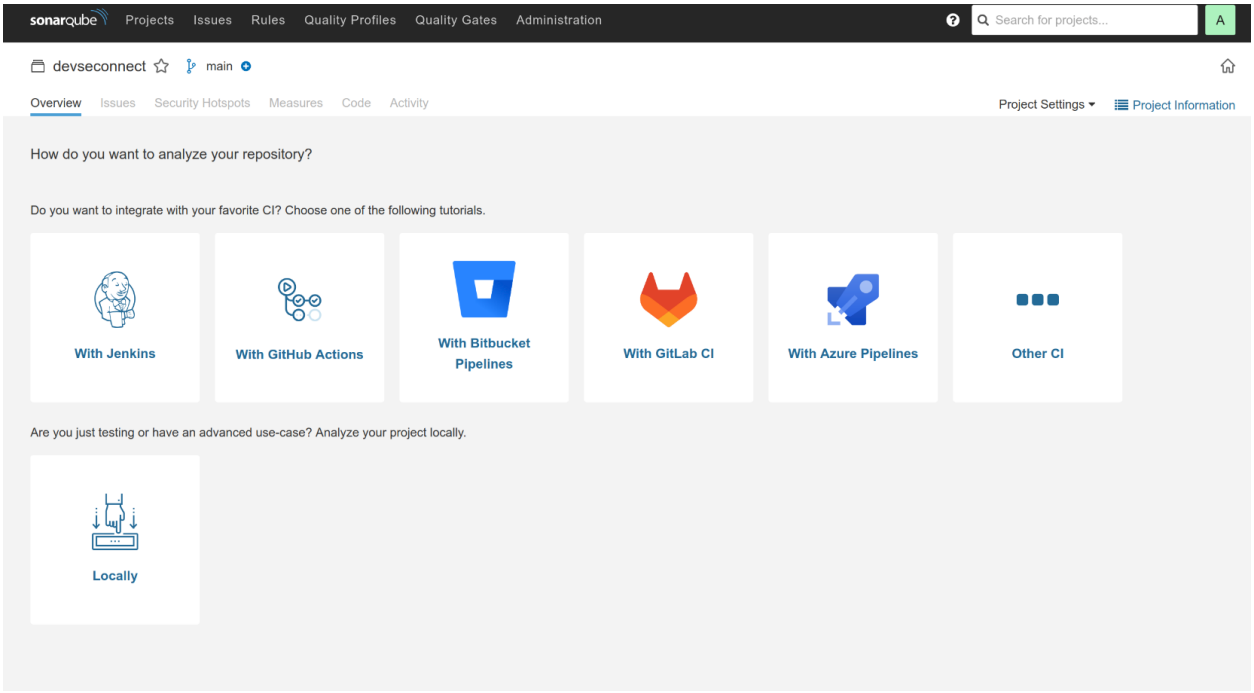


Abbildung 45 – Auswahl des Repositorys in SonarQube

Für die Projektkonfiguration in «Jenkins» wird ein Schlüssel aus der Datei «sonar-project.properties» benötigt. Dieser Schlüssel ist entscheidend für die Verbindung und Analyse mit dem «SonarQub»e Scanner in ««Jenkins. Zum Beispiel kann der Schlüssel wie folgt aussehen: sonar.projectKey=devseconnect. In der «Jenkins-Konfiguration» wird dies unter den «Build Steps» eingetragen, um den «SonarQube-Scanner» auszuführen.

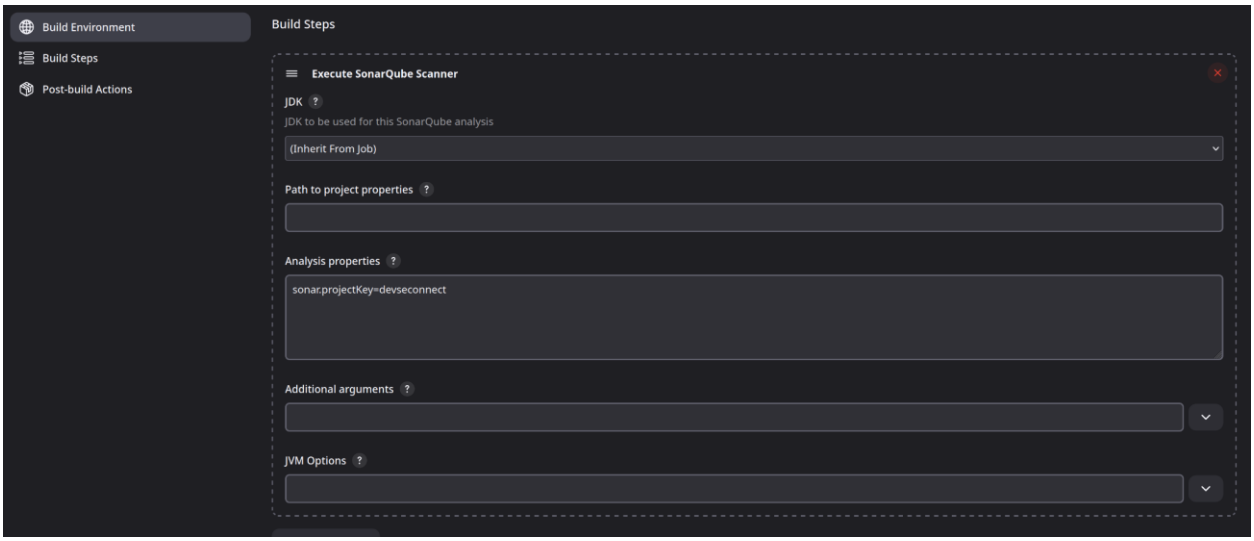


Abbildung 46 – Konfiguration von Jenkins für SonarQube

Die «Jenkins-Konfiguration» ermöglicht erfolgreich den Build-Prozess aus dem «GitHub-Repository». Dies deutet auf eine fehlerfreie Implementierung hin, bei der die Code-Basis automatisch abgerufen und erfolgreich kompiliert wird.

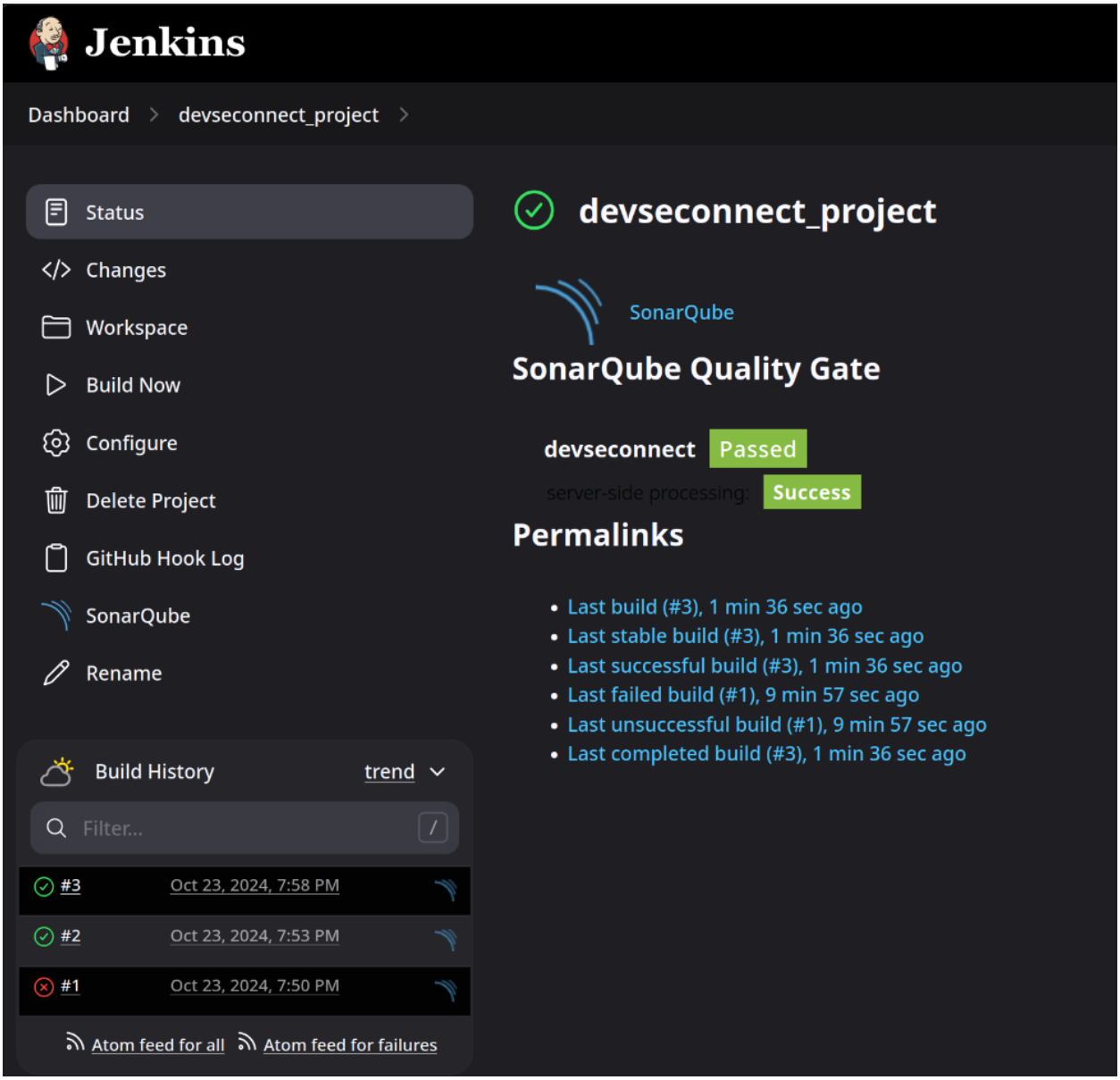


Abbildung 47 – Erfolgsstatus in Jenkins

Die Konsole gibt die Meldung «SUCCESS» aus, was auf einen fehlerfreien Abschluss des Prozesses hinweist. Dies deutet darauf hin, dass die Konfiguration korrekt implementiert wurde und der Build erfolgreich ausgeführt werden konnte.

```
19:58:42.474 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://sonarqube:9000/dashboard?id=devseconnect
19:58:42.474 INFO  Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
19:58:42.474 INFO  More about the report processing at http://sonarqube:9000/api/ce/task?id=AZK6h-om2JOZUsNUAdq2
19:58:42.977 INFO  Analysis total time: 5.293 s
19:58:42.978 INFO  EXECUTION SUCCESS
19:58:42.978 INFO  Total time: 6.041s
Finished: SUCCESS
```

Abbildung 48 – Protokollausgabe im Terminal von Jenkins

«SonarQube» bestätigt, dass der Code in «Golang» und «SvelteKit» fehlerfrei ist und den festgelegten Qualitätsstandards entspricht. Dies weist auf eine korrekte Implementierung hin, die den Code in seiner strukturellen Integrität und Effizienz validiert.

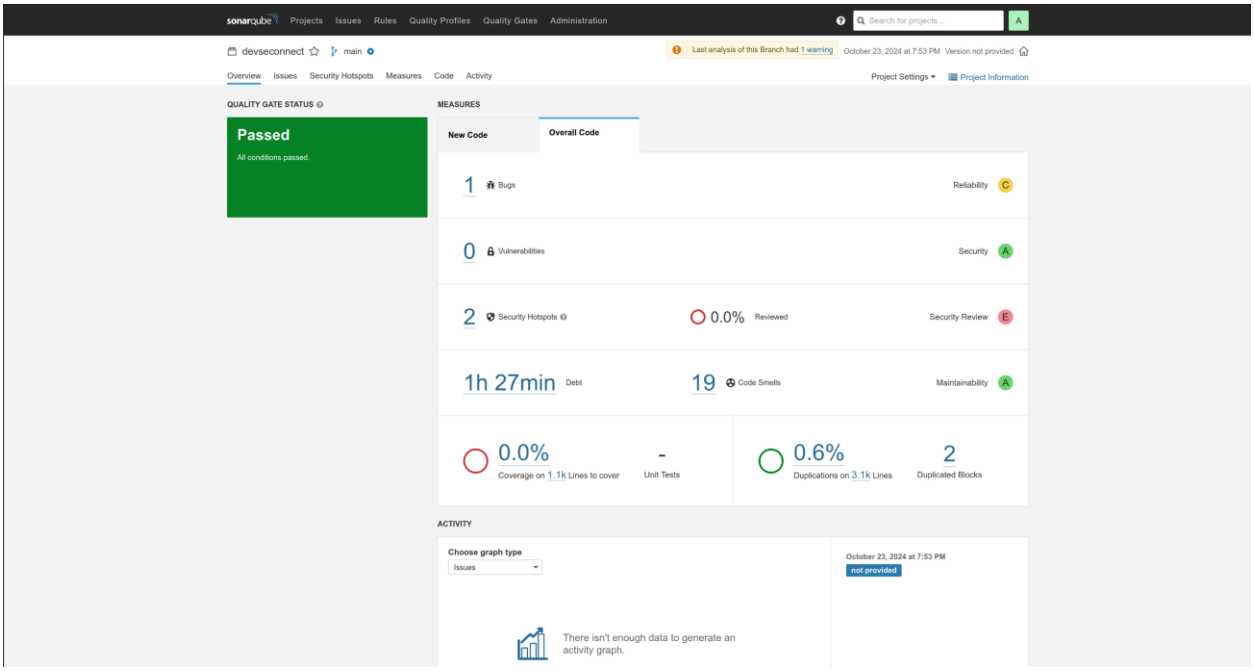


Abbildung 49 – Übersicht des Bewertungsdashboards in SonarQube

Anschliessend kann «Jenkins» einen Docker-Builder ausführen und ein Docker-Image für jede veränderte Version erstellen und automatisch auf Docker Hub bereitstellen.

7.13.3 «K3s» (Kubernetes) und «K9s»-Managementtool

«K3s» ist eine leichtgewichtige, für den Produktionsbetrieb optimierte Kubernetes-Distribution. Sie wurde speziell für IoT-Geräte, Edge-Computing und ressourcenbeschränkte Umgebungen entwickelt. «K3s» ist vollständig kompatibel mit Kubernetes, benötigt jedoch weniger Speicher und CPU-Ressourcen, wodurch es ideal für kleinere Geräte und Umgebungen mit begrenzten Ressourcen ist. Es wird häufig verwendet, um Kubernetes-Anwendungen in lokalen oder Cloud-nahen Umgebungen auszuführen.

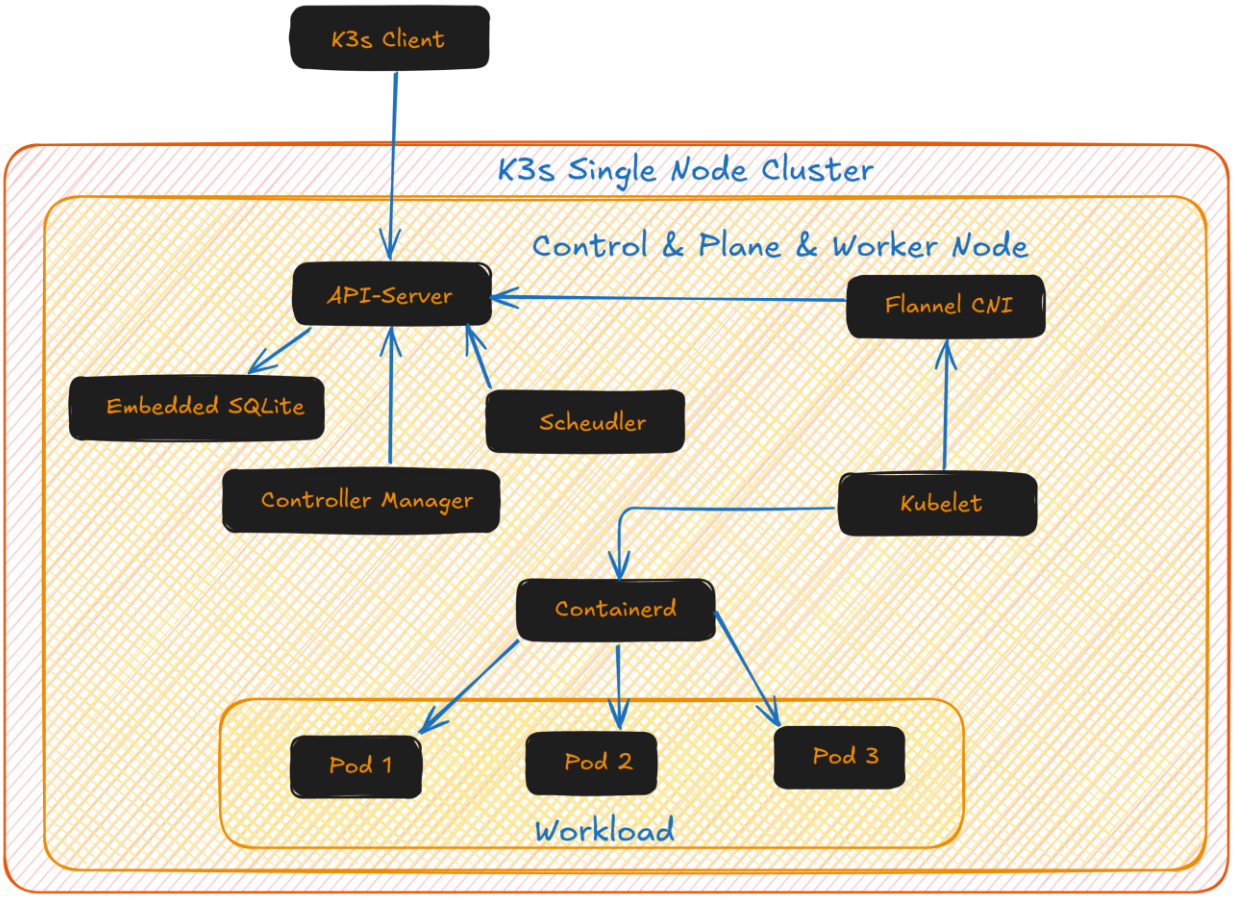


Abbildung 50 - Darstellung der K3s-Architektur

«K9s» stellt eine Open-Source-Terminal-Anwendung dar, welche speziell zur Erleichterung der Arbeit mit Kubernetes-Clustern entwickelt wurde. Es ermöglicht IT-Administratoren und DevOps-Ingenieuren, Kubernetes-Cluster zu überwachen und zu verwalten, ohne dass die Eingabe komplexer Befehle in der Kommandozeile erforderlich ist. Mit «K9s» ist es möglich, Informationen zu Pods, Deployments, Services und Logs abzurufen und auf eine benutzerfreundliche Weise zu visualisieren. Die interaktive Schnittstelle erlaubt eine einfache und effiziente Überwachung von Cluster-Ressourcen in Echtzeit, was insbesondere bei der Fehlerdiagnose und -behebung von grossem Vorteil ist.

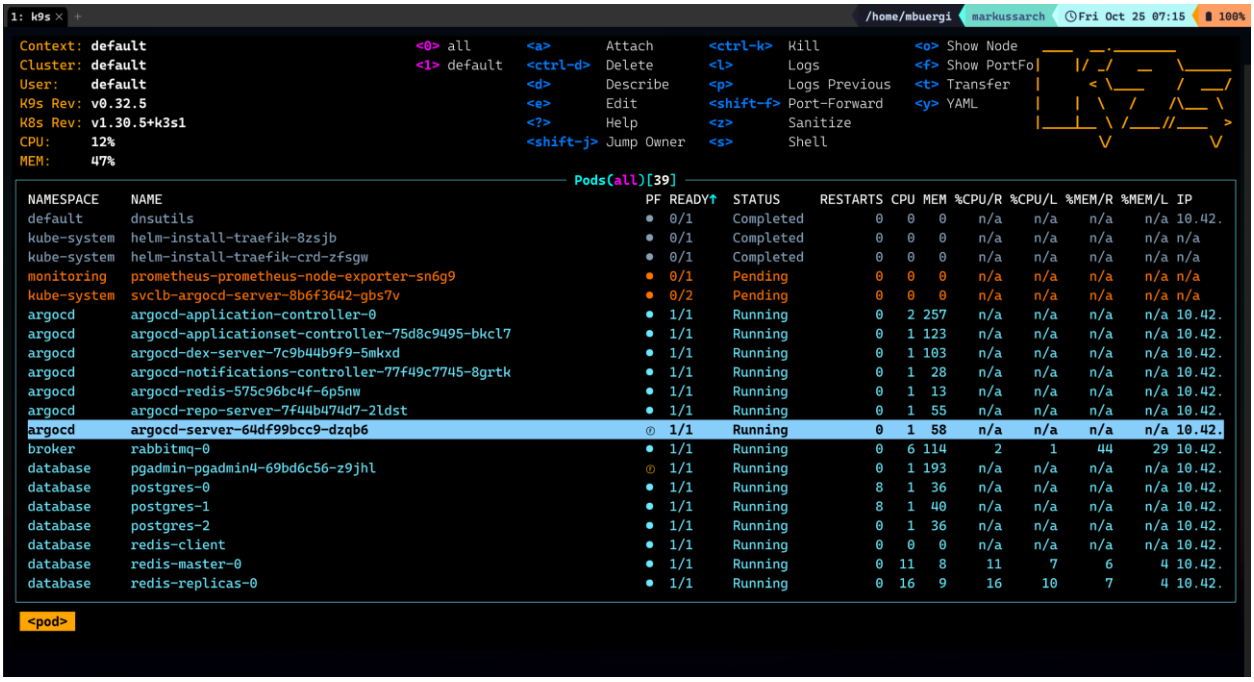


Abbildung 51 – Übersicht des K9s-Managementtools

7.13.4 «ArgoCD»

«Argo CD» ist ein Open-Source-Tool für Continuous Delivery (CD) in Kubernetes-Umgebungen, das auf dem «GitOps-Konzept» basiert. Es synchronisiert Kubernetes-Cluster mit dem gewünschten Zustand, der in einem «Git-Repository» definiert ist, und überwacht kontinuierlich Änderungen. Es bietet eine benutzerfreundliche Weboberfläche zur Überprüfung des Bereitstellungsstatus und unterstützt Tools wie Helm und Kustomize. «Argo CD» ermöglicht auch automatische Rollbacks und Selfhealing bei Abweichungen vom definierten Zustand.

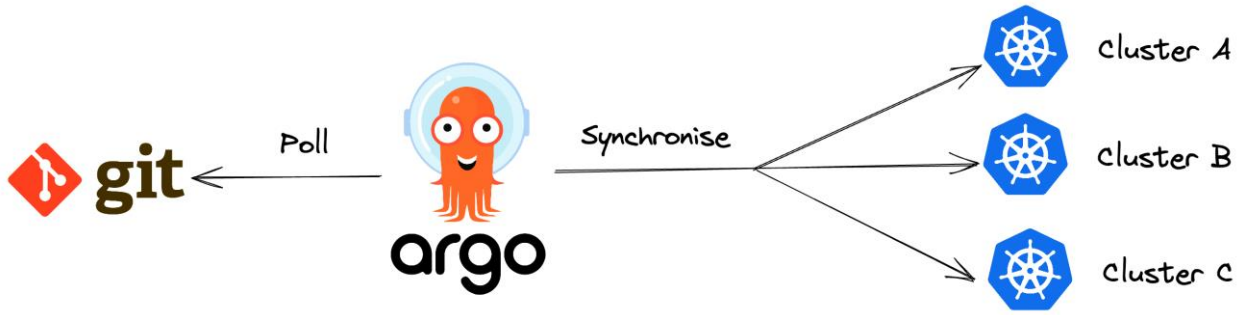


Abbildung 52 – Ansicht von ArgoCD

7.13.4.1 Installation «ArgoCD»

Die Installation von ArgoCD beginnt mit der Einrichtung des Namespace «argocd», um die Ressourcen innerhalb des Clusters klar zu gruppieren und zu organisieren. Dies stellt sicher, dass alle ArgoCD-Komponenten an einem logischen Ort platziert sind und das Management vereinfacht wird. Anschliessend wird der ArgoCD-Server installiert und der eigentliche Installationsprozess fortgeführt, bis ArgoCD vollständig eingerichtet und betriebsbereit ist.

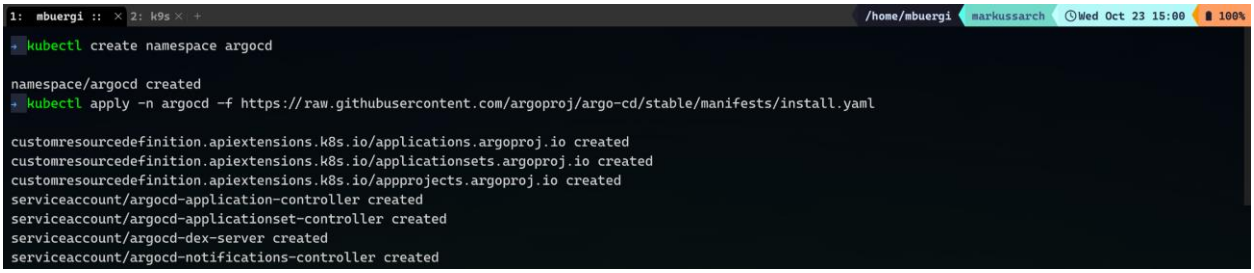


Abbildung 53 – Befehl in ArgoCD

Die Konsole zeigt das initiale Administratorpasswort an, welches kopiert und direkt eingefügt werden kann. Es sollte in einem Passwortmanager sicher gespeichert und nicht auf Papier aufbewahrt werden.

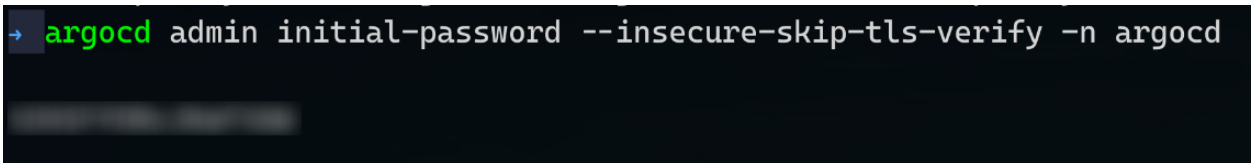


Abbildung 54 – Geheimschlüssel in ArgoCD

An dieser Stelle wird direkt das initiale Administratorpasswort eingefügt.

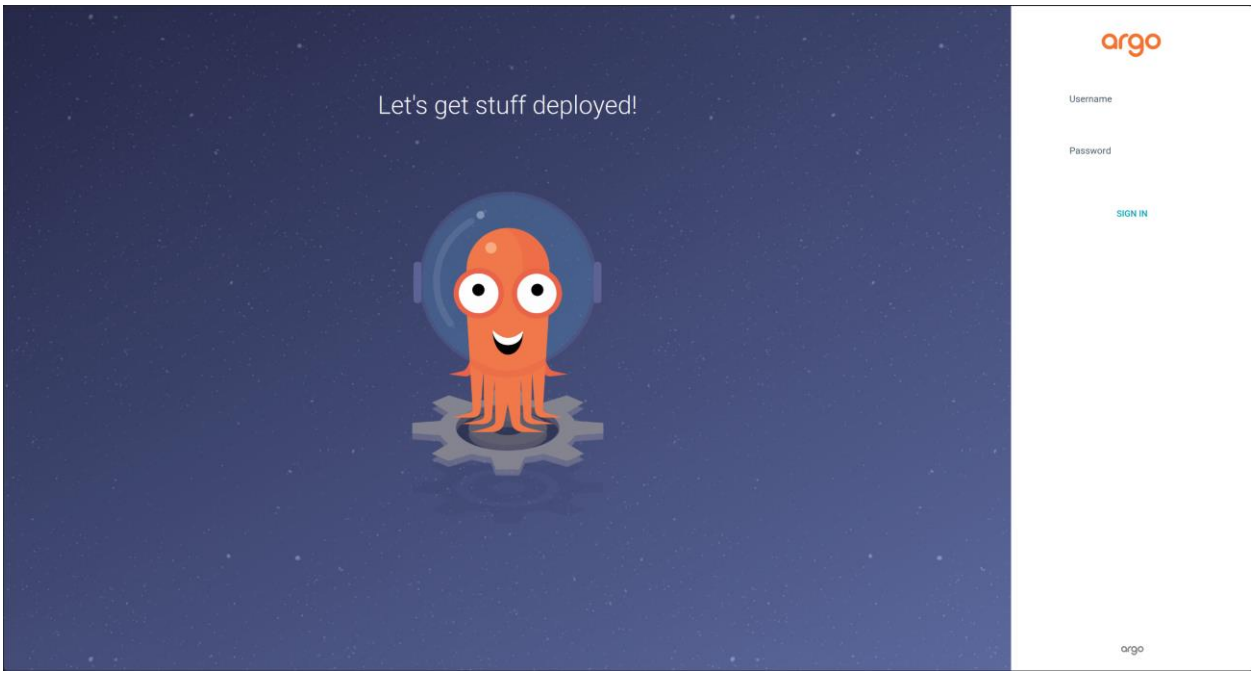


Abbildung 55 – Anmeldung in ArgoCD

7.13.4.2 Erstellung der «ArgoCD-Anwendung»

Anhand dieser Schritt-für-Schritt-Anleitung kann das Erstellen einer neuen ArgoCD-Anwendung ausgeführt werden:

1. **Neues App erstellen:**
   - Zuerst muss in «ArgoCD» auf die Schaltfläche „NEW APP“ geklickt werden, um den Prozess der Erstellung einer neuen Anwendung zu starten (siehe roten Kreis im Bild).
2. **Application Name und Project Name eingeben:**
   - Im erscheinenden Fenster müssen die Felder für „Application Name“ (z.B. „devseconnect“) und „Project Name“ (in diesem Fall standardmässig „default“) ausgefüllt werden (siehe roten Rahmen im Bild).
3. **Repository URL hinzufügen:**
   - Im Bereich „SOURCE“ wird die URL des Repositorys eingetragen, die in diesem Fall https://github.com/mBuergi86/DevSeConnect.git ist.
   - Dabei ist auch der Branch oder die Revision angegeben, die verwendet werden soll, standardmässig ist dies „HEAD“.

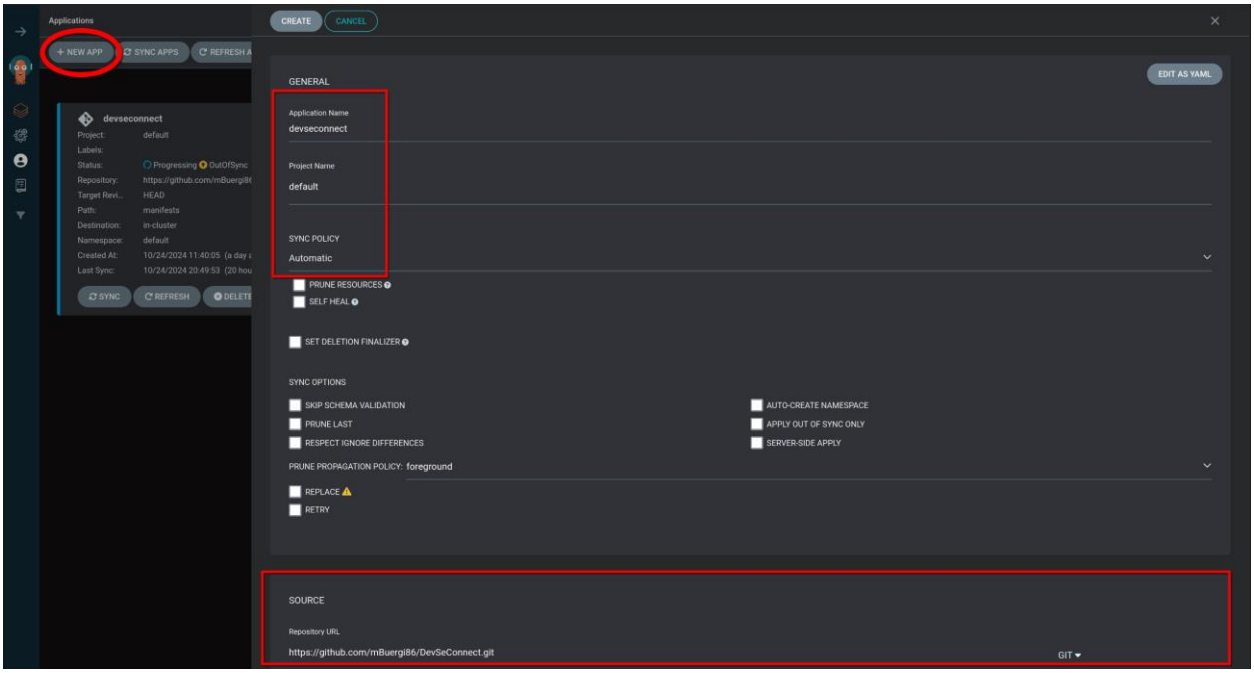


Abbildung 56 - Einstellungen in ArgoCD

4. **Pfad zur Manifestspezifikation:**
    - Geben Sie im Feld „Path“ den Pfad zu den Kubernetes-Manifests ein, der in diesem Fall ./manifests lautet.
5. **Namespace für die Anwendung:**
    - Der Namespace der Anwendung wird unter „DESTINATION“ festgelegt. In diesem Fall ist der Namespace „default“ (siehe roten Rahmen im Bild).

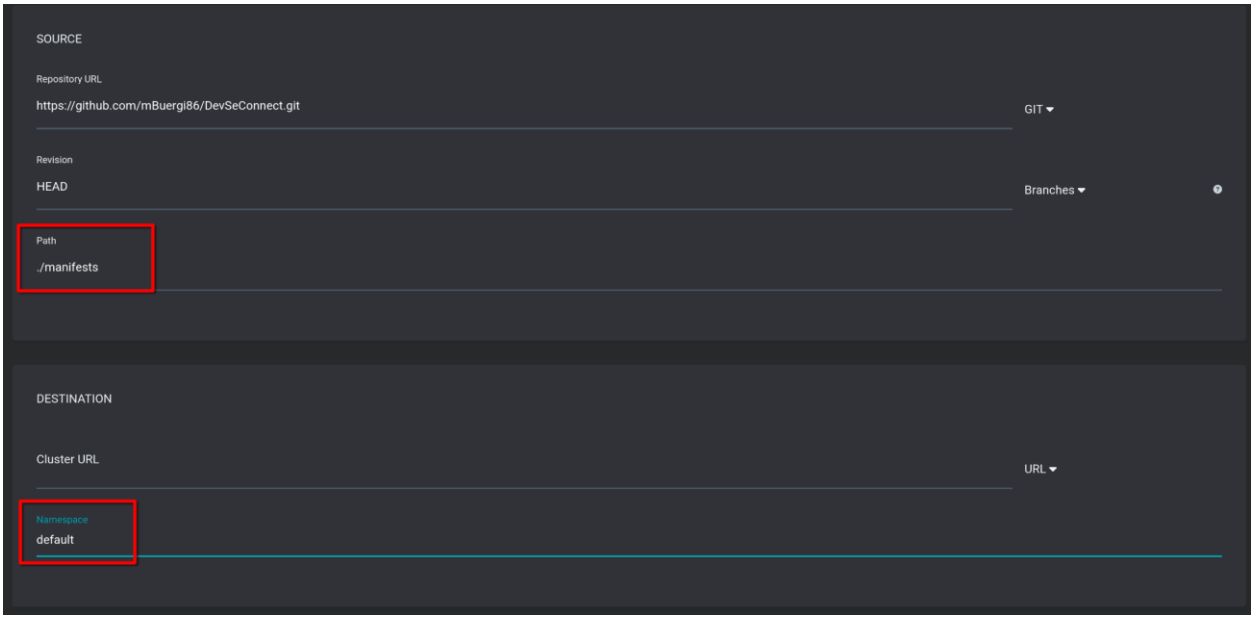


Abbildung 57 – Einstellungen in ArgoCD

6. **Erstellung abschliessen:**
    - Sobald alle Informationen eingegeben wurden, kann die Anwendung durch Klicken auf „Create“ erstellt werden.

7.13.4.3 Synchronisation

Nach der Erstellung der Applikation wird automatisch ein Synchronisationsprozess eingeleitet. Sobald dieser abgeschlossen ist, wird der aktuelle Status der Applikation in der Benutzeroberfläche sichtbar. Dies bedeutet, dass die konfigurierten Ressourcen erfolgreich im Cluster implementiert und aktualisiert wurden.

Die Synchronisation stellt sicher, dass der im Cluster laufende Zustand der Applikation mit den definierten Kubernetes-Manifesten vollständig übereinstimmt und eventuelle Abweichungen behoben werden.

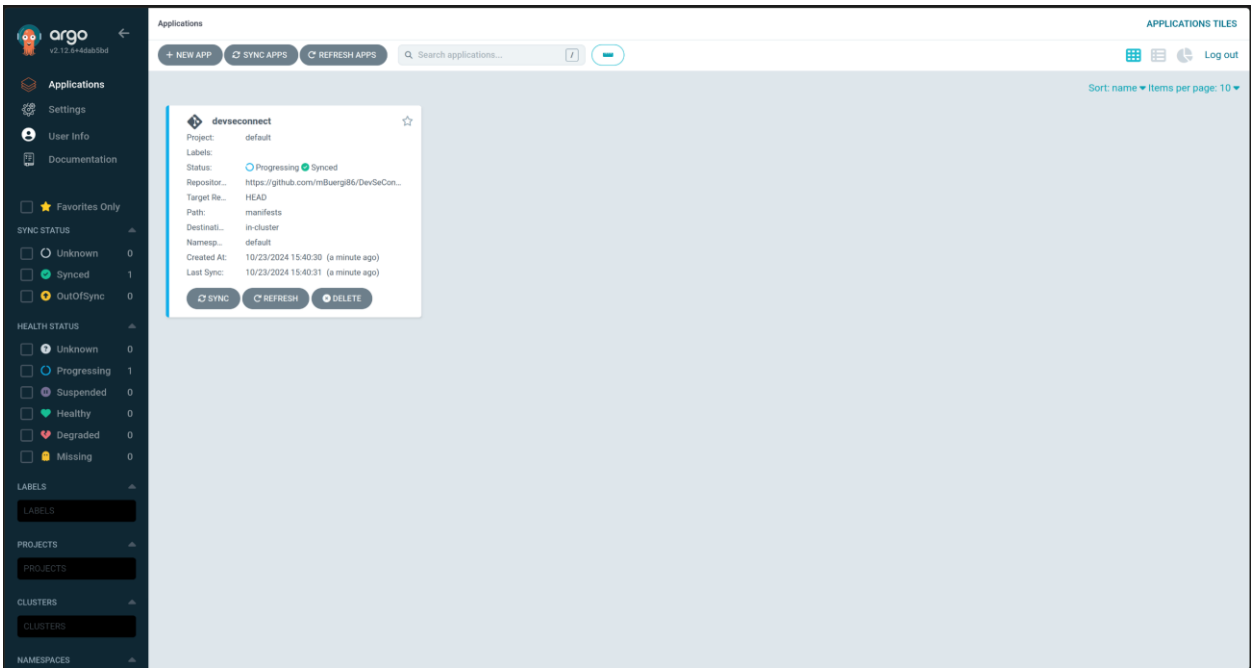


Abbildung 58 – Dashboard von ArgoCD

«ArgoCD» ermöglicht eine automatisierte Verwaltung von Kubernetes-Objekten und bietet eine deklarative Verwaltung der Anwendungszustände. Durch die Implementierung des «GitOps-Prinzips» wird der Bedarf an manuellen Eingriffen erheblich reduziert. Sobald die Kubernetes-Ressourcen, wie Deployments oder Services, in einem «Git-Repository» definiert und von «ArgoCD» überwacht werden, erfolgt eine automatische Synchronisierung dieser Objekte mit dem Kubernetes-Cluster, um die Änderungen im Repository zu berücksichtigen.

Es wird kontinuierlich überprüft, ob der Zustand des Clusters mit den in den Kubernetes-Manifesten definierten Vorgaben übereinstimmt. Bei Abweichungen passt ArgoCD den Cluster automatisch an den gewünschten Zustand an, wodurch manuelle kubectl-Befehle nicht erforderlich sind.

Diese Automatisierung minimiert das Risiko menschlicher Fehler und gewährleistet eine konsistente Verwaltung der Infrastruktur und Anwendungen.

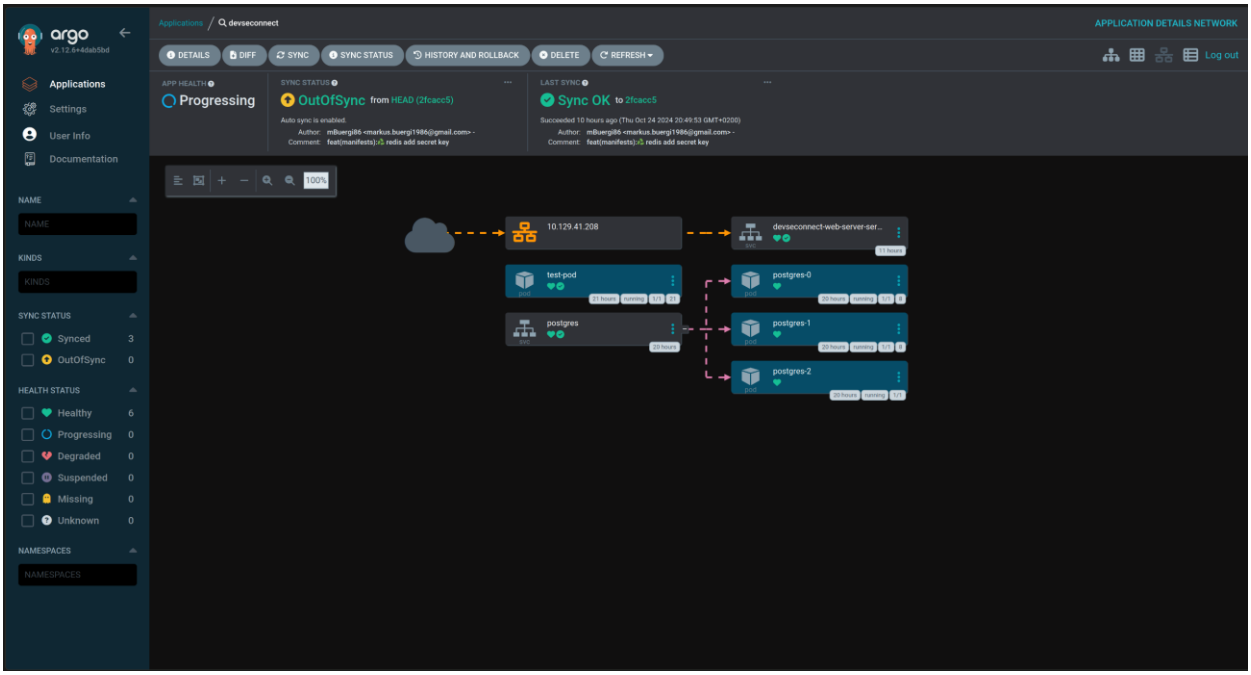


Abbildung 59 – Überwachung in ArgoCD

7.13.5 Traefik

Der Ingress-Controller «Traefik» zeichnete sich durch seine vielseitigen Einsatzmöglichkeiten und die nahtlose Integration in die K3s-Architektur aus. Speziell für die dynamische Verwaltung des Datenverkehrs in Kubernetes-Umgebungen konzipiert, erfüllte «Traefik» in DevSeConnect mehrere zentrale Funktionen zur Optimierung und Absicherung der Netzwerkverwaltung sowie des Datenverkehrs im Cluster. Die folgenden Abschnitte beschreiben die spezifischen Rollen, die «Traefik» übernahm.

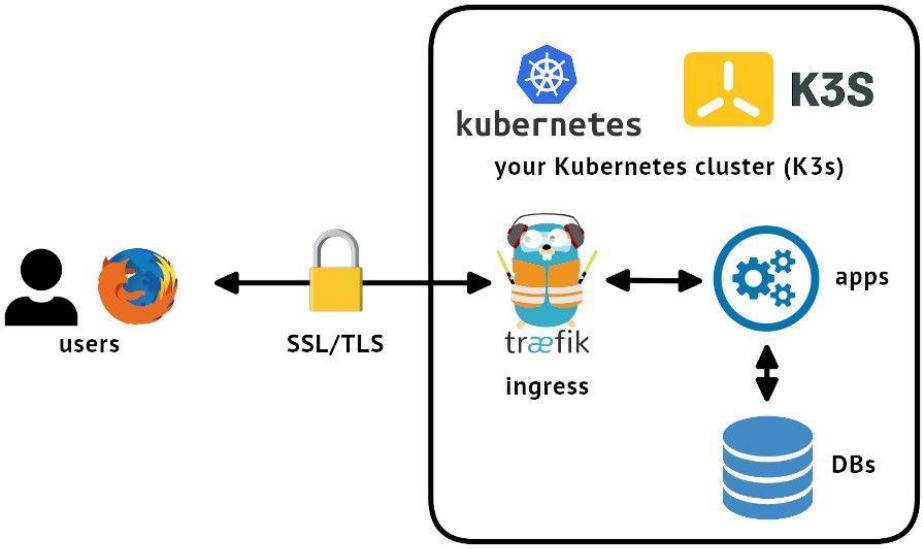


Abbildung 60 – Darstellung von Traefik in der K3s-Umgebung

7.13.5.1 Load Balancer

Traefik fungierte als Load Balancer innerhalb der Kubernetes-Umgebung. Hierbei verteilte er eingehende Anfragen automatisch auf verschiedene Instanzen derselben Anwendung, um die Last effizient zu verteilen und eine hohe Verfügbarkeit sicherzustellen.

YAML-Konfiguration für Load Balancing in einer lokalen Umgebung:

```yaml
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: app-load-balancer
  namespace: default
spec:
  entryPoints:
    - web
  routes:
    - match: Host(`localhost`)
      kind: Rule
      services:
        - name: app-service
          port: 80
  middlewares:
    - name: load-balance
```

Abbildung 61 – Beispielkonfiguration für Load Balancing mit Traefik

7.13.5.2 API-Gateway

In seiner Rolle als API-Gateway diente Traefik als zentraler Zugangspunkt zu den verschiedenen Diensten im Cluster. Er übernahm die Zugriffssteuerung und Routing-Verwaltung, sodass alle Anfragen basierend auf den definierten Routing-Regeln an die entsprechenden Dienste weitergeleitet wurden.

YAML-Konfiguration für das API-Gateway in einer lokalen Umgebung:

```yaml
apiVersion: traefik.containo.us/v1alpha1
kind: Middleware
metadata:
  name: api-gateway
  namespace: default
spec:
  stripPrefix:
    prefixes:
      - /api
```

Abbildung 62 – Beispielkonfiguration für das API-Gateway mit Traefik

7.13.5.3 Ingress-Routing

Traefik übernahm das Ingress-Routing, indem er externe Anfragen mit den Kubernetes-Services verband. Dadurch wurde der Datenverkehr zielgerichtet zu den jeweiligen Diensten weitergeleitet und eine effektive Kommunikation zwischen externen Anfragen und internen Services sichergestellt.

YAML-Konfiguration für Ingress-Routing ohne Domain:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: default
spec:
  rules:
    - host: localhost
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: app-service
                port:
                  number: 80
```

Abbildung 63 – Beispielkonfiguration für Ingress-Routing ohne Domain

7.13.5.4 Sicherheit und Zertifikate

Für eine erhöhte Sicherheit war Traefik in der Lage, SSL-Zertifikate auch lokal zu generieren und zu verwalten. Alternativ zur Nutzung echter Zertifikate kann in einer lokalen Umgebung ein selbstsigniertes Zertifikat erstellt und verwendet werden.

YAML-Konfiguration für SSL-Zertifikate in einer lokalen Umgebung:

```yaml
apiVersion: traefik.containo.us/v1alpha1
kind: TLSOption
metadata:
  name: secure-tls
  namespace: default
spec:
  minVersion: VersionTLS12
  cipherSuites:
    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
tls:
  certResolver: myresolver
```

Abbildung 64 – Beispielkonfiguration für SSL-Zertifikate mit Traefik

In einer reinen lokalen Umgebung ohne Domain kann die Nutzung von Traefik mit localhost-Werten erfolgen. Bei der Sicherheit und SSL-Zertifikaten ist jedoch zu beachten, dass diese meist nur mit selbstsignierten Zertifikaten verwendet werden können, da keine öffentlich validierten Zertifikate für localhost verfügbar sind.

8 PROJEKTABSCHLUSS

8.1 PROJEKTÜBERWACHUNG

In diesem Abschnitt wurde die Projektablaufplanung von Kalenderwoche 42 bis 45 analysiert, wobei Verzögerungen im Backend, Frontend und der CI/CD-Pipeline berücksichtigt wurden. Die CI/CD-Pipeline lief stabil und reibungslos, jedoch mussten im Backend noch unendliche Schleifen kontrolliert und im Frontend weitere Verbesserungen vorgenommen werden.

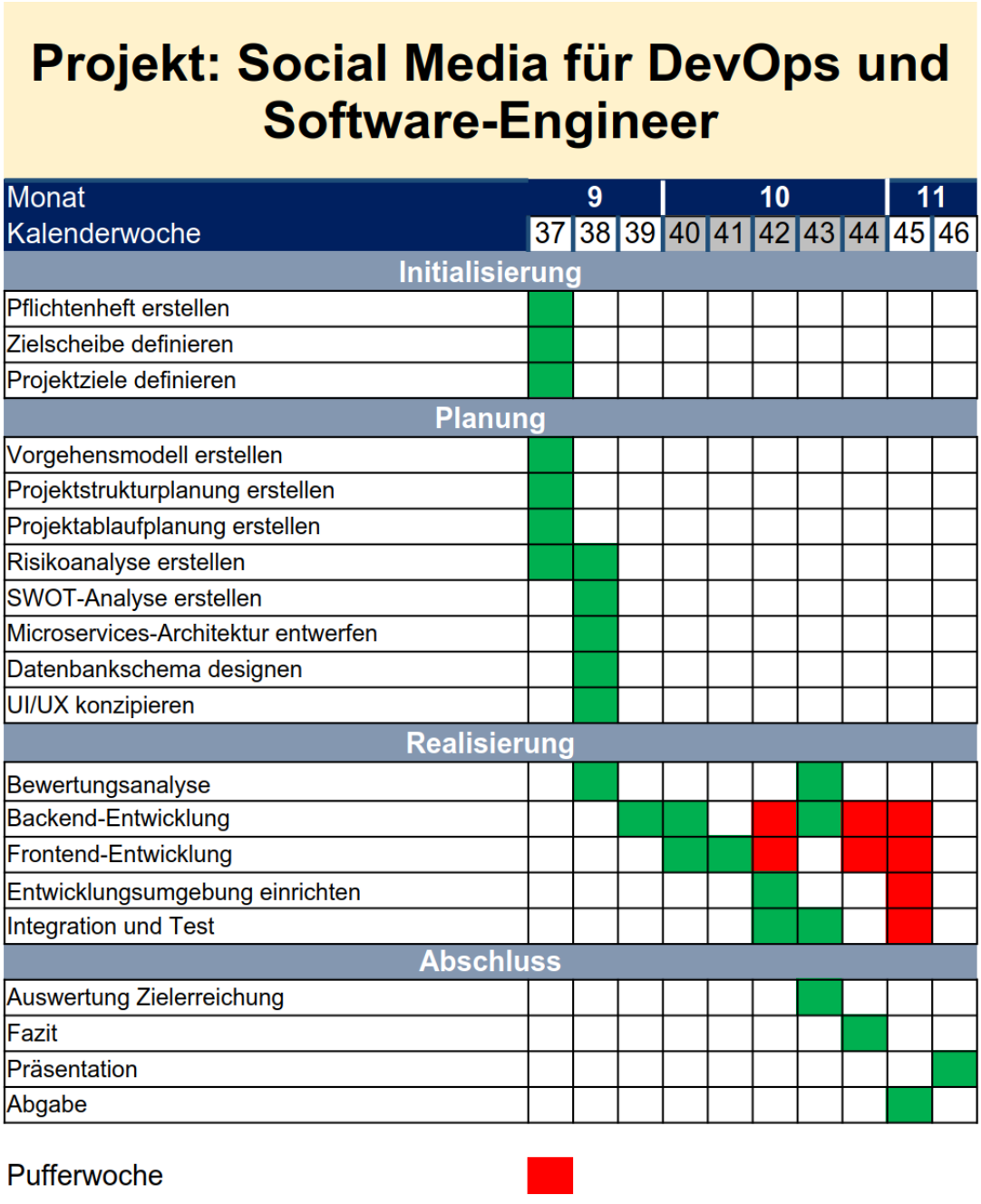


Abbildung 65 – Projektablaufplanung mit Verzögerungen

8.2 Evaluation der Zielerreichung

In dieser Evaluation werden die Endergebnisse mit den festgelegten Erfolgskriterien verglichen, um den Erfüllungsgrad zu ermitteln.

Die Analyse zeigt, dass:

- **Produktionsreife Plattform**: Die Plattform wurde erfolgreich in einer stabilen Umgebung bereitgestellt, jedoch sind einige Kernfunktionen nicht vollständig implementiert, was zu zeitlichen Engpässen geführt hat.
- **Technologische Exzellenz**: Die effektive Nutzung moderner Technologien wie Golang und Kubernetes wurde erreicht, wobei der Fokus stärker auf Backend und CI/CD lag, während das Frontend weniger ausgearbeitet ist.
- **Gemeinschaftsbildung**: Die Etablierung einer aktiven Community steht noch aus, was den Austausch und die Nutzung der Plattform einschränkt.

8.3 Reflexion Weg zum Ziel

Die Reflexion bezieht sich auf den Projektprozess und behandelt folgende Aspekte:

- **Förderliche Begebenheiten**: Die Nutzung von CI/CD-Praktiken und Best Practices haben die Zielerreichung positiv beeinflusst und ermöglicht, die Backend-Entwicklung effizient voranzutreiben.
- **Hinderliche Begebenheiten**: Die zeitlichen Ressourcen wurden durch die Komplexität des grossen Projekts stark beansprucht, was die vollständige Umsetzung aller Ziele erschwerte. Die Kommunikation im Team stellte sich ebenfalls als herausfordernd dar.

8.4 Lessons Learnt

Am Ende eines Projekts bietet eine Reflexion über die Erfahrungen und Erkenntnisse wertvolle Einblicke, die sowohl für zukünftige Projekte als auch für die persönliche Weiterentwicklung hilfreich sind. Diese Rückschau, oft als „Lessons Learnt“ bezeichnet, verdeutlicht Herausforderungen, Erfolge und potenzielle Verbesserungsansätze.

Lessons Learnt:

- **Vertiefung technischer Kenntnisse:**
    - Im Laufe des Projekts konnte ich fundierte Kenntnisse in den Bereichen CI/CD, K3s und Go entwickeln. Die praktische Anwendung dieser

Technologien hat mir ein umfassendes Verständnis ihrer Stärken, Schwächen und besten Einsatzmöglichkeiten vermittelt.

- **Umgang mit Herausforderungen und deren Auswirkungen:**
  - Die Arbeit an der Diplomarbeit brachte zahlreiche herausfordernde Situationen mit sich, die oft ein hohes Mass an Flexibilität und Problemlösungsvermögen erforderten. Diese Hürden wirkten sich teilweise auch auf mein Privatleben aus, was die Bedeutung eines ausgeglichenen Zeitmanagements und einer klaren Abgrenzung zwischen Beruflichem und Privatem unterstrich.
- **Weiterbildung in Golang:**
  - Die Programmierung mit Golang erwies sich als besonders interessant und anwendungsorientiert. Das Projekt weckte mein Interesse, meine Kenntnisse in dieser Sprache weiter zu vertiefen, da sie im Bereich moderner Backend-Lösungen vielseitige Einsatzmöglichkeiten bietet.
- **Stabilität und Zuverlässigkeit von K3s:**
  - K3s zeigte sich als stabil und effizient, insbesondere im Vergleich zu den Technologien, die ich in vorherigen Projekten eingesetzt hatte. Die Zuverlässigkeit von K3s war von grossem Vorteil und bestätigt die Eignung dieser Lösung für kompakte, lokale Kubernetes-Deployments.
- **Zeitmanagement und Workload-Balance:**
  - Der enge Zeitrahmen stellte eine signifikante Herausforderung dar und verlangte mir in den letzten acht Wochen viel ab. Die Fokussierung auf wesentliche Aufgaben war entscheidend, um den Anforderungen gerecht zu werden. Eine längere Planungs- und Testphase hätte jedoch zusätzliche Optimierungsmöglichkeiten eröffnet und den Projektdruck vermindert.

8.5 AUSBLICKE

Im Verlauf eines Projekts oder während der Diplomarbeit entstehen häufig neue, interessante Ideen. Diese können aus zeitlichen, finanziellen oder anderen Gründen nicht immer realisiert werden. Dennoch ist es sinnvoll, diese Ideen und Perspektiven festzuhalten, da sie von anderen aufgegriffen werden oder dem Auftraggeber neue Impulse für zukünftige Projekte geben können.

Ausblick:

- Der Autor würde gerne an diesem Projekt weiterarbeiten, da es viele gute Ideen bietet und viele weitere Möglichkeiten sich ergeben würden, wie GitHub, GitLab, Reddit, LinkedIn usw..
- Für die Weiterentwicklung der Plattform bieten sich unteranderem mehrere Alternativen an, darunter die Implementierung zusätzlicher Kommunikations- und Kollaborationsfunktionen sowie die Optimierung der Infrastruktur für eine grössere Benutzeranzahl.
- Weiterhin ist die Erhöhung der Testabdeckung möglich, um die langfristige Zuverlässigkeit und Stabilität der Plattform sicherzustellen.
- Langfristig könnte auch die Plattform als Open-Source-Projekt zur Förderung der IT-Community bereitgestellt werden.
- In Zukunft, wenn der Autor die Fachhochschule abgeschlossen hat, möchte er nicht mehr an so grossen Projekten arbeiten, sondern sich auf kleinere Projekte konzentrieren, die besser zu seinen Zeitressourcen passen.
- Der Autor fühlt sich bereits kompetent und erfahren in den Bereichen Backend, CI/CD, Containerisierung und Orchestrierung.

9 EIGENSTÄNDIGKEITSERKLÄRUNG

Die Die Verfasserinnen und Verfasser bestätigen mit ihrer Unterschrift, dass die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel erstellt wurde.

Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Inhalte sind als solche kenntlich gemacht.

Die Arbeit ist in gleicher oder ähnlicher Form noch nicht vorgelegt worden.

Unterschriften:	Datum/Ort:
	
Markus Bürgi	05.11.2024 / Egerkingen

10 VERZEICHNISSE

10.1 ABKÜRZUNGSVERZEICHNIS

A

API .....*Application Programming Interface* (Programmierschnittstelle)
AWS..... *Amazon Web Services*

C

CD.....*Kontinuierlicher Bereitstellung*
CI *Kontinuierlicher Integration*
CORS .....Cross-Origin Resource Sharing
CPU .....*Central Processing Unit*
CRUD ..... *Create, Read, Update, Delete*

D

DB.....*Database*
DDD .....*Domain-Driven Design*
DDoS ..... *Distributed Denial of Service*
DevOps.....*Development und Operations*
DSN .....***Data Source Name***

E

E2E ..... *End to End*
EDD .....*Event-Driven Development*
ERD .....*Entity-Relationship-Model*

G

GCP ..... *Google Cloud Platform*

H

HSTS .....*HTTP Strict Transport Security*
HTTP .....*Hypertext Transfer Protocol*
HTTPS ..... *Hypertext Transfer Protocol Secure*

I

ICT ....*Informations- und Kommunikationstechnik*
IT *Informationstechnik*

J

JSON ..... *JavaScript Object Notation*
JWT .....*JSON Web Token*

M

MVP .....*Minimum Viable Product*

O

ORM .....*Object Relational Mapping*

S

SQL .....Structured Query Language
SSG ..... Static Site Generation
SSH .....*Secure Shell*
SSL.....*Secure Sockets Layer*
SSR ..... Server-Side-Rendering

T

TCP .....Transmission Control Protocol
TDD ..... *Test-Driven Development*
TLS ..... Transportschichtsicherheit

U

UI *User Interface (Benutzeroberfläche)*
UML ..... *Unified Modeling Language*
URI ..... *Uniform Resource Identifier*
URL .....*Uniform Resource Locator*
UX..... *User Experience (Benutzererfahrung)*

W

WIP .....*Work in Progress*

X

XSS .....*Cross-Site Scripting Protection*

Z

z.B. .....*zum Beispiel*

10.2 ABBILDUNGSVERZEICHNIS

Abbildung 1 – Kanban des GitHub-Projekts 31
Abbildung 2 – Kanban des GitHub-Projekts 32
Abbildung 3 – Social Media Marketing (Quelle: wearesculpt.com) 39
Abbildung 4 – Design der Startseite 44
Abbildung 5 – Design des Dashboards (Dark-Mode) 45
Abbildung 6 – Design des Dashboards (Light-Mode) 46
Abbildung 7 – Farbkonzept für den Dark-Mode 47
Abbildung 8 – Farbkonzept für den Light-Mode 48
Abbildung 9 – Farbkonzept für den Akzentfarben 49
Abbildung 10 – Farbkonzept für den Funktionsfarben 49
Abbildung 11 – MindMap 51
Abbildung 12 – Monoliths vs Microservices vs Serverless (Quelle: community.ops.io) .52
Abbildung 13 – Monitoring von PostgreSQL in Grafana 75
Abbildung 14 – Verzeichnisstruktur des Domain-Driven Designs 80
Abbildung 15 – GO gibt Echo-Print für Serverstart im Terminal aus 81
Abbildung 16 – Testen von API-Endpunkten mit Postman 100
Abbildung 17 – Anzeige von Fehlermeldungen und Erfolgsnachrichten im Terminal ..101
Abbildung 18 – Darstellung von Prometheus-Metriken in Grafana 104
Abbildung 19 – Monitoring des Dashboard-Menüs in Grafana 105
Abbildung 20 – Monitoring von PostgreSQL in Grafana 106
Abbildung 21 – Monitoring von RabbitMQ in Grafana.....107
Abbildung 22 – Monitoring von Redis in Grafana 108
Abbildung 23 – Startseite des Internets 113
Abbildung 24 – Registrierung des Internets 114
Abbildung 25 – Login des Internets 115

Abbildung 26 – Dashboard des Internets....116
Abbildung 27 – Speicherung des JWT-Tokens in der Webanwendung....116
Abbildung 28 – Profil des Internets....117
Abbildung 29 – Dashboard des Internets....118
Abbildung 30 – Initialpasswort für Jenkins....120
Abbildung 31 – Installation von Jenkins....121
Abbildung 32 – Einrichtung von Jenkins....122
Abbildung 33 – Konfiguration von Jenkins....123
Abbildung 34 – Credentials in Jenkins....124
Abbildung 35 – Globale Credentials in Jenkins....124
Abbildung 36 – Neue Credential-Einrichtung in Jenkins....125
Abbildung 37 – Überblick über Credentials in Jenkins....125
Abbildung 38 – Neuer Eintrag in Jenkins....126
Abbildung 39 – Konfiguration von Jenkins....126
Abbildung 40 – Scan Repository Log in Jenkins....127
Abbildung 41 – Erfolgreicher Pipeline-Aufbau in Jenkins....128
Abbildung 42 – Erfolgreicher Pipeline-Aufbau in Jenkins....129
Abbildung 43 – Dashboard in SonarQube....131
Abbildung 44 – Neue Projekteinrichtung in SonarQube....132
Abbildung 45 – Auswahl des Repositorys in SonarQube....133
Abbildung 46 – Konfiguration von Jenkins für SonarQube....133
Abbildung 47 – Erfolgsstatus in Jenkins....134
Abbildung 48 – Protokollausgabe im Terminal von Jenkins....135
Abbildung 49 – Übersicht des Bewertungsdashboards in SonarQube....135
Abbildung 50 - Darstellung der K3s-Architektur....136
Abbildung 51 – Übersicht des K9s-Managementtools....137

Abbildung 52 – Ansicht von ArgoCD....138
Abbildung 53 – Befehl in ArgoCD....139
Abbildung 54 – Geheimschlüssel in ArgoCD....139
Abbildung 55 – Anmeldung in ArgoCD....140
Abbildung 56 - Einstellungen in ArgoCD....141
Abbildung 57 – Einstellungen in ArgoCD....142
Abbildung 58 – Dashboard von ArgoCD....143
Abbildung 59 – Überwachung in ArgoCD....144
Abbildung 60 – Darstellung von Traefik in der K3s-Umgebung....145
Abbildung 61 – Beispielkonfiguration für Load Balancing mit Traefik....146
Abbildung 62 – Beispielkonfiguration für das API-Gateway mit Traefik....146
Abbildung 63 – Beispielkonfiguration für Ingress-Routing ohne Domain....147
Abbildung 64 – Beispielkonfiguration für SSL-Zertifikate mit Traefik....148
Abbildung 65 – Projektablaufplanung mit Verzögerungen....149

10.3 TABELLENVERZEICHNIS

Tabelle 1 – Ziele....16
Tabelle 2 – Projektablaufplanung....34
Tabelle 3 – Risikoanalyse....35
Tabelle 4 – Risikomatrix....36
Tabelle 5 – SWOT-Analyse....37
Tabelle 6 – Priorisierungsmatrix für Serverarchitektur....57
Tabelle 7 – Präferenzmatrix für Backend-Programmiersprachen....61
Tabelle 8 – Präferenzmatrix für Frontend-Programmiersprachen....62
Tabelle 9 – Nutzeranalyse für Backend-Programmiersprachen....63

Tabelle 10 – Sensitivitätsanalyse für Szenario 1 zur Auswahl der Backend-Programmiersprache 64

Tabelle 11 – Sensitivitätsanalyse für Szenario 2 zur Auswahl der Backend-Programmiersprache 64

Tabelle 12 – Sensitivitätsanalyse für Szenario 3 zur Auswahl der Backend-Programmiersprache 64

Tabelle 13 – Nutzeranalyse für Frontend-Programmiersprachen 65

Tabelle 14 – Sensitivitätsanalyse für Szenario 1 zur Auswahl der Frontend-Programmiersprache 66

Tabelle 15 – Sensitivitätsanalyse für Szenario 2 zur Auswahl der Frontend-Programmiersprache 66

Tabelle 16 – Sensitivitätsanalyse für Szenario 3 zur Auswahl der Frontend-Programmiersprache 67

10.4 DIAGRAMMVERZEICHNIS

Diagramm 1 – Systemarchitektur....18

Diagramm 2 – Datenbankdesign 19

Diagramm 3 – CI/CD-Pipeline des Designs....20

Diagramm 4 – Vorgehensmodell 28

Diagramm 5 – Projektstrukturplanung 33

Diagramm 6 - Informationsarchitektur....43

Diagramm 7 – Sequenzendiagramm 72

Diagramm 8 – Aktivitätsdiagramm....73

Diagramm 9 – ERD (Entity-Relationship-Diagramm)....76

10.5 PROGRAMMIERVERZEICHNIS

Programmierung 1 – SQL-Skripte zur Tabellenerstellung 78

Programmierung 2 – SQL-Skript für Abfragen....79

Programmierung 3 – Backend-Modell für Entität erstellen.....82

Programmierung 4 - Beispiel für ein Backend-Modell einer Entität.....83

Programmierung 5 – Backend-Implementierung für das Repository 84

Programmierung 6 – Backend-Implementierung des Service.....86

Programmierung 7 - Backend-Implementierung des Handlers 88

Programmierung 8 – Backend-Implementierung des Producers für RabbitMQ 90

Programmierung 9 - Backend-Implementierung des Consumers für User in RabbitMQ91

Programmierung 10 – Backend-Implementierung für Redis 94

Programmierung 11 – Backend-Implementierung für die PostgreSQL-Datenbankanbindung.....95

Programmierung 12 – Backend-Implementierung des Routers 97

Programmierung 13 – Backend-Implementierung der Middleware für Logging mit „zerolog“ 97

Programmierung 14 – Backend-Implementierung der Middleware für CORS zur Steuerung von Cross-Origin-Anfragen.....98

Programmierung 15 – Backend-Implementierung der Middleware für Rate Limiting mit Memory Store 99

Programmierung 16 – Backend-Implementierung der Middleware für Sicherheitsmassnahmen in verschiedenen Bereichen 99

Programmierung 17 – Backend-Implementierung des Routers mit Logging von Meldungen.....103

Programmierung 18 – Backend-Implementierung der Authentifizierung mit JWT 110

Programmierung 19 – Backend-Implementierung der Environment 111

Programmierung 20 – Backend-Implementierung der Middleware für JWT 111

Programmierung 21 – Backend-Implementierung des Routers mit Handler-Verknüpfung 111

Programmierung 22 - Backend-Implementierung des Routers mit Handler-Verknüpfung und Authentifizierung.....112

Programmierung 23 – Implementierung der Docker-Compose YAML-Datei für Jenkins 120

Programmierung 24 – Kurze Implementierung eines Jenkinsfiles 127

Programmierung 25 - Implementierung der Docker-Compose YAML-Datei für SonarQube 130

10.6 LITERATUR- UND QUELLENVERZEICHNIS

2024 Stack Overflow developer survey. (o. J.). Stackoverflow.Co. Abgerufen 11. September 2024, von https://survey.stackoverflow.co/2024/

Ali, A. (o. J.). The Folder Structure for Every Golang Project.

Bitnami/postgresql/values.Yaml at main · bitnami/charts. (o. J.).

Budde, K. (o. J.). rabbitmq_exporter: Prometheus exporter for RabbitMQ.

Build images. (2024, Oktober 24). Docker Documentation. https://docs.docker.com/guides/golang/build-images/

DevSeConnect. (o. J.). Figma. Abgerufen 12. September 2024, von https://www.figma.com/design/9Lxq3F6HJwkxQlCHWELihE/DevSeConnect?node-id=25-2&node-type=frame&t=eSZmKu81BhBmht8p-0

Elisabet. (2022, September 11). Argo CD. Wilde-IT; Wilde-IT GmbH. https://www.wilde-it.com/argo-cd/

Gayatripawar. (2023, Dezember 19). Deploy Prometheus and Grafana on Kubernetes using Helm. Medium. https://medium.com/@gayatripawar401/deploy-prometheus-and-grafana-on-kubernetes-using-helm-5aa9d4fbae66

George, N. (o. J.). go-gin-sveltekit-jwt-cookie-example: An example of how to use JWT authentication with Go Gin and SvelteKit using cookies(*inside SvelteKit*).

Go guide. (o. J.). Redis.Io. Abgerufen 18. Oktober 2024, von https://redis.io/docs/latest/develop/connect/clients/go/

Goltsman, K. (2019, April 26). Deploying traefik as ingress controller for your kubernetes cluster. Supergiant.Io. https://medium.com/kubernetes-tutorials/deploying-traefik-as-ingress-controller-for-your-kubernetes-cluster-b03a0672ae0c

Helm. (o. J.). Helm.sh. Abgerufen 16. September 2024, von https://helm.sh/docs

How to design database for social media platform. (2024, Februar 27). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-design-database-for-social-media-platform/

Idogun, J. O. (o. J.). go-auth: A fullstack session-based authentication system using golang and sveltekit.

Idogun, J. O. (2023, Juni 2). Authentication system using Golang and Sveltekit - Initialization and setup. DEV Community. https://dev.to/sirneij/authentication-system-using-golang-and-sveltekit-initialization-and-setup-4oc9

Ilham, S. (2021, Juni 9). Golang REST API with Domain Driven Design. DEV Community. https://dev.to/iamsyahidi/food-app-1005

Implementing JWT authentication in Go. (o. J.). Permify.Co. Abgerufen 16. Oktober 2024, von https://permify.co/post/jwt-authentication-go/

Information architecture: a UX designer’s guide. (o. J.). Justinmind.com. Abgerufen 11. September 2024, von https://www.justinmind.com/wireframe/information-architecture-ux-guide

Introduction. (o. J.). Labstack.com. Abgerufen 19. September 2024, von https://echo.labstack.com/docs

Jenkins global setup & SonarQube. (o. J.). Sonarsource.com. Abgerufen 23. Oktober 2024, von https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/ci-integration/jenkins-integration/global-setup/

Johnson, L. (2022, April 5). Why nearly 80% of Fortune 100 companies rely on Slack Connect to build their digital HQ. Slack. https://slack.com/intl/de-ch/blog/transformation/fortune-100-rely-slack-connect-build-digital-hq

jwt: Go implementation of JSON Web Tokens (JWT). (o. J.).

K3S Single-Node installation (without HA). (o. J.). Expertflow.com. Abgerufen 25. Oktober 2024, von https://docs.expertflow.com/cx/4.3/k3s-single-node-installation-without-ha

Konissi, L. (2024, März 17). Integrating SonarQube with Jenkins - Liliane konissi. Medium. https://medium.com/@lilnya79/integrating-sonarqube-with-jenkins-fe20e454ccf4

Krakauer, J. (2022, Dezember 28). The Social Media Marketing to Developers Guide (B2D). Wearesculpt.com. https://wearesculpt.com/blog/developer-social-media-marketing/

Kroki! (o. J.). Kroki.Io. Abgerufen 18. September 2024, von https://kroki.io/

Landon, J. (2024, Januar 24). A guide to images in SvelteKit. Sveltekit.Io. https://svelte-kit.io/blog/sveltekit-images

Minikube start. (o. J.). Minikube. Abgerufen 17. September 2024, von https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2Fchocolatey

Mishra, A. (2024, August 28). How to efficiently check if a username exists among billions of users. Medium. https://medium.com/@aditimishra_541/how-to-efficiently-check-if-a-username-exists-among-billions-of-users-7ed1e5c60489

Monolith vs Microservices vs Serverless-Architektur für Ihre nächste App-Anwendung? (o. J.). Kruschecompany.com. Abgerufen 22. September 2024, von https://kruschecompany.com/de/monolith-vs-microservices-vs-serverless/

Odabasi, M. (2023, Februar 2). Installing ArgoCD on Minikube and deploying a test application. Medium. https://medium.com/@mehmetodabashi/installing-argocd-on-minikube-and-deploying-a-test-application-caa68ec55fbf

Olawanle, J. (2023, März 28). Die 3 besten Generatoren für statische Websites für maximale Leistung. Kinsta®; Kinsta. https://kinsta.com/de/blog/svelte-statischer-seiten-generator/

Online FlowChart & diagrams editor - mermaid live editor. (o. J.). Mermaid.Live. Abgerufen 18. September 2024, von https://mermaid.live/edit

Roy, H. (2021, September 20). Bootstrap K3S data: For beginners. Dzone.com; DZone. https://dzone.com/articles/bootstrap-k3s-data-for-beginners

State of JavaScript 2023. (o. J.). Stateofjs.com. Abgerufen 21. September 2024, von https://2023.stateofjs.com/en-US/libraries/

Traefik Kubernetes Gateway API documentation - traefik. (o. J.). Traefik.Io. Abgerufen 31. Oktober 2024, von https://doc.traefik.io/traefik/providers/kubernetes-gateway/

Traefik kubernetes ingress documentation - traefik. (o. J.). Traefik.Io. Abgerufen 31. Oktober 2024, von https://doc.traefik.io/traefik/providers/kubernetes-ingress/

UX Design. (o. J.). Opten.ch. Abgerufen 11. September 2024, von https://www.opten.ch/de/angebot/ux-design/

Verkhovskii, D. (2023, Dezember 22). SonarQube with Docker compose: complete tutorial. Medium. https://medium.com/@denis.verkhovsky/sonarqube-with-docker-compose-complete-tutorial-2aaa8d0771d4

Vinto, N., & Bueno, A. (2023). Gitops Cookbook: Kubernetes Automation in Practice. O'Reilly Media.

What is Jenkins? (2021, März 7). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-jenkins/

Wikipedia contributors. (o. J.). SonarQube. Wikipedia, The Free Encyclopedia. https://de.wikipedia.org/w/index.php?title=SonarQube&oldid=248828105

(O. J.). Iamondemand.com. Abgerufen 18. Oktober 2024, von https://iamondemand.com/wp-content/uploads/2019/11/Kubernetes-eBook.pdf

11 ANHANG

11.1 PROJEKTSTATUSBERICHTE

Projekt: **Stautsbericht:**

Projektleiter	**Projektziele**	**Verteiler**
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban

Gesamt-beurteilung	**Projektverlauf**	**Projektklima**	**Termine**	**Risiken**	**Ressourcen**
	■ □ □	□ ■ □	■ □ □	□ ■ □	□ ■ □
Tendenz	⇧	⇨	⇧	⇨	⇘

Aktueller Projektstand	**Was läuft gut?**
• Initialisierung • Planung	• Es ist wichtig, die vorgegebenen Zeiten und Termine einzuhalten. **Was läuft nicht gut?** • Noch weniger Erfahrung mit dem Konzept für UX/UI.

Geplante nächste Schritte / getroffene Massnahmen

- Realisierungen

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	**Projektziele**	**Verteiler**
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban

Gesamt-beurteilung	**Projektverlauf**	**Projektklima**	**Termine**	**Risiken**	**Ressourcen**
	☐ ■ ☐	☐ ☐ ■	■ ☐ ☐	☐ ■ ☐	☐ ■ ☐
Tendenz	↗	↘	↑	→	↘

Aktueller Projektstand
- Monitoring eingerichtet
- Datenbank eingerichtet und implementiert
- Backend GO als die Grundlage implementiert

Was läuft gut?
- Monitoring läuft gut.
- Datenbank läuft gut.
- Meilenstein läuft gut.

Was läuft nicht gut?
- Ich bin geringer motiviert.

Geplante nächste Schritte / getroffene Massnahmen
- Backend GO implementieren

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	**Projektziele**	**Verteiler**
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamt-beurteilung	**Projektverlauf**	**Projektklima**	**Termine**	**Risiken**	**Ressourcen**
	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐
Tendenz	↘	→	→	↗	↓

Aktueller Projektstand
- Bewertungsanalyse verschiedene Methode analysiert und dokumentiert
- Sequenzdiagramm und Aktivitätsdiagramm zwischen Frontend und Backend inkl. Datenbank und Broker Management
- Operation System (OS) gewechselt und komplette Infrastruktur aufgebaut

Was läuft gut?
- Bewertungsanalyse
- UML (Sequenzdiagramm und Aktivitätsdiagramm)

Was läuft nicht gut?
- Windows 11 ist abgestürzt und kaputt
- Kubernetes Dateien komplett verloren
- Arch Linux von Windows 11 verschoben

Geplante nächste Schritte / getroffene Massnahmen
- Backend GO weiter implementieren
- Monitoring, RabbitMQ und PostgreSQL in Docker implementieren
- Arch Linux weiter anpassen

Projekt-Statusbericht; Stefan Thöni, Josef Räber

TEKO Schweizerische Fachschule

Projekt: **Stautsbericht:**

Projektleiter	Projektziele	Verteiler
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamt-beurteilung	Projektverlauf	Projektklima	Termine	Risiken	Ressourcen
	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐	☐ ■ ☐
Tendenz	⇗	⇨	⇨	⇨	⇨

Aktueller Projektstand

- Backend Golang implementiert mit RabbitMQ und Redis
- PostgreSQL, Redis, RabbitMQ, Grafana, Prometheus in Docker Compose eingerichtet und ausgeführt
- Enivorment erfassen

Was läuft gut?

- Golang Backend kann ich mit der Implementierung fortfahren

Was läuft nicht gut?

- Im Golang-Backend muss ich viele Tabellen erstellen. Ich habe zu wenig Zeit, weil ich bald zum Frontend wechseln muss.
- Authentifizierung bis jetzt nicht implementiert.

Geplante nächste Schritte / getroffene Massnahmen

- Implementierung des Frontends mit Svelte

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	Projektziele	Verteiler
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamtbeurteilung	Projektverlauf	Projektklima	Termine	Risiken	Ressourcen
	☐ ☐ ■	☐ ☐ ■	☐ ■ ☐	☐ ■ ☐	☐ ☐ ■
Tendenz	⇨	⇨	⇨	⇨	⇨

Aktueller Projektstand

- Die Startseite der Darstellung, der Login der Darstellung, das Register der Darstellung sowie der API-Server von Svelte sind implementiert.

Was läuft gut?

- Svelte ist soweit gut

Was läuft nicht gut?

- Svelte hat eine neue Version und eine ganz andere Semantik. Ich bin sehr im Rückstand.
- Ich persönlich stecke gerade in einer schwierigen Lage, nicht wegen das Projekt.

Geplante nächste Schritte / getroffene Massnahmen

- Svelte für das Dashboard und die API implementiert. Dies umfasst die Umsetzung für das Register, den Login und das Dashboard mit WebSocket aus GET.

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	Projektziele	Verteiler
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamtbeurteilung

	Projektverlauf	Projektklima	Termine	Risiken	Ressourcen
Tendenz	⇗	⇨	⇨	⇨	⇨

Aktueller Projektstand

- Svelte verwendet API-Fetch, um Daten abzurufen, Script programmieren, und CSS für das Layout und die Farben.
- Golang sorgt mit JWT und XSS für Sicherheit.
- Svelte stellt sicher, dass der Fetch-Aufruf korrekt Daten überträgt.

Was läuft gut?

- Die Webseite unter Svelte scheint in Ordnung zu sein. Sie ermöglicht eine funktionierende Kommunikation für Registrierung und Login zwischen dem Webserver, der Datenbank und dem Cache.

Was läuft nicht gut?

- Die Webseite ist leider unvollständig, da ich nicht alle Anforderungen erfüllen konnte. Das Zeitmanagement geriet unter Druck, mehr als erwartet.

Geplante nächste Schritte / getroffene Massnahmen

- CI/CD einrichten und Dokumentation

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	Projektziele	Verteiler
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamtbeurteilung

	Projektverlauf	Projektklima	Termine	Risiken	Ressourcen
Tendenz	⇗	⇨	⇨	⇨	⇨

Aktueller Projektstand

- Jenkins und SonarQube installiert und eingerichtet
- K3s wurde eingerichtet und gesichert.
- K9s ist installiert und für Pod-Management konfiguriert.
- Helm Chart eingerichtet und verschiedene Images wie Postgres, RabbitMQ, Redis, Grafana, Prometheus, und Exporter hochgeladen und installiert.
- Exporter an Grafana und Prometheus angepasst.
- ArgoCD installiert und auf GitHub-Repository zugreifend, um automatisiert einzurichten.

Was läuft gut?

- K3s läuft stabil und wie geplant.
- Monitoring und Automatisierung funktionieren ohne Probleme.

Was läuft nicht gut?

- Aktuell gibt es keine bekannten Probleme.
- Nur Golang und SvelteKit benötigen aktuell noch Refaktorierung, um die Funktionalität und Struktur zu verbessern.

Geplante nächste Schritte / getroffene Massnahmen

- Dokumentation

Projekt-Statusbericht; Stefan Thöni, Josef Räber

Projekt: **Stautsbericht:**

Projektleiter	Projektziele	Verteiler
Markus Bürgi	Social Media für DevOps und Software-Engineer	• Zuber Suban • Graziano Felline

Gesamtbeurteilung

	Projektverlauf	Projektklima	Termine	Risiken	Ressourcen
Tendenz	↗	↑	↘	↑	↓

Aktueller Projektstand	Was läuft gut?
• Die Dokumentation erstellt, überprüft und angepasst • Golang-Code refaktoriert und getestet	• Die Dokumentation ist fertiggestellt. **Was läuft nicht gut?** • Golang läuft zwar besser, funktioniert jedoch in Kubernetes nicht ganz reibungslos – in Docker hingegen läuft es problemlos. Dies führt zu einem erhöhten Zeitaufwand. • SvelteKit muss noch implementiert werden.

Geplante nächste Schritte / getroffene Massnahmen

Projekt-Statusbericht; Stefan Thöni, Josef Räber

11.2 MEETINGPROTOKOLLE

20.09.2024

In der Besprechung mit Graziano Felline wurde eine Analyse der Bewertung vorgenommen. Der Fokus lag auf dem Backend und CI/CD. Es stellte sich heraus, dass diese Aspekte im Vergleich zum Frontend im MVP ausreichend waren.

18.10.2024

In einer weiteren Besprechung mit Graziano Felline wurden Fehler im CI/CD-Prozess erörtert. Es wurde festgestellt, dass das Frontend in einer schwierigen Lage war. Als Konsequenz wurde beschlossen, die Arbeiten am CI/CD-Prozess fortzusetzen.