Praxis der Softwareentwicklung

Karlsruher Institut für Technologie

Institut für Telematik

Lehrstuhl Prof. Dr. Martina Zitterbart

Modern Messaging Platform: ChatSphere

Implementierungsbericht

Alexander Wank Niklas Seyfarth

Julien Midedji Berthold Niemann

Alexander Brese

betreut von: Tim Gerhard und Markus Jung

16. September 2018

Inhaltsverzeichnis

1. Einführung

1.1. Absicht

Der Implementierungsbericht hält den Fortschritt der Implementierung und Abweichungen zu den Zielvorgaben des *Entwurfsdokuments* und des *Pflichtenheft* fest.

1.2. Systemorganisation

Das Projekt *ChatSphere* wird in 5 voneinander getrennten und zum Ende der Implementierung miteinander integrierten Komponenten realisiert.

- **Frontend** - Der im Webbrowser laufende *JavaScript-Anwendung*.
- **API** - Die *GraphQL Schema Definition* für die Kommunikation zwischen Frontend und Backend.
- **Backend** - Ein *Java-Servlet* welches einen API-Endpoint bereitstellt, unterteilt in 3 Modulen, welche als ein gemeinsamen Artefakt ausgeliefert werden.
  - **Hydra** - Ein API-Client für Abfragen am Authorisierungsserver
  - **Chatsphere-Database** - Die Datenbankmodelle mit zugehörigen Integrationstests
  - **Chatsphere-Server** - Die Implementierung des → GraphQL API-Endpoint.
- **Datenbank** - Die Speicherrepräsentation in einer *MariaDB* Datenbank.
- **Third-Party** - Authorisierungsserver *Ory Hydra* und WebServer sowie Reverse-Proxy *Caddyserver*.

2. Planung

2.1. Aufwandeinschätzung

Der Aufwand kann im wesentlichen unterteilt werden in *Environment*-Entwicklung und *Application*-Entwicklung. Aufgrund unseres Applikation-Stacks und sehr neuen Technologien, ist es erforderlich, dass wir unsere eigene Programmumgebung erstellen und pflegen, während für verbreitete und bekannte Technologie-Stacks schnell ausführliche Frameworks den ersten Teil der Arbeit hätten abnehmen können.

Ein Entscheidender Dritter Teil, der im Grunde zur *Testphase* gehört, aber im größeren Umfang bedingt durch die Ergebnisse der ersten beiden Teile auch vorgezogen wurde, ist die Orchestration mit zugehörigem Refactoring und Stabilisierung. Die Vorbereitungen hierfür wurden bereits vor den anderen beiden Phasen angefangen.

- **Enviroment** - Die Entwicklung von Core-Komponenten und der Zusammenschaltung von → JETTY , → GraphQL , → RxJava und → ORMLite im Backend. Das anknüpfen zwischen → ServiceWorker , → Apollo und → VueJS im Frontend

- **Application** - Die Implementierung der → Resolver und → Repository im Backend sowie im Frontend die Implementierung der *UI-Komponenten* und *GraphQL-Queries*.

- **Orchestration** - Das aufsetzen und Testen der vollständigen Laufzeitumgebung gemäß dem Architekturplan.

Der Aufwand kann für alle 3 Teile als ungefähr gleich Umfangreich angesehen werden.

2.2. Verzögerungen

Mit der *Orchestration* wurde bereits in der Entwurfsphase durch das Aufsetzen der → GITLab Pipeline angefangen, welche über → CD stetig auf einen *Testserver* ausliefert und die Artefakte generiert einschließlich der Binaries, Dokumentation und Reports. Es werden durchgehend Bedarfsgerecht Ergänzungen durchgeführt.

Aufgrund eines mehrwöchigen Einstiegs im *Environment*-Abschnitt zum zusammensetzen eines Backend-Frameworks kam es zu Verzögerungen bei der Implementierung der eigentlichen *Applikation*, da dadurch die ersten Code-Samples bereits die Anforderung an die Integration in die Architektur erfüllen mussten. Dadurch kam es zu Verzögerungen beim initialen Entwicklungsstart.

Die sehr sinnvolle Entwicklung von *Pilot-Prototypen* die auch das Frontend und dessen Architektur umfassen und schließlich im finalen *ChatSphere*-Projekt als Pilotbeispiele enthalten blieben, erforderten einen nicht einkalkulierten Aufwand. Inwiefern das Entwickeln jedoch die spätere Implementierung der eigentlich entworfenen *Applikation* beschleunigt hat, lässt sich nicht mit Gewissheit sagen, allerdings wird der Mehrwert im Team sehr geschätzt.

Die *Applikation*-Entwicklung begann dadurch erst Ende August, da zuvor die Implementierung des Entwurfs großteils auf theoretischer Basis stattfand und erst zu diesem Zeitpunkt die einzelnen Komponenten, außerhalb des Rahmens der Pilotbeispiele, zunehmend Lauffähig wurden. Da sich inzwischen neue Erkenntnisse boten, bremsten Diskussionen über den einst festgelegten Entwurf bzw. auch über fehlende Komponenten in diesem, die anfängliche Entwicklung.

Aufgrund von langfristiger Vorbereitung für Prüfungen und in der vorlesungsfreien Zeit vermehrten Aktivitäten außerhalb des Universitätswesen sowie Krankheiten konnte das Entwicklungstempo, welches während des Semesters vorlag, nicht gehalten werden, wodurch es zu unabsehbaren Überschreitungen der Zielvorgaben kam. Wenn ignoriert werden würde, dass mit zunehmendem Projektverlauf auch immer wieder neue Issues erstellt werden, würde das Projekt demnach in den ersten Oktober-Wochen fertigstellt werden können:

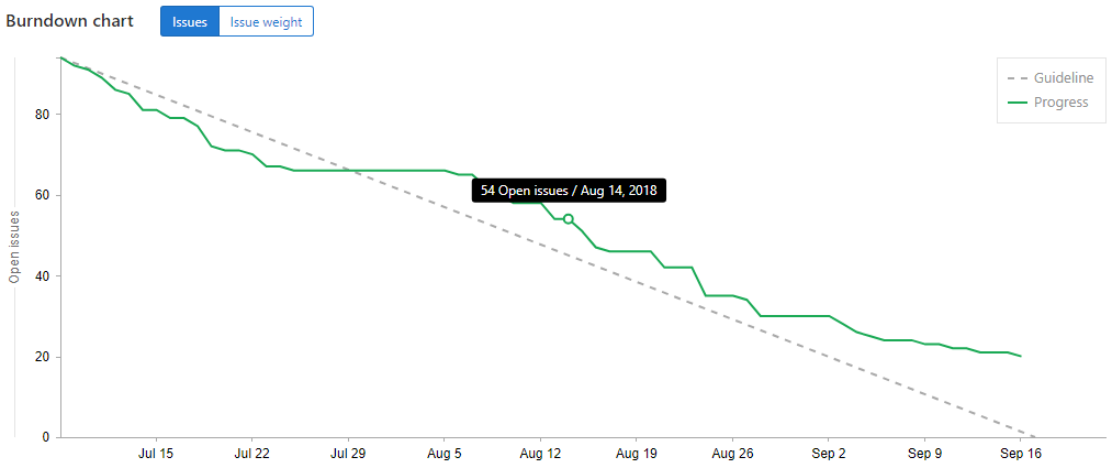


Abbildung 1: Burndown-Chart über die Anzahl der GITLab Issues zur Implementierungsphase

3. Implementierung und Abweichungen

3.1. Abweichungen zum Pflichtenheft

/**A010**/ Damit ein Passwort valide ist darf es zusätzlich höchstens 32 Zeichen enthalten.

/**A020**/ Der Benutzername darf zusätzlich höchstens 32 Zeichen enthalten.

/**A030**/ Zur Registrierung gehört keine Angabe der Telefonnumer mehr.

/**A040**/ Die *Password*-Tabelle der Datenbank benötigt kein *Salt*-Attribute, da von Argon2 ein salt generiert und in den Hash-String kodiert wird.

/**A050**/ Abweichung zu FA0110:

- Passwort schlägt in mehr Fällen fehl, als spezifiziert. (Spezifiziert war lediglich minimal 8 Zeichen).

/**A060**/ /MK350/ zugehörig /FA1010/ /FA1020/ und /FA1030/ wurden nicht imple-

mentiert.

/**A070**/ Abweichung zu /F0220/ - Eine Überprüfung von Auflösung und Dateigröße findet nicht statt. Es werden weitere Dateitypen erlaubt.

/**A080**/ Abweichung zu /F0230/ - Eine Statusnachricht kann nur 64 statt 256 Zeichen enthalten. Emojis werden als mehrere Zeichen gezählt.

/**A090**/ Abweichung zu /F0240/ - Es werden zusätzliche Freiheiten bei der Formatierung einer Telefonnummer gewährt.

/**A0100**/ Abweichung zu /F0310/ - Das alte Password muss nicht bestätigt werden.

/**A0110**/ Abweichung zu /D1020/ - Attribut ist nicht verpflichtend / transparent für den Endbenutzer

/**A0120**/ Für /D1040/ und /D1050/ werden zusätzlich Erstell- und Änderungsdatum sowie Sichtbarkeitseinstellungen gespeichert

/**A0130**/ /D1080/ umfasst folgende Informationen über den Webbrowser

- Webbrowser PUSH-API Endpoint
- Webbrowser Identifikation
- Secret zum verschlüsselten Nachrichtenaustausch mit dem Webbrowser

/**A0140**/ /D1120/ und ggf. zugehörig weitere Erhobene Daten wurden an die Third-Party Software → Ory Hydra ausgelagert.

/**A0150**/ /D2020/ umfasst ebenso das Datum der zuletzt gelesenen Nachricht

/**A0160**/ /D2040/ umfasst weitere Meta-Information

- Upload-Datum

- Dateityp

/**A0170**/ Abweichung zu /NF220/ - Der Wechsel der Sprache ist noch nicht implementiert

/**A0180**/ Abweichung zu Testszenario 8.2.1

- Bestätigung der Eingabe (welche zum nächsten Schritt führt) durch Button-Klicken zwischen der Passworteingabe.

- Einen 4. Registrierungsschritt zur Eingabe einer Telefonnummer (laut Testszenario "+49 178 43 04 37") existiert zurzeit nicht

/**A0190**/ Abweichung zu Testszenario 8.2.2 - Schritt 1 kann nicht ausgeführt werden. Es kann sofort mit Schritt 2 begonnen werden.

/**A0200**/ Abweichung zu Testszenario 8.2.5

- Der Benutzer muss auf seine Profil-Kurzübersicht klicken. Ein ProfilReiter existiert nicht im Menü.

- Der Benutzername kann nicht geändert werden.

- Avatarbilder werden aufgrund eines Bugs noch nicht übernommen

Es wurden keine Angaben zur Erfüllung von /NF310/ und /NF320/ sowie aller Testfälle /TXXX/s. Sämtlich WKs und NFOs wurden in dieser Übersicht nicht berücksichtigt.

Des weiteren sind noch Abweichungen für folgende Elemente zu Prüfen:

- MK 240

- Abschnitt MK 3.XX

- F0210

- Kapitel FA 5.4
- Kapitel FA 5.5
- Kapitel FA 5.6
- Kapitel FA 5.7
- Kapitel FA 5.8
- Kapitel FA 5.9
- Kapitel FA 5.11
- Testszenario 8.2.3
- Testszenario 8.2.4
- Testszenario 8.2.6
- Testszenario 8.2.7

3.2. Inbetriebnahme

Tabelle 1: Übersicht die einzelnen Server und ihre Ports

Software	DevPort	Port	Domain
Reverse-Proxy	-	80/443	-
WebServer	-	80/443	cdn.chatsphere.de
PWA	3020	-	chatsphere.de
API-Specs	3000	-	-
oAuth-Server	-	4444	auth.chatsphere.de
Backend-Server	8080	6000	api.chatsphere.de
Datenbank	3306	3306	-
Docs (Optional)	-	80/443	report.chatsphere.de

Zur Inbetriebnahme werden insgesamt 4 Server auf unserem Testsystem eingesetzt:

- → CaddyServer als → Reverse-Proxy für das Backend und den Authorisierungsserver sowie als WebServer für die → PWA und die Dateien-Auslieferung.
- → Jetty als → HTTP -Server für das → Java-Servlet
- → MariaDB als → SQL Datenbanksystem.
- → Ory Hydra als → oAuth2.0 Authorisierungsserver

3.3. Development Project Build

Um das Projekt lokal zu testen kann eine wesentlich simplere Konfiguration verwendet werden, als sie auf unserem Testserver läuft. Grundvorraussetzung ist die Installation der Entwicklungswerkzeuge, welche in der *Readme.md* beschrieben werden.

- Die PWA wird über einen Entwicklungsserver *cd client && yarn dev* ausgeliefert
- Der Jetty-WebServer wird über *cd server && mvn jetty:run* gestartet. Die Einstellungen können vor dem Kompilieren in der *server/chatsphere-server/src/ressources/properties.yaml*

vorgenommen werden.

- Das Datenbankschema muss in einem SQL-Server importiert werden, mit dem JETTY kommunizieren kann.

- Ein WebServer muss unter der in *properties.yaml* für den glswebserver konfigurierten URL für den Dateispeicher die Dateien bereitstellen.

Zur Entwicklung wird min. *6 GB* freier Arbeitsspeicher empfohlen.

3.4. Installation

Die Kurzversion der Installation sieht ähnlich aus. Mit *yarn build* wird das Frontend erstellt und mit *mvn site* das Artefakt für den → Jetty WebServer.

Für den Produktiveinsatz können die vorgefertigten Konfigurationsdateien und → systemd -Dienste aus dem Repository-Ordner *config* genutzt werden.

Erforderlich für den Produktiveinsatz ist der Einsatz über → TLS geschützte → HTTP - und → WS -Verbindungen, welche über einen → Reverse-Proxy bereitgestellt werden sollten, der weiterführend die erforderliche Konfiguration von → CORS vornehmen kann. Sonst kann es zu Probleme bei der Installation des → ServiceWorker kommen.

Sinnvoll aber nicht zwingend erforderlich ist die Konfiguration von benutzerdefinierten Fehlerseiten.

Empfohlen wird darüber hinaus die Konfiguration von → OSCP-Stapling und → CT für → TLS sowie Unterstützung der Erweiterungen → ALPN und → SNI ; → SPF , → CAA und ggf. → DANE in den DNS-Records; → CSP und → HSTS in den → HTTP -Header, sowie neben → HTTP - auch → QUIC -Protokoll support.

3.4.1. Ory Hydra Client einrichten

Ein neuer → oAuth2.0 -Client kann mittels folgendem Befehl angelegt werden:

```
1     ory-hydra clients create --endpoint "https://auth.chatsphere.de" --name "
          chatsphere-pwa" --is-public --callbacks "https://chatsphere.de/oauth2/
          callback" --grant-types "authorization_code" --response-types implicit
           --scope "chatsphere-pwa"
```

3.5. GITLab Pipeline: Continuous Deployment

Unser Testserver ist über einen GITLab-Runner in den GITLab-Workflow integriert und führt automatisch Tests für Merge-Requests aus und führt ein automatisches Continoues-Deployment von verabschiedeten Änderungen durch.

Die GITLab Pipeline Konfiguration kann in der *.gitlab-ci.yml* nachgelesen werden.

Grundlegend ist das Deployment in 8 Stages mit 21 Jobs unterteilt:

1. **systeminfo** prüft das Vorhandensein und die Versionsnummer aller benötigten Software-Abhängigkeiten

2. **prepare** bereitet das Repository auf die Ausführung nach einem Klonene des GIT-Repositories vor und lädt insbesondere Dependencies mit dem jeweiligen Dependency-Management-Werkzeug herunter.

3. **lint** führt einen Code-Style-Check durch und erkennt ebenso Syntax-Fehler.

4. **test** ist der Abschnitt in den die Unit-Tests automatisch ausgeführt udn fehlschläge gemeldet werden.

5. **build** erstellt die jeweiligen Artefakte für die Produktionsumgebung

6. **deploy** veröffentlicht und startete die Artefakte auf dem Live-System

7. **Integration** führt E2E-Tests mit Google Chrome und Firefox durch und kann auf den vollständigen Anwendungsstack hierfür zurückgreifen.

8. **docs** erstellt Reports und Dokumentationen und stellt diese unter report.chatsphere.de bereit.

Davon werden die Abschnitte *deploy* und *docs* lediglich bei gemergten Merge-Requests ausgeführt, wogegen alle weiteren Schritte um die Code-Qualität zu steigern und schnelles Feedback bei Änderungen zu erhalten, mit jedem Commit ausgeführt werden.

Zwischen den einzelnen Jobs werden Abhängigkeiten gecached und Artefakte der vorherigen Jobs wiederverwendet, um die Ausführungszeit zu beschleunigen.

- **Systeminfo** liefert Informationen zum Ort und Version der benötigten Anwendungen zum Bauen und Testen der Anwendung

- **Frontend Jobs**

  - **Download Dependencies** lädt die benötigt *YARN* / *NODEJS* Abhängigkeiten herunter.

  - **ESLint** führt einen Linting und Syntax-Check für den *JavaScript*-Code und die *GraphQL*-Queries aus.

  - **Unit Test** Führt die *NighWatch* Unit Tests in *Chromedriver* und *Geckodriver* aus.

  - **Build** baut die statischen PWA-Dateien zur Auslieferung durch einen → HTTP -WebServer

  - **Deploy** Aktualisiert die statischen Dateien auf dem Live-System

  - **Integration** Führt einen *E2E*-Integrationtest auf dem Livesystem mit *Chromeium* und *FireFox* aus.

  - **Coverage** Erstellt einen Coverage-Report und stellt diese unter *report.chatsphere.de* bereit.

  - **Docs** Erstellt die Dokumentation zum Frontend Source-Code und stellt diese

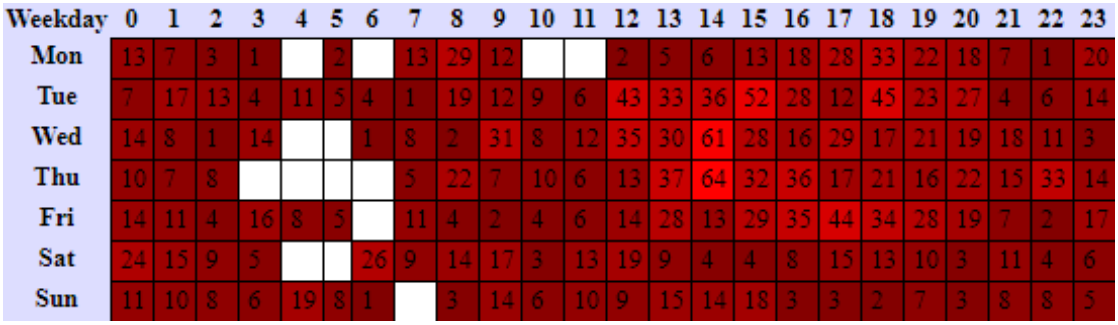
unter *report.chatsphere.de* bereit.

- **API Jobs**
  - **Download Dependencies** lädt die benötigt *YARN* / *NODEJS* Abhängigkeiten herunter
  - **ESLint** Führt einen Linting und Syntax-Check für die *GraphQL SDL* und den *JavaScript*-Code aus.
  - **Deploy** Aktualisiert den zugehörigen *NodeJS Express Server* via *HotReload* auf dem Live-System.
  - **Docs** Erstellt eine API-Dokumentation auf Basis des API-Endpoints des Live-System.
- **Backend Jobs**
  - **Download-Dependencies** - Aktualisiert die *Maven* Abhängigkeiten
  - **CheckStyle** führt einen Syntax und Linting-Check durch.
  - **Unit Test** führt die *JUNIT*-Tests sowie ein Mutation-Testing aus.
  - **Package** erstellt die Servlet-Artefakte
  - **Deploy** deployed das Backend auf das Livesystem
  - **Docs** Erstellt Reports über das Projekt, dessen Abhängigkeiten, Coverage, Mutation-Testing Ergebnisse und Code-Dokumentation und stellt diese unter *report.chatsphere.de* bereit
- **Database Jobs**
  - **Deploy Test** Setzt die Test-Datenbank für die Integration-Tests des Backends neu auf.

– **Deploy** Setzt die Datenbank des Livesystem neu auf.

4. Statistiken

Im Durchschnitt wurde am meisten unter der Woche zwischen 10:00 Uhr und 16:00 Uhr gearbeitet mit insgesamt mehr als 100 Commits pro Stunde in diesem Zeitraum. Montag Vormittag findet ein persönliches Treffen statt, weswegen zeitlich weiträumig der Montag hinter den Erwartungen zurückbleibt, aber trotzdem höhere Aktivitäten als am Wochenende verbucht.



Weekday	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Mon	13	7	3	1		2		13	29	12			2	5	6	13	18	28	33	22	18	7	1	20
Tue	7	17	13	4	11	5	4	1	19	12	9	6	43	33	36	52	28	12	45	23	27	4	6	14
Wed	14	8	1	14			1	8	2	31	8	12	35	30	61	28	16	29	17	21	19	18	11	3
Thu	10	7	8					5	22	7	10	6	13	37	64	32	36	17	21	16	22	15	33	14
Fri	14	11	4	16	8	5		11	4	2	4	6	14	28	13	29	35	44	34	28	19	7	2	17
Sat	24	15	9	5			26	9	14	17	3	13	19	9	4	4	8	15	13	10	3	11	4	6
Sun	11	10	8	6	19	8	1		3	14	6	10	9	15	14	18	3	3	2	7	3	8	8	5

Abbildung 2: Durschnittliche Commits pro Wochentag und Stunde

Beginnend mit dem Juni bis einschließlich Oktober wechseln sich *Alexander* und *Alex* mit dem ersten Platz im *Author of the Month*-Rating der Software *git-stats* ab. Die meisten GIT-Identitäten hat *Julien* mit 4 an der Zahl, welche diese Auswertung beeinträchtigen.

Aus der über die GITLab-Identitäten zusammengefasste Übersicht der Gesamtstatistik ergibt sich, das *Berthold* die meisten Commits und hinzugefügten Codezeilen hat, wogegen *Alex* die meisten Code-Zeilen löschte.

Tabelle 2: Gesamtübersicht über die GITLab-Aktivitäten im *develop*-Branch bis zum 16. September 2018 sortiert nach Anzahl der Commits.

Author	Commits	lines added	lines removed	First commit	Last commit
Berthold	710	54463	11499	2018-06-04	2018-09-13
Alex	600	33106	26270	2018-06-03	2018-09-04
Julein	436	17779	10635	2018-06-05	2018-09-13
Alexander	354	6757	4737	2018-06-10	2018-09-15
Niklas	152	28746	9349	2018-06-07	2018-09-16

A. Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

A.1. Glossar

ALPN *Application-Layer Protocol Negotiation (ALPN)* ist eine Erweiterung des → TLS -Protokolls, um Informationen über die angebotenen Application-Layer zu erhalten.

Apollo Apollo ist eine Plattform für die Entwicklung von GraphQL-APIs.

CAA

DNS Certification Authority Authorization (CAA) ist ein Ressource-Record-Typ, welcher die Ausstellung von Zertifikaten für eine Domain durch eine bestimmte Zertifizierungsstelle erlaubt. Ebenso wird angegeben, an welche EMail-Adresse Verletzungen durch andere Zertifizierungsstellen gemeldet werden.

CaddyServer Ein in Go programmierter WebServer, welche auch als → Reverse-Proxy eingesetzt werden kann.

Continuous Deployment Continuous Deployment bezeichnet eine Entwicklungsmethode mit der Änderungen kontinuierlich ausgeliefert werden.

CSP *Content-Security-Policy (CSP)* ist eine umfangreiche HTTP-Header-Directive-Sandbox über welche strikt reguliert werden kann, welche Ressourcentypenvon welchen Domains geladen werden können. Durch das automatische Reporten von Viaolations ist die Kontrolle von abgerufenen Quellen und das erkennen von Angriffen über bspw. → XSS möglich.

CT *Certificate Transparency* ist ein Standardisierter Prozess, welcher bei uns durch eine HTTP-Header-Directiven vorrausgesetzt wird, welcher die Überprüfung ausgestellter Zertifikate durch die Protokollierung der Ausstellung ermöglicht und dadurch die Missbräuchliche Austellung von Zertifikaten durch Dritte Zertifizierungsstellen vorbeugt.

DANE **DANE** ist ein weitgehend von Webbrowser nicht unterstütztes Feature, welches aufbauend auf der Vetrauenswürdigkeit von DNSSEC die Authorisierung bestimm-

ter, auch selbstsiginierter, Zertifikate für einen bestimmten Dienst unter der Domain erlaubt.

DNS Das *Domain Name System (DNS)* hat zur Aufgabe menschenlesbare Domainnamen in für den Computer verarbeitbare IP-Adressen umzuwandeln.

DNS-Record Ein → DNS -Eintrag.

GITLab Pipeline Ein Feature von GITLab, welches verschiedene Abschnitte der Projekt-Kompilierung in einzelnen Stages einer Pipeline konfigurieren lässt, welche anschließend automatisiert ausgeführt werden können.

GraphQL Eine Abfragesprache für API-Abfragen, welche API-Objekte in einem Graphen modelliert.

HSTS *HTTP Strict Transport Security (HSTS)* ist ein HTTP-Header-Directive, welche den Einsatz von HTTPS gegenüber HTTP erzwingt.

HTTP Hypertext Transfer Protokoll - Ein im Internet genutztes Protokoll von Hyper Text Markup Language, der Hervorhebeungssprache auf dem Webseiten basieren.

Java-Servlet Ein Java-Container welcher von einem Webserver ausgeführt wird und Anfragen beantwortet.

Jetty Ein Java WebServer welcher als WebServer-Umgebung für die Ausführung eines Servlet-Containers dient.

Jetty Ein HTTP-WebServer zur Ausführung von *Java-Servlets*.

MariaDB Ein SQL Datenbankserver.

name *Cross-Origin-Ressource-Sharing (CORS)* ist ein Sicherheitsprotokoll, welches Anfragen an andere Domains nur erlaubt, wenn über das CORS-Protokoll die Einbindung einer Ressource von einer bestimmten Domains aus erlaubt wurde.

oAuth 2.0 Ein Industrie-Standard Protokoll, dokumentiert unter https://oauth.net/2/ für die Authorisierung nach RFC 6749.

ORMLite Ein leichtgewichtige Bibliothek für Objekt Relational Mapping von OOP-Objekten zu Datenbanktabellen.

ORY Hydra Ein in Go geschriebener oAuth 2.0 Authorisierungsserver der mithifle einer REST-API an die eigene Anwendung angebunden werden kann.

OSCP-Stapling *OSCP Stapling* ist eine Technik bei derder Webserver von der Zertifizierungsstelle eine mit Zeitstempel signierte Auskunft über den Revocation-Status eines TLS-Zertifikates abruft und beim Verbindungsaufbau durch Endgeräte automatisch zusendet. Dadurch müssen Endgeräte keine eigene Abfrage des Revocation-Status eines Zertifikates durchführen.

PWA Eine *Progressive Web Application (PWA)* ist eine Webanwendung die alle Anzeichen einer nativen Anwendung zeigt.

QUIC Ein auf dem Protokoll *UDP* basierendes Protokoll welches eine schnelle verschlüsselte Verbindungsaushandlung zum Ziel hat.

Repository Als Repository wird das Modell eines → Resolver bezeichnet. In erster Linie unterstützt es bei der Verwaltung zwischen API-Objekten und Datenbankobjekten, aber auch bei der Implementierung von Hilfsbibliotheken.

Resolver Resolver werden die Controller einer GraphQL-API genannt, welche einzelne API-Methoden implementieren.

Reverse-Proxy Ein Proxy ist ein Vertreter, der anstelle einer selbst agiert. Während ein Forward-Proxy Anfragen für einen Client ausführt, beantwortet ein Reverse-Proxy Anfragen anstelle eines Servers. Der Proxy leitet häufig antwortet weiter und übernimmt zusätzliche Aufgaben wie Caching (Zwischenspeichern), Encryption (Verschlüsselung) oder überprüfung von vorgaben wie Policys (Richtlinien).

RxJava Eine Java-Bibliothek zur Implementierung von Reactive Streams. Siehe hierzu auch https://www.reactive-streams.org.

ServiceWorker Ein Hintergrundprozess wodurch eine Webseite die Möglichkeit erhällt wie eine native Anwendung zu agieren.

SNI *Server Name Indication (SNI)* ist eine TLS-Erweiterung bei der der angefragte Hostname bei der Aushanldung der verschlüsselten Verbindung mitübertragen wird, wodurch unter eine Adresse mehrere Domains mit eigenem Zertifikat gehostet werden können.

SPF Das **Sender Policy Framework (SPF)**erlaubt die Deklaration von für den Versand von Mails unter der Domain authorisierten IP-Adressen. Dadurch wird das unerkannte versenden von E-Mails durch andere Server an EMail-Provider die diesen Header auswerten unterbunden.

SQL Mit der »Structured Query Language« werden Anfragen an einen SQL-Datenbankserver formuliert.

systemd Ein Hintergrundprozess für das Linux-Prozess, welcher zum Starten und Verwalten von weiteren Programmen und Diensten dient.

TLS Über das *Transport Layer Secure (TLS) können verschiedene Protokolle wie → HTTP und → WS verschlüsselt eingesetzt werden. Unser Projekt nutzt kostenlose Zertifikate des Anbieters Let's Encrypt*.

VueJS VueJS ist ein JavaScript-Framework für die Frontend-Entwicklung.

WS *WebSockets (WS)* ist ein Protokoll welches die Bidirektionale Kommunikation zwischen Client und Server erlaubt. Es wird von Webbrowsern unterstützt.

XSS *Cross-Site-Scripting (XSS)* ist eine Sicherheitslücke in einer Web-Anwendung, welche die Einschleußung von Source-Code, meistens aufgrund unzureichend validierter Ausgabe von eingegebenen Formalinhalten, erlaubt.

```
MIT License

Copyright (c) 2018 Alexander Wank, Niklas Seyfarth, Julien Midedji, Berthold Niemann & Alexander Brese

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```