

PRAXIS DER SOFTWAREENTWICKLUNG

KARLSRUHER INSTITUT FÜR TECHNOLOGIE

INSTITUT FÜR TELEMATIK

LEHRSTUHL PROF. DR. MARTINA ZITTERBART



# Modern Messaging Platform: ChatSphere

Validierungsbericht

Alexander Wank   Niklas Seyfarth

Julien Midedji   Berthold Niemann

Alexander Brese

betreut von: Tim Gerhard und Markus Jung

23. September 2018

## Inhaltsverzeichnis

<b>Vorwort</b>	<b>3</b>
<b>1 Organisation</b>	<b>3</b>
1.1 Einleitend r Akt . . . . .	3
1.2 Testend r Akt . . . . .	5
1.3 Abnehmend r Akt . . . . .	5
<b>2 Testergebnisse</b>	<b>6</b>
2.1 Servir . . . . .	6
2.2 Client . . . . .	8
<b>3 Usability-Test</b>	<b>9</b>
3.1 Übersicht . . . . .	9
3.2 Usability-Metriken . . . . .	9
3.3 Benutzerprofil . . . . .	10
3.4 Durchführung . . . . .	10
3.5 Ergebnisse . . . . .	11
<b>4 Softwarefehler</b>	<b>11</b>
4.1 Klassifizierung . . . . .	12
4.2 Dringlich Bugs . . . . .	12
4.2.1 Sitzungsanführung . . . . .	12
4.2.2 Pipelinefehlrmldung in sämtlichen Merg Requists . . . . .	14
4.2.3 Platzhalter für kritischen Bug . . . . .	15
4.3 Statistiken . . . . .	15

## Vorwort

Die Qualitätssicherungsphase ist ein systematischer Vorgang, der sich erst vollziehen soll, dass die Anwendung die Anforderungen zu den gewünschten Qualitätsmerkmalen erfüllt.

Zur Qualitätssicherungsphase werden Maßnahmen implementiert um → externen Qualitätsmerkmalen wie *Korrektheit*, *Robustheit* und *Zuverlässigkeit*; sowie → internen Qualitätsmerkmalen wie *Testbarkeit*, *Wartungsfreundlichkeit* und *Verständlichkeit* zu verbessern.

In diesem Dokument werden Techniken genannt um die Aufgabe zu bewältigen. Testergebnisse werden dokumentiert und Statistiken präsentiert. Probleme, Fehler, Schwierigkeiten und Lösungen werden genannt und veranschaulicht.

## 1 Organisation

Die dreiwöchige Testphase soll in drei Phasen geteilt werden.

### 1.1 Einleitender Akt

Im Einleitenden Akt sollen Ideen für die Testphasen gesammelt, Ansätze der Strukturierung diskutiert werden und die Aufteilung der Arbeit stattfinden.

Libraries sollen installiert werden um Statistiken inszenieren zu können damit zum Ende des Aktes jeder in Entwicklungsumgebung mit Tools zum Testen hat mit der Umgebung hantieren kann.

Aus der Erfahrung vorheriger Phasen konnte man feststellen dass es in gewisser Zeit braucht bis jeder Person produktiv werden konnte.

Ziel der Phase soll ein langsamer Einstieg in die Hochphase der Qualitätssicherungsphase - dem tatsächlichen Akt - sein, der mit der meistigen Programmierarbeit zusammenhängt.

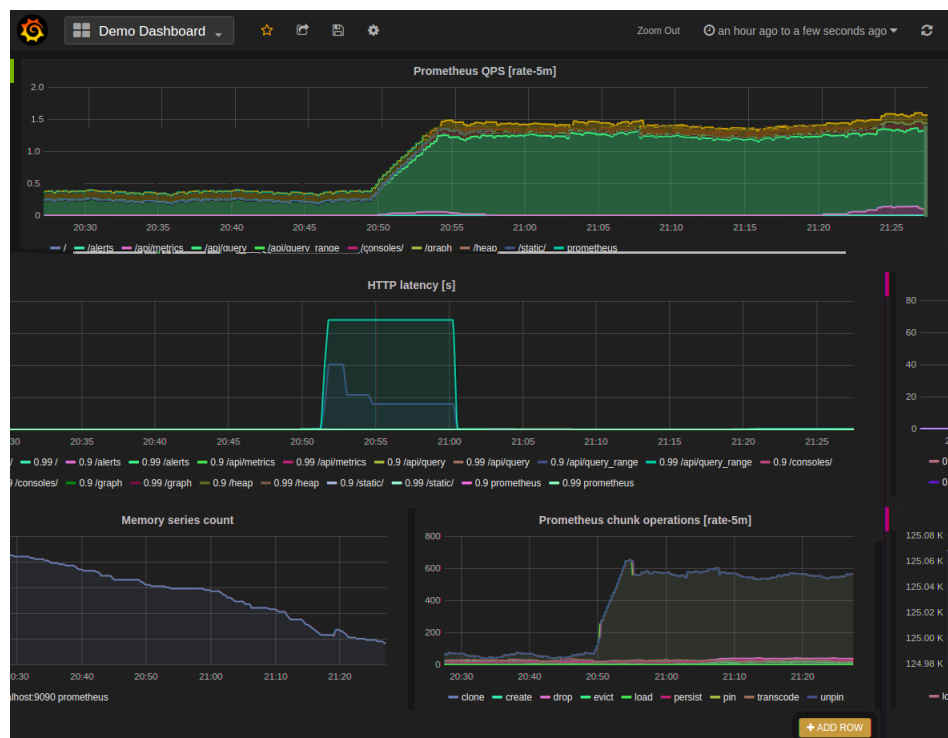
Zu den möglichen Ideen die gesammelt wurden gehört:

## 1 Organisation

- Namenskonventionen ("-Dto", Resolvernamen, ...) dokumentieren und Programmierkonventionen festlegen die für die Anwendung angewandt sind. Alle Stilbrüche beseitigen.
- Metriken finden um Qualität zu quantifizieren (Code Coverage, Lines of Code, Dependenzen, Anteil an FAs die formal inspiert wurden)
- Mittels Tools kritisch Codebereich analysieren und priorisieren

Zudem möglichen Tools gehören:

→ Prometheus zum sammeln und dokumentieren von Ereignissen die zum/im Server (HTTP-Anfragen, Datenbankabfragen, Latenzen) stattfinden. Diese können danach mit Grafana, welche sich → Dashboards bietet, visualisiert und ausgewertet werden.



**Abbildung 1:** Mögliche Visualisierung mit Grafana mit Daten die aus Prometheus stammen (<https://prometheus.io/docs/visualization/grafana/>).

## 1 Organisation

---

### 1.2 Testender Akt

Im Testenden Akt ist das hauptsächlich s Ziel die Test Coverage zu erhöhen. Zu den Tests kommt das nötige Refactoring um die Testbarkeit zu gewährleisten.

Die geplanten Arten von Tests und Verfahren in diesem Akt sind:

1. → Einheitstest mittels JUnit und Mocha
2. → Integrationstests
3. → Regressionstest automatisierbar u.a mit Karma
4. → Systemtest
5. → End-Zu-End-Test mittels Nightwatch
6. → Mutationen-Test mittels PIT

Es sind schon verinzelt Tests zu einigen Modulen aus der Implementierungsphase gegeben. Ziel ist es in umfangreicheren und aussagekräftigeren Testapparat zu entwickeln welcher seinen gewissens Grad an Robustheit für die Anwendung bezeugen kann.

### 1.3 Abnehmender Akt

Der Abnehmende Akt soll die Anwendung für einen Abnahme durch den Benutzer vorbereiten. Dazu werden manuelle Überprüfung auf dem Testsrv vorgenommen und möglich Szenarien durchgegangen. Fehler werden dokumentiert und Lösungen so weit wie möglich realisiert.

Die insgesamt 59 Funktionalanforderungen (FA) werden in sechs Rubriken gegliedert: Jedem Entwickler wird in Rubrik von FAs zugeordnet und er so weit wie möglich Beteiligung hat und welcher vom Umfangs in der richtigen verfügbaren Zeit entspricht. Die letzte Rubrik umfasst alle FAs die nicht in der Implementierungsphase realisiert worden sind.

Ziel des Aktes soll es sein alle Rufschaden Fehler im Programm, die bei in der Live-Deployment auftreten könnten zu beseitigen.

## 2 Testergebnisse

---

Rubrik 1	Rubrik 2	Rubrik 3
0410 Kontaktliste anzeigen	0310 Passwort ändern	0220 Profilbild ändern
0420 Kontaktoptionen anzeigen	0320 E-Mail ändern	0230 Statusnachricht ändern
0820 Öffentliche Gruppen-Chats suchen	0510 Benutzeroptionen anzeigen	0720 Nachricht verschicken
0610 Chat-Liste anzeigen	0520 Benutzer melden	0730 Nachricht empfangen
0810 Kontakte auswählen	0530 Benutzer blockieren	0740 Nachricht löschen
0820 Öffentliche Gruppen-Chats suchen	0540 Benutzer freigeben	0750 Nachricht bearbeiten
		0760 Nachricht archivieren
Rubrik 4	Rubrik 5	Unrealisiert
0450 Kontakt hinzufügen	0110 Registrierung	0470 Kontaktimportoptionen anzeigen
0460 Kontakte entfernen	0120 Login	0480 Kontakte importieren
0630 Chat öffnen	0130 Auto-Login	0660 Chat stumm schalten
0640 Chatoptionen anzeigen	0140 Logout	0680 Broadcast-Chat erstellen
0650 Chat löschen	0210 Profil anzeigen	0830 Öffentliche Gruppen-Chats beitreten
0670 Chat erstellen	0910 Multimedia-Optionen anzeigen	0850 Private Gruppen-Chats veröffentlichen
0690 Gruppen-Chat verlassen	0920 Multimedia-Nachricht verschicken	0860 Öffentliche Gruppen-Chats privat machen
		0930 Bild-Nachricht vergrößern
		0940 Link-Vorschau ansehen
		0950 Nachricht formatieren
		0960 Global-Chat Zuweisung

Abbildung 2: Unterteilung der FAs.

## 2 Testergebnisse

### 2.1 Server

1. d .chatsph r .io.databases .sch ma.Profil T st
2. d .chatsph r .io.databases .sch ma.us r.Us rR lationshipT st
3. d .chatsph r .io.databases .sch ma.us r.Us rL galT st
4. d .chatsph r .io.databases .sch ma.us r.Us rT st
5. d .chatsph r .io.databases .sch ma.us r.Us rPr f r nc sT st
6. d .chatsph r .io.databases .sch ma.chat.Lin PositionT st
7. d .chatsph r .io.databases .sch ma.chat.ChatParticipantT st
8. d .chatsph r .io.databases .sch ma.chat.ChatT st
9. d .chatsph r .io.databases .sch ma.chat.M ssag T st
10. d .chatsph r .io.databases .sch ma.chat.ChatLin T st

## 2 Testergebnisse

11. d.chatsph r.io.databases.sch.ma.Databas Conn ctionT st
12. d.chatsph r.io.databases.sch.ma.pr f r nc .PasswordT st
13. d.chatsph r.io.databases.sch.ma.pr f r nc . num ration.Notifiabl T st
14. d.chatsph r.io.databases.sch.ma.pr f r nc . num ration.VisibilityT st
15. d.chatsph r.io.databases.sch.ma.pr f r nc .Languag T st
16. d.chatsph r.io.databases.sch.ma.pr f r nc .EmailT st
17. d.chatsph r.io.databases.sch.ma.pr f r nc .EmailV rificationT st
18. d.chatsph r.io.databases.sch.ma.pr f r nc .Fil T st
19. d.chatsph r.io.databases.sch.ma.pr f r nc .ColorT st
20. d.chatsph r.io.databases.sch.ma.pr f r nc .Phon T st
21. d.chatsph r.io.databases.sch.ma.pr f r nc .LocationT st
22. d.chatsph r.io.databases.sch.ma.cat gory.Cat goryT st
23. d.chatsph r.io.databases.sch.ma.cat gory.Cat goryPathT st

G start t: 24, F hlg schlag n: 0, Üb rsprung n: 2

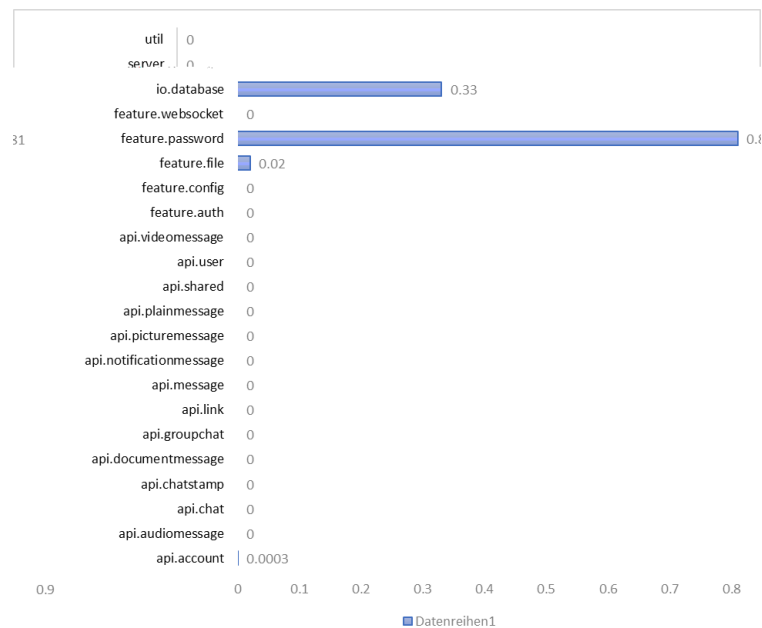


Abbildung 3: Coverage auf Serverseite.

## 2 Testergebnisse

### 2.2 Client

HeadlessChrome 69.0.3497 (Linux 0.0.0): Executed 2 of 2 SUCCESS (0.22 s cs / 0.107 s cs)

✓should have chatshop as title

✓should display a logo

Firefox 61.0.0 (Linux 0.0.0): Executed 2 of 2 SUCCESS (0.155 s cs / 0.049 s cs)

✓should have chatshop as title

✓should display a logo

TOTAL: 4 SUCCESS

Statements : 8.46% ( 34/402 )

Branches : 0% ( 0/153 )

Functions : 1.23% ( 1/81 )

Lines : 8.81% ( 34/386 )

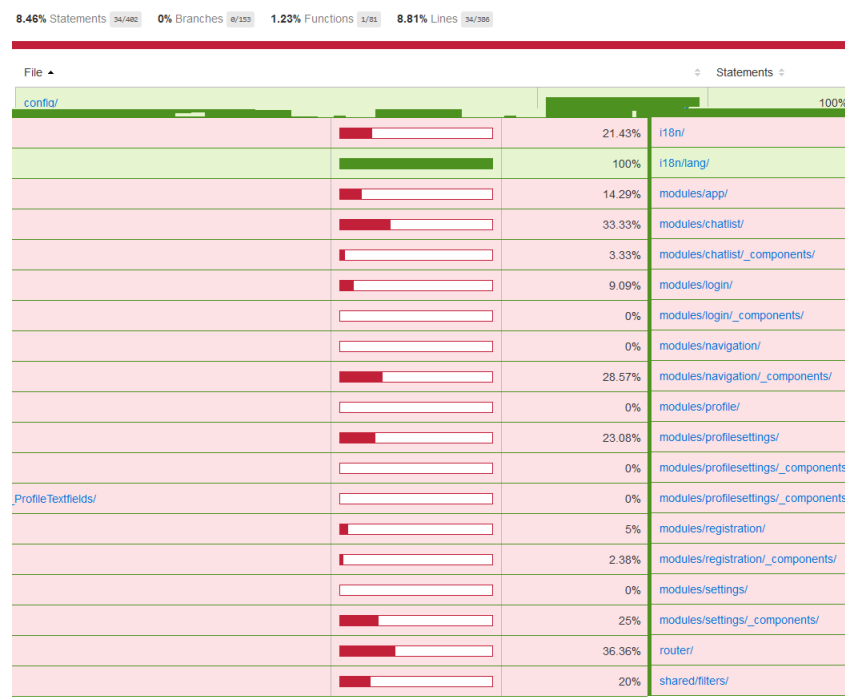


Abbildung 4: Coverage auf Clientseite.



## 3 Usability-Test

### 3.1 Übersicht

Dieses Kapitel beschreibt die Planung des Usability-Tests und zeigt die wesentlichen Ergebnisse. Ziel des Usability-Tests ist es Inkonsistenzen im Design und in der Nutzbarkeit des User Interface (UI) zu finden um die Benutzerzufriedenheit zu verbessern.

Die ausgewählte Gruppe von Benutzern für den Usability-Test besteht aus insgesamt fünf (je der Entwickler wählt führt mit einer Person den Test durch) technik-affinen Personen.

### 3 Usability-Test

---

**Subjektive Metriken** Komm ntar , B w rtung n, Fl üch , Monolog di w ähr nd d r Aufgab g macht w rd n.

#### 3.3 Benutzerprofile

(Noch k in w ahrhaft n Dat n)

#	Alter	Geschlecht	Internet erfahrung
1	20-30	Männlich	El m ntar
2	40-50	Männlich	El m ntar
3	50-60	W iblich	Fortg schritt n
4	30-40	Männlich	El m ntar
5	U20	W iblich	Fortg schritt n

#### 3.4 Durchführung

All Aufgab n w rd n d r R ih nach absolvi rt. Di rst Aufgab f ängt mit d m g - öfn t m Brows r an mit d n Brows r-Startbildschirm d s B nutz rs. Zu j d r Aufgab sind Hinw is worauf das Aug nm rk g l nkt w rd n kann.

1. D r B nutz r öfn t di Anw ndung "chatsph r .d "
  - a) Hat d r B nutz r d r Wortlaut "chatsph r " v rst h n könn n?
  - b) Gibt d r B nutz r di URL in di URL-L ist od r in Googl in?
  - c) Falls ja, kommt "chatsph r .d " als rst r Tr ff r?
  - d) Gibt d r B nutz r b im Lad n in B m rkung zur Lad z it d r W bs it ?
2. D r B nutz r soll sich r gistri r n.
  - a) W rd n Eingab n mit d r Maus b stätigt?
  - b) Navigi rt d r B nutz r mit d r Tastatur?
  - c) Sind di Anford rung n an das Passwort iriti r nd?
3. D r B nutz r soll d n Entwickl r als Kontakt hinzufüg n.

- a) Optimaler Weg: Hamburg → Icon > Contacts > Eingabe des Namens (Frage, wie man in der Anwendung hilft ist gar nicht richtig) > Klicken des Ergebnisses
  - b) Ist dem Benutzer klar geworden nach dem Klicken des Ergebnisses ist richtig?
4. Der Benutzer soll in den Chat mit dem Entwickler anfangen.
- a) Optimaler Weg: Hamburg → Icon > Chat > Stift Icon > Kontakt auswählen > Häkchen Icon
5. Der Benutzer soll in die Nachricht im Chat versenden.
- a) Bestätigt der Benutzer mit der Eingabe?
6. Hinweis kann der Benutzer in die Mitteilung geben.
- a) Ist die Anwendung intuitiv und leicht zu benutzen? Aufwendig?
  - b) Wie steht die Anwendung im Vergleich zu WhatsApp/Discord/Skype/Telegram/Facebook Messenger?
  - c) Gibt es Wünsche, um was die Anwendung erweitert werden könnte?

### 3.5 Ergebnisse

Sobald der Chat steht

## 4 Softwarefehler

Dieses Kapitel beschäftigt sich mit Bugs, die sich durch die Implementierungsphase und Qualitätssphase gezogen haben. Es wird kurz aufgefasst, wie Bugs dokumentiert werden, der in notorischen Bugs und ihre Lösungen werden beschrieben und Statistiken zum Projekt werden gezogen.

### 4.1 Klassifizierung

Bugs werden in drei Schweregrad eingeteilt entsprechend ihres Einflusses auf das System. Diese sind folgend:

**Kritisch** Der Bug führt zum Ausfall in der System /Subsystem.

**Mittelmäßig** Der Bug bewirkt dass das System inkonsistente Ergebnisse liefert oder die Nutzbarkeit behindert

**Niedrig** Ist in formaler ästhetischer Art/Warnung nicht konform zu Konvention oder Standards/Erweiterung bzw. in Erweiterung

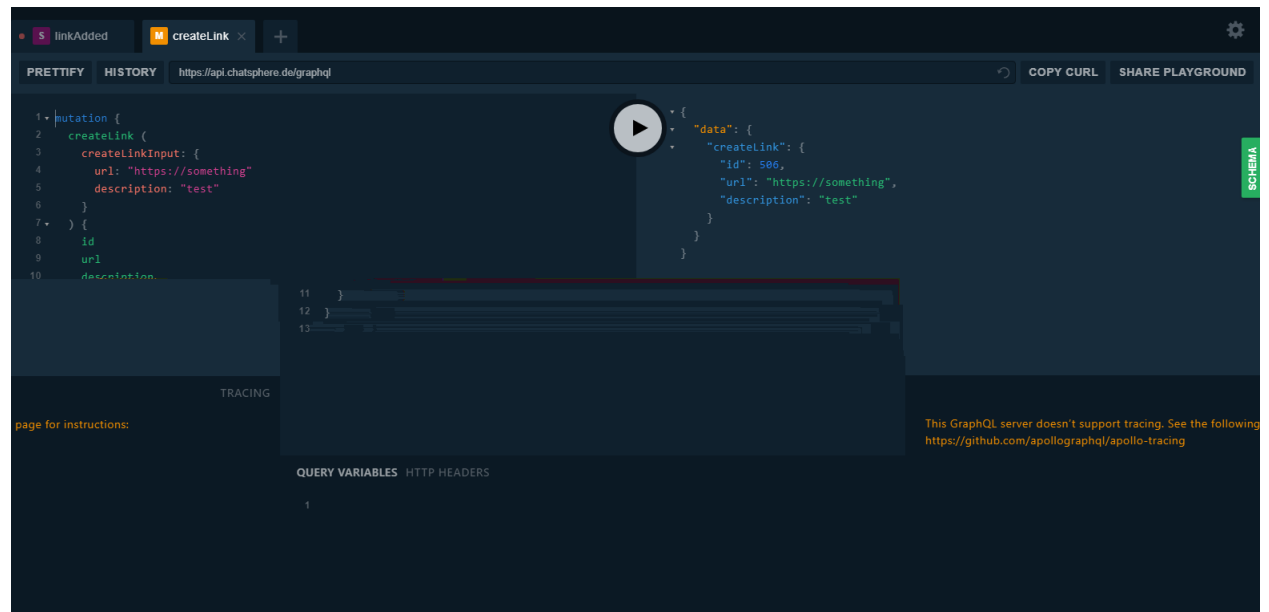
### 4.2 Drei notorische Bugs

#### 4.2.1 Sitzungsentführung

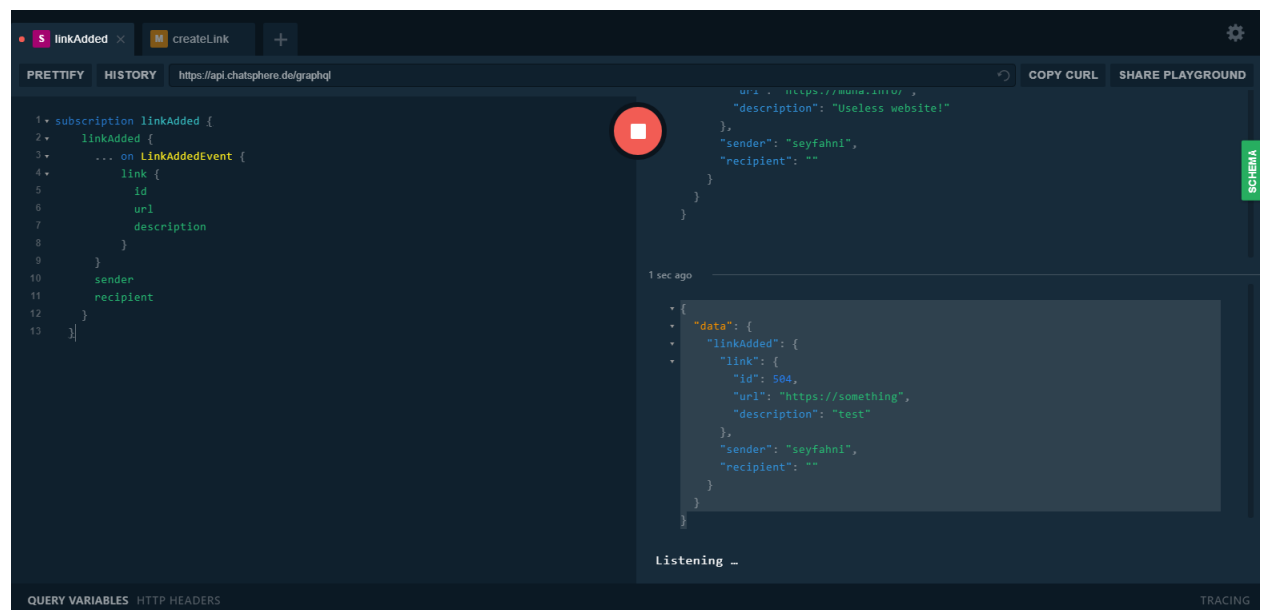
**Beschreibung** Das "Link-Bispiel" ist das → Proof-of-Concept für Subscriptionsgewissens, die erforderlich sind um Chats zu realisieren.

Das Bispiel umfasst einen Subscription-Query `linkAdded`. Sobald dies registriert

## 4 Softwarefehler

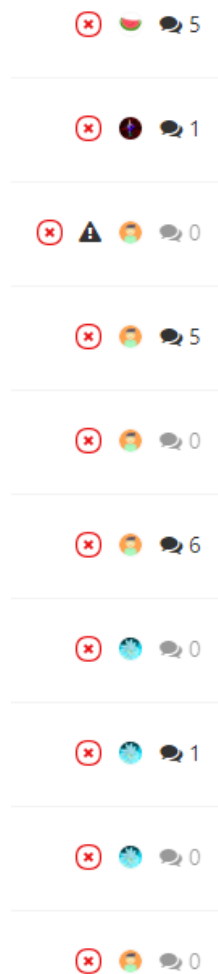


**Abbildung 5:** Person A ist bereits Beobachter von erstellten Links, ist aber nicht authentifiziert - bedeutet er hat keinen Benutzernamen. Mit dem `createLink` erstellt er einen Link



**Abbildung 6:** Nachdem Person A einen Link erstellt hat schaut er was für Daten er übermittelt gekriegt hat über das Abonnement: Der Empfänger (kein Benutzername) ist richtig. Der Sender ist von einer anderen angemeldeten Person.

### 4.2.2 Pipeline Fehlermeldung in sämtlichen Merge Requests



**Abbildung 7:** Fehlermeldung der Merge-Requests

**Beschreibung** Die Pipeline der Merge-Requests auf GitLab sind fehlerhaft. Damit lässt sich vorerst keine Änderung zum Testserver hochladen.

**Klassifizierung** Mittelmäßig

**Lösung** Ich muss mal Niklas fragen

### 4.2.3 Platzhalter für kritischen Bug

**Beschreibung**

**Klassifizierung** Kritisch

**Lösung**

### 4.3 Statistiken

Durch Abbildung 8 erkennt man dass dokumentierte Bugs zur Entwurfsphase und Implementierungsphase sehr kurz leben und nicht haben.

Erst mit dem Ende der Implementierungsphase und beim Übergang zur Testphase werden Bugs häufiger. Überprüfung der Anwendung auf ihre Korrektheit ist mit dem Beginn der Testphase.

Außerdem liegt die Priorität nicht mehr dabei Bugs frühzeitig zu bekämpfen, sondern letzt Funktion noch zur Deadline abzuschließen (auf möglichst Kosten von Bugs.)

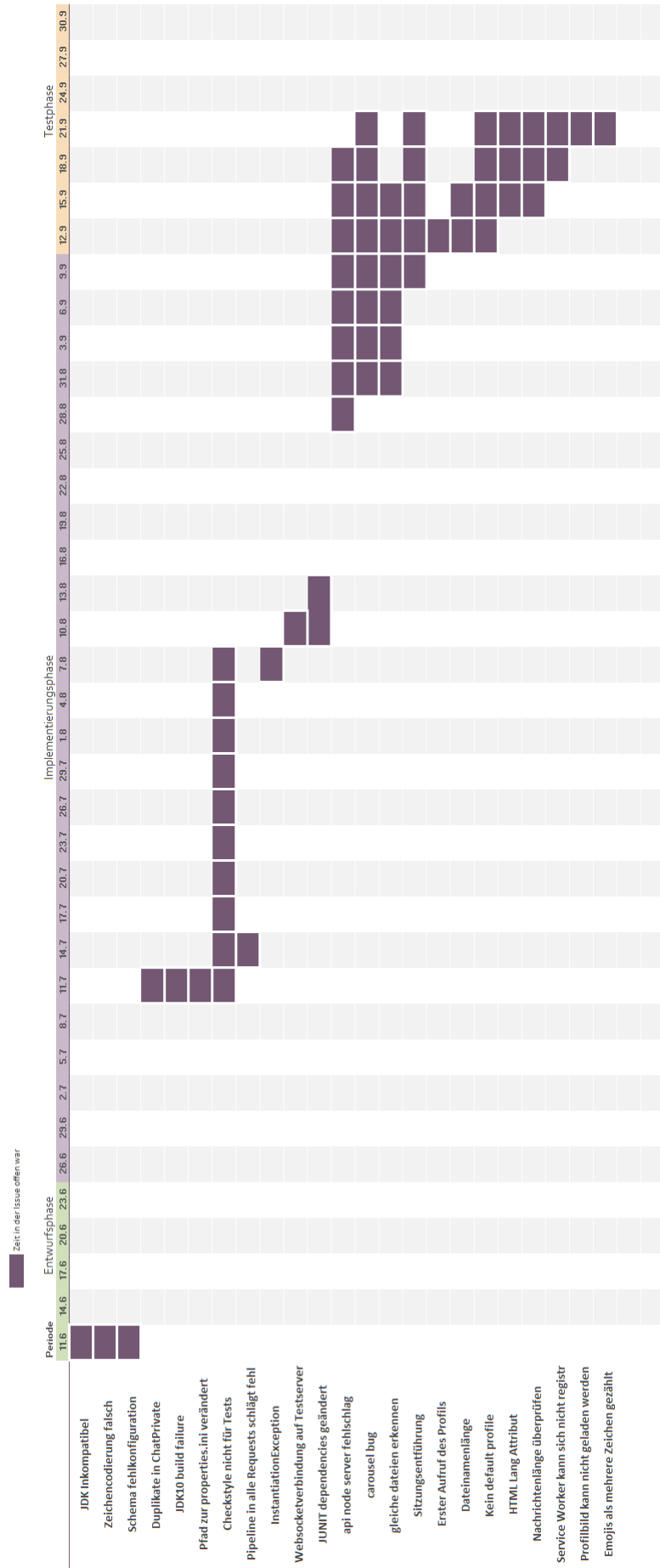


Abbildung 8: Wie lange die Issues zu Bugs offen blieben. Ein Kästchen entspricht drei Tagen.