TuiTalk

Projektarbeit

für die Vorlesung

Verteilte Systeme

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Heidenheim

von

Behr Tobias, Schönig Marc, Seidl Anian

September 2025

Bearbeitungszeitraum 8 Wochen

Erklärung

Wir versichern hiermit, dass wir unsere Projektarbeit mit dem Thema: *TuiTalk* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Heidenheim, September 2025

Behr Tobias, Schönig Marc, Seidl Anian

Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis IV

Abbildungsverzeichnis V

Tabellenverzeichnis VI

Listings VII

1 Einleitung 1

2 Architektur 2

2.1 Anforderungen 2

2.1.1 Funktionale Anforderungen 2

2.1.2 Nichtfunktionale Anforderungen 3

2.2 Systemkomponenten 3

3 Umsetzung 6

4 Reflektion 7

Anhang 9

Abkürzungsverzeichnis

WASM	Web Assembly
CLI	Command Line Interface

Abbildungsverzeichnis

2.1 Architekturdiagramm 5

Tabellenverzeichnis

Listings

1 Einleitung

Kommunikation erfolgt heutzutage über eine Vielzahl von Plattformen. Bekannte Anwendungen wie WhatsApp, Discord oder Signal bieten bereits umfangreiche Möglichkeiten zum Austausch von Nachrichten. Das Ziel dieses Projekts ist die Entwicklung einer einheitlichen Lösung, die sowohl über das Terminal als auch über den Browser genutzt werden kann. Hierfür wird ein Backend implementiert, das durch eine Terminal-Anwendung und eine Weboberfläche ergänzt wird. Ein wesentlicher Unterschied zu bestehenden Lösungen besteht darin, dass die Nutzung ohne vorherige Registrierung möglich ist.

2 Architektur

2.1 Anforderungen

Im Folgenden werden anhand der aufgelisteten internen und externen Stakeholder die funktionalen und nicht-funktionalen Anforderungen aufgeführt.

Interne Stakeholder

- Entwickler
- Betreuer
- Systemadministrator

Externe Stakeholder

- Nutzer

2.1.1 Funktionale Anforderungen

- Client Kommunikation: Als Nutzer und Entwickler möchte ich mich über WebSockets mit dem System verbinden können, wobei ich zwischen einem CLI- und einem WASM-Client entscheiden möchte, damit ich meine präferierte Umgebung verwenden kann. Ich möchte Nachrichten in Echtzeit senden und empfangen, Räumen beitreten und verlassen, sowie meinen Anzeigenamen ändern können. Darüber hinaus soll es möglich sein, vergangene Nachrichten laden zu können, damit ich den gesamten Chatverlauf einsehen kann.
- Loadbalancer: Als Nutzer und Systemadministrator möchte ich, dass ein Loadbalancer die WebSocket Verbindungen gleichmäßig auf die vorhandenen Backends verteilt, damit eine optimale Performance erreicht werden kann. Wenn ein Backend ausfällt, sollen die bestehenden Verbindungen nahtlos vom Loadbalancer zu anderen Backends umgeleitet werden, sodass ich beim Benutzen keine Unterbrechung bemerke.

- Backend Services: Als Entwickler möchte ich, dass die Client Verbindungen vom Backend verwaltet werden und Nutzer spezifische Buffer verwendet werden, um Nachrichten effizient und zuverlässig verarbeiten zu können. Nachrichten sollen über Redis verteilt werden, damit alle Backendinstanzen synchron sind und in einer Datenbank gespeichert werden, um Chatverläufe abrufen zu können. Das System soll Redis Publish/Subscribe im Cluster unterstützen, damit es Skalierbar und Ausfallsicher ist.

2.1.2 Nichtfunktionale Anforderungen

- Hohe Verfügbarkeit: Als Nutzer, Systemadministrator und Betreuer möchte ich, dass das System weiterhin funktioniert, auch wenn ein Backend ausfällt, ohne das der Nutzer eine Unterbrechung mitbekommt. Nach dem Ausfall eines Backends soll die Verbindung automatish von einem anderen Backend übernommen werden ohne das Nachrichten verloren gehen.
- Zuverlässigkeit: Als Nutzer und Systemadministrator möchte ich, dass Nachrichten zugestellt werden, solange Redis und mindestens ein Backend verfügbar sind, so dass Nachrichten nicht verloren gehen, auch wenn Teile des Systems ausfallen. Die Konsistenz der Daten soll durch eine Datenbank sichergestellt werden, sodass keine Nachrichten des Chatverlaufs fehlen und dieser damit eindeutig und zuverlässig ist.
- Skalierbarkeit: Als Entwickler und Betreuer möchte ich, dass das System horizontal skalierbar ist, damit es auch unter hoher Last performant bleibt. Es sollen mehrere Redis Caches und Backends verwendet werden.

2.2 Systemkomponenten

Das System besteht aus fünf Komponenten: *Client*, *Backend*, *Redis-Cluster*, *Loadbalancer* und *Datenbank*.

Client

Es gibt zwei verschiedene Arten von Clients: *Command Line Interface (CLI)* und *Web Assembly (WASM)*. Beide Clients verbinden sich mit dem Loadbalancer, dieser leitet auf eine Backendkomponente weiter, um Nachrichten zu senden und zu empfangen.

Backend

Die Backendkomponente ist ein in Rust geschriebener Websocket-Server. Diese verwaltet die Nachrichten und kommuniziert über das Redis-Cluster mit den anderen

Backendinstanzen. Zu dem persistiert diese die Nachrichten und Informationen über den Nutzer in einer Datenbank. Es werden zur Ausfallsicherheit drei Instanzen aufgesetzt.

Redis-Cluster

Das Redis-Cluster ist die Kernkomponente für die Synchronisation der Nachrichten und stellt die Kommunikation unter den Backendinstanzen her. Dafür wird die Publish und Subscribe Funktion von Redis genutzt. Jeder Chatraum wird als Channel in Redis abgebildet. Jede Nachricht des Clients wird auf den jeweiligen Channel in Redis gepublished. Um eine Nachricht zu erhalten erstellt das Backend für jeden Client einen Subscriber auf dem entsprechenden Channel. Das Redis-Cluster besteht aus sechs Redis-Nodes, drei davon sind Master-Nodes und die restlichen drei sind zu den Master-Nodes die jeweiligen Replicas.

Loadbalancer

Der Loadbalancer ist eine Komponente, die die Verbindungen zwischen den Clients und den Backends verteilt. Als Loadbalancer wird Nginx genutzt, da dieser einer der renomiertesten in dieser Technologie ist. Als Strategie zur Aufteilung auf die Backendinstanzen wird nach dem Verfahren der wenigsten Verbindungen vorgegangen.

Datenbank

Die Datenbank ist eine PostgreSQL-Datenbank, die die Nachrichten und Informationen der Nutzer speichert. Die Backendinstanzen verbinden sich mit der Datenbank, um die Nachrichten zu laden und zu speichern. Zusätzliche werden die Nutzerinformationen und in welchem Chatraum diese sich befinden gespeichert. Die Datenbank ist keine Systemkritischen Komponente und sorgt im Falle eines Ausfalls nur für eine Funktionseinschränkung.

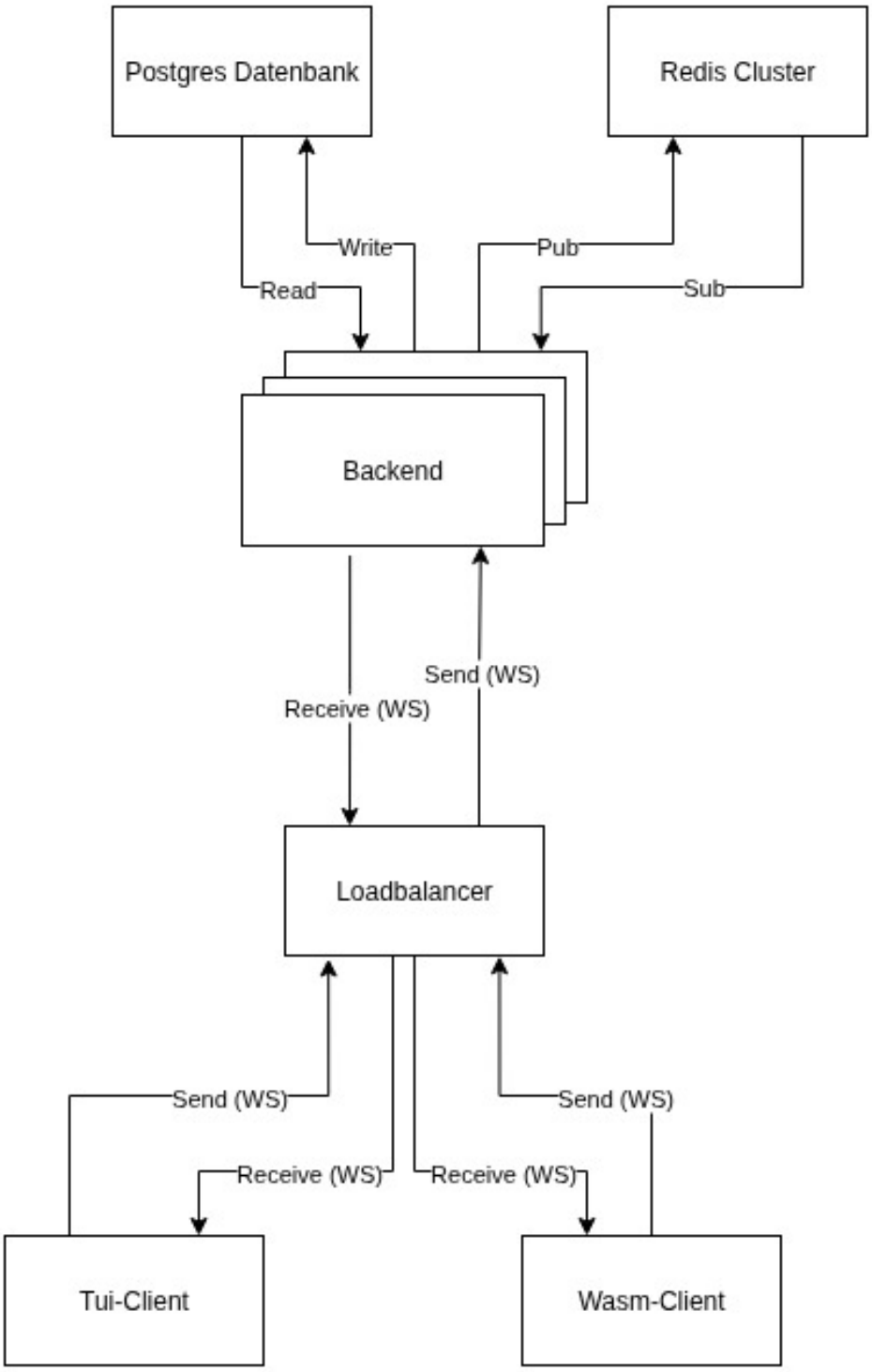

Abbildung 2.1: Architekturdiagramm

3 Umsetzung

4 Reflektion

Anhang