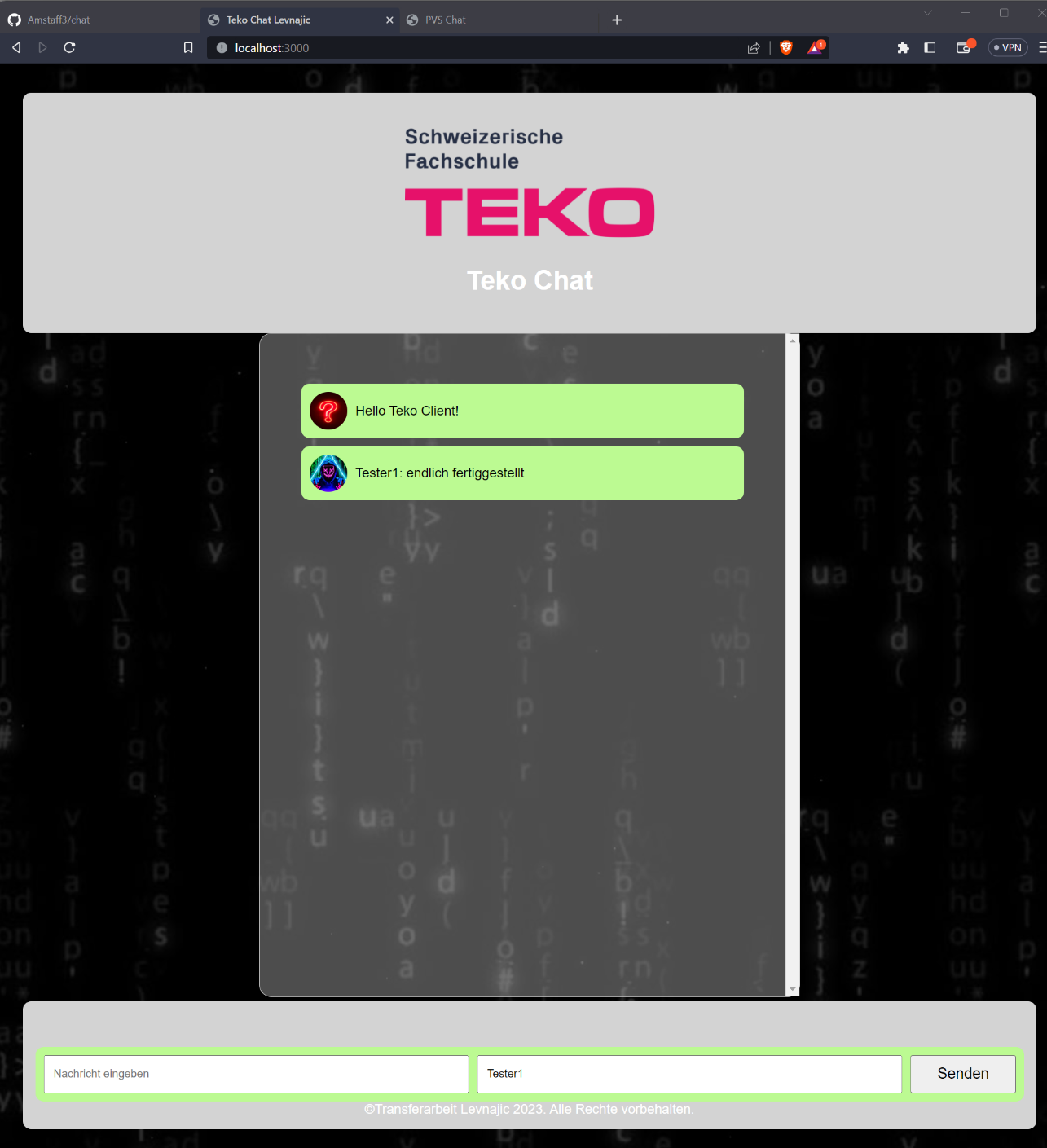
**Dokumentation der Teko Chat-Anwendung**



|  |
| --- |
| **Firma: TEKO Olten** |
| **Titel: Transferarbeit Parallele und verteilte Systeme** |
| **Author: Levnajic Zeljko** |
| **Klasse: O-TIN-20-S-a** |
| **Fachbetreuer: Patrick Michel** |

Inhaltsverzeichnis

[Einführung 4](#_Toc138194022)

[Architektur der Anwendung 4](#_Toc138194023)

[Client: 4](#_Toc138194024)

[Server: 4](#_Toc138194025)

[api.js: 4](#_Toc138194026)

[websocketserver.js: 4](#_Toc138194027)

[Docker: 4](#_Toc138194028)

[grobe Erklärung der Docker-Datei: 5](#_Toc138194029)

[app.js: 5](#_Toc138194030)

[Paketabhängigkeiten 5](#_Toc138194031)

[Installation und Starten der Anwendung 5](#_Toc138194032)

[Optionaler Starten der Anwendung 5](#_Toc138194033)

[Vorschläge zur ausfallsicherheit 6](#_Toc138194034)

[1. WebSocket-Server 6](#_Toc138194035)

[Um den WebSocket-Server ausfallsicherer zu machen, kann man folgende Massnahmen ergreifen: 6](#_Toc138194036)

[ Implementierung eines Lastenausgleichs: 6](#_Toc138194037)

[ Verwendung von Clustering: 6](#_Toc138194038)

[ Implementierung von Überwachungsmechanismen: 6](#_Toc138194039)

[2. Datenbank (Redis) 6](#_Toc138194040)

[ Implementierung von Replikation und Sharding: 6](#_Toc138194041)

[ Durchführung regelmässiger Backups: 6](#_Toc138194042)

[ Verwendung einer geeigneten Überwachungslösung: 6](#_Toc138194043)

[3. Webanwendung 6](#_Toc138194044)

[ Implementiere Caching-Mechanismen: 6](#_Toc138194045)

[ Nutze Content Delivery Networks (CDNs): 6](#_Toc138194046)

[ Verwende eine geeignete Fehlererkennung und -behandlung: 6](#_Toc138194047)

[Begrifserkläreungen 7](#_Toc138194048)

[Replikation 7](#_Toc138194049)

[Sharding 7](#_Toc138194050)

[RDB oder Redis Cluster 7](#_Toc138194051)

[Redis Sentinel 7](#_Toc138194052)

[Prometheus und Grafana 7](#_Toc138194053)

[Caching-Mechanismen 7](#_Toc138194054)

[Content Delivery Networks (CDNs) 7](#_Toc138194055)

[Quellen 8](#_Toc138194056)

[Redlichkeitserklärung 8](#_Toc138194057)

# Einführung

Im Rahmen der Transferarbeit wurde eine Chat-Anwendung entwickelt, die von mir Levnajic Zeljko und [Amstaff3](https://github.com/Amstaff3/chat/tree/main) auf GitHub bereitgestellt wird. Diese Anwendung stellt ein einfaches Chat-System dar, das Websockets zur Kommunikation zwischen Benutzern verwendet. Der Code der Anwendung basiert hauptsächlich auf JavaScript und umfasst auch CSS, HTML, Dockerfile.

Die Chat-Anwendung ermöglicht es Benutzern, in Echtzeit miteinander zu kommunizieren und Nachrichten auszutauschen. Durch die Verwendung von Websockets können Benutzer nahtlos und sofort auf Nachrichten reagieren, ohne die Seite aktualisieren zu müssen.

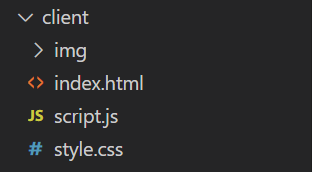
Der Code der Chat-Anwendung ist auf [GitHub](https://github.com/Amstaff3/chat/tree/main) verfügbar. Die Anwendung wurde mit einer Vielzahl von Technologien entwickelt, die uns bereitgestellt wurden, darunter JavaScript für die Server- und Client-Seite, Html und CSS für das Styling der Benutzeroberfläche und Docker für die einfache Bereitstellung in einem Container.

Diese Chat-Anwendung bietet eine grundlegende Funktionalität zum Senden und Empfangen von Nachrichten.

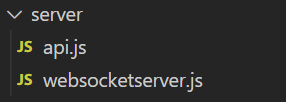
Um die Chat-Anwendung zu verwenden, müssen bestimmte Voraussetzungen erfüllt sein, wie z.B. die Installation von Docker und Node.js. Die Anwendung kann dann lokal gestartet werden, und Benutzer können über den Webbrowser auf die Chat-Oberfläche zugreifen.

# Architektur der Anwendung

Die Anwendung ist in drei Hauptkomponenten unterteilt:

Client: Die Client-Seite der Anwendung, die aus den HTML-, CSS- und JavaScript-Dateien besteht, die zur Darstellung der Benutzeroberfläche und zur Interaktion mit dem Server über Websockets verwendet werden. Die Hauptdateien sind `index.html`, script.js und style.css.

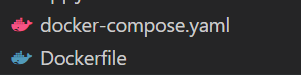
Server: Die Server-Seite der Anwendung, die die WebSocket-Kommunikation verwaltet und Anfragen vom Client beantwortet. Die Hauptdateien sind `api.js` und websocketserver.js.



api.js: Diese Datei definiert die API-Endpunkte für den Chat-Service. Sie enthält Funktionen zum Behandeln von Anfragen zum Senden und Empfangen von Nachrichten, zur Anmeldung und Abmeldung von Benutzern usw.

websocketserver.js: Dieses Skript implementiert den WebSocket-Server, der für die Echtzeitkommunikation zwischen dem Chat-Client und dem Server verwendet wird. Es verwaltet WebSocket-Verbindungen, behandelt eingehende Nachrichten und sendet Nachrichten an alle verbundenen Clients.

Docker: Die Anwendung verwendet Docker für die Containerisierung. Das **Dockerfile** beschreibt, wie der Docker-Container aufgebaut sein soll. Es definiert das Node.js-Image als Basis, installiert die notwendigen Abhängigkeiten und kopiert die notwendigen Dateien in den Container. Es legt auch die Ports fest, die die Anwendung öffnen soll und startet die Anwendung beim Start des Containers.



Die Anwendung verwendet auch **docker-compose.yaml**, um die Dienste der Anwendung zu definieren und zu konfigurieren. Es definiert zwei Dienste, **chat** und **redis**, legt ihre Abhängigkeiten fest und konfiguriert die Netzwerkports, die sie verwenden sollen.

## grobe Erklärung der Docker-Datei:

* Die Docker-Datei verwendet das offizielle Node.js-Image als Basis für den Container.
* Sie legt das Arbeitsverzeichnis im Container auf /app fest und kopiert die package.json und package-lock.json in den Container.
* Es installiert die Abhängigkeiten mithilfe von yarn install.
* Der Code wird in den Container kopiert und der Webserver wird gestartet.
* Die Anwendung öffnet den Port 3000 im Container für den Zugriff auf die Anwendung.
* Zusätzlich öffnet sie den Port 80 im Container, der für den Nginx-Webserver verwendet wird.

Diese grobe Analyse gibt Ihnen einen Überblick über den Server-Code und die Docker-Datei der Chat-Anwendung. Bei Bedarf können Sie weitere Details in den entsprechenden Dateien im Projekt Nachschauen.

app.js: Dieses Skript erstellt den Express-Server und konfiguriert die verschiedenen Routen und Middleware-Funktionen. Es behandelt HTTP-Anfragen und leitet sie an die entsprechenden Endpunkte weiter.

## Paketabhängigkeiten

Die Anwendung hat mehrere Abhängigkeiten, die in der package.json-Datei aufgeführt sind. Dazu gehören unter anderem express, ws (Websockets), redis (für die Datenspeicherung), und nodemon (für die Entwicklung).

Bitte beachten sie , dass ich in dieser Beschreibung nicht auf die Installation oder Konfiguration der einzelnen Pakete und Abhängigkeiten eingehen werde.

# Installation und Starten der Anwendung

Die Anwendung verwendet Docker und Node.js. Stellen Sie sicher, dass diese Tools auf Ihrem System installiert sind, bevor Sie fortfahren.

**1.** Zuerst muss das Repository geklont werden. Dies kann mit dem Befehl `git clone <https://github.com/Amstaff3/chat.git> erreicht werden.

**2.** Nach dem Klonen müssen die Abhängigkeiten installiert werden. Dies kann durch Ausführen von **yarn install** im Hauptverzeichnis des Projekts erreicht werden.

**3.** Anwendung in einer Docker-Umgebung zu starten, führen Sie **docker compose up -d** aus.

**4.** Entwicklungsumgebung zu starten, führen Sie **yarn dev** aus.

**5.** Produktionsumgebung zu starten, führen Sie **yarn prod** aus.

**6.** Die Anwendung kann dann über **http://localhost:3000** aufgerufen werden.

## Optionaler Starten der Anwendung

Die Anwendung kann gestartet werden, indem das Bash-Skript `**start.sh**` ausgeführt wird. Dieses Skript startet die Anwendung, indem es `node app.js` aus

# Vorschläge zur ausfallsicherheit

## WebSocket-Server

## Um den WebSocket-Server ausfallsicherer zu machen, kann man folgende Massnahmen ergreifen:

* Implementierung eines Lastenausgleichs:

Verwenden Sie eine Load Balancer-Software wie Nginx oder HAProxy, um den WebSocket-Verkehr auf mehrere Server zu verteilen.

* Verwendung von Clustering:

Nutzen Sie Tools wie Kubernetes oder Docker Swarm, um mehrere Instanzen des WebSocket-Servers zu erstellen und die Last zwischen ihnen zu verteilen.

* Implementierung von Überwachungsmechanismen:

Verwenden Sie Überwachungstools wie Prometheus und Grafana, um den Status des WebSocket-Servers zu überwachen und bei einem Ausfall automatisch einen Neustart oder eine Umschaltung auf eine andere Instanz durchzuführen.

2. Datenbank (Redis)

Um die Datenbank ausfallsicherer zu machen, kann man folgende Massnahmen ergreifen:

* Implementierung von Replikation und Sharding:

Konfigurieren Sie Redis für die Master-Slave-Replikation, um Kopien der Datenbank auf verschiedenen Servern zu erstellen. Verwenden Sie Sharding-Techniken, um die Datenbanklast auf mehrere Redis-Instanzen aufzuteilen.

* Durchführung regelmässiger Backups:

Konfigurieren Sie automatisierte Backups der Redis-Datenbank, z. B. mit Tools wie Redis RDB oder Redis Cluster.

* Verwendung einer geeigneten Überwachungslösung:

Nutzen Sie Tools wie Redis Sentinel oder Redis Cluster, um den Status der Redis-Instanzen zu überwachen und bei Bedarf automatisch Massnahmen zu ergreifen.

3. Webanwendung

Um die Webanwendung ausfallsicherer zu machen, kann man folgende Massnahmen ergreifen:

* Implementiere Caching-Mechanismen:

Die Leistung der Webanwendung zu verbessern und die Last auf den Server zu reduzieren. Dadurch werden auch bei einem Ausfall des Servers bestimmte Inhalte weiterhin verfügbar sein.

* Nutze Content Delivery Networks (CDNs):

Die statische Inhalte der Webanwendung zu verteilen und die Ladezeiten zu optimieren. Dadurch wird die Abhängigkeit vom Hauptserver reduziert und die Ausfallsicherheit verbessert.

* Verwende eine geeignete Fehlererkennung und -behandlung:

Die Fehlerzustände in der Webanwendung abzufangen und darauf angemessen zu reagieren.

Diese Vorschläge dienen als Ausgangspunkt, um die Ausfallsicherheit der Systemkomponenten zu verbessern. Je nach den spezifischen Anforderungen deiner Anwendung und der verfügbaren Ressourcen können weitere Massnahmen erforderlich sein.

# Begrifserkläreungen

Replikation: Kopien der Datenbank auf mehreren Servern erstellt. Wenn eine Änderung auf einem Server vorgenommen wird, wird sie automatisch auf die anderen Server repliziert. Dadurch erhöht sich die Verfügbarkeit der Datenbank, da im Falle eines Ausfalls eines Servers die Daten immer noch von den anderen Servern abgerufen werden können.

Sharding: Eine Technik, bei der die Datenbanklast auf mehrere Server verteilt wird. Anstatt alle Daten auf einem einzigen Server zu speichern, werden die Daten in logische Teilmengen aufgeteilt und auf verschiedene Server verteilt. Dadurch kann die Datenbank die Last besser bewältigen und eine bessere Skalierbarkeit erzielen.

RDB oder Redis Cluster: RDB steht für Redis Database Backup und ist ein Mechanismus zum Erstellen von Backups der Redis-Datenbank. Es ermöglicht Ihnen, regelmässige Snapshots der Datenbank zu erstellen und diese bei Bedarf wiederherzustellen. Redis Cluster ist eine spezielle Konfiguration von Redis, die es ermöglicht, die Datenbank auf mehreren Servern zu betreiben und die Daten über verschiedene Clusterknoten zu verteilen.

Redis Sentinel: Redis Sentinel ist ein Überwachungs- und Failover-System für Redis. Es überwacht den Status der Redis-Instanzen und erkennt automatisch Ausfälle. Bei einem Ausfall kann Redis Sentinel automatisch den Betrieb auf einen anderen Redis-Server umschalten, um die Verfügbarkeit der Datenbank sicherzustellen.

Prometheus und Grafana: Prometheus ist ein Open-Source-Überwachungssystem, das Daten von verschiedenen Quellen sammelt und speichert. Es ermöglicht das Überwachen von Metriken wie CPU-Auslastung, Speichernutzung und Netzwerkdaten. Grafana ist ein Open-Source-Dashboard-Tool, das die gesammelten Daten von Prometheus visualisiert und benutzerfreundlich darstellt.

Caching-Mechanismen: Caching ist ein Mechanismus, bei dem häufig abgerufene Daten temporär gespeichert werden, um die Ladezeiten zu verkürzen. Wenn ein Client eine Anfrage stellt, überprüft das System zuerst den Cache und gibt die gespeicherten Daten zurück, falls vorhanden. Dadurch wird die Notwendigkeit reduziert, die Daten erneut aus der eigentlichen Datenquelle abzurufen.

Content Delivery Networks (CDNs): CDNs sind Netzwerke von Servern, die weltweit verteilt sind und statische Inhalte wie Bilder, Videos oder Dateien speichern. Wenn ein Benutzer auf eine Webseite zugreift, werden die Inhalte über den nächstgelegenen Server des CDNs bereitgestellt. Dadurch werden die Ladezeiten verkürzt und die Bandbreitenanforderungen für den Hauptserver reduziert.

# Quellen

- Internet

- YouTube

- Git

- Schulunterlagen

- ChatGPT

# Redlichkeitserklärung

Ich erkläre hiermit, dass ich an diesem Projekt, einer Chat-App mit Client usw, allein und ohne fremde Hilfe gearbeitet habe. Alle Quellen sowie Inhalte, die hierfür verwendet wurden, sind korrekt angegeben.

Unterschrift [Levnajic] / [20.06.2023]