Client/Server Chat Anwendung mit WebSockets

Dokumentation für "Projekt Informatik"

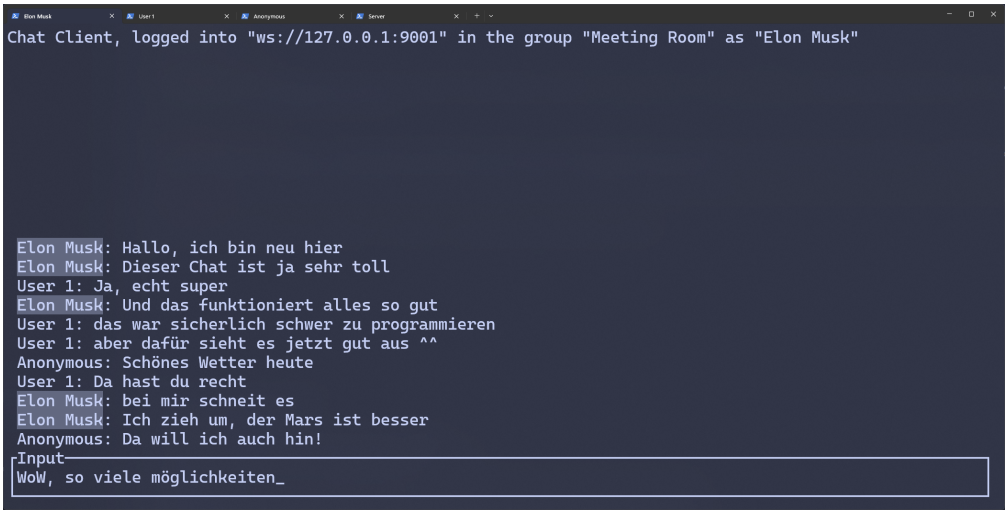
März 2024

Inhaltsverzeichnis

1 Einleitung

Diese Client/Server Chat Anwendung wurde in der Programmiersprache Rust entwickelt. Der Client tritt einem Chat bei, indem er seinen Namen, eine Gruppe und die Serveradresse angibt. Die Gruppe bestimmt, was für Nachrichten dem Client angezeigt werden und ermöglicht eine 'private' Unterhaltung in einer Art Chat-Raum. Der Server verwaltet die Gruppen und leitet Nachrichten an die entsprechenden Clients weiter. Beim Verbinden von einem neuen Client werden alle vorherigen Nachrichten aus der Gruppe an den Client gesendet. Die Kommunikation zwischen Client und Server erfolgt über WebSockets und ist somit im lokalen Netzwerk oder über das Internet möglich.

1.1 Ausführen



Die Binärdateien können direkt ausgeführt werden, ohne irgendwelche Dependencies. Oder das Projekt kann mit `cargo` kompiliert und ausgeführt werden. Zum Kompilieren wird Rust benötigt, welches auf dieser Seite heruntergeladen werden kann: `https://www.rust-lang.org/tools/install`.

```
// Kompilieren und ausführen
cargo run --release --bin chat-server
cargo run --release --bin chat-client

// Kompilieren
cargo build --release
./target/release/chat-server
./target/release/chat-client
```

Es können auch mehr detailierte Logs aktiviert werden, indem die Umgebungsvariable `RUST_LOG` gesetzt wird.

```
RUST_LOG=debug cargo run --release --bin chat-server
```

oder die logs können ausgeschaltet werden

```
RUST_LOG=off ./target/release/chat-client
```

1.2 Protokoll

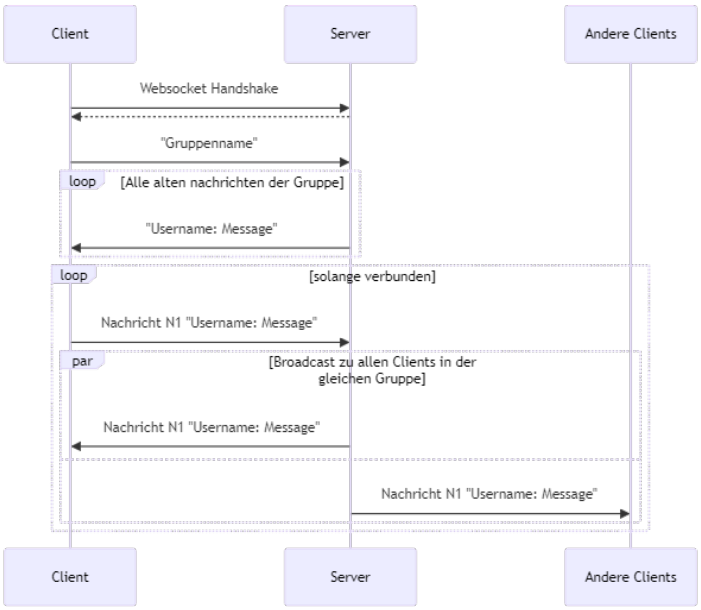


Abbildung 1: Das Protokoll

Der Client baut eine WebSocket-Verbindung zum Server auf und sendet einen Handshake. Danach wählt der Client eine Gruppe und sendet eine Nachricht an den Server. Der Server sendet alle gespeicherten Nachrichten in der Gruppe an den Client. Wenn der Server eine Nachricht von einem Client empfängt, leitet er diese an alle anderen Clients in der Gruppe weiter.

1.3 Bibliotheken

Das Projekt verwendet die folgenden Bibliotheken:

- `tokio` für asynchrone Operationen und Multithreading
- `tokio-tungstenite` für die WebSocket-Kommunikation
- `sqlx` für die Datenbankanbindung
- `ratatui` für die Benutzerschnittstelle
- `crossterm` für die Eingabeverarbeitung
- `env_logger` für das Logging

2 Client

2.1 main.rs

Die Datei `main.rs` enthält die Hauptfunktion des Chat-Clients. Sie startet die Anwendung, initialisiert den Logger, erfasst den Benutzernamen, die Gruppe und die Serveradresse über die Standardeingabe und führt die Anwendung aus. Wenn die Anwendung die Verbindung zum Server verliert, wird eine Fehlermeldung ausgegeben.

2.2 app.rs

Die `Application`-Klasse ist verantwortlich für die Verwaltung der Anwendung, die WebSocket-Kommunikation, die Benutzerschnittstelle und das Modell. Sie enthält Informationen über die URL des Websockets, den Benutzernamen, die Gruppe, die Eingabeverarbeitung, die Benutzerschnittstelle (TUI), das Datenmodell und den WebSocket.

2.3 input.rs

Die Datei `input.rs` implementiert die `EventHandler`-Klasse, die für die Verarbeitung von Eingabe Events im Terminal zuständig ist. Die `EventHandler`-Klasse erstellt einen Kanal (`mpsc`), um Ereignisse an den Hauptthread zu senden, und startet einen Thread, um Ereignisse aus dem Terminal zu lesen. Dieser Thread verarbeitet Tastatureingaben und gibt entsprechende Ereignisse zurück, wie z.B. `Quit`, `Send`, `Backspace`, `Resize`, `Refresh`, und `Input`. Die `next`-Methode wartet auf das nächste Ereignis und gibt es zurück.

2.4 model.rs und tui.rs

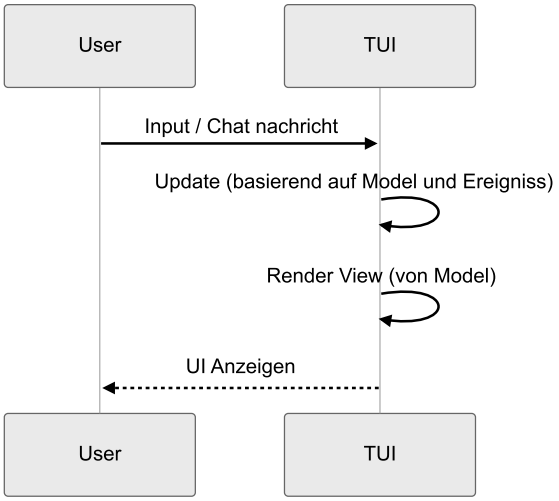


Abbildung 2: Die Elm Architektur

Die `Model`-Klasse entspricht dem Modell in der Elm-Architektur. Sie speichert Informationen über die URL des Websockets, den Benutzernamen, die aktuelle Eingabe im Textbereich, die Gruppe und die empfangenen Nachrichten. Ähnlich wie in Elm wird das Modell als einziger, unveränderbarer Zustand betrachtet, der die gesamte Anwendungsdaten enthält.

Die `ChatMessage`-Klasse repräsentiert eine einzelne Chat-Nachricht und implementiert Methoden zum Serialisieren und Deserialisieren.

Die `TUI`-Klasse ist für das Rendering der Benutzerschnittstelle verantwortlich. Sie verwendet die `ratatui`-Bibliothek, um Textelemente und Widgets anzuzeigen. Ähnlich wie in Elm wird die Benutzerschnittstelle in einem funktionalen Stil gerendert, wobei die render-Methode die Darstellung des aktuellen Modells auf dem Bildschirm steuert.

2.5 websocket.rs

Die Klasse `Websocket` repräsentiert eine WebSocket-Verbindung. Sie enthält Kanäle zum Lesen und Schreiben von Nachrichten mit asynchronem Code. Sobald eine neue Nachricht empfangen wird, wird diese über einen Kanal an den Hauptthread gesendet mithilfe der `receive`-Methode. Das Senden von Nachrichten erfolgt in einem separaten Thread.

3 Server

3.1 main.rs

Die Datei `main.rs` enthält die Hauptfunktion des Chat-Servers. Sie initialisiert die Anwendung, konfiguriert das Logging und fragt den Benutzer nach dem Server-Port. Falls der Benutzer keinen Port eingibt, wird der Standardport '9001' verwendet.

3.2 app.rs

Die Datei `app.rs` enthält die Implementierung der Anwendungslogik für den Chat-Server. Sie definiert die Klasse `Application`, die die zentrale Anwendung darstellt. Diese Klasse verwaltet die Verbindungen zu Clients, die Datenbankverbindung und koordiniert die Kommunikation zwischen den Clients über WebSockets.

- Die Klasse `Application` enthält Felder für die Serveradresse (`address`), die Verbindungen zu den Clients (`connections`) und die Datenbankverbindung (`db`).
- Die Methode `new` erstellt eine neue Anwendungsinstanz mit der angegebenen Serveradresse.
- Die Methode `run` führt die Anwendungslogik aus, indem sie eine Verbindung zur Datenbank herstellt, WebSocket-Verbindungen akzeptiert und eingehende Nachrichten von den Clients verarbeitet.
- Die Methode `on_connection` wird aufgerufen, wenn eine neue Verbindung zu einem Client hergestellt wird. Sie behandelt das Lesen und Senden von Nachrichten sowie das Speichern von Nachrichten in der Datenbank.

3.3 connection.rs

Wandelt eine WebSocket-Verbindung um in eine `Connection`-Klasse, die sicherstellt, dass dem Protokoll in der Abbildung 1 gefolgt wird.

3.4 websocket.rs

Siehe: 2.5 Client `websocket.rs`

3.5 database.rs

```
CREATE TABLE IF NOT EXISTS messages (
    id INTEGER PRIMARY KEY,
    group_name TEXT NOT NULL,
    username TEXT NOT NULL,
    message TEXT NOT NULL
)
```

Die Datei `database.rs` enthält Funktionen zur Verwaltung der SQLite-Datenbank.

- Die Funktion `establish_connection` erstellt eine Verbindung zur SQLite-Datenbank, die von mehreren Threads geteilt werden kann.
- Die Funktion `insert_message` fügt eine neue Nachricht in die Datenbank ein.
- Die Funktion `get_messages` ruft Nachrichten aus der Datenbank ab, die einer bestimmten Gruppe zugeordnet sind.