Dokumentation für FWE-NHIE

Ha Hoang Anh Le - 1114545
Vu Lam Le - 1113690
Dang Quang Tran - 1113204
Guirauld Noumboussi Tayon-768891
Johannes Jäger - 772518

I. Use Case

Am Anfang wurde die Webseite für die Erfüllung folgender Use Cases entwickelt. Es gibt vier relevante Bereiche für Use Cases: User, Playlist, Question und Room.

User

- Erhalten Information eines bestimmten Benutzers
- Erhalten alle existierten Benutzern
- Erhalten alle Wiedergabelisten von einem existierten Benutzern
- Registrieren neuen Benutzer
- Anmelden existierten Benutzer mit Benutzernamen und Passwort
- Verändern persönliche Informationen eines existierten Benutzers
- Fügt der Benutzer eine oder mehrere Wiedergabeliste/n zu seiner Lieblings-Playlist hinzu sowie verändert sie
- Löschen einen existierten Benutzer
- Löschen eine oder mehrere Wiedergabeliste/n aus der Lieblings-Playlist von einem existierten Benutzern

Playlist

- Erhalten eine bestimmte Wiedergabeliste
- Erhalten alle existierten Wiedergabelisten
- Erstellt der Benutzer seine neue Wiedergabeliste
- Verändert der Benutzer den Name seiner existierten Wiedergabeliste sowie fügen Frage/n zur ihr hinzu
- Löscht der Benutzer seine erstellte Wiedergabeliste
- Löscht der Benutzer eine oder mehrere Frage/n von seiner erstellten Wiedergabeliste

Question

- Erhalten eine bestimmte Frage
- Erhalten alle existierten Fragen
- Erstellt der Benutzer seine neue Frage
- Verändert der Benutzer seine erstellte Frage
- Löscht der Benutzer seine erstellte Frage

Room

- Erhalten alle Spieler von einem Raum
- Erstellen neuen Raum mit einer ausgewählten Default-Wiedergabeliste
- Schließen einen existierten Raum an
- Verlassen den Raum
- Stimmen die Spieler dafür/ dagegen im Raum
- Chatten die Spieler miteinanders im Raum
- Löschen einen existierten Raum

- Kicken ein oder mehrere Spielern von einem existierten Raum durch Mehrheit-Stimmen

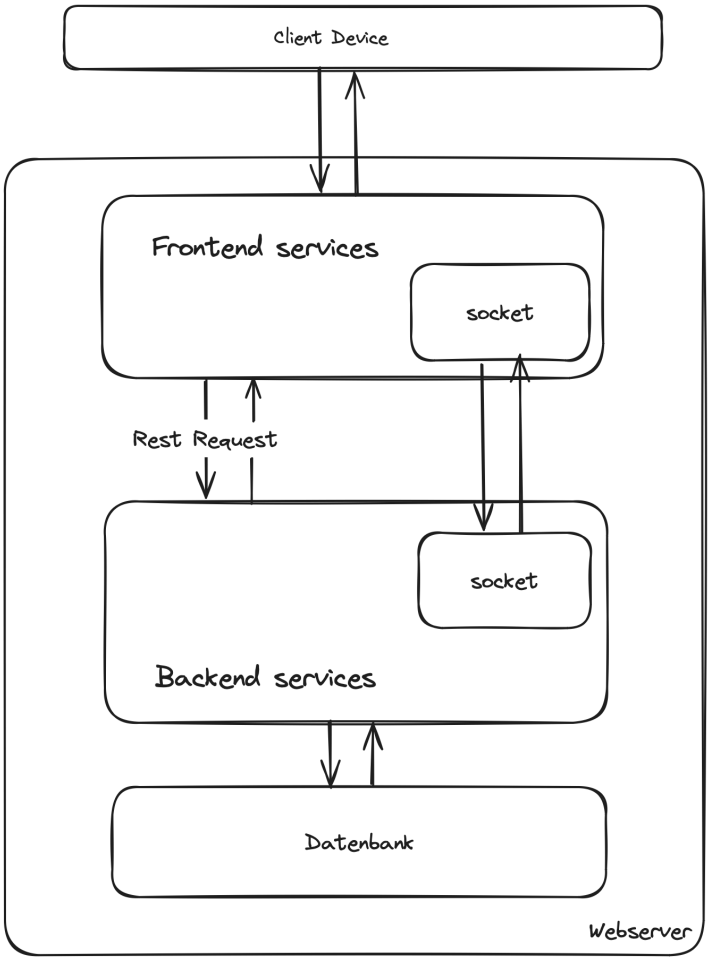
II. Überblick über Design

Wir haben uns entschieden, die Website als RESTful API zu gestalten. Daher erfolgt die Kommunikation über Rest- Request und Reply. Über geeignete URLs und Methoden werden entsprechende Aktionen vom Backend ausgeführt.

Die Website ist für das Multiplayer-Spielen konzipiert. Das bedeutet, dass wir Folgendes sicherstellen müssen:

- Benutzer können einen neuen Raum erstellen, dem andere Benutzer beitreten können.
- Alle Aktionen jedes Benutzers auf der Website müssen mit den Websites anderer Benutzer synchronisiert werden.

Da REST-API die oben genannten Dienste nicht bereitstellen kann, mussten wir Socket I/O verwenden. Nachfolgend ist die Zeichnung, die unser Design beschreibt.

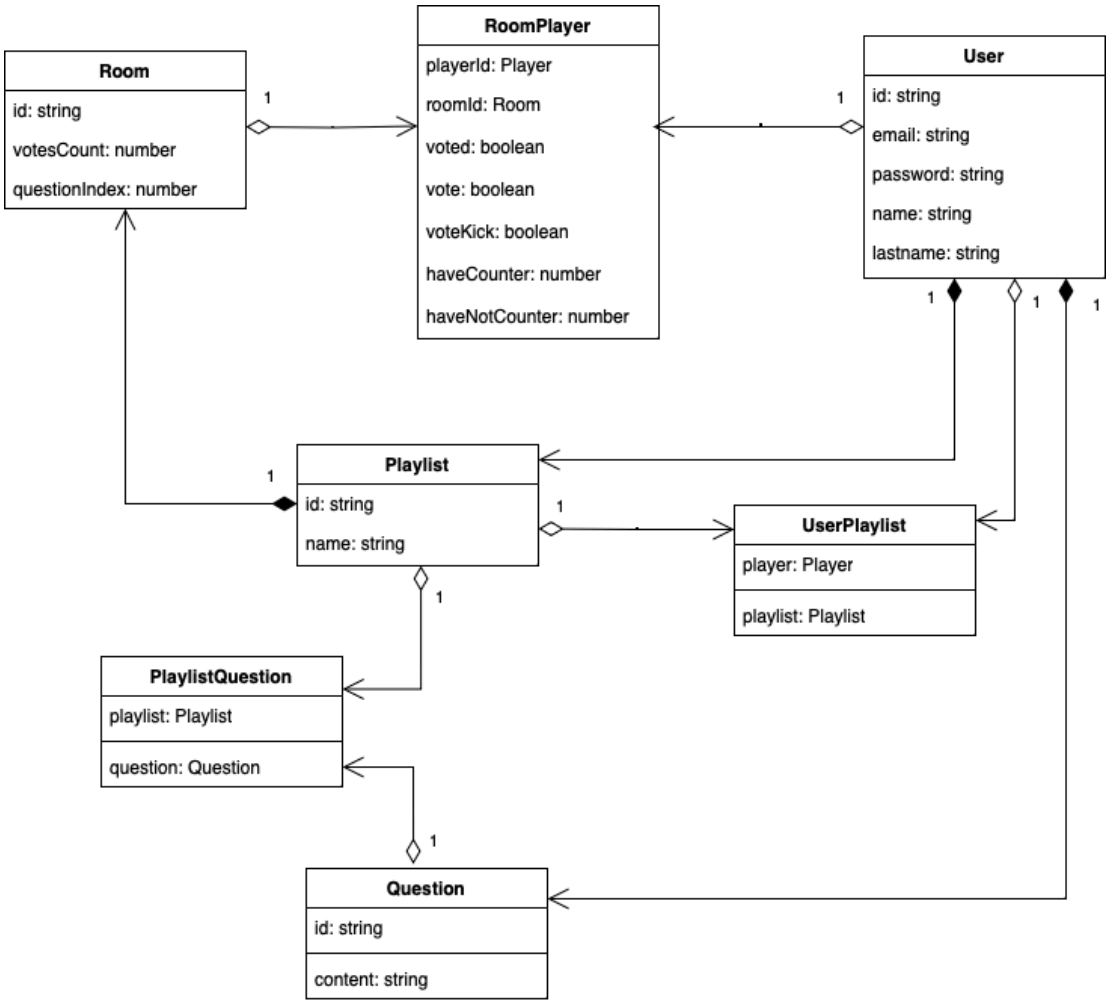


III. Backend Implementierung

1. Entität - Implementierung

Um die in Abschnitt 1 schon erwähnten Bereiche anzupassen, haben wir die folgenden Entitäten für die Datenbank wie folgt gestaltet. Außerdem werden alle Objekte durch ID als Primärschlüssel verwaltet, um leichter zuzugreifen.

Unten ist das vollständige UML-Diagramm.



User

Ein Benutzer hat seine persönliche Information wie E-Mail, Passwort, Vorname und Lastname. Er kann eine oder mehrere Playlists in seiner Lieblings-Playlist hinzufügen. Außerdem können eine oder mehrere Fragen oder Playlists von Benutzern erstellt werden. Er kann auch einen oder mehrere Spielräume eintreten.

Attribute:

id: string;
email: string;
password: string;
firstName: string;
lastName: string;
roomPlayers: array <RoomPlayer>;
userPlaylists: array <UserPlaylist>;
createdPlaylists: array <Playlist>;
createdQuestions: array <Question>

Playlist

Eine Playlist hat ihren Namen und ihren Ersteller. Eine oder mehrere Playlists können zur Lieblings-Playlist von einem oder mehreren Benutzern hinzugefügt werden. Eine Playlist enthält eine oder mehrere Fragen. Sie wird auch als Default-Playlist für einen Spielraum ausgewählt.

Attribute:

id: string;
name: string;
room: array <Room>;
userPlaylists: array <UserPlaylist>;
playlistQuestions: array <PlaylistQuestion>;
creator: User

Question

Eine Frage hat ihren Inhalt und ihren Ersteller. Eine oder mehrere Fragen können zu einer oder mehreren Playlists hinzugefügt werden.

Attribute:

id: string;
content: string;
playlistQuestions: array <PlaylistQuestion>;
creator: User

Room

Ein Spielraum muss eine Playlist als Default ausgewählt werden. Außerdem muss er sowohl die Anzahl der Stimmen von Spielern, als auch die aktuelle Frage zeigen. Er erlaubt, mehrere Spieler einzutreten.

Attribute:

id: string;
playlist: Playlist;
roomPlayers: array <RoomPlayer>;
votesCounter: number = 0;
questionIndex: number = 0;

Userplaylist

Dies ist eine Zwischentabelle zur Darstellung der Beziehung “Many-To-Many” zwischen dem Benutzern und die Playlists in seiner Lieblings-Playlist.

Attribute:

player: User;
playlist: Playlist

RoomPlayer

Dies ist eine Zwischentabelle zur Darstellung der Beziehung “Many-To-Many” zwischen dem Spielraum und seinen Spielern. Diese Tabelle wird nicht nur die Anzahl für das Kicken eines Spielers, sondern auch die Anzahl von Stimmen dafür sowie dagegen gespeichert.

Attribute:

playerId: User;
roomId: Room;
voted: boolean = false;
vote: boolean = false;
voteKick: number = 0;
haveCounter: number = 0;
haveNotCounter: number = 0;

PlaylistQuestion

Dies ist eine Zwischentabelle zur Darstellung der Beziehung “Many-To-Many” zwischen der Playlists und ihrer Fragen

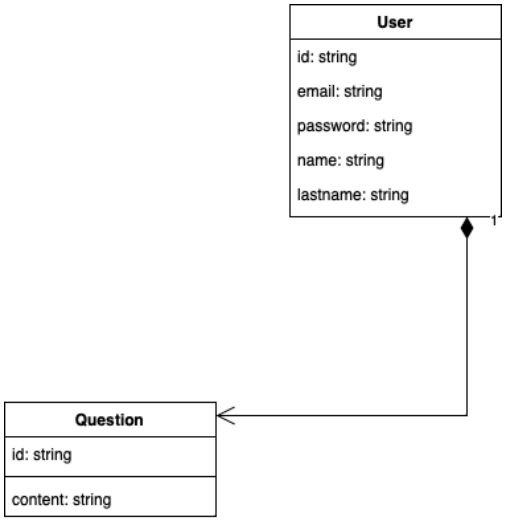
Attribute:

playlist: Playlist;
question: Question

2. Beziehung Implementierung

Wir werden unten näher auf die Beziehungen zwischen den einzelnen Entitäten eingehen.

a) User-Question Beziehung

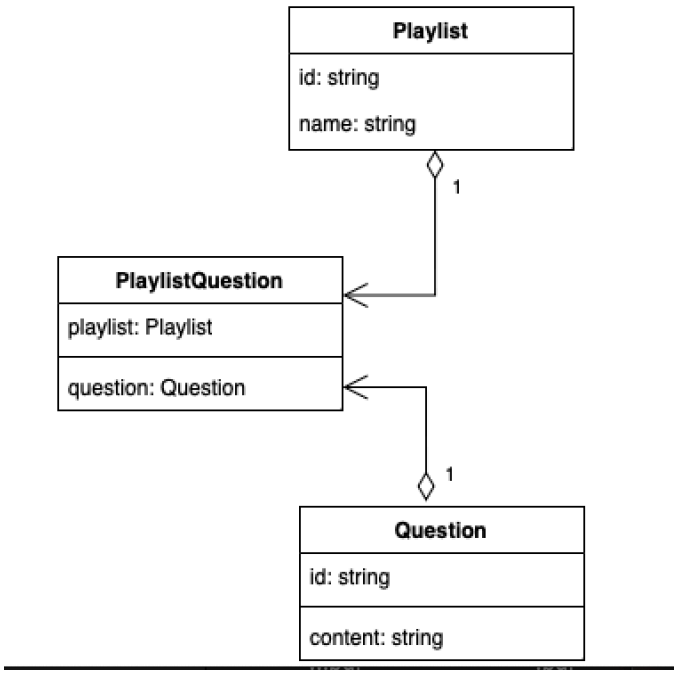


Jede Frage kann nur von einem Benutzer erstellt werden. Daher wurde Beziehung Composition verwendet. Dies gilt jedoch nicht für Default-Fragen. Die Default-Fragen werden direkt in der Datenbank beim Starten des Servers hinzugefügt. Um die Default-Fragen mit benutzergenerierten Fragen zu unterscheiden, werden wir den Erzeuger von Default-Fragen auf null setzen.

Durch diese Beziehung werden die folgenden Use Cases erfüllt:

- Erstellen neue Question durch Benutzer
- Verändern existierte Question durch ihr Creator
- Löschen existierte Question durch ihr Creator

b) Question-Playlist Beziehung

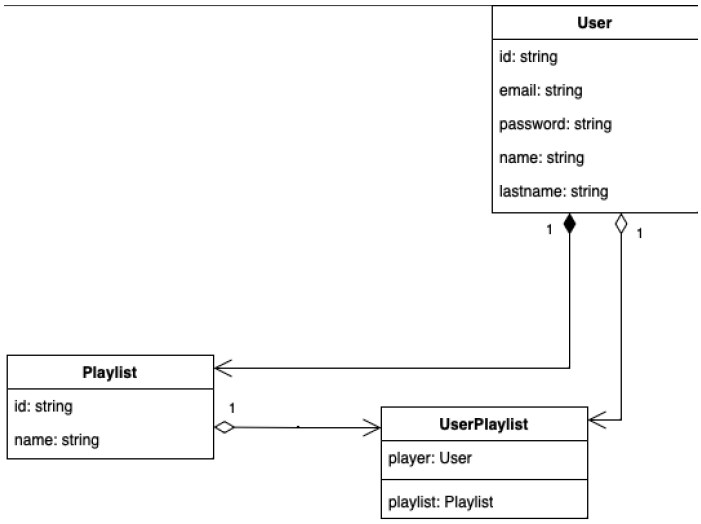


Ein Playlist kann viele Fragen enthalten und eine Frage kann zu vielen Playlists zuhören. Deshalb muss die Beziehung zwischen Playlist- und Question-Entity “Many to Many” sein. Wir möchten jedoch mehr Kontrolle über die Zwischentabelle haben, damit wir in Zukunft weitere Attribute hinzufügen können. Deshalb wird die Tabelle PlaylistQuestion als Zwischentabelle hinzugefügt. Die Beziehung zwischen PlaylistQuestion und Playlist/Question ist “One to Many”.

Durch diese Beziehung werden die folgenden Use Cases erfüllt:

- Erstellen neue Playlist sowie Question
- Verändern existierte Playlist sowie Question
- Löschen existierte Playlist sowie Question

c) User-Playlist Beziehung

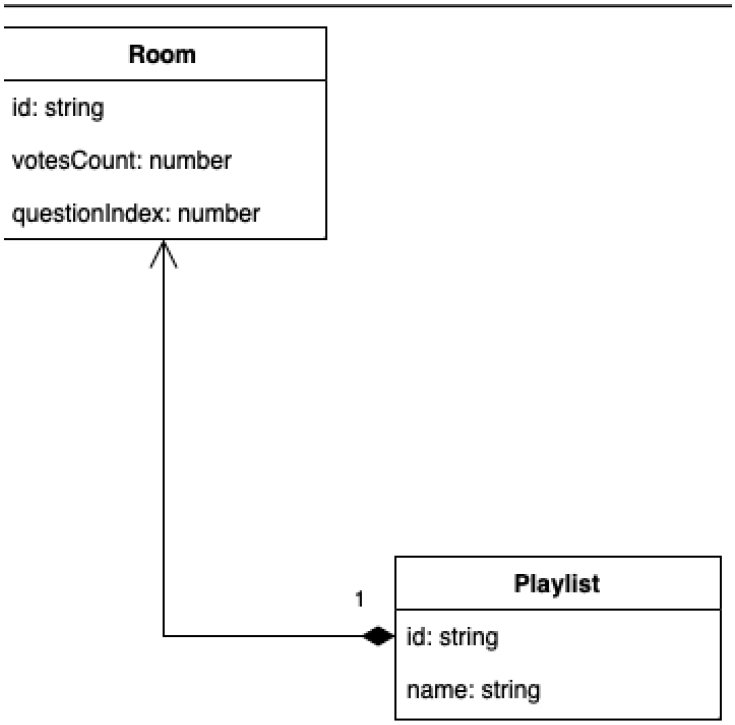


Diese Beziehung ist eine Kombination der beiden oben genannten Beziehungen. Die Composition Relation wird verwendet, um die Person zu speichern, die die Playlist erstellt hat. Die “One to Many”-Beziehung wird verwendet, um eine Liste von Playlists zu speichern, die einem Benutzer gefallen haben. Diese Beziehung zeigt auch, wer eine bestimmte Liste favorisiert hat

Durch diese Beziehung werden die folgenden Use Cases erfüllt:

- Erstellen neue Playlist durch Benutzer
- Verändern existierte Playlist durch ihr Creator
- Löschen existierte Playlist durch ihr Creator

d) Room-Playlist Beziehung

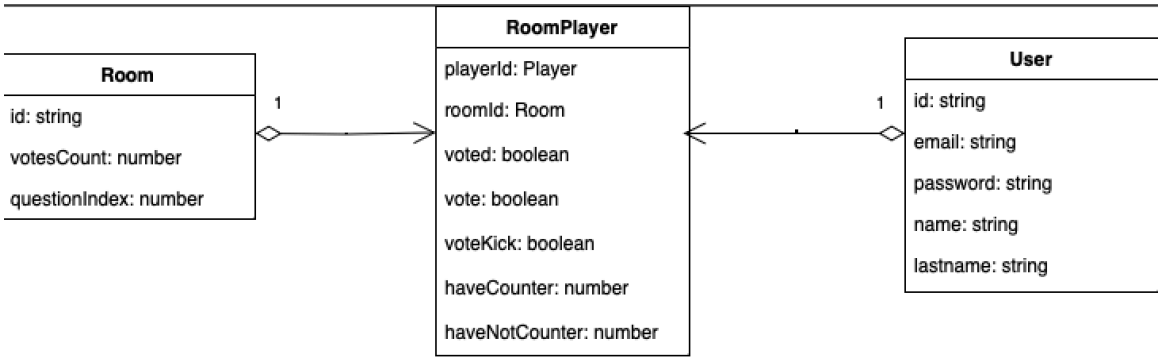


Ein Spielraum wird nur erstellt, wenn ein Playlist gewählt ist. Deshalb wird die Composition-Relation hier verwendet.

Durch diese Beziehung werden die folgenden Use Cases erfüllt:

- Erstellen den Room mit Default-Playlist

e) User-Room Beziehung

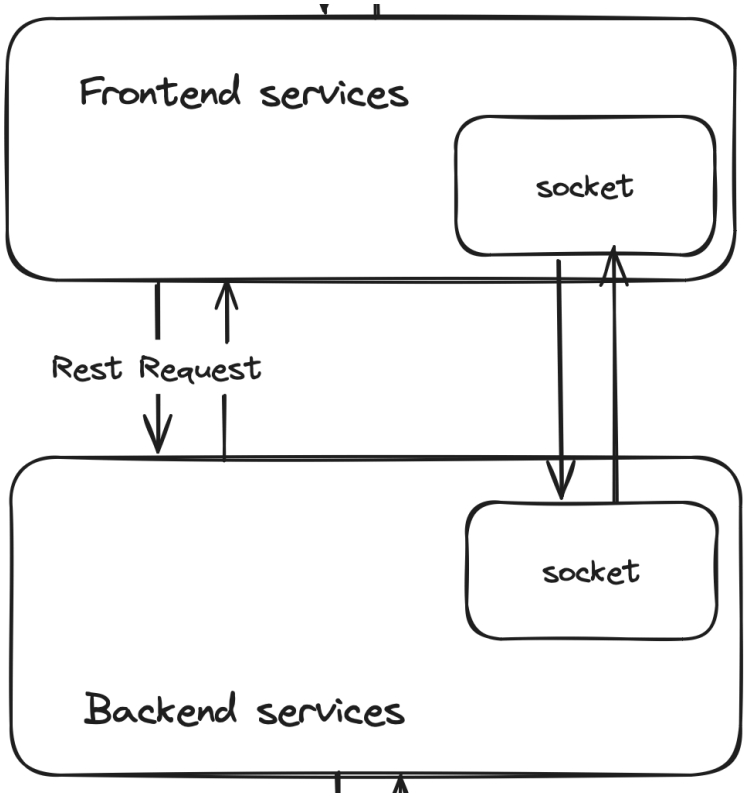


Ein oder mehrere Benutzer können in einem oder mehreren Spielräumen eintreten, um zu spielen. Deswegen muss die Beziehung zwischen User und Room als “Many-To-Many” sein. In dieser Situation erstellt man die aggregierte Beziehung “One-To-Many” zwischen Room/ User und RoomPlayer als Zwischentabelle.

Durch diese Beziehung werden die folgenden Use Cases erfüllt:

- Erhalten alle Spieler von einem Raum
- Schließen einen existierten Raum an
- Verlassen den Raum
- Stimmen die Spieler dafür/ dagegen im Raum
- Kicken ein oder mehrere Spielern von einem existierten Raum durch Mehrheit-Stimmen

3. REST-API Implementierung



Bei der Kommunikation zwischen Frontend und Backend erfolgt die Übertragung von REST-Anfragen über den Router zum entsprechenden Controller an der Backend-Seite. Der Router dient dazu, eingehende Anfragen an die richtigen Controller weiterzuleiten, basierend auf der angeforderten URL. Der Controller ist für die Verarbeitung dieser Anfragen zuständig und leitet sie entsprechend an die entsprechende Methode weiter, basierend auf den Parametern in der URL sowie der angeforderten Methode. Diese Methode im Controller führt dann die entsprechenden Aktionen aus, wie beispielsweise das Abrufen oder Ändern von Daten in der Datenbank, und sendet das Ergebnis zurück an das Frontend.

Auf unserer Website gibt es vier Haupteinheiten: Room, Playlist, Question, User. Daher wurden vier Controller implementiert.

```
// routes
app.use("/auth", AuthController);
app.use("/question", QuestionController);
app.use("/playlist", PlaylistController);
app.use("/room", RoomController);
```

IV. Backend Socket Server Implementierung

Wir brauchten einen Weg für das Backend, sich konstant mit dem Client zu kommunizieren, um das Spielen zu implementieren. Hier war eine restAPI nicht viel zu helfen, also brauchten wir ein andere weg das zu machen. Am Ende entscheidet sich das Team, mit Sockets und Events das Spielen zu implementieren und dafür haben wir die Library SocketIO genommen, da diese Library einfach mit Express integrierbar ist.

In dem Backend gibt es eine Socket.controller.ts Datei, die für das Handeln von Socket Verbindungen und Events zuständig ist. Der Socketserver wird dann in den server.ts Datei mit der Funktion setupSocketServer(io) hochgefahren.

Mit den Socketserver haben wir folgende Use Cases gelöst mittels die gegebenen Events:

1. Erstellen eines Spielraums

   Eventname: create-room
   Beschreibung: Dieses Ereignis wird ausgelöst, wenn ein Client die Erstellung eines neuen Raums anfordert. Es erwartet Daten mit der User-ID und der Playlist-ID. Der Server ruft den entsprechenden Benutzer und die Playlist aus der Datenbank ab, erstellt einen neuen Raum und fügt den Benutzer als Spieler hinzu. Bei Erfolg sendet er ein create-room-confirmation-Ereignis an den Client mit der neu erstellten Raum-ID und der nächsten Frage für den Raum. Wenn während des Vorgangs ein Fehler auftritt, sendet er ein error-Ereignis an den Client mit einem Fehlercode.

2. Beitreten einen existierenden Raum

   Eventname: join-room
   Beschreibung: Dieses Ereignis wird ausgelöst, wenn ein Client einem bestehenden Raum beitreten möchte. Es erwartet Daten mit der Raum-ID und der User-ID. Der Server ruft den entsprechenden Raum und Benutzer aus der Datenbank ab, erstellt einen neuen Raumspieler für den Benutzer und fügt ihn dem Raum hinzu. Bei Erfolg sendet er ein create-room-confirmation-Ereignis an den Client mit der Raum-ID und der aktuellen Frage für den Raum. Zusätzlich wird ein fetch-player-Ereignis an alle Clients im Raum gesendet, um die Spielerliste zu aktualisieren. Wenn ein Fehler auftritt, sendet der Server ein error-Ereignis an den Client mit einem entsprechenden Fehlercode.

3. Reload des Spiel-Seites soll Spieler wieder an den Raum hinzufügen.

   Eventname: reconnect-room
   Beschreibung: Dieses Ereignis wird ausgelöst, wenn ein Client die Seite aktualisiert und versucht, sich wieder mit einem Raum zu verbinden, den er zuvor betreten hat. Es erwartet Daten mit der Raum-ID. Der Server ruft den entsprechenden Raum aus der Datenbank ab und registriert den Socket des Clients im Raum, falls der Raum gefunden wurde. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

4. Ein Socket sollte von einem Socket-Raum entferntbar sein.

   Eventname: leave-room
   Beschreibung: Dieses Ereignis wird ausgelöst, wenn ein Spieler aus dem Spiel gekicked wird und den IO-Raum verlassen muss. Es erwartet Daten mit der Raum-ID. Der Server entfernt dann den Socket des Spielers aus dem angegebenen Raum.

5. Ein Spieler soll auf eine Frage wählen können.

   Eventname: vote
   Beschreibung: Dieses Ereignis wird ausgelöst, wenn ein Spieler sein Vote abgibt. Es erwartet Daten mit der Raum-ID, der Vote des Spielers und der User-ID des Spielers. Der Server ruft den entsprechenden Raum aus der Datenbank ab und überprüft, ob der Spieler bereits in diesem Raum ist. Wenn ja, wird die Stimme des Spielers registriert und die entsprechenden Zähler für die Statistik inkrementiert. Anschließend wird der Vote des Spielers in der Liste der Votes für den Raum registriert. Wenn alle Votes abgegeben wurden, wird ein "voting-finished"-Ereignis an alle Clients im Raum gesendet, um anzuzeigen, dass die Abstimmung abgeschlossen ist. Der Zähler für der Votes wird auf 0

gesetzt, um sich auf die nächste Frage vorzubereiten. Schließlich wird ein "fetch-player"-Ereignis an alle Clients im Raum gesendet, um die Spielerliste zu aktualisieren. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

6. Ein Spieler soll die nächste Frage bekommen können.

   Eventname: next-question
   Beschreibung: Dieses Ereignis wird ausgelöst, um die nächste Frage abzurufen und an die Clients zu senden. Es erwartet Daten mit der Raum-ID. Der Server ruft den entsprechenden Raum aus der Datenbank ab und extrahiert die ID der Playlist. Dann werden alle Spieler im Raum abgerufen und ihre Vote zurückgesetzt. Anschließend wird die nächste Frage für den Raum abgerufen und an alle Clients im Raum gesendet. Schließlich wird ein "fetch-player"-Ereignis an alle Clients im Raum gesendet, um die Spielerliste zu aktualisieren. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

7. Ein Spieler soll die aktuelle Frage bekommen können.

   Eventname: get-question
   Beschreibung: Dieses Ereignis wird ausgelöst, um die aktuelle Frage abzurufen und an den Client zu senden. Es erwartet Daten mit der Raum-ID. Der Server ruft den entsprechenden Raum aus der Datenbank ab und holt die aktuelle Frage für den Raum. Anschließend wird die Frage an den Client zurückgegeben über das bereitgestellte Callback. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

8. Spieler in einem Raum sollten einen anderen Spieler kicken können.

   Eventname: vote-kick
   Beschreibung: Dieses Ereignis wird ausgelöst, um die Abstimmung zum Kicken eines Spielers zu registrieren und zu verarbeiten. Es erwartet Daten mit der Spieler-ID, der Raum-ID und der Gesamtzahl der Spieler im Raum. Der Server ruft den entsprechenden Raum und den entsprechenden Spieler aus der Datenbank ab. Dann wird die Anzahl der Rauswurf-Stimmen des Spielers erhöht und in der Datenbank gespeichert. Wenn die Mehrheit der Spieler für den Rauswurf stimmt (mehr als die Hälfte der Spieler), wird der Spieler aus dem Raum entfernt und ein "kick-player"-Ereignis wird an alle Clients im Raum gesendet, um den ausgewählten Spieler zu informieren, dass er gekickt wurde. Schließlich wird ein "fetch-player"-Ereignis an alle Clients im Raum gesendet, um die

Spielerliste zu aktualisieren. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

9. Client sollte einen Raum löschen können.

   Eventname: destroy-game
   Beschreibung: Dieses Ereignis wird ausgelöst, um das Spiel am Ende des Spiels zu beenden und alle damit verbundenen Daten zu löschen. Es erwartet Daten mit der Raum-ID. Der Server ruft den entsprechenden Raum und alle Raumspieler aus der Datenbank ab. Dann werden alle Raumspieler und der Raum selbst aus der Datenbank entfernt. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

10. Spielern sollten mittels einem Chat sich kommunizieren können.

    Eventname: chat message
    Beschreibung: Dieses Ereignis wird ausgelöst, um die Chat-Funktionalität zu behandeln. Es erwartet Daten mit der Benutzer-ID und der Nachricht. Der Server ruft den entsprechenden Benutzer aus der Datenbank ab und extrahiert den Benutzernamen. Dann wird die vollständige Nachricht erstellt, die den Benutzernamen und die Nachricht enthält. Schließlich wird die Nachricht an alle Clients im Chatraum gesendet. Wenn ein Fehler auftritt, sendet der Server ein "error"-Ereignis an den Client mit einem entsprechenden Fehlercode.

V. Frontend Socket Events

Ein SocketProvider ist implementiert um zugriff auf die Socket funktionalitäten mittels den useSocket() hook zu haben. Das Frontend reagiert auch auf Events, die der Server an ihn schickt und auch Events zum Server schicken, um folgende Use Cases zu lösen.

1. Ein Spieler soll die nächste Frage bekommen können und diese darstellen.

   Eventname: next-question
   Beschreibung: Dieses Ereignis wird empfangen, wenn der Server die nächste Frage für das Spiel sendet. Die Frage wird aus den empfangenen Daten extrahiert. Anschließend wird die Methode setVoted(false) aufgerufen, um den Zustand "voted" zurückzusetzen, falls dies auf der Client-Seite erforderlich ist. Schließlich wird die Methode setQuestion(data.question) aufgerufen, um den Zustand der aktuellen Frage zu aktualisieren und die neue Frage anzuzeigen.

2. Den frontend muss immer die Aktualisierte liste von Spieler haben.

   Eventname: fetch-player
   Beschreibung: Beim Empfang dieses Events, sende dem Client einen HTTP-Request an die API, um die Liste von Spielern im Raum abzurufen. Der Server sendet diesen Event immer dann, wenn es eine Aktualisierung in dieser Liste gibt.

3. Spielern können andere Spieler aus dem Raum kicken.

   Eventname: kick-player
   Beschreibung:Dieses Ereignis wird empfangen, wenn ein Spieler aus dem Raum entfernt werden soll. Die Daten enthalten die ID des Spielers, der aus dem Raum entfernt werden soll (playerToKick). Wenn die ID des Spielers mit der des aktuellen Benutzers übereinstimmt, sendet das Client-Socket ein Event mit dem Namen "leave-room" an den Server und übermittelt die Raum-ID. Anschließend navigiert der Benutzer zur Startseite ("/home").

4. Spielern können erst nachdem alle anderen gewählt haben, eine neue Frage bekommen.

   Eventname: voting-finished
   Beschreibung:Dieses Ereignis wird empfangen, wenn die Abstimmung für die aktuelle Frage abgeschlossen ist. Nach Erhalt dieses Ereignisses wird die Methode der State NextQuestionAvailable auf true gesetst, um anzugeben, dass die nächste Frage verfügbar ist.

5. Nachdem der Spiel zum ende kommt, soll der Server den Raum löschen.
   Eventname: destroy-game
   Beschreibung: Dieses Ereignis wird vom Client-Socket ausgelöst, um das aktuelle Spiel und den zugehörigen Raum zu beenden. Es sendet die Raum-ID an den Server, um das Spiel zu zerstören. Nach dem Senden dieses Events wird die Methode onClose() aufgerufen, um die Modalfenster zu schließen. Schließlich navigiert der Benutzer zur Startseite ("/home").

6. Ein Spieler kann einen Raum erstellen.

   Eventname: create-room
   Beschreibung: Dieses Ereignis wird vom Client-Socket ausgelöst, um einen neuen Raum zu erstellen. Es sendet die Benutzer-ID (userId) und die ausgewählte Wiedergabelisten-ID (playlistId) an den Server, um den Raum zu erstellen.
7. Ein Spieler kann in einen existierenden Raum beitreten.

   Eventname: join-room
   Beschreibung: Dieses Ereignis wird vom Client-Socket ausgelöst, um einem bestehenden Raum beizutreten. Es sendet die Raum-ID (roomId) und die Benutzer-ID (userId) an den Server, um dem Raum beizutreten.

8. Um Use Case 6 und 7 zu realisieren, muss der Client auf das unten benannte Event reagieren.

   Eventname: create-room-confirmation
   Beschreibung: Dieses Ereignis wird empfangen, wenn die Erstellung eines neuen Raums vom Server bestätigt wird. Die empfangenen Daten enthalten die ID des erstellten Raums (data.roomId). Nach Erhalt dieses Ereignisses wird die Methode setRoomId() aufgerufen, um die Raum-ID zu setzen. Außerdem wird eine Erfolgsmeldung angezeigt, indem toast.success("Room created successfully") aufgerufen wird. Schließlich navigiert der Benutzer zur Spielseite des neu erstellten Raums (navigate(/game/${data.roomId})).

VI. Frontend Pages

Der `pages`-Ordner in unserem Frontend-Projekt ist strukturiert, um verschiedene Seiten und unsere zugehörigen Komponenten zu organisieren. Hier ist eine Zusammenfassung der Inhalte:

Seiten und ihre Komponenten:

- Play:
  - `GamePage.tsx`: Die Hauptseite für das Spiel ist der Ort, wo die Spiellogik und das UI zusammenkommen.
  - `components/`: Enthält spezifische Komponenten für die Spiel-Seite, wie `ChatRoom.tsx` für einen Chat-Bereich, `GameRoom.tsx` für die Spielumgebung und `ShowGame.tsx` für die Anzeige des Spiels.
- Authentifizierung (`auth`):
  - `LoginPage.tsx` und `RegisterPage.tsx`: Seiten für die Benutzeranmeldung und -registrierung.
  - `components/AuthCard.tsx`: Eine Komponente, die in den Authentifizierungsseiten verwendet wird, für Formulare oder Informationskarten.
- User:
  - `UserOverviewPage.tsx`: Auf dieser Seite gibt es zwei Sections
    - User Info Section: Der Benutzer kann hier persönliche Informationen einsehen und bearbeiten
    - Playlist Section: Der Benutzer kann hier alle von ihm erstellten Playlists sehen sowie neue Playlists erstellen oder bestehende Playlists löschen
  - `UserEditPage.tsx`: Der Benutzer kann hier seine/ihre Informationen (Name, Email, Passwort) bearbeiten
  - `components/ViewPlaylistModal`: Der Benutzer kann alle relevanten Informationen der ausgewählten Playlists wie Name und Fragen sehen
- Playlist
  - `CreatePlaylistPage.tsx`: Auf dieser Seite kann der Benutzer so viele vorhandene Fragen wie möglich aus der Datenbank hinzufügen oder eine neue Frage erstellen.
  - `EditPlaylistPage.tsx`: Auf dieser Seite kann der Benutzer Playlist-Daten bearbeiten, z. B. den Playlist-Namen ändern oder Fragen löschen
  - `components/EditQuestionModal.tsx`: Auf diesem Modal kann der Benutzer Fragendaten bearbeiten
  - `components/QuestionModal.tsx`: Auf diesem Modal kann der Benutzer neue Frage erstellen

        - `components/SelectExistingQuestionModal.tsx`: Auf diesem Modal kann der Benutzer eine Frage aus der vorhandenen Frageliste aus der DB
- Home:
    - `Home.tsx` : Es ist , als gäbe es zwei Komponenten für die Startseite, was auf verschiedene Layouts oder Zustände der Startseite hinweisen könnte.
    - `components/JoinRoom.tsx`: Eine Komponente, die eine Funktion bietet, um einem Spiel beizutreten.

Struktur: Die Struktur des pages-Ordners deutet darauf hin, dass das Projekt eine klare Trennung zwischen verschiedenen Funktionsbereichen der Anwendung vornimmt, mit spezifischen Seiten und zugehörigen Komponenten für jede Funktionalität. Dies ist eine gängige Praxis in modernen Webanwendungen, um die Wartbarkeit und Skalierbarkeit des Codes zu verbessern.