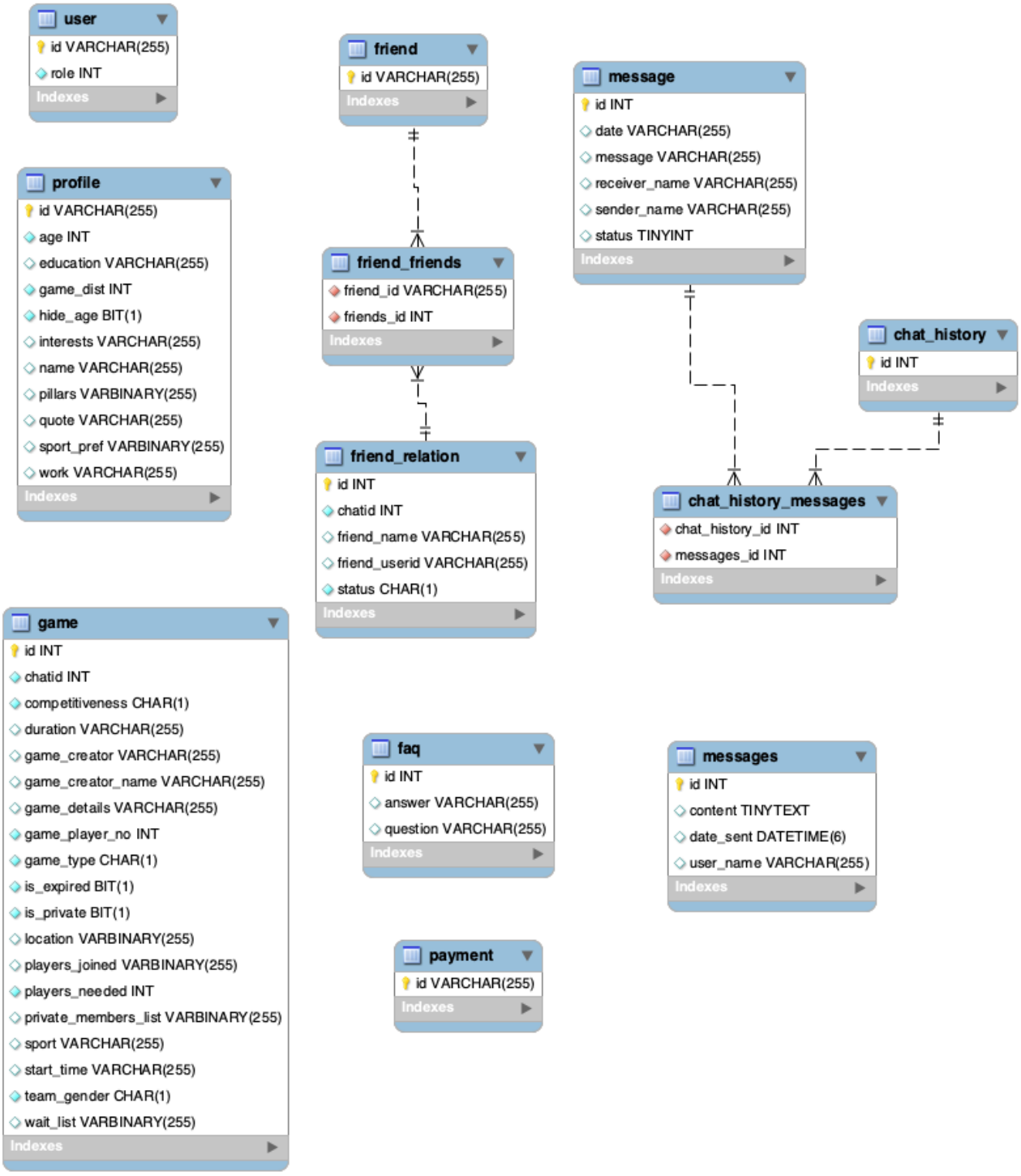
Technologies Used

Backend - SpringBoot, Hibernate , JAVA, Maven
Database - MySQL

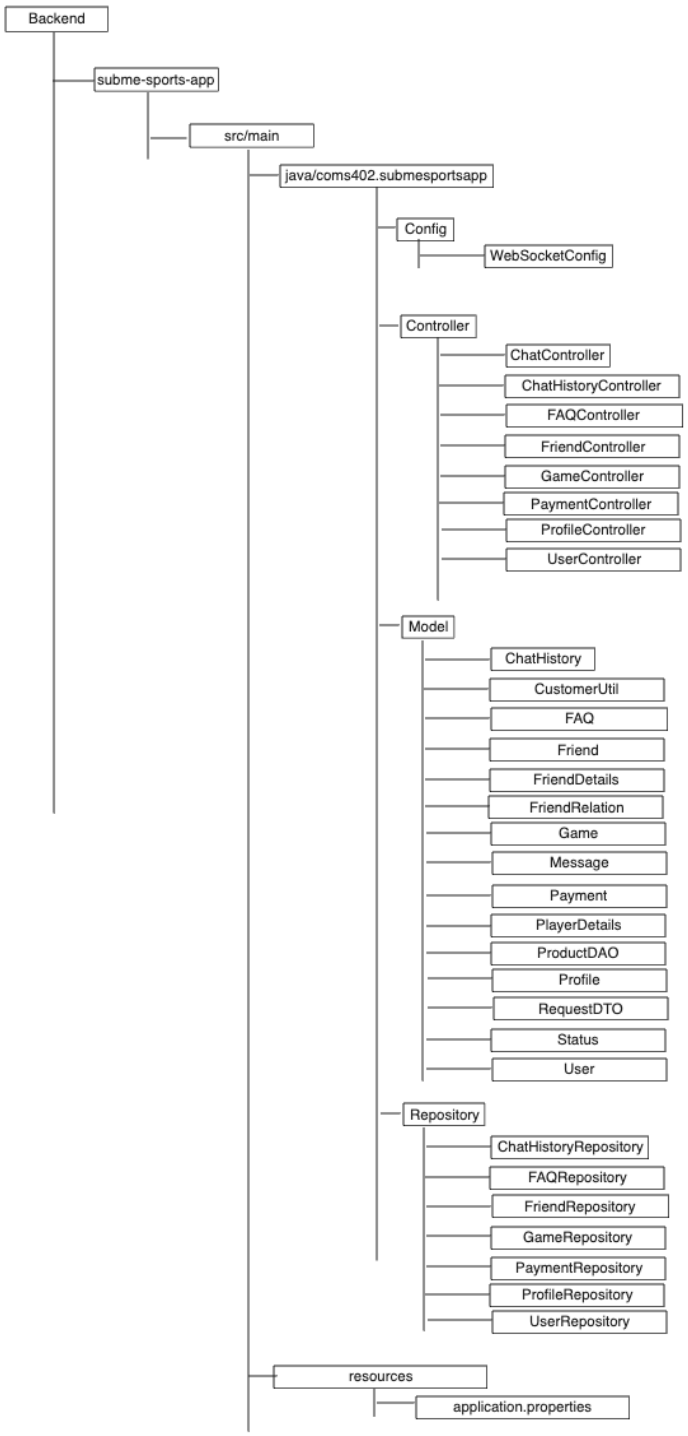
Database Schematic Diagram



Project Structure

The folder structure for the backend is structured in 4 main folders that ties to the general functionality of the components encapsulated within the folders.

Controller - Handles incoming HTTP requests and handles business logic for services
Model - Contains classes that represent the data model or business entities used in the application
Repository - Contains classes that are responsible for data access and persistence
Config - Contains configurations for websocket



Setup, Installation and Running the application

1. Once the repository for the frontend is downloaded onto the developer's machine, open the project via Intellij IDE (or your preferred choice of IDE)
    a. In the case of using Intellij, the maven and other necessary packages will be installed automatically
2. Create user “subme” in MySQL workbench and create a schema named “submedb” with “password” as its password

   (OPTIONAL) Navigate to the applications.properties folder (Refer to folder structure diagram for more details) to change the datasource.url to connect to other schemas or change the destination of the database connection address

3. Run the application. Instructions to run frontend application can be found in the frontend documentation docs.

Endpoints

Websocket

ChatController (Websocket endpoint, NOT a part of RESTful API)

Endpoint	Description
/group-message	Used for group chats, receives chatID as header to direct to private groups
/private-message	Used for private 1-1 chatting

Core Functionalities

ChatHistoryController

Endpoint	Verb	Description
/chatHistory/{id}	GET	Returns ChatHistory Object

FAQController

Endpoint	Verb	Description
/faqs	GET	Gets all FAQS
/faqs	POST	Creates an FAQ based on given FAQ object
/faqs{id}	GET	Retrieves an FAQ based on id
/faqs	PUT	Updates FAQ object with given FAQ object
/faqs/{id}	DELETE	Deletes an FAQ based on given id

FriendController

Endpoint	Verb	Description
/friends/{currUserID}	GET	Returns a list of Friends based on given currUserID
/friends/requests/{curUserID}	GET	Returns a list of friend requests based on given currUserID
/friends/checkFriendsStatus/{currUserID}/{targetUserID}	GET	Used to check if targetUser is a friend of currUser, returns FrienDetails with status = “N” if no relationship exists
/friends/{currUserID}	POST	Used for creating Friend Entry in database (mainly used for testing)
/friends/{currUserID}/{targetUserID}	POST	Used for creating friend requests from currUser to targetUser
/friends/{currUserID}/{targetUserID}	PUT	Confirms friend request of currUser
/friends/{currUserID}	PUT	Updates Friend Relationship of curr User based on given FriendRelation object
/friends/{currUserID}/{targetUserID}	DELETE	Rejects friend request, or remove existing friend relationship between 2 users

GameController

Endpoint	Verb	Description
/games/{userID}	GET	Returns a list of games the current user has access to **NOTE** : Games that the user has not joined
/games/active/{userID}	GET	Returns a list of games the current user has joined
/games/past/{userID}	GET	Returns a list of games the current user has played
/games/{id}	GET	Returns a game based on game id
/game/isJoined/{id}/{userID}	GET	Returns true if current user joined a game with game id = id, false otherwise
/game/isInWaitlist/{id}/{userID}	GET	Returns true if current user is in waitlist of a game with game id = id, false otherwise
/game/playersJoined/{id}	GET	Returns details of users who have joined a game
/game/playersInWaitlist/{id}	GET	Returns details of users who have joined the waitlist of a game
/game	POST	Creates a game based on given game object
/game/{id}	PUT	Updates a current game with game id = id with given game object
/game/{id}	DELETE	Deletes a game with game id = id

PaymentController
Only partially implemented

ProfileController
NOTE : id in ProfileController is not auto incremented, uses String ID generated by Clerk from the front end.

Endpoint	Verb	Description
/profile	GET	Returns the profile of all users
/profile/{id}	GET	Returns profile of user based on given id
/profile	POST	Creates a profile with given profile object
/profile/{id}	PUT	Updates a profile based on given id and profile object
/profile/{id}	DELETE	Deletes a profile based on given id

UserController
NOTE : id in UserController is not auto incremented, uses String ID generated by Clerk from the front end.

Endpoint	Verb	Description
/user	GET	Returns all users
/user/{id}	GET	Returns user based on given id
/user	POST	Creates a user with given user object NOTE : it also creates a friend object so that the user can use this friend object to add friends later
/user/{id}	PUT	Updates a user based on given id and user object
/user/{id}	DELETE	Deletes a user based on given id
/userExist/{id}	GET	Checks if a user exists based on given id, returns true if user with id exists, false otherwise