DuelHub

Task 11. Data design

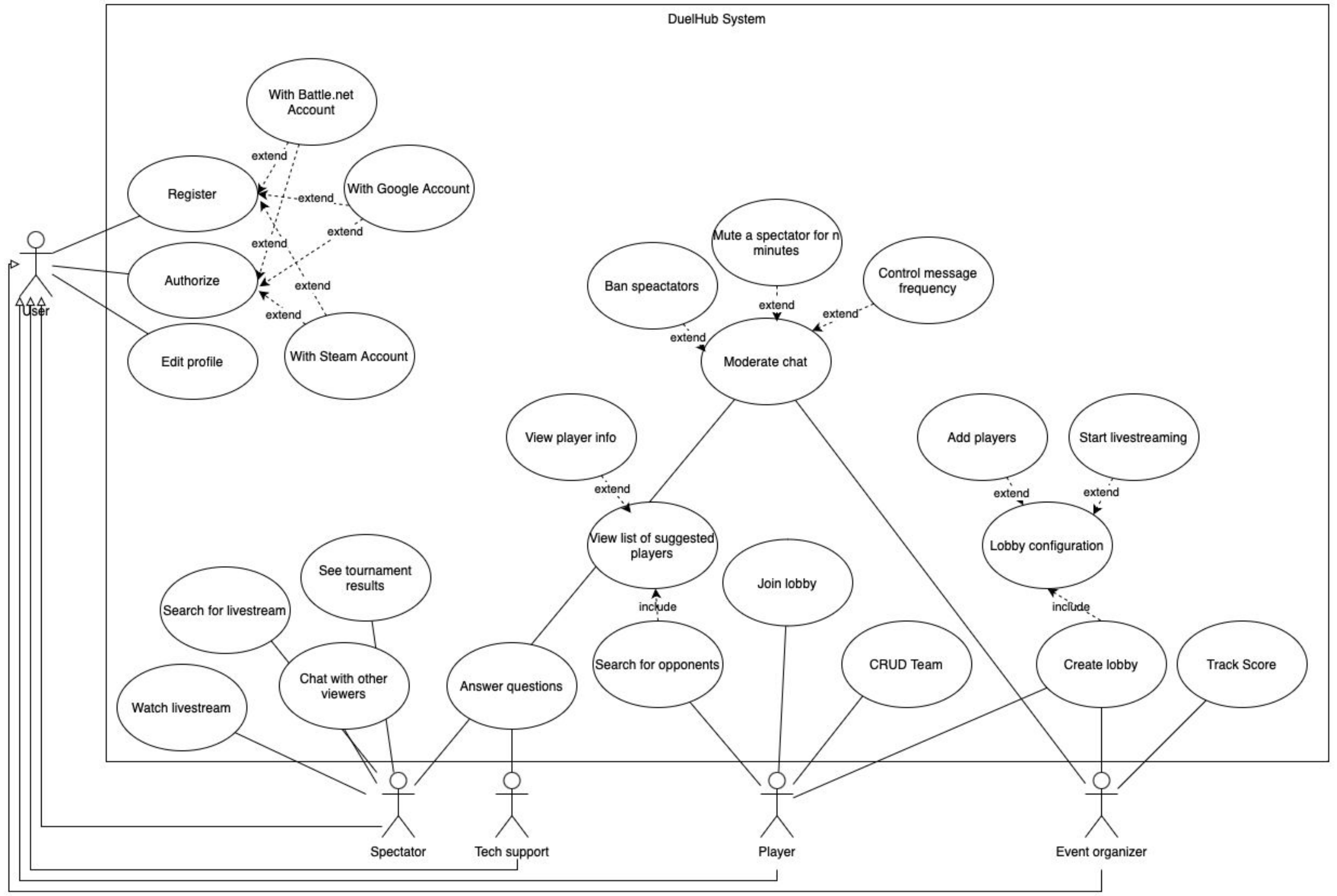
Product description

DuelHub is a web application, that provides a convenient online space for gamers to coordinate and engage in competitive matches across a wide range of computer games. By facilitating the organization of duels, our product ensures that players can easily connect with opponents and enjoy thrilling gaming experiences.

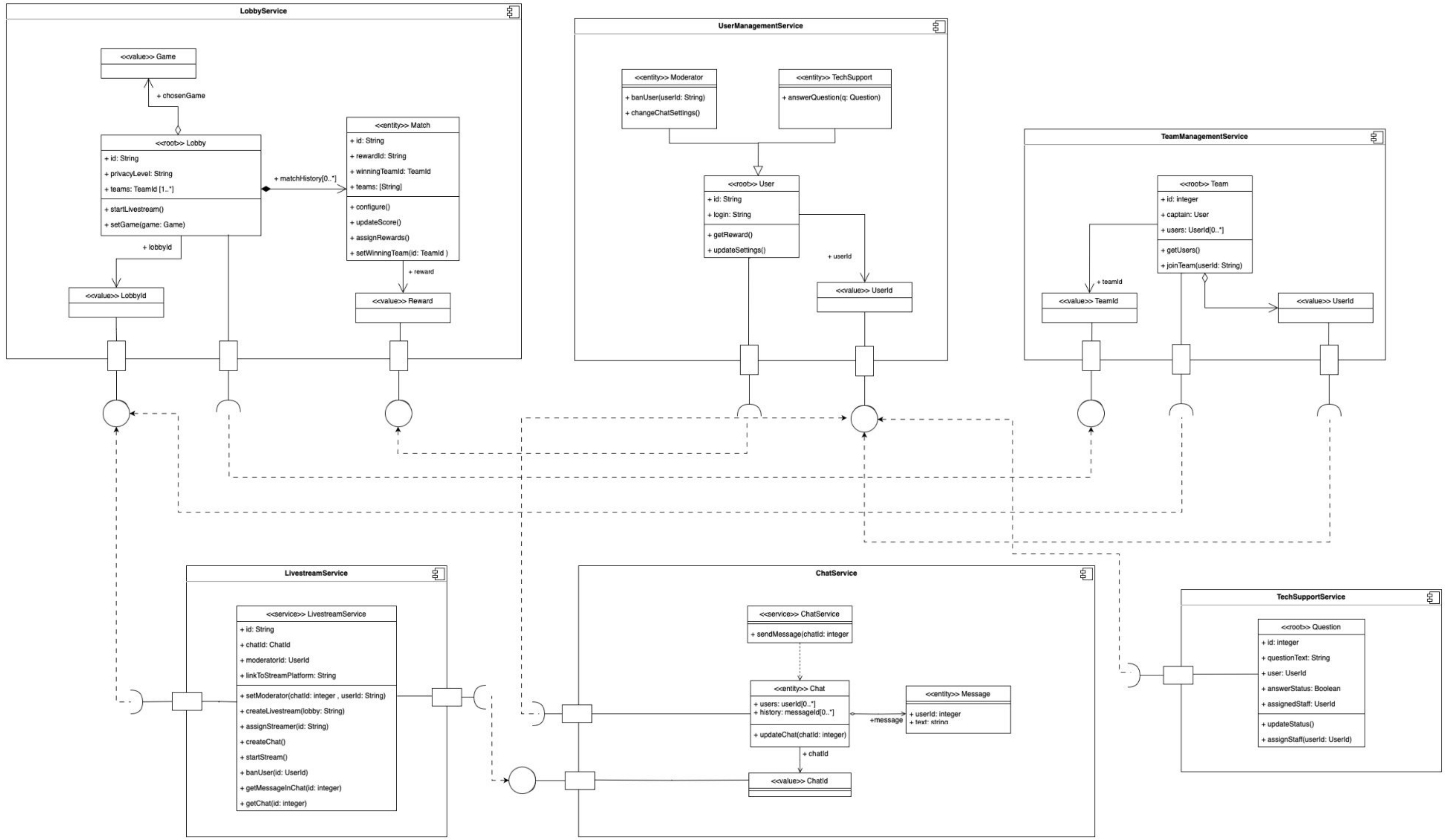
Team: Lyudmila Rezunik, Teona Sadulaeva

Repo: https://github.com/teopalmer/duelhub

Use case diagram



Service diagram



Service diagram

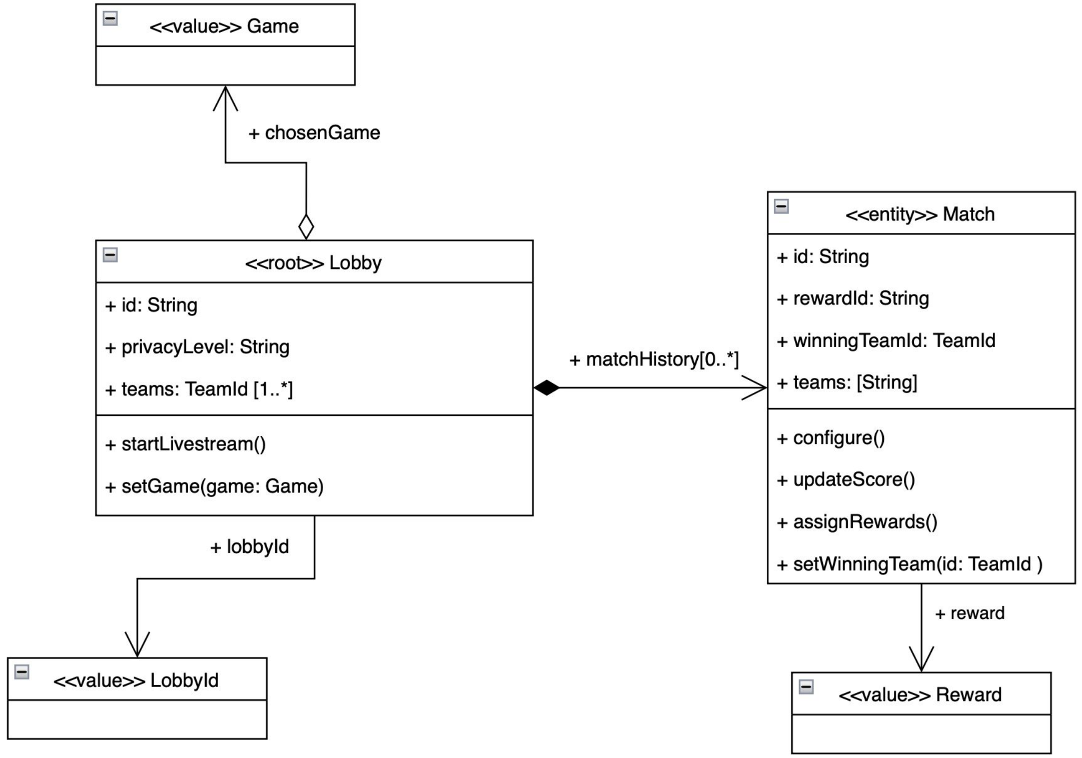
Our diagrams in full resolution are available here:

https://drive.google.com/file/d/12iX-DQJiogurJVrrsGgnssINR9I7YS4X/view?usp=sharing

Lobby Service

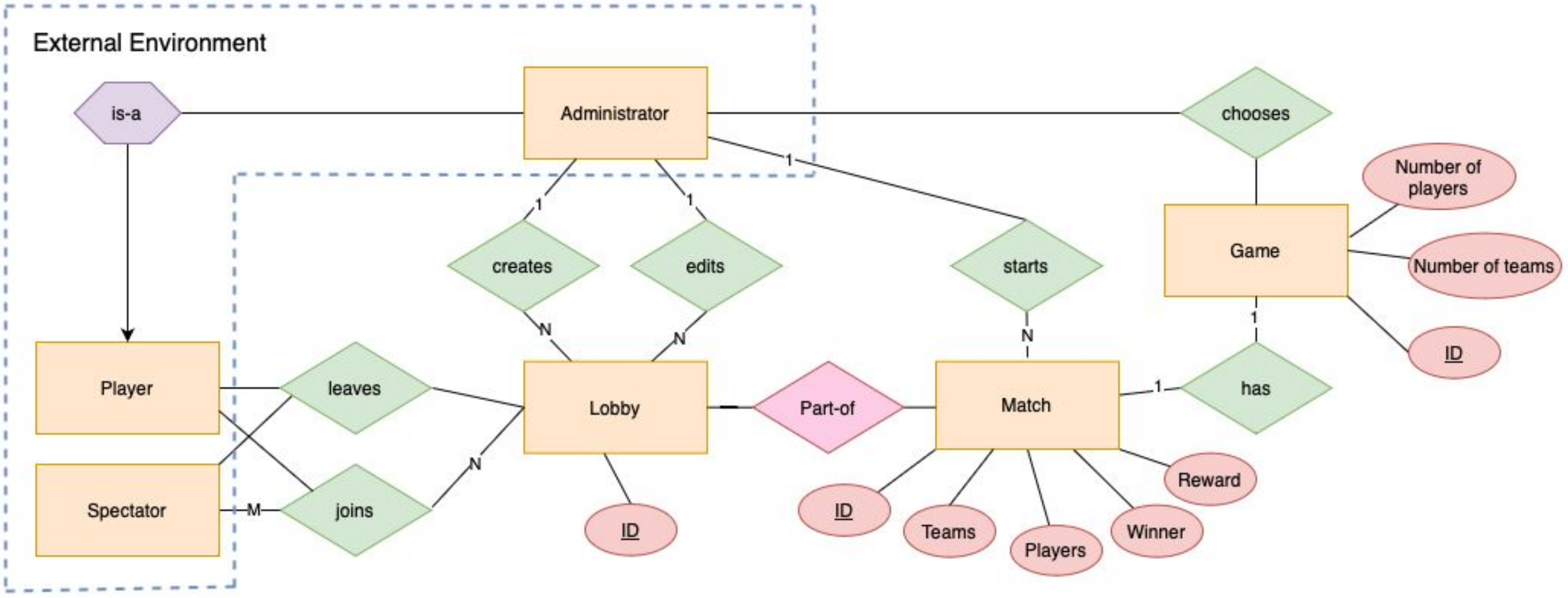
Lobby Service

Logical data model. Class Diagram



Lobby Service

Logical data model. ER Diagram



There are entities that are considered as “External Environment” (that interact with the service)

Lobby Service

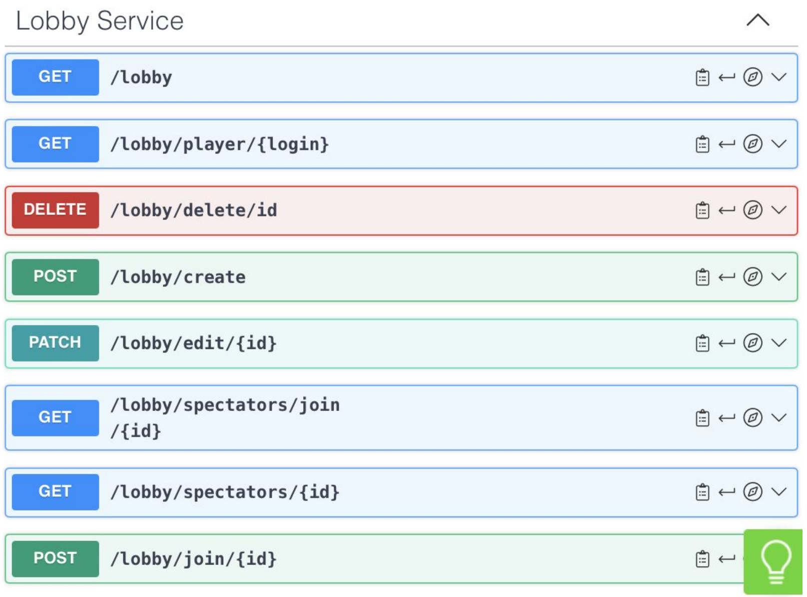
Logical data model. Event Flow

Lobby Service

API Summary

Link to the API:
https://app.swaggerhub.com/apis/LREZUNIK/DuelHub-Backend/1.0.0

Lobby Service API provides endpoints for creation of lobbies (virtual rooms for grouping players). Lobbies can be created by players. The player who created the lobby becomes its administrator. Different users can join the lobby as a player (if there are not enough players to start the game) or as spectators.
The lobby can be edited by the admin. He can choose the game to play, set its difficulty and number of players or teams if such settings are provided by the game.
If everything meets the requirements, administrator can start a match in lobby. While players play the game, the score of each team or player is updated. When the match finishes the winning team/player, players that participated and their score are saved and can be viewed in matches history of the player.
For each match there is a fixed reward that is assigned to the winners.
Lobby can be destroyed if everyone leaves it, or the admin decides to close it. The the lobby is deleted.



Lobby Service

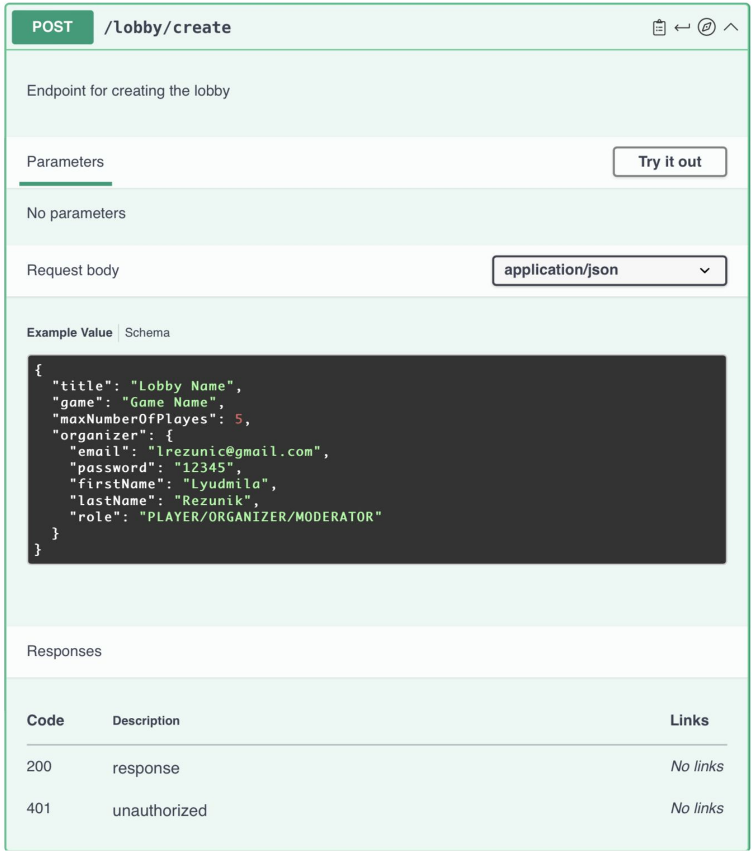
GET	/lobby
GET	/lobby/player/{login}
DELETE	/lobby/delete/id
POST	/lobby/create
PATCH	/lobby/edit/{id}
GET	/lobby/spectators/join/{id}
GET	/lobby/spectators/{id}
POST	/lobby/join/{id}

API usage <LobbyService>

Scenario: Create Lobby

Steps:

Player/Event organizer is on the main web page and chooses option “Create lobby” ->
The service is invoked ->
The service returns the status of the request to the client



Endpoint for creating the lobby

No parameters

Request body: application/json

```
{
  "title": "Lobby Name",
  "game": "Game Name",
  "maxNumberOfPlayes": 5,
  "organizer": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  }
}
```

Responses

Code	Description	Links
200	response	*No links*
401	unauthorized	*No links*

API usage <LobbyService>

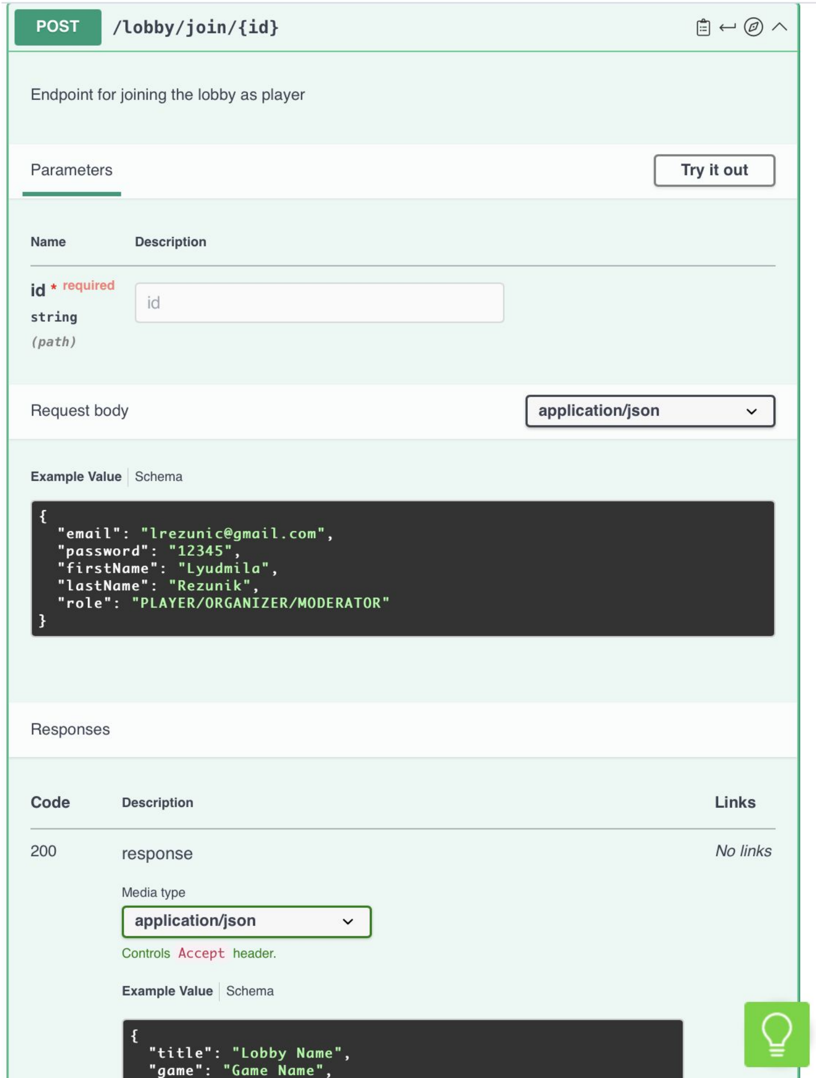
Scenario: Join Lobby

Steps:

Player is on the “Available lobbies”/notifications page and chooses “Join” option ->

The service is invoked ->

The service returns the lobby data to the client



POST /lobby/join/{id}

Endpoint for joining the lobby as player

Name	Description
id * required string (path)	id

Request body: application/json

```
{
  "email": "lrezunic@gmail.com",
  "password": "12345",
  "firstName": "Lyudmila",
  "lastName": "Rezunik",
  "role": "PLAYER/ORGANIZER/MODERATOR"
}
```

Responses

Code	Description	Links
200	response	No links

Media type: application/json

Controls `Accept` header.

```
{
  "title": "Lobby Name",
  "game": "Game Name",
```

API usage <LobbyService>

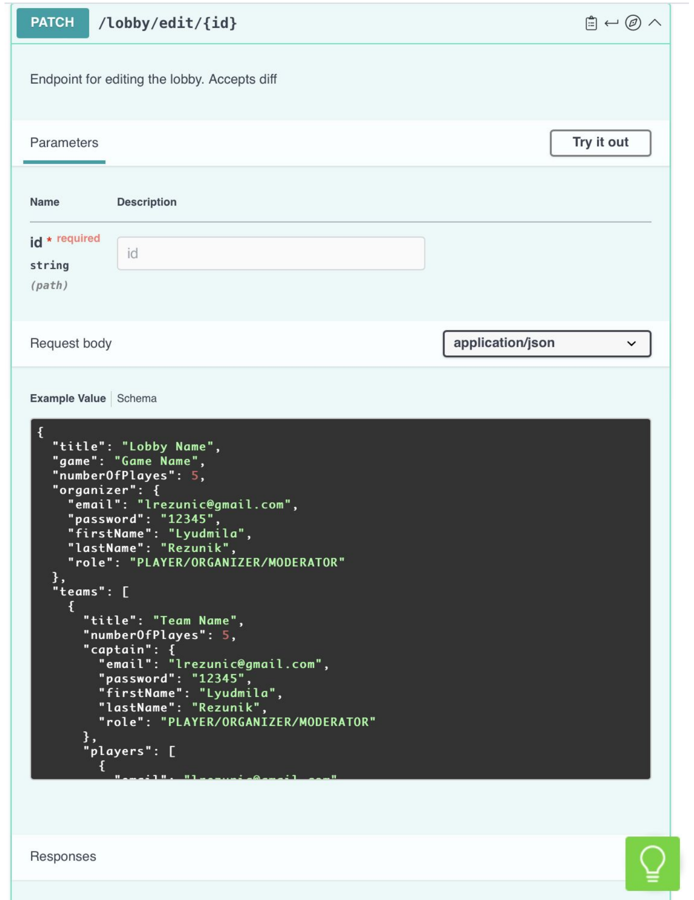
Scenario: Configure Lobby

Steps:

Player/Event organizer is on the lobby screen and chooses "Configure" option -> (the request is sent with a lobby id)

The service is invoked ->

The service returns the status of the request to the client



PATCH /lobby/edit/{id}

Endpoint for editing the lobby. Accepts diff

Parameters | Try it out

Name	Description
id * required string (path)	id

Request body: application/json

Example Value | Schema

```
{
  "title": "Lobby Name",
  "game": "Game Name",
  "numberOfPlayes": 5,
  "organizer": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "teams": [
    {
      "title": "Team Name",
      "numberOfPlayes": 5,
      "captain": {
        "email": "lrezunic@gmail.com",
        "password": "12345",
        "firstName": "Lyudmila",
        "lastName": "Rezunik",
        "role": "PLAYER/ORGANIZER/MODERATOR"
      },
      "players": [
        {
```

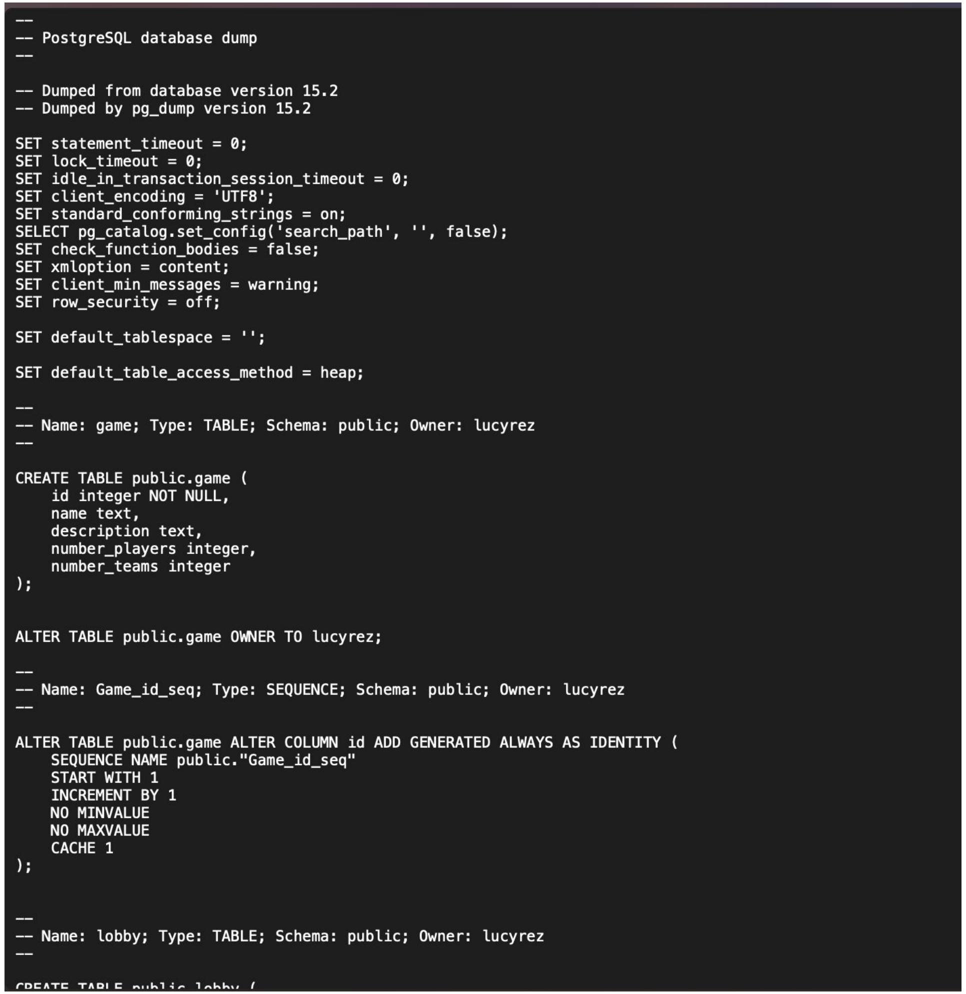
Responses

Lobby Service

Physical Schema

An SQL dump was performed. We got the SQL file containing all needed steps to create and fill the database.

The dump file can be found here: https://drive.google.com/file/d/10g60QWyst-pEcSAWx93LJ66VbVh7yJgx/view?usp=sharing

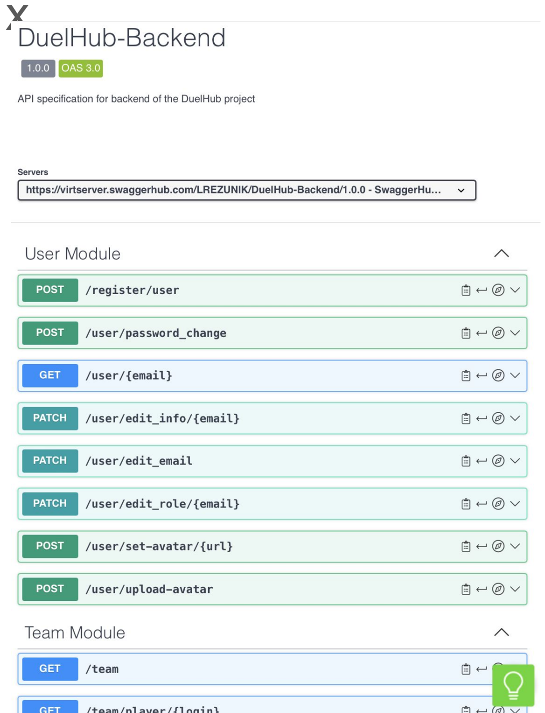


```
--
-- PostgreSQL database dump
--

-- Dumped from database version 15.2
-- Dumped by pg_dump version 15.2

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: game; Type: TABLE; Schema: public; Owner: lucyrez
--

CREATE TABLE public.game (
    id integer NOT NULL,
    name text,
    description text,
    number_players integer,
    number_teams integer
);


ALTER TABLE public.game OWNER TO lucyrez;

--
-- Name: Game_id_seq; Type: SEQUENCE; Schema: public; Owner: lucyrez
--

ALTER TABLE public.game ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
    SEQUENCE NAME public."Game_id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- Name: lobby; Type: TABLE; Schema: public; Owner: lucyrez
--

CREATE TABLE public.lobby (
```

Chat Service

Open API

Link to the API: https://app.swaggerhub.com/apis/LREZUNIK/DuelHub-Backend/1.0.0



DuelHub-Backend

1.0.0 OAS 3.0

API specification for backend of the DuelHub project

Servers

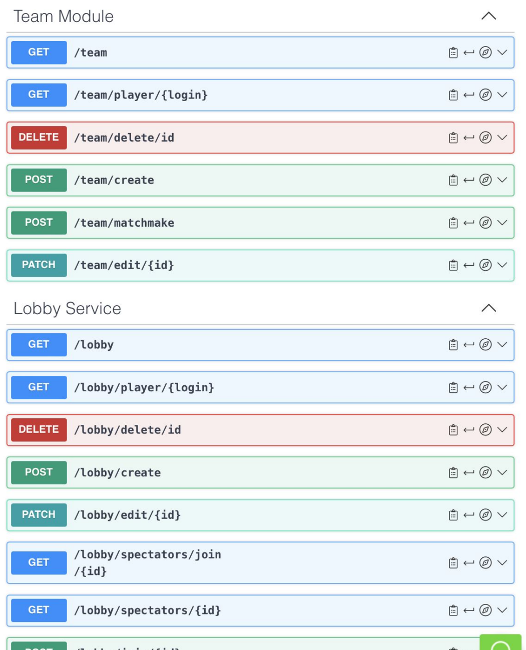
https://virtserver.swaggerhub.com/LREZUNIK/DuelHub-Backend/1.0.0 - SwaggerHu...

User Module

POST	/register/user
POST	/user/password_change
GET	/user/{email}
PATCH	/user/edit_info/{email}
PATCH	/user/edit_email
PATCH	/user/edit_role/{email}
POST	/user/set-avatar/{url}
POST	/user/upload-avatar

Team Module

GET	/team
GET	/team/player/{login}

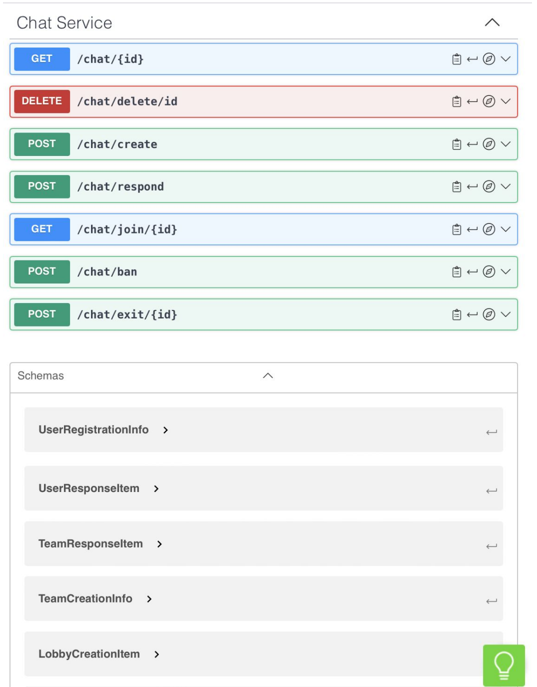


Team Module

GET	/team
GET	/team/player/{login}
DELETE	/team/delete/id
POST	/team/create
POST	/team/matchmake
PATCH	/team/edit/{id}

Lobby Service

GET	/lobby
GET	/lobby/player/{login}
DELETE	/lobby/delete/id
POST	/lobby/create
PATCH	/lobby/edit/{id}
GET	/lobby/spectators/join /{id}
GET	/lobby/spectators/{id}



Chat Service

GET	/chat/{id}
DELETE	/chat/delete/id
POST	/chat/create
POST	/chat/respond
GET	/chat/join/{id}
POST	/chat/ban
POST	/chat/exit/{id}

Schemas

UserRegistrationInfo

UserResponseItem

TeamResponseItem

TeamCreationInfo

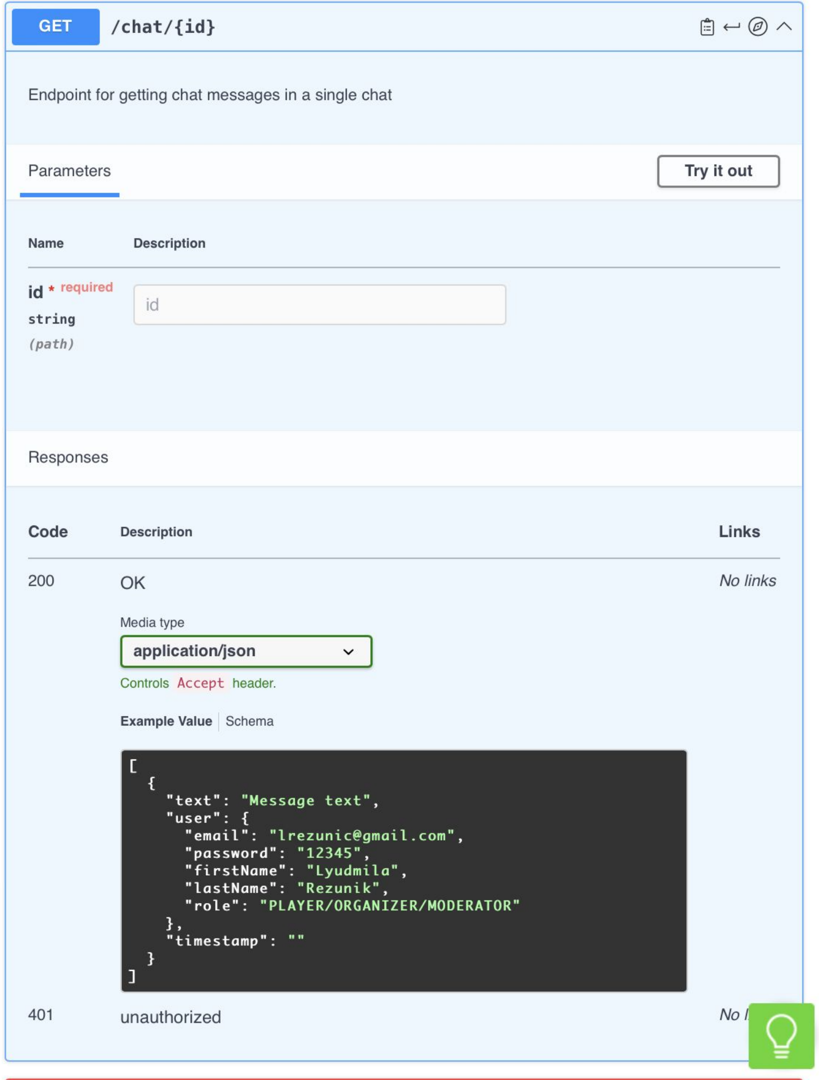
LobbyCreationItem

API usage <ChatService>

Scenario: Chat with viewers

Steps:

User selects the livestream to watch ->
Page with the livestream starts loading ->
The service is invoked ->
The service returns all the messages in chat



GET /chat/{id}

Endpoint for getting chat messages in a single chat

Parameters

Try it out

Name	Description
id * required string *(path)*	id

Responses

Code	Description	Links
200	OK	*No links*
401	unauthorized	*No l*

Media type: application/json

Controls Accept header.

Example Value | Schema

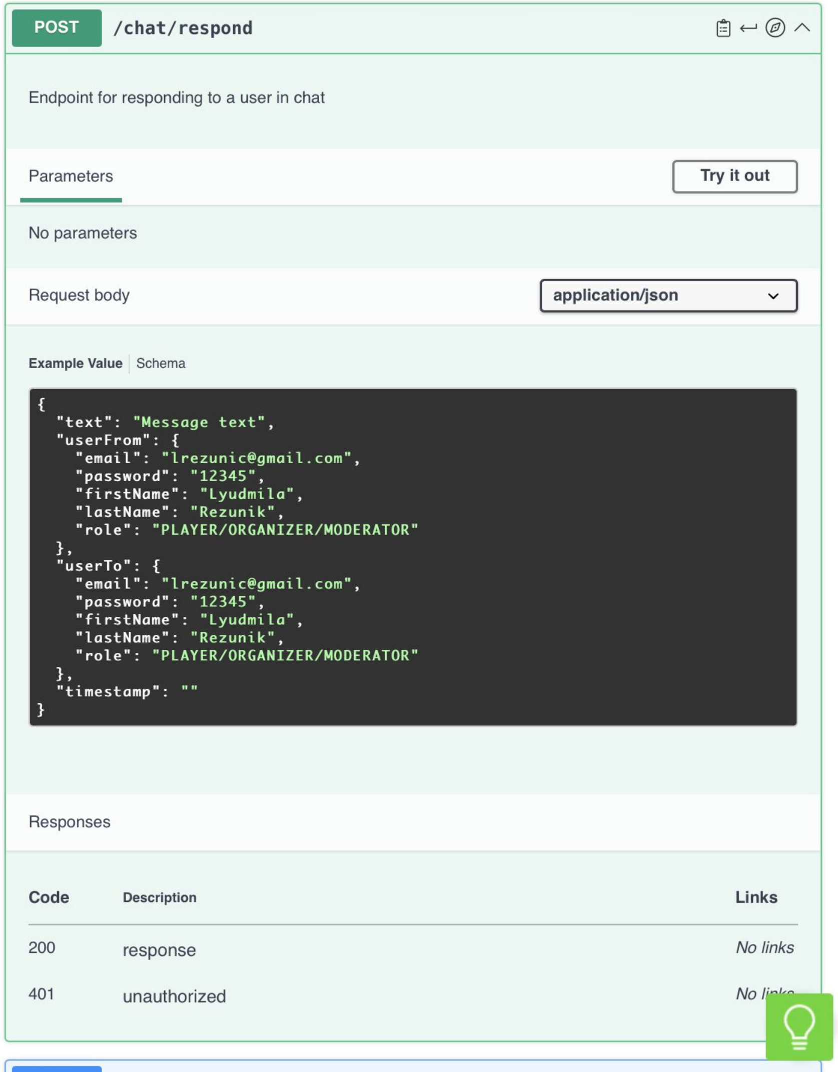
```
[
  {
    "text": "Message text",
    "user": {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    },
    "timestamp": ""
  }
]
```

API usage <ChatService>

Scenario: Chat with viewers

Steps:

User views all the messages in chat and selects
A message he wants to respond to ->
User writes the response and selects
“Send” option ->
The service is invoked ->
The service returns the status of the request
to the client



POST /chat/respond

Endpoint for responding to a user in chat

Parameters

Try it out

No parameters

Request body application/json

Example Value | Schema

```
{
  "text": "Message text",
  "userFrom": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "userTo": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "timestamp": ""
}
```

Responses

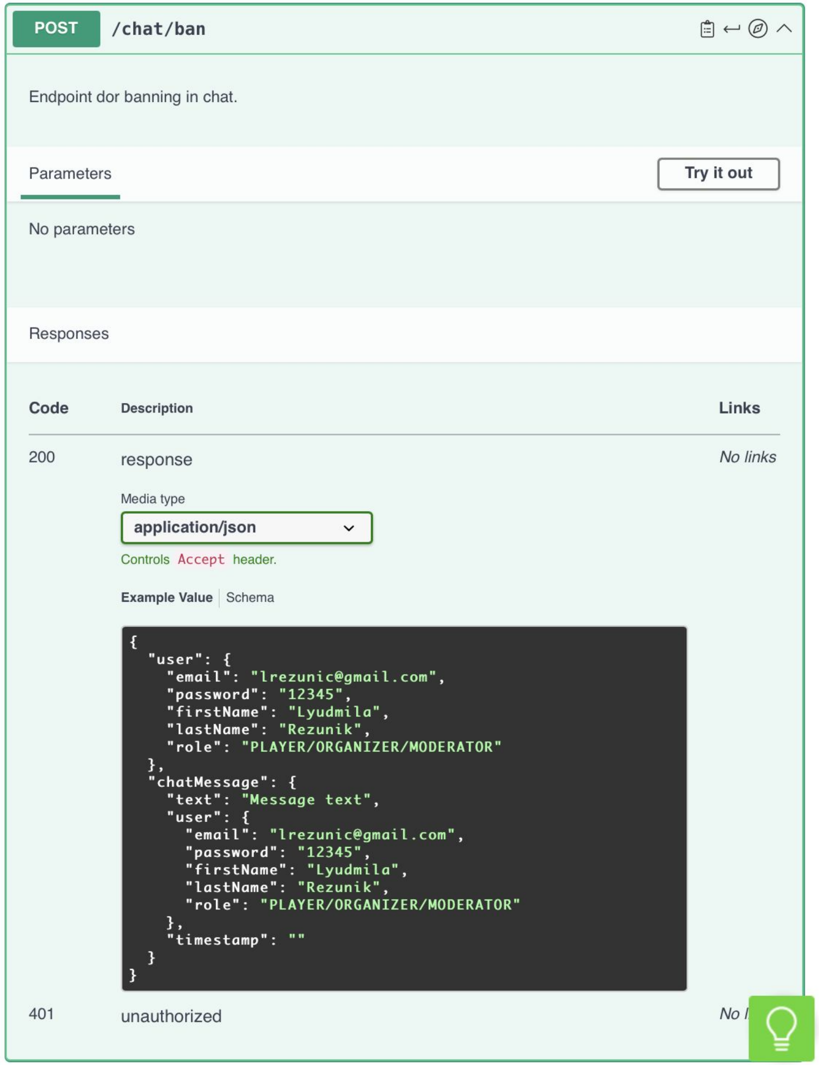
Code	Description	Links
200	response	No links
401	unauthorized	No links

API usage <ChatService>

Scenario: Ban user in chat

Steps:

Chat moderator views all the messages in chat
and selects a message he wants to ban the user for ->
Moderator chooses “Ban” option ->
The service is invoked ->
The service returns the status of the request
to the client



/chat/ban

Endpoint dor banning in chat.

Parameters

No parameters

Responses

Code	Description	Links
200	response	No links
401	unauthorized	No l

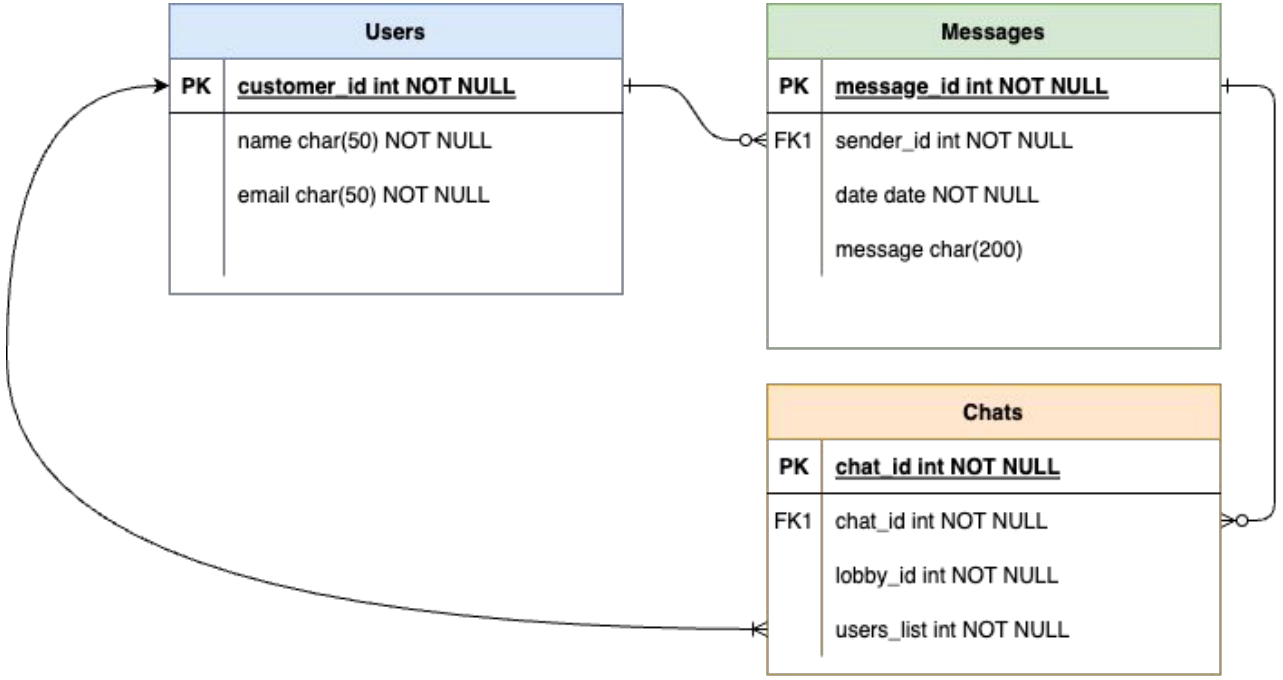
Media type: application/json

Controls `Accept` header.

```
{
  "user": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "chatMessage": {
    "text": "Message text",
    "user": {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    },
    "timestamp": ""
  }
}
```

Physical schema for chat microservice

Temporary table is created for keeping track of chats user list. Messages are connected to users via sender_id and connected to chats via chat_id, their primary keys.



Users	
PK	customer_id int NOT NULL
	name char(50) NOT NULL
	email char(50) NOT NULL

Messages	
PK	message_id int NOT NULL
FK1	sender_id int NOT NULL
	date date NOT NULL
	message char(200)

Chats	
PK	chat_id int NOT NULL
FK1	chat_id int NOT NULL
	lobby_id int NOT NULL
	users_list int NOT NULL

Team work

Lyudmila Rezunik – Lobby Service (logical data model and physical schema in the form of SQL dump)

Teona Sadulaeva – Chat Service (logical data model and physical schema)

Thank you