DuelHub

COMPETITIVE GAMING PLATFORM

Full Design

Project Description

The product aims to enhance gaming experience by providing a convenient and flexible platform for organizing online matches and tournaments.
DuelHub helps people connect and participate in numerous tournaments, letting viewers follow their favourite teams and players.

Team:
Lyudmila Rezunik
Teona Sadulaeva

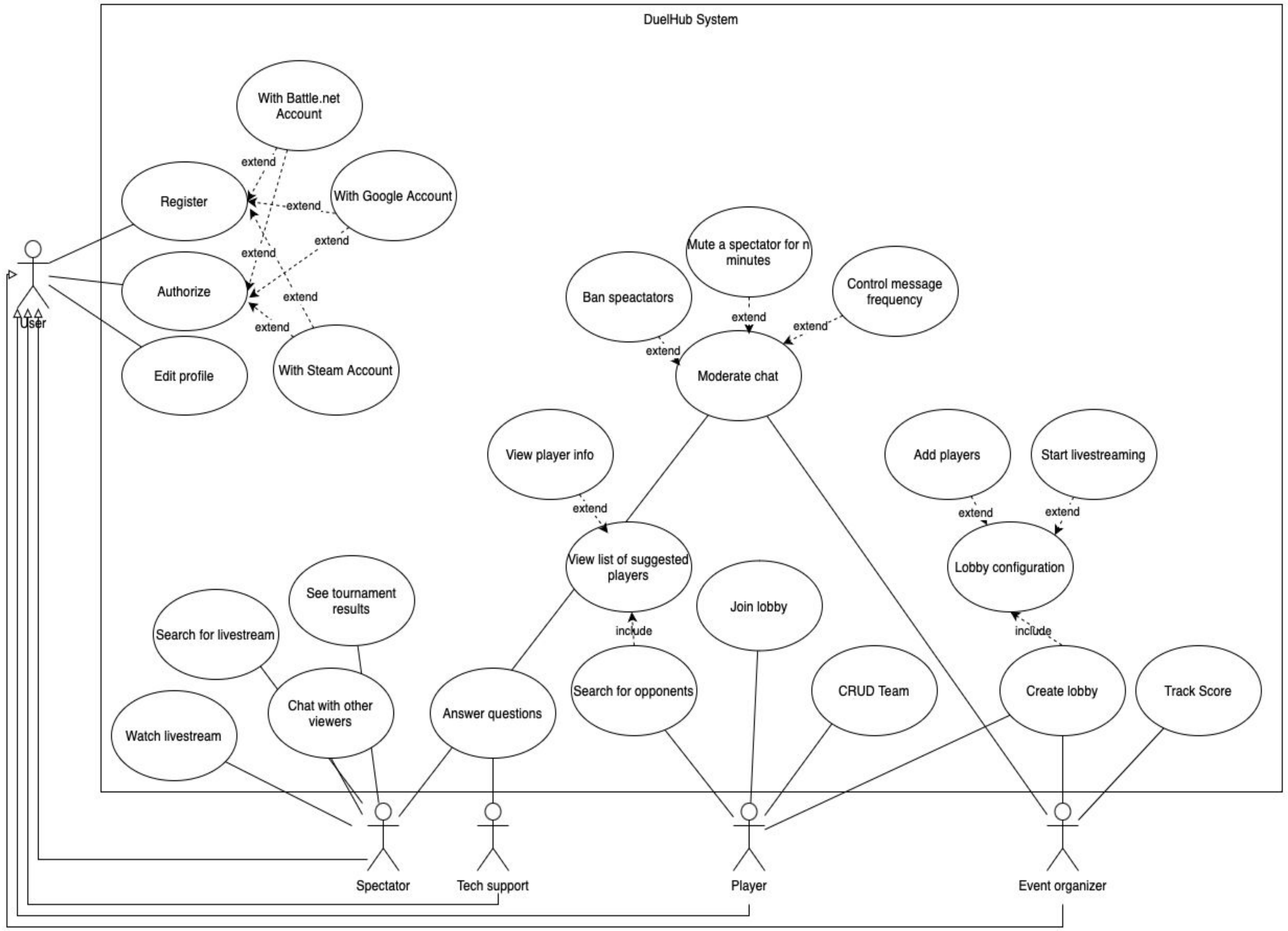
Repo:
https://github.com/teopalmer/duelhub

Report:

Goal

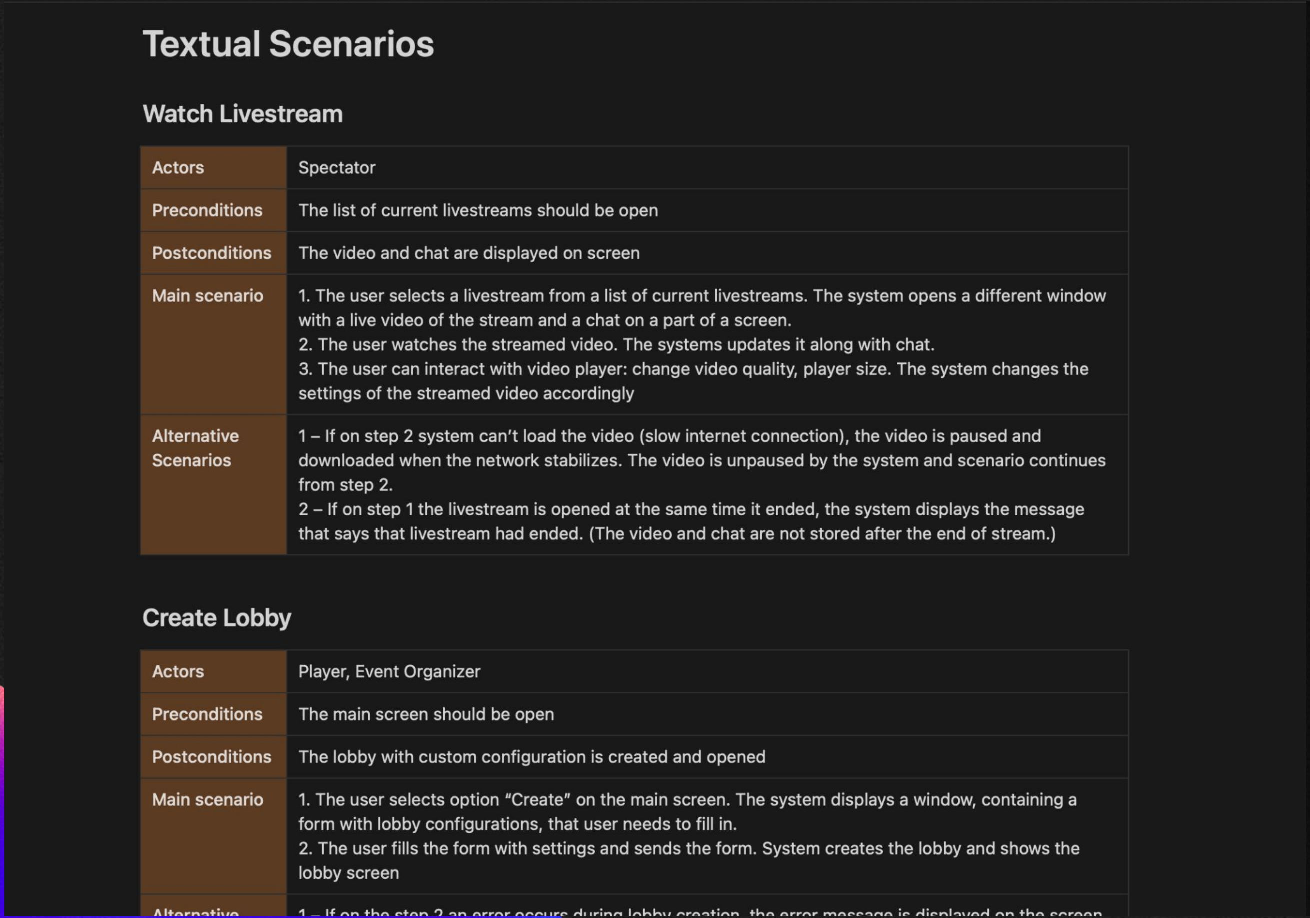
Design software that serves as an independent competitive gaming platform for online multiplayer PvP gamers.

Use Cases



Use Cases

The view of the whole diagram (in next sections each user role is shown separately)



Textual Scenarios

Watch Livestream

Actors	Spectator
Preconditions	The list of current livestreams should be open
Postconditions	The video and chat are displayed on screen
Main scenario	1. The user selects a livestream from a list of current livestreams. The system opens a different window with a live video of the stream and a chat on a part of a screen. 2. The user watches the streamed video. The systems updates it along with chat. 3. The user can interact with video player: change video quality, player size. The system changes the settings of the streamed video accordingly
Alternative Scenarios	1 – If on step 2 system can't load the video (slow internet connection), the video is paused and downloaded when the network stabilizes. The video is unpaused by the system and scenario continues from step 2. 2 – If on step 1 the livestream is opened at the same time it ended, the system displays the message that says that livestream had ended. (The video and chat are not stored after the end of stream.)

Create Lobby

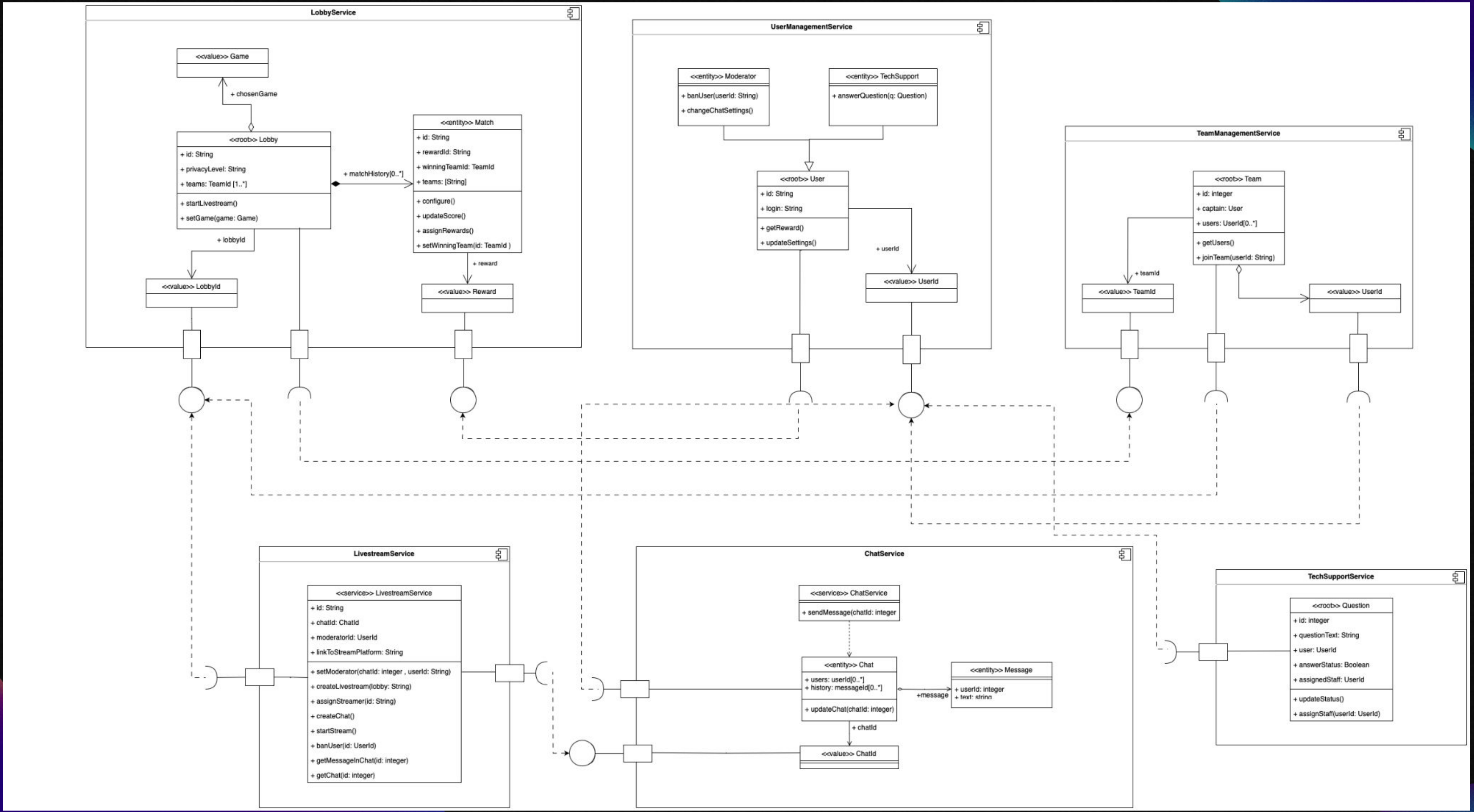
Actors	Player, Event Organizer
Preconditions	The main screen should be open
Postconditions	The lobby with custom configuration is created and opened
Main scenario	1. The user selects option "Create" on the main screen. The system displays a window, containing a form with lobby configurations, that user needs to fill in. 2. The user fills the form with settings and sends the form. System creates the lobby and shows the lobby screen
Alternative	1 – If on the step 2 an error occurs during lobby creation, the error message is displayed on the screen

Scenarios

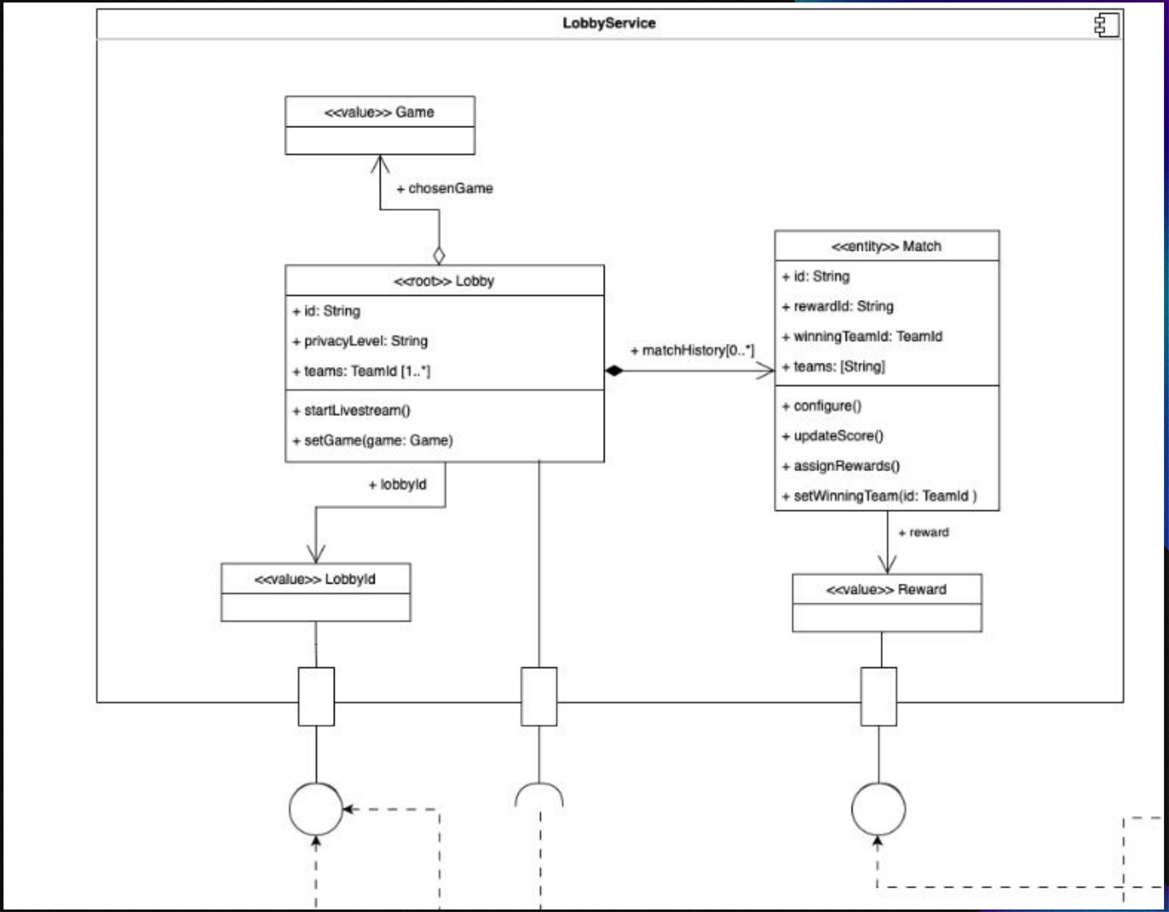
Textual scenarios are written in Wiki: https://equable-scorpio-c13.notion.site/DualHub-Project-62ef3aff8a6c46bc8b4909b44714bb89?pvs=4

System Architecture

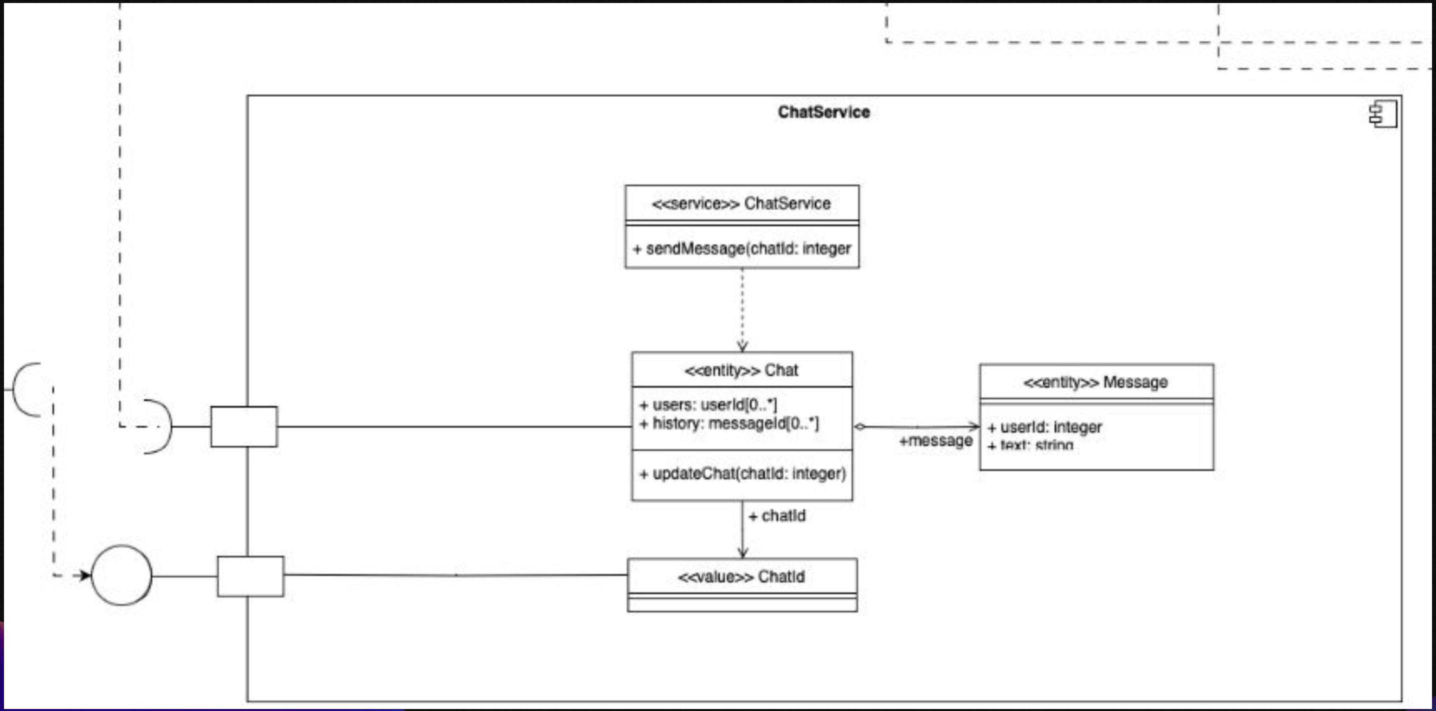
System Architecture



LobbyService



ChatService



Principles

RESTFUL PRINCIPLES

As we apply REST architecture style we need to conform to principles to make our system RESTful

1. Uniform interface
2. Stateless technology
3. Layered system
4. Caching
5. Code on demand

SINGLE CONCERN

A microservice should do one thing and one thing only. Its interface should expose only access points that are relevant

DESIGN FOR FAILURE

Anything can fail at any time, so there should be a way to detect failures and quickly recover. (e.g. Circuit Breaker)

SOLID...

Solution Stack

01.

Implementation

– Java backend with Spring Boot as a REST API framework
– Frontend developed with Typescript, React + Redux for state management

02.

Asyncronous Intercations

– RabbitMQ for message queues
– Sentry for application monitoring

03.

Testing

– JUnit for backend testing
– Jest for frontend testing

04.

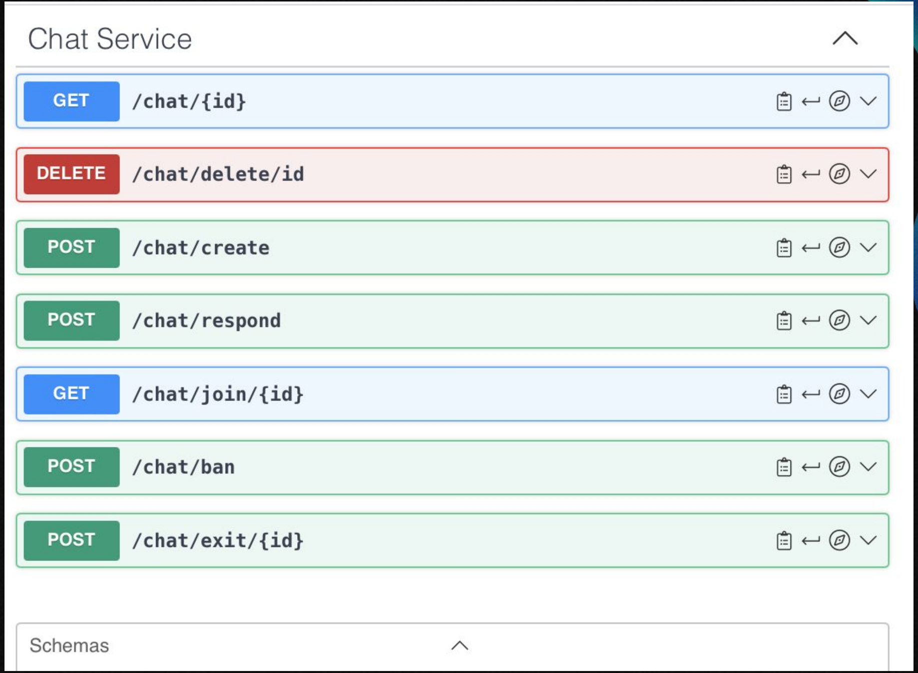
Operations

– Makefile code build
– Jenkins CI/CD pipeline
– Docker
– Kubernetes

API Usage

Chat Service. API Summary

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



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}

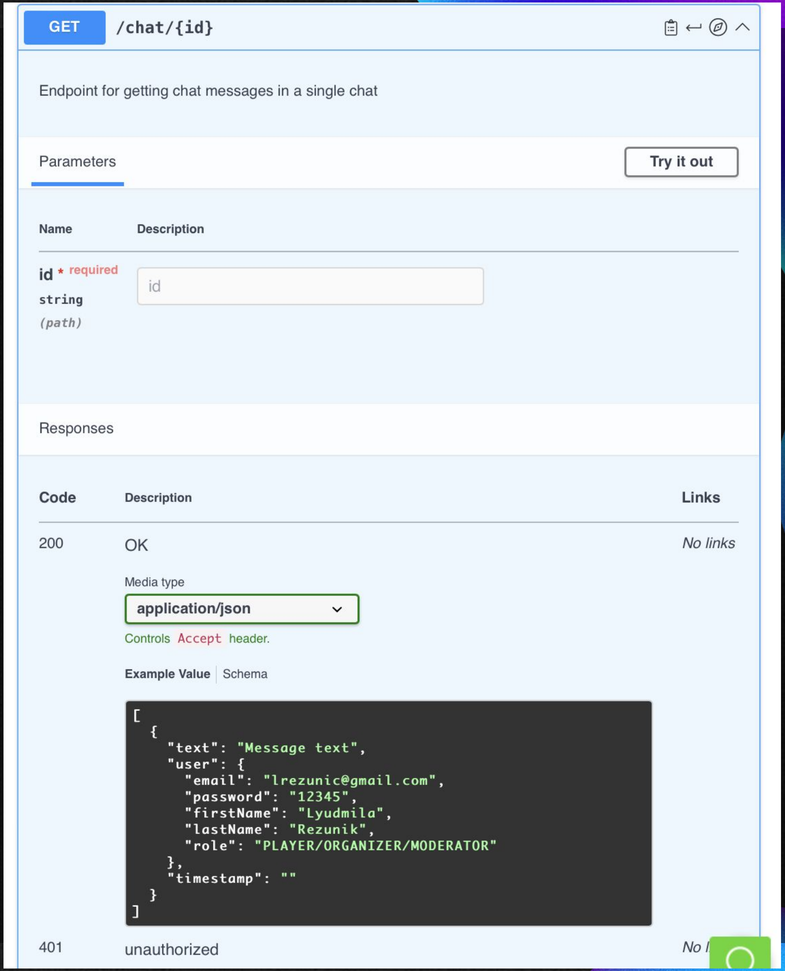
Schemas

API Usage

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

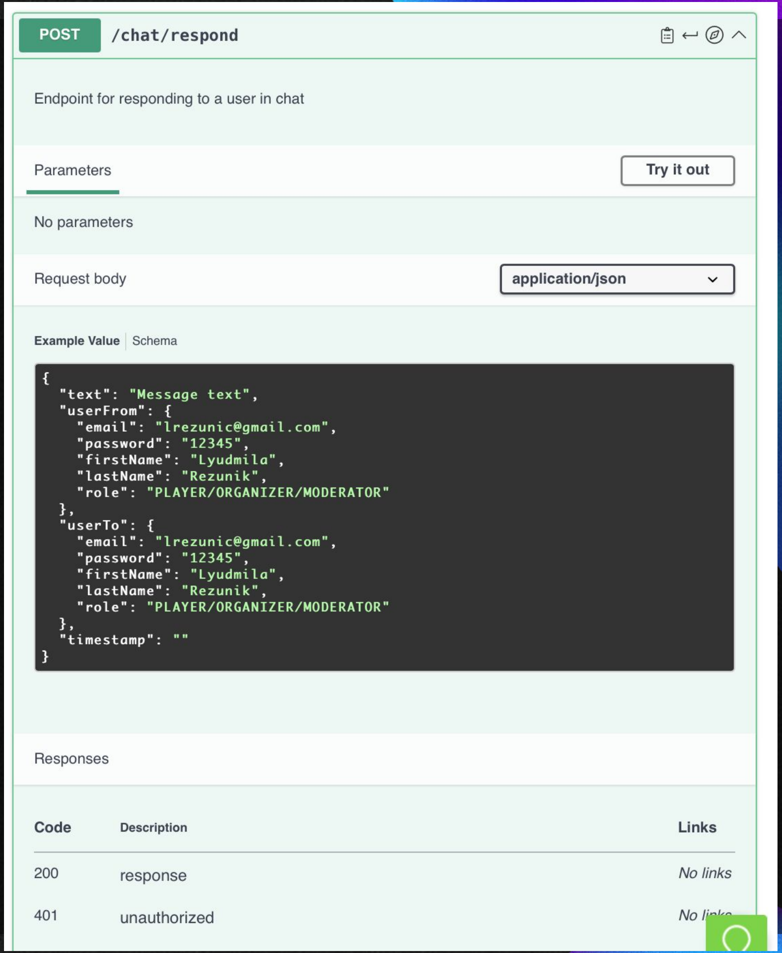
```
[
  {
    "text": "Message text",
    "user": {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    },
    "timestamp": ""
  }
]
```

API Usage

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": ""
}
```

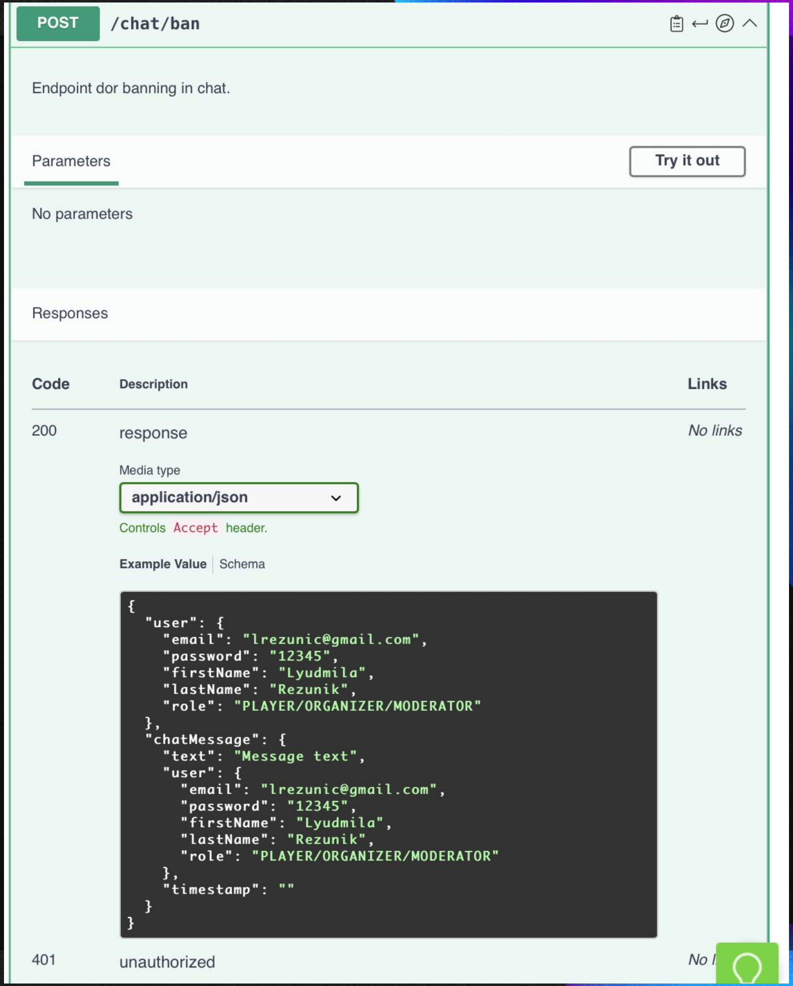
Responses

Code	Description	Links
200	response	No links
401	unauthorized	No links

API Usage

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


POST /chat/ban

Endpoint dor banning in chat.

Parameters — Try it out

No parameters

Responses

Code	Description	Links
200	response	No links
401	unauthorized	No l

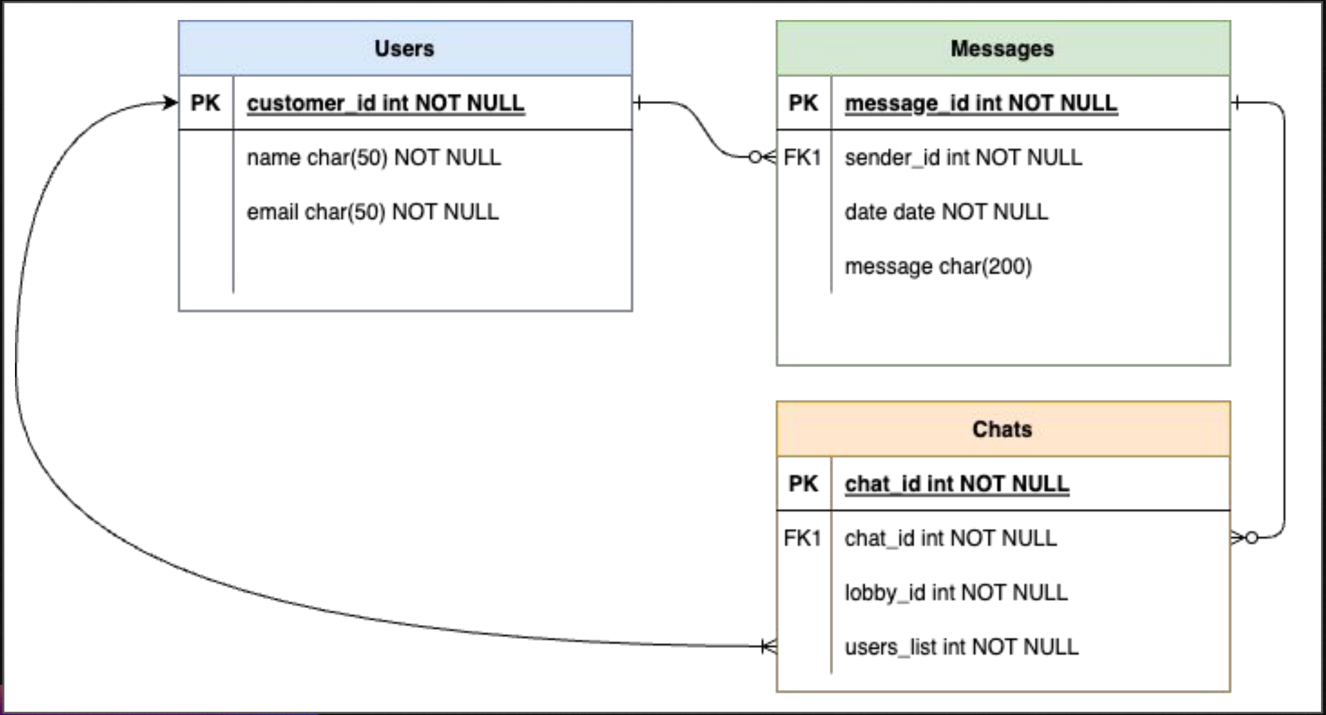
Media type: application/json

Controls Accept header.

Example Value | Schema

```
{
  "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": ""
  }
}
```

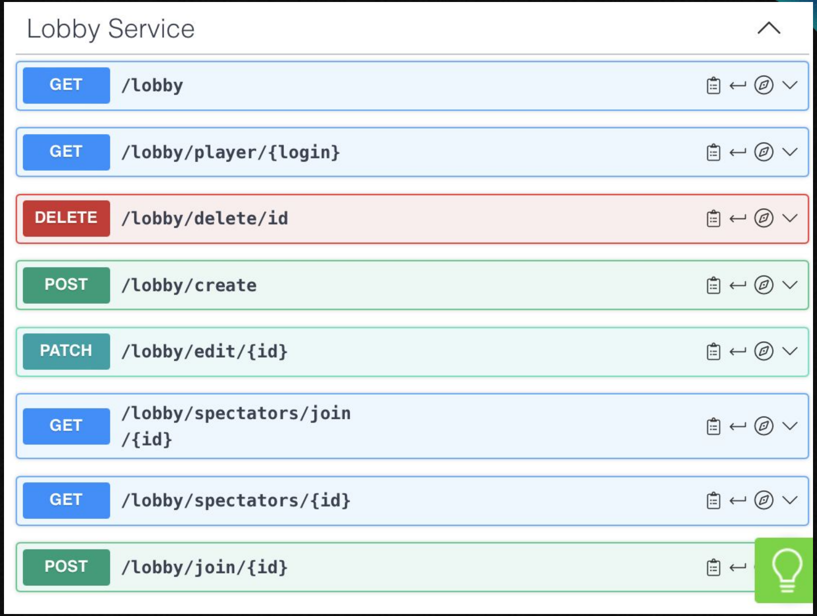
Chat Service. Logical Data Model



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.

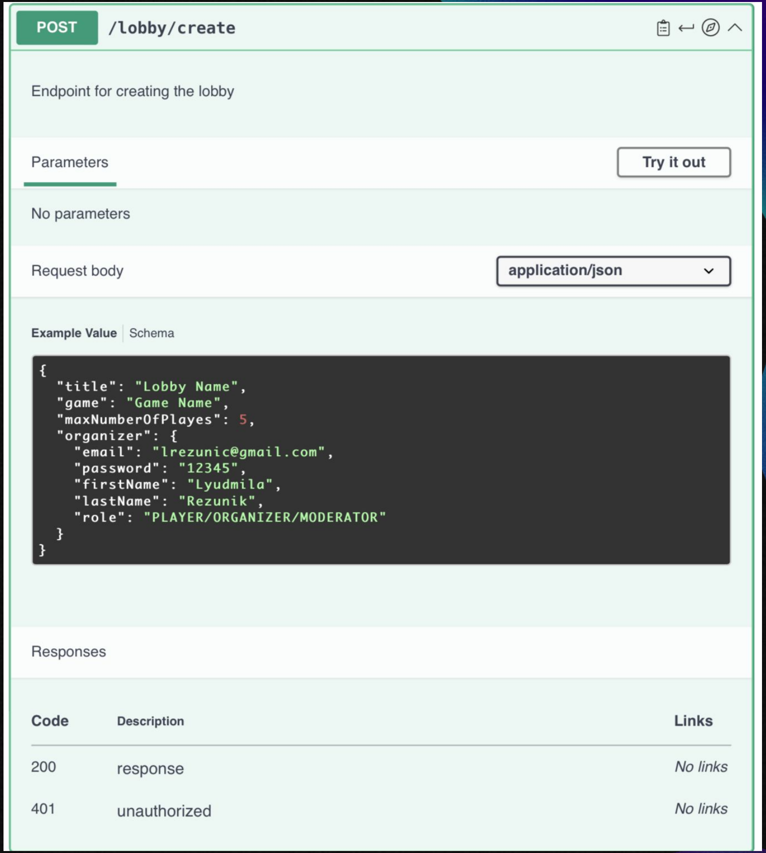

Lobby Service

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

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



/lobby/create

Endpoint for creating the lobby

Parameters

No parameters

Request body

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

Responses

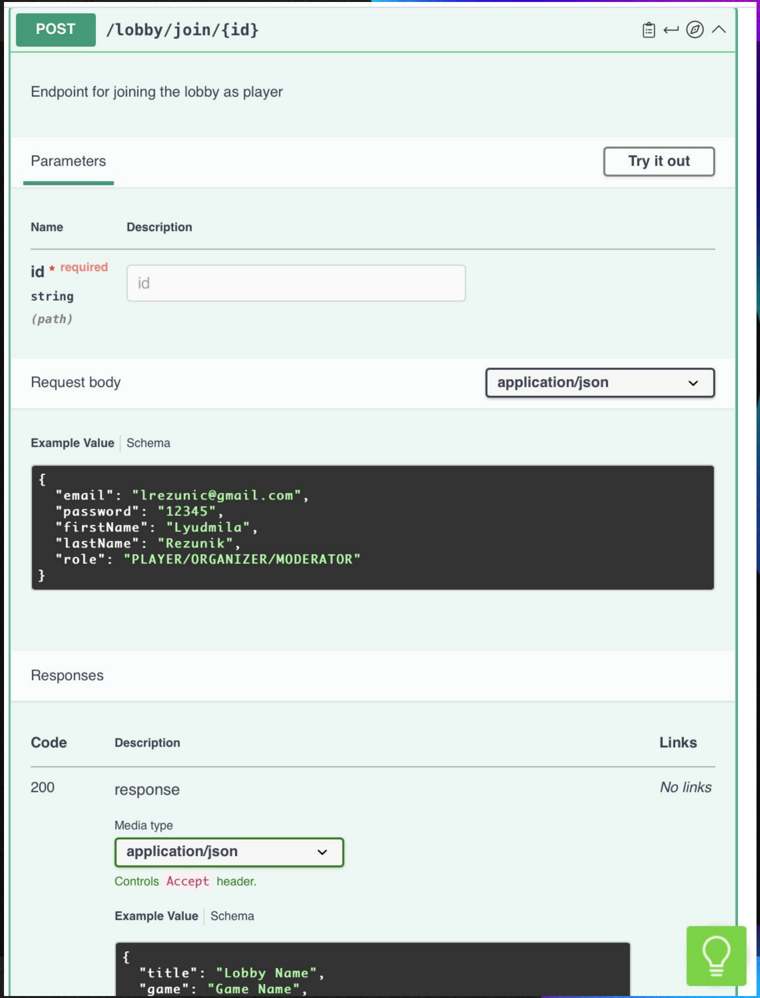
Code	Description	Links
200	response	*No links*
401	unauthorized	*No links*

API Usage

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

Parameters

Name	Description
id * required string (path)	id

Request body application/json

Example Value | Schema

```
{
  "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.

Example Value | Schema

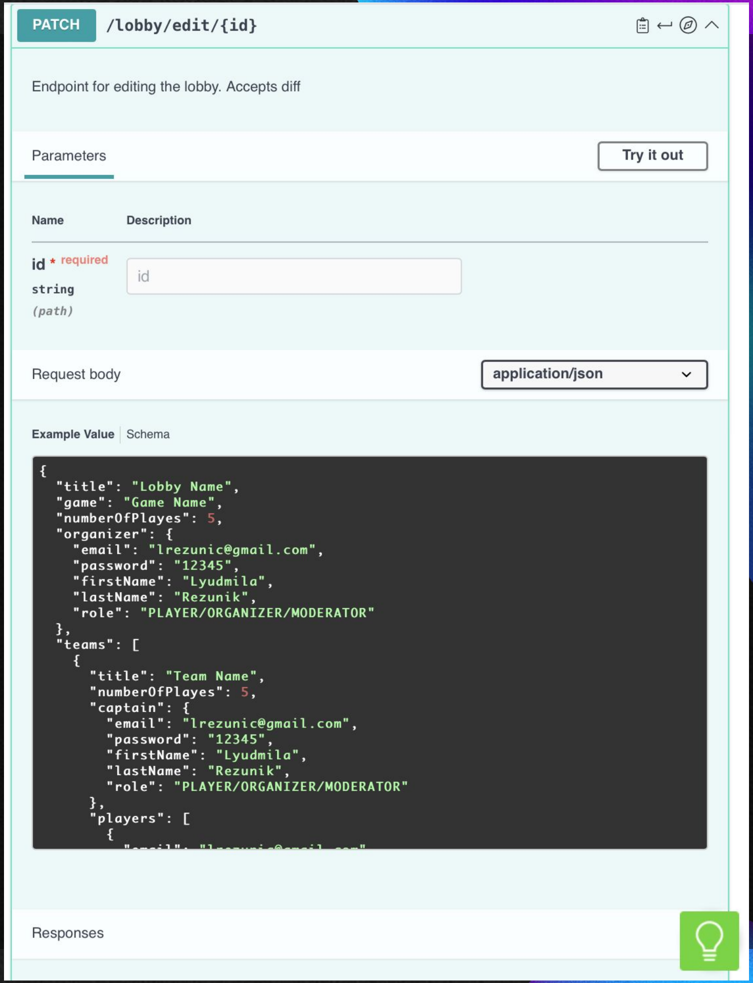
```
{
  "title": "Lobby Name",
  "game": "Game Name",
```

API Usage

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

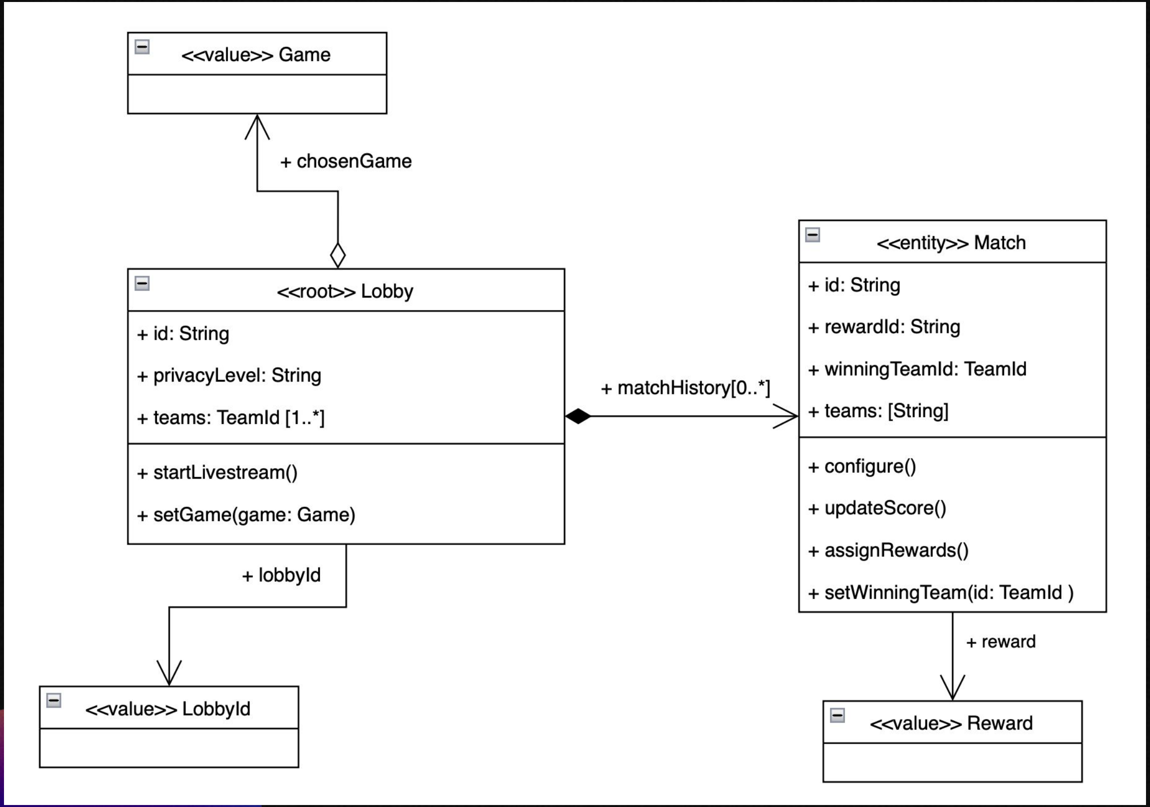
Request body: application/json

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": [
        {
```

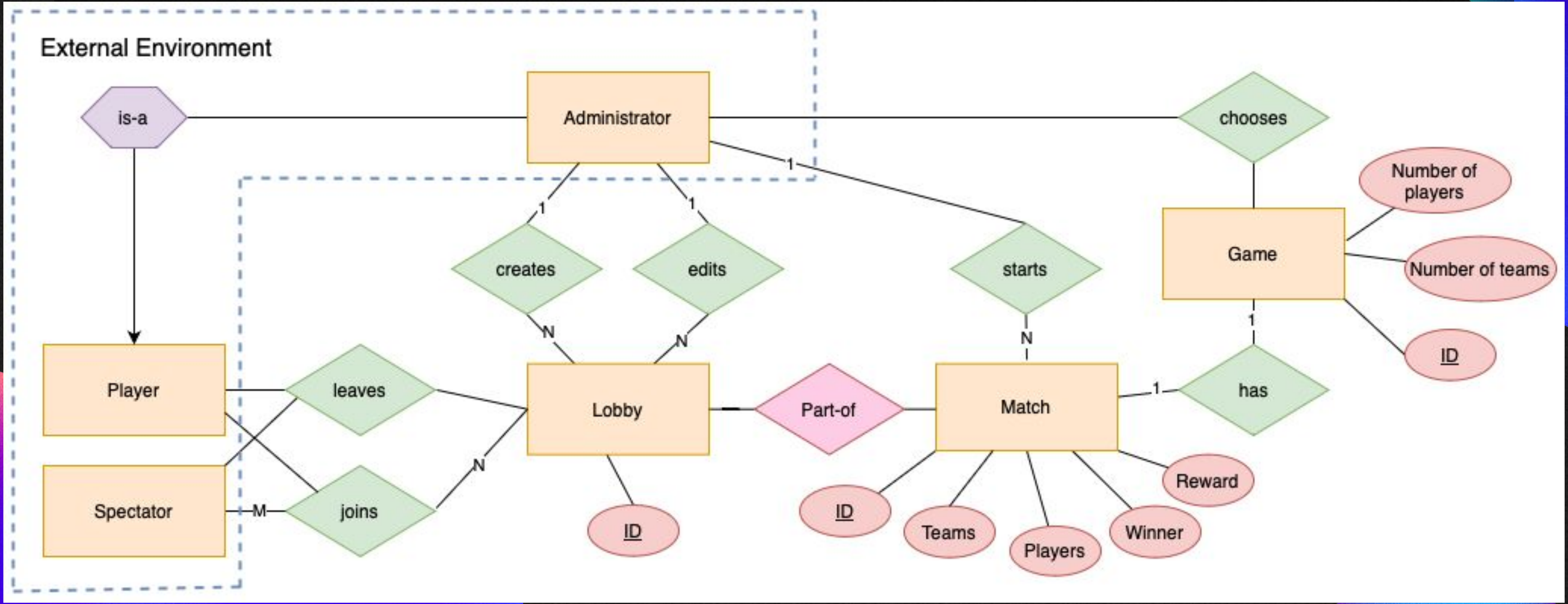
Responses

Lobby Service. Class Diagram



Lobby Service. ER Diagram

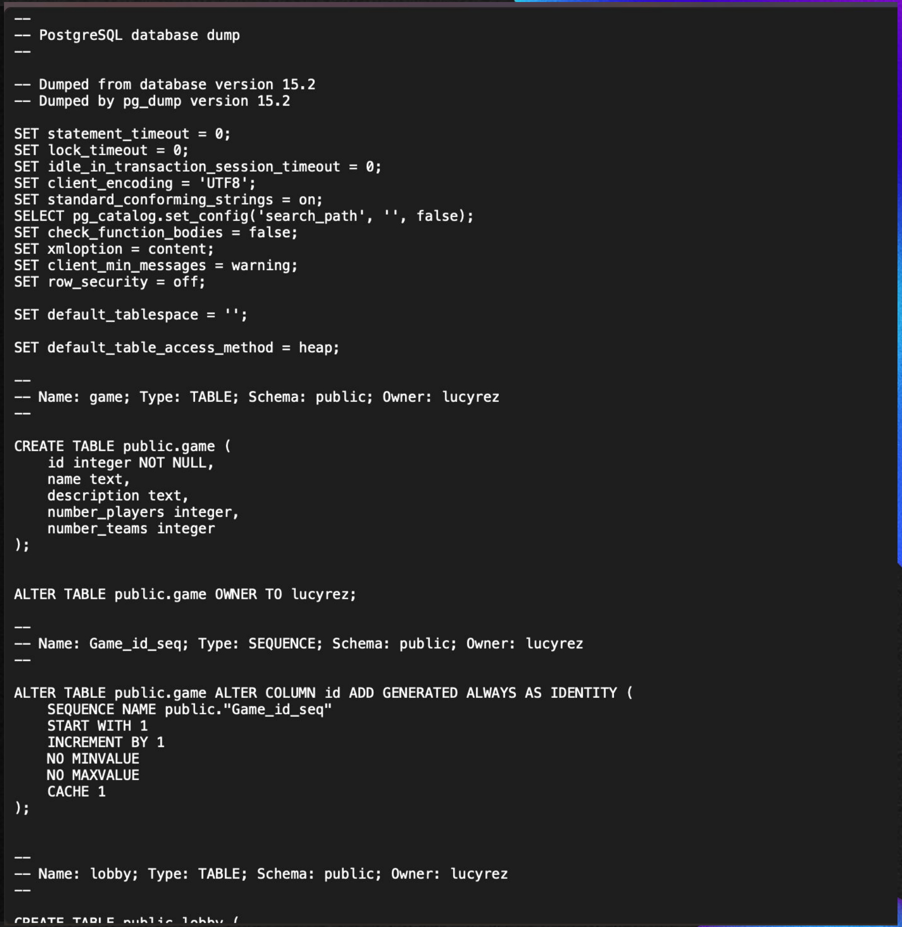
There are entities that are considered as “External Environment” (that interact with the service)



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 (
```

Design Cases

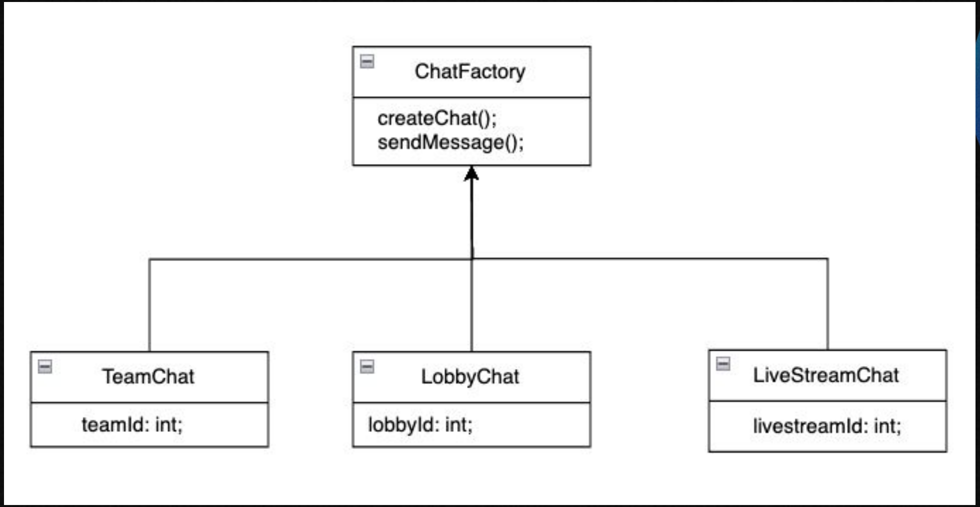
Design Case 1

Factory Pattern for Messages

All chats need to share the same functionality, new types of chats should be added easily.

Chats need to be interchangeable.

Solution: Apply factory pattern



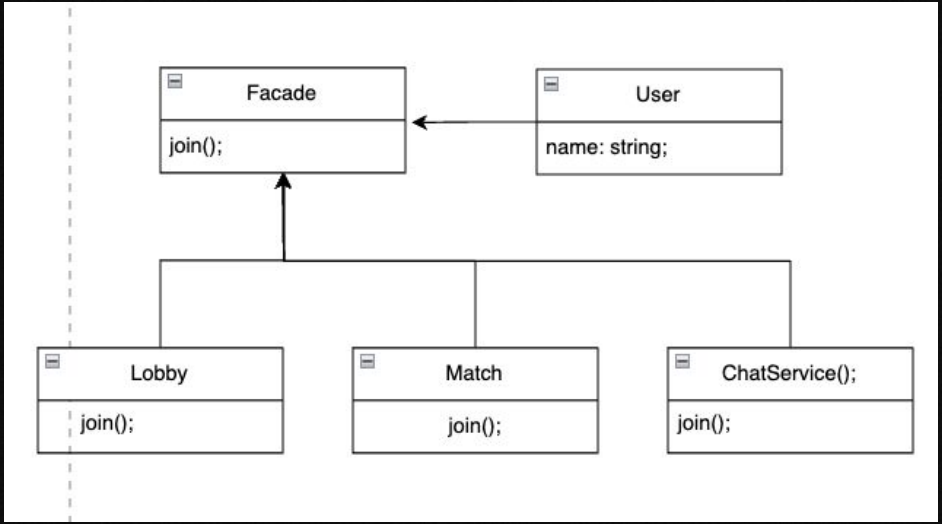
Design Case 2

Facade Pattern for Joining Matches

When user joins a match, he also joins lobby and its chat.

We need to ensure that all of this is accomplished.

Solution: Apply facade pattern



Repository

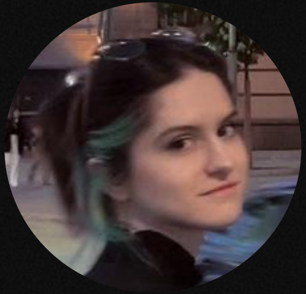
Project Repository

Link to GitHub

https://github.com/teopal
mer/duelhub

Team & Roles

Team



Sadulaeva Teona

Frontend Developer
Responsible for:
- Chat Service API and Data Modeling
- Design Cases

Rezunik Lyudmila

Backend Developer
Responsible for:
- Lobby Service API and Data Modeling
- Design Principles

Thanks !