DuelHub

Task 10. API Design

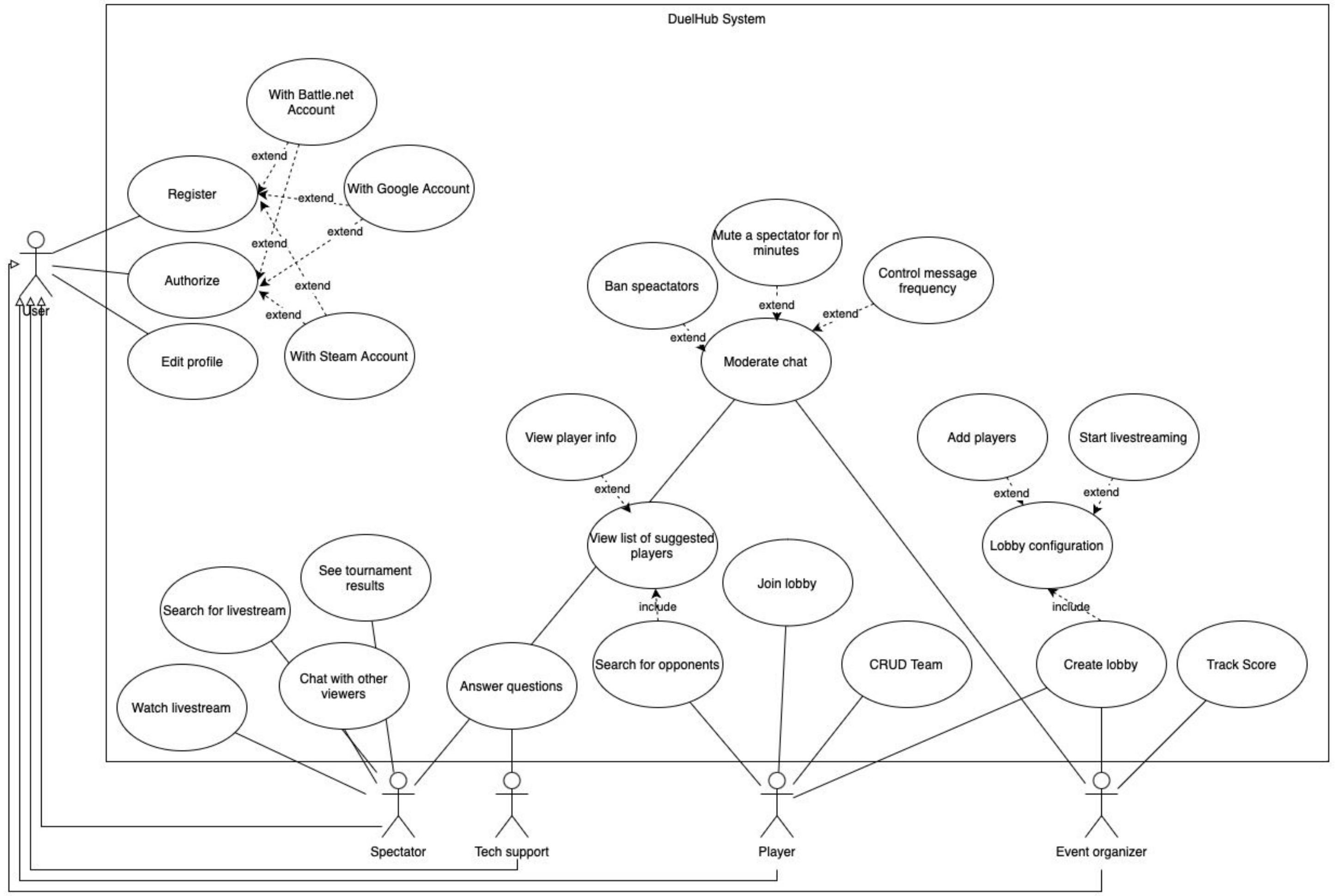
Product description

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.

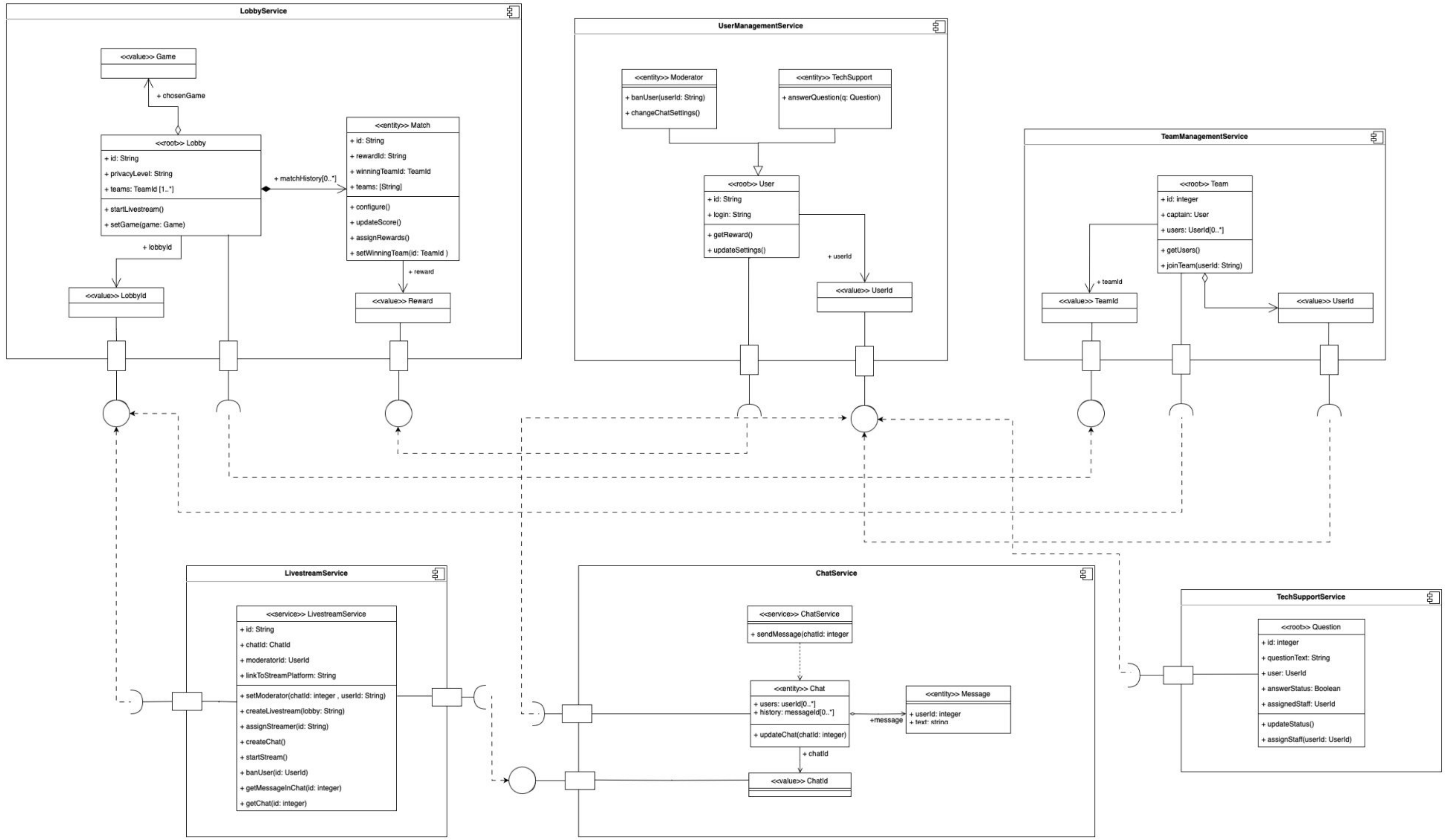
Team: Lyudmila Rezunik, Teona Sadulaeva

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

Use case diagram



Service diagram



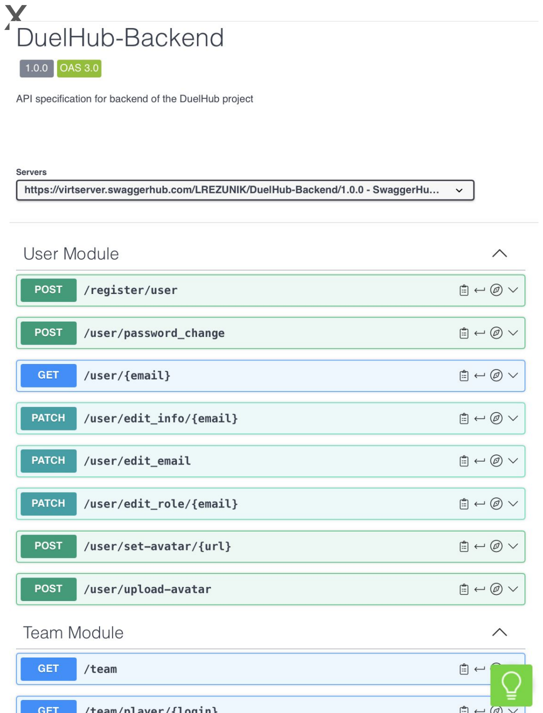
Service diagram

Our diagrams in full resolution are available here:

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

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

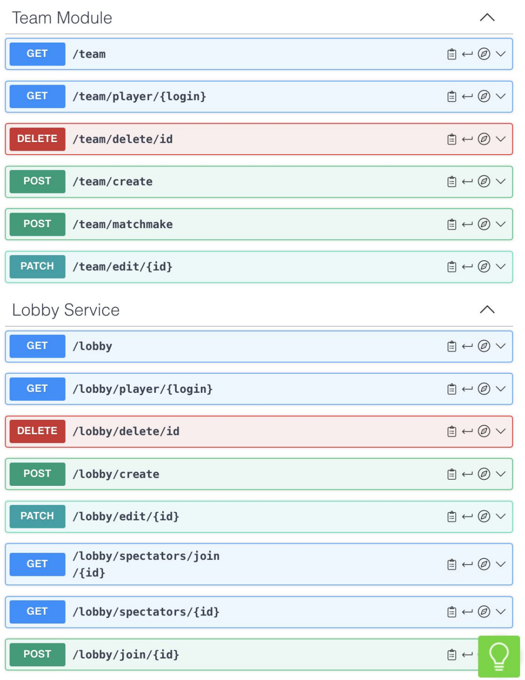
https://virtserver.swaggerhub.com/LREZUNIK/DuelHub-Backend/1.0.0 - SwaggerHu...

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}`

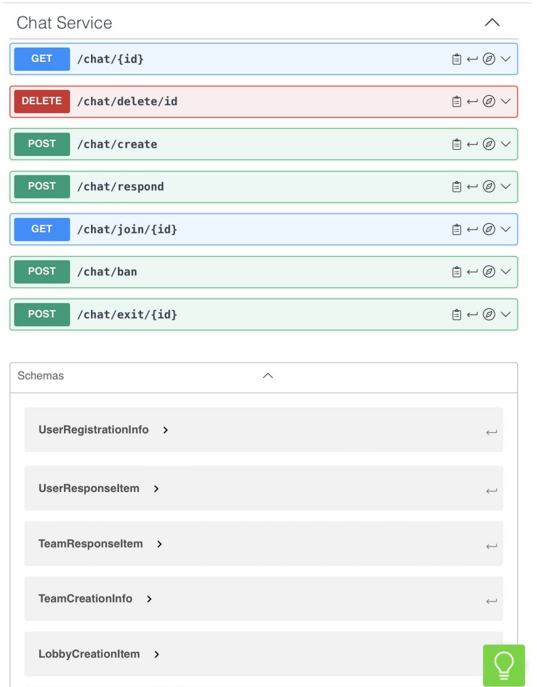


Team Module

- 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}`
- POST `/lobby/join/{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 <LobbyService>

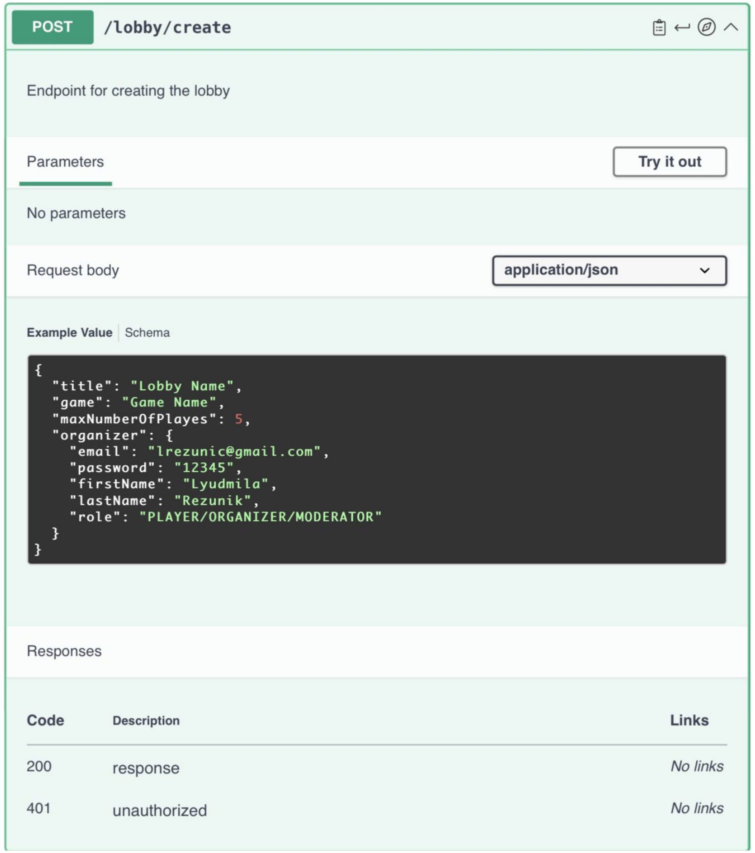
Scenario: Create Lobby

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



POST /lobby/create

Endpoint for creating the lobby

Parameters

Try it out

No parameters

Request body: application/json

Example Value | Schema

```
{
  "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>

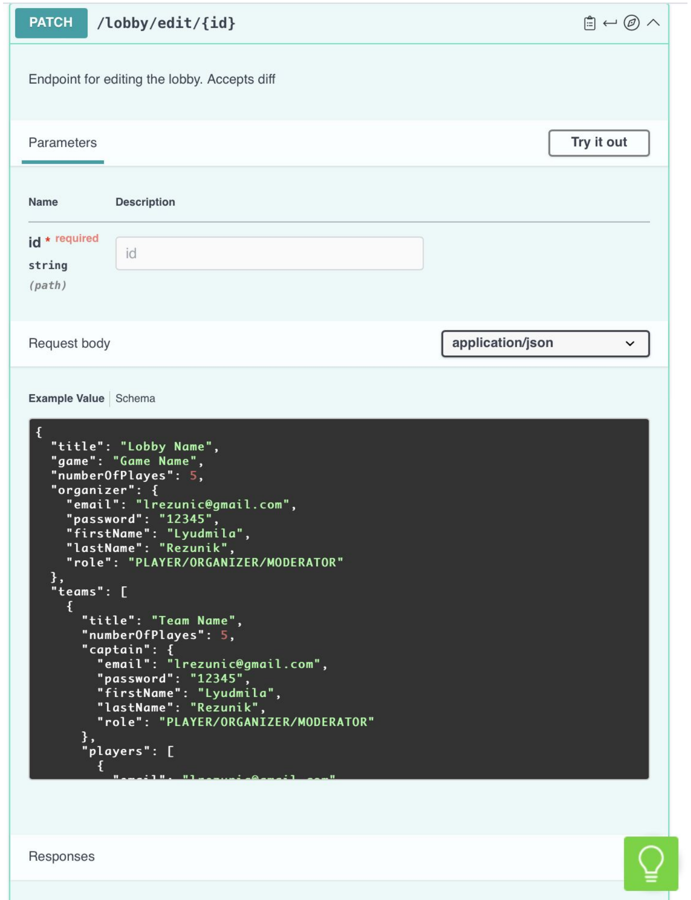
Scenario: Configure Lobby

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": [
        {
          "email": "lrezunic@gmail.com"
```

Responses

API usage <LobbyService>

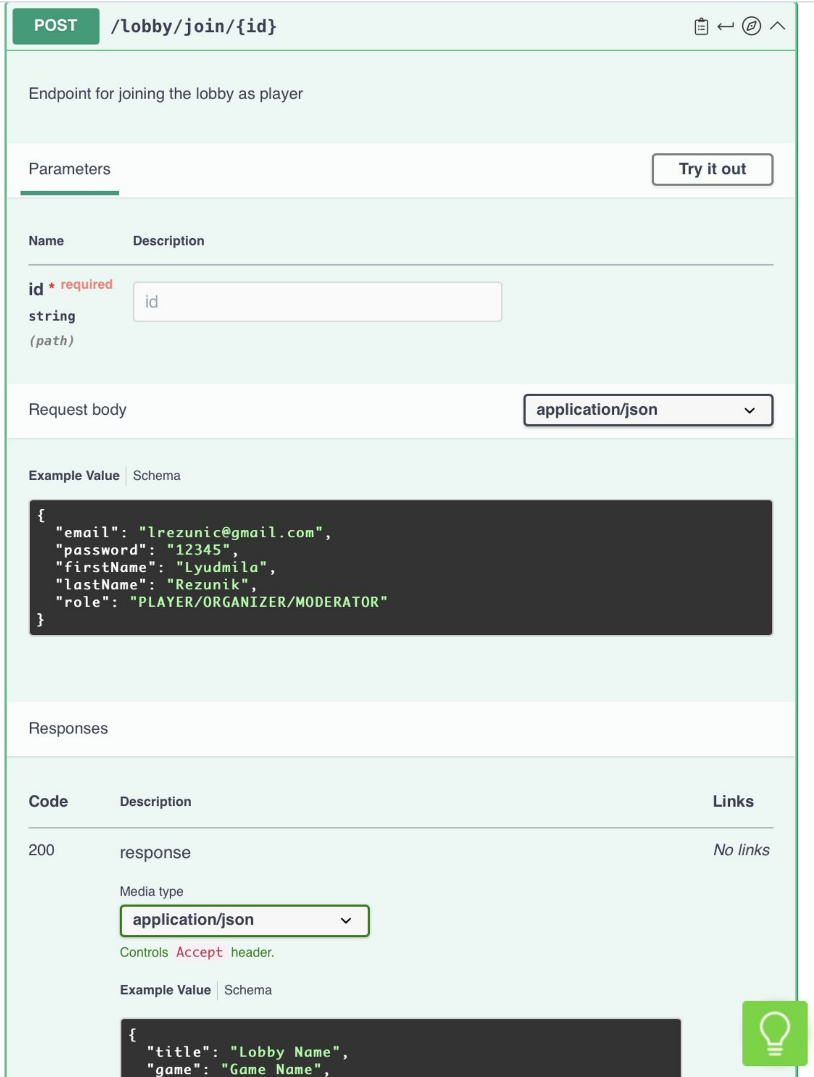
Scenario: Join Lobby

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

Parameters | Try it out

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

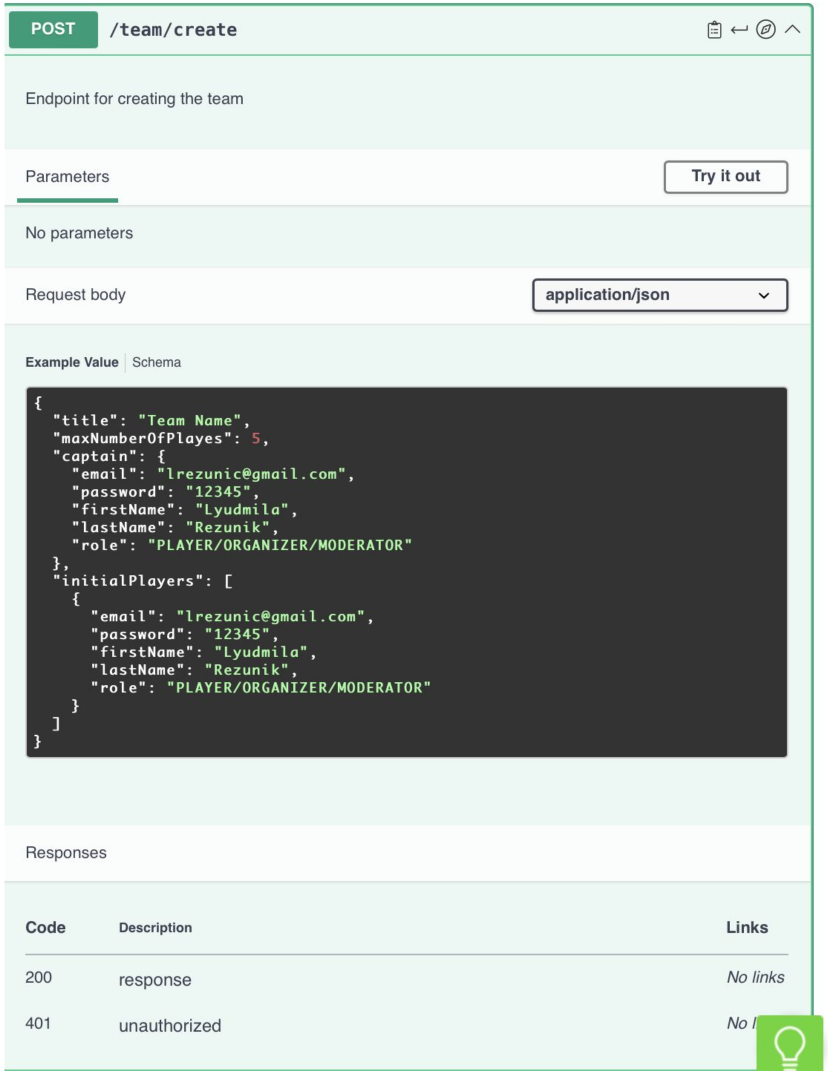
```
{
  "title": "Lobby Name",
  "game": "Game Name",
```

API usage <TeamManagementService>

Scenario: Create Team

Steps:

Player is on the main web page
and chooses “Create team” option -> then fills the data ->
The service is invoked ->
The service returns the status of the request
to the client



POST /team/create

Endpoint for creating the team

Parameters

Try it out

No parameters

Request body

application/json

Example Value | Schema

```
{
  "title": "Team Name",
  "maxNumberOfPlayes": 5,
  "captain": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "initialPlayers": [
    {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    }
  ]
}
```

Responses

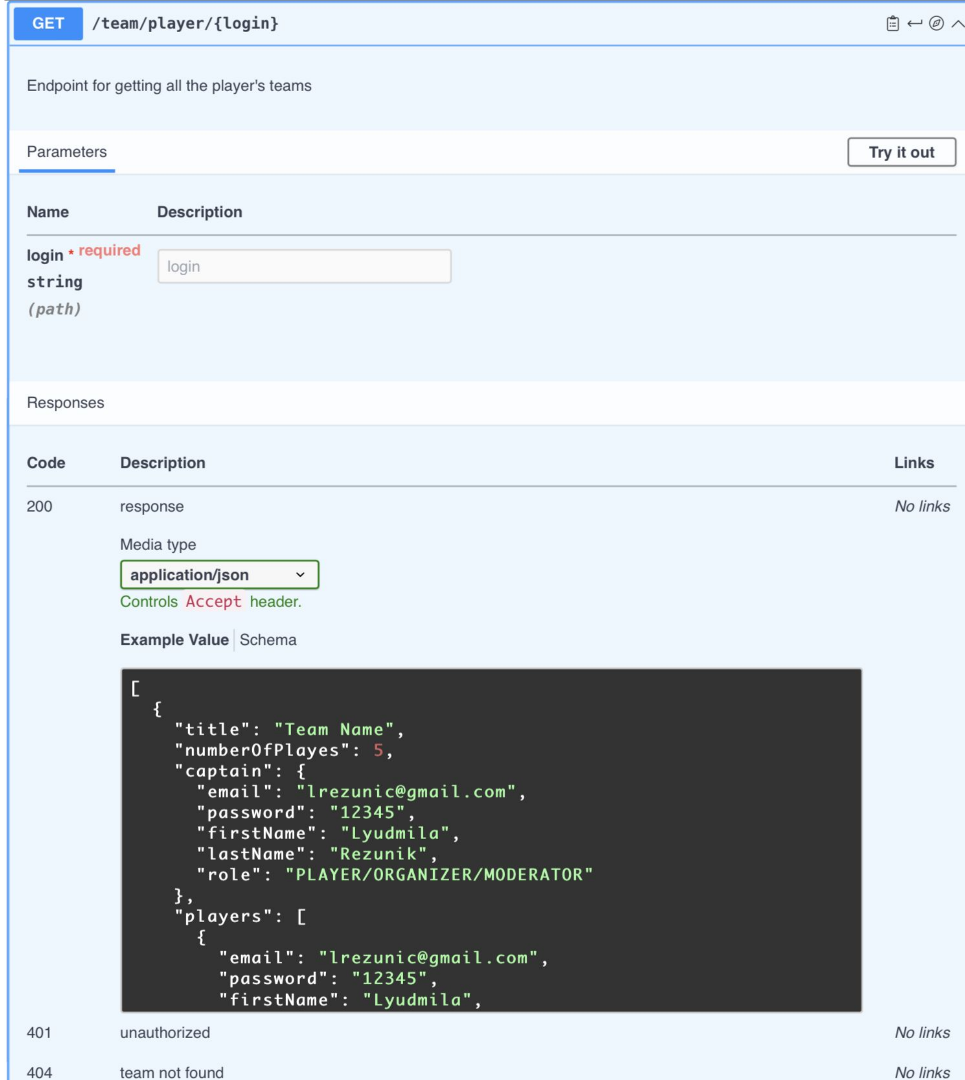
Code	Description	Links
200	response	No links
401	unauthorized	No l

API usage <TeamManagementService>

Scenario: Read Team

Steps:

Player views his profile web page ->
The service is invoked ->
The service returns data about
the player's teams



GET /team/player/{login}

Endpoint for getting all the player's teams

Parameters

Try it out

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

Responses

Code	Description	Links
200	response	*No links*
401	unauthorized	*No links*
404	team not found	*No links*

Media type: application/json
Controls Accept header.

Example Value | Schema

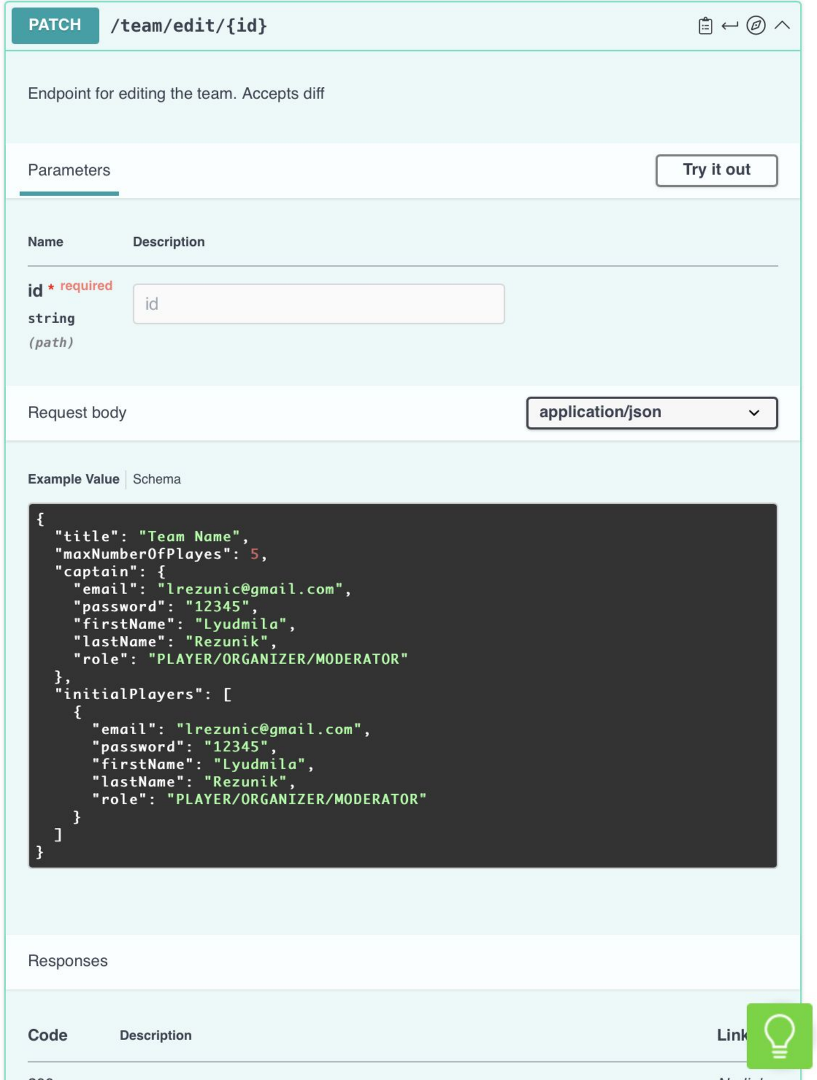
```
[
  {
    "title": "Team Name",
    "numberOfPlayes": 5,
    "captain": {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    },
    "players": [
      {
        "email": "lrezunic@gmail.com",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

API usage <TeamManagementService>

Scenario: Update Team

Steps:

Player (the captain of the team) is on the team's settings web page and chooses
"Edit team" option -> then edits team ->
The service is invoked ->
The service returns the status of the request to the client



PATCH /team/edit/{id}

Endpoint for editing the team. Accepts diff

Parameters

Try it out

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

Request body application/json

Example Value Schema

```
{
  "title": "Team Name",
  "maxNumberOfPlayes": 5,
  "captain": {
    "email": "lrezunic@gmail.com",
    "password": "12345",
    "firstName": "Lyudmila",
    "lastName": "Rezunik",
    "role": "PLAYER/ORGANIZER/MODERATOR"
  },
  "initialPlayers": [
    {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    }
  ]
}
```

Responses

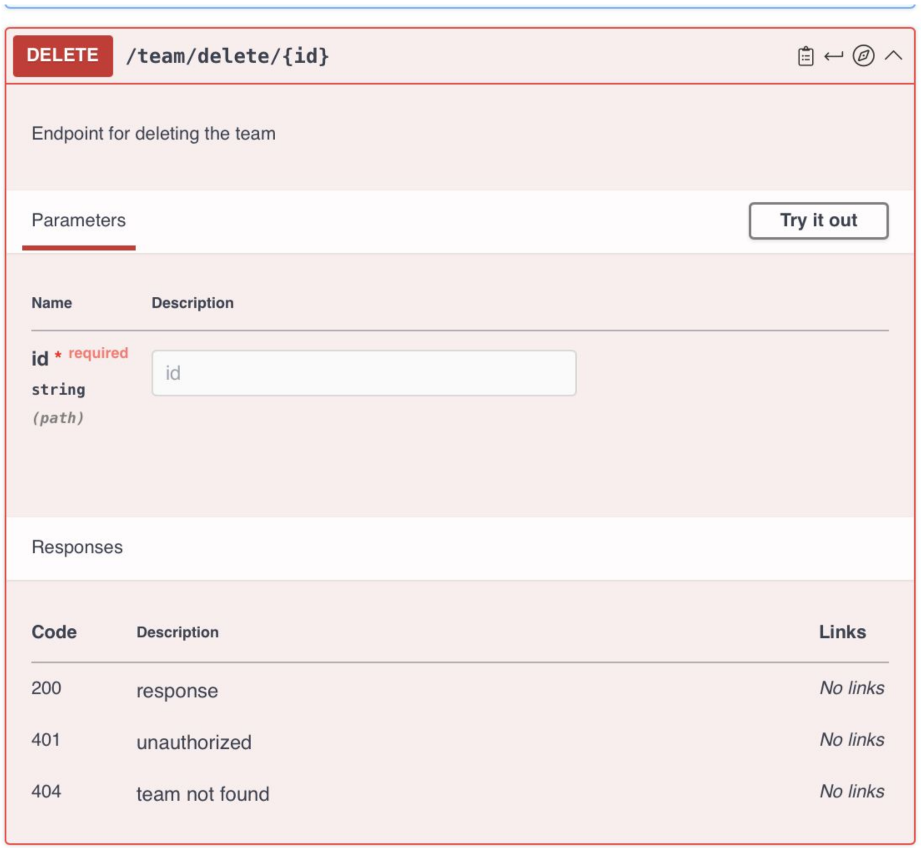
Code Description Link

API usage <TeamManagementService>

Scenario: Delete Team

Steps:

Player (the captain of the team) is on the team's settings web page and chooses "Delete team" option ->
The service is invoked ->
The service returns the status of the request to the client



/team/delete/{id}

Endpoint for deleting the team

Name	Description
id * required string (path)	id

Responses

Code	Description	Links
200	response	No links
401	unauthorized	No links
404	team not found	No links

API usage <ChatService>

Scenario: Chat with viewers

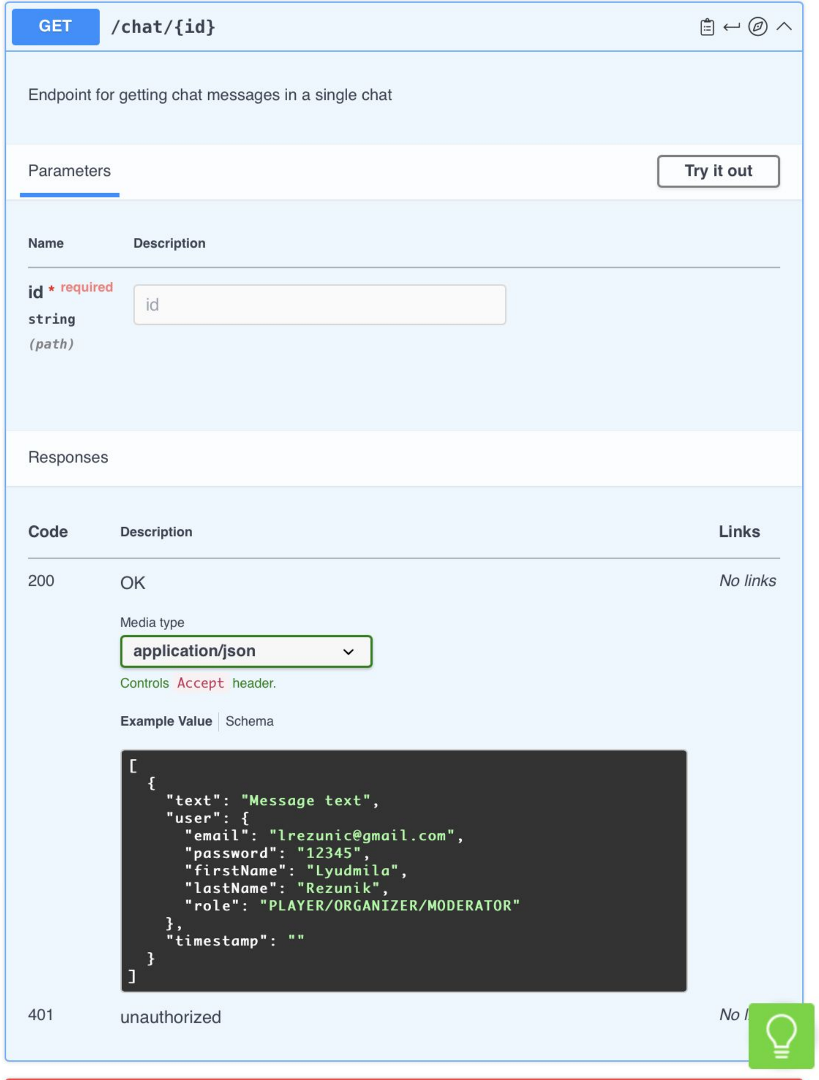
Steps:

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

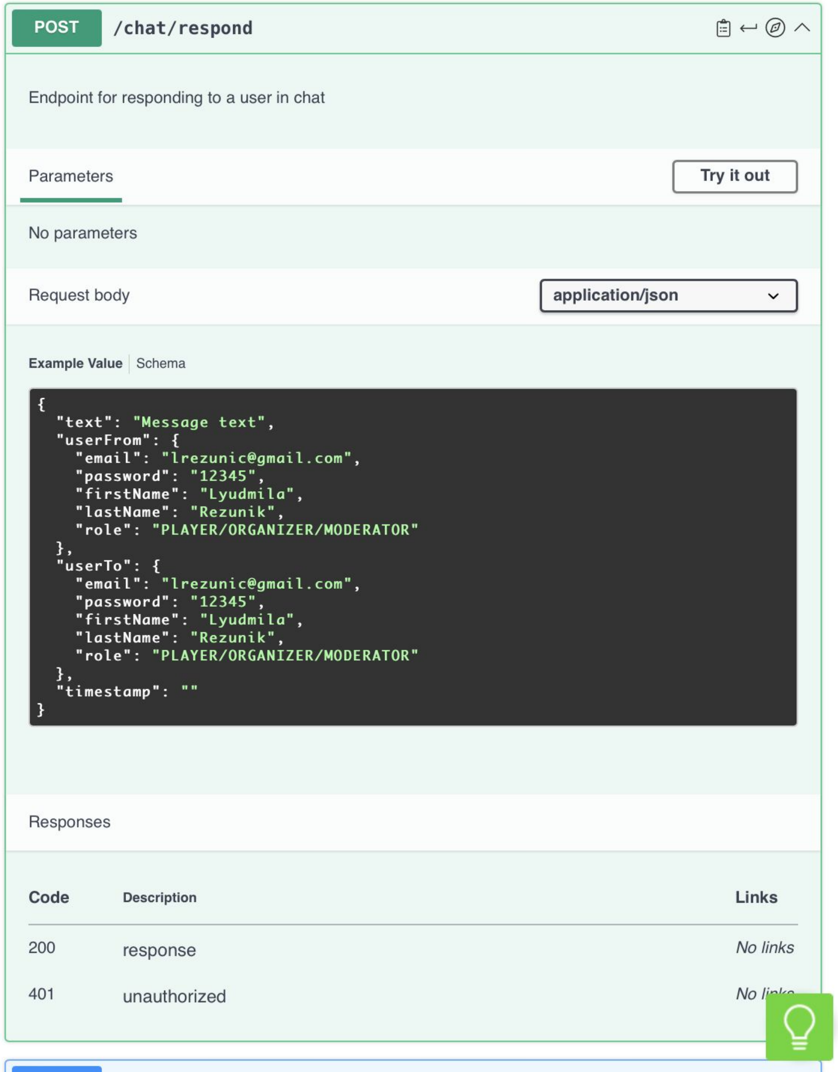
```
[
  {
    "text": "Message text",
    "user": {
      "email": "lrezunic@gmail.com",
      "password": "12345",
      "firstName": "Lyudmila",
      "lastName": "Rezunik",
      "role": "PLAYER/ORGANIZER/MODERATOR"
    },
    "timestamp": ""
  }
]
```

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

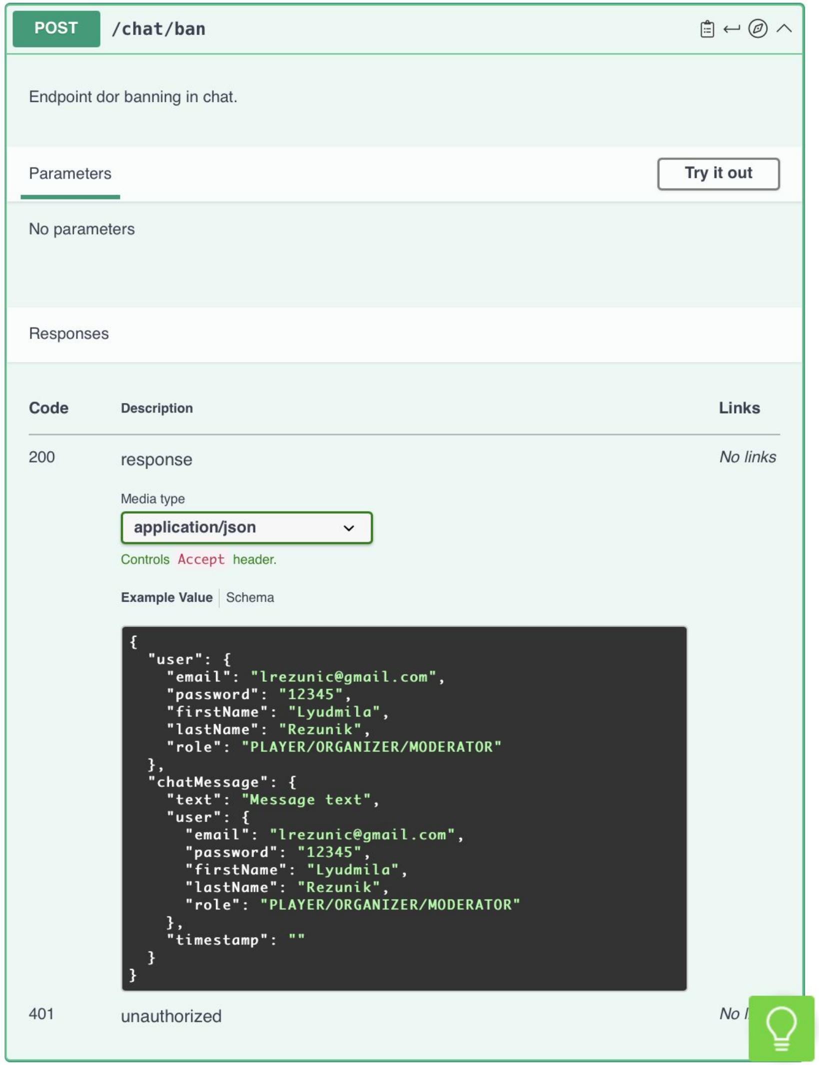
Code	Description	Links
200	response	*No links*
401	unauthorized	*No links*

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


POST /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.

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": ""
  }
}
```

Solution stack

Implementation

- Backend made with Java, Spring Boot for REST API
- Frontend made with Typescript, React + Redux for state management

Asynchronous interactions (optional)

- RabbitMQ for message queues
- Sentry for application monitoring

Testing tools

Junit for backend

Jest for frontend

Operations

- Makefile code build
- Jenkins CI/CD pipeline
- Docker

Thank you