Rakuten Games Interview

Chat Application

Instructions

Note: make sure you have 'pip' and 'virtualenv' installed and please use Python 3.7 and below to run the application. I opted for an older release for long-term support and risk management.

```
Initial installation: make install
To run test: make tests
To run application: make run
To run all commands at once : make all
```

Make sure to run the initial migration commands to update the database.

```
> flask db init
> flask db migrate --message 'initial database migration'
> flask db upgrade
```

Note: Make sure your Redis server is currently running before continuing.

Open the following url on your browser to view swagger documentation:

```
http://127.0.0.1:5000/
```

Authorization

Authorization header is in the following format:

```
Key: Authorization
Value: "token_generated_during_login"
```

Design Decisions

Python + Flask

Emir Ünlütürk (emunluturk@gmail.com)

I opted for using Python and Flask as the support for the frameworks I wished to use were significantly more supported. Using Flask as well gives the advantage of scalability and reduction in the dependencies required to build a ready-to-test application. Another benefit is a built-in unit testing framework and great resources for API deployment. Using Swagger allowed me to fully document the API so testers can easily navigate through the endpoints.

SQLAlchemy + SQLite

SQLAlchemy is an ORM for Python where developers may use as a wrapper to facilitate the communication between Python and databases. Multiple databases are currently supported and can be easily interchanged within the config file. I am currently using SQLite for ease of testing and deployment however Postgresql, MySQL, Oracle, MSSQL and other databases are also supported. The usage of SQLAlchemy allowed me to map the database objects to Python classes and gave me powerful filtering and query tools to retrieve chats, messages and users.

PubSub Approach using Redis

For a real-time chat application, I believe a PubSub approach to be one of the most effective ways of achieving almost instant messaging without the need for constant querying. Thanks to how PubSub works, once you subscribe to a topic you will constantly retrieve data from that event source without having to query it again. This allows many users to subscribe to a central topic where messages are sent/published. This design decision guarantees speed and reliability however as it is not persistent data I also decided to save chat history using the database to overcome this limitation.

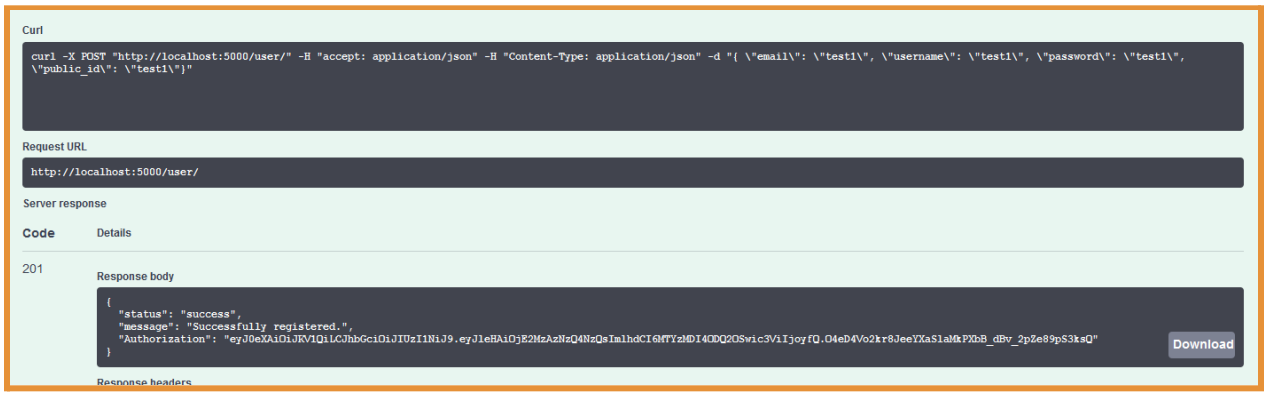
Usage Scenario

1. Create user with a name

User opens the mobile app. App shows a login/register screen. Once user clicks register and fills his information the app sends a request to the register endpoint.

```
POST "http://localhost:5000/user/"
{
  "email": "test@test.com",
```

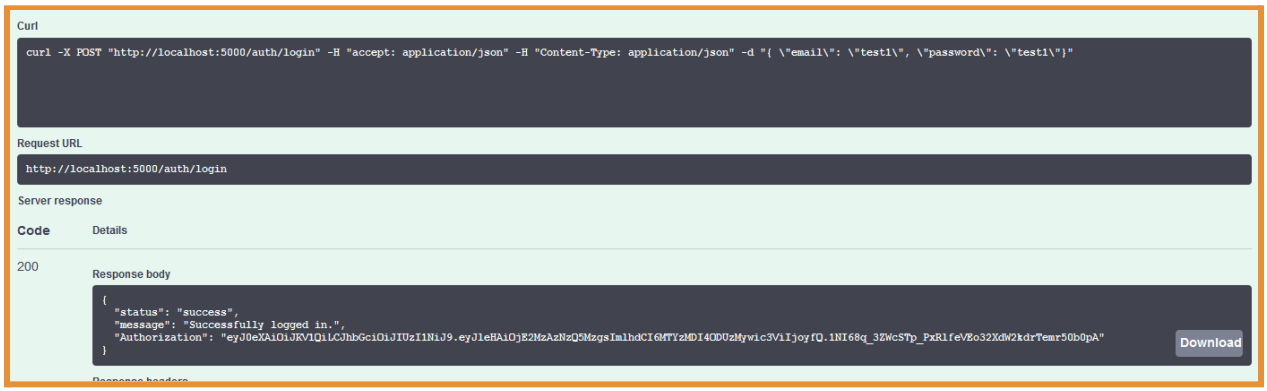
```
  "username": "test",
  "password": "test",
  "public_id": "test"
}
```



2. Log user in with a name

The mobile app can either log the user in through the authorization key in the response or return the user to the login screen. User may enter registered login info and click the login button. The response contains an authorization key where the mobile app will cache and use it as a header for further usage.

```
POST "http://localhost:5000/auth/login"
{
  "email": "test@test.com",
  "password": "test"
}
```



3. Get list of users in the chatroom

User clicks on a chatroom that he/she wishes to chat in. The list is given using the following endpoint.

```
GET "http://localhost:5000/chat/"
```

User clicks on the user list button to retrieve a list of users subscribed to the current chatroom.

```
GET "http://localhost:5000/chat/{chatroom_name}/users"
```

4. Get history messages from the chatroom

Once the chatroom is loaded the previous messages within the chatroom will be loaded by the mobile app. Currently, this is configured to retrieve the previous 100 messages.

```
GET "http://localhost:5000/chat/{chatroom_name}/messagehistory"
```

5. Receive message sent by the user

The mobile app will be subscribed to the chatroom topic through an event stream which will provide constant data to the mobile application. New messages coming from clients can be shown by subscribing using the following endpoint. In the screenshot you may view the SENT MSG and RECEIVED MSG logs from the clients for view.

```
GET "http://localhost:5000/chat/{chatroom_name}/message"
```

```
127.0.0.1 - - [30/Aug/2021 04:58:12] "POST /chat/test/message HTTP/1.1" 404 -
[testchat1] RECEIVED MSG: [2021-08-30T04:58:15.856410] test1: test message 1


127.0.0.1 - - [30/Aug/2021 04:58:17] "GET /chat/testchat1/message HTTP/1.1" 200 -
[testchat1] SENT MSG: [04:58:15] test1: test message 1
127.0.0.1 - - [30/Aug/2021 04:58:17] "POST /chat/testchat1/message HTTP/1.1" 204 -
[testchat1] RECEIVED MSG: [2021-08-30T04:58:21.590709] test1: test message 2


[testchat1] SENT MSG: [04:58:21] test1: test message 2
127.0.0.1 - - [30/Aug/2021 04:58:21] "POST /chat/testchat1/message HTTP/1.1" 204 -
[testchat1] RECEIVED MSG: [2021-08-30T04:58:25.554054] test1: test message 3 rakutenn


[testchat1] SENT MSG: [04:58:25] test1: test message 3 rakutenn
127.0.0.1 - - [30/Aug/2021 04:58:25] "POST /chat/testchat1/message HTTP/1.1" 204 -
```

6. Broadcast received message to other users in the chatroom

The user may then continue to send his/her own message if interested in the discussion.

```
POST "http://localhost:5000/chat/{chatroom_name}/message"
{
  "message": "test message"
}
```

Future Considerations

Scalability

The application can be containerized and switched over to cloud services such as Google PubSub or AWS SNS. This also allows greater ability to control the messages within the topics and will allow the developers to develop better features for the application. Another benefit is the easy load-balancing deployment and quick globalization.

New features

Currently, only group messaging is supported however one-to-one private messaging can also quickly be implemented. Another feature would be to include non-text messaging such as audio/video and allowing data transfer such as transferring personal documents.

Security

Security is only implemented through an Authorization JWT token however multiple security levels can be added such as a BFF API Gateway, database-level security, and other authorization techniques such as OAuth2.