

СОДЕРЖАНИЕ

1 Введение	2
2 Описание задач практики	3
3 Исполненное индивидуальное задание	4
3.1 Изучение библиотек на Python	4
3.2 Авторизация и аутентификация	4
3.3 Комната ожидания	5
4 Заключение	9
5 Список использованных источников	10

1 Введение

Целью проведения преддипломной практики является подготовка к защите выпускной квалификационной работы (ВКР).

Задачи практики:

- закрепление знаний и умений, полученных студентами в течение всего времени обучения.
- приобретение практического опыта, необходимого для профессиональной деятельности;
- проверка возможностей самостоятельной работы будущего специалиста;
- приобретение навыков самостоятельной научно-исследовательской работы.
- сбор, систематизация, обобщение материалов для подготовки выпускной квалификационной работы;
- изучение специальной литературы по теме ВКР;
- проведение исследований по теме ВКР;
- разработка программных реализаций алгоритмов и модулей по теме ВКР.

2 Описание задач практики

Для прохождения преддипломной практики были поставлены следующие задачи:

- Изучить современные библиотеки на языке Python, позволяющие реализовать соединение между клиентом и сервером по протоколу WebSocket.
- Реализовать безопасную систему авторизации и аутентификации для подключения к бэкенду приложения по протоколу WebSocket. Система должна открывать подключение только при наличии у пользователя прав на подключение к данной запланированной видеоконференции.
- Реализовать новый функционал на бэкенде приложения, необходимый для работы комнат ожидания с помощью протокола WebSocket. Система должна предоставить возможность организатору запланированной встречи создать комнату ожидания, в которой пользователи будут находиться, пока организатор встречи не разрешит войти им в комнату текущей видеоконференции. В комнате ожидания должен быть доступен чат для пользователей, ожидающих подключения.

3 Исполненное индивидуальное задание

3.1 Изучение библиотек на Python

В ходе выполнения индивидуального задания на практику было изучено несколько библиотек, позволяющих установить соединение между сервером и клиентом по протоколу WebSocket. Бэкенд системы видеоконференций разрабатывается на фреймворке Flask. Поэтому важнейшим критерием при выборе библиотеки стала возможность библиотеки работать с данным фреймворком. Основными библиотеками, работающими с Flask являются Flask-Sock [1] и Flask-SocketIO [2].

Библиотека Flask-SocketIO предоставляет больше возможностей для реализации логики приложения посредством таких дополнительных функций, как возможность распределять клиентов по комнатам (rooms) и возможность добавлять дополнительные заголовки к HTTP-запросу, который клиент посылает при открытии соединения по протоколу WebSocket.

3.2 Авторизация и аутентификация

Поскольку протокол WebSocket не предоставляет никаких инструментов для обеспечения безопасной авторизации и аутентификации пользователей, было решено использовать JWT-токены. Перед запросом на открытие соединения по протоколу WebSocket клиент отправляет запрос по протоколу HTTP на получение токена, предоставляя серверу данные, необходимые для его получения. На сервере происходит проверка на то, имеет ли пользователь права на подключение к данной видеоконференции. Если пользователь имеет все необходимые права, то сервер отправляет ему токен, в котором содержится вся необходимая для подключения информация (Рисунок 1).

```
def get_encoded_socketio_jwt(user, room_name, exp_time, avatar, moderator=False, content_id=None, pass_code_id="-1"):  
    jwt_data = {  
        "context": {  
            "user": {  
                "name": user.name,  
                "email": user.email,  
                "id": user.sub,  
                "roles": user.roles,  
                "pass_code_id": pass_code_id,  
                "avatar": avatar  
            },  
            "group": user.sub  
        },  
        "sub": JWT_SOCKETIO_SUB,  
        "aud": "socketio",  
        "iss": JWT_SOCKETIO_ISS,  
        "room": room_name,  
        "content_id": content_id,  
        "moderator": moderator,  
        "exp": exp_time  
    }  
    return jwt.encode(jwt_data, JWT_SOCKETIO_SECRET, algorithm="HS256")
```

Рисунок 1 - Данные, содержащиеся в JWT-токене, необходимом для открытия соединения по протоколу WebSocket

Помимо токена сервер отправляет клиенту время действия токена и `content_id`. Поле `content_id` необходимо для того, чтобы на бэкенде системы можно было отличать две конференции с одинаковыми названиями, которые были созданы в разное время.

После получения ответа содержащего токен, время его действия и `content_id`, на клиенте токен помещается в cookie с названием «`socket_io_token_{content_id}`». При открытии соединения по протоколу WebSocket клиент отправляет HTTP-запрос с предложением перейти на другой протокол. Библиотека SocketIO предоставляет возможность передать в заголовках данного запроса дополнительную информацию. Таким образом, клиент добавляет в данный запрос заголовок, содержащий название конференции, в которой находится пользователь (если пользователь является модератором) или к которой пользователь хочет подключиться (если пользователь не является модератором). Вместе с данным запросом отправляются и все cookie, содержащиеся на клиенте, в том числе и cookie с токеном.

Получив запрос на смену протокола, сервер, с помощью названия видеоконференции, которое было получено из заголовков HTTP-запроса, получает `content_id` из базы данных, производя поиск по запущенным в данный момент видеоконференциям, так как в один момент времени может быть запущена только одна видеоконференция с таким названием. Так как значение `content_id` содержится в названии cookie, содержащего токен, то с помощью `content_id` сервер определяет тот токен, который ему необходимо проверить. Если токен проходит проверку, то сервер открывает соединение с клиентом по протоколу WebSocket. Добавление токена в cookie позволяет хранить одновременно несколько токенов, если пользователь одновременно находится в нескольких видеоконференциях, а также позволяет не запрашивать токен каждый раз, если пользователь по тем или иным причинам отключается от конференции и затем присоединяется снова.

3.3 Комната ожидания

После успешного подключения к серверу по протоколу WebSocket пользователь может подключиться к комнате ожидания. Если пользователь является модератором, то он попадает в видеоконференцию минуя комнату ожидания. В этом случае подключение по протоколу WebSocket открывается в момент, когда модератор откроет окно комнаты ожидания для того, чтобы впустить пользователей или написать сообщение в чат (Рисунок 2).

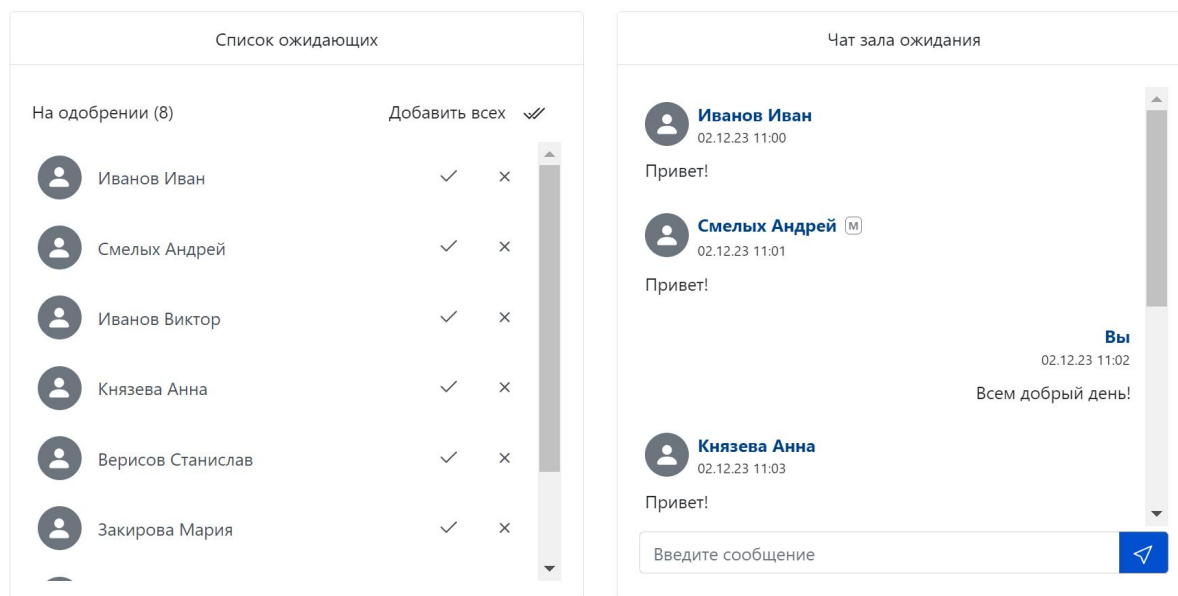


Рисунок 2 - Отображение комнаты ожидания для модератора конференции

Библиотека SocketIO позволяет на стороне сервера открывать дополнительные каналы, называемые в документации библиотеки «комнатами», с произвольным названием и добавлять в них пользователей, чтобы впоследствии отправлять сообщения только тем пользователям, которые состоят в конкретной комнате (Рисунок 3).

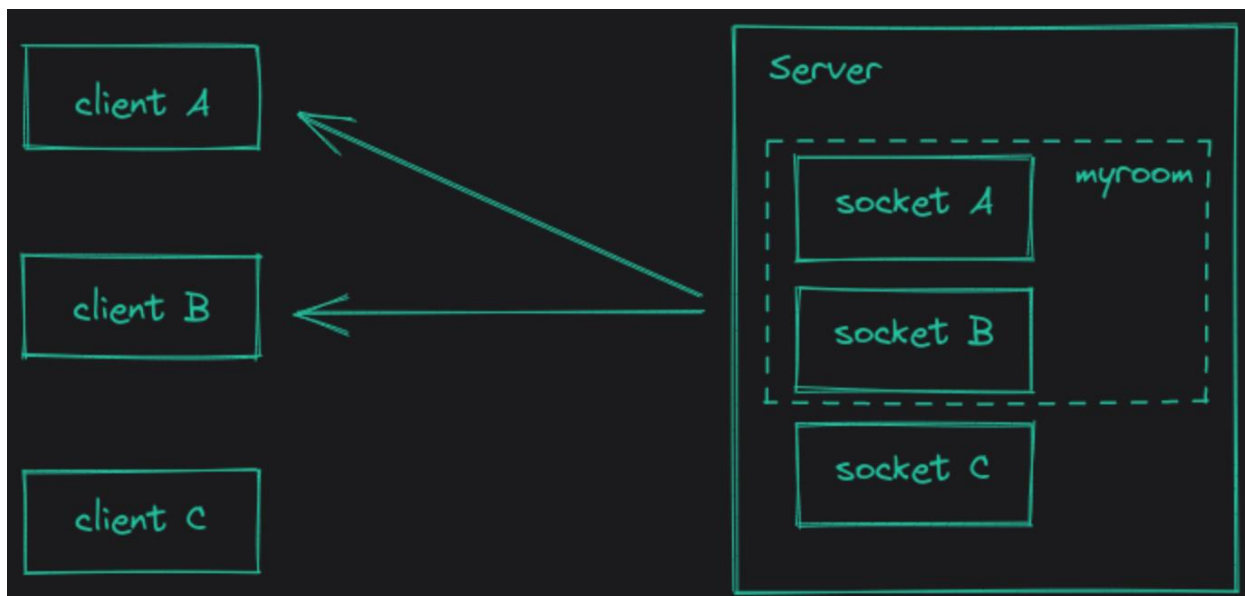
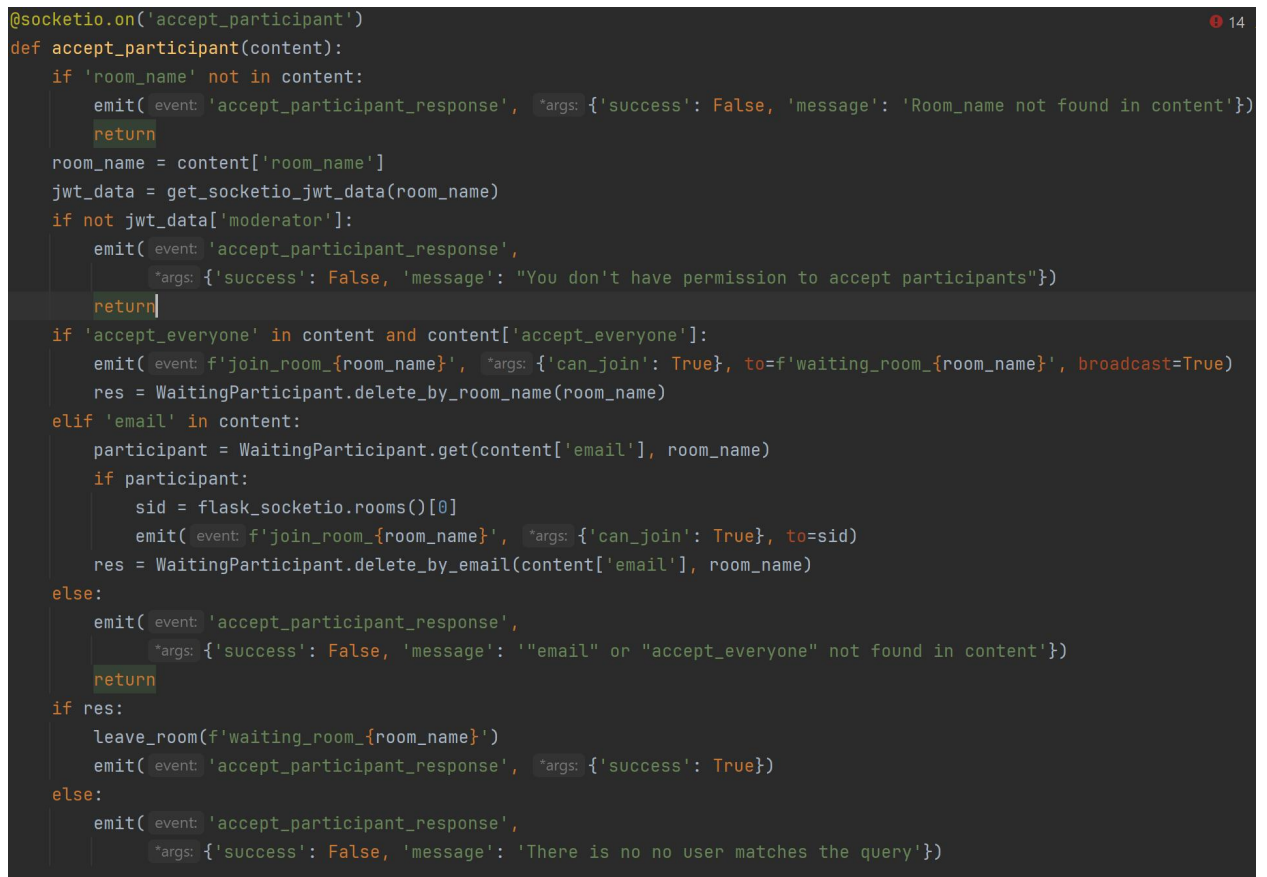


Рисунок 3 - Принцип работы комнат в библиотеке SocketIO.

При подключении к серверу срабатывает обработчик, который проверяет валидность токена и добавляет пользователя в комнату с названием «moderator_{название_конференции}», если он является модератором. Информация о том, является пользователь модератором, содержится в токене.

Когда обычный пользователь (не модератор) хочет подключиться к конференции, фронтенд системы вызывает событие «join_waiting_room» на сервере, отправляя при этом название конференции и email пользователя, желающего подключиться. Данное событие добавляет необходимую информацию о пользователе в таблицу «waiting_room» в базу данных и добавляет пользователя в комнату с названием «waiting_room_{название конференции}».

Когда модератор конференции решает впустить одного пользователя или всех пользователей в конференцию, вызывается событие «accept_participant» (Рисунок 4).



```
@socketio.on('accept_participant')
def accept_participant(content):
    if 'room_name' not in content:
        emit(event='accept_participant_response', *args: {'success': False, 'message': 'Room_name not found in content'})
        return
    room_name = content['room_name']
    jwt_data = get_socketio_jwt_data(room_name)
    if not jwt_data['moderator']:
        emit(event='accept_participant_response',
            *args: {'success': False, 'message': "You don't have permission to accept participants"})
        return
    if 'accept_everyone' in content and content['accept_everyone']:
        emit(event=f'join_room_{room_name}', *args: {'can_join': True}, to=f'waiting_room_{room_name}', broadcast=True)
        res = WaitingParticipant.delete_by_room_name(room_name)
    elif 'email' in content:
        participant = WaitingParticipant.get(content['email'], room_name)
        if participant:
            sid = flask_socketio.rooms()[0]
            emit(event=f'join_room_{room_name}', *args: {'can_join': True}, to=sid)
            res = WaitingParticipant.delete_by_email(content['email'], room_name)
        else:
            emit(event='accept_participant_response',
                *args: {'success': False, 'message': '"email" or "accept_everyone" not found in content'})
            return
    if res:
        leave_room(f'waiting_room_{room_name}')
        emit(event='accept_participant_response', *args: {'success': True})
    else:
        emit(event='accept_participant_response',
            *args: {'success': False, 'message': 'There is no no user matches the query'})
```

Рисунок 4 - Событие «accept_participant»

При этом фронтенд системы отправляет название конференции, информацию о том, надо ли впускать всех пользователей в комнате ожидания и email пользователя, которого надо впустить, если надо впустить только одного пользователя. Если модератор решил впустить всех пользователей, то сервер отправляет сообщение всем пользователям, находящимся в комнате «waiting_room_{название конференции}», что им разрешено присоединиться к конференции. Если же модератор решил впустить только одного пользователя, то с помощью email данного пользователя из базы данных получается информация о sid (идентификаторе сеанса) пользователя. С помощью идентификатора сеанса сервер отправляет разрешение на вступление в конференцию только одному пользователю.

Данное событие отправляет разрешение на вступление пользователям, которых впустил модератор, не требуя при этом запроса на вступление в конференцию от этих пользователей. Такое решение стало возможным только благодаря технологии WebSocket, так как по протоколу HTTP сервер не может отправлять информацию клиентам, не получив от них перед этим запроса.

Всем пользователям, находящимся в комнате ожидания, а также модератору должен быть доступен чат. В работе чата также реализован механизм отправки сообщения сервером пользователю без получения от него запроса. Когда на сервер приходит сообщение от пользователя, сервер отправляет данное сообщение всем пользователям, которые состоят в комнате «chat_{название конференции}», в том числе и модератору конференции.

4 Заключение

В ходе прохождения практики были изучены библиотеки на языке Python, работающие с фреймворком Flask и позволяющие обеспечить стабильное подключение по протоколу WebSocket. С помощью библиотеки Flask-SocketIO на бекенде системы видеоконференций были реализованы механизмы авторизации и аутентификации, использующие JWT-токены. Помимо данной функции, обеспечивающей безопасность подключения по протоколу WebSocket, был реализован функционал, позволяющий организатору конференции включить для конференции комнату ожидания с чатом для пользователей, желающих подключиться к конференции.

5 Список использованных источников

1. Flask-Sock // Readthedocs URL: <https://flask-sock.readthedocs.io/en/latest/> (дата обращения: 15.03.2024).
2. Flask-SocketIO // Readthedocs URL: <https://flask-socketio.readthedocs.io/en/latest/> (дата обращения: 15.03.2024).