

ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

Ciobanu Stanislav

Отчет

по индивидуальной работе по дисциплине
„ПРОГРАММИРОВАНИЕ В PYTHON”

Руководители: Плешка Наталья (Лектор), Felicia Schioru (Лаборант)

Автор: Ciobanu Stanislav

Кишинев, 2024

GitHub: <https://github.com/Rengeka/PythonIndividualWork.git>

Содержание:

1. Задача
2. Краткое описание принципа работы
3. Реализация сервера
4. Реализация клиента
5. Конечный результат
6. Дальнейшее развитие проекта
7. Выводы

(1) Задача:

Создать приложение - чат, в котором различные пользователи могли бы обмениваться сообщениями.

Реализовать взаимодействие клиент-сервер при помощи web-сокетов.

Создать базу данных, в которой будут храниться данные пользователей, чаты и сообщения.

(2) Краткое описание принципа работы:

Для реализации данного проекта потребуется создать два проекта. Один будет отвечать за работу сервера, второй за работу клиента. Для обеспечения безопасности, клиент будет посылать запросы на сервер, где они будут обрабатываться и, если это необходимо, будут отправляться

ответы с необходимыми данными. Таким образом доступ к базе данных будет только у сервера.

Для простоты, база данных будет реализована в виде json-файла.

(3) Реализация сервера:

Создадим json – базу данных со значениями – “users” и “chats”.

Каждый пользователь обладает значениями “ID”, “username”, “chats”, “password”, а каждый чат значениями “ID”, “users”, “messages”, где каждое из сообщений это пара значений “user”, “text”.

Устанавливаем библиотеки flask и flask_socketio и импортируем их в проект.

```
from flask import Flask, request
from flask_socketio import SocketIO, emit
import json
```

Создаём и запускаем сервер.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = '1234567890'
socketio = SocketIO(app)
```

```
if __name__ == '__main__':  
    socketio.run(app, host='0.0.0.0', port=5000, allow_unsafe_werkzeug=True)
```

Создаём методы для получения данных из базы данных.

```
usage  
✓ def LoadUsers(filename):  
    with open(filename, 'r') as file:  
        data = json.load(file)  
    return data['users']  
  
1 usage  
> def LoadChats(filename):...  
  
2 usages  
> def LoadDatabase(filename):...
```

Создаём глобальный список для отслеживания активных сессий.

```
activeSessions = []
```

Создаём функционал для принятия и отправки запросов клиенту.

```

13  @socketio.on('GetMessage')
14  > def GetMessage(data):...
46
    1 usage
47  > def SendMessageUpdate(session, data):...
50
51  > def SendUpdate(session, data):...
54
55  @socketio.on('ChatRequest')
56  > def ChatRequest(data):...
120
    1 usage
121 > def SendChatUpdate(session, data):...
124
125 @socketio.on('LoginVerification')
126 > def LoginRequest(data):...

```

(4) Реализация сервера:

Разделим клиент на два модуля. В одном будем хранить всё взаимодействие с UI, а в другом (Client) непосредственно методы для реализации сетевого соединения.

Для UI клиента воспользуемся библиотекой для создания desktop-приложений kivy.

```

17 > class LoginPage(Screen):...
67
1 usage
68 > class ChatSelectionPage(Screen):...
199
1 usage
200 class MyApp(MDApp):
201     def build(self):
202         chat_selection_page = ChatSelectionPage(name="chat_selection")
203         login_page = LoginPage(chat_selection_page)
204
205         client.SetLoginPage(login_page)
206         client.SetChatSelectionPage(chat_selection_page)
207
208         screen_manager = ScreenManager()
209         screen_manager.add_widget(login_page)
210         screen_manager.add_widget(chat_selection_page)
211
212         return screen_manager
213
214
215 if __name__ == "__main__":
216     app = MyApp()
217
218     app.run()
219

```

Создадим классы LoginPage и ChatSelectionPage. В этих классах при помощи библиотеки реализуем необходимый UI, который при взаимодействии с пользователем, будет обращаться к модулю Client и взаимодействовать с сервером.

Создадим необходимые методы, которые будут отправлять запросы на сервер и, получив данные, подгружать в UI.

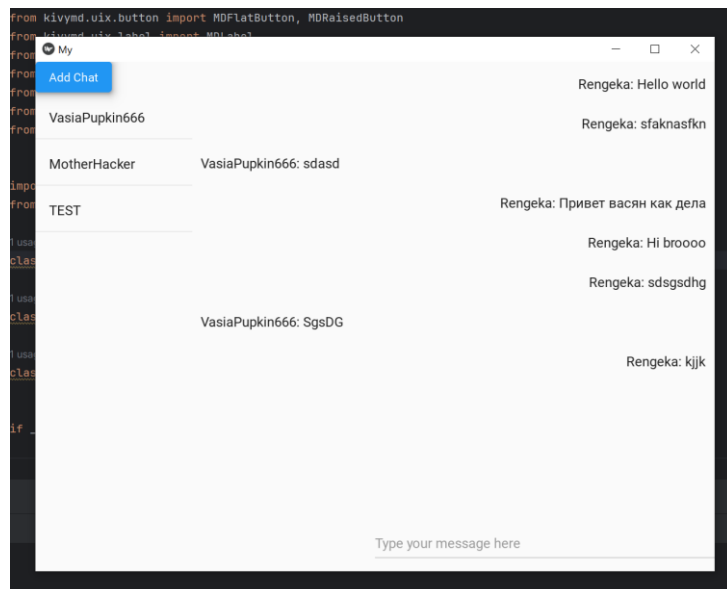
```

1 import socketio
2 from kivy.clock import Clock
3
4 sio = socketio.Client()
5
6 > def SetLoginPage(value):...
7
8 > def SetChatSelectionPage(value):...
9
10 @sio.event
11 def TEST(data):
12     print(data)
13
14 > ...
15
16 @sio.event
17 def GetChats(data):...
18
19 > ...
20
21 @sio.event
22 def GetUpdate(data):...
23
24 > ...
25
26 @sio.event
27 def StartChat(user, login):...
28
29 def LoginRequest(login, password):
30     print("LOGIN REQUEST")
31     sio.emit(event='LoginVerification', data={'login': login, 'password': password, 'sessionID': sio.sid})
32
33 @sio.event
34 def VerifyLogin(response):
35     print(response)
36     LoginPage.check_login_response_wrapper(response)
37
38 > def SendMessage(ID, text, user):...
39
40 @sio.event()
41 > def GetMessages(data):...
42
43
44 sio.connect('http://127.0.0.1:5000')

```

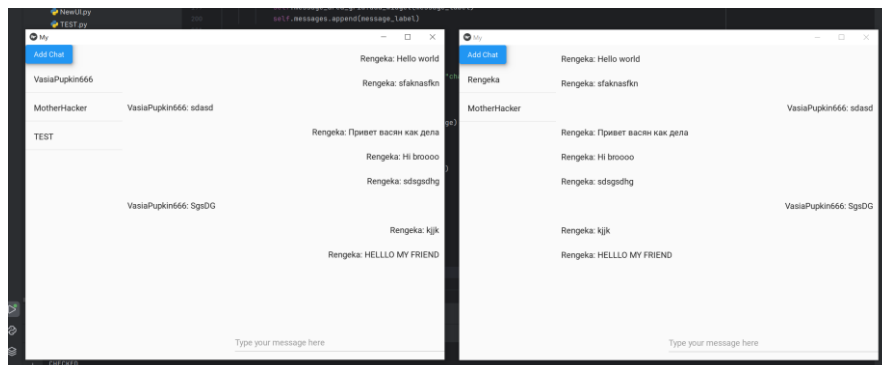
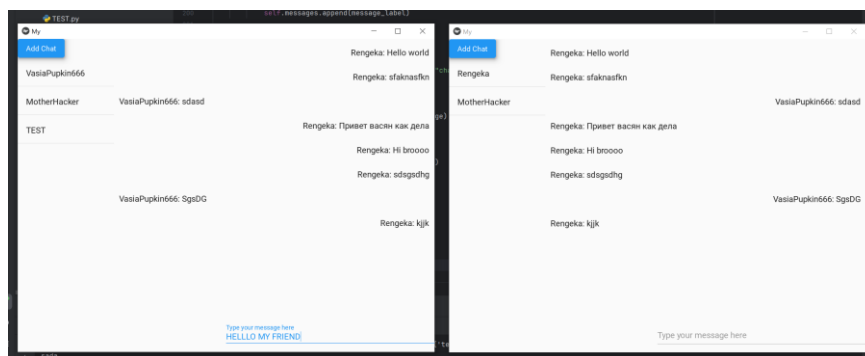
Результат работы клиента:

The screenshot shows a Kivy application window titled "My". Inside the window, there is a login form with three main components: a "Username" label followed by a text input field, a "Password" label followed by a text input field, and a "Login" button. The background of the window is a light gray, and the text labels are in a dark color. The window has standard OS window controls (minimize, maximize, close) in the top right corner.



(5) Конечный результат:

Конечная реализация проекта позволяет пользователям в реальном времени обмениваться сообщениями. Благодаря тому, что сервер хранит у себя адреса активных сессий (клиентов), он может посылать обновления всем необходимым пользователям (Даже если один пользователь авторизован в двух клиентах).



(6) Дальнейшее развитие проекта:

По причине неопытности и первоначального незнания технологий, которые тут используются, в базовой проектировке проекта было допущено несколько неприятных ошибок, что мешают дальнейшей комфортной работе над проектом. Поэтому в качестве первого шага по развитию проекта, я бы провёл рефакторинг кода, удаление ненужных частей и более грамотно продумал бы дальнейшую архитектуру.

Далее я бы реализовал функционал по созданию новых чатов и пользователей т.к в текущем виде это возможно только при помощи прямого редактирования базы данных.

Так же было бы интересно реализовать возможность отправки аудио сообщений и изображений, а так-же некоторые косметические изменения интерфейса (Например аватары пользователей).

(7) Выводы:

Не смотря на ряд трудностей и ошибок, с которыми я столкнулся, я доволен сделанной работой. Я изучил новые технологии и библиотеки (kivy, flask, web-сокеты), создал функциональный проект, гораздо лучше познакомился с работой с json-файлами и словарями и научился обеспечивать двустороннее взаимодействие сервера с клиентами.