**Веб-програмування**

## **Лабораторна робота №9**

«Створення real-time додатку»

*Виконав Заяць Ілля, група ІПЗ-41*

**Мета:**

Метою даної лабораторної роботи є отримання практичних навичок розробки веб-додатків з можливістю обміну даними в режимі реального часу з використанням WebSocket протоколу.

Завдання:

1. Розробити real-time веб-додаток з використанням WebSocket комунікації

2. Реалізувати відображення даних в реальному часі на клієнтській стороні

3. Забезпечити стабільність та масштабованість real-time комунікацій

**Хід роботи**

1. Опрацювати теоретичні відомості.

2. Встановити необхідне програмне забезпечення.

3. Розробити архітектуру real-time додатку.

4. Реалізувати серверну частину з підтримкою WebSocket.

5. Розробити клієнтський інтерфейс для відображення даних в реальному часі.

6. У якості звіту в LMS Moodle вставити посилання на репозиторій GitHub.

7. Під час захисту звіту продемонструвати готову реалізацію завдання.

## *Структура проєкту:*

my-realtime-app/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py # Основний файл сервера FastAPI

│ ├── websocket\_manager.py # Клас для управління WebSocket з'єднаннями

│ └── static/ # Статичні файли (CSS, JS, картинки)

| ├── └── index.html #Фронтенд для взаємодії

├── requirements.txt # Список залежностей для проекту

├── Dockerfile # Dockerfile для контейнеризації проекту

└── README.md # Документація проекту

## *Код (приведений лише код нових файлів чи змінений код старих):*

**app/main.py - Основний файл серверу, що включає налаштування WebSocket:**

# app/main.py

from fastapi import FastAPI, WebSocket, WebSocketDisconnect

from typing import List

from .websocket\_manager import WebSocketManager

app = FastAPI()

# Створюємо екземпляр WebSocketManager для управління підключеннями

manager = WebSocketManager()

@app.websocket("/ws")

async def websocket\_endpoint(websocket: WebSocket):

await manager.connect(websocket)

try:

while True:

data = await websocket.receive\_text()

await manager.broadcast(f"Message text: {data}")

except WebSocketDisconnect:

manager.disconnect(websocket)

await manager.broadcast("A user has disconnected")

**app/websocket\_manager.py - Клас для управління з'єднаннями WebSocket:**

# app/websocket\_manager.py

from fastapi import WebSocket

from typing import List

class WebSocketManager:

def \_\_init\_\_(self):

self.active\_connections: List[WebSocket] = []

async def connect(self, websocket: WebSocket):

await websocket.accept()

self.active\_connections.append(websocket)

def disconnect(self, websocket: WebSocket):

self.active\_connections.remove(websocket)

async def send\_personal\_message(self, message: str, websocket: WebSocket):

await websocket.send\_text(message)

async def broadcast(self, message: str):

for connection in self.active\_connections:

await connection.send\_text(message)

**app/static/index.html - Простий фронтенд для взаємодії через WebSocket:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>WebSocket Chat</title>

<style>

body {

font-family: Arial, sans-serif;

}

#messages {

border: 1px solid #ccc;

margin-bottom: 20px;

padding: 10px;

height: 200px;

overflow-y: scroll;

}

</style>

</head>

<body>

<h1>Real-Time Chat</h1>

<div id="messages"></div>

<input id="messageInput" type="text" placeholder="Type a message">

<button onclick="sendMessage()">Send</button>

<script>

const socket = new WebSocket("ws://localhost:8000/ws");

socket.onopen = () => {

console.log("Connected to WebSocket");

};

socket.onmessage = (event) => {

const messagesDiv = document.getElementById("messages");

const newMessage = document.createElement("div");

newMessage.textContent = event.data;

messagesDiv.appendChild(newMessage);

};

socket.onclose = () => {

console.log("Disconnected from WebSocket");

};

function sendMessage() {

const messageInput = document.getElementById("messageInput");

const message = messageInput.value;

socket.send(message);

messageInput.value = "";

}

</script>

</body>

</html>

**Dockerfile - Dockerfile для контейнеризації проекту:**

# Dockerfile

# Використовуємо офіційний Python образ

FROM python:3.9

# Встановлюємо робочий каталог

WORKDIR /app

# Копіюємо всі файли в контейнер

COPY . /app

# Встановлюємо залежності

RUN pip install --no-cache-dir -r requirements.txt

# Відкриваємо порт для веб-сервера

EXPOSE 8000

# Запускаємо сервер FastAPI

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]

**requirements.txt - Список залежностей для проекту:**

fastapi

uvicorn

websockets

redis

**README.md - Оновлений файл документації:**

# Real-Time WebSocket Chat Application

Цей проект реалізує веб-додаток для обміну повідомленнями в реальному часі з використанням WebSocket. Сервер побудований на FastAPI, а клієнт взаємодіє з ним через WebSocket.

## Структура проекту

- \*\*app/main.py\*\* - Основний серверний файл FastAPI.

- \*\*app/websocket\_manager.py\*\* - Менеджер для керування WebSocket підключеннями.

- \*\*app/static/index.html\*\* - Фронтенд для взаємодії з сервером через WebSocket.

- \*\*Dockerfile\*\* - Для створення Docker-контейнера.

- \*\*requirements.txt\*\* - Перелік необхідних бібліотек для проекту.

## Як запустити проект

1. Клонуйте репозиторій:

```bash

git clone https://github.com/db-user/vebprogLR9.git

cd real-time-chat

```

2. Створіть і активуйте віртуальне середовище:

```

python -m venv venv

source venv/bin/activate # для Linux/macOS

venv\Scripts\activate # для Windows

```

3. Встановіть залежності:

```

pip install -r requirements.txt

```

4. Запустіть сервер:

```

uvicorn app.main:app --reload

```

5. Відкрийте app/static/index.html в браузері, щоб почати використовувати чат.

## Документація

\*\*Основна архітектура:\*\*

Використовується WebSocket для двостороннього зв'язку між сервером і клієнтом.

Клієнт підключається до серверу через ендпоінт /ws.

\*\*Масштабованість:\*\*

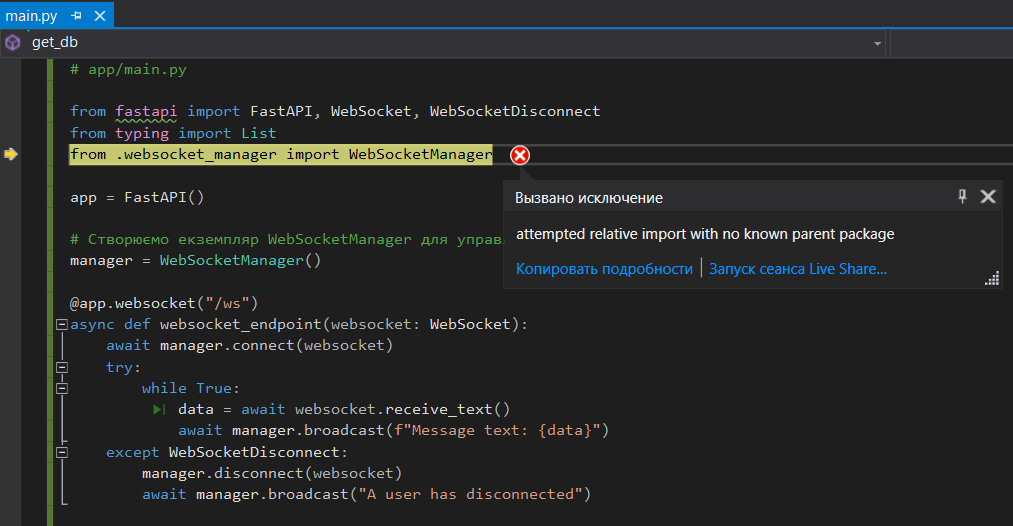
Можна масштабувати сервер за допомогою Redis для обміну повідомленнями між кількома інстансами.

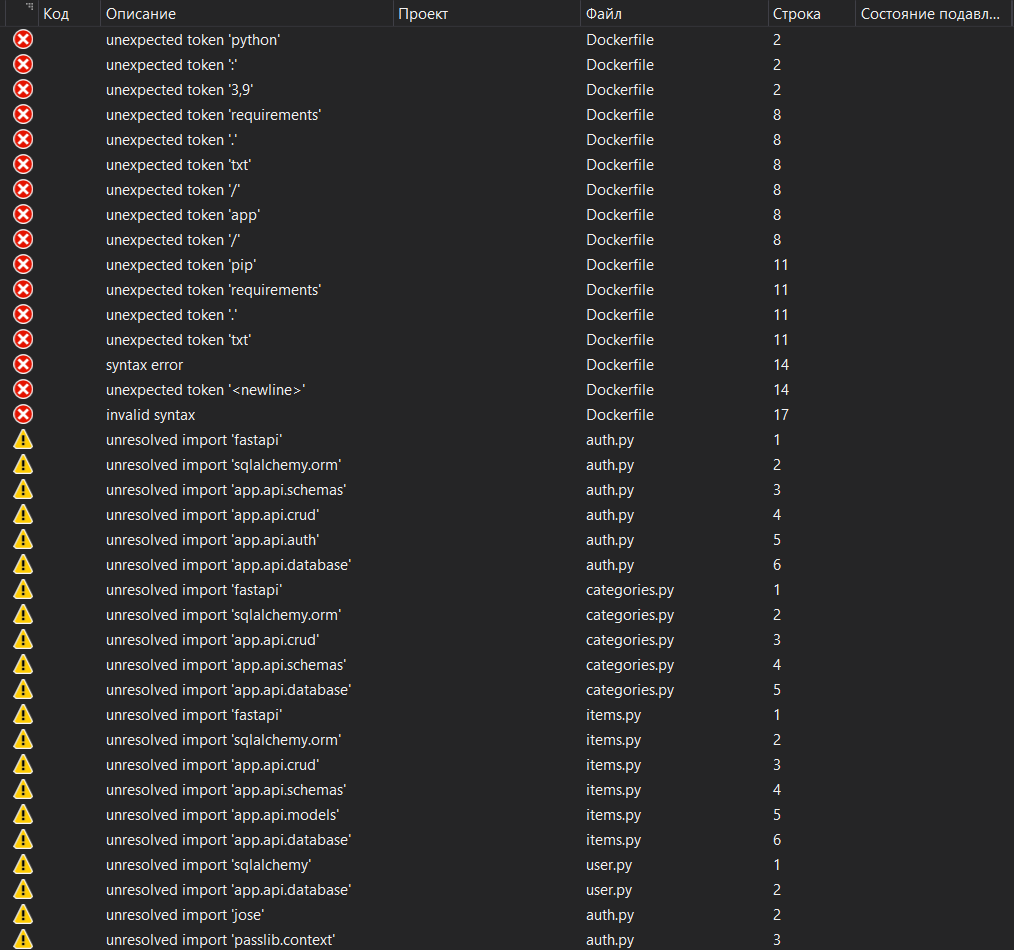
## Ліцензія

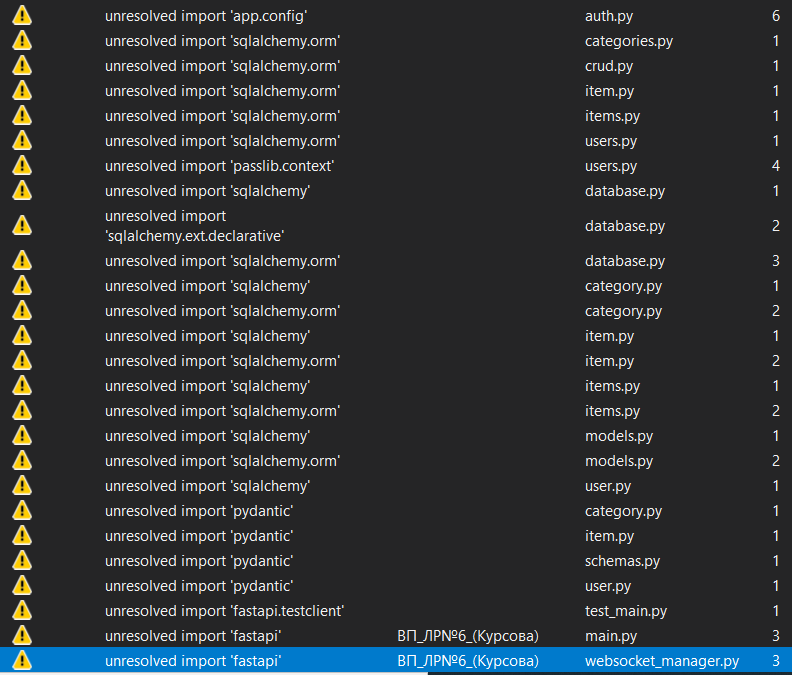
MIT License

## *Результат виведення::*

Оскільки я не виправив помилку в коді, то програма також не запускається коректно, але сам код, по ідеї, правильний. До того ж, до старих помилок, судячи зі всього, додалися нові.







**Висновки:** це доповнення значно покращує попередні версії, оскільки інтегрує WebSocket для реального часу, що дозволяє користувачам отримувати і надсилати повідомлення миттєво. Завдяки використанню FastAPI, сервіс стає високопродуктивним і масштабованим, що дозволяє легко додавати нові функції та підтримувати підключення великої кількості клієнтів. Крім того, додано використання Docker для контейнеризації, що полегшує розгортання та забезпечує відтворюваність середовища. Впровадження WebSocket-менеджера дозволяє ефективно керувати підключеннями. Це все разом дає більшу гнучкість, зручність та стабільність, порівняно з попередніми версіями. Через невиправлену помилку в програмі вона не працює коректно, але це проблема мого комп’ютера, на якому неправильно працюють деякі компоненти в програмі.