

WebSockets Project

Introduction

This document outlines the implementation of a WebSocket-based client-server application demonstrating real-time, full-duplex communication between a web browser client and a Python server.

WebSocket

WebSocket is a protocol that enables full-duplex, bidirectional communication between a client and a server over a single TCP connection. Unlike traditional HTTP, which follows a request-response model, WebSockets allow for:

- Real-time data transfer
- Lower latency
- Efficient communication with less overhead
- Simultaneous sending and receiving of data

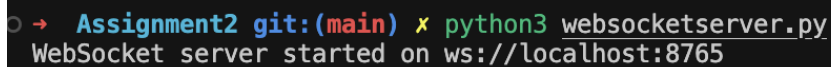
These features make WebSockets ideal for applications requiring live updates or real-time functionality.

Project Components

1. Python WebSocket Server
2. HTML/JavaScript WebSocket Client with User Interface

Project Flow

1. **Server Initialization**
 - The Python server starts and listens for WebSocket connections.



```
➤ → Assignment2 git:(main) ✖ python3 websocketserver.py
WebSocket server started on ws://localhost:8765
```

2. **Client Connection**
 - The HTML/JS client establishes a WebSocket connection to the server.

WebSocket Client

WebSocket connection established

Send One Message

Start 10,000 Messages

3. Message Sending

- The client can send either a single message or 10,000 messages in rapid succession.

WebSocket Client

WebSocket connection established
Sent: Hello there
Received: Hello there - 626
Sent: How are you?
Received: How are you? - 207

Send One Message

Start 10,000 Messages

4. Server Processing

- The server receives each message, appends a random number, and sends it back.

5. Client Reception

- The client receives the modified messages and displays them in the UI.

WebSocket Client

Received: Hello Server 9995 - 389

Sent: Hello Server 9996

Received: Hello Server 9996 - 360

Sent: Hello Server 9997

Received: Hello Server 9997 - 323

Sent: Hello Server 9998

Received: Hello Server 9998 - 900

Sent: Hello Server 9999

Received: Hello Server 9999 - 319

Send One Message

Start 10,000 Messages

Results

- The WebSocket connection successfully handled both single and multiple message scenarios.
- All 10,000 messages were processed without any losses, demonstrating the efficiency of WebSockets for high-frequency communication.
- The server effectively modified each message by appending a random number.
- Real-time updates were observed in the client UI, showcasing the low-latency nature of WebSockets.

Conclusion

This project successfully demonstrates the capabilities of WebSockets for real-time, bidirectional communication. The implementation showcases:

1. Efficient handling of both sporadic and high-frequency message exchanges
2. Low-latency communication between client and server
3. Seamless integration of WebSockets in a web-based user interface
4. Scalability potential for handling multiple clients and complex server-side processing

The project highlights the advantages of WebSockets over traditional HTTP communication methods for real-time applications. Future improvements could include implementing secure WebSocket connections (WSS) or scaling the server to handle a larger number of concurrent clients.