TrendStream: Real-time Trend Tracker

This exercise demonstrates how WebSockets enable real-time applications by creating a practical, engaging system that processes continuous data streams - a fundamental concept in modern web and application development.

Introduction

WebSockets provide a persistent, bidirectional communication channel between clients and servers. Unlike traditional HTTP, which follows a request-response pattern (like sending a letter and waiting for a reply), WebSockets maintain an open connection (like having a phone call) allowing for instant data transmission in both directions.

This exercise will help you understand how WebSockets enable real-time applications by building a trending topic tracker entirely in Python.

What Are WebSockets?

Traditional HTTP communication is like sending text messages - you send a request, wait for a response, and repeat the process for new information. This creates delays and requires constant reconnecting.

WebSockets, however, are like an ongoing phone call:

- Connection stays open after being established
- Data can flow in either direction at any time
- Updates are received instantly without polling
- Significantly reduced overhead for real-time applications

Task: Build a Trending Topics Tracker

Create a Python-based system with two components:

1. A WebSocket server that broadcasts trending topic data
2. A terminal client that connects to the server and displays filtered trends

Why This Requires WebSockets

This project demonstrates WebSockets' value because:

- Trends update frequently in real-time
- The server pushes data without being repeatedly asked
- Multiple clients can receive the same broadcast
- The persistent connection allows instant updates

Requirements

Server Component

Create a Python WebSocket server that:

- Generates trending topics across multiple categories (Music, Games, Fashion, Celebrities, Memes, etc.)
- Updates trend scores randomly every few seconds
- Broadcasts the complete dataset to all connected clients
- Runs continuously, maintaining connections with clients

Client Component

Create a Python terminal client that:

- Connects to the WebSocket server
- Allows users to subscribe/unsubscribe to categories
- Filters incoming data based on subscriptions
- Displays only the selected categories' trends
- Shows simple ASCII animations for trending changes
- Maintains a clean, readable terminal interface

Example User Experience

```
=== TRENDSTREAM CONNECTED ===
Categories: [M]usic, [G]ames, [F]ashion, [C]elebs, [T]ech, [A]ll, [Q]uit

>> M
Subscribed to Music! Watching trends...

🔥 #Taylor89Tour: 2,453 ⬆️
📈 #NewJeansComeback: 1,289 ⬆️
⚡ #GRAMMYSnub: 865 ⬇️

>> T
Now watching Music, Tech

🔥 #Taylor89Tour: 2,512 ⬆️
💻 #AI_Fashion: 1,723 ⬆️
📈 #NewJeansComeback: 1,302 ⬆️
📱 #iPhone17Leak: 945 ⬇️
```

Implementation Guide

Server Side

- Use Python's `websockets` library
- Create an asynchronous server that accepts connections
- Generate random trend data and scores
- Periodically update trend scores
- Broadcast updates to all connected clients
- Format data as JSON for easy parsing

Client Side

- Use Python's `websockets` library for connection
- Create a simple terminal UI for category selection
- Store user's category subscriptions
- Filter incoming JSON data based on subscriptions
- Display filtered trends with simple formatting
- Show changes in trend popularity (up/down arrows)

Learning Objectives

Through this exercise, you will learn:

- How WebSockets differ from traditional HTTP

- Implementing bidirectional communication in Python
- Managing real-time data streams
- Client-side filtering of broadcast data
- Basic terminal UI techniques

Bonus Features

If time allows, add these enhancements:

- Color coding in the terminal output
- Sorting trends by popularity
- "Hot trend" alerts for rapid increases
- Simple charts using ASCII art
- Historical tracking of trend changes

Reflection Questions

After completing the implementation, answer these questions: 1. How does the WebSocket approach improve the user experience compared to repeated HTTP requests? 2. What challenges did you encounter when working with continuous data streams? 3. How might this system be extended to handle thousands of concurrent users? 4. What other applications could benefit from WebSocket technology? 5. How does the client-side filtering approach affect server performance?