Explanations > WebSockets

WebSockets

Websockets are a protocol for two-way, persistent communication between a client and server. This is different from HTTP, which uses a request/response model where the client sends a request and the server responds. With websockets, either party can send messages at any time, and the other party can respond.

This allows for different applications to be built, including things like chat apps, live-updating dashboards, and real-time collaborative tools, which would require constant polling of the server for updates with HTTP.

In FastHTML, you can create a websocket route using the `@app.ws` decorator. This decorator takes a route path, and optional `conn` and `disconn` parameters representing the `on_connect` and `on_disconnect` callbacks in websockets, respectively. The function decorated by `@app.ws` is the main function that is called when a message is received.

Here's an example of a basic websocket route:

```
@app.ws('/ws', conn=on_conn, disconn=on_disconn)
async def on_message(msg:str, send):
    await send(Div('Hello ' + msg, id='notifications'))
    await send(Div('Goodbye ' + msg, id='notifications'))
```

The `on_message` function is the main function that is called when a message is received and can be named however you like. Similar to standard routes, the arguments to `on_message` are automatically parsed from the websocket payload for you, so you don't need to manually parse the message content. However, certain argument names are reserved for special purposes. Here are the most important ones:

- `send` is a function that can be used to send text data to the client.
- `data` is a dictionary containing the data sent by the client.
- `ws` is a reference to the websocket object.

For example, we can send a message to the client that just connected like this:

```
async def on_conn(send):
    await send(Div('Hello, world!'))
```

Or if we receive a message from the client, we can send a message back to them:

```
@app.ws('/ws', conn=on_conn, disconn=on_disconn)
async def on_message(msg:str, send):
    await send(Div('You said: ' + msg, id='notifications'))
    # or...
    return Div('You said: ' + msg, id='notifications')
```

On the client side, we can use HTMX's websocket extension to open a websocket connection and send/receive messages. For example:

```
from fasthtml.common import *

app = FastHTML(ws_hdr=True)

@app.get('/')
def home():
    cts = Div(
        Div(id='notifications'),
        Form(Input(id='msg'), id='form', ws_send=True),
        hx_ext='ws', ws_connect='/ws')
    return Titled('Websocket Test', cts)
```

This will create a websocket connection to the server on route `/ws`, and send any form submissions to the server via the websocket. The server will then respond by sending a message back to the client. The client will then update the message div with the message from the server using Out of Band Swaps, which means that the content is swapped with the same id without reloading the page.

Note

Make sure you set `ws_hdr=True` when creating your `FastHTML` object if you want to use websockets so the extension is loaded.

Putting it all together, the code for the client and server should look like this:

```
from fasthtml.common import *

app = FastHTML(ws_hdr=True)
rt = app.route

@rt('/')
def get():
    cts = Div(
        Div(id='notifications'),
        Form(Input(id='msg'), id='form', ws_send=True),
        hx_ext='ws', ws_connect='/ws')
    return Titled('Websocket Test', cts)

@app.ws('/ws')
async def ws(msg:str, send):
    await send(Div('Hello ' + msg, id='notifications'))

serve()
```

This is a fairly simple example and could be done just as easily with standard HTTP requests, but it illustrates the basic idea of how websockets work. Let's look at a more complex example next.

Real-Time Chat App

Let’s put our new websocket knowledge to use by building a simple chat app. We will create a chat app where multiple users can send and receive messages in real time.

Let’s start by defining the app and the home page:

```
from fasthtml.common import *

app = FastHTML(ws_hdr=True)
rt = app.route

msgs = []
@rt('/')
def home(): return Div(
    Div(Ul(*[Li(m) for m in msgs], id='msg-list')),
    Form(Input(id='msg'), id='form', ws_send=True),
    hx_ext='ws', ws_connect='/ws')
```

Now, let’s handle the websocket connection. We’ll add a new route for this along with an `on_conn` and `on_disconn` function to keep track of the users currently connected to the websocket. Finally, we will handle the logic for sending messages to all connected users.

```
users = {}
def on_conn(ws, send): users[str(id(ws))] = send
def on_disconn(ws): users.pop(str(id(ws)), None)

@app.ws('/ws', conn=on_conn, disconn=on_disconn)
async def ws(msg:str):
    msgs.append(msg)
    # Use associated `send` function to send message to each user
    for u in users.values(): await u(Ul(*[Li(m) for m in msgs], id='msg-list'))

serve()
```

We can now run this app with `python chat_ws.py` and open multiple browser tabs to `http://localhost:5001`. You should be able to send messages in one tab and see them appear in the other tabs.

A Work in Progress

This page (and Websocket support in FastHTML) is a work in progress. Questions, PRs, and feedback are welcome!

Report an issue