William Sanchez

SER401-Online

October 30th, 2022

**Websocket Basics**

Websockets is a communication protocol likt HTTP. It allows complete communication over a single TCP connection. It is used when you want your front-end to be notified when there is a change in the backend server.

Requirements: [Channels.](https://pypi.org/project/django-channels/)

To install channels type in **pip install channels**.

When done go into **settings.py** and under **INSTALLED\_APPS** type in **‘channels’** at the very top.

In order to use the ASGI/Channels development server we have to make some changes starting with the **asgi.py** file.

Text

Description automatically generated

In the code above, we defined handlers for different protocols inside the **ProtocolTypeRouter** object. It routes the incoming requests to different handlers depending on the type of request.

To handle ***HTTP/HTTPS*** requests, Django provides a handler method called **get\_asgi\_application()** which basically says “let the HTTP/HTTPS requests be handled by Django views like they were handled before”.

In the code above, you can see that the URL pattern path that is passed to the URLRouter is **myproject.urls.websocket\_urlpatterns**. We will define these URL patterns in a while.

Setting up this **asgi.py** file gives **channels** a pointer to the root application, which can then integrate itself in Django and take control of the **runserver** command.

So we need to do two things.

We need to define some URL patterns for our WebSocket connections. Channels will consult these URL patterns and then decide what to do with the request.

We need to write the logic for what needs to be done with the request after URL patterns are resolved. For normal HTTP requests, the logic is usually written in **views**. For WebSocket requests, the logic is written in **Consumers**.

Lets set up URL Patterns that we want Django to match and then call a particular consumer to handle the connection.

In the **urls.py**

A screenshot of a computer

Description automatically generated

Go back to **settings.py** and under **INSTALLED\_APPS** type in **ASGI\_APPLICATION = ‘projectname.asgi.application’**

Text

Description automatically generated

In order for our application to work we have to write consumers for it. Consumers as the same as Django Views. So it takes a web request and returns a web response.

Create a **consumers.py** file in the project directory.

Consumers.py will accept all incoming websocket connections, and then reply to all.

We’ll write a **UserConsumer** class which will extend the synchronous WebsocketConsumer class. This class will be responsible for accepting/rejecting incoming connections. This class will also handle receiving and sending a user’s data to that user’s client machine through the established WebSocket connection.

Generally, four methods are defined in a consumer class —

**connect** –

* Accept the incoming connection. This is done using **self.accept()**.
* It is recommended that **accept()** be called as the last action in the **connect()** method if you choose to accept the connection.
* We might want to reject a connection if a user is, let’s say, not authorized.
* **disconnect** - Close the ongoing connection. Done using **self.close()**
* **send** - Send message(s) back to the client.
* **receive** - Logic to handle incoming messages from the client.

Text

Description automatically generated

What all is going on in the above class?

If an incoming connection is noticed, the **connect** method is called, which accepts the connection. We can disconnect the connection by calling **disconnect** on the **UserConsumer** instance

In our given use case, as we’re not receiving any data from the client, our **receive** method doesn’t contain any logic. We’ll only be sending data to the client.

In our send method, we’re just sending some test data back to the client. You’ll notice that the name of our send method is **send\_user**. You can name it however you like. You’ll see later on that we can specify which send method we want to invoke when.

Next steps

* Look up channel layers.
* Connect consumers with channel layers.
* Connect Django signals with channel layers.
* Implement client-side socket connection.
* Test.