WebSockets

You can use WebSockets with **FastAPI**.

WebSockets client

In production

In your production system, you probably have a frontend created with a modern framework like React, Vue.js or Angular.

And to communicate using WebSockets with your backend you would probably use your frontend's utilities.

Or you might have a native mobile application that communicates with your WebSocket backend directly, in native code.

Or you might have any other way to communicate with the WebSocket endpoint.

But for this example, we'll use a very simple HTML document with some JavaScript, all inside a long string.

This, of course, is not optimal and you wouldn't use it for production.

In production you would have one of the options above.

But it's the simplest way to focus on the server-side of WebSockets and have a working example:

```
{!../../../docs_src/websockets/tutorial001.py!}
```

Create a `websocket`

In your **FastAPI** application, create a `websocket`:

```
{!../../../docs_src/websockets/tutorial001.py!}
```

!!! note "Technical Details" You could also use `from starlette.websockets import WebSocket`.

```
**FastAPI** provides the same `WebSocket` directly just as a convenience for you, the developer. But it comes directly from Starlette.
```

Await for messages and send messages

In your WebSocket route you can `await` for messages and send messages.

```
{!../../../docs_src/websockets/tutorial001.py!}
```

You can receive and send binary, text, and JSON data.

Try it

If your file is named `main.py`, run your application with:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Open your browser at http://127.0.0.1:8000.

You will see a simple page like:

You can type messages in the input box, and send them:

And your **FastAPI** application with WebSockets will respond back:

You can send (and receive) many messages:

And all of them will use the same WebSocket connection.

Using `Depends` and others

In WebSocket endpoints you can import from `fastapi` and use:

- `Depends`
- `Security`
- `Cookie`
- `Header`
- `Path`
- `Query`

They work the same way as for other FastAPI endpoints/*path operations*:

```
{!../../../docs_src/websockets/tutorial002.py!}
```

!!! info In a WebSocket it doesn't really make sense to raise an `HTTPException`. So it's better to close the WebSocket connection directly.

```
You can use a closing code from the <a
href="https://tools.ietf.org/html/rfc6455#section-7.4.1" class="external-link"
target="_blank">valid codes defined in the specification</a>.

In the future, there will be a `WebSocketException` that you will be able to `raise`
from anywhere, and add exception handlers for it. It depends on the <a
```

```
href="https://github.com/encode/starlette/pull/527" class="external-link"
target="_blank">PR #527</a> in Starlette.
```

Try the WebSockets with dependencies

If your file is named `main.py`, run your application with:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Open your browser at http://127.0.0.1:8000.

There you can set:

- The "Item ID", used in the path.
- The "Token" used as a query parameter.

!!! tip Notice that the query `token` will be handled by a dependency.

With that you can connect the WebSocket and then send and receive messages:

Handling disconnections and multiple clients

When a WebSocket connection is closed, the `await websocket.receive_text()` will raise a `WebSocketDisconnect` exception, which you can then catch and handle like in this example.

```
{!../../../docs_src/websockets/tutorial003.py!}
```

To try it out:

- Open the app with several browser tabs.
- Write messages from them.
- Then close one of the tabs.

That will raise the `WebSocketDisconnect` exception, and all the other clients will receive a message like:

```
Client #1596980209979 left the chat
```

!!! tip The app above is a minimal and simple example to demonstrate how to handle and broadcast messages to several WebSocket connections.

```
But have in mind that, as everything is handled in memory, in a single list, it will
only work while the process is running, and will only work with a single process.

If you need something easy to integrate with FastAPI but that is more robust,
supported by Redis, PostgreSQL or others, check <a
href="https://github.com/encode/broadcaster" class="external-link"
target="_blank">encode/broadcaster</a>.
```

More info

To learn more about the options, check Starlette's documentation for:

- The `WebSocket` class.
- Class-based WebSocket handling.