Authentication

FastAPI Users allows you to plug in several authentication methods.

How it works?

You can have **several** authentication methods, e.g. a cookie authentication for browser-based queries and a JWT token authentication for pure API queries.

When checking authentication, each method is run one after the other. The first method yielding a user wins. If no method yields a user, an `HTTPException` is raised.

For each backend, you'll be able to add a router with the corresponding `/login` and `/logout`. More on this in the routers documentation.

Transport + Strategy = Authentication backend

An authentication backend is composed of two parts:

Transport

It manages how the token will be carried over the request. We currently provide two methods:

Bearer

The token will be sent through an `Authorization: Bearer` header.

> **Pros and cons**
>
> - ✅ Easy to read and set in every requests.
> - ❌ Needs to be stored manually somewhere in the client.
>
> ➡️ Use it if you want to implement a mobile application or a pure REST API.

Cookie

The token will be sent through a cookie.

Pros and cons

- ✅ Automatically stored and sent securely by web browsers in every requests.
- ✅ Automatically removed at expiration by web browsers.
- ❌ Needs a CSRF protection for maximum security.
- ❌ Harder to work with outside a browser, like a mobile app or a server.

➡️ Use it if you want to implement a web frontend.

Strategy

It manages how the token is generated and secured. We currently provide three methods:

JWT

The token is self-contained in a JSON Web Token.

Pros and cons

- ✅ Self-contained: it doesn't need to be stored in a database.
- ❌ Can't be invalidated on the server-side: it's valid until it expires.

➡️ Use it if you want to get up-and-running quickly.

Database

The token is stored in a table (or collection) in your database.

Pros and cons

- ✅ Secure and performant.
- ✅ Tokens can be invalidated server-side by removing them from the database.
- ✅ Highly customizable: add your own fields, create an API to retrieve the active sessions of your users, etc.
- ❌ Configuration is a bit more complex.

➡️ Use it if you want maximum flexibility in your token management.

Redis

The token is stored in a Redis key-store.

Pros and cons

- ✅ Secure and performant.
- ✅ Tokens can be invalidated server-side by removing them from Redis.
- ❌ A Redis server is needed.

➡️ Use it if you want maximum performance while being able to invalidate tokens.