Get Current User

In the previous chapter the security system (which is based on the dependency injection system) was giving the *path operation function* a `token` as a `str`:

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

But that is still not that useful.

Let's make it give us the current user.

Create a user model

First, let's create a Pydantic user model.

The same way we use Pydantic to declare bodies, we can use it anywhere else:

=== "Python 3.6 and above"

~~~
```Python hl_lines="5  12-16"
{!> ../../../docs_src/security/tutorial002.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="3  10-14"
{!> ../../../docs_src/security/tutorial002_py310.py!}
```
~~~

Create a `get_current_user` dependency

Let's create a dependency `get_current_user`.

Remember that dependencies can have sub-dependencies?

`get_current_user` will have a dependency with the same `oauth2_scheme` we created before.

The same as we were doing before in the *path operation* directly, our new dependency `get_current_user` will receive a `token` as a `str` from the sub-dependency `oauth2_scheme`:

=== "Python 3.6 and above"

~~~
```Python hl_lines="25"
{!> ../../../docs_src/security/tutorial002.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="23"
{!> ../../../docs_src/security/tutorial002_py310.py!}
```
~~~

Get the user

`get_current_user` will use a (fake) utility function we created, that takes a token as a `str` and returns our Pydantic `User` model:

=== "Python 3.6 and above"

~~~
```Python hl_lines="19-22  26-27"
{!> ../../../docs_src/security/tutorial002.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="17-20  24-25"
{!> ../../../docs_src/security/tutorial002_py310.py!}
```
~~~

Inject the current user

So now we can use the same `Depends` with our `get_current_user` in the *path operation*:

=== "Python 3.6 and above"

~~~
```Python hl_lines="31"
{!> ../../../docs_src/security/tutorial002.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="29"
{!> ../../../docs_src/security/tutorial002_py310.py!}
```
~~~

Notice that we declare the type of `current_user` as the Pydantic model `User`.

This will help us inside of the function with all the completion and type checks.

!!! tip You might remember that request bodies are also declared with Pydantic models.

```
Here **FastAPI** won't get confused because you are using `Depends`.
```

!!! check The way this dependency system is designed allows us to have different dependencies (different "dependables") that all return a `User` model.

```
We are not restricted to having only one dependency that can return that type of data.
```

Other models

You can now get the current user directly in the *path operation functions* and deal with the security mechanisms at the **Dependency Injection** level, using `Depends`.

And you can use any model or data for the security requirements (in this case, a Pydantic model `User`).

But you are not restricted to using some specific data model, class or type.

Do you want to have an `id` and `email` and not have any `username` in your model? Sure. You can use these same tools.

Do you want to just have a `str`? Or just a `dict`? Or a database class model instance directly? It all works the same way.

You actually don't have users that log in to your application but robots, bots, or other systems, that have just an access token? Again, it all works the same.

Just use any kind of model, any kind of class, any kind of database that you need for your application. **FastAPI** has you covered with the dependency injection system.

Code size

This example might seem verbose. Have in mind that we are mixing security, data models, utility functions and *path operations* in the same file.

But here's the key point.

The security and dependency injection stuff is written once.

And you can make it as complex as you want. And still, have it written only once, in a single place. With all the flexibility.

But you can have thousands of endpoints (*path operations*) using the same security system.

And all of them (or any portion of them that you want) can take the advantage of re-using these dependencies or any other dependencies you create.

And all these thousands of *path operations* can be as small as 3 lines:

=== "Python 3.6 and above"

~~~
```Python hl_lines="30-32"
{!> ../../../docs_src/security/tutorial002.py!}
```
~~~

=== "Python 3.10 and above"

~~~
```Python hl_lines="28-30"
{!> ../../../docs_src/security/tutorial002_py310.py!}
```
~~~

Recap

You can now get the current user directly in your *path operation function*.

We are already halfway there.

We just need to add a *path operation* for the user/client to actually send the `username` and `password`.

That comes next.