

Simple OAuth2 with Password and Bearer

Now let's build from the previous chapter and add the missing parts to have a complete security flow.

Get the username and password

We are going to use FastAPI security utilities to get the `username` and `password`.

OAuth2 specifies that when using the "password flow" (that we are using) the client/user must send a `username` and `password` fields as form data.

And the spec says that the fields have to be named like that. So `user-name` or `email` wouldn't work.

But don't worry, you can show it as you wish to your final users in the frontend.

And your database models can use any other names you want.

But for the login *path operation*, we need to use these names to be compatible with the spec (and be able to, for example, use the integrated API documentation system).

The spec also states that the `username` and `password` must be sent as form data (so, no JSON here).

scope

The spec also says that the client can send another form field " `scope` ".

The form field name is `scope` (in singular), but it is actually a long string with "scopes" separated by spaces.

Each "scope" is just a string (without spaces).

They are normally used to declare specific security permissions, for example:

- `users:read` or `users:write` are common examples.
- `instagram_basic` is used by Facebook / Instagram.
- `https://www.googleapis.com/auth/drive` is used by Google.

Info

In OAuth2 a "scope" is just a string that declares a specific permission required.

It doesn't matter if it has other characters like `:` or if it is a URL.

Those details are implementation specific.

For OAuth2 they are just strings.

Code to get the username and password

Now let's use the utilities provided by FastAPI to handle this.

OAuth2PasswordRequestForm

First, import `OAuth2PasswordRequestForm`, and use it as a dependency with `Depends` in the *path operation* for `/token`:

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

```
class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

▶ Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
```

```
        return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from typing_extensions import Annotated

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
```

```
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Smith",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
```

```
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
    }
}
```

```
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}
```

OAuth2PasswordRequestForm is a class dependency that declares a form body with:

- The username .
- The password .
- An optional scope field as a big string, composed of strings separated by spaces.
- An optional grant_type .

Tip

The OAuth2 spec actually *requires* a field `grant_type` with a fixed value of `password`, but `OAuth2PasswordRequestForm` doesn't enforce it.

If you need to enforce it, use `OAuth2PasswordRequestFormStrict` instead of `OAuth2PasswordRequestForm`.

- An optional `client_id` (we don't need it for our example).
- An optional `client_secret` (we don't need it for our example).

Info

The `OAuth2PasswordRequestForm` is not a special class for FastAPI as is `OAuth2PasswordBearer`.

`OAuth2PasswordBearer` makes FastAPI know that it is a security scheme. So it is added that way to OpenAPI.

But `OAuth2PasswordRequestForm` is just a class dependency that you could have written yourself, or you could have declared `Form` parameters directly.

But as it's a common use case, it is provided by FastAPI directly, just to make it easier.

Use the form data**Tip**

The instance of the dependency class `OAuth2PasswordRequestForm` won't have an attribute `scope` with the long string separated by spaces, instead, it will have a `scopes` attribute with the actual list of strings for each scope sent.

We are not using `scopes` in this example, but the functionality is there if you need it.

Now, get the user data from the (fake) database, using the `username` from the form field.

If there is no such user, we return an error saying "Incorrect username or password".

For the error, we use the exception `HTTPException`:

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

    def get_user(db, username: str):
        if username in db:
            user_dict = db[username]
            return UserInDB(**user_dict)

    def fake_decode_token(token):
        # This doesn't provide any security at all
        return {"sub": token, "name": "fakeuser"}
```

```
# Check the next version
user = get_user(fake_users_db, token)
return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}
```

▶ 📱 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
```

```
    return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from typing_extensions import Annotated

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str
```

```
def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
    return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
```

```
full_name: str | None = None
disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

```
class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Check the password

At this point we have the user data from our database, but we haven't checked the password.

Let's put that data in the Pydantic `UserInDB` model first.

You should never save plaintext passwords, so, we'll use the (fake) password hashing system.

If the passwords don't match, we return the same error.

Password hashing

"Hashing" means: converting some content (a password in this case) into a sequence of bytes (just a string) that looks like gibberish.

Whenever you pass exactly the same content (exactly the same password) you get exactly the same gibberish.

But you cannot convert from the gibberish back to the password.

Why use password hashing

If your database is stolen, the thief won't have your users' plaintext passwords, only the hashes.

So, the thief won't be able to try to use those same passwords in another system (as many users use the same password everywhere, this would be dangerous).

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
```

```
fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
    return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

▶ Other versions and variants

Python 3.9+

```
from typing import Annotated, Union
```

```
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from typing_extensions import Annotated

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
```

```
):  
    return current_user
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm  
from pydantic import BaseModel  
  
fake_users_db = {  
    "johndoe": {  
        "username": "johndoe",  
        "full_name": "John Doe",  
        "email": "johndoe@example.com",  
        "hashed_password": "fakehashedsecret",  
        "disabled": False,  
    },  
    "alice": {  
        "username": "alice",  
        "full_name": "Alice Wonderson",  
        "email": "alice@example.com",  
        "hashed_password": "fakehashedsecret2",  
        "disabled": True,  
    },  
}  
  
app = FastAPI()  
  
def fake_hash_password(password: str):  
    return "fakehashed" + password  
  
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")  
  
class User(BaseModel):  
    username: str  
    email: str | None = None  
    full_name: str | None = None  
    disabled: bool | None = None  
  
class UserInDB(User):  
    hashed_password: str  
  
    def get_user(db, username: str):  
        if username in db:  
            user_dict = db[username]  
            return UserInDB(**user_dict)  
  
    def fake_decode_token(token):  
        # This doesn't provide any security at all  
        # Check the next version  
        user = get_user(fake_users_db, token)  
        return user  
  
async def get_current_user(token: str = Depends(oauth2_scheme)):  
    user = fake_decode_token(token)  
    if not user:  
        raise HTTPException(  
            status_code=status.HTTP_401_UNAUTHORIZED,  
            detail="Invalid authentication credentials",  
            headers={"WWW-Authenticate": "Bearer"},  
        )  
    return user  
  
async def get_current_active_user(current_user: User = Depends(get_current_user)):  
    if current_user.disabled:  
        raise HTTPException(status_code=400, detail="Inactive user")  
    return current_user  
  
@app.post("/token")  
async def login(form_data: OAuth2PasswordRequestForm = Depends()):  
    user_dict = fake_users_db.get(form_data.username)  
    if not user_dict:  
        raise HTTPException(status_code=400, detail="Incorrect username or password")  
    user = UserInDB(**user_dict)  
    hashed_password = fake_hash_password(form_data.password)  
    if not hashed_password == user.hashed_password:  
        raise HTTPException(status_code=400, detail="Incorrect username or password")  
  
    return {"access_token": user.username, "token_type": "bearer"}
```

```
@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
```

```
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}  
  
@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

About `**user_dict`

`UserInDB(**user_dict)` means:

Pass the keys and values of the `user_dict` directly as key-value arguments, equivalent to:

```
UserInDB(
    username = user_dict["username"],
    email = user_dict["email"],
    full_name = user_dict["full_name"],
    disabled = user_dict["disabled"],
    hashed_password = user_dict["hashed_password"],
)
```

Info

For a more complete explanation of `**user_dict` check back in [the documentation for Extra Models](#).

Return the token

The response of the `token` endpoint must be a JSON object.

It should have a `token_type`. In our case, as we are using "Bearer" tokens, the token type should be "`bearer`".

And it should have an `access_token`, with a string containing our access token.

For this simple example, we are going to just be completely insecure and return the same `username` as the token.

Tip

In the next chapter, you will see a real secure implementation, with password hashing and [JWT](#) tokens.

But for now, let's focus on the specific details we need.

Python 3.10+

```
from typing import Annotated
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None
```

```
class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

▶ 🌐 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
```

```
email: Union[str, None] = None
full_name: Union[str, None] = None
disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from typing_extensions import Annotated

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

```
class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()
```

```
def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
    return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Python 3.8+ · non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
    },
}
```

```
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Tip

By the spec, you should return a JSON with an `access_token` and a `token_type`, the same as in this example.

This is something that you have to do yourself in your code, and make sure you use those JSON keys.

It's almost the only thing that you have to remember to do correctly yourself, to be compliant with the specifications.

For the rest, FastAPI handles it for you.

Update the dependencies

Now we are going to update our dependencies.

We want to get the `current_user` *only* if this user is active.

So, we create an additional dependency `get_current_active_user` that in turn uses `get_current_user` as a dependency.

Both of these dependencies will just return an HTTP error if the user doesn't exist, or if is inactive.

So, in our endpoint, we will only get a user if the user exists, was correctly authenticated, and is active:

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
    return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me():

    
```

```
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

► 📚 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
```

```
    return {"access_token": user.username, "token_type": "bearer"}
```

```
@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from typing_extensions import Annotated

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)],
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
async def login(form_data: Annotated[OAuth2PasswordRequestForm, Depends()]):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
```

```
        hashed_password = fake_hash_password(form_data.password)
        if not hashed_password == user.hashed_password:
            raise HTTPException(status_code=400, detail="Incorrect username or password")

        return {"access_token": user.username, "token_type": "bearer"}
```

```
@app.get("/users/me")
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)],
):
    return current_user
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}

app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user

@app.post("/token")
```

```
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}  
  
@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "fakehashedsecret",
        "disabled": False,
    },
    "alice": {
        "username": "alice",
        "full_name": "Alice Wonderson",
        "email": "alice@example.com",
        "hashed_password": "fakehashedsecret2",
        "disabled": True,
    },
}
app = FastAPI()

def fake_hash_password(password: str):
    return "fakehashed" + password

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

class User(BaseModel):
    username: str
    email: Union[str, None] = None
    full_name: Union[str, None] = None
    disabled: Union[bool, None] = None

class UserInDB(User):
    hashed_password: str

def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)

def fake_decode_token(token):
    # This doesn't provide any security at all
    # Check the next version
    user = get_user(fake_users_db, token)
    return user

async def get_current_user(token: str = Depends(oauth2_scheme)):
    user = fake_decode_token(token)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
    if current_user.disabled:
```

```
raise HTTPException(status_code=400, detail="Inactive user")
return current_user

@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user_dict = fake_users_db.get(form_data.username)
    if not user_dict:
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    user = UserInDB(**user_dict)
    hashed_password = fake_hash_password(form_data.password)
    if not hashed_password == user.hashed_password:
        raise HTTPException(status_code=400, detail="Incorrect username or password")

    return {"access_token": user.username, "token_type": "bearer"}

@app.get("/users/me")
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user
```

Info

The additional header `WWW-Authenticate` with value `Bearer` we are returning here is also part of the spec.

Any HTTP (error) status code 401 "UNAUTHORIZED" is supposed to also return a `WWW-Authenticate` header.

In the case of bearer tokens (our case), the value of that header should be `Bearer`.

You can actually skip that extra header and it would still work.

But it's provided here to be compliant with the specifications.

Also, there might be tools that expect and use it (now or in the future) and that might be useful for you or your users, now or in the future.

That's the benefit of standards...

See it in action

Open the interactive docs: <http://127.0.0.1:8000/docs>.

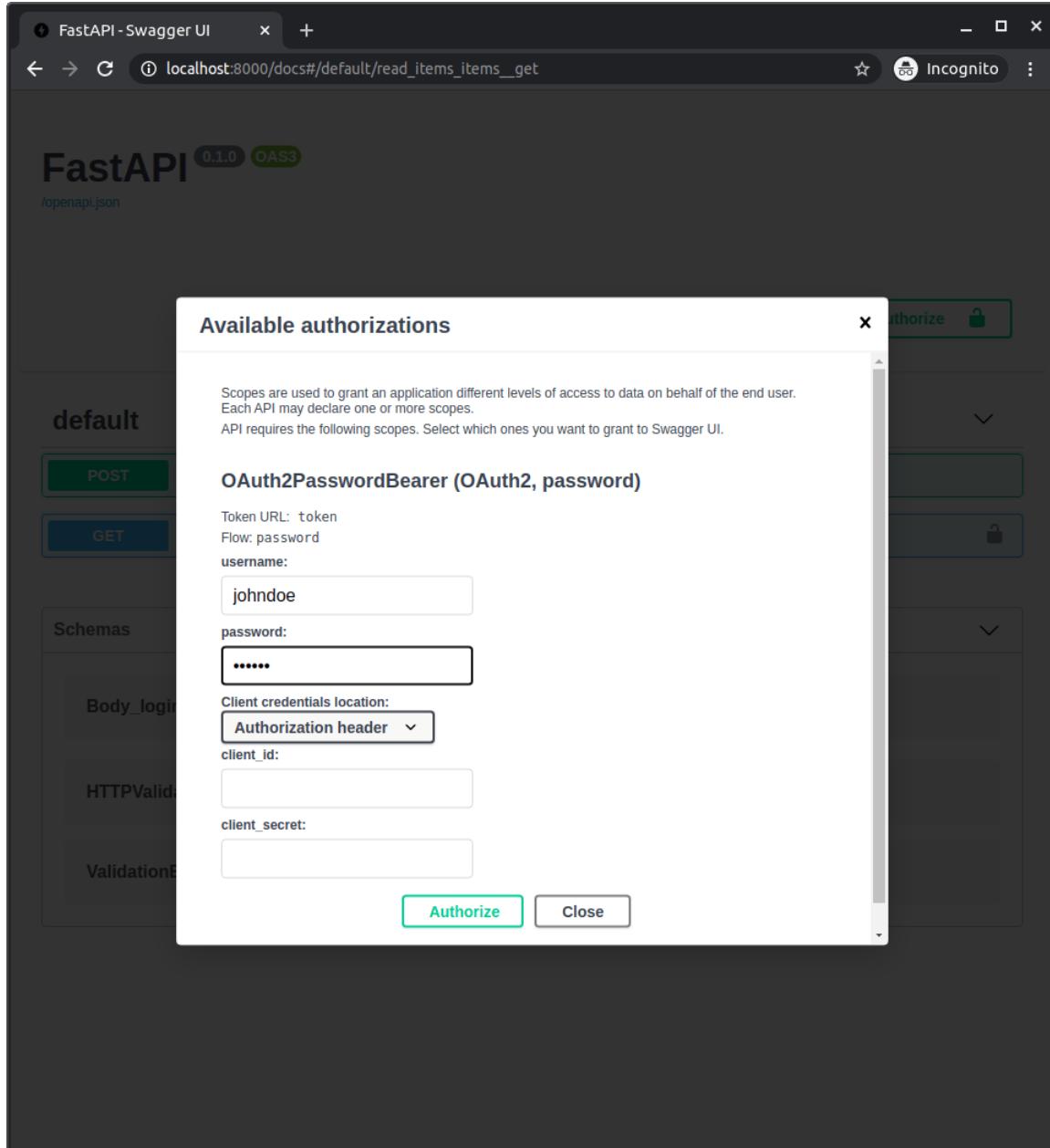
Authenticate

Click the "Authorize" button.

Use the credentials:

User: johndoe

Password: secret



After authenticating in the system, you will see it like:

The screenshot shows the FastAPI Swagger UI interface. On the left, there's a sidebar with sections like 'default', 'POST', 'GET', 'Schemas', 'Body_login', 'HTTPValid', and 'ValidationError'. A modal window titled 'Available authorizations' is open in the center. It contains instructions about scopes and API requirements, followed by a section for 'OAuth2PasswordBearer (OAuth2, password)'. This section shows 'Authorized' details: Token URL: token, Flow: password, username: john Doe, password: *****, Client credentials location: basic, and client secret: *****. At the bottom of the modal are 'Logout' and 'Close' buttons. In the top right corner of the main page, there's an 'Authorize' button with a lock icon.

Get your own user data

Now use the operation `GET` with the path `/users/me`.

You will get your user's data, like:

```
{  
    "username": "johndoe",  
    "email": "johndoe@example.com",  
    "full_name": "John Doe",  
    "disabled": false,  
    "hashed_password": "fakehashedsecret"  
}
```

The screenshot shows the Fast API - Swagger UI interface. At the top, there's a header bar with tabs for 'Fast API - Swagger UI' and a '+' icon. Below the header, the URL is set to '127.0.0.1:8000/docs'. The main content area is titled 'default'. It displays two operations: a POST operation for '/token' labeled 'Login Post' and a GET operation for '/users/me' labeled 'Read Users Me Get'. The 'GET /users/me' section has a 'Parameters' section with a 'Cancel' button. Below it is a 'Responses' section containing a 'Curl' command and a 'Request URL' field. Under the 'Responses' section, there's a detailed 'Code 200' response with 'Details' for 'Response body' and 'Response headers'. The 'Response body' example is a JSON object:

```
{  
    "username": "johndoe",  
    "email": "johndoe@example.com",  
    "full_name": "John Doe",  
    "disabled": false,  
    "hashed_password": "fakehashedsecret"  
}
```

A 'Download' button is located next to the response body example. The 'Response headers' section lists:

```
content-length: 129  
content-type: application/json  
date: Mon, 24 Dec 2018 05:25:36 GMT  
server: uvicorn
```

If you click the lock icon and logout, and then try the same operation again, you will get an HTTP 401 error of:

```
{  
    "detail": "Not authenticated"  
}
```

Inactive user

Now try with an inactive user, authenticate with:

User: alice

Password: secret2

And try to use the operation `GET` with the path `/users/me`.

You will get an "Inactive user" error, like:

```
{  
    "detail": "Inactive user"  
}
```

Recap

You now have the tools to implement a complete security system based on `username` and `password` for your API.

Using these tools, you can make the security system compatible with any database and with any user or data model.

The only detail missing is that it is not actually "secure" yet.

In the next chapter you'll see how to use a secure password hashing library and [JWT](#) tokens.

© 2018 Sebastián Ramírez

Licensed under the MIT License.

<https://fastapi.tiangolo.com/tutorial/security/simple-oauth2/>

Exported from DevDocs — <https://devdocs.io>