**FastAPI for Django Programmers: Commands and creating users**

[[Marc Nealer](https://medium.com/@marcnealer?source=post_page-----643557185632--------------------------------)](https://medium.com/@marcnealer?source=post_page-----643557185632--------------------------------)

[Marc Nealer](https://medium.com/@marcnealer?source=post_page-----643557185632--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F38491f9efab0&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40marcnealer%2Ffastapi-for-django-programmers-commands-and-creating-users-643557185632&user=Marc+Nealer&userId=38491f9efab0&source=post_page-38491f9efab0----643557185632---------------------post_header-----------)

4 min read

·

16 hours ago

1

On my journey to building an app with Django like components, more specifically, an authentication system, I’ve come across the need to run commands on the command line, to create users, and of course have view that generate html.

Commands

For the commands I need to run, what I really need is access to the ORM. With Django, most of the code around the commands, is linked to working with the django framework. For our commands, none of that is needed.

So to run a command, we need to link to the ORM, first and foremost, and we want to ensure that we link to the same ORM setup as our application. Since Tortoise is independent of FastAPI, we can run different functions to link to the ORM, than we do with FastAPI and we get the same result.

What we do need to have though , is the same database URL and modules parameters, that are used in our main.py file. To manage this, I moved both to variables in main.py, so that I can import them.

import os  
from fastapi import FastAPI  
from fastapi.staticfiles import StaticFiles  
from tortoise.contrib.fastapi import register\_tortoise  
from dotenv import find\_dotenv, dotenv\_values  
from routes.auth import auth  
  
file\_path = os.path.dirname(\_\_file\_\_)  
static\_dir = os.path.join(file\_path, "static")  
  
app = FastAPI()  
  
app.mount("/static", StaticFiles(directory=static\_dir), name="static")  
  
settings = dotenv\_values(find\_dotenv(".ide\_settings"))  
secret\_key = settings.get("secret\_key", "aVhMbVFvclczRjYxSm1oTzhBcVlGN0NnT2VoWDlVbE9uc3dYX0xsTnAxRT0=")  
db\_user = settings.get("db\_user", "clusteride")  
db\_password = settings.get("db\_password", "clusteride")  
db\_host = settings.get("db\_host", "localhost")  
db\_port = settings.get("db\_port", "5432")  
db\_name = settings.get("db\_name", "clusteride")  
  
db\_url = f"asyncpg://{db\_user}:{db\_password}@{db\_host}:{db\_port}/{db\_name}"  
db\_modules = {"models": ["models"]}  
  
register\_tortoise(  
 app,  
 db\_url=db\_url,  
 modules=db\_modules,  
 generate\_schemas=True,  
 add\_exception\_handlers=True,  
)  
  
  
app.include\_router(auth)

I then created a commands directory and a core.py file and put in the simple code to initiate a database connection

from tortoise import Tortoise  
from main import db\_url, db\_modules  
  
  
async def init\_db():  
 await Tortoise.init(  
 db\_url=db\_url,  
 modules=db\_modules,  
 )  
 await Tortoise.generate\_schemas()

Note that Tortoise.init() is creating the connection, not register\_tortoise. Also note that the function is a coroutine, so this will need to be called as such. Lucky for us, Tortoise comes with a nice function that creates a thread and runs a function called run\_async, though you could go through creating and starting a task instead.

Here is a simple command to print a list of users.

from tortoise import run\_async  
from models import Users  
from .core import init\_db  
  
  
async def main():  
 await init\_db()  
  
 users = await Users.all()  
 for user in users:  
 print(f"{user.username} - {user.email}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run\_asyn

To run this I just use.

python -m commands.listusers

That’s it. Nice and simple. I point I will make here, is that I have come across many instances where the features in Django, are not really features, but work arounds. With no framework controlling the layout of our projects and adding modules in, we don’t need a lot of these features.

Now, I have an example of a script to add users. This is a little more complex, as we have to deal with passwords and of course, we need to encrypt them.

I created a new directory called **contrib** so we can stay in line with Django terms and in there I added auth.py. In here, I added functions that I will use for creating and storing user records.

from models import Users  
from main import secret\_key  
from base64 import b64decode  
from cryptography.fernet import Fernet  
import datetime  
  
  
async def authenticate(username, password):  
 user = await Users.get\_or\_none(username=username)  
 if not user:  
 return None  
 else:  
 pwd = hash\_password(password)  
 if pwd == user.password:  
 return user  
 else:  
 return None  
  
  
async def get\_key():  
 return b64decode(secret\_key.encode())  
  
  
async def create\_user(username:str, password: str, email:str) -> Users:  
 pwd = await hash\_password(password)  
 if email:  
 user = await Users.create(username=username, password=pwd, email=email, last\_login=datetime.datetime.now())  
 else:  
 user = await Users.create(username=username, password=pwd, email="Not stated", last\_login=datetime.datetime.now())  
 return user  
  
  
async def hash\_password(password:str):  
 f = Fernet(await get\_key())  
 return f.encrypt(password.encode()).decode()

The first thing to discuss here is the use of the**cryptography** library and **Fernet** encryption. This is a reasonably strong encryption system, and an easy to use library. Passwords are encrypted and decrypted using a secret key, that will be pulled in, in main.py.

A point to note here, is that you need to ensure that your secret key is secret. You can’t hard code this into main.py. What I did, is have the key being pulled in using dotenv, but I also encoded the key using the base64 library. Base64 is not high level encryption, but it does add a small extra layer for security.

In commands I then added a createuser.py file and coded the following.

from tortoise import run\_async  
from commands.core import init\_db  
from contrib.auth import create\_user  
import getpass  
from models import Users  
  
  
async def validate\_password(password):  
 if len(password) < 8:  
 print("Error: Password must be at least 8 characters long.")  
 return False  
  
 elif len(set(password)) < 6:  
 print("Error: Your password must contain at least 6 different characters.")  
 return False  
  
 elif not any(char.isupper() for char in password):  
 print("Error: Your password must contain at least 1 uppercase letter.")  
 return False  
  
 elif not any(char.islower() for char in password):  
 print("Error: Your password must contain at least 1 lowercase letter.")  
 return False  
 return True  
  
  
async def main():  
 await init\_db()  
  
 while True:  
 username = input("Please enter a username: ")  
 if len(username) < 8:  
 print("Error: Username must be at least 8 characters long")  
 continue  
 if await Users.filter(username=username).exists():  
 print("Username already exists. Please try again.")  
 continue  
 password = getpass.getpass("Please enter a password: ")  
 if await validate\_password(password):  
 confirm\_password = getpass.getpass("Please verify the password: ")  
  
 if password == confirm\_password:  
 print("Password confirmed.")  
 break  
 else:  
 print("Passwords do not match. Please try again.")  
 continue  
 else:  
 print("Your password does not meet requirements")  
 print("Requirements: Min 8 chars long, Uppercase letters, Lowercase letters, 6 differnt characters.")  
 continue  
 email = input("Please enter your email address (Optional): ")  
 user = await create\_user(username, password, email)  
 print(f"User {user.username} created successfully!")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run\_async(main())

This command is run in a similar way to above

python -m command.createuser

That’s it. A nice simple way to add command like commands for your various tasks.