[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

Form data requires "python-multipart" to be installed. You can install "python-multipart" with: pip install python-multipart Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 25, in <module> @app.post("/token") ^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 994, in decorator self.add\_api\_route( File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 933, in add\_api\_route route = route\_class( ^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 554, in \_\_init\_\_ self.dependant = get\_dependant(path=self.path\_format, call=self.endpoint) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 292, in get\_dependant sub\_dependant = get\_param\_sub\_dependant( ^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 126, in get\_param\_sub\_dependant return get\_sub\_dependant( ^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 162, in get\_sub\_dependant sub\_dependant = get\_dependant( ^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 285, in get\_dependant param\_details = analyze\_param( ^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 482, in analyze\_param ensure\_multipart\_is\_installed() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\dependencies\utils.py", line 115, in ensure\_multipart\_is\_installed raise RuntimeError(multipart\_not\_installed\_error) from None RuntimeError: Form data requires "python-multipart" to be installed. You can install "python-multipart" with: pip install python-multipart what is this

**ChatGPT said:**

The error you're seeing means that your FastAPI app is trying to handle **form data** (e.g., from a POST request with application/x-www-form-urlencoded or multipart/form-data content type), but the required package python-multipart is **not installed**.

**📌 What does the error mean?**

FastAPI uses python-multipart to parse form data when an endpoint has parameters like:

python

CopyEdit

from fastapi import Form @app.post("/token") async def login(username: str = Form(...), password: str = Form(...)): ...

When FastAPI sees a Form(...) dependency, it tries to use python-multipart behind the scenes to read that data.

**🧯 How to fix it?**

You just need to install the required package:

bash

CopyEdit

pip install python-multipart

**✅ Steps to Resolve:**

1. Open your terminal or command prompt.
2. Run the install command:

bash

CopyEdit

pip install python-multipart

1. Restart your FastAPI app.

**💡 Optional Tip:**

If you're working on a project with multiple dependencies, it’s a good idea to maintain a requirements.txt file. After installing python-multipart, you can update the file with:

bash

CopyEdit

pip freeze > requirements.txt

Let me know if you want help checking your FastAPI form handling or if you're not sure where the form inputs are defined.

**You said:**

"GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjEsInVzZXJuYW1lIjoiZmRzZyJ9.AbbU\_J4IvCI\_g8qt-cQwDm8Q2wun36dE4VSiAtLOxgU HTTP/1.1" 401 Unauthorized what is this

**ChatGPT said:**

This log message:

arduino

CopyEdit

"GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjEsInVzZXJuYW1lIjoiZmRzZyJ9.AbbU\_J4IvCI\_g8qt-cQwDm8Q2wun36dE4VSiAtLOxgU HTTP/1.1" 401 Unauthorized

**🔍 What it means:**

* **Request**: A client (probably a browser or API client like Postman) made a GET request to your /verification endpoint, with a **JWT token** as a query parameter (?token=...).
* **Response code**: 401 Unauthorized — the server **rejected** the request because **authentication failed**.

**🧪 Why it failed:**

The token in the query parameter is a **JWT (JSON Web Token)**:

CopyEdit

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

This kind of error usually means **one of the following**:

1. **Invalid Signature**:
   * The token was signed with a different secret key than your backend is using to verify it.
2. **Expired or Malformed Token**:
   * If your JWT has a time-based expiry (exp claim), and it's expired.
   * Or if it’s incorrectly structured or missing required claims.
3. **Token Verification Code Issues**:
   * Your backend logic may not be correctly decoding or validating the token.
4. **Token not passed properly**:
   * Some systems expect JWTs in the **Authorization header** (Bearer <token>), not as a query parameter — but this depends on how your route is written.

**✅ What to do:**

**🔐 If you're verifying the token in code like this:**

python

CopyEdit

from fastapi import HTTPException, Depends from jose import JWTError, jwt def verify\_token(token: str): try: payload = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"]) user\_id = payload.get("id") if user\_id is None: raise HTTPException(status\_code=401, detail="Invalid token") return user\_id except JWTError: raise HTTPException(status\_code=401, detail="Invalid token")

Make sure:

* SECRET\_KEY matches the key used to generate the token.
* The token hasn’t expired.
* Your /verification endpoint is correctly retrieving the token.

**📋 You can also test the token at:**

* [https://jwt.io](https://jwt.io/) — to decode and inspect it.

**👀 Want help checking your verification route?**

If you show me your /verification endpoint code, I can point out exactly what’s wrong.

Let me know!

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } templates = Jinja2Templates(directory="templates") @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module from fastapi.exceptions import HTTPException, DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithm="HS256") user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token", headers = {"WWW-Authenticate": "Bearer"} ) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) return user async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): user = await User.get(username = username) if user and verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token this is my authentication module can you find out what is wrong

**ChatGPT said:**

Thanks for sharing your code — you're building a solid FastAPI authentication and registration system. From your main module and authentication module, here's a clear breakdown of what **is wrong** and **how to fix it**:

**❗ Problem Summary:**

You're getting a 401 Unauthorized error when calling:

bash

CopyEdit

GET /verification?token=eyJhbGciOi...

Despite the token being passed as a query parameter, the endpoint fails authentication. The problem lies in your very\_token() function in authentication.py.

**🔍 Main Issues Identified:**

**1. ✅ Token decoding is incorrect:**algorithm**vs**algorithms

In very\_token():

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithm="HS256")

Should be:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

algorithms is expected to be a **list**, not algorithm= (this will silently fail or misbehave).

**2. ❌**verify\_password**is not awaited in**authenticate\_user

You're calling:

python

CopyEdit

if user and verify\_password(password, user.password):

But verify\_password is async, so it should be:

python

CopyEdit

if user and await verify\_password(password, user.password):

Otherwise, the function returns a coroutine object and always evaluates True, leading to invalid logins.

**3. ✅ Better Error Handling in**very\_token

You have two except blocks, but FastAPI only hits the first one. Here's your current code:

python

CopyEdit

try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithm="HS256") user = await User.get(id = payload.get("id")) except: raise HTTPException(...) # Generic error except DoesNotExist: raise HTTPException(...) # Never reached

This is incorrect because except catches **everything**, including DoesNotExist.

**Fix**:

python

CopyEdit

from tortoise.exceptions import DoesNotExist async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

**✅ Summary of Fixes Needed:**

| **Issue** | **Fix** |
| --- | --- |
| Wrong keyword for JWT decoding | Change algorithm="HS256" to algorithms=["HS256"] |
| Missing await in password check | Add await before verify\_password(...) |
| Misplaced exception handling | Catch DoesNotExist **before** the general except |
| Optional: logging errors | Add logging inside the except blocks to help with debugging |

**🧪 Final Tip: Testing the Token**

You can decode and inspect your JWT at [https://jwt.io](https://jwt.io/) to make sure:

* It's well-formed.
* It contains the id field.
* It matches the secret key you use for decoding.

Would you like me to refactor and send back a clean version of the authentication.py file with fixes?

**You said:**

from fastapi.exceptions import HTTPException, DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms="HS256") user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token", headers = {"WWW-Authenticate": "Bearer"} ) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) return user async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): user = await User.get(username = username) if user and verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token what is wrong here did a typo

**ChatGPT said:**

Yes, there are **two critical issues** (including a **typo**) in your code that will cause incorrect behavior:

**❗ 1. Incorrect JWT decode parameter —**algorithms="HS256"**is a typo**

**❌ Your current line:**

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms="HS256")

**✅ Correct version:**

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

* algorithms must be a **list**, not a string.
* This typo can cause the JWT decode function to silently fail or raise an exception depending on the version of pyjwt.

**❗ 2.**verify\_password()**is**await**able but not awaited**

**❌ In**authenticate\_user**:**

python

CopyEdit

if user and verify\_password(password, user.password):

This is wrong because verify\_password is an async def, so this will always evaluate to True (since it's a coroutine object).

**✅ Correct:**

python

CopyEdit

if user and await verify\_password(password, user.password):

**❗ 3. Incorrect exception import (optional)**

You have:

python

CopyEdit

from fastapi.exceptions import HTTPException, DoesNotExist

But:

* HTTPException comes from fastapi, **correct**
* DoesNotExist is **not from**fastapi.exceptions

**✅ Correct import:**

python

CopyEdit

from tortoise.exceptions import DoesNotExist

**✅ Clean Fixed Version**

Here is the corrected and clean version of your authentication.py:

python

CopyEdit

from fastapi import status, HTTPException from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from models import User from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto") def get\_hashed\_password(password: str) -> str: return pwd\_context.hash(password) async def verify\_password(plain\_password: str, hashed\_password: str) -> bool: return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username: str, password: str): try: user = await User.get(username=username) except DoesNotExist: return False if user and await verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str) -> str: user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid username or password", headers={"WWW-Authenticate": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

Let me know if you'd like help writing tests for this module or want to add token expiry (exp claim).

**You said:**

Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 18 except: ^^^^^ SyntaxError: default 'except:' must be last WARNING: WatchFiles detected changes in 'authentication.py'. Reloading... Process SpawnProcess-2: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 16 payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=[]"HS256"]) ^ SyntaxError: closing parenthesis ']' does not match opening parenthesis '(' WARNING: WatchFiles detected changes in 'authentication.py'. Reloading... Process SpawnProcess-3: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 18 except: ^^^^^ SyntaxError: default 'except:' must be last

**ChatGPT said:**

You're running into **two syntax errors** in your authentication.py file that are preventing Uvicorn from running your FastAPI app.

**🔴 Error 1: Incorrect**jwt.decode**syntax**

From the log:

javascript

CopyEdit

SyntaxError: closing parenthesis ']' does not match opening parenthesis '('

This is the line causing it:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=[]"HS256"])

**✅ Fix:**

You accidentally mixed square brackets and a string.

Correct it to:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

**🔴 Error 2:**except:**block must come last**

From the log:

javascript

CopyEdit

SyntaxError: default 'except:' must be last

Your try-except block likely looks like this:

python

CopyEdit

try: ... except: ... except DoesNotExist: ...

Python does **not allow** the generic except: before specific exception types. You need to reorder the blocks.

**✅ Fix:**

python

CopyEdit

try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

**✅ Clean, Fixed Version of**very\_token

python

CopyEdit

async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

**🔁 After Fixing**

Once you've fixed these two syntax errors:

1. Save authentication.py.
2. Let uvicorn reload automatically or restart the server manually:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you want help adding token expiry or improving the logging.

**You said:**

Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 18 except: ^^^^^ SyntaxError: default 'except:' must be last

**ChatGPT said:**

The error you're seeing is a **Python**SyntaxError because the except: clause is not placed **last** in your try-except block.

**🔍 Error Summary:**

plaintext

CopyEdit

SyntaxError: default 'except:' must be last

This means you have something like:

python

CopyEdit

try: ... except: ... except SomeException: ...

And **this is not allowed** in Python. The generic except: must **always come last**, because it's a catch-all.

**✅ How to Fix It**

Update your authentication.py like this:

**❌ Incorrect:**

python

CopyEdit

try: payload = jwt.decode(...) user = await User.get(...) except: raise HTTPException(...) # <-- This must not come before a specific exception! except DoesNotExist: raise HTTPException(...)

**✅ Correct:**

python

CopyEdit

try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: # This goes LAST raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

**🧼 Tip for Clarity**

You might want to **avoid using**except:**entirely**, and use except Exception: instead. It's cleaner and avoids unexpected behavior.

**✅ Final Steps**

1. Save the file.
2. Restart the FastAPI server:

bash

CopyEdit

uvicorn main:app --reload

1. Confirm the issue is resolved.

Let me know if you want help setting up logging to debug exceptions properly.

**You said:**

Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 18 except: ^^^^^ SyntaxError: default 'except:' must be last WARNING: WatchFiles detected changes in 'authentication.py'. Reloading... Process SpawnProcess-2: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 6, in <module> from authentication import \* File "D:\Data Structures and Algorithms\Ecommerce\authentication.py", line 1, in <module> from fastapi.exceptions import DoesNotExist ImportError: cannot import name 'DoesNotExist' from 'fastapi.exceptions' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\exceptions.py)

**ChatGPT said:**

You're now encountering **two separate issues** in your code, both of which are stopping your FastAPI app from running. Let me help you fix both.

**✅ Problem 1:**except:**Must Be Last**

You've already encountered this and attempted to fix it. Here's how to do it correctly:

**✅ Correct Order of**except**Blocks:**

python

CopyEdit

try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user

✅ Ensure the **generic**except is **last**, after all specific exceptions.

**🚫 Problem 2:**ImportError: cannot import name 'DoesNotExist' from 'fastapi.exceptions'

**❌ This line is incorrect:**

python

CopyEdit

from fastapi.exceptions import DoesNotExist

DoesNotExist **does not exist in FastAPI**. It is actually a Tortoise ORM exception.

**✅ Fix:**

Replace this:

python

CopyEdit

from fastapi.exceptions import HTTPException, DoesNotExist

With this:

python

CopyEdit

from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist

**✅ Summary of Fixes**

Update the **top of your**authentication.py like this:

python

CopyEdit

from fastapi import HTTPException, status from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from tortoise.exceptions import DoesNotExist # <-- Correct import from mail import config\_credentials

And update your very\_token function like this:

python

CopyEdit

async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} )

**✅ Final Step: Restart Uvicorn**

Now that you've fixed these, restart your server:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you'd like help setting up structured logging or error tracking too.

**You said:**

Azm5vlg HTTP/1.1" 307 Temporary Redirect INFO: 127.0.0.1:58546 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsInVzZXJuYW1lIjoidnNmdmcifQ.uUAtHNpt0xrVLXLUWHiF6mnCJqQNG0omQnRpAzm5vlg HTTP/1.1" 500 Internal Server Error ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 113, in email\_verification return templates.TemplateResponse("verification.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates' INFO: 127.0.0.1:58550 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsInVzZXJuYW1lIjoidnNmdmcifQ.uUAtHNpt0xrVLXLUWHiF6mnCJqQNG0omQnRpAzm5vlg HTTP/1.1" 500 Internal Server Error ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 107, in email\_verification return templates.TemplateResponse("already\_verified.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'already\_verified.html' not found in search path: 'templates' on the web page it shows internal server error

**ChatGPT said:**

You're getting a 500 Internal Server Error because Jinja2 cannot find your HTML templates:

bash

CopyEdit

jinja2.exceptions.TemplateNotFound: 'verification.html'

jinja2.exceptions.TemplateNotFound: 'already\_verified.html'

**✅ What This Means**

FastAPI is using Jinja2 (via Starlette) to render HTML, but the templates it's trying to load (verification.html, already\_verified.html) **don’t exist in the expected directory**.

**🔧 How to Fix**

**1. Create a**templates/**directory**

In the **root of your project** (where main.py is located), create a folder named:

nginx

CopyEdit

templates

**2. Add required HTML files inside it**

Add the missing files:

* templates/verification.html
* templates/already\_verified.html

You can add basic placeholder HTML like this:

templates/verification.html

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Email Verified</title> </head> <body> <h1>Your email has been successfully verified!</h1> </body> </html>

templates/already\_verified.html

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Already Verified</title> </head> <body> <h1>This email has already been verified.</h1> </body> </html>

**3. Ensure you're configuring templates correctly in**main.py

You should have something like this at the top of your main.py:

python

CopyEdit

from fastapi.templating import Jinja2Templates templates = Jinja2Templates(directory="templates")

**4. Restart the Server**

After adding the templates, restart your FastAPI server:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you'd like to customize the HTML or need help dynamically passing data into the templates (e.g. username, time of verification).

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } templates = Jinja2Templates(directory="templates") @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta http-equiv="X-UA-Compactible" content="IE=edge"> <meta name="viewport" content="width=device-width, initial-sclae=1.0"> <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"> <title>Account Verification</title> </head> <body> <div class="container"> <div class="row" style="margin-top: 30vh;"> <div class="col-md-10 col-sm-10 col-xm-12 m-auto p-4"> <div class="card text-center"> <div class="card-header"> Email Verification </div> <div class="card-body"> <h5 class="card-title">Email Verification Successfully</h5> <p class="card-text">Hello {{username}}, you can now log into your account, enjoy </p> <span class="badge badge-primary p-3">BigTree</span> </div> </div> </div> </div> </div> <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script> <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script> <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script> </body> </html> this is verification.html i do not have a already\_verified.html

**ChatGPT said:**

Thanks for sharing everything—your main.py and verification.html are mostly solid. The error you're seeing:

bash

CopyEdit

jinja2.exceptions.TemplateNotFound: 'already\_verified.html'

means that **when a user is already verified**, the server tries to return already\_verified.html, but **you haven't created that file yet**.

**✅ Solution**

You **must** create an already\_verified.html template. Here's a minimal and styled example you can place in your templates/ folder:

**✅ Create**templates/already\_verified.html

html

CopyEdit

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Email Already Verified</title> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"> </head> <body> <div class="container"> <div class="row" style="margin-top: 30vh;"> <div class="col-md-10 col-sm-10 col-xm-12 m-auto p-4"> <div class="card text-center"> <div class="card-header"> Already Verified </div> <div class="card-body"> <h5 class="card-title">Your email is already verified</h5> <p class="card-text">Hi {{username}}, you can proceed to log in to your account.</p> <span class="badge badge-success p-3">BigTree</span> </div> </div> </div> </div> </div> </body> </html>

**🔄 Then Restart Your Server**

Make sure you **save the file** and restart your FastAPI server to pick up the new template:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you’d like to customize the appearance or add dynamic behavior (like a login link or redirect).

**You said:**

can you explain the need of already\_verfied.html file and where should i place it

**ChatGPT said:**

Absolutely — let me break it down clearly:

**✅ Why**already\_verified.html**Is Needed**

In your FastAPI code, this block inside the /verification endpoint checks if the user is **already verified**:

python

CopyEdit

if user.is\_verified: return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username})

So if someone clicks the verification link **again** (maybe by mistake, or they saved the email), FastAPI tries to render a page that says:

"Hey, you've already verified your account."

That response is intended to be **user-friendly feedback** rather than showing an error or blank page. Since you're using Jinja2Templates, that page must exist as a file.

**📁 Where to Place**already\_verified.html

You defined this in your code:

python

CopyEdit

templates = Jinja2Templates(directory="templates")

So FastAPI will **look for HTML templates in a folder named**templates in your project directory.

You should place your already\_verified.html file here:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\templates\already\_verified.html

📝 Your project structure should now look like this:

pgsql

CopyEdit

Ecommerce/

├── main.py

├── models.py

├── authentication.py

├── mail.py

├── database.sqlite3

└── templates/

├── verification.html

└── already\_verified.html ← This is the missing file

**🚨 What Happens Without It?**

If the file doesn't exist, you get this error (as you've seen):

bash

CopyEdit

jinja2.exceptions.TemplateNotFound: 'already\_verified.html'

Which results in a **500 Internal Server Error** on the frontend — not a good user experience.

**✅ Summary**

| **Purpose** | **Location** | **File Required** |
| --- | --- | --- |
| Show when email verified | templates/verification.html | ✅ Already present |
| Show when already verified | templates/already\_verified.html | ❌ You need to create it |

Once you add already\_verified.html to the templates/ folder, your app will handle both cases properly.

Let me know if you want a login button or redirect added to the page.

**You said:**

i think i am having another problem it always goes to account already exist even if i am using a new email and username can you confirm

**ChatGPT said:**

Yes — let's debug that behavior. You mentioned the system **always returns “account already exists”**, even when using a new username and email.

Let’s review the relevant part of your /registration endpoint:

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists")

**✅ What This Does**

1. **Checks if username exists**.
2. **Hashes the password**.
3. **Tries to create the user** — if the email or username already exists in the database (and is set to be unique), a Tortoise IntegrityError is raised.

**🧪 How to Confirm the Problem**

There are **2 main possibilities** for this unexpected behavior:

**🔸 Possibility 1: You’re registering users with already existing usernames or emails.**

To test this clearly:

1. **Check database contents**.  
   Run this in your FastAPI shell or create a debug route:

python

CopyEdit

@app.get("/debug/users") async def debug\_users(): return await User.all().values("id", "username", "email")

Then call /debug/users in your browser or with Postman to see what's already in the database.

1. Try with **very unique** credentials, like:

vbnet

CopyEdit

username: "testuser\_98423"

email: "unique\_98423@example.com"

**🔸 Possibility 2: Your**User**model has uniqueness constraints on both**username**and**email**, but your pre-check only validates**username**.**

In this case, if the **email** is already in the database, the pre-check passes, but User.create(...) will fail with IntegrityError.

That’s what triggers:

python

CopyEdit

raise HTTPException(status\_code=400, detail="Username or Email already exists")

**✅ Recommended Fix**

Add a pre-check for email as well:

python

CopyEdit

if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists")

This gives **clearer errors** before hitting the DB constraint.

**✅ Double Check: Your**User**model**

Make sure your User model has this (or similar):

python

CopyEdit

class User(Model): username = fields.CharField(max\_length=20, unique=True) email = fields.CharField(max\_length=100, unique=True) ...

Would you like help adding these pre-checks or testing a clean registration with a fresh database?

**You said:**

lets explore possibility 2 explain to me in detail

**ChatGPT said:**

Great — let’s **dive deeper into Possibility 2**: Your registration flow always returns "account already exists" **because you're only checking if the username exists**, but **not the email** — and your User model likely enforces that emails must be unique as well.

**🔍 Detailed Breakdown of What’s Going Wrong**

**✅ Step-by-Step of Your Existing Flow**

Here’s your code:

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Check only the username if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists")

**⚠️ Problem**

Let’s say:

* You try to register with:

json

CopyEdit

{ "username": "john123", "email": "john@example.com" }

* And maybe **another user already exists with**:

json

CopyEdit

{ "username": "alice99", "email": "john@example.com" <-- duplicate email }

**❌ What Happens?**

* ✅ Your code checks if john123 already exists → it **doesn't**, so it moves forward.
* ❌ It **does not check if the email is already taken**.
* 🚨 Then await User.create(...) tries to insert the new user.
* ❗ Because the email is set to unique=True in your database schema, the database throws an IntegrityError.
* 🧨 Your app catches that and returns:

arduino

CopyEdit

HTTP 400: Username or Email already exists

So, **even with a fresh username**, if the email is already taken, you’ll get this generic error, which is confusing if you're only testing the username field.

**✅ Solution**

You should pre-check **both** fields before trying to create the user.

**✔ Updated Registration Code**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Check if username already exists if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # Check if email already exists if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: user\_obj = await User.create(\*\*user\_info) except IntegrityError: raise HTTPException(status\_code=400, detail="Something went wrong during registration") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." }

**🧪 How to Confirm This Works**

1. Start with a fresh test user:
   * Username: testuser\_123
   * Email: test\_123@example.com
2. Try submitting again with:
   * Same email but different username → should get "Email already exists".
3. Try same username, different email → should get "Username already exists".
4. Use **both unique** values → should register successfully.

**📦 Bonus Tip**

For even cleaner UX, return **both conflicts at once**:

python

CopyEdit

errors = [] if await User.filter(username=user\_info["username"]).exists(): errors.append("Username already exists") if await User.filter(email=user\_info["email"]).exists(): errors.append("Email already exists") if errors: raise HTTPException(status\_code=400, detail=errors)

This way users get a **complete list of issues** in one response.

Would you like help testing this with a debug route or checking your current database contents?

**You said:**

ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 116, in email\_verification return templates.TemplateResponse("verification.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates' INFO: 127.0.0.1:59288 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjcsInVzZXJuYW1lIjoicm9ubmllIn0.t72Yh\_bFcLphmZ9wTzycyi25SEu7\_bnN4ZVzF6ukAz8 HTTP/1.1" 500 Internal Server Error ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 110, in email\_verification return templates.TemplateResponse("already\_verified.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'already\_verified.html' not found in search path: 'templates' this is showing

**ChatGPT said:**

Thanks for sharing the error. This is very clear: your application is trying to render a Jinja2 template called already\_verified.html, but that file **does not exist** in your templates/ directory. That’s why you're getting this error:

pgsql

CopyEdit

jinja2.exceptions.TemplateNotFound: 'already\_verified.html' not found in search path: 'templates'

**✅ What You Need to Do**

You **must create** the file already\_verified.html and place it in your templates directory (same location as verification.html).

**📄 Sample**already\_verified.html

Here’s a basic template you can copy:

html

CopyEdit

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta http-equiv="X-UA-Compactible" content="IE=edge"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Email Already Verified</title> <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"> </head> <body> <div class="container"> <div class="row" style="margin-top: 30vh;"> <div class="col-md-10 col-sm-10 col-xm-12 m-auto p-4"> <div class="card text-center"> <div class="card-header"> Email Verification </div> <div class="card-body"> <h5 class="card-title">Already Verified</h5> <p class="card-text"> Hello {{ username }}, your email address is already verified. </p> <span class="badge badge-success p-3">BigTree</span> </div> </div> </div> </div> </div> </body> </html>

**📁 File Placement**

Make sure your project has this structure:

css

CopyEdit

Ecommerce/

├── main.py

├── models.py

├── authentication.py

├── mail.py

├── templates/

│ ├── verification.html

│ └── already\_verified.html <--- ✅ place here

If templates/ is somewhere else, make sure Jinja2Templates(directory="...") points to the correct path.

**🧪 Test After Adding**

Once the file is in place:

1. Re-run your FastAPI app.
2. Visit the verification link again for a user who is already verified.
3. It should now render the already\_verified.html page correctly.

Let me know if you'd like to add a redirect instead of rendering a template.

**You said:**

whta will it show

**ChatGPT said:**

Here's what the already\_verified.html template I shared earlier will display in the browser when a user visits the email verification link **but their email is already verified**:

**💻 Page Content (Rendered View)**

**📍 Title:**

**"Email Verification"**

**🧑 Inside the Card:**

* **Header:** Email Verification
* **Title (inside card):** Already Verified
* **Message:**

*Hello ronnie, your email address is already verified.*  
(The {{ username }} gets replaced by the actual username.)

* **Badge:**  
  A blue "BigTree" badge appears below the message.

**📸 Visual Layout (Rough Sketch)**

markdown

CopyEdit

-----------------------------------------------------

| Email Verification |

-----------------------------------------------------

| [ Already Verified ] |

| |

| Hello ronnie, your email address is already |

| verified. |

| |

| [BigTree] (in a blue badge) |

-----------------------------------------------------

**✅ Summary**

This page is used to gracefully notify users that:

* Their email is already verified.
* They don’t need to click the verification link again.

It's part of good UX to confirm their status instead of showing an error or redirecting silently.

If you prefer a different message, color, or layout, I can help you tweak the HTML too.

**You said:**

ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 116, in email\_verification return templates.TemplateResponse("verification.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates' INFO: 127.0.0.1:59332 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjksInVzZXJuYW1lIjoicm9ubmkyIn0.--0C765eBmFromku2\_jzKtkA-W1b9W2tZR6KGDXM3Zw HTTP/1.1" 500 Internal Server Error ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 110, in email\_verification return templates.TemplateResponse("already\_verified.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'already\_verified.html' not found in search path: 'templates'

**ChatGPT said:**

Thanks for the detailed error log — it’s very clear. The issue you're running into is:

❌ jinja2.exceptions.TemplateNotFound: 'already\_verified.html'

**✅ What It Means**

Your FastAPI app is trying to render an HTML page using Jinja2 templates:

python

CopyEdit

templates.TemplateResponse("already\_verified.html", ...)

But it's **not able to find that template file** in the expected directory. Specifically, it’s looking inside the templates/ folder and not finding:

* already\_verified.html
* or possibly also verification.html (based on your earlier traceback).

**🛠️ Solution Steps**

1. **Create the**templates/**directory** (if it doesn't already exist):
   * Make sure it's in the **root of your FastAPI project** (i.e., same folder as main.py).
   * Example structure:

css

CopyEdit

Ecommerce/

├── main.py

├── templates/

│ ├── already\_verified.html

│ └── verification.html

1. **Place the required HTML files** in that folder:
   * Here’s a quick sample for already\_verified.html:

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Email Already Verified</title> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"> </head> <body class="d-flex justify-content-center align-items-center vh-100"> <div class="card p-4 shadow"> <h2 class="text-center text-success">Email Already Verified</h2> <p class="text-center">Hello {{ username }}, your email is already verified.</p> </div> </body> </html>

* + And a basic one for verification.html (which is shown after successful verification):

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Email Verified</title> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"> </head> <body class="d-flex justify-content-center align-items-center vh-100"> <div class="card p-4 shadow"> <h2 class="text-center text-primary">Email Verified</h2> <p class="text-center">Thank you {{ username }}, your email has been successfully verified.</p> </div> </body> </html>

1. **Verify your**Jinja2Templates**path is correct in**main.py**:**

In your code, somewhere you should have:

python

CopyEdit

from fastapi.templating import Jinja2Templates templates = Jinja2Templates(directory="templates")

This tells FastAPI to look in a folder named templates in the current working directory.

**🔁 Restart the Server**

After making these changes:

* Save all files.
* Restart the FastAPI server:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you want me to generate both templates again, or if you're still hitting issues after this fix.

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } templates = Jinja2Templates(directory="templates") @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module

**ChatGPT said:**

Thanks for sharing your complete main.py — the code looks structurally solid. Based on your setup, the email\_verification endpoint is rendering the following Jinja2 templates:

* ✅ verification.html — when the user's email is successfully verified
* ✅ already\_verified.html — when the user has already been verified

**❌ The Problem**

As your error log clearly shows:

bash

CopyEdit

jinja2.exceptions.TemplateNotFound: 'already\_verified.html'

FastAPI + Jinja2 is trying to find these templates in:

python

CopyEdit

templates = Jinja2Templates(directory="templates")

This means your directory structure **must include a folder named**templates**in the same directory as**main.py, and it **must contain**:

* already\_verified.html
* verification.html

**✅ Fix It in 2 Steps**

**1. 📁 Create the Templates Folder and Files**

Make sure your folder structure looks like this:

css

CopyEdit

Ecommerce/

├── main.py

├── models.py

├── authentication.py

├── mail.py

├── database.sqlite3

└── templates/

├── already\_verified.html

└── verification.html

**2. 📝 Add the HTML Templates**

Here’s an example of each:

**📄**templates/verification.html

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Verification Success</title> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"> </head> <body class="d-flex justify-content-center align-items-center vh-100"> <div class="card p-5 shadow text-center"> <h2 class="text-success">Email Verified</h2> <p>Thank you, <strong>{{ username }}</strong>. Your email has been successfully verified.</p> </div> </body> </html>

**📄**templates/already\_verified.html

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Already Verified</title> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"> </head> <body class="d-flex justify-content-center align-items-center vh-100"> <div class="card p-5 shadow text-center"> <h2 class="text-warning">Already Verified</h2> <p>Hey <strong>{{ username }}</strong>, your email is already verified.</p> </div> </body> </html>

**🔁 Final Step: Restart Your App**

After adding the files:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you want to customize the design, or integrate a redirect to a login/dashboard page after verification!

**You said:**

ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 116, in email\_verification return templates.TemplateResponse("verification.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates' INFO: 127.0.0.1:59376 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MzAsInVzZXJuYW1lIjoicm9ubmkzIn0.Hgqexndk6vFzPtuIqGsVvEEXSv50qfkUeKnO6nGKZ7s HTTP/1.1" 200 OK now this error shows up

**ChatGPT said:**

The error message:

pgsql

CopyEdit

jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates'

means that your FastAPI app cannot find the verification.html file in the folder you've set as your Jinja2 template directory.

**✅ To Fix This:**

**✅ 1. Verify your project structure**

Make sure your folder structure looks like this:

css

CopyEdit

Ecommerce/

├── main.py

├── ...

├── templates/

│ ├── verification.html

│ └── already\_verified.html

⚠️ The templates/ folder **must be in the same directory** as main.py, because you set:

python

CopyEdit

templates = Jinja2Templates(directory="templates")

If your folder is named anything else (like Templates, TEMPLATES, or it's nested inside another folder), **it won't work**. It must be exactly templates/ (case-sensitive on some systems).

**✅ 2. Check file name and extension**

Make sure:

* The file name is exactly: verification.html
* It's not accidentally saved as verification.html.txt or similar
* It's not empty or corrupted

**✅ 3. Sample**verification.html**(use this if you need one)**

You can paste this as a simple verification page:

html

CopyEdit

<!-- templates/verification.html --> <!DOCTYPE html> <html> <head> <title>Email Verified</title> </head> <body> <h1>Email Successfully Verified</h1> <p>Thank you, {{ username }}! Your email has been verified.</p> </body> </html>

Do the same for already\_verified.html.

**✅ 4. Restart the Server**

After adding the template, restart your app to clear cache and reload the files:

bash

CopyEdit

uvicorn main:app --reload

**🔁 If You're Still Stuck**

Run this diagnostic snippet in your main.py:

python

CopyEdit

import os print("Templates dir exists:", os.path.isdir("templates")) print("Files in templates dir:", os.listdir("templates"))

This will tell you whether the folder and files are actually visible to Python.

Let me know what the output is, or feel free to upload a screenshot of your folder layout if you'd like me to double-check it visually.

**You said:**

ERROR: Exception in ASGI application Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\protocols\http\httptools\_impl.py", line 409, in run\_asgi result = await app( # type: ignore[func-returns-value] ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\middleware\proxy\_headers.py", line 60, in \_\_call\_\_ return await self.app(scope, receive, send) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\applications.py", line 1054, in \_\_call\_\_ await super().\_\_call\_\_(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\applications.py", line 112, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 187, in \_\_call\_\_ raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\errors.py", line 165, in \_\_call\_\_ await self.app(scope, receive, \_send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\middleware\exceptions.py", line 62, in \_\_call\_\_ await wrap\_app\_handling\_exceptions(self.app, conn)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 714, in \_\_call\_\_ await self.middleware\_stack(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 734, in app await route.handle(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 288, in handle await self.app(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 76, in app await wrap\_app\_handling\_exceptions(app, request)(scope, receive, send) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 53, in wrapped\_app raise exc File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\\_exception\_handler.py", line 42, in wrapped\_app await app(scope, receive, sender) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\routing.py", line 73, in app response = await f(request) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 301, in app raw\_response = await run\_endpoint\_function( ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\fastapi\routing.py", line 212, in run\_endpoint\_function return await dependant.call(\*\*values) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 116, in email\_verification return templates.TemplateResponse("verification.html", ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 208, in TemplateResponse template = self.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\starlette\templating.py", line 131, in get\_template return self.env.get\_template(name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 1016, in get\_template return self.\_load\_template(name, globals) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\environment.py", line 975, in \_load\_template template = self.loader.load(self, name, self.make\_globals(globals)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 126, in load source, filename, uptodate = self.get\_source(environment, name) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\jinja2\loaders.py", line 209, in get\_source raise TemplateNotFound( jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates' INFO: 127.0.0.1:59427 - "GET /verification?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MzIsInVzZXJuYW1lIjoicm9ubmk1In0.NbTDZvTcpxEgvk\_Sj4rCzOqdCv5TtxC3pMjKRmStVSY HTTP/1.1" 200 OK

**ChatGPT said:**

You're still getting this error:

pgsql

CopyEdit

jinja2.exceptions.TemplateNotFound: 'verification.html' not found in search path: 'templates'

**✅ Let’s resolve it step by step.**

**✅ 1. Ensure the Folder Exists in the Right Location**

Your main.py is located in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\

You **must** have a folder here named templates (lowercase), like this:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\

├── main.py

├── templates\

├── verification.html

└── already\_verified.html

⚠️ templates folder **must be a sibling** of main.py. Not inside another folder like app/templates, or src/templates.

**✅ 2. Create the HTML File**

Create this file exactly as shown:

**Path:**  
D:\Data Structures and Algorithms\Ecommerce\templates\verification.html

**Content (example):**

html

CopyEdit

<!DOCTYPE html> <html> <head> <title>Email Verified</title> </head> <body> <h1>Email Successfully Verified</h1> <p>Thank you, {{ username }}! Your email has been verified.</p> </body> </html>

Do the same for already\_verified.html.

**✅ 3. (Optional) Debug Folder Programmatically**

Add this test code temporarily in main.py:

python

CopyEdit

import os @app.get("/debug") def debug(): return { "current\_dir": os.getcwd(), "template\_dir\_exists": os.path.isdir("templates"), "template\_files": os.listdir("templates") if os.path.isdir("templates") else [] }

Then visit:  
http://127.0.0.1:8000/debug

You should see something like:

json

CopyEdit

{ "current\_dir": "D:\\Data Structures and Algorithms\\Ecommerce", "template\_dir\_exists": true, "template\_files": ["verification.html", "already\_verified.html"] }

**✅ Final Checklist**

* Folder name is **exactly** templates
* Folder is in same dir as main.py
* HTML files are named **exactly** as in the code
* HTML file is not .txt or empty
* You restarted the server after making changes

Let me know what the /debug output is if it still fails.

**You said:**

can there be something else i have done all this

**ChatGPT said:**

If you’ve already done all the following:

* ✅ templates/ folder exists in the same directory as main.py
* ✅ verification.html and already\_verified.html exist in that folder
* ✅ File names are spelled correctly (case-sensitive on Linux/macOS!)
* ✅ Server was restarted

...and you **still** get:

bash

CopyEdit

jinja2.exceptions.TemplateNotFound: 'verification.html'

Then here's what else could be wrong **and how to fix it**:

**🔍 1. Check How You're Initializing**Jinja2Templates

You should have this line exactly like this:

python

CopyEdit

from fastapi.templating import Jinja2Templates templates = Jinja2Templates(directory="templates")

Make sure:

* directory="templates" is not a relative path pointing to the wrong place
* You're not using something like directory="./app/templates" (unless that's actually where your HTML files are)

**🧪 2. Print the Template Search Path**

Add this debug route temporarily:

python

CopyEdit

from starlette.templating import Jinja2Templates @app.get("/debug/templates") def debug\_templates(): return { "template\_dir": templates.directory }

Visit http://127.0.0.1:8000/debug/templates

You should see:

json

CopyEdit

{ "template\_dir": "templates" }

If it’s pointing to a different folder, adjust the path like:

python

CopyEdit

import os templates = Jinja2Templates(directory=os.path.join(os.path.dirname(\_\_file\_\_), "templates"))

**🧪 3. Make Sure You’re Not Inside a Package Directory**

If your structure is like:

css

CopyEdit

project/

├── app/

│ ├── main.py

│ └── templates/

│ └── verification.html

Then use:

python

CopyEdit

templates = Jinja2Templates(directory="app/templates")

or better (dynamic path):

python

CopyEdit

import os BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates"))

**🧪 4. Verify the File Extension**

Make sure it's verification.html, **not**:

* verification.html.txt
* Verification.html
* verfication.html (typo!)

Use this quick check:

bash

CopyEdit

dir templates

Or in Python:

python

CopyEdit

import os print(os.listdir("templates"))

**🧪 5. Permissions Issue (Rare on Windows)**

Make sure the file is readable. Try opening it in a basic Python script:

python

CopyEdit

with open("templates/verification.html", "r") as f: print(f.read())

If this fails, you’ve got a permission or path issue.

**✅ Summary**

If all else fails:

* Show me the **exact folder structure** (you can copy-paste the tree)
* Show the result of os.getcwd() and os.listdir("templates")
* Or just paste the output from /debug and /debug/templates

This will let me pinpoint the problem right away.

**You said:**

D:\Data Structures and Algorithms\Ecommerce this is my path where main module is located

**ChatGPT said:**

Thanks! Based on your main module being in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce

Let’s make sure your templates directory is in the **right place** and that FastAPI/Jinja2 can find it.

**✅ Your Project Should Look Like This**

css

CopyEdit

Ecommerce/

├── main.py

├── templates/

│ ├── verification.html

│ └── already\_verified.html

Make sure:

* templates/ is in the **same folder** as main.py
* The .html files are inside that folder

**✅ Your Code Should Initialize Templates Like This**

In main.py:

python

CopyEdit

from fastapi.templating import Jinja2Templates import os BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates"))

This avoids issues with working directory mismatches — especially on Windows or when running through VS Code or other IDEs.

**🔍 Add This Test Route**

This will help confirm your app is looking in the right folder:

python

CopyEdit

@app.get("/debug") def debug(): import os return { "cwd": os.getcwd(), "template\_dir": templates.directory, "template\_files": os.listdir(templates.directory) }

Then visit: http://127.0.0.1:8000/debug

Make sure:

* template\_dir ends with Ecommerce\templates
* template\_files includes verification.html and already\_verified.html

**🛠️ Final Notes**

If you’re running the server from **outside** the Ecommerce folder (e.g., from the parent directory), that can cause relative paths to fail. Use the updated BASE\_DIR path logic above to fix that once and for all.

Let me know what the /debug route returns if you’re still stuck — I can tell you exactly what’s wrong from that.

**You said:**

ok it was successful but the web page is blank

**ChatGPT said:**

Great! If the error is gone but the webpage is **blank**, that means:

✅ The verification.html template is found  
❌ But it's either **empty** or not rendering content correctly

**✅ Step-by-Step Fix**

**1. Open**templates/verification.html

Make sure it has **valid HTML content**, for example:

html

CopyEdit

<!-- templates/verification.html --> <!DOCTYPE html> <html> <head> <title>Verification Successful</title> </head> <body> <h1>Email Verified Successfully!</h1> <p>Welcome, {{ username }}!</p> </body> </html>

**2. Also check**already\_verified.html

html

CopyEdit

<!-- templates/already\_verified.html --> <!DOCTYPE html> <html> <head> <title>Already Verified</title> </head> <body> <h1>Hello, {{ username }}</h1> <p>Your email is already verified.</p> </body> </html>

**3. Restart the server**

Sometimes changes to .html templates aren’t picked up live. Restart FastAPI with:

bash

CopyEdit

uvicorn main:app --reload

**4. Test Again**

Try clicking the verification link again or access it manually:

arduino

CopyEdit

http://127.0.0.1:8000/verification?token=your\_token\_here

Let me know what you see after updating the HTML — if it's still blank, I can help debug that too.

**You said:**

ok it is done thnaks

**ChatGPT said:**

You're welcome — I'm glad it's working now! 🎉  
If you run into any more issues or need help with features like login, password reset, or deployment, feel free to reach out.

Happy coding and good luck with your eCommerce project! 🚀🛒

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

22-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is the main moadule of a project explain this to me in detail

**ChatGPT said:**

This FastAPI application is a **user registration and authentication system** that includes email verification, token-based authentication using JWT (JSON Web Tokens), and automatic business creation upon user registration. Let's walk through the code **section-by-section** in detail:

**✅ Imports**

python

CopyEdit

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \*

* **FastAPI**: Web framework for building APIs.
* **Tortoise ORM**: Async ORM used here for database models and operations.
* models: Presumably contains User, Business, and Pydantic schemas.
* Depends: Used for dependency injection.

python

CopyEdit

from authentication import \* from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm

* Authentication logic (like token generation, hashing, JWT decoding) is abstracted into the authentication.py module.
* OAuth2 is used for user login with **bearer tokens**.

python

CopyEdit

from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email

* post\_save is a Tortoise signal to perform actions after saving a model (e.g., auto-create a business).
* send\_email: Function to send verification emails.

python

CopyEdit

from fastapi.responses import HTMLResponse from fastapi.templating import Jinja2Templates

* Used to return HTML responses with dynamic data using **Jinja2 templates**.

**✅ Application Setup**

python

CopyEdit

app = FastAPI() BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates"))

* Initialize the FastAPI app.
* Configure the path for email verification HTML templates.

**🔐 Token Generation (Login Endpoint)**

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"}

* Accepts form data for username and password.
* Returns a **JWT token** if authentication is successful.

**🔍 Current User Dependency**

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException(...) return await user

* Decodes JWT to identify user and retrieve them from the database.
* If the token is invalid or expired, raises an HTTP 401 error.

**👤 "Me" Endpoint**

python

CopyEdit

@app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } }

* Requires a valid JWT token.
* Returns user profile info and business data.
* user\_pydanticIn is a schema used to serialize the User object.

**🏢 Post-Save Signal: Create Business + Send Email**

python

CopyEdit

@post\_save(User) async def create\_business(...): if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) await send\_email([instance.email], instance)

* Automatically called after a User is created.
* Creates a related Business object.
* Sends a verification email.

**📝 User Registration Endpoint**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): ... if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") ... user\_info["password"] = get\_hashed\_password(user\_info["password"]) user\_obj = await User.create(\*\*user\_info) new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return {"status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services."}

* Validates uniqueness of username and email.
* Hashes the password.
* Creates a new User.
* Returns a welcome message.

**📧 Email Verification Page**

python

CopyEdit

@app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: return templates.TemplateResponse("already\_verified.html", {...}) else: user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {...}) raise HTTPException(...)

* Accepts a token from a verification link.
* Verifies the user's email.
* Serves an HTML response via Jinja2.

**🏠 Home Endpoint**

python

CopyEdit

@app.get("/") def index(): return {"Message": "Hello World"}

* Basic home route. Good for testing API status.

**🔧 Database Configuration with Tortoise ORM**

python

CopyEdit

register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True )

* Initializes Tortoise ORM with a **SQLite database**.
* Automatically generates schemas based on models.

**🔚 Summary**

This FastAPI project does the following:

| **Feature** | **Description** |
| --- | --- |
| **User Registration** | Secure sign-up with duplicate checks and password hashing |
| **Login with JWT** | Returns access token using OAuth2 scheme |
| **Get Current User** | Uses JWT token to identify and authenticate users |
| **Business Auto-Creation** | Uses post\_save signal to create a business record after user creation |
| **Email Verification** | Sends verification emails and confirms them via a browser |
| **Tortoise ORM Integration** | Async ORM for clean, simple DB interaction |

Would you like a diagram or database schema overview as well?

**You said:**

explain to me how token generation work

**ChatGPT said:**

Great question! Let's walk through **how token generation works** in your FastAPI project using **JWT (JSON Web Tokens)**. Here's a step-by-step breakdown of the logic and flow behind it.

**📍 Overview of Token Generation**

The /token endpoint issues an **access token** after a user logs in with a valid username and password. The token is a **JWT** that securely encodes user information (like their ID) and is later used to authenticate API requests.

**📦 Related Code**

**Token Endpoint**

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"}

* This uses FastAPI's OAuth2PasswordRequestForm, which expects form data (username and password).
* It calls a function token\_generator to create the token.
* Returns a JSON response with the token.

**Likely**token\_generator**Function (from**authentication.py**)**

python

CopyEdit

from datetime import datetime, timedelta from jose import jwt from passlib.context import CryptContext from models import User pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto") SECRET\_KEY = "your-secret-key" # Should be in env/config ALGORITHM = "HS256" ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30 async def token\_generator(username: str, password: str): user = await User.get(username=username) if not pwd\_context.verify(password, user.password): raise Exception("Invalid credentials") payload = { "id": user.id, "exp": datetime.utcnow() + timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES), "sub": user.username } token = jwt.encode(payload, SECRET\_KEY, algorithm=ALGORITHM) return token

**🔄 Step-by-Step Flow**

1. **User Logs In**
   * Sends a POST request to /token with username and password.
2. **FastAPI Handles the Form**
   * OAuth2PasswordRequestForm extracts the username and password.
3. **Fetch the User from DB**

python

CopyEdit

user = await User.get(username=username)

* + The app looks up the user in the database.

1. **Verify Password**

python

CopyEdit

if not pwd\_context.verify(password, user.password): raise Exception("Invalid credentials")

* + Compares the input password with the hashed password stored in the DB using bcrypt.

1. **Generate JWT Token**

python

CopyEdit

payload = { "id": user.id, "exp": datetime.utcnow() + timedelta(minutes=30), "sub": user.username } token = jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

* + Creates a token with:
    - id: user ID (used to identify them later)
    - exp: expiration time (adds security)
    - sub: subject, commonly the username

1. **Return the Token**
   * Returns the token to the client in this format:

json

CopyEdit

{ "access\_token": "eyJhbGciOiJIUzI1NiIsInR...", "token\_type": "bearer" }

1. **Client Stores Token**
   * The frontend stores it (e.g., in localStorage or memory) and includes it in headers for future requests:

makefile

CopyEdit

Authorization: Bearer <token>

**🛡 Why Use JWT?**

* **Stateless**: No need to store session data on the server.
* **Secure**: Signed with a secret key, making it hard to tamper with.
* **Self-contained**: All user data needed for auth is inside the token.

**📥 How Token is Used Later**

For example, in get\_current\_user():

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id"))

* FastAPI will extract the token from the Authorization header.
* Decode it to retrieve user info.
* Load the user and give you access to authenticated routes.

**🔁 Summary**

| **Step** | **Action** |
| --- | --- |
| User submits credentials | /token endpoint handles form |
| Verify password | Using bcrypt, compare hashed password |
| Generate JWT | Include id, exp, and username |
| Send back token | Client receives it in response |
| Use in later requests | Sent in the Authorization: Bearer <token> header for protected routes |

Would you like me to help you **add token refresh support** or explain **JWT expiration handling** next?

**You said:**

from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: # This goes LAST raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): user = await User.get(username = username) if user and await verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token thisis my authentication module explain to in this context

**ChatGPT said:**

Great! This is your **authentication module**, and it handles password hashing, user authentication, JWT token generation, and token verification. Let's walk through each function and explain its role in the **authentication flow** within your FastAPI app.

**🔐 1.**get\_hashed\_password(password)

python

CopyEdit

def get\_hashed\_password(password): return pwd\_context.hash(password)

**✅ Purpose:**

* Takes a **plain-text password** and returns a **hashed version** using bcrypt.
* Called during user registration to securely store passwords in the database.

**🔒 Why Hash?**

* Passwords should never be stored in plain text.
* Even if the DB is leaked, hashed passwords are much harder to crack.

**🔍 2.**very\_token(token)

python

CopyEdit

async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) ... return user

**✅ Purpose:**

* **Decodes a JWT** to find the user ID.
* Checks if that user still exists.
* If token is invalid or expired, raises appropriate HTTP exceptions.

**🔑 Use Case:**

* Used in **email verification** and **auth-protected routes** to verify a token is still valid and get the associated user.

**🔑 3.**verify\_password(plain\_password, hashed\_password)

python

CopyEdit

async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password)

**✅ Purpose:**

* Verifies whether a plain password matches the stored hashed password.
* Called during **login authentication**.

**👤 4.**authenticate\_user(username, password)

python

CopyEdit

async def authenticate\_user(username, password): user = await User.get(username=username) if user and await verify\_password(password, user.password): return user return False

**✅ Purpose:**

* Checks:
  1. Whether the username exists.
  2. Whether the password is correct (via verify\_password).
* Returns the user if valid, or False if invalid.

**🔁 Used By:**

* The token\_generator() function to validate credentials before issuing a token.

**🪙 5.**token\_generator(username, password)

python

CopyEdit

async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException(...) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token

**✅ Purpose:**

* Authenticates the user.
* If valid, creates a **JWT token** containing:
  + id: user ID
  + username: username
* Uses jwt.encode with a secret key.

**📦 Result:**

* A signed token like eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
* Sent back to the user upon login via the /token endpoint.

**🔁 Complete Flow**

Here’s how these functions **work together**:

| **Step** | **What Happens** | **Function(s)** |
| --- | --- | --- |
| 📝 User Registers | Password is hashed and stored | get\_hashed\_password |
| 🔐 User Logs In | Credentials validated | authenticate\_user, verify\_password |
| 🎫 Token Issued | If valid, a JWT is returned | token\_generator |
| 🔍 Token Verified | For protected or email-verification routes | very\_token |

**📌 Summary of Components**

| **Function** | **Responsibility** |
| --- | --- |
| get\_hashed\_password | Securely hash a plain password |
| verify\_password | Compare plain and hashed passwords |
| authenticate\_user | Validate credentials from login form |
| token\_generator | Create a JWT if login is successful |
| very\_token | Decode and verify JWT, fetch the user |

**🔐 Security Notes:**

* JWT tokens should **always include an expiration** (exp). Your current token\_generator is missing this (add it for security).
* Store SECRET in a .env file and load with dotenv\_values(), which you're doing correctly with config\_credentials.

Would you like help modifying token\_generator() to include token expiration or refresh tokens?

**You said:**

explain to me the whole login process in order

**ChatGPT said:**

Sure! Here's a complete, step-by-step explanation of the **login process** in your FastAPI project — from the moment a user submits their credentials to the system verifying them and returning a token.

**🔁 LOGIN FLOW: Step-by-Step**

**🧑‍💻 1. User Submits Login Form**

**Client (frontend or Postman):**

* Makes a POST request to the /token endpoint.
* Sends username and password as **form data** (not JSON!).

Example request:

h

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=johndoe&password=secret123

**🧰 2. FastAPI Handles the Request**

In your FastAPI app:

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"}

* FastAPI uses OAuth2PasswordRequestForm to parse username and password from the form.
* Calls token\_generator() with those values.

**✅ 3. Validate Credentials**

In authentication.py:

python

CopyEdit

async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException(...)

* Calls authenticate\_user() to verify the credentials.

**🔍 4. Find and Verify User**

python

CopyEdit

async def authenticate\_user(username, password): user = await User.get(username=username) if user and await verify\_password(password, user.password): return user return False

* User.get() queries the database for a user with the given username.
* verify\_password() checks if the given plain password matches the **hashed** password stored in the DB.

**Password Check Logic:**

python

CopyEdit

async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password)

* Uses bcrypt to compare passwords securely.

**🪙 5. Generate JWT Token**

python

CopyEdit

token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"])

* Encodes a JWT with:
  + id: user ID (used to identify the user in protected routes)
  + username: useful for frontend/UI
* Signed with your secret key (config\_credentials["SECRET"]).

Example token payload (decoded):

json

CopyEdit

{ "id": 5, "username": "johndoe" }

**📦 6. Return the Token**

python

CopyEdit

return {"access\_token": token, "token\_type": "bearer"}

* This token is returned to the user.
* It’s typically stored in localStorage, sessionStorage, or memory on the frontend.

**🔐 7. User Sends Token with Future Requests**

Client includes token in Authorization header:

makefile

CopyEdit

Authorization: Bearer <access\_token>

FastAPI uses get\_current\_user() dependency to extract and decode it:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id"))

This verifies the token and fetches the corresponding user for any protected route.

**✅ Summary Flow Diagram**

text

CopyEdit

[Client] ---> POST /token (username & password) | v [FastAPI /token route] --> token\_generator() | v [Authenticate user] --> check DB & password hash | v [JWT created] --> Signed with SECRET key | v [Response] ---> { "access\_token": "JWT", "token\_type": "bearer" } (Client stores token and uses it in headers for all future requests)

**🔒 Best Practices to Add**

1. ✅ **Add Expiration**:  
   Add exp claim to token so it auto-expires after a time window.

Example:

python

CopyEdit

from datetime import datetime, timedelta ... token\_data = { "id": user.id, "username": user.username, "exp": datetime.utcnow() + timedelta(minutes=30) }

1. ✅ **Use HTTPS** in production to protect tokens in transit.
2. ✅ **Blacklist or Refresh**: Consider refresh tokens and blacklisting for more secure logout.

Would you like help modifying your code to include token expiration or refresh tokens next?

**You said:**

in encoding is it not needed to pass algorithms

**ChatGPT said:**

Great catch — and you're absolutely right to ask this!

**🔐 Short Answer:**

Yes, you **should** explicitly pass the algorithm when encoding a JWT using the jwt.encode() function.

**✅ Correct Usage of**jwt.encode()

You are using the jwt module (likely from PyJWT, i.e., import jwt), which expects:

python

CopyEdit

jwt.encode(payload, secret, algorithm="HS256")

**❌ Incorrect (missing algorithm):**

python

CopyEdit

token = jwt.encode(token\_data, config\_credentials["SECRET"]) # No algorithm!

* This may work **in older versions** of PyJWT **because**HS256**was the default**, but:
  + It's **not safe to rely on defaults**.
  + It may **fail or throw a warning** in newer versions.
  + It can lead to **inconsistent decoding** later.

**✅ Correct:**

python

CopyEdit

token = jwt.encode(token\_data, config\_credentials["SECRET"], algorithm="HS256")

This ensures:

* The token is signed using the **HS256** algorithm.
* It can be correctly and securely **verified** with jwt.decode() (which already uses algorithms=["HS256"] in your code).

**🔁 Consistency in Your Code**

You already use algorithm="HS256" when decoding:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

So to stay consistent and safe, you should also specify it on **encoding**.

**🧠 Final Tip:**

If you’re using a library like python-jose instead of PyJWT, it’s the same idea — always **explicitly pass the algorithm** when encoding.

**✅ Action Item:**

Update this in your token\_generator():

python

CopyEdit

token = jwt.encode(token\_data, config\_credentials["SECRET"], algorithm="HS256")

Let me know if you'd like help adding expiration time (exp) too.

**You said:**

in main.py what is the role of this line oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token")

**ChatGPT said:**

Great question! This line in your main.py:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

is a **crucial part of the authentication system** in FastAPI using **OAuth2 with bearer tokens** (i.e., JWTs). Let’s break it down in detail:

**🧠 What Is**OAuth2PasswordBearer**?**

OAuth2PasswordBearer is a **dependency class** provided by FastAPI that tells the app:

“Expect the client to send a bearer token (like a JWT) in the Authorization header for protected endpoints.”

Example header:

makefile

CopyEdit

Authorization: Bearer <access\_token>

**🔍 What Does**tokenUrl="token"**Mean?**

This value is used **only for documentation and OpenAPI (Swagger UI)**. It tells FastAPI:

* Where the user must go to **get a token**.
* In your case, this is the /token endpoint.

So in Swagger UI, when you click “Authorize”, it knows that it should prompt for username & password and send them to /token to get the access token.

**📦 Role in Code**

Here’s how it's used:

**Step 1: Define the scheme**

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

This returns a **dependency** that extracts the token from the Authorization header.

**Step 2: Use it in a function**

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)): ...

This does 2 things:

1. Automatically pulls the Bearer <token> from the HTTP request headers.
2. Passes just the token string (token) to your get\_current\_user() function.

**🔐 Real-world Example**

Say you call a protected endpoint like /user/me:

h

CopyEdit

GET /user/me Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

FastAPI:

* Uses oath2\_scheme to extract the token.
* Passes it into get\_current\_user(token: str).
* get\_current\_user decodes the JWT and returns the user.

**🧾 Summary**

| **Part** | **Purpose** |
| --- | --- |
| OAuth2PasswordBearer | Extracts the token from the Authorization header |
| tokenUrl="token" | Used in Swagger UI to guide where to get a token |
| Depends(oath2\_scheme) | Used in dependencies to inject the token into your function |

**✅ Without This Line**

If you skip defining oath2\_scheme, you can't extract the JWT from incoming requests easily, and your whole protected route system breaks.

Let me know if you'd like a quick visual flow of this or to see a working Swagger UI example!

**You said:**

explain the token endpoint in detail

**ChatGPT said:**

Absolutely! Let's break down the /token endpoint in your FastAPI app step-by-step and explain each part in detail.

**🧾 Your Token Endpoint Code**

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"}

**🔍 What's the Purpose of This Endpoint?**

This endpoint is responsible for **user login**. It:

1. **Accepts credentials** (username and password).
2. **Authenticates the user**.
3. **Generates a JWT token** if valid.
4. **Returns the token** so the client can use it for protected routes.

**🔢 Detailed Breakdown**

**1.**@app.post("/token")

* This sets up an HTTP POST route at /token.
* This is your **login endpoint** — used to get an access token.

**2.**request\_form: OAuth2PasswordRequestForm = Depends()

* FastAPI dependency injection.
* Tells FastAPI to expect a form submission using:
  + username
  + password
* These are sent as application/x-www-form-urlencoded (like a typical login form).

🧠 Example request body:

ini

CopyEdit

username=johndoe&password=secret123

**3.**token\_generator(request\_form.username, request\_form.password)

* Calls your custom function token\_generator() with the provided credentials.
* This function:
  1. Authenticates the user.
  2. If valid, generates a **JWT token** signed with a secret key.
  3. Returns that token string.

✅ If authentication fails, token\_generator() raises an HTTPException.

**4. Return Token**

python

CopyEdit

return {"access\_token": token, "token\_type": "bearer"}

* The token is returned in JSON format.
* access\_token: the actual JWT.
* token\_type: usually "bearer" (this is a standard format).

📦 Example response:

json

CopyEdit

{ "access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "token\_type": "bearer" }

This is what the frontend or API client will use in the Authorization header like this:

makefile

CopyEdit

Authorization: Bearer eyJhbGciOi...

**🔐 OpenAPI / Swagger Integration**

Because you defined:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

* Swagger UI understands that /token is your login URL.
* You’ll see an "Authorize" button in the docs where you can input username/password and get a token.

**🔁 Summary of the**/token**Endpoint**

| **Step** | **Action** |
| --- | --- |
| 1 | Accepts username and password as form data |
| 2 | Validates credentials using token\_generator() |
| 3 | Creates a JWT token on success |
| 4 | Returns token in a JSON response |
| 5 | Token is used to access protected routes |

**🛡️ Final Note on Security**

You should **always** use HTTPS in production to keep tokens safe during transmission.

Would you like help modifying your token system to include expiration or refresh tokens?

**You said:**

Depends() what does this do

**ChatGPT said:**

Great question! Depends() is **one of the most important building blocks in FastAPI**. It's part of FastAPI's **dependency injection system** — a powerful and clean way to **reuse and share logic across routes** like authentication, database sessions, validations, etc.

**🧠 What is**Depends()**?**

**🔧 Definition:**

Depends() tells FastAPI to:

Automatically **call another function** and **pass its return value** as a parameter to your endpoint function.

**✅ Example: Authentication**

In your code:

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()):

**What's Happening Here?**

* FastAPI sees Depends(OAuth2PasswordRequestForm) and:
  + **Reads the request body**.
  + **Extracts** the username and password form fields.
  + Creates an instance of OAuth2PasswordRequestForm with those values.
  + Passes that instance as request\_form.

So request\_form.username and request\_form.password are ready to use!

**🔐 Another Common Use: Authentication Dependency**

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)):

* Here, Depends(oath2\_scheme) extracts the JWT token from the request headers.
* It automatically runs the logic defined in OAuth2PasswordBearer.
* Passes the token into get\_current\_user().

**🔁 Summary Table**

| Depends(...)**Used On** | **What It Does** |
| --- | --- |
| OAuth2PasswordRequestForm | Parses login form fields (username, password) |
| oath2\_scheme | Extracts Authorization header and returns the token |
| Any custom function (e.g., get\_current\_user) | Runs that function and injects its return value |

**📦 You Can Use Depends With:**

* **Classes** (like OAuth2PasswordRequestForm)
* **Functions** (like get\_current\_user)
* **Other dependencies**, forming chains (nesting logic)

**🧪 Quick Visual Example**

python

CopyEdit

def get\_token(): return "123ABC" @app.get("/test") def read\_data(token: str = Depends(get\_token)): return {"token": token}

* GET /test will return:

json

CopyEdit

{ "token": "123ABC" }

Because get\_token() was automatically called and its result was injected.

**🔥 Final Tip:**

Using Depends():

* Keeps your route functions clean
* Makes your code modular
* Enables features like **auth**, **middleware-like logic**, and **reusability**

Would you like to see how to chain multiple Depends() to validate both token and permissions?

**You said:**

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): here what is the function that it is calling

**ChatGPT said:**

Excellent follow-up! Let's look at this line again:

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()):

You're asking:

🔍 **"What function is**Depends()**actually calling here?"**

**✅ Answer:**

FastAPI is **automatically calling the**\_\_call\_\_**method** of the OAuth2PasswordRequestForm class behind the scenes.

More simply:

* OAuth2PasswordRequestForm is **not just a class**, it acts like a function because it implements the \_\_call\_\_() method.
* So when you do Depends(OAuth2PasswordRequestForm), FastAPI is really doing:

python

CopyEdit

request\_form = await OAuth2PasswordRequestForm().\_\_call\_\_(request)

You don't see this directly — FastAPI handles this internally.

**🔍 What Does**OAuth2PasswordRequestForm**Do?**

This class is defined in FastAPI like this:

python

CopyEdit

class OAuth2PasswordRequestForm: def \_\_init\_\_(self, ...): # FastAPI will extract form fields from the request body ... async def \_\_call\_\_(self, request: Request): # It reads the form data from the request form = await request.form() self.username = form.get("username") self.password = form.get("password") return self

So effectively:

* FastAPI **calls the class** with the current HTTP request.
* It **reads the form data** (username, password).
* Then it **injects** the OAuth2PasswordRequestForm instance into your route.

**🧪 Result in Your Endpoint**

python

CopyEdit

request\_form.username # The username from form request\_form.password # The password from form

You can now use these in your token generation logic.

**🔁 Summary**

| **What You Wrote** | **What FastAPI Internally Does** |
| --- | --- |
| request\_form: OAuth2PasswordRequestForm = Depends() | Calls OAuth2PasswordRequestForm().\_\_call\_\_() with the request |
| request\_form.username | Extracted from form data (e.g., "username=admin") |
| request\_form.password | Extracted from form data (e.g., "password=1234") |

Let me know if you’d like to see a custom implementation of this for full control over form parsing or authentication logic!

**You said:**

explain the authentication dependency again

**ChatGPT said:**

Absolutely! Let's walk through the **authentication dependency** again — slowly and clearly — so you fully understand how it works in your FastAPI app.

**🔐 Authentication Dependency in FastAPI**

Authentication dependencies are functions that:

* **Check a token** (e.g., a JWT)
* **Decode it**
* **Look up the user in the database**
* **Return the user object** if the token is valid
* Raise an error if the token is invalid or expired

**🧩 Where It’s Used in Your Code**

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid username or password", headers={"WWW-AUTHENTICATE": "Bearer"} ) return await user

Let’s break this down step by step.

**🔁 Step-by-Step Explanation**

**✅ Step 1: Declare the Dependency**

python

CopyEdit

token: str = Depends(oath2\_scheme)

* oath2\_scheme is created like this:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

* This tells FastAPI:

“Look for an Authorization: Bearer <token> header in the request.”

* FastAPI extracts the token and passes it to the get\_current\_user() function.

**✅ Step 2: Decode the Token**

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

* Decodes the JWT using the secret key.
* If it's invalid or expired, an exception is raised.

**Example JWT Payload:**

json

CopyEdit

{ "id": 3, "username": "john" }

**✅ Step 3: Fetch the User from the Database**

python

CopyEdit

user = await User.get(id=payload.get("id"))

* Uses the user ID from the token to look up the user in the database.

**✅ Step 4: Handle Errors**

python

CopyEdit

except: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid username or password", headers={"WWW-AUTHENTICATE": "Bearer"} )

* If the token is bad or user doesn’t exist, raise a 401 Unauthorized error.

**✅ Step 5: Return the User**

python

CopyEdit

return await user

* This user is returned to the route that depends on get\_current\_user.

**📦 Using the Dependency in a Route**

python

CopyEdit

@app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): ...

* This route **requires a valid JWT token**.
* FastAPI calls get\_current\_user(), gets the user, and passes it to the route.

**✅ Summary Table**

| **Concept** | **Explanation** |
| --- | --- |
| OAuth2PasswordBearer | Extracts token from the Authorization header |
| Depends(oath2\_scheme) | Injects the token string into your function |
| get\_current\_user() | Decodes the token and returns the user |
| Used In Routes | Any route that requires login/authentication (e.g., /user/me) |

**🧠 Real Example Flow**

http

CopyEdit

Request: GET /user/me Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5... FastAPI: - Extracts token using `oath2\_scheme` - Calls `get\_current\_user(token)` - Decodes token → Gets user → Returns user to route

Would you like to see how to customize get\_current\_user() to check user roles or permissions?

**You said:**

async def get\_current\_user(token: str = Depends(oath2\_scheme)): explain this line and what is request header

**ChatGPT said:**

Perfect! You're asking a very important question about how FastAPI uses **dependency injection** and how it interacts with **request headers** — specifically in this line:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)):

Let’s break it down step-by-step and explain what it does, what the request header is, and how they work together.

**🧠 Line Breakdown:**token: str = Depends(oath2\_scheme)

**✅ What This Means:**

* You're defining a parameter token in the get\_current\_user() function.
* You use Depends(oath2\_scheme) to tell FastAPI:

"I want you to **extract a token** from the **Authorization header** of the HTTP request."

* The return value from oath2\_scheme (which is a class: OAuth2PasswordBearer) gets passed to token.

**💡 Behind the Scenes:**

FastAPI automatically:

1. Looks for the Authorization header in the incoming request.
2. Extracts the JWT token after the word Bearer.
3. Passes that token as the value of the token parameter.

**🔐 Example Request Header**

The **request header** is extra data sent with the HTTP request. The one used here is:

http

CopyEdit

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

This is what a frontend or API client (like Postman) sends to authenticate.

| **Header Key** | **Header Value** |
| --- | --- |
| Authorization | Bearer <your-JWT-token> |

FastAPI will:

* Strip the Bearer part,
* Grab the actual token (like eyJhbGciOiJIUzI1NiIs...),
* Inject it into get\_current\_user(token=...).

**🔁 Full Flow**

Let’s say a client sends this request:

http

CopyEdit

GET /user/me Authorization: Bearer abc123.jwt.token

**FastAPI Internally Does This:**

python

CopyEdit

# Extract token from the request header token = "abc123.jwt.token" # Inject it into this function: await get\_current\_user(token)

So this line:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)):

Really means:

"Get the token from the header and give it to me as a function argument."

**🔒 Summary**

| **Component** | **Role** |
| --- | --- |
| Depends(oath2\_scheme) | Tells FastAPI to extract token from the Authorization header |
| token: str | Will receive the token extracted by oath2\_scheme |
| Authorization Header | Standard HTTP header used to send tokens for authentication |
| Value Format | Must start with Bearer followed by a space and the actual token |

Would you like to see how to **manually test this with Postman or curl** to better understand how headers work in practice?

**You said:**

The return value from oath2\_scheme (which is a class: OAuth2PasswordBearer) gets passed to token. what does it return

**ChatGPT said:**

Great follow-up!

**🔁 So, what does**oath2\_scheme**(which is an instance of**OAuth2PasswordBearer**) actually return?**

**✅ Short Answer:**

It returns the **token string** from the Authorization header of the HTTP request.

**📦 Example:**

Say the client sends this HTTP request:

http

CopyEdit

GET /some-protected-endpoint Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Then this line:

python

CopyEdit

token: str = Depends(oath2\_scheme)

Results in:

python

CopyEdit

token == "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

The token part **after**Bearer is extracted and passed into your function as the token variable.

**⚙️ Internally, What**OAuth2PasswordBearer**Does**

1. **Reads** the Authorization header.
2. **Verifies** it starts with Bearer .
3. **Returns** just the token portion as a string.
4. If no token or invalid format, it raises:

http

CopyEdit

401 Unauthorized { "detail": "Not authenticated" }

**🧠 What it does NOT do:**

* It does **not** decode the JWT.
* It does **not** verify if the token is valid.
* It simply **extracts the token** from the header.

🔒 The actual verification happens in your custom logic like:

python

CopyEdit

payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"])

**🔁 Visual Summary**

text

CopyEdit

Incoming Request: GET /user/me Authorization: Bearer abc.def.ghi FastAPI: - oath2\_scheme finds "Bearer abc.def.ghi" - Returns "abc.def.ghi" - Passes it to: get\_current\_user(token="abc.def.ghi")

Let me know if you'd like to write your own version of a simplified OAuth2PasswordBearer to see how it works under the hood!

**You said:**

oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") what does this line do again

**ChatGPT said:**

Great — let’s focus purely on this line:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

**✅ In one sentence:**

This line **creates a dependency** that tells FastAPI:

“For protected endpoints, expect a token in the Authorization header, and extract it when a request is made.”

**🔍 Detailed Breakdown**

**🔧 1.**OAuth2PasswordBearer(...)**is a class**

* Provided by FastAPI (from fastapi.security).
* Used for **OAuth2 password flow** where the user gets a token by sending username/password to a login endpoint.

**🔑 2.**tokenUrl="token"

* This is **only used in the Swagger UI (OpenAPI docs)**.
* It tells FastAPI that tokens can be obtained from the /token endpoint.
* So when you click “Authorize” in the docs, it knows where to send the login request.

**📦 3.**oath2\_scheme**becomes a dependency function**

After this line, oath2\_scheme can be used in your route or another function like this:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)):

This tells FastAPI to:

* Look for this header in the request:

makefile

CopyEdit

Authorization: Bearer <your\_token>

* Strip out the Bearer part.
* Return just the token string to the token parameter.

**🧪 Real Request Example**

**Request:**

http

CopyEdit

GET /user/me Authorization: Bearer abc123.jwt.token

**FastAPI Internally:**

python

CopyEdit

token = "abc123.jwt.token" # extracted by oath2\_scheme

**🧾 Summary Table**

| **Code** | **What It Does** |
| --- | --- |
| OAuth2PasswordBearer(...) | Sets up a security dependency to extract bearer tokens |
| tokenUrl="token" | Swagger UI hint: where to get tokens (from /token) |
| oath2\_scheme | The callable you use with Depends() to extract the token |

Would you like a visual diagram of how this fits into your login and authorization flow?

**You said:**

ok so when there is token in header it guides it to token endpoint

**ChatGPT said:**

Not quite — but that’s a **very common misunderstanding**, so let me clarify it clearly for you.

**❌ Misconception:**

*“When there is a token in the header, FastAPI routes it to the /token endpoint.”*

**❗No, that’s not how it works.**

**✅ Correct Understanding:**

**🔑 The**/token**endpoint is only for logging in and getting a token.**

**🛡️ After you get the token, you use it to access other protected endpoints, not**/token**.**

**Let's Break it Down:**

**1.**/token**endpoint:**

* You send **username** and **password** (not a token).
* It returns a **JWT token**.

**Request to**/token**:**

http

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=admin&password=secret

**Response:**

json

CopyEdit

{ "access\_token": "abc.def.ghi", "token\_type": "bearer" }

**2. After getting the token:**

You use that token to access protected routes like /user/me.

**Request to**/user/me**:**

h

CopyEdit

GET /user/me Authorization: Bearer abc.def.ghi

* Now FastAPI uses the dependency:

python

CopyEdit

token: str = Depends(oath2\_scheme)

* Which tells FastAPI:

“Look in the request header for Authorization: Bearer <token> and extract the token.”

**3.**tokenUrl="token"**is just for Swagger docs**

This:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

Means:

* In Swagger UI, when you click "Authorize", FastAPI knows it should send login info to /token.
* It’s not a router or redirect.
* It's only used for OpenAPI (docs), not for actual logic during requests.

**🧠 Simple Analogy**

| **Action** | **Endpoint** | **Description** |
| --- | --- | --- |
| Login with username/password | POST /token | You get a token |
| Use token to access data | GET /user/me | Include token in Authorization header |
| tokenUrl="token" | (Swagger-only) | Just tells the docs where the login form lives |

**✅ Summary**

* ✅ The token in the header is used to access **protected routes**, not the /token endpoint.
* 🔁 The /token endpoint is only called **once during login**, to **get the token**.
* 📚 tokenUrl="token" is only for OpenAPI UI guidance.

Let me know if you'd like me to draw a simple flowchart to make this clearer!

**You said:**

yeah draw a flow chart

**ChatGPT said:**

Sure! Here's a simple **flowchart-style explanation** of the token-based login and authentication flow in FastAPI using OAuth2PasswordBearer.

**🔁 FastAPI Token Authentication Flow**

pgsql

CopyEdit

[Client: User enters login info]

|

v

[POST /token with username & password]

|

v

FastAPI calls token\_generator():

- Validates credentials

- Creates JWT token

|

v

[Returns access\_token to client as JSON]

|

v

[Client stores token (e.g. in browser, mobile app)]

|

v

[Client makes a request to protected endpoint, e.g. /user/me]

|

v

[Sends request with header:]

Authorization: Bearer <access\_token>

|

v

FastAPI (Depends(oauth2\_scheme)) extracts the token

|

v

[get\_current\_user(token) is called]

- Decodes the token

- Verifies it

- Retrieves user from DB

|

v

✅ If valid: route runs normally and returns data

❌ If invalid: 401 Unauthorized error

**📦 Quick Reference**

| **Step** | **What Happens** | **FastAPI Role** |
| --- | --- | --- |
| 1 | Client logs in at /token | Calls token\_generator() |
| 2 | Token returned | JWT generated and sent in response |
| 3 | Client stores and uses token | Sends it in Authorization header |
| 4 | Client accesses protected route | Depends(oath2\_scheme) extracts token |
| 5 | Token validated in get\_current\_user() | Decoded, checked, user loaded |
| 6 | Response returned | Route logic continues or 401 error raised |

Would you like a downloadable diagram (image or PDF), or is this text-based flow enough?

**You said:**

which function takes its value

**ChatGPT said:**

Great question!

You're asking:

💬 “Which function takes the value returned by oath2\_scheme?”

**✅ Short Answer:**

The function that takes the value from oath2\_scheme is:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme))

**🔍 In Detail:**

* oath2\_scheme is called via Depends(...).
* It **extracts the token** from the Authorization header.
* That token (as a string) gets passed as the token**argument** to the get\_current\_user() function.

**🔁 Example Flow:**

**Header sent in the request:**

http

CopyEdit

Authorization: Bearer abc.def.ghi

oath2\_scheme**extracts this:**

python

CopyEdit

token = "abc.def.ghi"

**And then:**

python

CopyEdit

# This function receives that token async def get\_current\_user(token: str = Depends(oath2\_scheme)): payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) return user

So now, token holds the actual JWT, and get\_current\_user() can:

* Decode it
* Validate it
* Use the payload to fetch the user

**✅ Summary:**

| **Component** | **Role** |
| --- | --- |
| oath2\_scheme | Extracts token from the Authorization header |
| Depends(...) | Tells FastAPI to call oath2\_scheme automatically |
| get\_current\_user() | Receives the token as the token parameter |

Would you like to see how to **pass this user object to other routes** for role-based access or profile data?

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

23-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: # This goes LAST raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user async def verify\_password(plain\_password, hashed\_password): return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): user = await User.get(username = username) if user and await verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token this is my verification module explain how user login works

**ChatGPT said:**

Your FastAPI application implements user authentication and login using OAuth2 with JWT (JSON Web Tokens). Here's a breakdown of how the **user login** process works step-by-step:

**🔑 1. User Sends Credentials to**/token**Endpoint**

**Endpoint:**

python

CopyEdit

@app.post("/token")

**Request:**  
The user sends a POST request with form data:

bash

CopyEdit

username=myusername&password=mypassword

This is handled by:

python

CopyEdit

OAuth2PasswordRequestForm = Depends()

which parses form fields username and password.

**🔍 2. Backend Authenticates User**

In the /token route, this function is called:

python

CopyEdit

token = await token\_generator(request\_form.username, request\_form.password)

token\_generator()**does:**

1. Calls authenticate\_user():

python

CopyEdit

user = await authenticate\_user(username, password)

1. authenticate\_user():
   * Fetches user from DB by username:

python

CopyEdit

user = await User.get(username=username)

* + Verifies password using:

python

CopyEdit

pwd\_context.verify(plain\_password, hashed\_password)

* + If password is valid, returns the User object.

1. If authentication succeeds, token\_generator():
   * Encodes user ID and username into a JWT:

python

CopyEdit

token\_data = {"id": user.id, "username": user.username} jwt.encode(token\_data, SECRET)

1. Returns the JWT token.

**📦 3. Frontend Receives JWT**

**Response from**/token**route:**

json

CopyEdit

{ "access\_token": "jwt\_token\_string", "token\_type": "bearer" }

The frontend stores this token (e.g., in localStorage or as a cookie).

**🔐 4. User Accesses Protected Route:**/user/me

**Endpoint:**

python

CopyEdit

@app.post("/user/me")

**Auth Mechanism:**

* Requires a bearer token in the Authorization header:

makefile

CopyEdit

Authorization: Bearer jwt\_token\_string

**Backend does:**

1. Calls get\_current\_user(token) with the bearer token:

python

CopyEdit

payload = jwt.decode(token, SECRET) user = await User.get(id=payload.get("id"))

1. Returns the authenticated user.

**✅ 5. Login Response Includes User Info**

Finally, the /user/me route responds with:

json

CopyEdit

{ "status": "ok", "data": { "username": "user1", "email": "user1@example.com", "verified": true, "joined\_date": "May 23 2025" } }

This gives the frontend enough data to display the user's profile or navigate to the main application.

**🛡 Summary Flow**

1. **Client** → /token → username + password
2. **Backend** → Authenticates & returns JWT
3. **Client** → Sends JWT to protected endpoints like /user/me
4. **Backend** → Verifies JWT → Returns user info

Let me know if you'd like a visual flowchart or code-specific explanations.

**You said:**

async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user is it any use to return user here

**ChatGPT said:**

Yes, **returning the**user**in**get\_current\_user()**is essential and serves a clear purpose**—especially in **dependency injection** for **protected routes**.

Here’s why:

**✅ Purpose of**get\_current\_user()

This function is designed to:

1. **Extract and decode** the JWT from the request.
2. **Identify the user** from the token payload.
3. **Fetch the user from the database.**
4. **Return the user object** so it can be used in protected routes.

**👇 Example Where It's Used**

python

CopyEdit

@app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): ...

* The user parameter in user\_login comes from get\_current\_user().
* If get\_current\_user() didn’t return the user, you’d have nothing to work with in your route logic.
* FastAPI's Depends() uses the return value of get\_current\_user().

**❌ If You Don't Return**user**?**

* The route using Depends(get\_current\_user) will receive None.
* That would break your app or require redundant DB queries inside each route to refetch the user.
* Example: You’d have to write something like:

python

CopyEdit

user = await get\_current\_user()

**💡 Clean Alternative (Optional Improvement)**

If you want type-checking and clear return typing, update the function like this:

python

CopyEdit

from models import User # Make sure you import your User model from typing import Annotated async def get\_current\_user(token: Annotated[str, Depends(oath2\_scheme)]) -> User: try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user

**✅ Final Verdict**

Yes — **returning**user**is critical**. It enables **authenticated routes** to work correctly by injecting the logged-in user directly into route functions. Without it, you'd lose the ability to associate requests with a user session.

**You said:**

why cant we write async def get\_current\_user(token: str = Depends(generate\_token)):

**ChatGPT said:**

Great question! Here's a clear explanation of **why you shouldn't** (and generally **can’t**) write:

python

CopyEdit

async def get\_current\_user(token: str = Depends(generate\_token)):

**🚫 1.**generate\_token**is not meant to extract token, it's meant to create one**

* generate\_token(username, password) **creates a new JWT token** using user credentials.
* It expects username and password from an OAuth2 form.
* But get\_current\_user() is supposed to **decode and verify an existing token from the request header**, **not generate a new one.**

**✅ The correct pattern uses:**

python

CopyEdit

token: str = Depends(oauth2\_scheme)

* This uses OAuth2PasswordBearer, which tells FastAPI:

“Extract the Bearer token from the Authorization header of the request.”

Example header:

http

CopyEdit

Authorization: Bearer eyJhbGciOiJIUzI1...

**🔁 So, what actually happens:**

| **Function** | **Purpose** | **Where it's used** |
| --- | --- | --- |
| generate\_token() | Authenticate user & generate JWT | Inside /token endpoint |
| oauth2\_scheme | Extract token from header | In Depends() call |
| get\_current\_user() | Decode and verify existing token | As a dependency in routes |

**❌ Why**Depends(generate\_token)**is incorrect**

* FastAPI will try to call generate\_token() as a dependency.
* But generate\_token() requires username and password, which:
  + Are not available in an authenticated request
  + Don't make sense when all we want is to **extract and validate** a token
* It will fail with a 422 error or throw an exception about missing required parameters.

**✅ Correct usage:**

python

CopyEdit

from fastapi.security import OAuth2PasswordBearer oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token") async def get\_current\_user(token: str = Depends(oauth2\_scheme)): # decode and validate the token here ...

This ensures:

* Token is **extracted from the Authorization header**
* You can **decode and validate** it to get the user

**🔐 Summary**

You **can’t** (and shouldn't) write Depends(generate\_token) because:

* It’s meant to create a token from credentials.
* At this stage, you already **have the token**.
* You need to **verify**, not generate.

Let me know if you’d like a diagram or flowchart to visualize the process!

**You said:**

then how does this work async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)):

**ChatGPT said:**

Excellent follow-up! Let's break it down:

**✅ How This Works:**

python

CopyEdit

async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)):

**Step-by-Step Flow:**

1. **Client makes a request to**/user/me:

http

CopyEdit

POST /user/me Authorization: Bearer eyJhbGciOiJIUzI1...

1. **FastAPI sees**Depends(get\_current\_user):
   * This tells FastAPI: “Before running user\_login(), run get\_current\_user() and inject its return value into user.”
2. get\_current\_user()**is executed**:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oauth2\_scheme)):

* + Depends(oauth2\_scheme) extracts the JWT token from the **Authorization header**.
  + The token is decoded and validated.
  + It fetches the User from the database.
  + Returns a User instance (Tortoise ORM model).

1. **The returned**User**model is passed into**user\_login()**as**user

**⚠️ But wait — there's a small issue in your function:**

python

CopyEdit

async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)):

You're typing the user as user\_pydanticIn, but you're actually returning a User**ORM model** from get\_current\_user().

**✅ Correct Usage:**

You should do one of the following:

**✅ Option 1: Use ORM directly**

python

CopyEdit

async def user\_login(user: User = Depends(get\_current\_user)):

Use the User model directly inside the function, and serialize it manually or with a Pydantic schema if needed.

**✅ Option 2: Convert to Pydantic model (cleaner for API output)**

Update get\_current\_user() to return a Pydantic schema:

python

CopyEdit

from models import user\_pydantic async def get\_current\_user(token: str = Depends(oauth2\_scheme)) -> user\_pydantic: try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except: raise HTTPException(...) return await user\_pydantic.from\_tortoise\_orm(user)

Then your user\_login() function will work perfectly:

python

CopyEdit

async def user\_login(user: user\_pydantic = Depends(get\_current\_user)): return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } }

**🔁 Summary**

* Depends(get\_current\_user) **injects the return value** of get\_current\_user() into the route.
* This works because FastAPI runs all Depends() functions before entering the route.
* You must ensure that the **type** returned by get\_current\_user() matches the type you declare in the route.

Let me know if you’d like an example of how this flow looks with real requests and responses!

**You said:**

Depends(oauth2\_scheme) ok so this line role is to extract the token while async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): this line role is to generate a jwt using username and password in header

**ChatGPT said:**

Yes — you're **exactly right**, and your understanding is on point! Here's a clean breakdown for clarity:

**🔄**Depends(oauth2\_scheme)

**✅ Purpose: Extract token from header**

This line:

python

CopyEdit

token: str = Depends(oauth2\_scheme)

* Uses the OAuth2PasswordBearer tool.
* Extracts the **Bearer token** from the Authorization header:

makefile

CopyEdit

Authorization: Bearer <your.jwt.token>

* This is used in routes that require **token-based authentication**, like:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oauth2\_scheme)):

Think of it as:

“Give me the token from the request’s header so I can verify who the user is.”

**🔐**Depends()**with**OAuth2PasswordRequestForm

**✅ Purpose: Accept credentials and generate token**

This line:

python

CopyEdit

async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()):

* Tells FastAPI to extract **form data** (not headers) for:
  + username
  + password
* These are sent from the frontend to the /token endpoint via a POST request like this:

bash

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=myuser&password=mypass

* Then you call:

python

CopyEdit

user = await authenticate\_user(username, password)

* And if valid, you:

python

CopyEdit

token = jwt.encode({...}, SECRET) return {"access\_token": token, "token\_type": "bearer"}

**🧠 In Summary:**

| **Line / Dependency** | **Purpose** | **Used In** |
| --- | --- | --- |
| Depends(oauth2\_scheme) | Extracts Bearer token from header | In protected routes (e.g. get\_current\_user) |
| Depends(OAuth2PasswordRequestForm) | Extracts username and password from form | In /token route to **generate** token |

Let me know if you want a quick diagram to visualize the request/response flow — it's a great way to see it end-to-end.

**You said:**

from tortoise.models import Model #When you define a class that inherits from Model, you're creating a representation of a table in your database. Each attribute defined with a fields.\*Field is a column in that table. from tortoise import fields # Field module is a submodule of Tortoise ORM that provides field definitions for model classes. Each item in fields corresponds to a type of column you would find in a relational database. from datetime import datetime from tortoise.contrib.pydantic import pydantic\_model\_creator """ It converts Tortoise ORM models (like User, Business, Product) into Pydantic schemas, which are typically used for: Request validation (e.g. when creating or updating data) Response serialization (e.g. when returning data to the client) 📌 Why It's Used: Manually creating Pydantic schemas for each model can be repetitive. pydantic\_model\_creator reduces boilerplate by automatically generating them from the Tortoise model definitions. 🔍 Example: Let's say you have this Tortoise ORM model: python Copy Edit class User(Model): id = fields.IntField(pk=True) username = fields.CharField(max\_length=20) email = fields.CharField(max\_length=100) You can generate a Pydantic schema like this: python Copy Edit from tortoise.contrib.pydantic import pydantic\_model\_creator user\_pydantic = pydantic\_model\_creator(User, name="User") Now, user\_pydantic is a Pydantic class that you can use in FastAPI like this: python Copy Edit @app.get("/users/{id}", response\_model=user\_pydantic) async def get\_user(id: int): user = await User.get(id=id) return await user\_pydantic.from\_tortoise\_orm(user)""" class User(Model): id = fields.IntField(pk=True, index=True) username = fields.CharField(max\_length=20, null=False, unique=True) email = fields.CharField(max\_length=200, null=False, unique=True) password = fields.CharField(max\_length=100, null=False) is\_verified = fields.BooleanField(default=False) join\_date = fields.DatetimeField(default=datetime.utcnow) class Business(Model): id = fields.IntField(pk=True, index=True) business\_name = fields.CharField(max\_length=20, null=False, unique=True) city = fields.CharField(max\_length=100, null=False, default="Unspecified") region = fields.CharField(max\_length=100, null=False, default="Unspecified") business\_description = fields.TextField(null=True) logo = fields.CharField(max\_length=200, null=False, default="default.jpg") owner = fields.ForeignKeyField("models.User", related\_name="businesses") """This means: Each Business has one owner, who is a User. Each User can have many businesses. This is equivalent to a FOREIGN KEY (owner\_id) REFERENCES user(id) in SQL. 🔎 Let’s break it down: 📄 Business model: python Copy Edit class Business(Model): ... owner = fields.ForeignKeyField("models.User", related\_name="businesses") "models.User": This is a string reference to the User model, to avoid circular import issues. related\_name="businesses": This tells Tortoise to add a reverse relation so that you can do user.businesses to get all businesses owned by that user. 🔧 What it does behind the scenes: It creates a column like this in the Business table: sql Copy Edit owner\_id INTEGER REFERENCES user(id) And allows you to do: Forward Access (from Business to User): python Copy Edit business = await Business.get(id=1) owner = await business.owner # This fetches the related User object Reverse Access (from User to Business list): python Copy Edit user = await User.get(id=1) businesses = await user.businesses.all() # Thanks to related\_name="businesses" ✅ Real Example: python Copy Edit # Create a user user = await User.create(username="bob", email="bob@mail.com", password="123") # Create a business for that user business = await Business.create(business\_name="Bob's Burgers", owner=user) # Fetch owner from business owner = await business.owner print(owner.username) # "bob" # Fetch businesses from user user\_businesses = await user.businesses.all() print([b.business\_name for b in user\_businesses]) # ["Bob's Burgers"] """ class Product(Model): id = fields.IntField(pk=True, index=True) name = fields.CharField(max\_length=100, null=False, unique=True) category = fields.CharField(max\_length=30, index=True) original\_price = fields.DecimalField(max\_digits=12, decimal\_places=2) new\_price = fields.DecimalField(max\_digits=12, decimal\_places=2) percentage\_discount = fields.IntField() offer\_expiration\_date = fields.DateField(default=datetime.utcnow) product\_image = fields.CharField(max\_length=200, null=False, default="productDefault.jpg") business\_owner = fields.ForeignKeyField("models.Business", related\_name="products") # Pydantic Schemas user\_pydantic = pydantic\_model\_creator(User, name="User", exclude=("is\_verified",)) user\_pydanticIn = pydantic\_model\_creator(User, name="UserIn", exclude\_readonly=True, exclude=("is\_verified", "join\_date")) user\_pydanticOut = pydantic\_model\_creator(User, name="UserOut", exclude=("password",)) business\_pydantic = pydantic\_model\_creator(Business, name="Business") business\_pydanticIn = pydantic\_model\_creator(Business, name="BusinessIn", exclude\_readonly=True) product\_pydantic = pydantic\_model\_creator(Product, name="Product") product\_pydanticIn = pydantic\_model\_creator(Product, name="ProductIn", exclude=("percentage\_discount", "id")) """✅ 1. user\_pydantic → General Model (exclude is\_verified) python Copy Edit exclude=("is\_verified",) This model includes all fields from the User model except is\_verified. Useful if you don't want the is\_verified status exposed or accepted by default. Can be used for general views or documentation where this field shouldn't be visible. ✅ 2. user\_pydanticIn → Input Schema (for creating users) python Copy Edit exclude\_readonly=True exclude\_readonly=True excludes read-only fields, like: id (auto-generated) join\_date (has a default) This makes sure clients only provide fields they're allowed to set manually, like: username, email, password ✅ Used for: python Copy Edit @app.post("/users", response\_model=UserOut\_Pydantic) async def create\_user(user: UserIn\_Pydantic): ... ✅ 3. user\_pydanticOut → Output Schema (for returning users) python Copy Edit exclude\_readonly=("password",) This is a bit tricky — here, you're using exclude\_readonly=("password",) but it likely should be: python Copy Edit exclude=("password",) password is not a read-only field, so exclude\_readonly=("password",) won't exclude it. To remove password from the output, use: python Copy Edit exclude=("password",) ✅ Purpose: Prevent exposing sensitive fields (like passwords) when returning user data.""" here is my model module isnt user\_pydanticIn and User same thing

**ChatGPT said:**

Great question — and one that often confuses developers when using **Tortoise ORM** and **Pydantic together** in FastAPI.

**❓ Is**user\_pydanticIn**the same as**User**?**

**Short answer:** **No**, they are not the same.

**🔍 Here's the difference:**

**1.**User**(Tortoise ORM Model)**

This is your actual **database model**:

python

CopyEdit

class User(Model): id = fields.IntField(pk=True, index=True) username = fields.CharField(max\_length=20, null=False, unique=True) ...

* It's used to **interact with the database**.
* You use it to create, retrieve, update, or delete records.
* It's asynchronous (you use await User.create(...), await User.get(...), etc.)
* It includes logic like relationships (e.g., user.businesses).

**2.**user\_pydanticIn**(Pydantic Schema)**

This is a **Pydantic model generated from the Tortoise model** using:

python

CopyEdit

user\_pydanticIn = pydantic\_model\_creator( User, name="UserIn", exclude\_readonly=True, exclude=("is\_verified", "join\_date") )

* It's used for **input validation** in FastAPI.
* It only includes the fields a **client is allowed to send** (like username, email, and password).
* It excludes things like:
  + id (which is auto-generated)
  + join\_date (which has a default)
  + is\_verified (should not be manually set by the client)
* It's a **pure data class** (not tied to the DB, doesn't run queries).

**🔄 When you use them:**

| **Purpose** | **Use**User**ORM** | **Use**user\_pydanticIn |
| --- | --- | --- |
| Talking to the DB | ✅ Yes | ❌ No |
| Validating request body | ❌ No | ✅ Yes |
| Returning safe output (no password) | ❌ No | Use user\_pydanticOut instead |

**✅ Real Example Use**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): # user is validated already (username, email, password) user\_info = user.dict(exclude\_unset=True) user\_info["password"] = get\_hashed\_password(user\_info["password"]) user\_obj = await User.create(\*\*user\_info) # Uses the ORM return {"status": "ok"}

Here:

* user\_pydanticIn: Validates what the client sends
* User: Creates the actual DB record

**🧠 TL;DR**

* User → ORM model for the **database**
* user\_pydanticIn → Pydantic model for **validating incoming data**

They represent **the same structure**, but serve **different roles**:

* One talks to the **database**
* The other talks to the **client**

Let me know if you'd like a diagram showing how the request flows from client → FastAPI → DB and back!

Get smarter responses, upload files and images, and more.

Log in

Sign up for free

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

24-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import image #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config @app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module of a project Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) INFO: Stopping reloader process [13916] PS D:\Data Structures and Algorithms\Ecommerce> pip install pillow Requirement already satisfied: pillow in c:\users\asus\appdata\local\programs\python\python311\lib\site-packages (11.2.1) PS D:\Data Structures and Algorithms\Ecommerce> uvicorn main:app --reload INFO: Will watch for changes in these directories: ['D:\\Data Structures and Algorithms\\Ecommerce'] INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit) INFO: Started reloader process [16272] using WatchFiles Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) this is the error i am getting

**ChatGPT said:**

The error you're getting is very clear:

pgsql

CopyEdit

ImportError: cannot import name 'image' from 'PIL'

**🔍 The Problem:**

In your main.py, you wrote:

python

CopyEdit

from PIL import image

But the correct import should be:

python

CopyEdit

from PIL import Image

Python is **case-sensitive**, and the Pillow library provides the Image class (capital "I"), not image.

**✅ Fix:**

Change this line in your main.py:

python

CopyEdit

from PIL import image

To:

python

CopyEdit

from PIL import Image

Also, in your /uploadfile/profile route, you used:

python

CopyEdit

img = Image.open(generated\_name)

This will work **only if you've corrected the import to**:

python

CopyEdit

from PIL import Image

**🔁 After Fixing:**

After making this change, **save the file** and run the app again:

bash

CopyEdit

uvicorn main:app --reload

It should now start without that import error.

Let me know if you face any further issues!

**You said:**

Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) INFO: Stopping reloader process [16272] PS D:\Data Structures and Algorithms\Ecommerce> uvicorn main:app --reload INFO: Will watch for changes in these directories: ['D:\\Data Structures and Algorithms\\Ecommerce'] INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit) INFO: Started reloader process [26344] using WatchFiles Process SpawnProcess-1: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 21, in <module> from PIL import image ImportError: cannot import name 'image' from 'PIL' (C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\PIL\\_\_init\_\_.py) WARNING: WatchFiles detected changes in 'main.py'. Reloading... Process SpawnProcess-2: Traceback (most recent call last): File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in \_bootstrap self.run() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run self.\_target(\*self.\_args, \*\*self.\_kwargs) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\\_subprocess.py", line 80, in subprocess\_started target(sockets=sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 66, in run return asyncio.run(self.serve(sockets=sockets)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 190, in run return runner.run(main) ^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\runners.py", line 118, in run return self.\_loop.run\_until\_complete(task) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\asyncio\base\_events.py", line 653, in run\_until\_complete return future.result() ^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 70, in serve await self.\_serve(sockets) File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\server.py", line 77, in \_serve config.load() File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\config.py", line 435, in load self.loaded\_app = import\_from\_string(self.app) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\uvicorn\importer.py", line 19, in import\_from\_string module = importlib.import\_module(module\_str) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\Asus\AppData\Local\Programs\Python\Python311\Lib\importlib\\_\_init\_\_.py", line 126, in import\_module return \_bootstrap.\_gcd\_import(name[level:], package, level) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "<frozen importlib.\_bootstrap>", line 1206, in \_gcd\_import File "<frozen importlib.\_bootstrap>", line 1178, in \_find\_and\_load File "<frozen importlib.\_bootstrap>", line 1149, in \_find\_and\_load\_unlocked File "<frozen importlib.\_bootstrap>", line 690, in \_load\_unlocked File "<frozen importlib.\_bootstrap\_external>", line 940, in exec\_module File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed File "D:\Data Structures and Algorithms\Ecommerce\main.py", line 33, in <module> @app.mount("/static", StaticFiles(directory="static"), name="static") ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ TypeError: 'NoneType' object is not callable now i get this

**ChatGPT said:**

You're encountering **two separate errors** in your FastAPI/Uvicorn application:

**✅ Error 1:**

**❌**ImportError: cannot import name 'image' from 'PIL'

**✅ Fix:**

Change:

python

CopyEdit

from PIL import image

To:

python

CopyEdit

from PIL import Image

Image is capitalized — **case matters** in Python! image doesn't exist in the PIL module.

**✅ Error 2:**

**❌**TypeError: 'NoneType' object is not callable**on:**

python

CopyEdit

@app.mount("/static", StaticFiles(directory="static"), name="static")

**✅ Fix:**

You're trying to use @app.mount(...) — **but**mount()**is not a decorator**. It's a **method**, not meant to be used with @.

**Correct usage:**

python

CopyEdit

from fastapi.staticfiles import StaticFiles app.mount("/static", StaticFiles(directory="static"), name="static")

Just call app.mount(...) **directly** — don't prefix it with @.

**✅ Summary of Fixes:**

Make sure main.py contains:

python

CopyEdit

from fastapi import FastAPI from fastapi.staticfiles import StaticFiles from PIL import Image # Correct import app = FastAPI() # Mount static files app.mount("/static", StaticFiles(directory="static"), name="static")

After making these changes, restart your Uvicorn server:

bash

CopyEdit

uvicorn main:app --reload

Let me know if you want help verifying the file structure or any other errors after this.

**You said:**

what is a decorator

**ChatGPT said:**

A **decorator** in Python is a special function that **modifies the behavior** of another function, method, or class — **without changing its source code**.

You use it with the @ symbol, like this:

python

CopyEdit

@my\_decorator def say\_hello(): print("Hello")

**🔧 What It Really Does:**

This:

python

CopyEdit

@my\_decorator def say\_hello(): print("Hello")

Is **equivalent to**:

python

CopyEdit

def say\_hello(): print("Hello") say\_hello = my\_decorator(say\_hello)

The decorator wraps say\_hello() and can add extra logic **before or after** it runs.

**✅ Real Example:**

python

CopyEdit

def my\_decorator(func): def wrapper(): print("Before function runs") func() print("After function runs") return wrapper @my\_decorator def greet(): print("Hi!") greet()

**Output:**

pgsql

CopyEdit

Before function runs

Hi!

After function runs

**🔥 In FastAPI:**

You’ve likely seen this:

python

CopyEdit

@app.get("/") def home(): return {"message": "Welcome!"}

That @app.get("/") is a **decorator** that tells FastAPI:

“Run the home() function whenever someone visits / with a GET request.”

**TL;DR:**

* A **decorator** is a shortcut to **wrap functions with extra behavior**.
* Use @decorator\_name above your function.
* FastAPI uses decorators to define routes like @app.get("/path").

Let me know if you want to create your own custom decorator — it's a great way to learn how they work!

**You said:**

what is the use fo app.

**ChatGPT said:**

Great question!

In web frameworks like **FastAPI** or **Flask**, app is the main **application object** that represents your **web server**.

**🚀 What is**app**in FastAPI?**

In FastAPI, you typically create it like this:

python

CopyEdit

from fastapi import FastAPI app = FastAPI()

Here, app is an instance of the FastAPI class — it's the **core of your web application**.

**✅ What can you do with**app**?**

1. **Define routes**:

python

CopyEdit

@app.get("/") def home(): return {"message": "Welcome to my API"}

This tells FastAPI:

When someone makes a GET request to /, run the home() function.

1. **Include routers** (for modular apps):

python

CopyEdit

from fastapi import APIRouter router = APIRouter() @router.get("/users") def get\_users(): return [{"name": "Alice"}, {"name": "Bob"}] app.include\_router(router)

1. **Add middleware**:

python

CopyEdit

@app.middleware("http") async def add\_process\_time\_header(request, call\_next): response = await call\_next(request) response.headers["X-Process-Time"] = "0.1" return response

1. **Serve static files**:

python

CopyEdit

from fastapi.staticfiles import StaticFiles app.mount("/static", StaticFiles(directory="static"), name="static")

This tells FastAPI to serve files like CSS, JS, or images from the static/ folder at /static.

**🧠 Why does this matter?**

Without app, FastAPI wouldn't know:

* What routes exist
* How to handle requests
* What middleware or routers are added

**Summary:**

* app = FastAPI() creates your main web app.
* app is how you **register routes**, **mount static files**, **add middleware**, and more.
* Think of app as the brain of your API — it routes, handles, and responds to incoming HTTP requests.

Let me know if you want a working example of a full FastAPI app.py!

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my edited code but rather than getting the image at image folder i am getting it at static folder below is the image folder path for context D:\Data Structures and Algorithms\Ecommerce\static\images

**ChatGPT said:**

You're **almost there** — the issue is with how you're building the file path when saving the image.

**🔍 Problem:**

In this part of your code:

python

CopyEdit

FILEPATH = "./static/images" ... generated\_name = FILEPATH + token\_name

You're **concatenating** the folder path and filename **without a separator** — meaning, it ends up like:

arduino

CopyEdit

./static/imagesimagename.png

which is an **invalid path**. So Python likely falls back and creates it in the base of the static/ folder instead.

**✅ Fix:**

Use os.path.join() to safely construct the path:

python

CopyEdit

import os FILEPATH = "./static/images" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10) + "." + extension generated\_name = os.path.join(FILEPATH, token\_name) # 🔧 FIXED LINE file\_content = await file.read() with open(generated\_name, "wb") as f: f.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size=(200, 200)) img.save(generated\_name) f.close()

**📁 Resulting path:**

With os.path.join(), you'll get:

swift

CopyEdit

./static/images/abc123def456.png

Which maps to:

vbnet

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\static\images\abc123def456.png

And that’s where your image will correctly be saved.

Let me know if you'd like help adding image URL return logic or serving images dynamically!

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y") } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is the main module can you explain to me the uploadfile product and uploadfile profile endpoints

**ChatGPT said:**

Certainly! The /uploadfile/profile and /uploadfile/product/{id} endpoints in your FastAPI application are used to upload image files (e.g., profile pictures and product images). Here's a breakdown of each, what they do, and how they work:

**✅ 1. Endpoint:**/uploadfile/profile

**Purpose:**

To upload a **profile image** for the currently logged-in user's **business profile**.

**Endpoint Type:**

POST

**Required:**

* An image file (.png or .jpg)
* Authenticated user (via Depends(get\_current\_user))

**Functionality:**

1. **Get the uploaded file** from the request (UploadFile = File(...)).
2. **Validate the file extension**:
   * Only allow .png and .jpg.
   * Return an error if invalid.
3. **Generate a unique filename** using secrets.token\_hex() and append the file extension.
4. **Save the file** to the ./static/images/ directory.
5. **Resize the image** to (200, 200) using the Pillow (PIL) library.
6. **Find the business** associated with the current user.
7. **Check authorization**: Ensure the logged-in user is the **owner** of the business.
8. **Save the filename** as the business’s logo.
9. **Return a success message** with the image URL.

**Response:**

json

CopyEdit

{ "status": "ok", "filename": "localhost:8000/static/images/generated\_filename.jpg" }

**✅ 2. Endpoint:**/uploadfile/product/{id}

**Purpose:**

To upload an image for a specific **product** owned by the logged-in user.

**Endpoint Type:**

POST

**Required:**

* Product id in the URL
* An image file (.png or .jpg)
* Authenticated user (via Depends(get\_current\_user))

**Functionality:**

1. **Receive product ID** from the path parameter.
2. **Get the uploaded file** and **validate the extension** just like the profile upload.
3. **Generate a unique filename** and **save** the file in the same way.
4. **Resize the image** using Pillow.
5. **Retrieve the product** by its id.
6. **Get the business** the product belongs to.
7. **Verify ownership**: Make sure the user owns the business that owns the product.
8. **Update the product’s**product\_image**field** with the filename.
9. **Return success or raise an HTTP 401 error** if the user is not authorized.

**🔐 Common Authorization Check**

Both routes rely on this check:

python

CopyEdit

if owner == user: # Allow update else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this action" )

This ensures that only the **owner** of the business (for profile) or the **owner of the product’s business** (for product) can upload or modify images.

**📂 Directory Structure:**

All uploaded images are saved in:

arduino

CopyEdit

/static/images/

So, the URLs returned will look like:

bash

CopyEdit

http://localhost:8000/static/images/<filename>.jpg

**✅ Summary Table**

| **Endpoint** | **Purpose** | **Who can call it** | **Target** |
| --- | --- | --- | --- |
| /uploadfile/profile | Upload profile pic | Authenticated business owner | Business.logo |
| /uploadfile/product/{id} | Upload product image | Authenticated business owner | Product.product\_image |

Let me know if you want help turning this into reusable logic or want error handling improvements.

**You said:**

explain this filename = file.filename

**ChatGPT said:**

Sure! The line:

python

CopyEdit

filename = file.filename

is retrieving the **original name of the file** uploaded by the client (user) through an HTTP request.

**📦 Context:**

In FastAPI, when a client uploads a file using:

python

CopyEdit

file: UploadFile = File(...)

the UploadFile object includes several attributes:

* file.filename: The **name of the file as provided by the client**.
* file.file: The actual file-like object (binary stream).
* file.content\_type: The MIME type of the file (e.g., "image/jpeg").

**🧠 Why is this important?**

You're using filename to:

1. **Extract the file extension**:

python

CopyEdit

extension = filename.split(".")[1]

So you know whether it's a .jpg or .png file.

1. (Optionally) Use part of the filename in generating a **new, unique filename** to avoid collisions.

**📝 Example:**

Suppose a user uploads a file called:

arduino

CopyEdit

"myprofilepic.jpg"

Then:

python

CopyEdit

filename = file.filename # "myprofilepic.jpg" extension = filename.split(".")[1] # "jpg"

**⚠️ Note:**

file.filename is **user-controlled input**, so never trust it blindly (e.g., avoid using it directly to name files). That’s why your code correctly creates a unique name with:

python

CopyEdit

token\_name = secrets.token\_hex(10) + "." + extension

This prevents overwriting existing files and avoids any issues with malicious filenames.

Let me know if you want to see how to fully sanitize or validate filenames too.

**You said:**

what does this do file\_content = await file.read()

**ChatGPT said:**

Great question!

The line:

python

CopyEdit

file\_content = await file.read()

is used to **read the entire contents of the uploaded file into memory** asynchronously.

**🔍 Breakdown:**

* file is an instance of UploadFile, which is an asynchronous file-like object.
* .read() is an **async method** that reads the whole file content (like .read() on a normal file, but async).
* await is needed because file.read() is a **coroutine**, and must be awaited in an async function.

**📥 Purpose:**

This line is reading the uploaded file into the variable file\_content, so that you can:

* Write it to disk:

python

CopyEdit

with open(generated\_name, "wb") as file: file.write(file\_content)

* Optionally, inspect or process it in memory (e.g., virus scanning, checking size, etc.)

**📝 Example:**

If the user uploads a JPEG file:

* FastAPI gets the file.
* await file.read() will give you the **binary content** (bytes) of that image.
* You then write that binary data to a file with open(..., "wb").

**⚠️ Important:**

* This reads the **entire file into memory**. For **large files**, that could be risky (memory usage).
* If you want to handle large files safely, you should **read and write in chunks** instead of reading everything at once.

Let me know if you want an example for chunked reading too.

**You said:**

explain this code segment with open(generated\_name, "wb") as file: file.write(file\_content)

**ChatGPT said:**

Absolutely! Let's break down this code segment:

python

CopyEdit

with open(generated\_name, "wb") as file: file.write(file\_content)

This is a standard Python block for **writing binary data to a file on disk**.

**🔍 Line-by-line Explanation:**

**✅**with open(generated\_name, "wb") as file:

* open(): Opens a file.
* generated\_name: This is the **path + filename** where the uploaded image will be saved. Example:

python

CopyEdit

"./static/images/abc12345def.jpg"

* "wb" mode:
  + "w": Write mode (create file if it doesn't exist, overwrite if it does).
  + "b": Binary mode (important for images, videos, etc.—not text).
* with: A context manager that automatically closes the file afterward, even if an error occurs.
  + Prevents file corruption and resource leaks.

**✅**file.write(file\_content)

* Writes the **binary content** (file\_content, which came from await file.read()) into the opened file.
* This is how the uploaded file gets saved on your server (under ./static/images/).

**📝 Example:**

Let’s say the user uploaded photo.jpg and it was renamed to:

ini

CopyEdit

generated\_name = "./static/images/f39bcff123456.jpg"

After running:

python

CopyEdit

file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content)

➡️ The server saves the binary image data as ./static/images/f39bcff123456.jpg.

**📌 Summary:**

| **Part** | **Meaning** |
| --- | --- |
| open(..., "wb") | Open a file in binary write mode |
| with | Ensures the file is closed properly |
| file.write() | Writes binary content to disk |

This is a reliable and safe way to save uploaded images, videos, or any non-text file.

Let me know if you’d like an alternative using async file writes or how to handle large files!

**You said:**

business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() here user will be same as owner as we got owner by searching the business.owner and we got business by searching the user's business so why are we checking

**ChatGPT said:**

You're absolutely right to question that check — it's a good observation!

Let’s break it down step by step:

**✅ The Code:**

python

CopyEdit

business = await Business.get(owner = user) # Step 1 owner = await business.owner # Step 2 if owner == user: # Step 3 business.logo = token\_name await business.save()

**🧠 What it's doing:**

1. **Step 1**:

python

CopyEdit

business = await Business.get(owner = user)

Fetches the Business object where the owner is the currently authenticated user.

1. **Step 2**:

python

CopyEdit

owner = await business.owner

Retrieves the owner again from the fetched business — which will obviously be the same user.

1. **Step 3**:

python

CopyEdit

if owner == user:

Compares owner and user — but since you just queried the business by owner=user, they **must** be the same.

**🟡 So, why is this check redundant?**

You're absolutely correct: **yes, it's redundant**.

If:

python

CopyEdit

business = await Business.get(owner=user)

succeeds, then it's guaranteed that:

python

CopyEdit

business.owner == user

— otherwise that get() would have failed.

**🧽 When would this check make sense?**

* If you were **querying the business or product by some other identifier**, like:

python

CopyEdit

business = await Business.get(id=some\_id)

THEN you’d need to check:

python

CopyEdit

if business.owner == user:

* But since you're already filtering by owner=user, the check is unnecessary **unless you're being overly defensive or cautious**.

**✅ Recommendation:**

You can safely **remove the redundant check**:

python

CopyEdit

business = await Business.get(owner=user) business.logo = token\_name await business.save()

It keeps your code cleaner without any loss of safety — because Business.get(owner=user) already guarantees the relationship.

Let me know if you want to make this a reusable utility function or add permission checks for general-purpose queries.

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

27-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # Optional pre-check if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # Create user user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 product\_obj = await Product.create(\*\*product, business = user) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await product.get(id=id), business = await product.business, owner = await business.owner, if user == owner: product.delete() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is my main module from tortoise.models import Model #When you define a class that inherits from Model, you're creating a representation of a table in your database. Each attribute defined with a fields.\*Field is a column in that table. from tortoise import fields # Field module is a submodule of Tortoise ORM that provides field definitions for model classes. Each item in fields corresponds to a type of column you would find in a relational database. from datetime import datetime from tortoise.contrib.pydantic import pydantic\_model\_creator """ It converts Tortoise ORM models (like User, Business, Product) into Pydantic schemas, which are typically used for: Request validation (e.g. when creating or updating data) Response serialization (e.g. when returning data to the client) 📌 Why It's Used: Manually creating Pydantic schemas for each model can be repetitive. pydantic\_model\_creator reduces boilerplate by automatically generating them from the Tortoise model definitions. 🔍 Example: Let's say you have this Tortoise ORM model: python Copy Edit class User(Model): id = fields.IntField(pk=True) username = fields.CharField(max\_length=20) email = fields.CharField(max\_length=100) You can generate a Pydantic schema like this: python Copy Edit from tortoise.contrib.pydantic import pydantic\_model\_creator user\_pydantic = pydantic\_model\_creator(User, name="User") Now, user\_pydantic is a Pydantic class that you can use in FastAPI like this: python Copy Edit @app.get("/users/{id}", response\_model=user\_pydantic) async def get\_user(id: int): user = await User.get(id=id) return await user\_pydantic.from\_tortoise\_orm(user)""" class User(Model): id = fields.IntField(pk=True, index=True) username = fields.CharField(max\_length=20, null=False, unique=True) email = fields.CharField(max\_length=200, null=False, unique=True) password = fields.CharField(max\_length=100, null=False) is\_verified = fields.BooleanField(default=False) join\_date = fields.DatetimeField(default=datetime.utcnow) class Business(Model): id = fields.IntField(pk=True, index=True) business\_name = fields.CharField(max\_length=20, null=False, unique=True) city = fields.CharField(max\_length=100, null=False, default="Unspecified") region = fields.CharField(max\_length=100, null=False, default="Unspecified") business\_description = fields.TextField(null=True) logo = fields.CharField(max\_length=200, null=False, default="default.jpg") owner = fields.ForeignKeyField("models.User", related\_name="businesses") """This means: Each Business has one owner, who is a User. Each User can have many businesses. This is equivalent to a FOREIGN KEY (owner\_id) REFERENCES user(id) in SQL. 🔎 Let’s break it down: 📄 Business model: python Copy Edit class Business(Model): ... owner = fields.ForeignKeyField("models.User", related\_name="businesses") "models.User": This is a string reference to the User model, to avoid circular import issues. related\_name="businesses": This tells Tortoise to add a reverse relation so that you can do user.businesses to get all businesses owned by that user. 🔧 What it does behind the scenes: It creates a column like this in the Business table: sql Copy Edit owner\_id INTEGER REFERENCES user(id) And allows you to do: Forward Access (from Business to User): python Copy Edit business = await Business.get(id=1) owner = await business.owner # This fetches the related User object Reverse Access (from User to Business list): python Copy Edit user = await User.get(id=1) businesses = await user.businesses.all() # Thanks to related\_name="businesses" ✅ Real Example: python Copy Edit # Create a user user = await User.create(username="bob", email="bob@mail.com", password="123") # Create a business for that user business = await Business.create(business\_name="Bob's Burgers", owner=user) # Fetch owner from business owner = await business.owner print(owner.username) # "bob" # Fetch businesses from user user\_businesses = await user.businesses.all() print([b.business\_name for b in user\_businesses]) # ["Bob's Burgers"] """ class Product(Model): id = fields.IntField(pk=True, index=True) name = fields.CharField(max\_length=100, null=False, unique=True) category = fields.CharField(max\_length=30, index=True) original\_price = fields.DecimalField(max\_digits=12, decimal\_places=2) new\_price = fields.DecimalField(max\_digits=12, decimal\_places=2) percentage\_discount = fields.IntField() offer\_expiration\_date = fields.DateField(default=datetime.utcnow) product\_image = fields.CharField(max\_length=200, null=False, default="productDefault.jpg") business\_owner = fields.ForeignKeyField("models.Business", related\_name="products") # Pydantic Schemas user\_pydantic = pydantic\_model\_creator(User, name="User", exclude=("is\_verified",)) user\_pydanticIn = pydantic\_model\_creator(User, name="UserIn", exclude\_readonly=True, exclude=("is\_verified", "join\_date")) user\_pydanticOut = pydantic\_model\_creator(User, name="UserOut", exclude=("password",)) business\_pydantic = pydantic\_model\_creator(Business, name="Business") business\_pydanticIn = pydantic\_model\_creator(Business, name="BusinessIn", exclude\_readonly=True) product\_pydantic = pydantic\_model\_creator(Product, name="Product") product\_pydanticIn = pydantic\_model\_creator(Product, name="ProductIn", exclude=("percentage\_discount", "id")) """✅ 1. user\_pydantic → General Model (exclude is\_verified) python Copy Edit exclude=("is\_verified",) This model includes all fields from the User model except is\_verified. Useful if you don't want the is\_verified status exposed or accepted by default. Can be used for general views or documentation where this field shouldn't be visible. ✅ 2. user\_pydanticIn → Input Schema (for creating users) python Copy Edit exclude\_readonly=True exclude\_readonly=True excludes read-only fields, like: id (auto-generated) join\_date (has a default) This makes sure clients only provide fields they're allowed to set manually, like: username, email, password ✅ Used for: python Copy Edit @app.post("/users", response\_model=UserOut\_Pydantic) async def create\_user(user: UserIn\_Pydantic): ... ✅ 3. user\_pydanticOut → Output Schema (for returning users) python Copy Edit exclude\_readonly=("password",) This is a bit tricky — here, you're using exclude\_readonly=("password",) but it likely should be: python Copy Edit exclude=("password",) password is not a read-only field, so exclude\_readonly=("password",) won't exclude it. To remove password from the output, use: python Copy Edit exclude=("password",) ✅ Purpose: Prevent exposing sensitive fields (like passwords) when returning user data.""" this is my models module

**ChatGPT said:**

You've built a solid FastAPI application using Tortoise ORM and Pydantic for modeling and serialization, with features like:

* **JWT-based authentication**
* **Email verification**
* **Image upload and processing (with Pillow)**
* **Product management with business linkage**
* **Signal-based business creation on user registration**

Here’s a review highlighting strengths, concerns, and actionable suggestions:

**✅ Strengths**

1. **Well-structured use of Pydantic & Tortoise ORM**:
   * Your separation of user\_pydanticIn, user\_pydanticOut, etc., follows best practices and improves input/output clarity.
2. **JWT Auth Implementation**:
   * OAuth2 with Bearer token using FastAPI’s OAuth2PasswordBearer is correctly implemented.
3. **Image Handling with PIL and Security**:
   * You securely rename uploaded images using secrets.token\_hex.
   * Proper resizing and format validation ensure good hygiene.
4. **Signal Use with**@post\_save:
   * Automatically creating a business post user registration is efficient and DRY.
5. **Static Files Serving**:
   * Use of StaticFiles is appropriate for media delivery.

**⚠️ Issues / Areas for Improvement**

**1. Incorrect Field Reference in Product Creation**

In the endpoint:

python

CopyEdit

product\_obj = await Product.create(\*\*product, business = user)

This line will **fail** because Product expects a business\_owner, not business.

🔧 **Fix**:

python

CopyEdit

business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business)

**2. Incorrect Product Deletion Logic**

This line:

python

CopyEdit

product = await product.get(id=id),

* product is the model name here, not an instance (and there's a trailing comma making it a tuple).

🔧 **Fix**:

python

CopyEdit

product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete()

**3. Hardcoded URLs (localhost)**

This:

python

CopyEdit

logo\_path = "localhost:8000/static/images/"+logo

Will break in production.

🔧 **Fix**:  
Use a config value or construct dynamic URLs from the request:

python

CopyEdit

logo\_path = request.url\_for("static", path=f"images/{logo}")

**4. Error Handling in Upload Routes**

Current check:

python

CopyEdit

if extension not in ["png", "jpg"]:

Won’t catch .jpeg or .PNG.

🔧 **Fix**:

python

CopyEdit

allowed\_extensions = {"jpg", "jpeg", "png"} if extension.lower() not in allowed\_extensions:

**5. Security (Exposed Password Field)**

Although you’re hashing the password on registration, you must also:

* **Never expose it in any response**
* **Never store raw passwords (you already don’t – good!)**
* Ensure your user\_pydanticOut excludes it, which you’re doing.

Still, your user\_login returns user.email, user.username — if you're not filtering through a schema, ensure nothing sensitive leaks.

**🧠 Suggestions / Improvements**

**1. Organize Code into Routers**

Split routes into modules:

bash

CopyEdit

app/ ├── routers/ │ ├── auth.py │ ├── user.py │ ├── product.py │ └── upload.py

In main.py:

python

CopyEdit

app.include\_router(auth.router)

**2. Use Environment Variables**

Replace hardcoded values like:

python

CopyEdit

db\_url = "sqlite://database.sqlite3"

With:

python

CopyEdit

import os db\_url = os.getenv("DATABASE\_URL")

Use .env file + python-dotenv.

**3. Add Expiration to JWT Tokens**

Currently, your JWTs never expire. Include exp claim for security.

**4. Optional: Add Rate Limiting / Throttling**

Use packages like slowapi or API gateways if deployed.

**✅ Summary**

You're on a great track with this FastAPI project. It demonstrates practical full-stack backend implementation including:

* DB models
* Authentication
* Media uploads
* Automated email and business creation

If you resolve the mentioned issues (especially the product creation & deletion bugs and URL handling), it’ll be robust and production-ready.

Would you like me to refactor the code to fix the mentioned bugs and organize it into modules?

**You said:**

Server response Code Details 422 Error: Unprocessable Entity Response body Download { "detail": [ { "loc": [], "msg": "NOT NULL constraint failed: product.business\_owner\_id", "type": "IntegrityError" } ] } Response headers content-length: 109 content-type: application/json date: Mon,26 May 2025 22:01:35 GMT server: uvicorn this is a error i got

**ChatGPT said:**

Thanks — this error is very clear and ties directly to a key issue in your /products POST endpoint:

**❌ Error Explained**

json

CopyEdit

{ "detail": [ { "loc": [], "msg": "NOT NULL constraint failed: product.business\_owner\_id", "type": "IntegrityError" } ] }

This means:  
👉 **You tried to create a**Product**without providing a required field:**business\_owner\_id, which is the foreign key to the Business model.

**✅ Root Cause**

In your /products endpoint:

python

CopyEdit

@app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 product\_obj = await Product.create(\*\*product, business = user) # ❌ This is wrong

Here, you're passing business = user — but your model expects:

python

CopyEdit

business\_owner = fields.ForeignKeyField("models.Business", related\_name="products")

So the ORM is missing the business\_owner\_id value during insert, hence the NOT NULL error.

**✅ Fix: Correct the Business Lookup and Assignment**

Replace the line:

python

CopyEdit

product\_obj = await Product.create(\*\*product, business = user)

With:

python

CopyEdit

business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business)

**🔧 Full Corrected Endpoint:**

python

CopyEdit

@app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product\_data = product.dict(exclude\_unset=True) if product\_data["original\_price"] > 0: product\_data["percentage\_discount"] = ( (product\_data["original\_price"] - product\_data["new\_price"]) / product\_data["original\_price"] ) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product\_data, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} return {"status": "error", "detail": "Original price must be greater than 0"}

**✅ Also Double-check**

Ensure the product\_pydanticIn **does not** include business\_owner, which you already excluded:

python

CopyEdit

product\_pydanticIn = pydantic\_model\_creator(Product, name="ProductIn", exclude=("percentage\_discount", "id"))

If not excluded, and a user supplies business\_owner, it could lead to validation mismatch or security issues.

Let me know if you'd like help with testing or adding better error messages!

**You said:**

while testing the delete endpoint it runs correctly but when i use get product id endpoint it return s the deleted product

**ChatGPT said:**

Great observation — if the **DELETE** endpoint appears to run successfully but the **GET** endpoint still returns the deleted product, there are a few likely causes.

**🔍 Likely Causes**

**✅ 1. Incorrect Deletion Syntax**

In your current delete\_product endpoint:

python

CopyEdit

product = await product.get(id=id),

* ❌ You're calling get on product, which is a **variable** not defined yet.
* ❌ The trailing comma makes it a **tuple**, not a model instance.
* ❌ Then you call product.delete() on a **tuple**, which doesn’t do anything — but doesn’t raise an error either.

📌 **Result**: The product **never gets deleted**, but you think it did.

**✅ Solution: Fix the delete endpoint**

Update your /product/{id} delete route to:

python

CopyEdit

@app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() # ✅ Correct usage return {"status": "ok", "message": "Product deleted successfully"} raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Not authenticated to perform this action", headers={"WWW-Authenticate": "Bearer"} )

**🧪 Optional: Add Validation**

To ensure it’s actually deleted, modify the GET /product/{id} route:

python

CopyEdit

@app.get("/product/{id}") async def get\_product(id: int): try: product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(Product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } except: raise HTTPException(status\_code=404, detail="Product not found")

**✅ Summary**

* Fix the **product deletion** logic (currently it’s not executing properly).
* Ensure await product.delete() is called on a valid model instance.
* Add try/except to GET endpoint to catch requests for deleted products.

Would you like me to help write tests for these endpoints using FastAPI’s TestClient?

**You said:**

@app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} is this correct

**ChatGPT said:**

You're *very close*, but there's one critical issue in your delete\_product function:

**❌ Problem:**

You're calling:

python

CopyEdit

product.delete()

This is **not awaited**, and in async Python, failing to await an async function means it **won’t actually run**.

**✅ Fix:**

You need to await the deletion:

python

CopyEdit

await product.delete()

**✅ Corrected Version:**

python

CopyEdit

@app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() # ✅ This is the key fix return {"status": "ok", "message": "Product deleted successfully"} else: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Not authenticated to perform this action", headers={"WWW-Authenticate": "Bearer"} )

**✅ Optional Improvements:**

1. **Add Try/Except Block** in case the product ID does not exist:

python

CopyEdit

try: product = await Product.get(id=id) except: raise HTTPException(status\_code=404, detail="Product not found")

1. **Log or return some detail** in response, like the deleted product name.

Let me know if you'd like help writing a test case for this endpoint using pytest and httpx or TestClient.

Top of Form

Attach

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

28-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

ERROR: docker: 'docker buildx build' requires 1 argument Usage: docker buildx build [OPTIONS] PATH | URL | - Run 'docker buildx build --help' for more information this error shows when i do this docker build -t main

**ChatGPT said:**

The error you're seeing:

java

CopyEdit

ERROR: docker: 'docker buildx build' requires 1 argument

...indicates that you're likely using docker buildx build (explicitly or implicitly) **without specifying the required**PATH**to the build context**.

**Fix**

You need to specify the **path** to the Docker build context. Usually, if your Dockerfile is in the current directory, you should add a . at the end of the command:

bash

CopyEdit

docker build -t main .

Or, if you're explicitly using buildx:

bash

CopyEdit

docker buildx build -t main .

**Explanation**

* docker build -t main → Incorrect, because it’s missing the PATH.
* docker build -t main . → ✅ Correct.
* docker buildx build -t main . → ✅ Also correct (if using Buildx).

Let me know if you're using a custom Dockerfile name or building from a different context directory.

**You said:**

PS D:\Data Structures and Algorithms\Ecommerce> docker build -t ecommerce . [+] Building 0.1s (1/1) FINISHED docker:desktop-linux => [internal] load build definition from Dockerfile 0.0s => => transferring dockerfile: 31B 0.0s ERROR: failed to solve: the Dockerfile cannot be empty

**ChatGPT said:**

The error:

vbnet

CopyEdit

ERROR: failed to solve: the Dockerfile cannot be empty

...means that Docker **found a file named**Dockerfile**, but it's empty**—it has no content.

**🔧 How to Fix**

You need to **add valid instructions** to your Dockerfile. For example, a basic Node.js app Dockerfile might look like:

Dockerfile

CopyEdit

# Use a base image FROM node:18 # Set working directory WORKDIR /app # Copy package files and install dependencies COPY package\*.json ./ RUN npm install # Copy the rest of the app code COPY . . # Expose a port and define the command to run the app EXPOSE 3000 CMD ["npm", "start"]

Or, if you're using something else (like Python, Java, etc.), I can help write the appropriate Dockerfile for your stack. Just let me know the tech you're using (e.g., Node.js, Python Flask, etc.).

**✅ Next Steps**

1. Open your Dockerfile in a text editor.
2. Add appropriate content based on your application.
3. Save the file.
4. Run the build again:

bash

CopyEdit

docker build -t ecommerce .

Let me know what stack you're using if you'd like help writing the Dockerfile.

**You said:**

PS D:\Data Structures and Algorithms\Ecommerce> docker build -t ecommerce . [+] Building 0.1s (1/1) FINISHED docker:desktop-linux => [internal] load build definition from Dockerfile 0.0s => => transferring dockerfile: 101B 0.0s Dockerfile:3 -------------------- 1 | FROM python 2 | WORKDIR /app 3 | >>> COPY ./app 4 | CMD ["python3", "app.py"] -------------------- ERROR: failed to solve: dockerfile parse error on line 3: COPY requires at least two arguments, but only one was provided. Destination could not be determined whats wrong

**ChatGPT said:**

The error:

sql

CopyEdit

COPY requires at least two arguments, but only one was provided. Destination could not be determined

...means your COPY instruction is incomplete.

**🔧 Problem in Your Dockerfile**

Your line:

Dockerfile

CopyEdit

COPY ./app

...is missing the **destination** path inside the container.

**✅ Fix**

You need to specify **both** source and destination:

Dockerfile

CopyEdit

COPY ./app /app

Or if you want to copy app.py from a directory named app into the working directory (/app), do:

Dockerfile

CopyEdit

COPY ./app/app.py .

**📄 Corrected Dockerfile Example**

Here’s what your Dockerfile might look like:

Dockerfile

CopyEdit

FROM python WORKDIR /app # Copy the contents of the local 'app' folder into the container's '/app' folder COPY ./app /app # Run the app CMD ["python3", "app.py"]

**✅ Next Step**

After saving the corrected Dockerfile, re-run the build:

bash

CopyEdit

docker build -t ecommerce .

Let me know if you want to install dependencies (like with requirements.txt) or need the Dockerfile tailored further.

**You said:**

what if my progrma has multiple modules

**ChatGPT said:**

If your Python program has **multiple modules** (i.e. multiple .py files, possibly in subdirectories), the Dockerfile needs to:

1. **Copy the entire application directory** (not just a single file).
2. Optionally, **install dependencies** (e.g. from requirements.txt).
3. Set the correct **working directory** and **entry point**.

**✅ Recommended Project Structure**

Suppose your project structure looks like this:

markdown

CopyEdit

Ecommerce/

│

├── app/

│ ├── \_\_init\_\_.py

│ ├── app.py

│ ├── utils.py

│ └── models/

│ └── model.py

│

├── requirements.txt

├── Dockerfile

**✅ Dockerfile for a Multi-Module Python App**

Dockerfile

CopyEdit

FROM python:3.10-slim # Set working directory WORKDIR /app # Copy all project files (including subdirectories) COPY . . # Install Python dependencies RUN pip install --no-cache-dir -r requirements.txt # Run the main module CMD ["python", "app/app.py"]

Adjust the CMD line to point to your actual entry point (whichever script starts the app).

**🔄 Build and Run**

bash

CopyEdit

# Build the Docker image docker build -t ecommerce . # Run the container docker run -p 8000:8000 ecommerce

**✅ Notes**

* If you're using packages or relative imports in submodules, make sure you include \_\_init\_\_.py files where necessary.
* If using Flask, FastAPI, or Django, the run command (CMD) would change accordingly.

Let me know what your main file is and how you're organizing modules—I'll tailor the Dockerfile precisely.

**You said:**

D:\Data Structures and Algorithms\Ecommerce\app this is the path to the files

**ChatGPT said:**

Thanks! Based on your folder structure:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\app

And assuming that all your Python code (including app.py and other modules) is inside the app folder, here’s how you should set up your Dockerfile.

**✅ Recommended Directory Layout**

cpp

CopyEdit

Ecommerce/

├── app/

│ ├── app.py

│ ├── module1.py

│ ├── module2.py

│ └── \_\_init\_\_.py (optional but recommended)

├── requirements.txt (if you have it)

└── Dockerfile

**✅ Dockerfile (place this in**Ecommerce/**, alongside the**app/**folder)**

Dockerfile

CopyEdit

FROM python:3.10-slim # Set working directory inside the container WORKDIR /app # Copy contents of the local app folder to the container's /app directory COPY ./app /app # If you have dependencies COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Run the main script (adjust if your entry point is different) CMD ["python", "app.py"]

If you don’t have a requirements.txt, just remove the related lines.

**✅ Build and Run**

From D:\Data Structures and Algorithms\Ecommerce, run:

bash

CopyEdit

docker build -t ecommerce . docker run -p 8000:8000 ecommerce

Adjust the port if your app runs on a different one.

**❓Let Me Know:**

* What is the entry-point file? (app.py?)
* Do you have a requirements.txt?
* Are you using a web framework like Flask, FastAPI, or Django?

With that info, I can fine-tune this for you.

**You said:**

what should my dockerfile look like

**ChatGPT said:**

Thanks! Since your code is in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\app

I'll assume the following:

**✅ Assumptions**

* Your Python app lives **inside the**app/**folder**
* app.py is the **main entry point**
* You may or may not have a requirements.txt (I'll show both cases)
* You're working with **pure Python**, not Flask, Django, etc. (let me know if you are!)

**📄 Final Dockerfile (Place it in the Ecommerce/ folder — one level above**app/**)**

Dockerfile

CopyEdit

# Use a lightweight Python base image FROM python:3.10-slim # Set the working directory inside the container WORKDIR /app # Copy the entire app folder into the container COPY ./app /app # If you have dependencies, install them # If not, you can remove these two lines COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Set the default command to run your app CMD ["python", "app.py"]

**🧱 Directory Layout Should Be:**

cpp

CopyEdit

Ecommerce/

├── app/

│ ├── app.py

│ ├── other\_module.py

│ └── \_\_init\_\_.py (optional but good for packages)

├── requirements.txt (optional, if you have dependencies)

└── Dockerfile

**🚀 How to Build and Run**

Open PowerShell in D:\Data Structures and Algorithms\Ecommerce, then:

bash

CopyEdit

docker build -t ecommerce . docker run -p 8000:8000 ecommerce

You can omit the -p flag if your app doesn’t expose a web server.

**❗ If You Don't Have**requirements.txt

Then use this **simpler Dockerfile**:

Dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app COPY ./app /app CMD ["python", "app.py"]

Let me know if you're using Flask, FastAPI, Django, or something else, so I can customize it further!

**You said:**

python: can't open file '/app/app.py': [Errno 2] No such file or directory

**ChatGPT said:**

Thanks — the error:

bash

CopyEdit

python: can't open file '/app/app.py': [Errno 2] No such file or directory

...means Docker couldn't find the app.py file **inside the container's**/app/**directory**.

This usually happens when:

1. The path in the COPY command doesn't match your actual file layout.
2. You're assuming a structure that doesn't line up with where Dockerfile is placed.

**✅ Fix Step-by-Step**

Let’s go over this carefully using your actual structure:

Your local file structure is:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\

├── Dockerfile

├── app\

│ ├── app.py

│ ├── other\_module.py

This means:

* Your Dockerfile is in Ecommerce/
* Your code is in Ecommerce/app/

**✅ Correct Dockerfile**

Dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app # Copy everything inside ./app into /app in the container COPY app/ /app/ # Optional: install dependencies # COPY requirements.txt . # RUN pip install --no-cache-dir -r requirements.txt CMD ["python", "app.py"]

✅ This works because:

* You're telling Docker to copy the **contents** of the app/ folder (i.e., app.py, etc.) **into**/app/**inside the container**.
* Then, Docker runs python app.py inside /app.

**🧪 Try These Commands**

Run them from this folder:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce

1. **Build**:

bash

CopyEdit

docker build -t ecommerce .

1. **Run**:

bash

CopyEdit

docker run -it ecommerce

If your app starts a server or listens on a port (like with Flask or FastAPI), add -p:

bash

CopyEdit

docker run -p 8000:8000 ecommerce

Let me know what app.py actually does (e.g. console app, web server, etc.), and I’ll make sure the Dockerfile and run command match.

**You said:**

PS D:\Data Structures and Algorithms\Ecommerce> docker run -p 8000:8000 ecommerce >> python: can't open file '/app/app.py': [Errno 2] No such file or directory i have files names such as main.py, models.py, mail.py

**ChatGPT said:**

Got it — the issue is that your Dockerfile tries to run app.py, **but your main script is actually called**main.py, not app.py.

**✅ Fix**

Update your Dockerfile to run main.py instead:

Dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app # Copy all files from the local app/ folder into the container's /app folder COPY app/ /app/ # Optional: install dependencies # COPY requirements.txt . # RUN pip install --no-cache-dir -r requirements.txt # Run your main script CMD ["python", "main.py"]

**🧱 Your Folder Layout Should Look Like:**

arduino

CopyEdit

Ecommerce/

├── Dockerfile

├── app/

│ ├── main.py <- Your entry point

│ ├── models.py

│ ├── mail.py

│ └── \_\_init\_\_.py <- (optional but helpful)

**🚀 Rebuild & Run**

1. From inside Ecommerce/, rebuild the image:

bash

CopyEdit

docker build -t ecommerce .

1. Run the container:

bash

CopyEdit

docker run -p 8000:8000 ecommerce

Or if it's a CLI app (not a web server), just run:

bash

CopyEdit

docker run -it ecommerce

**✅ Summary**

* Your app starts with main.py, so Docker should run main.py, not app.py.
* Always ensure your CMD in the Dockerfile points to your actual entry point.

Let me know if you’re using any frameworks (like Flask or FastAPI), and I’ll adjust the setup accordingly.

**You said:**

Traceback (most recent call last): File "/app/main.py", line 1, in <module> from fastapi import FastAPI, HTTPException, Request, status, Depends ModuleNotFoundError: No module named 'fastapi'

**ChatGPT said:**

The error:

vbnet

CopyEdit

ModuleNotFoundError: No module named 'fastapi'

...means that **FastAPI isn’t installed** in your Docker container. You need to:

1. Create a requirements.txt that lists your dependencies (like fastapi, uvicorn, etc.).
2. Update your Dockerfile to install them.

**✅ Step 1: Create**requirements.txt

In your Ecommerce/ folder (next to the Dockerfile), create a file named requirements.txt with at least:

txt

CopyEdit

fastapi uvicorn[standard]

Add any other libraries you use as needed (e.g., pydantic, email-validator, etc.).

**✅ Step 2: Update**Dockerfile

Here’s your updated Dockerfile:

Dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app # Copy and install dependencies COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copy your application files COPY app/ /app/ # Run FastAPI app with Uvicorn CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

✅ This assumes:

* Your FastAPI instance is declared in main.py like this:

python

CopyEdit

app = FastAPI()

**✅ Step 3: Build and Run**

From the Ecommerce/ folder:

bash

CopyEdit

docker build -t ecommerce . docker run -p 8000:8000 ecommerce

Then visit: [http://localhost:8000](http://localhost:8000/)

**✅ Bonus: Test API Docs**

Once it’s running, FastAPI will serve interactive API docs at:

* <http://localhost:8000/docs>
* <http://localhost:8000/redoc>

Let me know if you’d like help generating the requirements.txt automatically or running in development mode.

**You said:**

aiofiles==24.1.0 aiosmtplib==3.0.2 aiosqlite==0.16.1 aniso8601==8.0.0 annotated-types==0.7.0 anyio==4.9.0 asyncpg==0.30.0 atlastk==0.13.4 bcrypt==4.0.1 blinker==1.9.0 certifi==2025.1.31 charset-normalizer==3.4.1 click==7.1.2 colorama==0.4.6 Cython==3.1.0 dnspython==2.7.0 email\_validator==2.2.0 fastapi==0.115.12 fastapi-mail==1.4.2 Flask==1.1.2 Flask-RESTful==0.3.8 Flask-SQLAlchemy==2.4.3 h11==0.16.0 httptools==0.6.4 idna==3.10 iso8601==2.1.0 itsdangerous==1.1.0 Jinja2==3.1.6 MarkupSafe==3.0.2 numpy==2.0.1 orjson==3.10.18 passlib==1.7.4 pillow==11.2.1 pydantic==2.11.4 pydantic-settings==2.9.1 pydantic\_core==2.33.2 PyJWT==2.10.1 pypika-tortoise==0.5.0 python-dotenv==1.1.0 python-multipart==0.0.20 pytz==2020.1 PyYAML==6.0.2 requests==2.32.3 six==1.15.0 sniffio==1.3.1 SQLAlchemy==1.3.18 starlette==0.46.2 tortoise==0.1.1 tortoise-orm==0.25.0 typing-inspection==0.4.0 typing\_extensions==4.13.2 urllib3==2.4.0 uvicorn==0.34.2 watchfiles==1.0.5 websockets==15.0.1 Werkzeug==1.0.1 this is my requirements.txt it is in D:\Data Structures and Algorithms\Ecommerce

**ChatGPT said:**

Perfect — since your requirements.txt is located in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\requirements.txt

…and your actual app code is in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\app\

You now have everything needed to build a working Docker image.

**✅ Final**Dockerfile

Make sure this is saved in:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\Dockerfile

Dockerfile

CopyEdit

FROM python:3.10-slim # Set working directory WORKDIR /app # Copy requirements and install dependencies COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copy the entire app folder into the container COPY app/ /app/ # Run the FastAPI app with uvicorn CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

This assumes main.py contains something like:

python

CopyEdit

from fastapi import FastAPI app = FastAPI()

**🧱 Directory Layout Recap**

cpp

CopyEdit

Ecommerce/

├── app/

│ ├── main.py

│ ├── models.py

│ ├── mail.py

│ └── \_\_init\_\_.py (optional)

├── requirements.txt

└── Dockerfile

**🚀 Build and Run the Docker Container**

Open PowerShell in D:\Data Structures and Algorithms\Ecommerce and run:

bash

CopyEdit

docker build -t ecommerce . docker run -p 8000:8000 ecommerce

Then go to: <http://localhost:8000/docs> to view your FastAPI Swagger UI.

**🛠 Troubleshooting Tips**

If it fails at runtime:

* Double-check main.py for app = FastAPI()
* If the main.py file is actually nested in a subfolder inside app/, let me know

Let me know if you'd like to enable hot reloading for development or deploy it to the cloud.

**You said:**

INFO: Started server process [1] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit) INFO: 172.17.0.1:41348 - "GET /docs HTTP/1.1" 200 OK INFO: 172.17.0.1:41348 - "GET /openapi.json HTTP/1.1" 200 OK

**ChatGPT said:**

🎉 **Awesome! Your FastAPI app is running successfully inside Docker!**

You can now access it from your browser at:

👉 <http://localhost:8000/docs>

That’s the **Swagger UI**, which FastAPI auto-generates for testing your API endpoints.

**✅ Summary of What’s Working:**

* ✅ Docker container built with all dependencies from requirements.txt
* ✅ main.py is running via Uvicorn
* ✅ FastAPI served at http://0.0.0.0:8000
* ✅ Swagger docs loaded successfully (/docs)
* ✅ OpenAPI schema loaded (/openapi.json)

**🚀 Next Steps (Optional)**

Here are a few things you might want to consider:

**🔁 Enable Auto Reloading (Dev Mode Only)**

Add this to your Dockerfile **only for development**:

Dockerfile

CopyEdit

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]

Note: For reload to work inside Docker, you'll also need to **mount your code into the container** using -v, like so:

bash

CopyEdit

docker run -p 8000:8000 -v "$(pwd)/app:/app" ecommerce

**🌐 Deploy to the Cloud**

When you're ready, you can deploy this image to:

* **Render**
* **Railway**
* **DigitalOcean App Platform**
* **AWS ECS or Lambda**
* **Azure Container Apps**
* **Google Cloud Run**

Let me know if you want help with:

* Adding a database like PostgreSQL
* Using .env for secrets
* Setting up email or auth
* Deploying to a platform

You're in great shape now. Well done!

**You said:**

This site can’t be reached The webpage at http://0.0.0.0:8000/ might be temporarily down or it may have moved permanently to a new web address. ERR\_ADDRESS\_INVALID this is showing

**ChatGPT said:**

Ah — good catch. The message:

**This site can’t be reached —**http://0.0.0.0:8000**— ERR\_ADDRESS\_INVALID**

...is expected **outside Docker**, because 0.0.0.0 is not a valid address from your **host machine's browser**.

**✅ The Fix**

**From your browser**, you should visit:

👉 [http://localhost:8000](http://localhost:8000/)  
or  
👉 http://127.0.0.1:8000

**✅ Why This Happens**

* Inside Docker, the FastAPI app runs on 0.0.0.0, which means **"listen on all interfaces"** inside the container.
* But your browser outside the container should use localhost or 127.0.0.1, which Docker maps to the right interface via the -p 8000:8000 flag.

**✅ Bonus Test**

If you're using PowerShell and want to confirm it's listening:

bash

CopyEdit

curl http://localhost:8000

Or to check Swagger docs:

bash

CopyEdit

curl http://localhost:8000/docs

Let me know if that works or if you want help testing an endpoint!

**You said:**

how could i bring it to provide the link of the local sever

**ChatGPT said:**

Great question!

To make your FastAPI app **automatically print the correct local link (e.g.**[http://localhost:8000](http://localhost:8000/)**)** when it starts — instead of just saying http://0.0.0.0:8000 — you can do **one of these options**:

**✅ Option 1: Print the Link Yourself (Simple & Clear)**

You can add a **startup event** to your FastAPI app in main.py:

python

CopyEdit

from fastapi import FastAPI app = FastAPI() @app.on\_event("startup") async def startup\_message(): print("🚀 FastAPI is running at http://localhost:8000 (or your Docker IP)")

This doesn’t change how Uvicorn runs, but **gives users a clear local address**.

**✅ Option 2: Add a Startup Script (Advanced Control)**

You could wrap Uvicorn with a custom Python script:

**🔹 Create a file called**run.py**:**

python

CopyEdit

import uvicorn if \_\_name\_\_ == "\_\_main\_\_": print("🚀 FastAPI is starting at http://localhost:8000") uvicorn.run("main:app", host="0.0.0.0", port=8000)

**🔹 Update your Dockerfile:**

Dockerfile

CopyEdit

CMD ["python", "run.py"]

This way, it prints a **friendly message with the local link** before launching the app.

**✅ Option 3: Use Uvicorn's**--log-level**with**info

While Uvicorn **always logs**http://0.0.0.0:8000, you can add clarity using your own logging or combining with option 1.

**💡 Bonus: Friendly Tip in Docker Logs**

Docker logs will always show the 0.0.0.0 version (inside the container). You can make it friendlier for team members by echoing a message during container startup.

Example:

Dockerfile

CopyEdit

CMD echo "🚀 App is running at http://localhost:8000" && uvicorn main:app --host 0.0.0.0 --port 8000

Let me know which method you like best — or if you want this printed as a clickable link in a dev environment!

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

29-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

how to dockerize my python program which have multiple modules and also which needs to run at localhost

**ChatGPT said:**

To dockerize a Python program with multiple modules that runs on localhost, you need to follow these general steps:

**🧩 Step 1: Project Structure**

Here’s an example Python project layout:

css

CopyEdit

my\_python\_app/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py

│ ├── module1.py

│ └── module2.py

├── requirements.txt

└── Dockerfile

* main.py is the entry point (e.g., with a Flask app or CLI).
* module1.py and module2.py are your custom modules.

**🐳 Step 2: Create**Dockerfile

Here’s a sample Dockerfile:

Dockerfile

CopyEdit

# Use an official Python runtime as a parent image FROM python:3.11-slim # Set the working directory WORKDIR /app # Copy requirements and install them COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copy the rest of your app COPY app/ ./app # Set the entrypoint (update if your entry point is different) CMD ["python", "app/main.py"]

**📦 Step 3: requirements.txt**

Make sure you have a requirements.txt file listing all dependencies:

txt

CopyEdit

flask requests

Use pip freeze > requirements.txt if unsure.

**🏗️ Step 4: Build the Docker Image**

Run the following in the root project directory (where your Dockerfile is):

bash

CopyEdit

docker build -t my-python-app .

**🚀 Step 5: Run the Docker Container (localhost accessible)**

If you're running a web server (e.g., Flask), you need to expose ports:

bash

CopyEdit

docker run -p 5000:5000 my-python-app

If main.py uses Flask, ensure it’s set to run on 0.0.0.0 so Docker exposes it outside:

python

CopyEdit

# Inside app/main.py from flask import Flask app = Flask(\_\_name\_\_) @app.route('/') def home(): return 'Hello from Docker!' if \_\_name\_\_ == '\_\_main\_\_': app.run(host='0.0.0.0', port=5000)

**🧪 Optional: Docker Compose (if you need DB or multi-service setup)**

Example docker-compose.yml:

yaml

CopyEdit

version: '3' services: web: build: . ports: - "5000:5000"

Run with:

bash

CopyEdit

docker-compose up

Would you like me to generate a working example main.py, module1.py, and requirements.txt to get you started?

**You said:**

ERROR: error during connect: Head "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/\_ping": open //./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.

**ChatGPT said:**

This error typically means that **Docker is either not running** or it's **misconfigured on your system**. The message:

perl

CopyEdit

ERROR: error during connect: Head "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/\_ping":

open //./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.

…is telling us Docker cannot connect to its engine (the backend service that runs containers).

**✅ Fixes Based on Your Platform**

**🪟 If You're on Windows**

This is common when:

1. Docker Desktop is not started.
2. Docker is set to use the wrong backend (e.g. Linux vs Windows containers).
3. The pipe file Docker uses for communication is missing or inaccessible.

**✅ Try the Following:**

1. **Start Docker Desktop**
   * Press Win + S, search for **Docker Desktop**, and run it.
   * Wait for the Docker whale icon in the taskbar to stop saying "Starting...".
2. **Switch to Linux Containers (if needed)**
   * Right-click the Docker icon in your system tray.
   * If it says "Switch to Linux containers...", click it.
   * If it says "Switch to Windows containers...", you’re already on Linux containers (which is what you want).
3. **Verify Docker is running**  
   Run this in Command Prompt or PowerShell:

bash

CopyEdit

docker info

If Docker is running, it will return system info. If not, it’ll show a similar connection error.

**🧰 If This Doesn’t Work:**

* Restart your computer.
* Uninstall and reinstall Docker Desktop from https://www.docker.com/products/docker-desktop.

**🧪 Test a Simple Container**

Once Docker is up, test it:

bash

CopyEdit

docker run hello-world

This should pull a test image and print a "Hello from Docker!" message.

Let me know your OS (Windows, macOS, or Linux) if you'd like more tailored steps.

**You said:**

This site can’t be reached The webpage at http://0.0.0.0:8000/ might be temporarily down or it may have moved permanently to a new web address. ERR\_ADDRESS\_INVALID this is shown at the site

**ChatGPT said:**

The error ERR\_ADDRESS\_INVALID when visiting http://0.0.0.0:8000/ means your **browser cannot reach**0.0.0.0**directly**, because:

🔴 0.0.0.0 is a *non-routable meta-address* used to tell a program to listen on all network interfaces. It’s **not a valid address** for a browser to connect to.

**✅ Correct Way to Access It**

If your Python app (inside Docker or not) is listening on:

python

CopyEdit

app.run(host="0.0.0.0", port=8000)

You should **open this URL instead** in your browser:

arduino

CopyEdit

http://localhost:8000/

or

cpp

CopyEdit

http://127.0.0.1:8000/

These point to your machine's loopback interface and are accessible from your browser.

**📦 When Using Docker**

If you started your container with something like:

bash

CopyEdit

docker run -p 8000:8000 my-python-app

Then the app running inside Docker on 0.0.0.0:8000 is mapped to your **host's**localhost:8000, so just open:

arduino

CopyEdit

http://localhost:8000/

**✅ Summary**

* 0.0.0.0 = bind to all interfaces (for servers to listen on).
* localhost or 127.0.0.1 = what clients (like browsers) should use.

Would you like me to check your Python code or Dockerfile setup to ensure everything is configured properly?

**You said:**

cant we do anything that makes the docker give us the link to the site

**ChatGPT said:**

Great question! Yes, there **are a few ways** to make Docker tell you where your app is accessible—especially when working with containers running web servers. Here are the most effective approaches:

**✅ Option 1: Use Docker Compose + Logs**

If you use **Docker Compose**, and your app logs a startup message like:

python

CopyEdit

print("Server running on http://0.0.0.0:8000")

You can easily **see the correct URL** (translated to localhost) by viewing the logs:

bash

CopyEdit

docker-compose up

Then Docker will show something like:

arduino

CopyEdit

Server running on http://0.0.0.0:8000

👉 You then know to open http://localhost:8000.

**✅ Option 2: Improve Python Logging (print proper URL)**

Update your Python code (main.py) to print the correct host:

python

CopyEdit

print("🌐 App running at: http://localhost:8000") app.run(host='0.0.0.0', port=8000)

This way, **Docker logs will show the correct URL**, not just 0.0.0.0.

**✅ Option 3: Use Docker CLI + Auto Port Mapping**

If you let Docker assign a random port:

bash

CopyEdit

docker run -p 0:8000 my-python-app

Then you can **ask Docker where it mapped the port**:

bash

CopyEdit

docker ps

Output:

perl

CopyEdit

CONTAINER ID IMAGE PORTS NAMES

abcd1234 my-python-app 0.0.0.0:49153->8000/tcp my-python-app

Now open:

arduino

CopyEdit

http://localhost:49153

**✅ Option 4: Use a Startup Script to Echo the URL**

Create a start.sh file:

bash

CopyEdit

#!/bin/sh echo "🌐 App will be available at: http://localhost:8000" python app/main.py

Update your Dockerfile to use it:

Dockerfile

CopyEdit

COPY start.sh . RUN chmod +x start.sh CMD ["./start.sh"]

Now the container logs will clearly say what URL to open.

**💡 TL;DR**

You **can’t open**http://0.0.0.0**in a browser**, but you *can*:

* Print or log http://localhost:8000
* Use Docker Compose for clearer logs
* Use random ports and docker ps to find them
* Use a startup script that prints the correct info

Would you like me to generate a ready-to-run Dockerfile, main.py, and start.sh that does this for your project?

**You said:**

lets go with option 4 FROM python:3.10-slim # Set working directory WORKDIR /app # Copy requirements and install dependencies COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copy the entire app folder into the container COPY app/ /app/ # Run the FastAPI app with uvicorn CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"] this is my docker file

**ChatGPT said:**

Great! Since you're using **FastAPI with Uvicorn**, and you'd like Docker to **print a helpful startup message with the correct URL**, let's implement **Option 4: a custom startup script**.

**✅ Here's How to Do It**

**📁 Folder Structure (updated)**

pgsql

CopyEdit

your\_project/

├── app/

│ ├── main.py

│ └── other\_module.py

├── requirements.txt

├── start.sh 👈 NEW

└── Dockerfile

**📝**start.sh**– Startup Script**

Create a file named start.sh in your root project folder with:

bash

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" # Start the app uvicorn main:app --host 0.0.0.0 --port 8000

**🐳 Updated**Dockerfile

Replace the last part of your Dockerfile with this:

dockerfile

CopyEdit

# Copy the startup script and make it executable COPY start.sh /app/start.sh RUN chmod +x /app/start.sh # Run the app via the script CMD ["sh", "/app/start.sh"]

Make sure you also update the earlier COPY app/ /app/ so that it doesn't overwrite the start.sh.

Your final Dockerfile should look like:

dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY app/ /app/ COPY start.sh /app/start.sh RUN chmod +x /app/start.sh CMD ["sh", "/app/start.sh"]

**🐳 Build and Run**

bash

CopyEdit

docker build -t my-fastapi-app . docker run -p 8000:8000 my-fastapi-app

Output will include:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

Now you know exactly where to go in your browser 🎯

Let me know if you want to add automatic reloading (--reload) or support for environment variables.

**You said:**

This site can’t be reached The webpage at http://0.0.0.0:8000/ might be temporarily down or it may have moved permanently to a new web address. ERR\_ADDRESS\_INVALID again this is showing at the site

**ChatGPT said:**

Got it — and no worries, this is a **very common misunderstanding** when working with local development + Docker + browsers.

**🛑 Problem Recap**

You're visiting this URL in your **browser**:

cpp

CopyEdit

http://0.0.0.0:8000

…but that’s **not a valid address to browse to**, because:

❗ 0.0.0.0 is **not accessible from a browser** — it’s a **server-side bind address** only.

**✅ The Fix: Use**localhost**or**127.0.0.1**Instead**

If you ran:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

…then in your **browser**, visit:

arduino

CopyEdit

http://localhost:8000

or

http://127.0.0.1:8000

✅ This will work.

**✅ Extra: Why This Works**

* uvicorn runs on 0.0.0.0:8000 **inside the container** — this binds to all interfaces in the container.
* The -p 8000:8000 Docker flag maps **container port 8000** to **your local machine’s port 8000**.
* You connect to it from your host using localhost:8000.

**💬 TL;DR**

In your browser, **never** use http://0.0.0.0:8000.

Instead, visit:

➡️ http://localhost:8000

Let me know if you want to make your start.sh script *also print* the localhost address (just for clarity).

**You said:**

docker run -p 8000:8000 my-fastapi-app isnt this the same command as earlier

**ChatGPT said:**

Yes — you're absolutely right — the command:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

**is correct** and **exactly what you should be running** to expose your FastAPI app from inside Docker to your local browser.

**🔄 Recap of What This Command Does:**

* -p 8000:8000: Maps your **host machine's port 8000** ⬅️ to the **container's port 8000**.
* my-fastapi-app: The name/tag of your Docker image.
* Inside your Docker container, uvicorn is running on 0.0.0.0:8000, meaning: "accept connections from anywhere."

**✅ So the Command is Fine**

What’s **not** okay is the URL you're trying to open:

❌ http://0.0.0.0:8000 (does **not** work in a browser — this is a server binding address)

✅ http://localhost:8000 (correct)

✅ http://127.0.0.1:8000 (also correct)

**✅ Fix Your**start.sh**(optional improvement)**

If you want to make it super clear for anyone reading logs, update start.sh to print the *right* URL:

bash

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" uvicorn main:app --host 0.0.0.0 --port 8000

**🎯 Final Action**

* ✅ Keep running: docker run -p 8000:8000 my-fastapi-app
* ✅ Open in browser: http://localhost:8000

Let me know if it's still not loading at that URL — we can test for port conflicts, firewall rules, or confirm main:app is properly wired.

**You said:**

are asking me to bash this command in docker terminal

**ChatGPT said:**

Ah — thanks for the clarification!

Let me explain it **step by step**, based on what you're seeing:

**🤔 What You Asked**

Are you asking me to bash this command in Docker terminal?

Not quite — let’s break it down clearly.

**✅ What You Should Be Doing**

You should run this command **in your own system's terminal**, **not** inside Docker:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

**🔄 What This Does:**

* It starts a container from your image my-fastapi-app.
* It maps container's port 8000 to your **host's** port 8000.
* It runs the FastAPI app (as defined in your Dockerfile).

**✅ Where to Run This**

* **On Windows**: Use **PowerShell**, **Command Prompt**, or **Windows Terminal**.
* **On macOS/Linux**: Use the **Terminal**.
* ❌ **Do NOT** run this *inside* a Docker container — this command is what creates and starts a container.

**✅ After Running the Command**

If you see logs like this:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

That means your FastAPI app is running.

Now open your browser and go to:

arduino

CopyEdit

http://localhost:8000

**🧪 Double Check**

If nothing shows up in the browser:

1. Ensure Docker is running.
2. Make sure nothing else is using port 8000.
3. Check that your main.py looks like this (roughly):

python

CopyEdit

# app/main.py from fastapi import FastAPI app = FastAPI() @app.get("/") def read\_root(): return {"message": "Hello from FastAPI"}

Would you like me to verify or help you test your main.py and structure just to be sure it’s set up correctly?

**You said:**

cant we do something so that when we run the docker image of this program it automatically gives us the corrected url

**ChatGPT said:**

Yes, absolutely — you can **automatically print the correct URL (**http://localhost:8000**)** every time you run your Docker container. This is exactly what the **startup script (**start.sh**)** is for.

Let’s finish setting it up so that running:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

automatically prints:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

**✅ Final Setup (Recap + Polished)**

**1. 📄**start.sh**– Startup Script**

Create this file at the **root of your project (outside the**app/**folder)**:

bash

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" uvicorn main:app --host 0.0.0.0 --port 8000

Make sure it’s executable:

bash

CopyEdit

chmod +x start.sh

**2. 🐳**Dockerfile**– Updated to Use the Script**

Update your Dockerfile like this:

Dockerfile

CopyEdit

FROM python:3.10-slim # Set the working directory WORKDIR /app # Install dependencies COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copy your FastAPI app code COPY app/ /app/ # Copy and set permission for the startup script COPY start.sh /app/start.sh RUN chmod +x /app/start.sh # Run the app using the startup script CMD ["sh", "/app/start.sh"]

**3. 🏗️ Build the Image**

bash

CopyEdit

docker build -t my-fastapi-app .

**4. 🚀 Run the Container**

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

And now you’ll see something like:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Uvicorn running on http://0.0.0.0:8000

**✅ Bonus: Why This Works**

* 0.0.0.0 is still the correct *bind address* for Docker.
* But localhost is the correct *URL* to show to the user.
* The script bridges that gap and shows exactly what the user should use.

Let me know if you want to make this dynamic (e.g., showing the mapped port if using -p 0:8000), or if you'd like to include other checks in the script (like dependency status or health checks).

**You said:**

ok if run the command on cmd where should be loaction

**ChatGPT said:**

Great question! When you run the Docker command from your terminal (like Command Prompt or PowerShell on Windows), your **current working directory** matters, especially when you’re **building the Docker image**.

Here’s how it works:

**✅ When Running the Docker Container**

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

This command runs an already built image (my-fastapi-app), so you can run it from **any directory**.

✅ **No special location required** to run the container — once the image is built.

**✅ When Building the Docker Image**

bash

CopyEdit

docker build -t my-fastapi-app .

This command **must be run from the root of your project**, where your Dockerfile is located.

**🔍 For example:**

Given this project structure:

css

CopyEdit

my-fastapi-project/

├── Dockerfile

├── start.sh

├── requirements.txt

└── app/

├── main.py

└── other\_module.py

You should run this from:

bash

CopyEdit

cd path/to/my-fastapi-project docker build -t my-fastapi-app .

The . at the end means “build from this folder.”

**📝 Summary**

| **Action** | **Where to Run It From** |
| --- | --- |
| docker build -t my-fastapi-app . | 📂 In the folder with your Dockerfile |
| docker run -p 8000:8000 my-fastapi-app | 📂 Anywhere |

Let me know if you'd like help checking that your project files are in the right place. You can paste a directory listing if you're unsure.

**You said:**

INFO: Started server process [1] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit) this is showing i want the correct link to show

**ChatGPT said:**

Got it! You want **the logs to show the user-friendly URL**, i.e.,

arduino

CopyEdit

http://localhost:8000

instead of

cpp

CopyEdit

http://0.0.0.0:8000

**Why does Uvicorn print**0.0.0.0**?**

* Uvicorn logs the **bind address** (0.0.0.0 means "listen on all interfaces"), which is correct internally.
* But from the user perspective, when accessing the app **from your host machine**, you use localhost.

**How to print the *correct URL* on startup alongside Uvicorn logs?**

You can do this by **printing a custom message** right before starting the app in your startup script or in your Python code.

**🔥 Quick fix using your startup script (**start.sh**)**

If you use the startup script as we discussed:

bash

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" uvicorn main:app --host 0.0.0.0 --port 8000

You’ll see this message before Uvicorn’s own logs:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

**✅ Alternative: Print the URL inside your Python app**

You can add a print in your FastAPI app’s startup event:

python

CopyEdit

from fastapi import FastAPI app = FastAPI() @app.on\_event("startup") async def startup\_event(): print("\n🚀 FastAPI app starting...") print("🌐 Visit: http://localhost:8000\n") @app.get("/") async def root(): return {"message": "Hello from FastAPI"}

When your container starts, the message appears before Uvicorn logs:

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

**Summary:**

* **You can’t change the Uvicorn log line** itself (it always prints the bind address).
* But you can **print a user-friendly message** right before it starts, either in the shell script or inside FastAPI’s startup event.
* This gives the *correct URL* to the user every time.

Want me to help you set up either the startup script or the Python code snippet?

**You said:**

where will i see this message

**ChatGPT said:**

Great question! Here’s exactly **where and how you’ll see the message**:

**Where do you see the printed startup messages?**

* The messages printed by your startup script (start.sh) or your FastAPI app (e.g., using print()) will show up in the **container’s standard output (stdout)**.
* When you run the container **in the terminal** (e.g., docker run -p 8000:8000 my-fastapi-app), **all logs and prints appear in your terminal window**.

**How to see the message?**

1. **Run your container from your terminal:**

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

1. **Look at the terminal output** — you’ll see:

yaml

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Started server process [1]

INFO: Waiting for application startup.

INFO: Application startup complete.

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

**What if you run the container detached (**-d**)?**

If you do:

bash

CopyEdit

docker run -d -p 8000:8000 my-fastapi-app

The container runs in the background, so you won’t see output immediately.

To check logs, run:

bash

CopyEdit

docker logs <container\_id\_or\_name>

This will show the startup messages and Uvicorn logs, including your printed URL.

**Summary:**

| **Run Command** | **Where you see logs** |
| --- | --- |
| docker run -p 8000:8000 ... | Right in your terminal (immediate output) |
| docker run -d -p 8000:8000 ... | Run logs with docker logs <container\_id> after startup |

If you want, I can help you with commands to find the container ID and view logs!

**You said:**

C:\Users\Asus>docker run -p 8000:8000 my-fastapi-app INFO: Started server process [1] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit) this is showing in my cmd prompt

**ChatGPT said:**

Perfect! That means your container is running and your FastAPI app started successfully.

**Why don't you see the custom message like:**

arduino

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

Most likely **your**start.sh**script is not running**, so Uvicorn is being launched directly from the Dockerfile CMD without your message.

**How to fix it**

Make sure you:

1. Have a start.sh file in your project root (where Dockerfile is):

bash

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" uvicorn main:app --host 0.0.0.0 --port 8000

1. Your Dockerfile **copies** the script and runs it as the container command:

dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY app/ /app/ COPY start.sh /app/start.sh RUN chmod +x /app/start.sh CMD ["sh", "/app/start.sh"]

1. **Rebuild your Docker image** after making these changes:

bash

CopyEdit

docker build -t my-fastapi-app .

1. Run the container again:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

**After this, your CMD output should include the message:**

yaml

CopyEdit

🚀 FastAPI app starting...

🌐 Visit: http://localhost:8000

INFO: Started server process [1]

INFO: Waiting for application startup.

INFO: Application startup complete.

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

If you want, I can help you check your files or give you a full working example to copy-paste.

**You said:**

D:\Data Structures and Algorithms\Ecommerce this is the locatoin of the start file while D:\Data Structures and Algorithms\Ecommerce this is the locaton of dockerfile is this location correct

**ChatGPT said:**

Yes, **both files are in the same folder** (D:\Data Structures and Algorithms\Ecommerce), so that’s the **correct setup** for your Docker build context.

**What this means for building:**

* When you run this command from inside D:\Data Structures and Algorithms\Ecommerce:

bash

CopyEdit

docker build -t my-fastapi-app .

Docker will:

* See the Dockerfile in that folder.
* Copy start.sh into the image (since you have COPY start.sh /app/start.sh).
* Copy your app files (if you also do COPY app/ /app/).
* Build the image with the start.sh script included.

**Just make sure:**

* Your start.sh is named exactly start.sh (not start.sh.txt).
* start.sh has execute permission (chmod +x start.sh — this is less critical on Windows, but your Dockerfile RUN chmod +x /app/start.sh fixes it inside the image).
* You are running the build command **from the directory where both files are** (D:\Data Structures and Algorithms\Ecommerce).

**In summary:**

Your folder looks like:

makefile

CopyEdit

D:\Data Structures and Algorithms\Ecommerce\

├── Dockerfile

├── start.sh

├── requirements.txt

└── app\

├── main.py

└── ... other modules

This is perfect for building your Docker image.

**Final check before building:**

Run this in PowerShell or CMD inside the folder:

bash

CopyEdit

dir start.sh dir Dockerfile

Both should list the files.

Then build:

bash

CopyEdit

docker build -t my-fastapi-app .

Let me know if you want me to help check your files or commands!

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

30-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

import logging logging.basicConfig(filename='employee.log', level-logging.INFO, format='%(levelname)s:%(name)s:%(message)s') class Employee: """A sample Employee class""" def \_\_init\_\_(self, first, last): self.first = first self.last = last logging.info('Created Employee: {}{}'.format(self.fullname, self.email)) @property def email(self): return {}.{}@email.com'.format(self.first, self.last) @property def fullname(self): return '{) {}'.format(self.first, self.last) emp\_1 = Employee ('John', 'Smith') emp\_2 = Employee('Corey', 'Schafer') emp\_3= Employee('Jane', 'Doe')

**ChatGPT said:**

You're attempting to create a Python script that uses logging and defines an Employee class. However, your script has several **syntax and formatting errors** that will prevent it from running. Here's a corrected and working version of your code:

**✅ Corrected Code:**

python

CopyEdit

import logging # Configure logging logging.basicConfig(filename='employee.log', level=logging.INFO, format='%(levelname)s:%(name)s:%(message)s') class Employee: """A sample Employee class""" def \_\_init\_\_(self, first, last): self.first = first self.last = last # Logging employee creation logging.info('Created Employee: {} - {}'.format(self.fullname, self.email)) @property def email(self): return '{}.{}@email.com'.format(self.first, self.last) @property def fullname(self): return '{} {}'.format(self.first, self.last) # Creating Employee instances emp\_1 = Employee('John', 'Smith') emp\_2 = Employee('Corey', 'Schafer') emp\_3 = Employee('Jane', 'Doe')

**🔍 Summary of Fixes:**

1. **Incorrect logging level syntax:** Changed level-logging.INFO to level=logging.INFO.
2. **Indentation and docstring:** Fixed indentation for the class and docstring placement.
3. **Syntax in**email**and**fullname**:**
   * Fixed broken string in email return.
   * Corrected format string syntax in fullname (used '{) {}'.format(...) which was invalid).
4. **Removed extra spacing and fixed formatting for readability.**

If you run this corrected code, it will:

* Create three Employee objects.
* Log their creation to employee.log with their full name and email.

Let me know if you'd like to expand this class with more features!

**You said:**

one question is if we are already setting the level to info isnt it redundant to bash logging.info everytime is there any shartcut to it

**ChatGPT said:**

Great question! You're right to think about redundancy — let’s clarify how the logging system works and whether there’s a shortcut.

**✅ Understanding**logging.basicConfig(level=logging.INFO)

This **sets the minimum severity level** of messages that the logging system will handle. So with level=logging.INFO, any log message that is:

* DEBUG → **ignored**
* INFO or higher (WARNING, ERROR, CRITICAL) → **processed and written**

However, **you still need to explicitly call**logging.info() (or logging.warning(), etc.) to **create** the log message. The level setting does not automatically log all INFO-level events — it just tells the logger: *“Only show messages of this level or higher.”*

**❓ So, is there a shortcut?**

Yes, but not exactly to avoid logging.info() — more to **simplify the syntax**. One common approach is to **create a logger object**, especially for larger applications:

python

CopyEdit

import logging logger = logging.getLogger(\_\_name\_\_) logger.setLevel(logging.INFO) handler = logging.FileHandler('employee.log') formatter = logging.Formatter('%(levelname)s:%(name)s:%(message)s') handler.setFormatter(formatter) logger.addHandler(handler) class Employee: def \_\_init\_\_(self, first, last): self.first = first self.last = last logger.info('Created Employee: %s - %s', self.fullname, self.email) @property def email(self): return '{}.{}@email.com'.format(self.first, self.last) @property def fullname(self): return '{} {}'.format(self.first, self.last)

This way, instead of calling logging.info(...), you just use logger.info(...), and if needed, you can configure different loggers per module.

**🧠 TL;DR:**

* **You must still call**logging.info() or similar to log a message.
* basicConfig(level=...) sets the **threshold**, not the action.
* For cleaner code, **use a named logger** (logger = logging.getLogger(\_\_name\_\_)) and reuse it.

Let me know if you want to automate logging inside classes even further (e.g., decorators or metaclasses)!

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

31-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

how to apply logging in your multi module project

**ChatGPT said:**

Applying logging in a **multi-module project** (e.g., a Java, Python, or any modularized codebase) requires a centralized, consistent configuration and strategy across modules. Below is a general guide tailored primarily to **Java** and **Python**, but the principles apply across languages.

**🧩 General Strategy**

**✅ 1. Choose a Logging Framework**

Use a widely supported logging library:

* **Java**: SLF4J with Logback or Log4j2
* **Python**: Built-in logging module or loguru
* **Node.js**: winston, pino

**💡 Java Example (Maven Multi-Module Project)**

**1. Set Up a Common Logging Dependency**

In the parent pom.xml, declare shared dependencies:

xml

CopyEdit

<dependencyManagement> <dependencies> <dependency> <groupId>org.slf4j</groupId> <artifactId>slf4j-api</artifactId> <version>2.0.9</version> </dependency> <dependency> <groupId>ch.qos.logback</groupId> <artifactId>logback-classic</artifactId> <version>1.4.14</version> </dependency> </dependencies> </dependencyManagement>

**2. Add to Each Module**

Each module that needs logging should add:

xml

CopyEdit

<dependencies> <dependency> <groupId>org.slf4j</groupId> <artifactId>slf4j-api</artifactId> </dependency> <dependency> <groupId>ch.qos.logback</groupId> <artifactId>logback-classic</artifactId> </dependency> </dependencies>

**3. Logging Configuration File**

Create logback.xml (in resources directory of your parent or main module):

xml

CopyEdit

<configuration> <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"> <encoder> <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern> </encoder> </appender> <root level="INFO"> <appender-ref ref="STDOUT" /> </root> </configuration>

**4. Use Logging in Classes**

java

CopyEdit

import org.slf4j.Logger; import org.slf4j.LoggerFactory; public class ExampleService { private static final Logger logger = LoggerFactory.getLogger(ExampleService.class); public void process() { logger.info("Processing something..."); } }

**🐍 Python Example (Package-Based Project)**

**1. Create a Central Logging Config**

In a shared module or logging\_config.py:

python

CopyEdit

import logging import logging.config LOGGING\_CONFIG = { 'version': 1, 'disable\_existing\_loggers': False, 'formatters': { 'default': { 'format': '[%(asctime)s] %(levelname)s in %(name)s: %(message)s', }, }, 'handlers': { 'console': { 'class': 'logging.StreamHandler', 'formatter': 'default', }, }, 'root': { 'level': 'INFO', 'handlers': ['console'], }, } def setup\_logging(): logging.config.dictConfig(LOGGING\_CONFIG)

**2. Initialize at Entry Point**

In main.py or app.py:

python

CopyEdit

from shared.logging\_config import setup\_logging def main(): setup\_logging() logger = logging.getLogger(\_\_name\_\_) logger.info("App started") if \_\_name\_\_ == "\_\_main\_\_": main()

**3. Use Logging in Modules**

python

CopyEdit

import logging logger = logging.getLogger(\_\_name\_\_) def run\_task(): logger.debug("Running a task")

**📌 Best Practices**

* **Centralize config**: One config file (XML for Logback, dict for Python).
* **Use logger per class/module**: Always LoggerFactory.getLogger(ClassName.class) or getLogger(\_\_name\_\_).
* **Avoid printing to stdout**: Only use logging.
* **Environment-specific configs**: Use external files or env vars for log levels and destinations (e.g., file, console, log aggregator).
* **Test logs in CI** to ensure each module logs properly in integrated builds.

Would you like me to generate a sample project structure (Java or Python) with this applied?

**You said:**

give an example of a multi module project in python where and how should i use logging

**ChatGPT said:**

Here's a full example of a **multi-module Python project** with proper **logging setup**. This will show:

1. 📁 Project structure
2. 🛠️ Logging configuration
3. ✅ How to use logging in different modules
4. 🚀 Where to initialize logging

**📁 Project Structure**

plaintext

CopyEdit

my\_project/ ├── main.py ├── logging\_config/ │ └── logger.py ├── module\_a/ │ ├── \_\_init\_\_.py │ └── service\_a.py ├── module\_b/ │ ├── \_\_init\_\_.py │ └── service\_b.py └── requirements.txt

**🛠️ 1. Logging Configuration (**logging\_config/logger.py**)**

python

CopyEdit

# logging\_config/logger.py import logging import logging.config def setup\_logging(): logging\_config = { 'version': 1, 'disable\_existing\_loggers': False, 'formatters': { 'standard': { 'format': '[%(asctime)s] %(levelname)s [%(name)s]: %(message)s' }, }, 'handlers': { 'console': { 'level': 'DEBUG', 'class': 'logging.StreamHandler', 'formatter': 'standard' }, }, 'root': { 'handlers': ['console'], 'level': 'INFO', }, } logging.config.dictConfig(logging\_config)

**🧩 2. Module A (**module\_a/service\_a.py**)**

python

CopyEdit

# module\_a/service\_a.py import logging logger = logging.getLogger(\_\_name\_\_) def process\_data(): logger.info("Module A: Starting data processing.") # Simulate work logger.debug("Module A: Processing step 1...") logger.info("Module A: Finished processing.")

**🧩 3. Module B (**module\_b/service\_b.py**)**

python

CopyEdit

# module\_b/service\_b.py import logging logger = logging.getLogger(\_\_name\_\_) def fetch\_data(): logger.info("Module B: Fetching data from source.") # Simulate fetch logger.warning("Module B: Data source is slow.") logger.info("Module B: Data fetched.")

**🚀 4. Entry Point (**main.py**)**

python

CopyEdit

# main.py from logging\_config.logger import setup\_logging from module\_a.service\_a import process\_data from module\_b.service\_b import fetch\_data def main(): setup\_logging() fetch\_data() process\_data() if \_\_name\_\_ == '\_\_main\_\_': main()

**✅ How It Works**

* setup\_logging() is only called once in the main.py (entry point).
* Each module gets its own logger using getLogger(\_\_name\_\_) — this means logs will show their module path.
* You can now control **log level**, **format**, and **handlers** centrally in logger.py.

**🔧 Optional: Environment-Based Logging**

To make it more robust:

python

CopyEdit

import os level = os.getenv("LOG\_LEVEL", "INFO") # Then use `level` instead of hardcoded 'INFO' in logging config.

Would you like a zipped version of this structure or want to expand this to include file logging or error handling?

**You said:**

what is a better practice: to put logs while writing your code or to introduce the logs after everything is done

**ChatGPT said:**

Great question — the **best practice is to add logging *while writing your code***, not after everything is done.

Here's why:

**✅ Why Logging Should Be Added While Writing Code**

**1. Catches Issues Early**

* You can see how the code behaves as you build it.
* Logs help debug problems in real-time, especially when writing complex logic or integrating modules.

**2. Better Context**

* You have a deeper understanding of what the function is doing **as you're writing it**, so:
  + You write more meaningful messages.
  + You place logs in the right spots (not just after-the-fact guesses).

**3. Enables Test Logging**

* While writing unit or integration tests, meaningful logs help trace what's wrong when tests fail.

**4. Avoids Log Dumping**

* Adding logs later often leads to dumping generic logs everywhere like "Function entered" or "Processing started" — instead of **targeted, helpful logs**.

**🆚 What Happens If You Add Logging Later?**

| **Risk** | **Description** |
| --- | --- |
| 🧠 Loss of context | You forget key parts of the logic and place less useful logs. |
| 🪤 Overlogging | You might log everything to be safe, leading to noise. |
| 😵‍💫 Underlogging | You might miss edge cases because you only log success or failure. |
| 🔍 Harder debugging | When things break, you don’t know what happened unless you retrofit logs. |

**🔧 Recommended Approach**

| **When** | **What to Log** |
| --- | --- |
| While writing code | Key operations, decisions, branches, and potential failures. |
| During testing | Add debug logs if tests are failing or unclear. |
| Before release | Review and clean up: remove noisy debug logs, downgrade/upgrade levels appropriately. |

**✍️ Example (While Writing Code)**

python

CopyEdit

def process\_order(order): logger.info("Processing order ID: %s", order.id) if not order.items: logger.warning("Order ID %s has no items!", order.id) return False try: total = calculate\_total(order) logger.debug("Calculated total: %s", total) except Exception as e: logger.exception("Error calculating total for order ID %s", order.id) return False logger.info("Successfully processed order ID %s", order.id) return True

**✅ TL;DR**

**Add logging as you write code**, just like you would add comments or write tests. Logging is part of writing maintainable, observable software — not a final polish.

Let me know if you'd like a checklist or logging style guide too!

**2/2**

Get smarter responses, upload files and images, and more.

Log in

Sign up for free

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

31-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

in python lets say we made some objects(Car1, Car2, Car3, ....) from a particular class(Car) can we do something like if Car.(id=3):

**ChatGPT said:**

In Python, you **can't use a class like**Car**and then do**Car.(id=3) as in your example. That syntax is invalid. However, there are several **valid and clean ways** to achieve the effect you're aiming for: checking if a car with a specific id exists among instances.

Here are some common approaches, depending on how you're managing the Car objects:

**✅ Option 1: Store Instances in a List or Dictionary**

If you're creating multiple Car instances, store them in a collection:

**Using a List**

python

CopyEdit

class Car: def \_\_init\_\_(self, id): self.id = id # Create car objects car1 = Car(1) car2 = Car(2) car3 = Car(3) # Store in a list cars = [car1, car2, car3] # Check if a car with id=3 exists if any(car.id == 3 for car in cars): print("Car with id 3 exists!")

**Using a Dictionary (More Efficient Lookup)**

python

CopyEdit

cars = { 1: Car(1), 2: Car(2), 3: Car(3) } if 3 in cars: print("Car with id 3 exists!")

**✅ Option 2: Track All Instances Automatically**

If you want all Car instances to automatically register themselves, you can use a class variable:

python

CopyEdit

class Car: instances = [] def \_\_init\_\_(self, id): self.id = id Car.instances.append(self) # Create cars Car(1) Car(2) Car(3) # Check for a car with id=3 if any(car.id == 3 for car in Car.instances): print("Car with id 3 exists!")

**🚫 Invalid Example (Just to Clarify)**

python

CopyEdit

# This will raise a SyntaxError or AttributeError if Car.(id=3): ...

Python doesn’t support that syntax—Car is a class, not a container or a queryable object by default.

Would you like help writing a version that auto-registers cars and can fetch by ID efficiently?

**You said:**

no i mean if Car.get(id=3):

**ChatGPT said:**

Ah! That makes much more sense — you're asking if you can do something like:

python

CopyEdit

Car.get(id=3)

This **can work**, but only if you **define** a get method on the Car class yourself. Python classes don’t have a built-in get() method like that unless you explicitly implement it.

**✅ How to Implement**Car.get(id=3)

You can make a class method that looks up instances by id. Here’s a clean way to do it:

python

CopyEdit

class Car: \_instances = {} def \_\_init\_\_(self, id): self.id = id Car.\_instances[id] = self @classmethod def get(cls, id): return cls.\_instances.get(id) # Create some cars Car(1) Car(2) Car(3) # Now you can do: car = Car.get(3) if car: print(f"Found car with ID: {car.id}") else: print("Car not found.")

**✅ Explanation**

* \_instances: A dictionary where keys are ids and values are Car instances.
* Car.get(id): A class method that returns the instance with the given ID or None if not found.
* Works just like Django ORM's .get() or .filter() if you're familiar with that.

**🔒 Optional Safety Enhancements**

You can add checks to ensure IDs are unique, or raise errors if needed:

python

CopyEdit

def \_\_init\_\_(self, id): if id in Car.\_instances: raise ValueError(f"Car with id {id} already exists.") self.id = id Car.\_instances[id] = self

Would you like to expand this idea to support filtering by other attributes, like color or model?

**You said:**

does tortoise orm pydantic model creator have one on them

**ChatGPT said:**

Yes, Tortoise ORM's Pydantic integration provides a way to create Pydantic models from Tortoise models, but it doesn't include a built-in get() method for querying instances by ID. However, you can easily implement this functionality yourself.

**🔧 Implementing a**get()**Method**

To retrieve a Car instance by its ID, you can use Tortoise ORM's query methods. Here's how you can do it:

python

CopyEdit

from tortoise import fields from tortoise.models import Model class Car(Model): id = fields.IntField(pk=True) name = fields.CharField(max\_length=100) # Retrieve a car by ID car = await Car.get(id=3)

This approach uses Tortoise ORM's get() method to fetch a single instance by its primary key.

**🛠️ Creating a Pydantic Model**

If you want to create a Pydantic model from your Car model for serialization purposes, you can use Tortoise ORM's pydantic\_model\_creator function:

python

CopyEdit

from tortoise.contrib.pydantic import pydantic\_model\_creator Car\_Pydantic = pydantic\_model\_creator(Car)

This generates a Pydantic model that you can use to serialize Car instances.[tortoise.github.io+1tortoise.github.io+1](https://tortoise.github.io/contrib/pydantic.html?utm_source=chatgpt.com)

**🧩 Combining Both: Fetching and Serializing**

To fetch a Car instance by ID and serialize it using the generated Pydantic model, you can do the following:

python

CopyEdit

car = await Car.get(id=3) car\_pydantic = await Car\_Pydantic.from\_tortoise\_orm(car)

This retrieves the Car instance with ID 3 and then creates a Pydantic model instance from it.

**⚠️ Handling Missing Instances**

If there's a possibility that no Car instance exists with the given ID, you should handle this case to avoid exceptions:

python

CopyEdit

try: car = await Car.get(id=3) car\_pydantic = await Car\_Pydantic.from\_tortoise\_orm(car) except Car.DoesNotExist: car\_pydantic = None # Handle the case where the car doesn't exist

This approach ensures that your code gracefully handles the absence of the requested Car instance.

In summary, while Tortoise ORM's Pydantic integration doesn't provide a get() method, you can easily implement this functionality using Tortoise ORM's query methods and combine it with Pydantic serialization for your needs.

Sources

**You said:**

if await User.filter(username=user\_info["username"]).exists(): this line is in my project

**ChatGPT said:**

Great! That line is a **perfectly valid and common pattern** in **Tortoise ORM** when you're checking if a user already exists in the database. Here's what it's doing and a couple of tips:

**🔍 Explanation of Your Line**

python

CopyEdit

if await User.filter(username=user\_info["username"]).exists():

* User.filter(...): Builds a query that filters rows in the User table where the username matches the given value.
* .exists(): Checks **if any** record matches the filter — it doesn't retrieve the data, just returns True or False.
* await: Since .exists() is an async method, you need to await it (Tortoise ORM is async-first).

✅ This is efficient because .exists() translates to a SQL query like:

sql

CopyEdit

SELECT EXISTS(SELECT 1 FROM user WHERE username = 'some\_user');

**✅ When to Use**.exists()

Use this when you want to **check for existence only**, and don't need the actual data yet. It's more efficient than:

python

CopyEdit

user = await User.filter(username=...).first() if user: ...

because .first() fetches the row, whereas .exists() does not.

**🧠 Example Use Case**

python

CopyEdit

user\_info = {"username": "alice"} if await User.filter(username=user\_info["username"]).exists(): print("User already exists.") else: await User.create(username=user\_info["username"]) print("User created.")

Let me know if you'd like help wrapping this into a reusable function or validating with Pydantic before querying!

**You said:**

does the class have an in-built method to search for its objects

**ChatGPT said:**

Yes, in **Tortoise ORM**, all model classes like your User class **do have built-in methods** for searching/querying their objects. These methods are provided by the Tortoise ORM base Model class.

Here are the **main built-in search/query methods**:

**🔍**filter(\*\*kwargs)

Returns a queryset filtered by the given conditions. Can return 0 or more objects.

python

CopyEdit

users = await User.filter(is\_active=True)

**🔍**get(\*\*kwargs)

Returns a **single object** matching the condition. Raises DoesNotExist or MultipleObjectsReturned if not exactly one match.

python

CopyEdit

user = await User.get(username="alice")

Use try/except to safely handle it:

python

CopyEdit

try: user = await User.get(username="alice") except User.DoesNotExist: user = None

**🔍**first()**,**all()**,**limit()**,**offset()**,**order\_by()

These are queryset chaining methods:

python

CopyEdit

user = await User.filter(active=True).order\_by('-created\_at').first()

**🔍**exists()

Returns True if any record matches the query:

python

CopyEdit

if await User.filter(email="test@example.com").exists(): ...

**🔍**get\_or\_none()

Returns one object or None (instead of raising an exception):

python

CopyEdit

user = await User.get\_or\_none(username="alice")

**✅ Summary**

Yes — Tortoise ORM model classes come with powerful, Django-like **query capabilities** baked in, including:

* filter()
* get()
* get\_or\_none()
* exists()
* first()
* ...and more

These methods allow you to **search/query objects** in a very expressive and efficient way.

Would you like a cheat sheet or real-world example combining several of these methods in one flow?

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

31-05-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) main module of a project

**ChatGPT said:**

You've shared the main module of a **FastAPI** project using **Tortoise ORM**, JWT-based **authentication**, **image upload**, **email verification**, and a full **CRUD system for products and businesses**. It’s structured well, with endpoints handling:

* **User registration & login**
* **Email verification**
* **Profile & product image uploads**
* **Product CRUD**
* **Business update**
* **Token-based authentication**

Here’s a review with **suggestions**, **corrections**, and **enhancements** you can make:

**✅ General Observations**

**✔ Strengths:**

* Good use of Depends() for authentication.
* Clean JWT integration.
* Image handling with resizing and secure filenames.
* Use of Tortoise post\_save signal for auto business creation.
* Static file mounting and HTML response for verification pages.
* Use of pydantic for request/response models.

**🛠️ Suggestions for Improvement**

**1. Bug:**user\_login**returning wrong model**

You're returning user: user\_pydanticIn from Depends(get\_current\_user), but get\_current\_user returns a **User model instance**.

**✅ Fix:**

Change:

python

CopyEdit

@app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)):

To:

python

CopyEdit

@app.post("/user/me") async def user\_login(user: User = Depends(get\_current\_user)):

And similarly update other endpoints that use user\_pydantic.

**2. Bug:**delete\_product**has incorrect indentation**

python

CopyEdit

if user == owner: await product.delete() #return {"status": "YAYA"}

**✅ Fix:**

python

CopyEdit

if user == owner: await product.delete()

**3. Insecure domain in URLs**

You're returning paths like localhost:8000/static/images/.... In deployment, this should be dynamic.

**✅ Suggest:**

Use an environment variable or FastAPI request.url\_for for generating full URLs dynamically.

**4. Better Extension Checking**

python

CopyEdit

extension = filename.split(".")[1]

This can crash if filename doesn’t contain a period.

**✅ Fix:**

python

CopyEdit

if "." not in filename: raise HTTPException(status\_code=400, detail="Invalid file name") extension = filename.rsplit(".", 1)[-1].lower()

**5. Better Error Handling in**generate\_token

Add logging or detailed feedback for failed login attempts.

**✅ Suggest:**

python

CopyEdit

except Exception as e: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid credentials" )

**6. Use Constants for Static Paths**

Refactor "./static/images/" into a constant like:

python

CopyEdit

STATIC\_IMAGE\_DIR = os.path.join(BASE\_DIR, "static", "images")

Then use:

python

CopyEdit

generated\_name = os.path.join(STATIC\_IMAGE\_DIR, token\_name)

**7. Missing**await**in**business.save()

In the /business/{id} endpoint:

python

CopyEdit

business.save()

Should be:

python

CopyEdit

await business.save()

**🧼 Optional Cleanups**

* Use .lower() for checking email uniqueness.
* Use FastAPI logging module to log errors or important events.
* Create a separate router (APIRouter) for auth, products, etc., for modularity.

**🧪 Suggested Tests**

Here are the key areas to write automated tests for:

* ✅ Registration + email verification flow
* ✅ Login and token issuance
* ✅ Upload image (valid and invalid extensions)
* ✅ CRUD product operations with auth
* ✅ Unauthorized access attempts

**🧾 Final Tip: Use**.env**for sensitive configs**

Move values like SECRET and database URL to a .env file and use python-dotenv or pydantic.BaseSettings.

Would you like a **folder structure**, **Docker setup**, or **unit test templates** for this project as well?

**You said:**

@post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # send email await send\_email([instance.email], instance) here tell me how do created is assigned

**ChatGPT said:**

Great question!

In the @post\_save(User) function:

python

CopyEdit

@post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None:

You're using **Tortoise ORM's**post\_save**signal**, and you're asking: **Where does the**created**value come from?**

**🔍 Explanation of**created**in**post\_save

When Tortoise ORM saves a model (like User), it triggers the post\_save signal. Tortoise passes the following to the signal handler:

* sender: the model class (User)
* instance: the instance that was saved
* created: a boolean indicating if this was a **new** record (True) or an **update** to an existing record (False)
* using\_db: the DB connection used
* update\_fields: fields that were updated (if any)

**✅ So, how is**created**assigned?**

Tortoise determines created based on whether the object was **just inserted** or **updated**.

Behind the scenes, when you do:

python

CopyEdit

user\_obj = await User.create(\*\*user\_info)

Tortoise:

1. Inserts a **new row** into the DB.
2. Internally recognizes this is a **new object**.
3. Fires post\_save with created=True.

If you instead called .save() on an existing object, like:

python

CopyEdit

existing\_user.email = "newemail@example.com" await existing\_user.save()

Then Tortoise would fire post\_save with created=False.

**📌 Summary**

* created=True → When .create() or .save() inserts a new DB row.
* created=False → When .save() updates an existing row.

This allows your signal to take different actions based on whether a record is new (e.g., create a related Business object and send a welcome email).

Let me know if you’d like a visual flow of this process or a test example to see it in action.

**You said:**

so created value is assigned by tortoise itself but how we declared the created variable

**ChatGPT said:**

Excellent follow-up question! Let's break this down.

**❓ You're asking:**

“If created**is assigned by Tortoise ORM**, **where** or **how** did we declare the created variable in the post\_save function?”

**✅ Short Answer:**

You didn't explicitly **declare** or **assign** created in your code.  
You **defined it as a parameter** of the signal function:

python

CopyEdit

async def create\_business( sender: "Type[User]", instance: User, created: bool, # 👈 here! using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None:

Tortoise ORM automatically **injects** (calls the function with) that created value when the post\_save event occurs.  
So, **you define the parameter**, and **Tortoise supplies the value** when calling the function.

**🧠 Think of it like this analogy:**

You're defining the function like this:

python

CopyEdit

def on\_event(event\_type: str, user: str): print(f"{user} did {event\_type}")

You don't assign event\_type and user—you just define them as parameters. Whoever **calls** the function passes the values.

That’s what Tortoise is doing under the hood when it triggers your signal.

**🔧 Under the Hood:**

Tortoise’s internal signal dispatcher does something like this:

python

CopyEdit

await your\_signal\_function( sender=User, instance=user\_instance, created=True, using\_db=some\_connection, update\_fields=[] )

**🧾 Summary:**

* ✅ You declare created: bool as a parameter in the signal function.
* 🚀 Tortoise ORM **injects the value** (True if .create() was called, False if .save() updated).
* 🧼 You never assign created—Tortoise handles that behind the scenes.

Would you like a simple logging example to see this created value in action during runtime?

**You said:**

do need to write the created value as parameter

**ChatGPT said:**

✅ **Yes**, you **do need to write**created**as a parameter** in your signal function **if you want to use it** — for example, to check if a User was just created and take some action (like creating a Business or sending an email).

**🔍 Why?**

Tortoise ORM uses the function's **parameter list** to know **what data you want access to**. It can pass the following parameters to a post\_save signal:

* sender: the model class (e.g., User)
* instance: the saved instance
* created: True if a new object was created, False if updated
* using\_db: the database connection used
* update\_fields: the fields that were updated (if any)

**🧪 What if you *don’t* include**created**?**

If you leave it out:

python

CopyEdit

@post\_save(User) async def create\_business(sender, instance, using\_db, update\_fields): # no `created` parameter here ...

Then Tortoise **won’t pass** created into your function, and you **won’t be able to check** whether the object was newly created or just updated. No error will occur — you'll just **miss out on that context**.

**✅ So when should you include it?**

Include created if you want to do something **only when the object is first created**, like:

python

CopyEdit

if created: # do something special

If you don’t need that logic, you can omit it.

**🔚 Summary:**

| **Do you *have* to write**created**?** | **Answer** |
| --- | --- |
| Is it **required** by Tortoise? | ❌ No |
| Is it **optional** to include? | ✅ Yes |
| Should you include it if you need to know whether it was a new object? | ✅ Yes |

Would you like a sample log to demonstrate how created behaves differently on .create() vs .save()?

**You said:**

here instance: User shouldnt it be something like instance: User.username

**ChatGPT said:**

Great question! Let's clarify this — it's a common confusion when dealing with signals and object instances in Python ORMs like **Tortoise ORM**.

**🔍 You're asking:**

Shouldn't it be instance: User.username instead of instance: User?

**✅ Short Answer: No —**instance: User**is correct.**

Because in the signal, instance represents the **entire model object** that was saved — not a single field like .username.

**🧠 Explanation:**

In the signal function:

python

CopyEdit

@post\_save(User) async def create\_business( sender: "Type[User]", instance: User, # 👈 This is the full User object created: bool, using\_db: Optional[BaseDBAsyncClient], update\_fields: List[str] ) -> None:

* instance is the **actual**User**object** that was just saved to the database.
* You can access any of its fields like:

python

CopyEdit

instance.username instance.email instance.id

**❌ What would happen if you wrote**instance: User.username**?**

That’s not valid Python syntax — User.username is a **field**, not a type.

In type hints (instance: ...), Python expects a **type**, like:

* int
* str
* User
* List[str]

So User.username would make no sense here, and would raise a NameError or AttributeError.

**✅ Correct Use (with example):**

Here’s how you use it properly:

python

CopyEdit

@post\_save(User) async def create\_business( sender: Type[User], instance: User, created: bool, using\_db: Optional[BaseDBAsyncClient], update\_fields: List[str] ) -> None: if created: print(f"New user created: {instance.username}") # 👈 access .username here

**✅ TL;DR**

| **What you wrote** | **Correct?** | **Why / Why Not** |
| --- | --- | --- |
| instance: User | ✅ Yes | instance is the full User model object |
| instance: User.username | ❌ No | Invalid syntax — User.username is a field, not a data type |
| How to access .username | ✅ | Inside the function, via instance.username, instance.email, etc. |

Let me know if you'd like a print/log demo showing what instance looks like in runtime!

**You said:**

so here owner is the object itself

**ChatGPT said:**

✅ **Exactly!** In your code, owner is the **object itself**, specifically an instance of the User model.

Let’s break this down with a concrete example from your code:

**📍 Example:**

python

CopyEdit

business = await Business.get(owner=user) owner = await business.owner # 👈 This returns the full User object

Here’s what’s happening:

1. business.owner is a **ForeignKey relationship** to the User model.
2. So await business.owner returns the **full**User**object** that owns that business.
3. owner.username, owner.email, etc., are now accessible.

**🧠 Think of it this way:**

If your models look like this:

python

CopyEdit

class User(Model): username = fields.CharField(...) email = fields.CharField(...) class Business(Model): business\_name = fields.CharField(...) owner = fields.ForeignKeyField("models.User", related\_name="businesses")

Then business.owner is a **foreign key field** (a link to a user), and await business.owner gives you the actual User object.

**✅ Real-world example:**

Imagine this user is in your database:

json

CopyEdit

{ "id": 1, "username": "john\_doe", "email": "john@example.com" }

When you do:

python

CopyEdit

business = await Business.get(id=5) owner = await business.owner print(owner.username) # 👈 prints "john\_doe"

**✅ Summary**

| **Expression** | **Type** | **Description** |
| --- | --- | --- |
| business.owner | ForeignKeyField | A relation link to the User model |
| await business.owner | User instance | The actual User object that owns it |
| owner.username | str | The username of the business owner |

So yes — in your code, owner is indeed the full User object.

Let me know if you'd like a diagram to visualize how these model relationships work!

**You said:**

yes show the diagram

**ChatGPT said:**

Great! Here's a simple diagram that visually explains the relationship between User and Business models using **Tortoise ORM**, specifically how owner refers to the full User object.

**🧩 Models Relationship Diagram (ForeignKey from Business → User)**

yaml

CopyEdit

┌──────────────┐ owns ┌────────────────────┐

│ User │◄──────────────────────┤ Business │

├──────────────┤ ├────────────────────┤

│ id: 1 │ │ id: 101 │

│ username: │ │ business\_name: │

│ "john\_doe" │ │ "John's Store" │

│ email: │ │ owner\_id: 1 │

│ "john@..." │ └────────────────────┘

└──────────────┘

* **ForeignKey Field**: Business.owner points to User.id.
* When you run: await business.owner, you get the full User object (john\_doe in this case).

**🧠 In Code Terms**

python

CopyEdit

user = await User.get(id=1) business = await Business.create(business\_name="John's Store", owner=user) # Later... business = await Business.get(id=101) owner = await business.owner # 👈 This gives you back the full `User` object print(owner.username) # prints: "john\_doe"

**📝 How the ForeignKey works internally:**

* owner\_id is stored in the Business table as a foreign key.
* business.owner is a **relationship manager**, which Tortoise uses to fetch the related User when you await it.

Would you like a version of this diagram as an image or want to add more models (e.g., Product)?

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

02-06-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: #LOG creatin and saving a business profile of user in database bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # LOG the sending of email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # If already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # If not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is main module

**ChatGPT said:**

Your main.py FastAPI module is structured quite well and covers a comprehensive set of features for a small-scale e-commerce or product management app, including:

* **User registration and authentication with JWT**
* **Business and product management**
* **Image uploads for profile and product**
* **Email verification with HTML templates**
* **Pydantic for data validation**
* **CRUD operations with Tortoise ORM**
* **Signal handling with**@post\_save

**✅ Here’s a quick checklist of what's working well:**

1. **JWT Token-based authentication** — implemented securely using OAuth2PasswordBearer and a tokenUrl.
2. **Email verification** with token and HTML rendering using Jinja2Templates.
3. **File/Image Upload** using UploadFile, saved and resized with Pillow.
4. **Use of Signals** (@post\_save) to automatically create business objects post-registration.
5. **Proper Pydantic use** for request and response models.
6. **CRUD APIs** for Product and Business.
7. **Static file serving** and well-defined directory paths.

**🔍 Issues & Recommendations:**

**1. Variable Name Bug:**

In add\_new\_product:

python

CopyEdit

product = product.dict(exclude\_unset = True)

This overwrites the input model. **Fix:**

python

CopyEdit

product\_data = product.dict(exclude\_unset=True)

**2. Inconsistent naming –**business\_owner**vs**business**:**

In your Product model, you refer to both product.business and product.business\_owner. Stick to one for clarity.

**3. Missing return or redundant line:**

In:

python

CopyEdit

business.save()

This is not awaited. Use:

python

CopyEdit

await business.save()

**4. Bug in delete\_product():**

python

CopyEdit

if user == owner: await product.delete() #return {"status": "YAYA"}

Indentation issue. Fix:

python

CopyEdit

if user == owner: await product.delete() return {"status": "ok"}

**5. Security: Hardcoded**localhost**in image URLs**

python

CopyEdit

"logo": "localhost:8000/static/images/" + logo

Replace with dynamic host:

python

CopyEdit

from fastapi.requests import Request # Inside endpoint logo\_path = request.url\_for("static", path="images/" + logo)

**6. Image format validation:**

You check only:

python

CopyEdit

if extension not in ["png", "jpg"]:

But extensions can be uppercase (.JPG). Fix:

python

CopyEdit

if extension.lower() not in ["png", "jpg", "jpeg"]:

**7. Improper user data return type**

Your get\_current\_user returns ORM User, not user\_pydantic. Fix:

python

CopyEdit

user\_obj = await User.get(id=payload.get("id")) return await user\_pydantic.from\_tortoise\_orm(user\_obj)

Or: If you're intentionally returning the ORM object, ensure consistency in how you use it.

**8. Verification token function not shown:**

You use very\_token(token) — this method is not defined in the code you posted. Make sure it exists and handles expiration properly.

**✅ Suggested Enhancements**

* **Logging:** Consider integrating logging module to replace # LOG comments.
* **Rate limiting / brute force protection** for login/token endpoints.
* **Password reset endpoint** — useful for production.
* **Tests:** Add pytest + httpx tests for endpoints.

**📁 Directory Structure (Suggested):**

cpp

CopyEdit

.

├── main.py

├── models.py

├── authentication.py

├── mail.py

├── static/

│ └── images/

├── templates/

│ ├── verification.html

│ └── already\_verified.html

├── database.sqlite3

Let me know if you want help with:

* Defining models.py
* Adding tests
* JWT expiration handling
* Dockerizing this app

Or anything else.

**You said:**

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user how does this work and when it is needed

**ChatGPT said:**

The generate\_token and get\_current\_user functions together form the **authentication system** of your FastAPI app using **OAuth2 + JWT** (JSON Web Tokens). Here's a breakdown of how they work and **when** and **why** you use them:

**🔐 1.**@app.post("/token")**→**generate\_token

**✅ Purpose:**

To **authenticate a user** (login) and return a **JWT access token**.

**🧠 How It Works:**

* FastAPI provides OAuth2PasswordRequestForm to handle form-encoded login data (username, password).
* You pass that form to a token\_generator() function (defined elsewhere).
* The token\_generator validates the credentials, and if correct, returns a **JWT token**.

**💡 Example Request:**

http

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=admin&password=secret

**✅ Example Response:**

json

CopyEdit

{ "access\_token": "eyJhbGciOiJIUzI1NiIsInR...", "token\_type": "bearer" }

**🧑‍💼 2.**get\_current\_user()**→ Dependency Function**

**✅ Purpose:**

To extract the user **from the JWT token** for **authentication** in protected endpoints.

**🧠 How It Works:**

1. The token is automatically extracted from the Authorization: Bearer <token> header using Depends(oauth2\_scheme).
2. The JWT is decoded using your SECRET key.
3. It retrieves the user from the DB using the ID from the token payload.
4. If any step fails, it raises a 401 Unauthorized.

**🔁 How They Work Together**

1. **Login:**
   * Client sends username and password to /token.
   * Gets a JWT access token.
2. **Protected Endpoints:**
   * The client sends the JWT token in the Authorization header:

makefile

CopyEdit

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...

* + get\_current\_user() runs automatically (via Depends()).
  + If valid, the current user is injected into the route function.

**🔐 When You Use This**

You use get\_current\_user() in any endpoint that should only be accessible by **logged-in/authenticated users**.

**🔒 Example:**

python

CopyEdit

@app.get("/dashboard") async def dashboard(user: user\_pydantic = Depends(get\_current\_user)): return {"message": f"Welcome, {user.username}"}

If the token is missing or invalid, the user gets a 401 Unauthorized.

**✅ Summary**

| **Function** | **Role** | **When it's Called** |
| --- | --- | --- |
| generate\_token | Logs in the user | When client POSTs credentials to /token |
| get\_current\_user | Authenticates JWT, loads user | Automatically used in protected routes |

If you'd like, I can show you a working example with the token\_generator() implementation or explain how to secure certain routes more effectively.

**You said:**

what is a protected endpoiint

**ChatGPT said:**

A **protected endpoint** is an API route that requires the user to be **authenticated** (logged in) before they can access it. This usually means the user must provide a **valid JWT token** in the request headers.

**🔐 Why Use Protected Endpoints?**

To ensure that **only authorized users** can:

* Access sensitive data (e.g., their account, order history)
* Modify content (e.g., update profile, delete a product)
* Perform business actions (e.g., upload a product image)

**🔐 Example of a Protected Endpoint**

python

CopyEdit

from fastapi import Depends @app.get("/user/profile") async def get\_profile(user: user\_pydantic = Depends(get\_current\_user)): return {"message": f"Welcome {user.username}"}

**✅ What Makes It Protected:**

* It uses Depends(get\_current\_user) to **require a valid JWT**.
* If the user **doesn’t include a valid token**, FastAPI automatically returns:

json

CopyEdit

{ "detail": "Not authenticated" }

**🔓 Example of an Open (Unprotected) Endpoint**

python

CopyEdit

@app.get("/") def homepage(): return {"message": "Welcome to our public site!"}

Anyone can access this — no login/token required.

**🛡️ How the Protection Works**

1. **Client logs in** at /token and gets a JWT token.
2. **Client includes token** in the Authorization header of requests:

makefile

CopyEdit

Authorization: Bearer <your\_token\_here>

1. **FastAPI + Depends(get\_current\_user)**:
   * Extracts and verifies the token.
   * Loads the user from the database.
   * Injects the user into the route.

If anything fails (missing/invalid/expired token), the request is rejected.

**🔐 Common Use Cases for Protected Endpoints**

| **Endpoint** | **Why It's Protected** |
| --- | --- |
| /user/me | Shows private user info |
| /uploadfile/profile | Uploads profile picture for logged-in user |
| /products (POST) | Only logged-in users can create products |
| /product/{id} (DELETE/PUT) | Only the owner should be able to edit/delete |

Let me know if you'd like a visual diagram or real API test examples with curl, Postman, or Python requests.

**You said:**

tell me what can we do by goin in the token endpoint and login our credentials what does it do and if it logs the user in then fast api have other ways to do so so why is it necessary

**ChatGPT said:**

Great question — you're getting at the **core of how authentication works in APIs**.

**🔑 What Happens When You "Log In" via the**/token**Endpoint?**

When you POST your credentials to /token, here's what really happens:

1. **Your credentials (username & password) are verified.**
2. If valid, a **JWT (JSON Web Token)** is generated.
3. The token is returned to the client.

json

CopyEdit

{ "access\_token": "<JWT\_TOKEN>", "token\_type": "bearer" }

1. The client then **includes this token in all future requests** to protected endpoints using:

makefile

CopyEdit

Authorization: Bearer <JWT\_TOKEN>

**⚙️ So What Does This Mean?**

You're not "logged in" like in a browser session with cookies. Instead:

* You're **issued a token** (like a key or pass).
* This token **proves your identity** in every request.
* FastAPI reads this token to **authenticate** and **authorize** you.

**🤔 Why Is**/token**Necessary?**

Because APIs are **stateless** — they don’t remember who you are between requests.

FastAPI doesn't track sessions unless you build that manually. Instead, JWT-based authentication is used for stateless systems like:

* Mobile apps
* Single-page applications (React, Vue, etc.)
* RESTful APIs

**🔐 Why Not Just "Log In" a Different Way?**

**Option 1: Session Cookies**

* Used in traditional websites (Django, Flask with templates).
* Browser stores a session cookie.
* Good for full-page HTML apps.

**FastAPI does not do this by default.** It's designed more for APIs than full websites.

**Option 2: Basic Authentication**

* You send username/password with every request.
* Not secure unless you're using HTTPS, and it's inefficient.

**Option 3: JWT (What you're using)**

* More secure, stateless, widely used in modern web apps.
* Easy to validate with secret key.
* Token can carry useful payload (like user ID, role, expiration).

**🧠 Summary**

| **Action** | **Result** |
| --- | --- |
| POST to /token | Verifies credentials, gives back JWT token |
| Use token in headers | Proves you're logged in, allows protected access |
| Why this approach is needed | Because APIs are stateless — they don’t remember |
| Why not traditional login? | APIs don’t use sessions like browser-based sites |

If you're building a public website where users log in with a form and navigate pages, you might prefer **session-based auth**.

But if you're building an API for a frontend app, mobile app, or third-party client, **JWT via**/token**is the right choice**.

Let me know your project context — I can suggest the best fit.

**You said:**

ok how can we manually use this token

**ChatGPT said:**

Great! Let’s walk through **how to manually use a JWT token** after getting it from the /token endpoint — whether you're testing it with **Postman**, **curl**, or **Python code**.

**✅ Step-by-Step: Manual Token Usage**

**🔐 Step 1: Log in and Get the Token**

Send a POST request to /token with your credentials:

**Using**curl**:**

bash

CopyEdit

curl -X POST http://localhost:8000/token \ -H "Content-Type: application/x-www-form-urlencoded" \ -d "username=your\_username&password=your\_password"

**Response:**

json

CopyEdit

{ "access\_token": "eyJhbGciOiJIUzI1NiIsInR...", "token\_type": "bearer" }

Copy the access\_token.

**🔒 Step 2: Use the Token on a Protected Endpoint**

Now, send a request to any **protected endpoint**, such as /user/me, and include the token in the Authorization header.

**🛠️ Example Methods to Use the Token**

**✅ 1. Using**curl**:**

bash

CopyEdit

curl -X POST http://localhost:8000/user/me \ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR..."

**✅ 2. Using Postman:**

1. Go to your endpoint (e.g., POST /user/me)
2. Click on the "Authorization" tab.
3. Select:
   * **Type:** Bearer Token
   * **Token:** Paste the token
4. Send the request

**✅ 3. Using Python**requests**:**

python

CopyEdit

import requests token = "eyJhbGciOiJIUzI1NiIsInR..." headers = { "Authorization": f"Bearer {token}" } response = requests.post("http://localhost:8000/user/me", headers=headers) print(response.json())

**🧠 How FastAPI Handles It**

* The get\_current\_user() dependency is triggered.
* It extracts and decodes the token.
* If valid, it passes the user object to the endpoint.
* If not, FastAPI returns:

json

CopyEdit

{"detail": "Not authenticated"}

**🧪 Test It Yourself**

Try:

* Calling /token with bad credentials — you'll get a 401.
* Sending /user/me without a token — you'll also get a 401.
* Then send it with a valid token — it works!

**💬 Let Me Know:**

If you’d like:

* An example using JavaScript/React
* A way to auto-refresh tokens
* Or building your own token validator

I’d be happy to show you!

**You said:**

) return await user here cant we write return user

**ChatGPT said:**

Yes, **you technically can write**return user**instead of**return await user — but **only if**user**is not a coroutine**.

Let's look at the context of your code:

python

CopyEdit

try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return await user

**🧠 What’s Going On?**

* user = await User.get(...) means you're **already awaiting the database call**.
* At this point, user is no longer a coroutine — it's a fully loaded object.
* Therefore, await user **again** is unnecessary and incorrect (and may even raise an error in some Python versions).

**✅ Correct Version:**

python

CopyEdit

return user

No need to await it again.

**✅ Summary**

| **Expression** | **Explanation** |
| --- | --- |
| await User.get() | Needed — async DB call |
| await user | ❌ Wrong — user is already resolved |
| return user | ✅ Correct — return the object |

**✅ Final Correct Version:**

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except: raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid username or password", headers={"WWW-AUTHENTICATE": "Bearer"} ) return user # ✅ this is correct

Let me know if you'd like to convert this returned ORM user into a Pydantic model too.

**You said:**

app.mount("/static", StaticFiles(directory="static"), name="static") what is the use of this

**ChatGPT said:**

Great question! This line in your FastAPI code:

python

CopyEdit

app.mount("/static", StaticFiles(directory="static"), name="static")

is used to **serve static files** (like images, CSS, JavaScript) from a folder on your server.

**🧱 What It Does:**

* It **mounts** a folder named static/ in your project directory.
* Any files inside that folder become accessible via URL starting with /static.

**📁 Folder Structure Example:**

cpp

CopyEdit

your\_project/

├── main.py

├── static/

│ └── images/

│ └── logo.png

With this setup, a file at static/images/logo.png becomes accessible at:

bash

CopyEdit

http://localhost:8000/static/images/logo.png

**💡 Use Cases in Your Project**

In your code, you're using this to:

* Save and serve **uploaded profile and product images**.

python

CopyEdit

FILEPATH = "./static/images/"

* Return image URLs like:

python

CopyEdit

"logo": "localhost:8000/static/images/" + logo

Without app.mount(...), FastAPI would not serve those image files — it only handles API routes unless told otherwise.

**🔐 Security Tip**

Don't mount folders that contain private or sensitive files unless you want them to be publicly accessible. The /static route is fully open to the web.

**✅ Summary**

| **Feature** | **Purpose** |
| --- | --- |
| StaticFiles(...) | Tells FastAPI to serve static files |
| directory="static" | The folder that holds your files |
| mount("/static", ...) | URL path prefix for accessing files |

Let me know if you want help organizing your static files or using them in frontend templates!

**You said:**

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) what is the significance of these lines

**ChatGPT said:**

These two lines:

python

CopyEdit

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates"))

are used to configure **HTML template rendering** with **Jinja2** in your FastAPI application.

**🔍 Breakdown and Purpose**

**🔹**BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

This gets the **absolute path of the directory** where your current Python file (main.py) is located.

* \_\_file\_\_ → the current file's name (main.py)
* os.path.abspath(\_\_file\_\_) → full absolute path to the file
* os.path.dirname(...) → removes the filename, leaving the directory

✅ **Result:** BASE\_DIR is the full path to your project directory.

**🔹**templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates"))

This sets up Jinja2 with FastAPI by telling it **where to find your HTML template files**.

* It joins BASE\_DIR with "templates" → like /path/to/project/templates
* Then creates a Jinja2Templates instance using that folder.

✅ **This is required if you want to render HTML pages** in FastAPI using templates like this:

python

CopyEdit

return templates.TemplateResponse("verification.html", {"request": request, "username": user.username})

**📁 Project Structure Example:**

css

CopyEdit

your\_project/

├── main.py

├── templates/

│ ├── verification.html

│ └── already\_verified.html

**🧠 Why Use This?**

* So you can **render dynamic HTML pages**, like email verification messages, success pages, etc.
* Useful when building hybrid apps (HTML + API) or sending HTML in emails.

**✅ Summary**

| **Line** | **Purpose** |
| --- | --- |
| BASE\_DIR = ... | Get the root path of your project |
| Jinja2Templates(directory=...) | Tell FastAPI where your .html templates live |
| Used with TemplateResponse(...) | To render HTML pages with dynamic data |

Let me know if you want a sample HTML template or want to customize your email verification page!

**You said:**

@app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url}

**ChatGPT said:**

You're working with an image upload endpoint in FastAPI. Let's break down what this @app.post("/uploadfile/profile") function does, how it works, and what it's used for:

**🔍 What This Endpoint Does**

This route:

* Allows an **authenticated user** to upload a **profile image** (PNG or JPG).
* Saves the image to the ./static/images/ directory.
* Resizes the image using **Pillow** (to 200×200).
* Updates the user's Business.logo with the image filename.
* Returns the public URL for that image.

**🔁 Step-by-Step Breakdown**

**1. Authentication**

python

CopyEdit

user: user\_pydantic = Depends(get\_current\_user)

* This makes the route **protected** — only logged-in users with a valid JWT can upload.

**2. File Input**

python

CopyEdit

file: UploadFile = File(...)

* This allows the user to send an image file (from a form, frontend, etc.).

**3. Validate File Type**

python

CopyEdit

extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"}

* Only allows .png and .jpg files.
* You might want to also check for jpeg and lowercase extensions.

**4. Generate Secure Filename**

python

CopyEdit

token\_name = secrets.token\_hex(10)+"."+extension

* Prevents name collisions and potential overwrites.
* Example: a1b2c3d4e5f6.png

**5. Save File**

python

CopyEdit

with open(generated\_name, "wb") as file: file.write(file\_content)

* Saves the uploaded image in the static/images/ folder.

**6. Resize Image**

python

CopyEdit

img = Image.open(generated\_name) img = img.resize((200, 200)) img.save(generated\_name)

* Standardizes the image size using **Pillow**.

**7. Update Database**

python

CopyEdit

business = await Business.get(owner=user) business.logo = token\_name await business.save()

* Links the saved image filename to the user's Business.logo field.

**8. Return Image URL**

python

CopyEdit

file\_url = "localhost:8000" + generated\_name[1:]

* Returns something like:

json

CopyEdit

{ "status": "ok", "filename": "localhost:8000/static/images/a1b2c3d4e5f6.png" }

**⚠️ Important Notes & Improvements**

1. **File Extension Check:**  
   Add jpeg and convert extension to lowercase:

python

CopyEdit

extension = filename.split(".")[-1].lower()

1. **Full Image URL:**  
   You should include http:// or https://:

python

CopyEdit

file\_url = f"http://localhost:8000/static/images/{token\_name}"

1. **Security Tip:**  
   Validate file contents using file.content\_type or inspect the binary — not just file extension.
2. **Cleanup:**  
   No need to file.close() after with open(...); it's auto-handled.

**🧠 What This Enables**

This endpoint powers **profile image uploads** in your app. Users can:

* Upload and replace their profile logo.
* See that logo displayed elsewhere using the image URL.

Let me know if you'd like to:

* Limit file size
* Support deleting old images
* Upload product images to cloud storage (e.g., S3)

Happy to help!

**You said:**

in this whole main module do you think logging is necessary

**ChatGPT said:**

Yes — **logging is absolutely necessary**, especially in a production-level application like the one you're building.

Your FastAPI app has:

* User registration and authentication
* File uploads
* Database operations
* Email sending
* Protected endpoints

Without logging, when something fails (e.g., a user can't register, a file doesn't upload, or a DB operation fails), you’d be **flying blind**.

**✅ Why Logging Matters in Your App**

**1. Debugging**

When something breaks (a crash, 500 error, or invalid behavior), logs help you trace:

* Which endpoint was hit
* Which user caused it
* What exception occurred
* What input caused the problem

**2. Security**

You want to log:

* Unauthorized access attempts
* Suspicious activity (e.g., someone uploading unexpected file types)
* Login attempts (both success and failure)

**3. Audit Trails**

You might want to track:

* When users register
* When a product is added or deleted
* When a file is uploaded

This is especially useful if multiple admins are managing data.

**4. Performance Monitoring**

Logging how long things take (e.g., file uploads, DB queries) helps detect bottlenecks.

**🛠️ Where You Should Add Logging in Your Module**

**✅ Good spots to add logs:**

| **Section** | **Example Log** |
| --- | --- |
| /registration | New user registered: {username} |
| /token | Login attempt for: {username} |
| get\_current\_user | Token decoded for user ID: {user\_id} |
| /uploadfile/profile | Profile image uploaded for user: {user.username} |
| create\_business signal | Business created for new user: {username} |
| File extension check fails | File upload rejected: unsupported extension |
| Exception handling | Database error: {e} or JWT decode failed |

**🧰 How to Add Logging**

Add this at the top of your main.py:

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s" ) logger = logging.getLogger(\_\_name\_\_)

Then inside routes or handlers:

python

CopyEdit

logger.info(f"User registered: {new\_user.username}") logger.warning(f"Login failed for: {username}") logger.error(f"File upload failed: {str(e)}")

**🚀 Bonus: Log to File**

Instead of just printing to the console:

python

CopyEdit

logging.basicConfig( filename="app.log", filemode="a", level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s" )

**✅ Final Takeaway**

**Yes, you definitely should add logging** — it's essential for:

* Debugging
* Monitoring
* Security
* Support
* Auditing

Would you like me to help you plug logging into your code and show sample logs for each route?

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

ChatGPT says: Yes — \*\*logging is absolutely necessary\*\*, especially in a production-level application like the one you're building. Your FastAPI app has: - User registration and authentication - File uploads - Database operations - Email sending - Protected endpoints Without logging, when something fails (e.g., a user can't register, a file doesn't upload, or a DB operation fails), you’d be \*\*flying blind\*\*. --- ## ✅ Why Logging Matters in Your App ### 1. \*\*Debugging\*\* When something breaks (a crash, 500 error, or invalid behavior), logs help you trace: - Which endpoint was hit - Which user caused it - What exception occurred - What input caused the problem ### 2. \*\*Security\*\* You want to log: - Unauthorized access attempts - Suspicious activity (e.g., someone uploading unexpected file types) - Login attempts (both success and failure) ### 3. \*\*Audit Trails\*\* You might want to track: - When users register - When a product is added or deleted - When a file is uploaded This is especially useful if multiple admins are managing data. ### 4. \*\*Performance Monitoring\*\* Logging how long things take (e.g., file uploads, DB queries) helps detect bottlenecks. --- ## 🛠️ Where You Should Add Logging in Your Module ### ✅ Good spots to add logs: | Section | Example Log | |-------------------------------|--------------------------------------------------| | `/registration` | `New user registered: {username}` | | `/token` | `Login attempt for: {username}` | | `get\_current\_user` | `Token decoded for user ID: {user\_id}` | | `/uploadfile/profile` | `Profile image uploaded for user: {user.username}` | | `create\_business` signal | `Business created for new user: {username}` | | File extension check fails | `File upload rejected: unsupported extension` | | Exception handling | `Database error: {e}` or `JWT decode failed` | --- ## 🧰 How to Add Logging Add this at the top of your `main.py`: ```python import logging logging.basicConfig( level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s" ) logger = logging.getLogger(\_\_name\_\_) ``` Then inside routes or handlers: ```python logger.info(f"User registered: {new\_user.username}") logger.warning(f"Login failed for: {username}") logger.error(f"File upload failed: {str(e)}") ``` --- ## 🚀 Bonus: Log to File Instead of just printing to the console: ```python logging.basicConfig( filename="app.log", filemode="a", level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s" ) ``` --- ## ✅ Final Takeaway \*\*Yes, you definitely should add logging\*\* — it's essential for: - Debugging - Monitoring - Security - Support - Auditing Would you like me to help you plug logging into your code and show sample logs for each route?

03-06-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config #becomes available by http://localhost:8000/static/images/logo.png #improve for security purposes app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): #LOG generating token token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id = payload.get("id")) except: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: #LOG creatin and saving a business profile of user in database bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # LOG the sending of email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: # LOG already verified, return a message indicating they are already verified return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # LOG user was not verified, verify the user user.is\_verified = True await user.save() return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) #LOG error raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} #LOG upload pictures @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() #LOG image is saving with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() business = await Business.get(owner = user) owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) this is main module of a project

**ChatGPT said:**

Thank you for sharing your main FastAPI module. Here’s a **quick review and summary**, along with **suggestions** to improve **code structure, security, and functionality**:

**✅ Overview**

Your app includes:

* **User Registration and Authentication** (JWT-based).
* **Email verification system**.
* **Image upload** (with Pillow resizing).
* **CRUD for Products and Business**.
* **Static file serving** (e.g., logo uploads).
* **Signal-based business auto-creation** after user registration.

**✅ Strengths**

* ✅ **Separation of concerns**: models, authentication, and mail modules are used.
* ✅ **Post-save signal** to automate business creation.
* ✅ **Image processing** using PIL.
* ✅ **Proper use of Pydantic models** with Tortoise ORM.
* ✅ **JWT token security** via OAuth2PasswordBearer.

**⚠️ Issues and Suggested Improvements**

**🔒 Security Issues**

1. **Hardcoded URLs (e.g., localhost)**:
   * Replace "localhost:8000" with dynamic Request.base\_url or environment variables.
   * ✅ Fix:

python

CopyEdit

file\_url = str(request.base\_url) + generated\_name[9:]

1. **Allowed image extensions** are only "png" and "jpg":
   * You may want to add "jpeg" and check MIME types, not just extensions.
   * ✅ Use:

python

CopyEdit

import imghdr if imghdr.what(generated\_name) not in ["jpeg", "png"]: return {"status": "error", "detail": "Invalid image file"}

1. **No rate limiting / brute-force protection**:
   * Add middleware or third-party tools like [SlowAPI](https://github.com/laurentS/slowapi) to protect /token.
2. **Email verification token** should expire or be signed using a safer method (itsdangerous or JWT with short expiry).

**🧹 Code Quality and Readability**

1. **Redundant calls to**.get()**and**from\_queryset\_single():

python

CopyEdit

product = await Product.get(id=id) ... response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) # redundant

✅ Fix:

python

CopyEdit

response = await product\_pydantic.from\_tortoise\_orm(product)

1. **DRY image upload logic**:
   * Create a helper function for repeated logic in profile and product upload routes.

✅ Suggested helper:

python

CopyEdit

async def save\_image(file: UploadFile, path: str = "./static/images/"): extension = file.filename.split(".")[-1] if extension not in ["png", "jpg", "jpeg"]: raise HTTPException(status\_code=400, detail="Invalid file type") token\_name = secrets.token\_hex(10) + "." + extension full\_path = os.path.join(path, token\_name) content = await file.read() with open(full\_path, "wb") as f: f.write(content) img = Image.open(full\_path) img = img.resize((200, 200)) img.save(full\_path) return token\_name

1. await business.save()**is missing in some routes** (e.g., update\_business).
2. **Return structured errors**:
   * For example, in /uploadfile/product/{id}:

python

CopyEdit

return {"status": "error", "detail": "Unauthorized"}

is better handled with HTTPException.

**🐞 Bugs**

* In /delete/product/{id}:

python

CopyEdit

if user == owner: await product.delete()

But there's a **tab indentation bug** — return {"status": "ok"} is outside the if.

**🚀 Additional Improvements**

1. **Environment Variables**:
   * Move secrets like "SECRET" and file paths into .env using python-dotenv.
2. **Logging**:
   * Replace comments like #LOG with Python's logging module.
3. **Unit Tests**:
   * Consider adding a tests/ folder with test cases using pytest and httpx.AsyncClient.
4. **Swagger Docs**:
   * Your current setup integrates well with FastAPI's Swagger UI.
   * You might want to **add tags and summaries** for cleaner documentation:

python

CopyEdit

@app.post("/products", tags=["Products"], summary="Add a new product")

**✅ Overall Assessment**

You're building a solid **e-commerce or business catalog platform** backend. The foundations are great. With a few **security**, **code organization**, and **maintainability enhancements**, this could be production-ready.

Would you like help refactoring your code with helper functions and .env integration next?

**You said:**

where should i apply logs here

**ChatGPT said:**

Great question — adding **logging** to your FastAPI app improves observability, debugging, and long-term maintainability. You currently have #LOG comments, which are good placeholders. Let's turn those into proper log statements.

**✅ First: Set up Python's Logging**

At the top of your main.py, add:

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, # or DEBUG in dev format="%(asctime)s - %(levelname)s - %(message)s", handlers=[ logging.StreamHandler() ] ) logger = logging.getLogger(\_\_name\_\_)

**🧠 Where to Add Logging?**

Here’s a detailed breakdown of **where and what** to log, based on your current code:

**🧾 1. Authentication Logs**

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): logger.info(f"Login attempt for user: {request\_form.username}") token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"}

python

CopyEdit

async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) logger.debug(f"Token payload decoded for user ID: {payload.get('id')}") ... except Exception as e: logger.warning(f"Invalid token access attempt: {e}") raise HTTPException(...)

**🧾 2. User Registration**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}, email: {user.email}") ... logger.info(f"User created: {new\_user.username}") ...

**🧾 3. Post-save Signal**

python

CopyEdit

@post\_save(User) async def create\_business(...): if created: logger.info(f"Business profile auto-created for user: {instance.username}") ... logger.info(f"Verification email sent to: {instance.email}")

**🧾 4. Email Verification**

python

CopyEdit

@app.get("/verification") async def email\_verification(...): logger.info(f"Verification attempt with token: {token}") ... logger.info(f"User {user.username} verified successfully") ...

**🧾 5. File Uploads (Profile/Product Image)**

python

CopyEdit

@app.post("/uploadfile/profile") async def create\_upload\_file(...): logger.info(f"User {user.username} uploading profile image: {file.filename}") ... logger.info(f"Image saved as: {token\_name}")

Same pattern for /uploadfile/product/{id}.

**🧾 6. Product CRUD Operations**

python

CopyEdit

@app.post("/products") async def add\_new\_product(...): logger.info(f"New product being added by user: {user.username}") ...

python

CopyEdit

@app.delete("/product/{id}") async def delete\_product(...): logger.info(f"Delete attempt for product ID {id} by user {user.username}")

python

CopyEdit

@app.put("/product/{id}") async def update\_product(...): logger.info(f"Update attempt on product ID {id} by user {user.username}")

**🧾 7. Business Update**

python

CopyEdit

@app.post("/business/{id}") async def update\_business(...): logger.info(f"Business update request for business ID {id} by user {user.username}")

**🔐 Logging Best Practices**

* Avoid logging sensitive data (passwords, full tokens).
* Use logger.debug for verbose details, info for key actions, and warning or error for problems.

**🎯 Bonus: Separate Log File**

If you're planning to run this in production, log to a file:

python

CopyEdit

logging.basicConfig( filename="app.log", filemode="a", level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s" )

Would you like me to give you a **full version of**main.py**with logging added**, or just logging decorators/helpers to minimize repeated code?

**You said:**

import logging logging.basicConfig( level=logging.INFO, # or DEBUG in dev format="%(asctime)s - %(levelname)s - %(message)s", handlers=[ logging.StreamHandler() ] ) logger = logging.getLogger(\_\_name\_\_) explain this code

**ChatGPT said:**

Absolutely! Let's break down this logging setup line by line so you understand exactly what's happening:

python

CopyEdit

import logging

This imports Python’s built-in logging module, which provides a flexible framework for emitting log messages from Python programs.

**Configuration Block:**

python

CopyEdit

logging.basicConfig( level=logging.INFO, # or DEBUG in dev format="%(asctime)s - %(levelname)s - %(message)s", handlers=[ logging.StreamHandler() ] )

This configures the **root logger** with some specific behaviors.

**🔹**level=logging.INFO

* This sets the **minimum severity level** for logs that will be handled.
* Available levels (in order of severity): DEBUG, INFO, WARNING, ERROR, CRITICAL.
* INFO means:
  + Logs at INFO, WARNING, ERROR, and CRITICAL levels will be shown.
  + DEBUG messages will be ignored unless you set level=logging.DEBUG.

**🔹**format="%(asctime)s - %(levelname)s - %(message)s"

This is the **output format** of each log message.

* %(asctime)s: Timestamp (e.g., 2025-06-03 12:45:23,456)
* %(levelname)s: Logging level (e.g., INFO, ERROR)
* %(message)s: The log message itself

**Example Output:**

pgsql

CopyEdit

2025-06-03 12:45:23,456 - INFO - User registered successfully

**🔹**handlers=[logging.StreamHandler()]

* This tells Python **where to send the log output**.
* StreamHandler() means log messages go to the **console (stdout)**.
* You can also use FileHandler() to log to a file instead.

**Logger Object:**

python

CopyEdit

logger = logging.getLogger(\_\_name\_\_)

* getLogger(\_\_name\_\_) returns a **logger instance** specific to the current module.
* \_\_name\_\_ ensures the logger is **named after the module it's in** (main, authentication, etc.).
* You use this logger object to actually **write log messages**:

python

CopyEdit

logger.info("Something happened") logger.warning("Be careful!") logger.error("Something went wrong")

**✅ Summary**

| **Line** | **Purpose** |
| --- | --- |
| logging.basicConfig(...) | Sets up the default logging behavior |
| level=logging.INFO | Only logs INFO and above (ignores DEBUG) |
| format="..." | Formats each log line with time, level, and message |
| handlers=[StreamHandler()] | Outputs logs to the console |
| logger = getLogger(\_\_name\_\_) | Creates a logger for your module |

Would you like a quick example using logger in your registration endpoint?

**You said:**

write the loga in a different file

**ChatGPT said:**

To **write logs to a file instead of the console**, you just need to change the handlers section of your logging.basicConfig() setup.

Here’s how to modify your code so logs are saved in a file called app.log:

**✅ Updated Logging Configuration (to write to**app.log**)**

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, # Change to DEBUG for more verbosity format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Log file path/name filemode="a" # Append to the log file (use "w" to overwrite each time) ) logger = logging.getLogger(\_\_name\_\_)

**💡 What Each Part Does**

| **Parameter** | **Description** |
| --- | --- |
| filename="app.log" | Log messages are written to this file |
| filemode="a" | Append mode – new logs will be added at the end of the file |
| format=... | Log format with timestamp, level, and message |
| level=logging.INFO | Log only INFO and above severity messages |

**📝 Example Usage in Your Code**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}, email: {user.email}") if await User.filter(username=user.username).exists(): logger.warning(f"Username already exists: {user.username}") raise HTTPException(status\_code=400, detail="Username already exists") ... logger.info(f"User registered successfully: {user.username}") return {...}

**📂 Result**

When your app runs and someone hits /registration, your app.log will contain entries like:

yaml

CopyEdit

2025-06-03 13:17:12,345 - INFO - Registration attempt for username: johndoe, email: john@example.com

2025-06-03 13:17:12,346 - INFO - User registered successfully: johndoe

**✅ Optional: Log to Both File and Console**

If you want to log to both the **console** and **file**, do this:

python

CopyEdit

from logging import FileHandler, StreamHandler logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", handlers=[ FileHandler("app.log", mode="a"), StreamHandler() ] )

Let me know if you'd like to rotate logs automatically (e.g., daily or by size), or log per module!

**You said:**

rest are same as above?

**ChatGPT said:**

Yes — once you configure logging to write to a file, **everything else in your code remains exactly the same**.

**✅ To Clarify:**

* Only the **logging configuration block** changes — where and how logs are written.
* All your logger.info(), logger.error(), etc. statements **stay the same** and will now write to app.log instead of (or in addition to) the console.

**🧠 Example Recap:**

**✅ Logging Config (File Only):**

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Logs go to this file filemode="a" # Append mode ) logger = logging.getLogger(\_\_name\_\_)

**✅ Your Routes/Logic (No Change):**

python

CopyEdit

@app.post("/registration") async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}") ... logger.info(f"User registered: {user.username}") ...

These logs will now go into the app.log file instead of (or along with) your terminal.

**✅ Final Tip (Optional But Useful):**

To confirm logging is working:

1. Start your app.
2. Trigger a route like /registration.
3. Open app.log — you should see your log entries.

Would you like to add **log rotation** next (e.g., auto-archive after file size limit or daily)?

**You said:**

dont we need to use handlers

**ChatGPT said:**

Great question! Let's clarify when and why you'd use handlers explicitly.

**✅ When**handlers**Are Needed**

* **Yes, you need to use**handlers if you want **more control** — like:
  + Logging to **multiple destinations** (e.g., file + console)
  + Using **rotation** (log files that archive automatically)
  + Setting different levels per handler

**🤔 But in your current case...**

When you're **only logging to a file**, and you use this:

python

CopyEdit

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", filemode="a" )

You **don't need to define a**FileHandler**manually** — basicConfig() automatically creates one for you.

So this is ✅ **correct and sufficient for basic logging to a file**.

**📌 When You DO Need to Use**handlers**Explicitly**

You **must use**handlers**manually** if you want something like:

**✅ Logging to both console and file:**

python

CopyEdit

from logging import FileHandler, StreamHandler logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", handlers=[ FileHandler("app.log", mode="a"), StreamHandler() ] )

**✅ Logging with rotation:**

python

CopyEdit

from logging.handlers import RotatingFileHandler rotating\_handler = RotatingFileHandler( "app.log", maxBytes=1000000, backupCount=5 ) logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", handlers=[rotating\_handler] )

This keeps app.log under ~1MB and rotates it (e.g., app.log.1, app.log.2, etc.)

**✅ Summary**

| **Scenario** | **Use**handlers**?** |
| --- | --- |
| Log to a file only (basic) | ❌ No need |
| Log to file **and** console | ✅ Yes |
| Log rotation (size-based or daily) | ✅ Yes |
| Log to different files per module | ✅ Yes |

Let me know if you’d like to add rotation or have different log files for different modules (like auth.log, product.log, etc.)

**You said:**

do we need handler if we have multiple modules

**ChatGPT said:**

✅ **Short answer:**  
**No**, you **don’t *have* to** use handlers just because you have multiple modules — but you **should** use them **if you want per-module log control or outputs**.

**🧠 Explanation: Logging with Multiple Modules**

Python’s logging system is **hierarchical**, and works across modules out-of-the-box:

**✅ Basic Setup (Works Across Modules)**

In your **main file (e.g.,**main.py**)**, you can configure logging:

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", filename="app.log", filemode="a" )

Then, in any module (e.g., authentication.py, product.py, etc.):

python

CopyEdit

import logging logger = logging.getLogger(\_\_name\_\_) logger.info("Authentication attempt") # Appears in app.log

No handlers needed — because they inherit from the root logger setup in main.py.

**📌 When You SHOULD Use Handlers in Multi-Module Projects**

Use **custom handlers** if you want to:

**✅ 1. Separate Log Files Per Module**

python

CopyEdit

# In authentication.py auth\_handler = logging.FileHandler("auth.log") auth\_logger = logging.getLogger("auth") auth\_logger.addHandler(auth\_handler) auth\_logger.setLevel(logging.INFO) auth\_logger.info("User logged in")

This way, logs related to authentication go to auth.log, not app.log.

**✅ 2. Send Critical Errors to Email or External Services**

python

CopyEdit

from logging.handlers import SMTPHandler mail\_handler = SMTPHandler( mailhost=("smtp.example.com", 587), fromaddr="errors@example.com", toaddrs=["admin@example.com"], subject="Critical Error", credentials=("username", "password"), secure=() ) mail\_handler.setLevel(logging.CRITICAL) logging.getLogger().addHandler(mail\_handler)

**✅ 3. Log to Console + File in All Modules**

python

CopyEdit

from logging import StreamHandler, FileHandler file\_handler = FileHandler("app.log") console\_handler = StreamHandler() logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", handlers=[file\_handler, console\_handler] )

Now all modules log to both the file and terminal.

**✅ Summary**

| **You Want To...** | **Use Handlers?** | **Notes** |
| --- | --- | --- |
| Log from multiple modules to one file | ❌ No | Basic setup is enough |
| Use different log files per module | ✅ Yes | Add module-specific handlers |
| Log to both console and file | ✅ Yes | Use StreamHandler + FileHandler |
| Send critical logs via email | ✅ Yes | Use SMTPHandler |
| Customize log levels per module | ✅ Yes | Use getLogger(module\_name) |

Would you like a working example that sets up logging in main.py and captures logs from other modules like authentication.py and models.py?

**You said:**

here wherever i am logging status return already exist so what is the benefit of logging

**ChatGPT said:**

That’s a great and very common question: **“If I’m already returning errors to the client (e.g., 'Username already exists'), what’s the point of also logging it?”**

**✅ Quick Answer:**

**Logging is for you (the developer/admin).  
Returning a response is for the user/client.**

They serve **very different purposes**.

**📊 Think of it like this:**

| **Purpose** | **Client Return (**raise HTTPException**)** | **Logging (**logger.info**, etc.)** |
| --- | --- | --- |
| For the user/client | ✅ Yes | ❌ No |
| For debugging | ❌ No | ✅ Yes |
| For audit trails | ❌ No | ✅ Yes |
| For monitoring | ❌ No | ✅ Yes |
| For alerting admins | ❌ No | ✅ Yes |

**💡 Why Logging Is Still Useful**

**✅ 1. Audit and Visibility**

python

CopyEdit

logger.warning(f"Registration failed — Username already exists: {user.username}")

* You can later check how many users tried registering with an already taken username.
* You can't do that from HTTP responses — they're not stored unless you log.

**✅ 2. Debugging**

python

CopyEdit

logger.info(f"User login attempt: {username}")

* If there's a bug or strange behavior, logs help you trace what actually happened.
* Logs may show patterns of abuse (e.g., brute-force login attempts).

**✅ 3. Incident Investigation**

* Imagine a user says, *“I signed up but never got an email.”*
* You can check logs to see:
  + Was the registration endpoint hit?
  + Was the send\_email function triggered?
  + Was there an exception?

**✅ 4. Silent Failures**

Sometimes your raise HTTPException(...) might get masked by reverse proxies, frontend issues, etc.

Logs give you ground-truth info no matter what the client sees.

**🔄 Example**

**Your current code:**

python

CopyEdit

if await User.filter(username=user.username).exists(): raise HTTPException(status\_code=400, detail="Username already exists")

**With logging added:**

python

CopyEdit

if await User.filter(username=user.username).exists(): logger.warning(f"Registration failed — Username already exists: {user.username}") raise HTTPException(status\_code=400, detail="Username already exists")

**Benefit:**

Now you can see that someone (possibly a bot) tried registering a dozen times with the same username — and take action.

**🧠 Final Thought**

You **return** status codes and messages to help the **client/user**.

You **log** status, context, and errors to help the **developer/admin**.

✅ Do both — and you’ll thank yourself when something goes wrong.

Would you like a template for writing clean log messages (with trace IDs, timestamps, etc.)?

**You said:**

async def email\_verification(request: Request, token: str): what is request in this line

**ChatGPT said:**

Great question.

In this line:

python

CopyEdit

async def email\_verification(request: Request, token: str):

the request: Request parameter is an instance of **FastAPI’s**Request**class**, and it represents the **incoming HTTP request** made by the client (browser, frontend, Postman, etc.).

**🔍 Why You Need**request: Request

Even if you're not reading the body or headers, you **need it** in certain cases — especially when using **templating engines like Jinja2**.

For example:

python

CopyEdit

return templates.TemplateResponse("verification.html", { "request": request, "username": user.username })

Jinja2 **requires** the request object in the context to render templates properly, especially when using things like:

* {{ request.url }} in the HTML
* Conditional checks like if request.user.is\_authenticated (in some setups)

**🧠 So, what is**request**?**

It is an object of type starlette.requests.Request.

You can access things like:

python

CopyEdit

request.url # Full URL request.method # GET, POST, etc. request.headers # Dict of HTTP headers request.client # IP address info request.cookies # Access cookies

**✅ When to use**request

* When rendering templates with Jinja2 or other templating systems
* When you need to inspect request metadata (headers, IPs, query params)
* When building more complex middleware-like behavior (e.g., auditing or analytics)

**✅ Example**

python

CopyEdit

@app.get("/verification", response\_class=HTMLResponse) async def email\_verification(request: Request, token: str): user = await very\_token(token) if user: if user.is\_verified: return templates.TemplateResponse("already\_verified.html", { "request": request, "username": user.username })

Without passing request to TemplateResponse, you’d get an error like:

vbnet

CopyEdit

TypeError: request must be provided to render a template

Would you like a quick example showing how to access headers or client IP from request?

**You said:**

return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) what does request do here

**ChatGPT said:**

Great follow-up!

In this line:

python

CopyEdit

return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username})

the request is passed to the **Jinja2 template context**. This is **required** for FastAPI + Jinja2 templates to render properly.

**🧠 Why Is**request**Needed in Jinja2?**

Jinja2 templates used in FastAPI (via Jinja2Templates) expect a request object so they can:

* Access things like the full URL ({{ request.url }})
* Determine the HTTP method, headers, cookies, etc.
* Support template helpers or filters that depend on request context

**🔍 Here's What Happens:**

**✅ FastAPI expects this:**

python

CopyEdit

return templates.TemplateResponse("template.html", {"request": request})

If you **omit** the request, you'll get:

vbnet

CopyEdit

TypeError: request must be provided to render a template

**🧪 In Your HTML Template**

You can now do this inside already\_verified.html:

html

CopyEdit

<p>You are already verified, {{ username }}.</p> <p>Your verification request came from: {{ request.client.host }}</p> <p>Request URL: {{ request.url }}</p>

**✅ Summary**

| **Element** | **Purpose** |
| --- | --- |
| "request": request | Gives the Jinja2 template access to the HTTP request |
| TemplateResponse | Renders the template using that context |
| username | Just extra data you're passing to show in HTML |

If you're using templates in FastAPI, **always include**request in the context dictionary.

Would you like an example of a custom Jinja2 template that uses request info like IP or headers?

**You said:**

business = await Business.get(owner = user) #shouldnt it be before saving image? owner = await business.owner if owner == user: business.logo = token\_name await business.save() here we get owner using user so isnt it redundant

**ChatGPT said:**

Yes — you're absolutely right to question that.

Let's take a closer look at this code:

python

CopyEdit

business = await Business.get(owner=user) # ① Get the business linked to current user owner = await business.owner # ② Get the owner of the business (again) if owner == user: # ③ Compare again business.logo = token\_name await business.save()

**✅ You're Right: This Is Redundant**

Since you already filtered the business by owner=user in step ①, you **already know** that user is the owner.

So this line:

python

CopyEdit

owner = await business.owner

...and the conditional check:

python

CopyEdit

if owner == user:

...are **redundant**. You've already confirmed that user is the owner when you did:

python

CopyEdit

Business.get(owner=user)

**✅ Cleaned-Up Version**

You can simplify it to:

python

CopyEdit

business = await Business.get(owner=user) # You already verified ownership business.logo = token\_name await business.save()

Much cleaner and logically sound ✅

**🧠 Why Might It Have Been Written That Way?**

Sometimes developers write checks like this out of caution, especially if:

* They're reusing code across routes with different access conditions
* The .owner relationship might change dynamically
* They copied from another block where ownership wasn't already guaranteed

But in **this case**, since you're querying by owner=user, it's safe to drop the extra check.

Let me know if you want me to review other redundant or inefficient spots in your code — happy to help clean it up.

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

04-06-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up

**You said:**

from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: # This goes LAST raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user async def verify\_password(plain\_password, hashed\_password): #LOG verifying password return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): #LOG verifying username user = await User.get(username = username) if user and await verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token this is my main module is the logging ok here

**ChatGPT said:**

You're on the right track with your authentication logic in FastAPI using Tortoise ORM, JWT, and passlib. However, **logging** is currently only mentioned in comments like #LOG verifying password — but no actual logging is implemented. For real-world applications, especially in authentication and error handling, **proper logging is crucial**.

**✅ What’s good**

* Modular and clean structure.
* Good use of exception handling.
* Sensible HTTP status codes for different error types.
* Secure password hashing using bcrypt via passlib.
* Token generation with JWT.

**❗ What's Missing: Actual Logging**

To add logging, you should use Python’s built-in logging module. Here’s how you can enhance your code:

**✅ Step 1: Set up logging**

At the top of your module:

python

CopyEdit

import logging logging.basicConfig(level=logging.INFO) logger = logging.getLogger(\_\_name\_\_)

**✅ Step 2: Add log statements**

Now, replace or supplement your #LOG comments with actual log calls. Example with improvements:

python

CopyEdit

async def verify\_password(plain\_password, hashed\_password): logger.info("Verifying password") return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): logger.info(f"Authenticating user: {username}") try: user = await User.get(username=username) if user and await verify\_password(password, user.password): logger.info(f"Authentication successful for user: {username}") return user except DoesNotExist: logger.warning(f"Authentication failed: User '{username}' not found") except Exception as e: logger.error(f"Unexpected error during authentication: {e}") logger.warning(f"Authentication failed for user: {username}") return False async def very\_token(token: str): try: logger.info("Decoding token") payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) logger.info(f"Token is valid for user ID: {user.id}") except DoesNotExist: logger.warning("Token user does not exist") raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception as e: logger.error(f"Token verification failed: {e}") raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user

**🧠 Optional: Environment-Based Logging Levels**

To reduce logging verbosity in production, you can control levels using environment variables:

python

CopyEdit

import os log\_level = os.getenv("LOG\_LEVEL", "INFO").upper() logging.basicConfig(level=log\_level)

**✅ Summary**

**Logging is not currently implemented — only hinted at in comments.** You should use the logging module for actual log output. This will help with:

* Debugging
* Monitoring auth activity
* Auditing security events

Would you like me to help integrate this logging setup into your full project structure?

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #logging import logging logging.basicConfig( level=logging.INFO, # Change to DEBUG for more verbosity format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Log file path/name filemode="a" # Append to the log file (use "w" to overwrite each time) ) logger = logging.getLogger(\_\_name\_\_) #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config #becomes available by http://localhost:8000/static/images/logo.png #improve for security purposes app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): #LOG generating token logger.info(f"Login attempt for user: {request\_form.username}") token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) logger.debug(f"Token payload decoded for user ID: {payload.get('id')}") user = await User.get(id = payload.get("id")) except: logger.warning(f"Invalid token access attempt: {e}") raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: #LOG creatin and saving a business profile of user in database bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # LOG the sending of email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}, email: {user.email}") user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) logger.info(f"User created: {new\_user.username}") except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): #what is request logger.info(f"Verification attempt with token") user = await very\_token(token) if user: if user.is\_verified: # LOG already verified, return a message indicating they are already verified logger.info(f"User {user.username} already verified ") return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # LOG user was not verified, verify the user user.is\_verified = True await user.save() logger.info(f"User {user.username} verified successfully") return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) #LOG error raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} #LOG upload pictures @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"User {user.username} uploading profile image: {file.filename}") FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() #LOG image is saving with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() logger.info(f"Image saved") business = await Business.get(owner = user) #shouldnt it be before saving image? owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"New product being added by user: {user.username}") product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Delete attempt for product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Update attempt on product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): logger.info(f"Business update request for business ID {id} by user {user.username}") update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) sorry this is my main module

**ChatGPT said:**

Thanks for sharing the full main module — it's a well-structured FastAPI app with authentication, image uploads, and product management features. Now that logging is explicitly implemented, here’s a **review focused on your logging usage and a few other best practices**.

**✅ What You're Doing Well**

1. **Logging Setup**:
   * logging.basicConfig(...) is correctly used.
   * Logs are saved to app.log.
   * You’re using logger.info(), logger.warning(), and logger.debug() appropriately.
2. **Authentication and Verification**:
   * JWT and password hashing are properly implemented.
   * Secure checks for image file types and user ownership.
3. **App Modularity and Signal Usage**:
   * Clean separation of concerns.
   * Use of Tortoise ORM signals like @post\_save for automatic business creation.
4. **Static file serving**:
   * Good use of app.mount("/static", ...).

**⚠️ Logging: Improvements & Fixes**

**1. ❌ Missing**Exception**object in some logs**

In get\_current\_user, you're using except: but trying to log e, which will crash:

python

CopyEdit

except: logger.warning(f"Invalid token access attempt: {e}") # e is undefined!

✅ Fix:

python

CopyEdit

except Exception as e: logger.warning(f"Invalid token access attempt: {e}")

**2. ❌ Logging**new\_user**before it’s defined**

In /registration route:

python

CopyEdit

user\_obj = await User.create(\*\*user\_info) logger.info(f"User created: {new\_user.username}") # new\_user not yet defined

✅ Fix:

python

CopyEdit

user\_obj = await User.create(\*\*user\_info) new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) logger.info(f"User created: {new\_user.username}")

**3. 🔁 Repetitive log messages like:**

python

CopyEdit

# LOG user was not verified, verify the user

You already have similar logger.info(...) messages. You can safely remove these comments once proper log calls are in place.

**4. ⚠️ Suggestion: Use**logger.error()**for real errors**

Where you’re logging exceptions or unexpected failures, like decoding tokens, use logger.error() instead of warning().

**🛠️ General Suggestions (Outside Logging)**

**✅ Use**HTTP\_403\_FORBIDDEN**for Unauthorized Actions**

For cases where a user is authenticated but **not authorized** (e.g. not the owner), return 403 instead of 401:

python

CopyEdit

raise HTTPException( status\_code=status.HTTP\_403\_FORBIDDEN, detail="Not authorized to perform this action", headers={"WWW-Authenticate": "Bearer"} )

**✅ Sanitize file extension logic**

Split filename safely:

python

CopyEdit

extension = filename.rsplit(".", 1)[-1].lower()

Also check file.content\_type for added security.

**✅ Close PIL image objects**

Ensure images are closed properly (even though with block handles file I/O):

python

CopyEdit

img = Image.open(generated\_name) img = img.resize((200, 200)) img.save(generated\_name) img.close() # optional but good habit

**✅ Add Logging on Upload Failures**

Currently, if a file upload fails (e.g., wrong extension), you return an error but don’t log it.

python

CopyEdit

if extension not in ["png", "jpg"]: logger.warning(f"Disallowed file type uploaded by {user.username}: {extension}") return {"status": "error", "detail": "File extension not allowed"}

**✅ Summary**

Your app is solid and the logging foundation is **almost there**. With a few quick fixes (especially catching exceptions correctly and logging them properly), your logging will be reliable and production-ready.

Would you like help adding custom log levels or sending logs to external services (e.g., Sentry, Logstash)?

**You said:**

from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from fastapi import status from mail import config\_credentials pwd\_context = CryptContext(schemes=["bcrypt"], deprecated = "auto") def get\_hashed\_password(password): return pwd\_context.hash(password) async def very\_token(token: str): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user = await User.get(id=payload.get("id")) except DoesNotExist: raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception: # This goes LAST raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user async def verify\_password(plain\_password, hashed\_password): #LOG verifying password return pwd\_context.verify(plain\_password, hashed\_password) async def authenticate\_user(username, password): #LOG verifying username user = await User.get(username = username) if user and await verify\_password(password, user.password): return user return False async def token\_generator(username: str, password: str): user = await authenticate\_user(username, password) if not user: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) return token this is my authentication module apply logging where necessary

**ChatGPT said:**

Great — your authentication module is solid but, like your main app, it needs **real logging** instead of placeholder comments. Below is your **updated authentication module** with logging properly added at key points:

**✅ Key Improvements Made:**

* Integrated logging
* Added informative logs for:
  + Password verification
  + Authentication attempts
  + Token generation and verification
  + Error cases with exception details

**🔧 Updated Module with Logging**

python

CopyEdit

from fastapi import HTTPException, status from tortoise.exceptions import DoesNotExist from passlib.context import CryptContext import jwt from dotenv import dotenv\_values from models import User from mail import config\_credentials import logging # Set up logger logger = logging.getLogger(\_\_name\_\_) # Password hashing context pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto") def get\_hashed\_password(password): logger.debug("Hashing password") return pwd\_context.hash(password) async def very\_token(token: str): try: logger.info("Verifying token") payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) user\_id = payload.get("id") logger.debug(f"Token payload: {payload}") user = await User.get(id=user\_id) logger.info(f"Token valid for user ID {user\_id}") except DoesNotExist: logger.warning(f"User not found for token ID: {payload.get('id')}") raise HTTPException( status\_code=status.HTTP\_404\_NOT\_FOUND, detail="User no longer exists" ) except Exception as e: logger.error(f"Invalid or expired token: {e}") raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid or expired token", headers={"WWW-Authenticate": "Bearer"} ) return user async def verify\_password(plain\_password, hashed\_password): logger.info("Verifying user password") result = pwd\_context.verify(plain\_password, hashed\_password) if result: logger.debug("Password verification successful") else: logger.debug("Password verification failed") return result async def authenticate\_user(username, password): logger.info(f"Authenticating user: {username}") try: user = await User.get(username=username) except DoesNotExist: logger.warning(f"Authentication failed: User '{username}' not found") return False if user and await verify\_password(password, user.password): logger.info(f"Authentication successful for user: {username}") return user logger.warning(f"Authentication failed for user: {username}") return False async def token\_generator(username: str, password: str): logger.info(f"Generating token for user: {username}") user = await authenticate\_user(username, password) if not user: logger.warning(f"Token generation failed for user: {username}") raise HTTPException( status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid username or password", headers={"WWW-Authenticate": "Bearer"} ) token\_data = { "id": user.id, "username": user.username } token = jwt.encode(token\_data, config\_credentials["SECRET"]) logger.info(f"Token generated for user: {username}") return token

**🧠 Notes**

* Logging levels used:
  + debug – internal values or flow tracking.
  + info – high-level successful actions (e.g., login attempt).
  + warning – expected but undesired events (e.g., user not found).
  + error – unexpected failures (e.g., token decoding errors).
* This logging setup assumes you've already configured logging in your main app with logging.basicConfig(...).

Would you like me to help you add a centralized logging utility or configure logs to rotate or send to an external logging system (like Sentry, ELK, etc.)?

**You said:**

where would the logging output shown

**ChatGPT said:**

Great question. The **logging output** will appear based on how you configured logging.basicConfig() in your **main module**. Here's how it works with your current setup:

**✅ Your Current Logging Configuration:**

python

CopyEdit

import logging logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # <--- Output file filemode="a" # Append mode )

**🔍 Result:**

* **All logs at level**INFO**and above** (INFO, WARNING, ERROR, CRITICAL) will be written to:

lua

CopyEdit

app.log

* This file will be located in the **same directory where you run your FastAPI app** (typically the root of your project).

**🧪 Example Output in**app.log**:**

yaml

CopyEdit

2025-06-04 12:01:34,823 - INFO - Authenticating user: johndoe

2025-06-04 12:01:34,901 - INFO - Authentication successful for user: johndoe

2025-06-04 12:01:35,001 - INFO - Token generated for user: johndoe

2025-06-04 12:02:12,567 - WARNING - Authentication failed for user: hacker

2025-06-04 12:03:10,123 - ERROR - Invalid or expired token: Signature has expired

**🧠 Optional: Also See Logs in Console**

If you'd like to **see logs in the terminal** *and* save them to the file, you need to **add a**StreamHandler like this:

python

CopyEdit

# In your main.py or a separate logging config module console\_handler = logging.StreamHandler() console\_handler.setLevel(logging.INFO) console\_formatter = logging.Formatter("%(asctime)s - %(levelname)s - %(message)s") console\_handler.setFormatter(console\_formatter) file\_handler = logging.FileHandler("app.log", mode="a") file\_handler.setLevel(logging.INFO) file\_handler.setFormatter(console\_formatter) # Add both handlers logger = logging.getLogger() logger.setLevel(logging.INFO) logger.addHandler(console\_handler) logger.addHandler(file\_handler)

This way, your logs go **both to the console** and to app.log.

Let me know if you'd like a reusable logging setup across modules or want log rotation (e.g., daily new log files).

**2/2**

**You said:**

wont we have to add app.log for it to show there

**ChatGPT said:**

Great observation — here's the full clarification:

**❓Do you need to create**app.log**manually?**

**No**, you **don’t need to manually create**app.log. Python’s logging module will **automatically create the file** if it doesn’t exist when the first log is written.

**✅ What Must Be in Place**

For the logs to actually show up in app.log, the following **must be true**:

1. ✅ You've configured filename="app.log" in logging.basicConfig(...).
2. ✅ You have at least one log statement (like logger.info(...)) that runs.
3. ✅ Your app has permission to write in the directory where it's running (i.e., the current working directory is writeable).

**📝 Example**

If you run your FastAPI app and have:

python

CopyEdit

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", filemode="a" )

...and then in any part of the code:

python

CopyEdit

logger.info("Server started")

Then on first run, Python will automatically create a file named app.log in the current directory and write:

pgsql

CopyEdit

2025-06-04 14:17:02,123 - INFO - Server started

**🔁 If You’re Not Seeing**app.log**:**

* Make sure the logging setup is placed **before any log statements**.
* Ensure your app is **actually running** the code that includes log calls.
* Try setting filemode="w" to force creation or overwrite (for testing).
* Check for errors in the console — if Python can't write the file (e.g., permission error), it might silently fail or print a warning.

Would you like me to help you extract this into a dedicated logging config file or utility module for reuse?

**You said:**

no i am asking if the logging output of authentication will be shown together with main and if yes then how as we didnt specify anything

**ChatGPT said:**

Ah — thank you for the clarification! You're asking:

**"Will the logging from the**authentication**module be shown in the same**app.log**file as the main module?"**

**✅ Short Answer: Yes, it will.**

**🔍 Why?**

Python’s logging system is **global** by default — all modules share the same logging configuration **once it's set** (typically in your main module). So even if authentication.py doesn't set up logging itself (no basicConfig), its logs will still be sent to the same handlers (like app.log) **if you use the same logger or root logger**.

**🔧 How It Works (Under the Hood)**

In authentication.py you have:

python

CopyEdit

import logging logger = logging.getLogger(\_\_name\_\_)

This creates a **module-level logger**, e.g., authentication, which is a child of the root logger.

Since you set up the logging config in your main.py like this:

python

CopyEdit

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", filemode="a" )

All loggers — including the one from authentication.py — will inherit this configuration **unless they explicitly override it**.

**✅ So Yes:**

* logger.info(...) in authentication.py
* logger.warning(...) in product.py
* logger.error(...) in any other module

➡️ **All will write to**app.log automatically.

**🧪 To See It in Action:**

Add this in authentication.py:

python

CopyEdit

logger.info("This is from the authentication module")

Run your app, and then check app.log. You'll see:

vbnet

CopyEdit

2025-06-04 15:05:01,432 - INFO - This is from the authentication module

**🧠 Pro Tip: Want to see which module logged it?**

Update your format like this in main.py:

python

CopyEdit

format="%(asctime)s - %(name)s - %(levelname)s - %(message)s"

Then logs will show:

pgsql

CopyEdit

2025-06-04 15:10:30,123 - authentication - INFO - Token generated for user: johndoe

Let me know if you'd like a logging setup that supports multiple log files per module or color-coded console output.

**You said:**

how does it make contact or know there is logging configuration on other module

**ChatGPT said:**

Great follow-up — this gets into **how Python's logging system works across modules**.

**✅ Summary Answer:**

When you import and use logging.getLogger(\_\_name\_\_) in **any module**, Python’s logging system checks whether a configuration (like basicConfig()) has already been set — **even if that configuration was defined in a different file** (e.g., your main.py).

➡️ **It works because all loggers are part of a global hierarchy managed by Python's logging module.**

**🧠 How It Works in Detail:**

**1. Python's logging is centralized**

The logging module has a single, shared **global logging configuration** in memory — think of it as a global registry.

**2.**logging.basicConfig(...)**sets this global configuration**

You typically call it **once**, usually in main.py, when the app starts.

python

CopyEdit

# In main.py import logging logging.basicConfig( level=logging.INFO, filename="app.log", format="%(asctime)s - %(levelname)s - %(message)s" )

This configures the **root logger**.

**3. Other modules use**logging.getLogger(\_\_name\_\_)

python

CopyEdit

# In authentication.py import logging logger = logging.getLogger(\_\_name\_\_)

This returns a child logger under the root logger — e.g., "authentication".

But it **inherits the root logger’s handlers and settings**, unless it defines its own.

**🧪 Real-World Analogy:**

Think of basicConfig() like setting the default rules in a company:

* You set them **once** in the headquarters (main.py).
* All branch offices (authentication.py, products.py, etc.) use those same rules **unless they override them**.

**🔄 Flow in Your App:**

1. main.py sets up logging via basicConfig(...).
2. Later, authentication.py is imported.
3. Inside authentication.py, getLogger(\_\_name\_\_) just asks:  
   "What logger should I use for authentication?"
4. Python says: "Here's one — it inherits settings from the root, which is already configured to log to app.log."

**🧪 Want to Prove It?**

Try this:

1. In authentication.py, add:

python

CopyEdit

import logging logger = logging.getLogger(\_\_name\_\_) logger.info("This is from authentication")

1. **Do not call**basicConfig()**in**authentication.py**.**
2. Run the app from main.py.

✅ You'll see the log appear in app.log.

Would you like an advanced example showing per-module log files or JSON log formatting?

**You said:**

from fastapi import BackgroundTasks, UploadFile, File, Form, Depends, HTTPException, status from fastapi\_mail import FastMail, MessageSchema, ConnectionConfig from dotenv import dotenv\_values from pydantic import BaseModel, EmailStr from typing import List from models import User import jwt # Load environment variables config\_credentials = dotenv\_values(".env") # Email configuration conf = ConnectionConfig( MAIL\_USERNAME=config\_credentials["EMAIL"], MAIL\_PASSWORD=config\_credentials["PASSWORD"], MAIL\_FROM=config\_credentials["EMAIL"], MAIL\_PORT=587, MAIL\_SERVER="smtp.gmail.com", MAIL\_STARTTLS=True, # instead of MAIL\_TLS MAIL\_SSL\_TLS=False, # instead of MAIL\_SSL USE\_CREDENTIALS=True ) # Send verification email async def send\_email(email: List[str], instance: User): token\_data = { "id": instance.id, "username": instance.username } # Generate JWT token token = jwt.encode(token\_data, config\_credentials["SECRET"], algorithm="HS256") # Email HTML template template = f""" <!DOCTYPE html> <html> <head></head> <body> <div style="display: flex; align-items: center; justify-content: center; flex-direction: column;"> <h3>Account Verification</h3> <p>Thanks for choosing our services. Please click the button below to verify your account:</p> <a style="margin-top: 1rem; padding: 1rem; border-radius: 0.5rem; font-size: 1rem; text-decoration: none; background: #0275d8; color: white;" href="http://localhost:8000/verification/?token={token}"> Verify your email </a> <p>If you did not register for our services, please ignore this email.</p> </div> </body> </html> """ message = MessageSchema( subject="Verification Email", recipients=email, # must be a list of strings body=template, subtype="html" ) # LOG sending verification email fm = FastMail(conf) await fm.send\_message(message=message) this is my mail module apply necessary log here

**ChatGPT said:**

Thanks! You've got a clean and functional mail module — all that's missing is proper **logging** for traceability and debugging.

Here’s your updated mail.py module with **logging** added in the right places:

**✅ What’s Logged:**

* When the email is about to be sent
* The recipient(s) and user ID
* Token creation
* Successful email dispatch
* Error handling (in case sending fails)

**🔧 Updated**mail.py**with Logging:**

python

CopyEdit

from fastapi\_mail import FastMail, MessageSchema, ConnectionConfig from dotenv import dotenv\_values from pydantic import EmailStr from typing import List from models import User import jwt import logging # Initialize logger logger = logging.getLogger(\_\_name\_\_) # Load environment variables config\_credentials = dotenv\_values(".env") # Email configuration conf = ConnectionConfig( MAIL\_USERNAME=config\_credentials["EMAIL"], MAIL\_PASSWORD=config\_credentials["PASSWORD"], MAIL\_FROM=config\_credentials["EMAIL"], MAIL\_PORT=587, MAIL\_SERVER="smtp.gmail.com", MAIL\_STARTTLS=True, MAIL\_SSL\_TLS=False, USE\_CREDENTIALS=True ) # Send verification email async def send\_email(email: List[str], instance: User): try: logger.info(f"Preparing to send verification email to {email} for user ID {instance.id}") # Create token token\_data = { "id": instance.id, "username": instance.username } token = jwt.encode(token\_data, config\_credentials["SECRET"], algorithm="HS256") logger.debug(f"JWT token created for user {instance.username}: {token}") # Email HTML content template = f""" <!DOCTYPE html> <html> <head></head> <body> <div style="display: flex; align-items: center; justify-content: center; flex-direction: column;"> <h3>Account Verification</h3> <p>Thanks for choosing our services. Please click the button below to verify your account:</p> <a style="margin-top: 1rem; padding: 1rem; border-radius: 0.5rem; font-size: 1rem; text-decoration: none; background: #0275d8; color: white;" href="http://localhost:8000/verification/?token={token}"> Verify your email </a> <p>If you did not register for our services, please ignore this email.</p> </div> </body> </html> """ message = MessageSchema( subject="Verification Email", recipients=email, body=template, subtype="html" ) fm = FastMail(conf) await fm.send\_message(message=message) logger.info(f"Verification email sent successfully to {email}") except Exception as e: logger.error(f"Failed to send verification email to {email}: {e}") raise

**📌 Notes:**

* logger.info(...): Used for normal operations (preparing/sending email).
* logger.debug(...): Used for sensitive/internal data like the JWT token.
* logger.error(...): Captures unexpected issues like SMTP failures.

**📁 Reminder:**

Make sure the logger config (with basicConfig(...) pointing to app.log) is initialized before any send\_email(...) call — which you already do in main.py.

Would you like help creating a log\_utils.py to share log config globally across modules more cleanly?

**You said:**

in context to all logs in all the module explain which level is used for logging and why

**ChatGPT said:**

Great — let's break down **which log levels are used** across your modules and **why they’re appropriate** in each context.

**🔢 Logging Levels Recap (from lowest to highest severity):**

| **Level** | **Use Case** |
| --- | --- |
| DEBUG | Detailed diagnostic info for developers. |
| INFO | Normal operations and flow confirmations. |
| WARNING | Something unexpected, but not breaking. |
| ERROR | An operation failed, but the app keeps running. |
| CRITICAL | Serious failure; app may not recover. |

**🔍 Your Code's Logging Levels — Explained Module by Module**

**✅ 1.**main.py

This is where the app is started, and core routes run.

| **Log Example** | **Level** | **Why?** |
| --- | --- | --- |
| App starts, user registers, uploads a file, etc. | INFO | These are standard operations — success logs. |
| User attempts unauthorized access | WARNING | The app isn't broken, but it's a security concern. |
| User fails verification, bad token | WARNING or ERROR | Possibly malicious or expired — worth attention. |
| User not found or unauthorized deletion attempt | ERROR | Important to log — indicates failure in access control. |

✅ **Appropriate use:**  
Use of INFO, WARNING, and ERROR is well balanced. Consider adding DEBUG for things like request body, headers, or query parameters when debugging.

**✅ 2.**authentication.py

Handles login, token validation, password check, etc.

| **Log Action** | **Level** | **Why?** |
| --- | --- | --- |
| Verifying password or username | INFO or DEBUG | Normal checks; DEBUG if you want more verbosity. |
| Token validation success | DEBUG | This is internal and expected behavior. |
| Authentication failure | WARNING or ERROR | Might be user error or brute force. |
| Token expired or invalid | ERROR | This is a failed login, worthy of alerting. |

✅ **Suggestion:**  
Add a few DEBUG logs for JWT payload inspection or failed password match reasons (without exposing passwords).

**✅ 3.**mail.py

Handles sending verification email.

| **Log Action** | **Level** | **Why?** |
| --- | --- | --- |
| Email prepared and being sent | INFO | Normal operation — success indicator. |
| JWT token created for email link | DEBUG | Internal tracking only — useful in development. |
| Email send failure (SMTP or otherwise) | ERROR | Important — user didn’t get email. |

✅ **Good practice here:** Logging email attempts without exposing actual token or passwords.

**✅ 4. CRUD & Upload Routes**

In your various endpoints (products, uploads, business edits):

| **Log Action** | **Level** | **Why?** |
| --- | --- | --- |
| Uploading image, creating product, updating business | INFO | Expected successful user actions. |
| File format wrong | WARNING | Minor error — user can fix it. |
| Unauthorized attempt to upload or modify | ERROR | Serious failure, log for auditing. |

**🧠 Summary Table**

| **Module** | **Used Levels** | **Improvements** |
| --- | --- | --- |
| main.py | INFO, WARNING, ERROR | Add DEBUG for trace logs |
| authentication.py | INFO, ERROR | Add WARNING and DEBUG where appropriate |
| mail.py | INFO, DEBUG, ERROR | Well balanced |
| Upload/CRUD | INFO, WARNING, ERROR | Add DEBUG for request details (optional) |

**🛠 Best Practices Going Forward**

1. **Use**DEBUG: In development mode, add logger.debug() to trace variables, payloads, and flow.
2. **Reserve**ERROR: For failed operations that could indicate bugs or require admin attention.
3. **Use**WARNING: For suspicious but non-breaking behavior.
4. **Log context**: Include user IDs, resource IDs, or IP addresses for traceability (but never passwords or tokens).

Would you like me to help you set different log levels based on the environment (e.g., DEBUG in dev, WARNING in prod)?

Get smarter responses, upload files and images, and more.

Log in

Sign up for free

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

06-06-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up for free

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #logging import logging logging.basicConfig( level=logging.INFO, # Change to DEBUG for more verbosity format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Log file path/name filemode="a" # Append to the log file (use "w" to overwrite each time) ) logger = logging.getLogger(\_\_name\_\_) #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config #becomes available by http://localhost:8000/static/images/logo.png #improve for security purposes app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): #LOG generating token logger.info(f"Login attempt for user: {request\_form.username}") token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) logger.debug(f"Token payload decoded for user ID: {payload.get('id')}") user = await User.get(id = payload.get("id")) except Exception as e: logger.warning(f"Invalid token access attempt: {e}") raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: #LOG creatin and saving a business profile of user in database bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # LOG the sending of email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}, email: {user.email}") user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) logger.info(f"User created: {new\_user.username}") except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): #what is request logger.info(f"Verification attempt with token") user = await very\_token(token) if user: if user.is\_verified: # LOG already verified, return a message indicating they are already verified logger.info(f"User {user.username} already verified ") return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # LOG user was not verified, verify the user user.is\_verified = True await user.save() logger.info(f"User {user.username} verified successfully") return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) #LOG error raise HTTPException( status\_code = status.HTTP\_403\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} #LOG upload pictures @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"User {user.username} uploading profile image: {file.filename}") FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() #LOG image is saving with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() logger.info(f"Image saved") business = await Business.get(owner = user) #shouldnt it be before saving image? owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business\_owner owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"New product being added by user: {user.username}") product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Delete attempt for product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Update attempt on product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): logger.info(f"Business update request for business ID {id} by user {user.username}") update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True )

**ChatGPT said:**

Your FastAPI application looks robust, with a clear structure for authentication, user registration, image uploads, and CRUD operations for products and businesses. Here's a review and some suggestions for improvements or corrections:

**✅ Positives**

* **Logging** is well-integrated across all routes.
* **JWT Authentication** is correctly applied using Depends(get\_current\_user).
* **Database Integration** with Tortoise ORM is properly configured.
* **Image Uploads** are handled with validation and resizing via Pillow.
* **Templates** are used for email verification responses, giving it a more polished user experience.

**⚠️ Recommendations / Potential Issues**

**1. Token URL Case Typo**

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token")

**Fix:** Rename oath2\_scheme to oauth2\_scheme to match convention and avoid confusion.

**2. Redundant Conversion to Pydantic Model**

In registration:

python

CopyEdit

new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj)

You're calling this twice—once after creating the user and again before returning the response. Only one is needed.

**3. File Closing Logic in Upload**

python

CopyEdit

file.close()

You are closing the uploaded file after already reading it into memory:

python

CopyEdit

file\_content = await file.read() # file.close() after open(generated\_name, ...) not affecting `UploadFile`

**Fix:** Remove file.close()—it's not doing anything meaningful for the UploadFile (already handled by FastAPI).

**4. Hardcoded URLs (e.g., logo and uploaded files)**

python

CopyEdit

file\_url = "localhost:8000"+generated\_name[1:]

**Fix suggestion:**

* Use Request object to get the base URL dynamically.
* Example:

python

CopyEdit

from fastapi import Request base\_url = str(request.base\_url) file\_url = base\_url + "static/images/" + token\_name

**5. Extension Validation Should Be Case-Insensitive**

python

CopyEdit

if extension not in ["png", "jpg"]:

**Fix:**

python

CopyEdit

if extension.lower() not in ["png", "jpg", "jpeg"]:

**6. Unawaited Coroutine**

python

CopyEdit

business.save() # Should be awaited

**Fix:**

python

CopyEdit

await business.save()

**7. Improper Dictionary Access**

python

CopyEdit

product = product.dict(exclude\_unset = True) if product["original\_price"] > 0:

If original\_price is not passed, it will raise a KeyError.

**Fix:**  
Use .get() with a default:

python

CopyEdit

if product.get("original\_price", 0) > 0:

**8. Verification Route – Clarify Purpose of**Request

python

CopyEdit

async def email\_verification(request: Request, token: str):

❓ *You asked: "what is request?"*

**Answer:**  
request is the FastAPI Request object needed for rendering templates using Jinja2Templates. It injects request context into the template so it can generate proper URLs or paths.

**9. Response After Product Deletion**

python

CopyEdit

await product.delete() # return {"status": "YAYA"} — Commented

**Fix:** Un-comment or refine to something like:

python

CopyEdit

return {"status": "ok", "message": "Product deleted successfully"}

**10. Error Handling on Object Fetches**

You use:

python

CopyEdit

product = await Product.get(id=id)

If id doesn't exist, this raises an exception.

**Fix:** Use get\_or\_none() and check for None:

python

CopyEdit

product = await Product.get\_or\_none(id=id) if not product: raise HTTPException(status\_code=404, detail="Product not found")

**✅ Suggested Minor Improvements**

* Use @router with APIRouter() for modular route organization.
* Consider separating your upload and auth logic into different files (e.g., routes/products.py, routes/users.py).
* Add rate limiting or CAPTCHA for registration/login to prevent abuse.
* Return more descriptive error messages for client-side UX improvements.

Let me know if you want help modularizing this app or setting up tests or Docker support.

**You said:**

here in post token we do not manually operate the function am i right

**ChatGPT said:**

You're absolutely right — in the /token endpoint, **you do not manually operate the function** in the traditional sense (like calling it with explicit arguments in code). Instead, **FastAPI uses dependency injection** to:

1. Automatically parse the form data.
2. Pass it to the endpoint function as parameters.

**Here's what's happening:**

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): ...

* Depends() tells FastAPI to **inject** the dependency.
* OAuth2PasswordRequestForm is a special class provided by FastAPI.
* FastAPI extracts username and password from a form using the application/x-www-form-urlencoded content type.
* Then it automatically **creates an instance of**OAuth2PasswordRequestForm and passes it into the function as request\_form.

**Summary:**

You don’t manually instantiate OAuth2PasswordRequestForm.  
FastAPI **does that for you** based on the form data sent in the request body.

**Example Client Request:**

When a client sends a POST like this:

http

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=admin&password=1234

FastAPI parses it and gives you request\_form.username == "admin" and request\_form.password == "1234" automatically.

Let me know if you’d like to see how to test this with curl or Postman.

**You said:**

can we do something to hide the endpoint in fastapi for better user interaction

**ChatGPT said:**

Yes, you **can hide or obscure an endpoint** in FastAPI in several ways, depending on what you mean by "hide" and your goals (e.g., improve security, reduce UI clutter, or restrict public access).

Here are practical options:

**✅ 1. Hide from OpenAPI docs (Swagger UI)**

If you want to prevent an endpoint from appearing in the interactive documentation:

python

CopyEdit

@app.post("/token", include\_in\_schema=False) async def generate\_token(...): ...

* This hides /token from Swagger and Redoc.
* The endpoint still works normally; it’s just not visible in auto-generated docs.

**✅ 2. Use a less obvious path**

To discourage casual discovery (security through obscurity — not real protection):

python

CopyEdit

@app.post("/api/v1/internal/auth/token") async def generate\_token(...): ...

This doesn't **secure** the endpoint, but makes it less obvious than /token.

**✅ 3. Require special headers or keys**

If you want only certain clients to access the endpoint:

python

CopyEdit

from fastapi import Header @app.post("/token") async def generate\_token( request\_form: OAuth2PasswordRequestForm = Depends(), x\_internal\_key: str = Header(...) ): if x\_internal\_key != "expected-secret-value": raise HTTPException(status\_code=403, detail="Forbidden") ...

You could also move this logic into a custom dependency.

**✅ 4. Restrict access based on IP, roles, or environment**

* Use a middleware or dependency to allow access only from internal IPs or specific users.
* You could limit access in development and disable it in production.

**✅ 5. Use API Gateway / Reverse Proxy**

In production, you can:

* Hide sensitive routes behind an API Gateway.
* Restrict access to endpoints using NGINX or a firewall.

**✳️ Bonus: Customizing Swagger UI**

If you're concerned with user interaction in the docs:

* You can **customize Swagger UI** to:
  + Change layout,
  + Remove Authorize button,
  + Add descriptions or tags to explain what's public vs internal.

**Summary**

| **Goal** | **Method** |
| --- | --- |
| Hide from docs | include\_in\_schema=False |
| Make path less obvious | Use less common route name |
| Add extra auth barrier | Require custom header or token |
| Restrict access in prod | API Gateway / Middleware / NGINX |
| Customize API docs | Use tags, disable Swagger for some |

Let me know which type of "hiding" you're aiming for — security, UX, or both — and I can help tailor the implementation.

**You said:**

app.mount("/static", StaticFiles(directory="static"), name="static") what is the significance of this line

**ChatGPT said:**

Great question!

This line in your FastAPI app:

python

CopyEdit

app.mount("/static", StaticFiles(directory="static"), name="static")

is used to **serve static files** (like images, CSS, JavaScript, etc.) from a specific directory in your project. Here's a breakdown of what it does:

**✅ What does**app.mount(...)**do?**

* It tells FastAPI:  
  "Make everything inside the static/ directory accessible via URLs that start with /static."

**✅ How it works:**

* Suppose you have this file structure:

cpp

CopyEdit

project/

├── main.py

├── static/

│ └── images/

│ └── logo.png

* With app.mount(...) as written, the file logo.png becomes available at:

bash

CopyEdit

http://localhost:8000/static/images/logo.png

**🔍 Parameters explained:**

| **Parameter** | **Meaning** |
| --- | --- |
| "/static" | The **URL prefix** used to access the files. |
| StaticFiles(directory="static") | Tells FastAPI to serve files from the static/ folder. |
| name="static" | Internal name for the route; useful for url\_for() in templates. |

**✅ Why is it useful?**

* You can use it to serve:
  + Uploaded images (as in your case).
  + Static assets for HTML templates.
  + Frontend builds (e.g., React/Vue apps).

**🔐 Security Tip:**

Make sure your static/ folder doesn't contain sensitive files (like .env, credentials, or unprocessed uploads), because anything in that directory becomes publicly accessible via URL.

**🛠️ Optional Improvement**

You can add a subdirectory, e.g., for images only:

python

CopyEdit

app.mount("/images", StaticFiles(directory="static/images"), name="images")

Then access like:

bash

CopyEdit

http://localhost:8000/images/logo.png

Let me know if you want to serve a full frontend (like React or HTML) with FastAPI — that uses similar concepts!

**You said:**

oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token")

**ChatGPT said:**

You're pointing out this line:

python

CopyEdit

oath2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

Let’s clarify what this does — and correct a small typo:

**🔧 Typo Alert**

You probably meant:

python

CopyEdit

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")

✅ Use oauth2\_scheme, not oath2\_scheme — unless the name is intentional (which could cause confusion).

**🔐 What is**OAuth2PasswordBearer**doing?**

This line sets up the **authentication dependency** for your API.

**✅ Purpose:**

It tells FastAPI:

“Expect the client to send a **Bearer token** in the Authorization header for protected routes. The token was obtained from the /token endpoint.”

**🧠 In plain terms:**

When a request comes in with this header:

http

CopyEdit

Authorization: Bearer <access\_token>

FastAPI will extract that token using oauth2\_scheme, and you can use it like this:

python

CopyEdit

async def get\_current\_user(token: str = Depends(oauth2\_scheme)): payload = jwt.decode(token, ...) ...

So token will be the raw string "abc123..." from the header.

**🔄**tokenUrl="token"**means...**

It tells OpenAPI docs (Swagger UI) **where clients can get the token**.

When you use the Swagger UI:

* There's an **Authorize** button.
* It shows a form based on this:

python

CopyEdit

tokenUrl="token"

* That form submits credentials (username & password) to your /token endpoint.

**🧪 Example Usage:**

python

CopyEdit

@app.get("/profile") async def get\_profile(user: User = Depends(get\_current\_user)): return user

This route will:

1. Use oauth2\_scheme to extract and validate the token.
2. Then call get\_current\_user to retrieve the user.

**✅ Summary**

| **Part** | **What it does** |
| --- | --- |
| OAuth2PasswordBearer | Tells FastAPI to look for a Bearer token in headers |
| tokenUrl="token" | Tells Swagger where the token can be obtained |
| Depends(oauth2\_scheme) | Injects the token string into your route logic |

Let me know if you'd like a visual or code flow for how the token is used and validated!

**You said:**

what is bearer token

**ChatGPT said:**

A **Bearer Token** is a type of **access token** used to authorize and authenticate users in APIs — particularly in OAuth2-based systems.

**🛡️ What is a Bearer Token?**

A **Bearer token** is a **string** that represents a user's authenticated session. When a client (browser, mobile app, etc.) sends this token along with API requests, the server uses it to verify the user's identity.

**✅ Example Format**

Bearer tokens usually look like long, encoded strings:

http

CopyEdit

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

The Bearer keyword means:  
“The client presenting this token is authorized to access this resource.”

**🔁 How it works in your FastAPI app:**

1. **User logs in** via /token endpoint with username & password.
2. You generate a token (usually a JWT — JSON Web Token).
3. The client stores the token (e.g., in localStorage, cookies, or memory).
4. For future API calls, the client sends:

http

CopyEdit

Authorization: Bearer <token>

1. FastAPI extracts the token via oauth2\_scheme, decodes it, and identifies the user.

**✅ Why use Bearer Tokens?**

| **Feature** | **Benefit** |
| --- | --- |
| Stateless | No server-side sessions needed |
| Secure (with HTTPS) | Only clients with a valid token can access protected resources |
| Compact | Tokens are short strings, easy to send in headers |
| Portable | Work across frontend-backend apps, mobile, etc. |

**🚫 Security Tips**

* Always use HTTPS — otherwise, tokens can be intercepted.
* Tokens should expire (short-lived).
* Don't store tokens in localStorage on frontend if you're worried about XSS.
* Use refresh tokens or re-authentication for long sessions.

**🔐 Example in FastAPI**

python

CopyEdit

from fastapi.security import OAuth2PasswordBearer from fastapi import Depends oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token") @app.get("/profile") async def get\_profile(token: str = Depends(oauth2\_scheme)): # token is the Bearer token from the Authorization header return {"token": token}

Let me know if you want to walk through how to **generate**, **decode**, or **validate** a bearer token (e.g. JWT) in FastAPI.

**You said:**

@app.post("/token", include\_in\_schema=False) async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): #LOG generating token logger.info(f"Login attempt for user: {request\_form.username}") token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} explain this to me and how does token generation works and where it is needed

**ChatGPT said:**

Great — let's break this line by line and understand both **what it does** and **how token generation works** in FastAPI with OAuth2.

**✅ Full Breakdown of This Code**

python

CopyEdit

@app.post("/token", include\_in\_schema=False) async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()):

**🔹**@app.post("/token")

* This creates a **POST endpoint** at /token.
* It's typically used for **user login** in OAuth2.

**🔹**include\_in\_schema=False

* This hides the endpoint from the automatic OpenAPI docs (Swagger UI).
* This is good for "internal" or sensitive endpoints like authentication.

**🔹**request\_form: OAuth2PasswordRequestForm = Depends()

* FastAPI automatically reads the login form (username & password) from the request body.
* OAuth2PasswordRequestForm expects form-encoded data:

makefile

CopyEdit

Content-Type: application/x-www-form-urlencoded

username=someuser&password=somepassword

**▶️ Then inside the function:**

python

CopyEdit

logger.info(f"Login attempt for user: {request\_form.username}")

* Logs the login attempt (for monitoring or debugging).

python

CopyEdit

token = await token\_generator(request\_form.username, request\_form.password)

* Calls your own custom async function (token\_generator) to:
  + **Authenticate** the user using username and password
  + **Generate a token** (usually a **JWT** — JSON Web Token)

python

CopyEdit

return {"access\_token": token, "token\_type": "bearer"}

* Returns a dictionary as a JSON response:

json

CopyEdit

{ "access\_token": "<your\_token\_here>", "token\_type": "bearer" }

* This token is then used by the client in future requests.

**🔐 What is**token\_generator()**doing?**

Although you didn't paste that code here, typically a token\_generator() function does the following:

**✅ Steps (standard for JWT):**

1. **Validate user credentials**
   * Look up the user in the DB.
   * Verify password (usually hashed using bcrypt or similar).
2. **Generate JWT token**

python

CopyEdit

from jose import jwt token = jwt.encode( {"id": user.id, "exp": datetime.utcnow() + timedelta(minutes=30)}, SECRET\_KEY, algorithm="HS256" )

1. **Return the token**
   * This is what you see as access\_token.

**🛠️ Where is this token used?**

**✅ In all protected routes**

You use it like this:

http

CopyEdit

Authorization: Bearer <access\_token>

**✅ FastAPI uses this in:**

python

CopyEdit

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token") async def get\_current\_user(token: str = Depends(oauth2\_scheme)): payload = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"]) user\_id = payload.get("id") user = await User.get(id=user\_id) return user

This makes sure:

* The token is valid
* The user exists
* You're allowed to access whatever you're requesting

**🔁 How the full flow works:**

1. **Client logs in** via POST /token with username & password.
2. **Server returns a JWT** (token).
3. **Client stores the token** (e.g. in localStorage, sessionStorage, or memory).
4. For every protected API call, client sends:

makefile

CopyEdit

Authorization: Bearer <token>

1. **Server extracts the token**, validates it, and gets the user info.

**🧠 Summary**

| **Concept** | **Purpose** |
| --- | --- |
| /token | Login endpoint for OAuth2 |
| OAuth2PasswordRequestForm | Automatically reads username/password |
| token\_generator() | Authenticates user and returns a JWT |
| access\_token | Sent in Authorization: Bearer <token> for future requests |
| get\_current\_user | Uses token to get the logged-in user |

Let me know if you want help writing a secure token\_generator() function or decoding a token step-by-step!

**You said:**

what does OAuth2PasswordRequestForm do here

**ChatGPT said:**

Great question — understanding OAuth2PasswordRequestForm is key to working with FastAPI's OAuth2 flow.

**🔑 What is**OAuth2PasswordRequestForm**?**

OAuth2PasswordRequestForm is a **FastAPI utility class** that:

* Parses incoming form data (from a login request),
* Validates it,
* And gives you easy access to:
  + username
  + password
  + (and a few optional OAuth2 fields)

It’s used in login endpoints like this:

python

CopyEdit

@app.post("/token") async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()):

**🧪 How does it work?**

It expects the **form data** to be submitted like this:

**📨 Example Request:**

http

CopyEdit

POST /token Content-Type: application/x-www-form-urlencoded username=admin&password=1234

FastAPI automatically parses this and gives you access via:

python

CopyEdit

request\_form.username # → "admin" request\_form.password # → "1234"

**⚙️ Under the Hood**

This is what it expects:

python

CopyEdit

class OAuth2PasswordRequestForm: def \_\_init\_\_( self, grant\_type: str = Form(None, regex="password"), username: str = Form(...), password: str = Form(...), scope: str = Form(""), client\_id: Optional[str] = Form(None), client\_secret: Optional[str] = Form(None), ): ...

But **you never need to manually deal with that** — Depends() and FastAPI do all the work.

**✅ Why Use It?**

* **Standardized** way to handle OAuth2 logins.
* Works automatically with FastAPI’s OAuth2PasswordBearer.
* Integrates seamlessly with **Swagger UI** login:
  + When you click "Authorize" in the docs, it uses this format under the hood.

**🧠 Summary**

| **Feature** | **Purpose** |
| --- | --- |
| OAuth2PasswordRequestForm | Reads form-encoded login data |
| username / password | Available as properties |
| Used with Depends() | Injected automatically by FastAPI |
| Required by /token route | To generate JWTs / access tokens |

Let me know if you want to see how to build your own custom login form or swap it with JSON input!

Top of Form

Attach

Search

Reason

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.

17-06-2025

[Skip to content](https://chatgpt.com/#main)

ChatGPT

Log in

Sign up for free

**You said:**

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY app/ /app/ COPY start.sh /app/start.sh RUN chmod +x /app/start.sh CMD ["sh", "/app/start.sh"] this is my docker file 🚀 FastAPI app starting... 🌐 Visit: http://127.0.0.1:8000⁠ INFO: Started server process [8] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000⁠ (Press CTRL+C to quit) this is the result that comes on docker desktop however when i click on this Visit: http://127.0.0.1:8000⁠ this is shown This site can’t be reached 127.0.0.1 refused to connect. Try: Checking the connection Checking the proxy and the firewall ERR\_CONNECTION\_REFUSED but when i open my application then it works perfectly what can i do to start my application automatically without opening it in my local device

**ChatGPT said:**

You're almost there with your Docker setup! The issue you're facing is related to **binding your FastAPI application to the correct network interface** and possibly **exposing the right port in Docker**.

**🔍 Problem Summary**

In your logs:

nginx

CopyEdit

Uvicorn running on http://0.0.0.0:8000

This means your app is correctly **listening on all interfaces**, which is good. But you're trying to access:

cpp

CopyEdit

http://127.0.0.1:8000

Which refers to **your local host**, *not* the container’s network from outside Docker.

**✅ Solution: Expose the Port Properly**

To make your FastAPI app accessible from your host machine when running inside Docker:

**1. Add**EXPOSE**to Dockerfile (optional but good practice)**

Dockerfile

CopyEdit

EXPOSE 8000

**2. Run your Docker container with port mapping**

When running your Docker container, use the -p flag to map container port 8000 to your local port 8000:

bash

CopyEdit

docker build -t my-fastapi-app . docker run -p 8000:8000 my-fastapi-app

This maps:

lua

CopyEdit

localhost:8000 (your host) --> container:8000

**3. Make sure your**start.sh**starts the server like this:**

If you're using uvicorn, your start.sh should include:

bash

CopyEdit

#!/bin/sh uvicorn main:app --host 0.0.0.0 --port 8000

Ensure main is the Python file (main.py) and app is the FastAPI instance (app = FastAPI()).

**🧪 Final Checklist**

* App starts with uvicorn main:app --host 0.0.0.0 --port 8000
* Docker container is run with -p 8000:8000
* You're accessing http://localhost:8000 (not 127.0.0.1 in container)

Let me know if you're using docker-compose, and I can give you a sample file too.

**You said:**

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://127.0.0.1:8000" echo "" # Start the app uvicorn main:app --host 0.0.0.0 --port 8000 this is my start.sh is it okay

**ChatGPT said:**

Your start.sh is **almost perfect** — just one small tweak is needed to avoid confusion when accessing the app:

**🚨 Problem**

You're running the app on 0.0.0.0 (which is correct), **but** you're displaying this message:

sh

CopyEdit

echo "🌐 Visit: http://127.0.0.1:8000"

This implies the app is accessible on the container's own loopback interface, which is **not** true from your host.

**✅ Solution**

Update your start.sh to show the correct **host-accessible** URL:

sh

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" # Start the app uvicorn main:app --host 0.0.0.0 --port 8000

This ensures:

* You're binding to **all interfaces** (0.0.0.0)
* The message tells users to visit localhost, which will work if Docker is run with -p 8000:8000

**🧪 Final Test**

Make sure you run the container like this:

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

Then go to:

arduino

CopyEdit

http://localhost:8000

It should open your FastAPI app without needing to "open" anything manually.

Let me know if you're deploying this to the cloud or using Docker Compose — I can help with those too.

**You said:**

ok it started working you have both my dockerfile and start file can you explain to each line by line

**ChatGPT said:**

Great to hear it's working! Let's go through your **Dockerfile** and **start.sh** line by line so you understand exactly what each part does.

**🐳 Dockerfile Explained**

Dockerfile

CopyEdit

FROM python:3.10-slim

* **Base Image**: Uses a minimal Python 3.10 image to reduce size.
* slim means it has fewer packages pre-installed, so it's lighter but faster to download and deploy.

Dockerfile

CopyEdit

WORKDIR /app

* **Working Directory**: Sets /app as the directory where all commands (COPY, RUN, etc.) will be run from.
* Also becomes the default path when the container runs.

Dockerfile

CopyEdit

COPY requirements.txt .

* **Copy Requirements**: Brings the requirements.txt from your local machine into the /app directory in the container.

Dockerfile

CopyEdit

RUN pip install --no-cache-dir -r requirements.txt

* **Install Dependencies**:
  + Installs Python packages listed in requirements.txt.
  + --no-cache-dir avoids storing pip’s cache, reducing image size.

Dockerfile

CopyEdit

COPY app/ /app/

* **Copy App Code**:
  + Recursively copies everything inside your local app/ folder into the container’s /app/ directory.
  + Assumes your FastAPI code (like main.py) is inside app/.

Dockerfile

CopyEdit

COPY start.sh /app/start.sh

* **Copy Startup Script**:
  + Adds your shell script (start.sh) to the container at /app/start.sh.

Dockerfile

CopyEdit

RUN chmod +x /app/start.sh

* **Make Script Executable**:
  + Gives execution permission to start.sh.

Dockerfile

CopyEdit

CMD ["sh", "/app/start.sh"]

* **Default Command**:
  + When the container runs, it executes your startup script using sh.

**📂 Directory Structure Assumed**

Your Dockerfile assumes this structure:

css

CopyEdit

project/

│

├── Dockerfile

├── requirements.txt

├── start.sh

└── app/

├── main.py ← FastAPI code

└── other\_files\_or\_dirs

**🖥️ start.sh Explained**

sh

CopyEdit

#!/bin/sh

* **Shebang**: Specifies the script should run with the sh shell (standard shell interpreter).

sh

CopyEdit

echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo ""

* **User Feedback**: Prints messages to the terminal when the app starts.

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000

* **Start FastAPI**:
  + main:app tells Uvicorn to find the app = FastAPI() object inside main.py.
  + --host 0.0.0.0 allows connections from any IP (so Docker port mapping works).
  + --port 8000 defines which port the app listens on **inside** the container.

**🧠 Bonus Tip: Making it more flexible**

You could optionally allow the host and port to be set via environment variables:

sh

CopyEdit

uvicorn main:app --host "${HOST:-0.0.0.0}" --port "${PORT:-8000}"

This way, you can customize them at runtime like:

bash

CopyEdit

docker run -e HOST=0.0.0.0 -e PORT=9000 -p 9000:9000 my-fastapi-app

Let me know if you'd like to improve this further with production-ready options like **gunicorn**, **Docker Compose**, or **multi-stage builds**.

**You said:**

COPY app/ /app/ what is /app/ here what is the role of start.sh

**ChatGPT said:**

Great follow-up questions! Let’s break them down clearly:

**📁**COPY app/ /app/

**What is**/app/**?**

* /app/ is the **destination folder** **inside the Docker container**.
* You're telling Docker to copy everything from the **local**app/**directory on your machine** into /app/ **inside the container**.

**What’s happening exactly?**

dockerfile

CopyEdit

COPY app/ /app/

* **Left side**: app/ is your local directory containing FastAPI code like main.py.
* **Right side**: /app/ is the **target location inside the container’s filesystem**.

**Example:**

If you have this locally:

css

CopyEdit

project/

├── Dockerfile

├── app/

│ ├── main.py

│ └── utils.py

After the COPY command, your container’s file system will look like:

bash

CopyEdit

/app/

├── main.py

└── utils.py

**🧾 What is the Role of**start.sh**?**

start.sh**is your startup script.**

It tells Docker **how to run your app** once the container is started.

**Inside**start.sh**:**

sh

CopyEdit

#!/bin/sh echo "" echo "🚀 FastAPI app starting..." echo "🌐 Visit: http://localhost:8000" echo "" uvicorn main:app --host 0.0.0.0 --port 8000

This script:

1. **Prints startup messages** (so you see something friendly in logs).
2. **Launches your FastAPI app using Uvicorn**, telling it to:
   * Look for app = FastAPI() inside main.py (main:app)
   * Bind to all interfaces with --host 0.0.0.0
   * Use port 8000

**Why use a shell script at all?**

* It gives you flexibility.
* Later, you can:
  + Add logging.
  + Run database migrations.
  + Add health checks.
  + Set environment variables.

**Without**start.sh**, you could run:**

Dockerfile

CopyEdit

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

But using a script is **cleaner**, more **readable**, and **scales better** as complexity grows.

Let me know if you'd like to turn this into a production setup or run it with Docker Compose!

**You said:**

how can i add logging

**ChatGPT said:**

Great question! Adding logging to your FastAPI app and Docker container helps you debug, monitor, and scale effectively. There are **two main types** of logging to consider:

**✅ 1. Application-Level Logging (in FastAPI)**

This logs events from **inside your FastAPI app**, like errors, incoming requests, etc.

**🔧 Step-by-Step:**

Edit your main.py (or wherever your FastAPI app is defined):

python

CopyEdit

import logging from fastapi import FastAPI # Configure logging logging.basicConfig( level=logging.INFO, # or DEBUG, WARNING, ERROR format="%(asctime)s - %(levelname)s - %(message)s", ) app = FastAPI() @app.get("/") def read\_root(): logging.info("Root endpoint was called.") return {"message": "Hello, world!"}

This will print log messages like:

pgsql

CopyEdit

2025-06-17 10:00:00 - INFO - Root endpoint was called.

**💡 Tip:**

Use different log levels depending on context:

* logging.debug() for dev info
* logging.info() for general app events
* logging.warning() for expected problems
* logging.error() for errors
* logging.critical() for serious failures

**✅ 2. Shell Script Logging (**start.sh**)**

This logs things related to **container or environment startup**, not your FastAPI app directly.

**🛠️ Add logging to**start.sh**:**

sh

CopyEdit

#!/bin/sh LOGFILE="/app/logs/startup.log" mkdir -p /app/logs echo "" | tee -a $LOGFILE echo "🚀 FastAPI app starting..." | tee -a $LOGFILE echo "🌐 Visit: http://localhost:8000" | tee -a $LOGFILE echo "" | tee -a $LOGFILE # Start FastAPI with logging uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

**🔍 This will:**

* Create a log file at /app/logs/startup.log
* Append logs from both echo messages and Uvicorn output

**✅ 3. Docker Logging**

Docker logs are automatically captured by running:

bash

CopyEdit

docker logs <container\_id\_or\_name>

So anything your app prints to stdout/stderr (like logging via logging.info() or print()) will be shown there.

**📁 Directory Structure (with logs):**

css

CopyEdit

project/

├── app/

│ └── main.py

├── logs/

│ └── (created in container)

├── Dockerfile

├── requirements.txt

└── start.sh

**✅ Summary**

| **What You Want to Log** | **How to Do It** |
| --- | --- |
| App events (requests, errors) | Use logging in main.py |
| Startup behavior / Uvicorn | Log inside start.sh with tee |
| All output from container | Use docker logs <container\_id> |

Let me know if you want logs to go to an external file, cloud provider, or logging service like **ELK**, **Loki**, or **Datadog**.

**You said:**

where can i see the logs after i run the application in docker

**ChatGPT said:**

Great question! Once your FastAPI app is running in Docker, you have **two main ways** to see the logs:

**✅ 1. Using**docker logs

This is the most common and straightforward method.

**🧾 Command:**

bash

CopyEdit

docker logs <container\_name\_or\_id>

**🔍 Example:**

If you ran your container like this:

bash

CopyEdit

docker run -p 8000:8000 --name my-fastapi-app my-fastapi-image

You can view the logs with:

bash

CopyEdit

docker logs my-fastapi-app

This will show all the output from:

* start.sh script (echo commands)
* uvicorn server logs
* Any Python print() or logging calls in your FastAPI code

**✅ 2. Using Docker Desktop (GUI)**

If you're using **Docker Desktop** (Windows/Mac):

1. Open Docker Desktop.
2. Go to **Containers**.
3. Click on your container (e.g., my-fastapi-app).
4. Go to the **Logs** tab.

You’ll see the full logs live, just like in the terminal.

**🧠 Optional: Logging to a File (Inside Container)**

If you followed the earlier start.sh logging suggestion with:

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a /app/logs/startup.log

Then inside the running container, the logs would also be saved to:

bash

CopyEdit

/app/logs/startup.log

**To view that file (inside the container):**

1. Enter the container:

bash

CopyEdit

docker exec -it my-fastapi-app sh

1. Then check the log:

bash

CopyEdit

cat /app/logs/startup.log

**📌 TL;DR**

| **Where to Look** | **How** |
| --- | --- |
| Docker CLI logs | docker logs <container> |
| Docker Desktop UI | "Containers" → Click → "Logs" tab |
| Log file inside container | docker exec -it <container> sh → cat /app/logs/startup.log |

Let me know if you want to send logs to a file **on your host machine** or to a cloud logging service (like AWS CloudWatch, ELK, or Grafana Loki).

**You said:**

from fastapi import FastAPI, HTTPException, Request, status, Depends from tortoise.contrib.fastapi import register\_tortoise from models import \* import os #logging import logging logging.basicConfig( level=logging.INFO, # Change to DEBUG for more verbosity format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # Log file path/name filemode="a" # Append to the log file (use "w" to overwrite each time) ) logger = logging.getLogger(\_\_name\_\_) #authentication from authentication import \* from fastapi.security import (OAuth2PasswordBearer, OAuth2PasswordRequestForm) #signals from tortoise.signals import post\_save from typing import List, Optional, Type from tortoise import BaseDBAsyncClient from tortoise.exceptions import IntegrityError from mail import send\_email #image uplaod from fastapi import File, UploadFile import secrets from fastapi.staticfiles import StaticFiles from PIL import Image #response classes from fastapi.responses import HTMLResponse #datetime from datetime import datetime app = FastAPI() #templates from fastapi.templating import Jinja2Templates oath2\_scheme = OAuth2PasswordBearer(tokenUrl = "token") # static file setup config #becomes available by http://localhost:8000/static/images/logo.png #improve for security purposes app.mount("/static", StaticFiles(directory="static"), name="static") @app.post("/token", include\_in\_schema=False) async def generate\_token(request\_form: OAuth2PasswordRequestForm = Depends()): #LOG generating token logger.info(f"Login attempt for user: {request\_form.username}") token = await token\_generator(request\_form.username, request\_form.password) return {"access\_token": token, "token\_type": "bearer"} async def get\_current\_user(token: str = Depends(oath2\_scheme)): try: payload = jwt.decode(token, config\_credentials["SECRET"], algorithms=["HS256"]) logger.debug(f"Token payload decoded for user ID: {payload.get('id')}") user = await User.get(id = payload.get("id")) except Exception as e: logger.warning(f"Invalid token access attempt: {e}") raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Invalid username or password", headers = {"WWW-AUTHENTICATE": "Bearer"} ) return user @app.post("/user/me") async def user\_login(user: user\_pydanticIn = Depends(get\_current\_user)): business = await Business.get(owner = user) logo = business.logo logo\_path = "localhost:8000/static/images/"+logo return { "status": "ok", "data": { "username": user.username, "email": user.email, "verified": user.is\_verified, "joined\_date": user.join\_date.strftime("%b %d %Y"), "logo": logo\_path } } @post\_save(User) async def create\_business( sender: "Type[User]", instance: User, created: bool, using\_db: "Optional[BaseDBAsyncClient]", update\_fields: List[str] ) -> None: if created: #LOG creatin and saving a business profile of user in database bussiness\_obj = await Business.create( business\_name = instance.username, owner = instance ) await business\_pydantic.from\_tortoise\_orm(bussiness\_obj) # LOG the sending of email await send\_email([instance.email], instance) @app.post("/registration") # LOG registration process started async def user\_registration(user: user\_pydanticIn): logger.info(f"Registration attempt for username: {user.username}, email: {user.email}") user\_info = user.dict(exclude\_unset=True) # LOG Optional pre-check(username) if await User.filter(username=user\_info["username"]).exists(): raise HTTPException(status\_code=400, detail="Username already exists") # LOG pre-check(email) if await User.filter(email=user\_info["email"]).exists(): raise HTTPException(status\_code=400, detail="Email already exists") # Hash the password user\_info["password"] = get\_hashed\_password(user\_info["password"]) try: # LOG Create user and save user\_obj = await User.create(\*\*user\_info) new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) logger.info(f"User created: {new\_user.username}") except IntegrityError as e: raise HTTPException(status\_code=400, detail="Username or Email already exists") # LOG oncverting from orm model to pydantic model new\_user = await user\_pydantic.from\_tortoise\_orm(user\_obj) return { "status": "ok", "data": f"Hello {new\_user.username}, thanks for choosing our services." } BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) templates = Jinja2Templates(directory=os.path.join(BASE\_DIR, "templates")) @app.get("/verification", response\_class=HTMLResponse) # LOG verification starting async def email\_verification(request: Request, token: str): #what is request logger.info(f"Verification attempt with token") user = await very\_token(token) if user: if user.is\_verified: # LOG already verified, return a message indicating they are already verified logger.info(f"User {user.username} already verified ") return templates.TemplateResponse("already\_verified.html", {"request": request, "username": user.username}) else: # LOG user was not verified, verify the user user.is\_verified = True await user.save() logger.info(f"User {user.username} verified successfully") return templates.TemplateResponse("verification.html", {"request": request, "username": user.username}) #LOG error raise HTTPException( status\_code = status.HTTP\_403\_UNAUTHORIZED, detail = "Invalid Token or expired token", headers = {"WWW-Authenticate": "Bearer"} ) @app.get("/") # LOG app started def index(): return {"Message": "Hello World"} #LOG upload pictures @app.post("/uploadfile/profile") async def create\_upload\_file(file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"User {user.username} uploading profile image: {file.filename}") FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() #LOG image is saving with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() logger.info(f"Image saved") business = await Business.get(owner = user) #shouldnt it be before saving image? owner = await business.owner if owner == user: business.logo = token\_name await business.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) file\_url = "localhost:8000"+generated\_name[1:] return {"status": "ok", "filename": file\_url} @app.post("/uploadfile/product/{id}") async def create\_upload\_file(id: int, file: UploadFile = File(...), user: user\_pydantic = Depends(get\_current\_user)): FILEPATH = "./static/images/" filename = file.filename extension = filename.split(".")[1] if extension not in ["png", "jpg"]: return {"status": "error", "detail": "File extension not allowed"} token\_name = secrets.token\_hex(10)+"."+extension generated\_name = FILEPATH + token\_name file\_content = await file.read() with open(generated\_name, "wb") as file: file.write(file\_content) # PILLOW img = Image.open(generated\_name) img = img.resize(size = (200, 200)) img.save(generated\_name) file.close() product = await Product.get(id = id) business = await product.business\_owner owner = await business.owner if owner == user: product.product\_image = token\_name await product.save() else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) # CRUD functionality @app.post("/products") async def add\_new\_product(product: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"New product being added by user: {user.username}") product = product.dict(exclude\_unset = True) if product["original\_price"] > 0: product["percentage\_discount"] = ((product["original\_price"] - product["new\_price"]) / product["original\_price"]) \* 100 business = await Business.get(owner=user) product\_obj = await Product.create(\*\*product, business\_owner=business) product\_obj = await product\_pydantic.from\_tortoise\_orm(product\_obj) return {"status": "ok", "data": product\_obj} else: return {"status": "error"} @app.get("/product") async def get\_product(): response = await product\_pydantic.from\_queryset(Product.all()) return {"status": "ok", "data": response} @app.get("/product/{id}") async def get\_product(id: int): product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner response = await product\_pydantic.from\_queryset\_single(product.get(id=id)) return { "status": "ok", "data": { "product\_details": response, "business\_details": { "name": business.business\_name, "city": business.city, "region": business.region, "description": business.business\_description, "logo": business.logo, "owner\_id": owner.id, "business\_id": business.id, "email": owner.email, "join\_date": owner.join\_date.strftime("%b %d %Y") } } } @app.delete("/product/{id}") async def delete\_product(id: int, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Delete attempt for product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner if user == owner: await product.delete() #return {"status": "YAYA"} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated to perform this acton", headers = {"WWW-Authenticate": "Bearer"} ) return {"status": "ok"} @app.put("/product/{id}") async def update\_product(id: int, update\_info: product\_pydanticIn, user: user\_pydantic = Depends(get\_current\_user)): logger.info(f"Update attempt on product ID {id} by user {user.username}") product = await Product.get(id=id) business = await product.business\_owner owner = await business.owner update\_info = update\_info.dict(exclude\_unset=True) update\_info["date\_published"] = datetime.utcnow() if user == owner and update\_info["original\_price"] > 0: update\_info["percentage\_discount"] = ((update\_info["original\_price"] - update\_info["new\_price"]) / update\_info["original\_price"]) \* 100 product = await product.update\_from\_dict(update\_info) await product.save() response = await product\_pydantic.from\_tortoise\_orm(product) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) @app.post("/business/{id}") async def update\_business(id: int, update\_business: business\_pydanticIn, user: user\_pydantic=Depends(get\_current\_user)): logger.info(f"Business update request for business ID {id} by user {user.username}") update\_business = update\_business.dict() business = await Business.get(id=id) business\_owner = await business.owner if user == business\_owner: await business.update\_from\_dict(update\_business) business.save() response = await business\_pydantic.from\_tortoise\_orm(business) return {"status": "ok", "data": response} else: raise HTTPException( status\_code = status.HTTP\_401\_UNAUTHORIZED, detail = "Not authenticated or invalid input", headers = {"WWW-Authenticate": "Bearer"} ) register\_tortoise( app, db\_url = "sqlite://database.sqlite3", modules = {"models": ["models"]}, generate\_schemas = True, add\_exception\_handlers = True ) is the logging ok in here

**ChatGPT said:**

You've done a solid job integrating logging into your FastAPI app! ✅  
You're using Python’s built-in logging module properly, and you're already logging useful events like login attempts, registrations, uploads, and CRUD actions.

Let’s walk through and evaluate your logging configuration and usage:

**✅ 1. Logging Configuration (at the top)**

python

CopyEdit

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", filemode="a" ) logger = logging.getLogger(\_\_name\_\_)

**🔍 Evaluation:**

* **Level**: INFO is good for production. You can switch to DEBUG for development.
* **Format**: Great — includes timestamp, level, and message.
* **Filename**: Logs go to app.log in the working directory — this is good.
* **Filemode**: Appending (a) is safe. Overwrite (w) only if you want fresh logs per run.

✅ **This is correct and well-structured.**

**✅ 2. Logging Usage Throughout the App**

You’re logging:

* ✅ Login attempts
* ✅ Token decoding and validation
* ✅ Registration attempts and results
* ✅ Email verifications
* ✅ Upload events
* ✅ CRUD operations (create, update, delete)

You're using the right **log levels** for the right situations:

* info() for standard actions
* warning() for suspicious/invalid actions
* debug() for deeper internal details (like token decoding)
* error() would be great to add for try/except blocks (e.g., database errors)

✅ **Excellent coverage and good practices.**

**🛠️ Suggested Improvements**

Here are a few small but useful suggestions:

**🔹 1. Add**error()**logs for exceptions**

In your try/except blocks, especially for database operations or JWT decoding:

python

CopyEdit

except Exception as e: logger.error(f"Token validation error: {e}")

python

CopyEdit

except IntegrityError as e: logger.error(f"Database integrity error: {e}") raise HTTPException(status\_code=400, detail="Username or Email already exists")

**🔹 2. Make log file path configurable**

Right now it's just "app.log" — you could make this configurable using an env variable:

python

CopyEdit

import os LOG\_PATH = os.getenv("LOG\_PATH", "app.log") logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename=LOG\_PATH, filemode="a" )

Then you can control it from Docker or .env files.

**🔹 3. Output logs to both file and console (stdout)**

If you're running this in Docker, you may want logs in both the file **and** the terminal (for docker logs to show them).

Here’s how you can modify the logging setup:

python

CopyEdit

import logging logger = logging.getLogger() logger.setLevel(logging.INFO) # File handler file\_handler = logging.FileHandler("app.log") file\_handler.setFormatter(logging.Formatter("%(asctime)s - %(levelname)s - %(message)s")) # Stream (console) handler stream\_handler = logging.StreamHandler() stream\_handler.setFormatter(logging.Formatter("%(asctime)s - %(levelname)s - %(message)s")) # Add both handlers logger.addHandler(file\_handler) logger.addHandler(stream\_handler)

✅ Now logs will go both to app.log and stdout — useful for Docker and debugging.

**📦 Inside Docker: Where are the logs?**

With this current setup, logs will be written to a file in the container: /app/app.log.

To view logs **while the container is running**:

bash

CopyEdit

docker exec -it <container-name> cat /app/app.log

If you use the **console stream logging (above)**, you can also do:

bash

CopyEdit

docker logs <container-name>

**✅ Final Verdict**

Your logging setup is **already solid** and well-used across the application.  
With just a few enhancements (dual logging, more error() logs, optional config), you'll have a production-grade logging system. 👏

Let me know if you'd like to integrate with external tools like:

* **ELK stack (Elasticsearch, Logstash, Kibana)**
* **Grafana Loki**
* **AWS CloudWatch**
* **Sentry (for error reporting)**

These are all next-level improvements you can consider for deployment and monitoring.

**You said:**

🚀 FastAPI app starting... 🌐 Visit: http://127.0.0.1:8000⁠ INFO: Started server process [8] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000⁠ (Press CTRL+C to quit) 🚀 FastAPI app starting... 🌐 Visit: http://127.0.0.1:8000⁠ INFO: Started server process [7] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000⁠ (Press CTRL+C to quit) only this are showing in the logs in docker container i have login after running the app why that is not showing

**ChatGPT said:**

Excellent observation — what you're seeing is expected *if* you’ve only configured logging to write to a file (app.log) but not to the console (stdout). Here's what's happening:

**🧠 What's Going On?**

You’re currently using this setup in your code:

python

CopyEdit

logging.basicConfig( level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s", filename="app.log", # 🔴 Only logging to a file filemode="a" )

**🔍 Problem:**

* **Logs are going into the**app.log**file inside the container**, not printed to the console.
* Docker captures **only stdout/stderr**, not files, so docker logs <container> only shows what’s printed (like Uvicorn’s startup messages).
* Hence, login attempts, image uploads, etc., do log — but only **in the file**, not in Docker logs.

**✅ Solution: Log to Both File and Console**

Update your logging config like this:

python

CopyEdit

import logging import os # Create a logger logger = logging.getLogger() logger.setLevel(logging.INFO) # Create format formatter = logging.Formatter("%(asctime)s - %(levelname)s - %(message)s") # File handler file\_handler = logging.FileHandler("app.log") file\_handler.setFormatter(formatter) # Stream handler (console) stream\_handler = logging.StreamHandler() stream\_handler.setFormatter(formatter) # Add both handlers to the logger logger.addHandler(file\_handler) logger.addHandler(stream\_handler)

Now:

* Your logs will show **inside the container (**app.log**)**
* And they'll also show in your terminal or Docker logs:

bash

CopyEdit

docker logs <container-name>

**🔧 Bonus: Logging only once**

If you're running this inside a module (e.g., main.py) and it gets imported elsewhere, Python might attach multiple handlers and duplicate logs.

Add this guard:

python

CopyEdit

if not logger.hasHandlers(): logger.addHandler(file\_handler) logger.addHandler(stream\_handler)

**✅ After That**

You should see entries like:

pgsql

CopyEdit

2025-06-17 14:02:00 - INFO - Login attempt for user: admin

2025-06-17 14:02:01 - INFO - User created: admin

...directly in your terminal **or** from docker logs <container>.

Let me know if you want to **tail** the log file live from the container or want these logs shipped externally (e.g., to AWS, Logstash, etc.).

**You said:**

This site can’t be reached 127.0.0.1 refused to connect. Try: Checking the connection Checking the proxy and the firewall ERR\_CONNECTION\_REFUSED this is shown when i run it

**ChatGPT said:**

Thanks for the detail — that error message is a common issue when working with **Dockerized web apps**, especially FastAPI.

Let’s break down why **“127.0.0.1 refused to connect”** appears and how to fix it:

**🚨 Why You're Seeing This**

You're seeing:

nginx

CopyEdit

This site can’t be reached

127.0.0.1 refused to connect.

Because you're trying to access http://127.0.0.1:8000 **on your host machine**, but:

* Your **FastAPI app is running *inside* the Docker container**, not directly on your host.
* And inside the container, it's listening on 0.0.0.0:8000, not 127.0.0.1 on your host.

**✅ Fix: Expose Port to Host**

**Make sure you're running the Docker container with port mapping, like this:**

bash

CopyEdit

docker run -d -p 8000:8000 your-image-name

✅ This means:

* The container's 0.0.0.0:8000 is forwarded to your host's localhost:8000.

Then visit:

arduino

CopyEdit

http://localhost:8000

❗ **NOT**127.0.0.1:8000**from inside the container** — that's internal to the container itself.

**🧪 Check If the Container is Listening Correctly**

Run:

bash

CopyEdit

docker ps

You should see something like:

nginx

CopyEdit

PORTS

0.0.0.0:8000->8000/tcp

That confirms the host can access the container’s port.

**✅ Double-Check: Uvicorn Host Binding**

In your start.sh or wherever you launch Uvicorn, ensure it's binding to 0.0.0.0, like you already have:

bash

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000

✅ Good: This makes Uvicorn listen on **all network interfaces**, including Docker's internal bridge.

**🛠 If You're Using Docker Desktop (Windows/macOS)**

Use:

arduino

CopyEdit

http://localhost:8000

Not:

cpp

CopyEdit

http://127.0.0.1:8000

Docker routes localhost differently on some OSes, but it's generally the correct access point.

**🧩 TL;DR Checklist**

| **Item** | **Status** |
| --- | --- |
| Docker port mapped? | -p 8000:8000 |
| Uvicorn host is 0.0.0.0? | ✅ |
| Using localhost:8000? | ✅ (not 127.0.0.1 inside container) |
| Docker container running? | docker ps to confirm |

If you’ve done all of the above and it **still doesn't work**, let me know:

* How you're running the container
* OS you're on (Windows, macOS, Linux)
* Docker run command you're using

And I’ll help debug it further.

**You said:**

#!/bin/sh LOGFILE="/app/logs/startup.log" mkdir -p /app/logs echo "" | tee -a $LOGFILE echo "🚀 FastAPI app starting..." | tee -a $LOGFILE echo "🌐 Visit: http://localhost:8000" | tee -a $LOGFILE echo "" | tee -a $LOGFILE # Start FastAPI with logging uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE this is my start.sh file

**ChatGPT said:**

Thanks for sharing your updated start.sh file — it’s **almost perfect** and does what it’s supposed to: log both to the terminal (stdout) *and* to a file inside the container.

But there are a couple of things worth reviewing and potentially improving.

**✅ What’s Working Well**

1. **Creates a log directory**:

sh

CopyEdit

mkdir -p /app/logs

1. **Logs startup messages** to both:
   * The terminal (so docker logs can show them)
   * A log file (/app/logs/startup.log) via tee
2. **Runs Uvicorn and pipes its output** to both places:

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

This is all good practice. 👌

**🔍 Small Improvements to Consider**

**1. Use**exec**for Uvicorn to handle shutdown signals properly**

Right now:

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

But this means sh is the parent process, and signal forwarding (like CTRL+C or Docker stop signals) can be flaky.

Instead, do:

sh

CopyEdit

exec uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

✅ exec replaces the shell process with Uvicorn directly, making signal handling cleaner and your container behave properly with docker stop.

**2. Optional: Add timestamp or container name to logs**

For multi-container environments or production, you might want to include:

sh

CopyEdit

echo "$(date) - 🚀 FastAPI app starting..." | tee -a $LOGFILE

That way you know when the app started in the logs.

**3. Optional: Avoid overwriting logs on container restart**

Right now you’re using tee -a, so it appends to startup.log, which is good.

But if your container is removed and re-run, the log file is deleted anyway — unless you **mount a volume**.

To persist logs across runs:

bash

CopyEdit

docker run -v $(pwd)/logs:/app/logs -p 8000:8000 your-image-name

This saves logs to your host machine’s ./logs/ directory.

**✅ Final Version of**start.sh

Here’s the slightly improved version:

sh

CopyEdit

#!/bin/sh LOGFILE="/app/logs/startup.log" mkdir -p /app/logs echo "" | tee -a $LOGFILE echo "$(date) - 🚀 FastAPI app starting..." | tee -a $LOGFILE echo "$(date) - 🌐 Visit: http://localhost:8000" | tee -a $LOGFILE echo "" | tee -a $LOGFILE # Run the app and ensure signals are handled properly exec uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

**🧩 Summary**

| **Feature** | **Status** |
| --- | --- |
| Logs to file and console | ✅ |
| Works with docker logs | ✅ |
| Uses exec for clean shutdown | ✅ (after fix) |
| Creates /app/logs safely | ✅ |
| Optionally persists logs | ✅ (with volume) |

Let me know if you want help setting up **log rotation**, forwarding logs to the host, or integrating with external log collectors like **ELK**, **Grafana Loki**, or **CloudWatch**.

**You said:**

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY app/ /app/ COPY start.sh /app/start.sh RUN chmod +x /app/start.sh CMD ["sh", "/app/start.sh"] this is my dockerfile

**ChatGPT said:**

Great! Your Dockerfile is clean and functional. Let's walk through it line-by-line and break down what **each line does**, and how it ties into your overall FastAPI app setup:

**✅ 1. Base Image**

dockerfile

CopyEdit

FROM python:3.10-slim

* **Purpose**: This sets the base image to Python 3.10 using the lightweight slim variant, which is smaller in size (faster build & deploy).
* Ideal for production unless you need build tools or extra libraries.

**✅ 2. Working Directory**

dockerfile

CopyEdit

WORKDIR /app

* **Purpose**: Sets the working directory **inside the container** to /app.
* Any subsequent commands (like COPY or RUN) will operate from /app.

**✅ 3. Install Dependencies**

dockerfile

CopyEdit

COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt

* COPY requirements.txt .: Copies your requirements.txt file into the container at /app/requirements.txt.
* RUN pip install --no-cache-dir -r requirements.txt: Installs all Python dependencies without caching (to save space).

**✅ 4. Copy Application Code**

dockerfile

CopyEdit

COPY app/ /app/

* **Purpose**: Copies the contents of your local app/ folder into the container’s /app/ folder.
* Your main.py, models, routes, etc., should be inside the local app/ directory.
* The trailing / ensures the contents go into /app/, not /app/app.

✅ app/ is a **local directory** (next to your Dockerfile), /app/ is the **destination inside the container**.

**✅ 5. Add and Make**start.sh**Executable**

dockerfile

CopyEdit

COPY start.sh /app/start.sh RUN chmod +x /app/start.sh

* **COPY**: Brings in your startup script.
* **RUN chmod +x**: Makes the script executable inside the container.

**✅ 6. Set the Default Command**

dockerfile

CopyEdit

CMD ["sh", "/app/start.sh"]

* **Purpose**: When the container starts, it will run the start.sh script.
* This launches Uvicorn and starts your FastAPI app.

**🔍 How it all works together**

1. Docker builds a lightweight Python 3.10 container.
2. Your dependencies are installed.
3. Your FastAPI app code and start.sh script are copied into the image.
4. When the container starts, start.sh:
   * Logs a startup message.
   * Launches Uvicorn (FastAPI server) on 0.0.0.0:8000.

**🔄 Optional Enhancements**

| **Feature** | **How to Add** |
| --- | --- |
| Auto-reload in development | Add --reload to uvicorn in start.sh (not for prod) |
| Persist logs outside container | -v $(pwd)/logs:/app/logs in docker run |
| Use .env file for secrets | Add python-dotenv, and load .env inside main.py |
| Minimize Docker image size | Use multi-stage builds (if needed) |

**✅ TL;DR: Your Dockerfile is correct and production-ready for most uses.**

Let me know if you want to:

* Serve it behind **Nginx**
* Add **HTTPS**
* Deploy it to **Docker Compose**, **Heroku**, **Fly.io**, or **EC2**  
  I'm happy to walk you through that as well.

**You said:**

This page isn’t working 127.0.0.1 didn’t send any data. ERR\_EMPTY\_RESPONSE now this is showing

**ChatGPT said:**

Thanks for the update — the error:

**"This page isn’t working — 127.0.0.1 didn’t send any data. ERR\_EMPTY\_RESPONSE"**

means your **browser tried to connect, but the server didn't respond** — likely because it's either:

1. Not running at all
2. Running but not accessible from your host
3. Misconfigured networking or wrong port access

**✅ Let's Troubleshoot Step-by-Step**

**✅ 1. Did you expose the port when running the container?**

You must map the container’s internal port (8000) to your local machine:

bash

CopyEdit

docker run -p 8000:8000 your-image-name

Without -p 8000:8000, your app is running **inside** the container but **not reachable** from your browser.

**✅ 2. Check if the app is running**

Run:

bash

CopyEdit

docker ps

You should see something like:

nginx

CopyEdit

CONTAINER ID IMAGE PORTS STATUS

abcd1234 your-image-name 0.0.0.0:8000->8000/tcp Up 2 minutes

If you don’t see 8000->8000, then the port wasn't mapped — go back to step 1.

**✅ 3. Confirm Uvicorn is listening**

Your start.sh should include:

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000

✅ Good: 0.0.0.0 allows access from outside the container (your host machine).

🚫 **Bad**: If it’s set to 127.0.0.1, it won’t be accessible from the host.

**✅ 4. Check container logs**

Run:

bash

CopyEdit

docker logs <container\_id>

Look for:

nginx

CopyEdit

Uvicorn running on http://0.0.0.0:8000

If you see errors (like “ModuleNotFound” or “main not found”), the app isn't running.

**✅ 5. Access via**localhost**, not 127.0.0.1 (if on Docker Desktop)**

Sometimes on Windows/macOS with Docker Desktop, 127.0.0.1:8000 might fail.

Try this instead:

arduino

CopyEdit

http://localhost:8000

Or from **WSL/Linux host**:

bash

CopyEdit

curl http://localhost:8000

You should get a JSON response like:

json

CopyEdit

{"Message":"Hello World"}

**✅ 6. If you see**ERR\_EMPTY\_RESPONSE**, the container might have crashed**

Try:

bash

CopyEdit

docker ps -a

If the container **exited**, inspect its logs with:

bash

CopyEdit

docker logs <container\_id>

Check for Python tracebacks or FastAPI import errors.

**✅ Final Checklist**

| **Check** | **Status** |
| --- | --- |
| -p 8000:8000 used in docker run? | ✅ Required |
| Uvicorn uses --host 0.0.0.0? | ✅ Required |
| FastAPI app is named main:app? | ✅ Check module path |
| Docker container is **still running**? | ✅ docker ps |
| Accessing via http://localhost:8000? | ✅ Try instead of 127.0.0.1 |

If you're still stuck, paste your:

* docker run command
* docker ps output
* Full logs from docker logs <container-id>

I'll help pinpoint it in seconds.

**You said:**

cant i run the app using docker without running it in my local device

**ChatGPT said:**

Yes, **you absolutely can run your FastAPI app entirely inside Docker**, **without running it manually on your local device**.

But let's clarify what that actually means and how to do it:

**✅ What You *Want* to Do**

You're asking:

"Can I run my app inside Docker, and have it start automatically, without me having to run it manually on my local system?"

**Yes. That’s exactly what Docker is designed for.** Once the container starts, your FastAPI app should run **automatically** — with no need to launch anything manually on your machine.

**🧠 How It Works**

Here’s how the full setup should behave:

1. You **build the Docker image**:

bash

CopyEdit

docker build -t my-fastapi-app .

1. Then you **run the container**:

bash

CopyEdit

docker run -d -p 8000:8000 my-fastapi-app

1. Docker:
   * Starts the container
   * Runs your start.sh
   * Launches uvicorn
   * Binds to 0.0.0.0:8000 inside the container
   * Forwards port 8000 to your local localhost:8000
   * You can now visit: http://localhost:8000

✅ You **do not need to run the app manually** in VSCode, Terminal, or anything else.

**🔍 What Might Be Going Wrong?**

If it's not working as expected, here are **possible reasons**:

| **Issue** | **Explanation** | **Fix** |
| --- | --- | --- |
| Port not mapped | You're not using -p 8000:8000 in docker run | Add -p 8000:8000 |
| App not starting inside container | CMD or start.sh is broken | Check logs with docker logs <id> |
| Using 127.0.0.1 | Container's 127.0.0.1 ≠ host's | Use http://localhost:8000 |
| App crashes | Code error (e.g. import, DB) | Check logs for traceback |
| Uvicorn binding to wrong host | If --host 127.0.0.1, it's not reachable | Use --host 0.0.0.0 |

**🔁 Full Working Example**

Here’s a **working setup**:

**📄**Dockerfile

Dockerfile

CopyEdit

FROM python:3.10-slim WORKDIR /app COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt COPY app/ /app/ COPY start.sh /app/start.sh RUN chmod +x /app/start.sh CMD ["sh", "/app/start.sh"]

**📄**start.sh

sh

CopyEdit

#!/bin/sh LOGFILE="/app/logs/startup.log" mkdir -p /app/logs echo "$(date) - 🚀 FastAPI app starting..." | tee -a $LOGFILE echo "$(date) - 🌐 Visit: http://localhost:8000" | tee -a $LOGFILE echo "" | tee -a $LOGFILE # Start Uvicorn exec uvicorn main:app --host 0.0.0.0 --port 8000 | tee -a $LOGFILE

Then run:

bash

CopyEdit

docker build -t my-fastapi-app . docker run -d -p 8000:8000 my-fastapi-app

Visit:  
👉 [http://localhost:8000](http://localhost:8000/)  
You should see {"Message": "Hello World"} or your FastAPI root response.

**🧪 Want It to Start on System Boot?**

If you want the app to run **without you even running the**docker run**command**, you can:

* Use **Docker Compose** with restart: always
* Or run the container in background and set it to restart automatically:

bash

CopyEdit

docker run -d --restart=always -p 8000:8000 my-fastapi-app

This ensures your app **auto-starts even after a system reboot**.

Let me know your OS and how you're launching the container if you're still facing issues — I can guide you with exact commands.

**You said:**

8726e5dd98c21eab1d47cd4ac619924cff75b131cffdea6429dec12d3068c22d docker: Error response from daemon: failed to set up container networking: driver failed programming extern al connectivity on endpoint practical\_blackburn (88e29b136e9486c46021893f982847b327d9ccbeca80731444151296e1 48ec7d): Bind for 0.0.0.0:8000 failed: port is already allocated this is shown

**ChatGPT said:**

Thanks — that error message makes it **crystal clear** what's happening:

**❌ Error Explained**

nginx

CopyEdit

Bind for 0.0.0.0:8000 failed: port is already allocated

This means:

* Something (probably another Docker container or a local app) is **already using port 8000** on your system.
* Docker **can’t bind to that port** again — because only one process can use a specific port on the same IP.

**✅ Solutions**

**🔧 Option 1: Stop the container that’s using port 8000**

Run:

bash

CopyEdit

docker ps

Look for a container using 0.0.0.0:8000->8000. Then stop it:

bash

CopyEdit

docker stop <container\_id>

And optionally remove it:

bash

CopyEdit

docker rm <container\_id>

Then re-run your app:

bash

CopyEdit

docker run -d -p 8000:8000 my-fastapi-app

**🔄 Option 2: Use a different host port**

If you want to run **multiple apps** or **don’t want to stop the other one**, map a different host port:

bash

CopyEdit

docker run -d -p 8080:8000 my-fastapi-app

* This maps container’s 8000 ➝ your local 8080
* Access it at: [http://localhost:8080](http://localhost:8080/)

**🧹 Optional: Free All Containers Using 8000**

If you want to force stop all containers using 8000:

bash

CopyEdit

docker ps --filter "publish=8000"

Then stop any listed container IDs.

**✅ Summary**

| **Action** | **Command** |
| --- | --- |
| See running containers | docker ps |
| Stop the container using port 8000 | docker stop <id> |
| Use another port | docker run -p 8080:8000 my-fastapi-app |
| See what’s using 8000 (Linux/macOS) | lsof -i :8000 or `netstat -anp |

Let me know what OS you’re on if you want exact commands to check for port conflicts.

**You said:**

This site can’t be reached 127.0.0.1 refused to connect. Try: Checking the connection Checking the proxy and the firewall ERR\_CONNECTION\_REFUSED

**ChatGPT said:**

You're seeing this error again:

**"127.0.0.1 refused to connect. ERR\_CONNECTION\_REFUSED"**

Which usually means:

* The Docker container is either **not running**, or
* The container is **not exposing port 8000 correctly**, or
* The app inside the container **isn't starting properly**, or
* You're trying to connect to the **wrong port or host**.

**✅ Let's Fix It Step-by-Step**

**🟢 Step 1: Confirm Container is Running**

Run this command:

bash

CopyEdit

docker ps

✅ You should see your container listed, and it should show something like:

bash

CopyEdit

0.0.0.0:8000->8000/tcp

If not, it means the container has **crashed or exited**.

**🟢 Step 2: Check Docker Logs**

Get the container ID from docker ps, then:

bash

CopyEdit

docker logs <container\_id>

Look for:

* Errors like ModuleNotFoundError, ImportError, or SyntaxError
* Or if Uvicorn failed to start

You should see:

nginx

CopyEdit

Uvicorn running on http://0.0.0.0:8000

**🟢 Step 3: Ensure Port is Mapped Correctly**

When you start your container, use:

bash

CopyEdit

docker run -d -p 8000:8000 my-fastapi-app

Or use another port (like 8080):

bash

CopyEdit

docker run -d -p 8080:8000 my-fastapi-app

Then access:  
👉 [http://localhost:8080](http://localhost:8080/)

**🟢 Step 4: Check if App is Listening on the Right Host**

Make sure your start.sh has:

sh

CopyEdit

uvicorn main:app --host 0.0.0.0 --port 8000

❌ Do not use 127.0.0.1 inside Docker — the container’s internal loopback isn’t exposed to your browser.

✅ Use 0.0.0.0 so Docker can forward traffic to it.

**🟢 Step 5: Access It the Right Way**

After confirming the app is running, use your **browser or curl** to check:

* ✅ [http://localhost:8000](http://localhost:8000/)
* ✅ Or whatever port you mapped (like 8080)

**🧪 Test with**curl

Try running this in terminal:

bash

CopyEdit

curl http://localhost:8000

You should see a response like:

json

CopyEdit

{"Message": "Hello World"}

**🛠 Still Not Working?**

Please copy-paste the results of:

bash

CopyEdit

docker ps docker logs <your\_container\_id>

I’ll help debug it directly from there.

Also let me know:

* Are you on **Windows, macOS, or Linux**?
* Are you using **Docker Desktop**, WSL, or native Docker?

This helps give exact steps for your environment.

**You said:**

ok there was an existing container when i bashed the command in cmd it made a new container that worked so explain how this works how docker runs applications and what are the roles of containers and images

**ChatGPT said:**

Great! You figured out the core issue — **an existing container was using the port**, and once you launched a new one correctly, your app worked. That’s classic Docker behavior.

Let’s break this down so you understand **how Docker works**, including:

**🧠 What Is Docker?**

Docker is a **containerization platform**. It lets you:

* Package your app and its dependencies together
* Run it reliably on any system
* Avoid “it works on my machine” problems

**📦 Docker Concepts You Must Know**

| **Term** | **What It Is** | **Analogy** |
| --- | --- | --- |
| **Image** | A snapshot of an app + environment (like a recipe) | A blueprint or template |
| **Container** | A running instance of an image (like a virtual environment) | A running app based on a blueprint |
| **Dockerfile** | Instructions to build an image | A recipe for how to bake the app |
| **Docker Engine** | The background service that runs everything | The kitchen |

**🔄 Docker Workflow Overview**

1. **Write a Dockerfile**
   * Tells Docker how to install dependencies and run your app
2. **Build an Image**

bash

CopyEdit

docker build -t my-fastapi-app .

* + Creates an image using the Dockerfile

1. **Run a Container**

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

* + Starts a new container based on the image
  + Maps **container’s port 8000** → **your host’s port 8000**
  + This makes it accessible via localhost:8000

**⚙️ What Happens When You Run**docker run

bash

CopyEdit

docker run -p 8000:8000 my-fastapi-app

Docker:

1. Uses the image my-fastapi-app
2. Starts a container
3. Executes the command from your Dockerfile → runs start.sh → starts uvicorn
4. Forwards port 8000 from container → your system (host)
5. Keeps it isolated from the rest of your machine

You can start **multiple containers** from the same image.  
Each is **isolated**, and each needs its **own port** if exposed.

**🛢️ Images vs Containers: In Practice**

| **Feature** | **Image** | **Container** |
| --- | --- | --- |
| **Static** | Yes — it doesn’t change | No — it can write logs, mutate files |
| **Reusable** | Yes — can launch many containers | No — each is a separate instance |
| **Stopped** | Never — it’s just data | Yes — a container can be stopped |
| **Run Command** | docker build | docker run |

**🧹 What Happened in Your Case**

You had:

* A **running container** already using port 8000
* So when you ran a new one, Docker said:

“I can’t start a second one on the same port”

Once you stopped or removed the old one, the new container was free to bind to port 8000.

**🔄 Common Docker Commands You Should Know**

| **Action** | **Command** |
| --- | --- |
| Build image | docker build -t my-app . |
| Run container | docker run -p 8000:8000 my-app |
| List running containers | docker ps |
| List all containers | docker ps -a |
| Stop a container | docker stop <container\_id> |
| Remove a container | docker rm <container\_id> |
| Remove an image | docker rmi <image\_id> |
| See logs from a container | docker logs <container\_id> |
| Open a shell inside container | docker exec -it <container\_id> sh |

**🧠 Think of It Like This**

* **Dockerfile** = your cooking recipe
* **Image** = a frozen pizza based on your recipe
* **Container** = the pizza in the oven — hot and running
* You can make many pizzas (containers) from one recipe (image)
* But only one can use the oven (port 8000) at a time

Let me know if you'd like a visual or Docker Compose introduction next — it helps manage multiple containers easily (DB + backend + frontend).

Top of Form

Attach

Search

**Voice**

Bottom of Form

ChatGPT can make mistakes. Check important info. See Cookie Preferences.