ROUTERS :-

⇒ Keeping all the path operation in a single file may be tedious for large projects.
Thus, using Routers, we can seperate and categorise path operations into multiple files, for better organisation.

In our project, we will keep all path operations handling CRUD features in a single file, and all the path operations related to user functionality in another file.

⇒ We create a folder (inside our project folder) named routers, and in it create 2 files, name:-

```
crud_api_project
  → database.py
  → models.py
  → __init__.py
  → schemas.py
  → routers
      → user.py
      → post.py
```

These files will contain the path operations, seperately and categorically.

NOTE

To import files from a folder, that is one folder above in hierarchy that current file folder, we use double dot (..) as follow:-

In test.py, importing abc.py:-

```
dir1
 ├─ abc.py
 └─ dir2
     └─ test.py
```

```
from .. import abc
```

double dot mean file to be imported is one folder above the current folder.

APIRouter :-

⇒ We will import everything in the new files, that are required by the path operations in the respective files.

But, we will not import the necessary, or make the `app = FastAPI()` object into those files. which now contains the path operations.

Insted, in these new files (ex:- post.py & user.py) we will use APIRouter as follow:-

```
from fastapi import APIRouter

router = APIRouter()
```

router → router object

Now, in place of app object in decorators, we need to replace and use the router object.

Ex:-

```
@router.get("/posts", ....
  - -
@router.post("/users", ...)
@router.delete("/posts/{id}", ...)
```

— and so on.

6:23:20

So, by this, our path operations are ready. Now, we need to link those path operations, in our main.py file, using routers :-

In main.py :-

```
from .routers import post, user
```

(.routers → folder; post, user → router files containing the path operations)

```
app.include_router(post.router)
app.include_router(user.router)
```

(router → object used in path operation decorators)

This is the thing that we do in Django urlpatterns including from various file.

As soon as we request to the API, the app instance is found. Now, the include_router() method redirects the search for url pattern matching with the path operations contained in the file, (associated with router object) that is being passed as parameter.

Let's take include_router(post.router) for e.g.. Here, redirection for search will happen, such that the url pattern will be matched with all the path operations that are associated with the post.router object, with matching HTTP method.

Thus, basically, in simple terms, we can think of it as the specified path operations gets imported.

Thus, like this, path operations are organised, and main.py file acts as link, containing the main code.

In the URL endpoint pattern, we can eliminate the similar prefix for every path operator decorator, so that it looks clean & organised.

For example, in our project, the path operations for Post crud features look like this:-

@router.get("/posts"...) → Get all posts

@router.get("/posts/{id}",...) → Get single post

@router.post("/posts",...) → Create post

@router.put("/posts/{id}",...) → Update post

@router.delete("/posts/{id}",...) → delete post

In every decorator, as url path, it starts with "/posts" in all the cases.

For long endpoint paths, writing this everytime may look odd.

So, we can remove writing the common path prefix as follow :-

common prefix in all path operation

```
router = APIRouter(prefix = "/posts")
```

Thus, now, all the decorators, used with this router object will be prefixed by "/posts", and we need to just mention the rest part of the endpoints.

Thus, now, we will write the Post CRUD related path operation decorators as follows :-

We must put single slash, for nothing after prefix

```
@router.get("/")          → Get all posts
@router.get("/{id}")      → Get single post
@router.post("/", ...)    → Create post
@router.put("/{id}", ...) → Update post
@router.delete("/{id}", ...) → delete post
```

This is the part after the router prefix

Structuring FastAPI's Interactive Documentation

⇒ We can see interactive documentation of our API at 127.0.0.1:8000/docs. But, as of now, all the path operations will be grouped there as a single unit, together.

⇒ To group path operations based on their functionality in the documentation, we need to set the tags attribute of the APIRouter() method, in each router.
We provide the path operation group names as a list, to the tags attribute:-

For post.py:-

```
router = APIRouter(
    prefix = "/posts"
    tags = ["Posts"]
)
```

group name in the documentation, containing all path operations linked to its own router object.

For user.py

```
router = APIRouter(
    prefix = "/users"
    tags = ["Users"]
)
```

passed as list

attribute of APIRouter() method