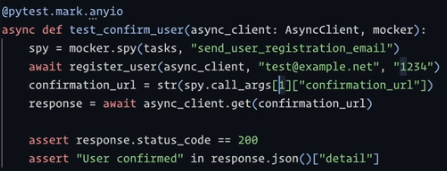
**Note 1**: Here We Pass The confirmation\_url AS Named Args, So We Can Access It By call\_args[1]

**Note 2**: The call\_args[0] For Tuple Of Arguments



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Add BackgroundTasks To Our APP:

**Note 1**: If We Have Async/Await Function, Then Background Tasks Will Await It Until It Finish.

**Note 2**: We Can Pass The Parameters To Background Task Function By Position, OR By Name

from fastapi import APIRouter, BackgroundTasks, HTTPException, status, Request, Depends

@router.post("/register", status\_code=201)

async def register(user: UserIn, background\_tasks: BackgroundTasks, request: Request):

    t1 = await get\_user\_by\_email(user.email)

    if t1:

        raise HTTPException(

            status\_code=status.HTTP\_400\_BAD\_REQUEST,

            detail="User Already Exists With That Email",

        )

    query = users\_table.insert().values(

        email=user.email, password=get\_password\_hash(user.password),

        confirmed=False,

    )

    logger.debug(f"The Query For Creating User: {query}")

    result = await database.execute(query)

    background\_tasks.add\_task(

        send\_user\_registeration\_email,

        email=user.email,

        confirmation\_url=str(

            request.url\_for(

                "confirm\_user\_email",

                token=create\_confirm\_token(user.email)

            )

        )

    )

    return {

        "msg": "User Created Successfully",

        "id": result,

        "confirmation\_url": request.url\_for(

            "confirm\_user\_email",

            token=create\_confirm\_token(user.email)

        )

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For Handling The Async I/O Operations Using FastAPI, We Can Use: aiofiles

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Upload Files Using FastAPI, we Must Install: *pip install python-multipart*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

from fastapi import APIRouter, UploadFile, File, HTTPException

from fastapi.responses import JSONResponse

import aiofiles

import os

from pathlib import Path

router = APIRouter()

UPLOAD\_DIRECTORY = "uploads"  # Directory where files will be saved

MAX\_FILE\_SIZE = 1024 \* 1024 \* 2  # 2 MB limit

ALLOWED\_FILE\_TYPES = {"image/jpeg", "image/png", "application/pdf"}  # Allowed MIME types

# Ensure upload directory exists

Path(UPLOAD\_DIRECTORY).mkdir(parents=True, exist\_ok=True)

@router.post('/upload/')

async def upload\_file(file: UploadFile = File(...)):

    try:

        # Validate file size

        file.file.seek(0, 2)  # Move to end of file

        file\_size = file.file.tell()

        if file\_size > MAX\_FILE\_SIZE:

            raise HTTPException(status\_code=413, detail="File too large")

        file.file.seek(0)  # Reset file pointer

        # Validate file type

        if file.content\_type not in ALLOWED\_FILE\_TYPES:

            raise HTTPException(status\_code=400, detail="Invalid file type")

        # Create safe filename

        file\_name = file.filename

        file\_path = os.path.join(UPLOAD\_DIRECTORY, file\_name)

        # Check if file exists and modify filename if needed

        counter = 1

        while os.path.exists(file\_path):

            name, ext = os.path.splitext(file\_name)

            file\_path = os.path.join(UPLOAD\_DIRECTORY, f"{name}\_{counter}{ext}")

            counter += 1

        # Save file asynchronously

        async with aiofiles.open(file\_path, 'wb') as out\_file:

# This Will Read Only 1KB

            while content := await file.read(1024):  # Read in chunks

                await out\_file.write(content)

        return JSONResponse(

            status\_code=200,

            content={

                "message": "File uploaded successfully",

                "file\_path": file\_path,

                "file\_size": file\_size,

                "content\_type": file.content\_type

            }

        )

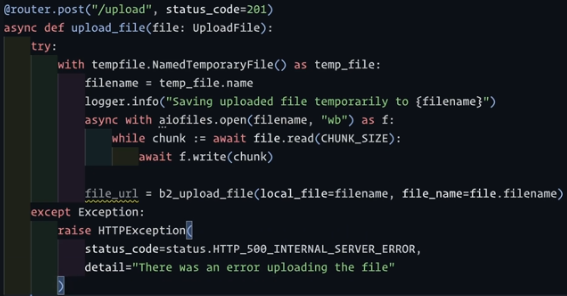
    except Exception as e:

        raise HTTPException(status\_code=500, detail=str(e))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For Creating Temp Files That Deleted When *The Context Manager* (*using with-Keyword*) Is Finished, We Can Use tempfile-Python-Module.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For Faking The File System That Are Used To Store Uploaded Files We Can Use: *pip install pyfakefs*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@router.post("/", response\_model=UserPost, status\_code=201)

async def create\_post(post: UserPostIn, current\_user: Annotated[User, Depends(get\_current\_user)]):

    data = {\*\*post.model\_dump(), "user\_id": current\_user.id}

    query = posts\_table.insert().values(data)

    logger.debug(f"The Query For Create Post Is: {query}")

    last\_id = await database.execute(query)

    return  {\*\*data, "id": last\_id}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Annotated[str, Depends(oauth2\_schema)] --> That Means The Token Type Is String

# And Will Be Populated From oauth2\_schema

async def get\_current\_user(token: Annotated[str, Depends(oauth2\_schema)]):

    email = get\_subject\_for\_token\_type(token=token, token\_type='access')

    user = await get\_user\_by\_email(email=email)

    if user is None:

        raise create\_credentials\_exception(detail="Invalid Email OR Password")

    return user

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def get\_subject\_for\_token\_type(token: str, token\_type: Literal['access', 'confirmation']) -> str:

    try:

        payload = jwt.decode(token=token, key=config.SECRET\_KEY, algorithms=[config.ALGORITHM])

    except ExpiredSignatureError as e:

        raise create\_credentials\_exception(

            detail="Token Has Been Expired",

        ) from e

    except JWTError as e:

        raise create\_credentials\_exception(detail="Invalid Token") from e

    email = payload.get('sub', None)

    if email is None:

        raise create\_credentials\_exception(detail="Email Not Found")

    t1 = payload.get('type')

    if t1 is None or t1.lower() != token\_type.lower():

        raise create\_credentials\_exception(detail="Invalid Type For Token")

    return email

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*