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

```python
@pytest.mark.anyio
async def test_confirm_user(async_client: AsyncClient, mocker):
    spy = mocker.spy(tasks, "send_user_registration_email")
    await register_user(async_client, "test@example.net", "1234")
    confirmation_url = str(spy.call_args[1]["confirmation_url"])
    response = await async_client.get(confirmation_url)

    assert response.status_code == 200
    assert "User confirmed" in response.json()["detail"]
```

****

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

```python
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)
```

```python
@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.

****

```python
@router.post("/upload", status_code=201)
async def upload_file(file: UploadFile):
    try:
        with tempfile.NamedTemporaryFile() as temp_file:
            filename = temp_file.name
            logger.info("Saving uploaded file temporarily to {filename}")
            async with aiofiles.open(filename, "wb") as f:
                while chunk := await file.read(CHUNK_SIZE):
                    await f.write(chunk)

            file_url = b2_upload_file(local_file=filename, file_name=file.filename)
    except Exception:
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="There was an error uploading the file"
        )
```

****

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

****

```python
@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}
```

****

```python
# 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
**********************************************************************
```