To Define The Endpoint That Make The User Create The Token For Make Authenticated Requested.

So, It Is Populates The Documentation With Authenticated Token, And Grab The Token From URL.

from fastapi.security import OAuth2PasswordBearer

oauth2\_schema = OAuth2PasswordBearer(tokenUrl='/user/token')

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

To Use The Token Authentication For Protect Posts Endpoints, We Can Do:

from social\_network.models.user import User

from social\_network.security import get\_current\_user, oauth2\_schema

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

async def create\_post(post: UserPostIn, request: Request):

    # In This Way We Protect The Endpoint From Un-Authenticated Requests

    current\_user: User = await get\_current\_user(await oauth2\_schema(request=request))

    data = post.model\_dump()

    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}

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

For Testing We Need To Define This Fixture, To Get Token, To Use It In Creating Post, And Creating Comment:

@pytest.fixture()

async def get\_token(async\_client: AsyncClient, registered\_user: dict) -> str:

    response  = await async\_client.post(url='/user/token', json=registered\_user)

    return response.json()["access\_token"]

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

To Use Dependency Injection For Getting User Data From Token:

from fastapi import Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer

from typing import Annotated

oauth2\_schema = OAuth2PasswordBearer(tokenUrl='/user/token')

# 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)]):

    try:

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

        email = payload.get('sub', None)

        if email is None:

            raise credentials\_exception

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

        if user is None:

            raise credentials\_exception

        return user

    except ExpiredSignatureError as e:

        raise HTTPException(

            status\_code=status.HTTP\_401\_UNAUTHORIZED,

            detail="Token Has Been Expired",

            headers={

                "WWW-Authenticate": "Bearer",

            },

        ) from e

    except JWTError as e:

        raise credentials\_exception from e

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

**Note 1**: In The Previous Way, We Don’t Need To Call oath2\_schema(request) For Each Function.

**Note 2**: The Depends Function Will Get The Value Of Token Instead Of Calling The Function.

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

@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()

    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}

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

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

async def create\_comment(comment: UserCommentIn, current\_user: Annotated[User, Depends(get\_current\_user)]):

    data = comment.model\_dump()

    post = await find\_post(data['post\_id'])

    if post is None:

        raise HTTPException(status\_code=404, detail="Post Not Found")

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

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

    last\_id = await database.execute(query)

    return {\*\*data, "id": last\_id}

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

**Note**: The New Way To Use Dependency Injection With Depends Is By Using Annotated

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

To Use Authentication With Swagger API, We Can Do:

from fastapi import APIRouter, HTTPException, status, Form, Depends

from fastapi.security import OAuth2PasswordRequestForm

@router.post('/token', status\_code=200)

async def login\_user(

    username: Annotated[str, Form()],

    password: Annotated[str, Form()],

    grant\_type: Annotated[str, Form()]):

    access\_token = await authenticate\_user(username, password)

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

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

@router.post('/token', status\_code=200)

async def login\_user(form\_data:  Annotated[OAuth2PasswordRequestForm, Depends()]):

    access\_token = await authenticate\_user(form\_data.username, form\_data.password)

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

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

**Note**: The Previous Configuration, Because OAuth2 Send Data AS: application/x-www-form-urlencode

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

To Define The Likes Table:

### Create The Likes Table ###

likes\_table = sqlalchemy.Table(

    "likes",

    metadata,

    sqlalchemy.Column("id", sqlalchemy.Integer, primary\_key=True),

    # Here We Don't Need To Tell The Type Of Column, Because It's ForeignKey

    # So It Will Give It The Same Type Of Id Of Posts Table.

    sqlalchemy.Column("post\_id", sqlalchemy.ForeignKey("posts.id"), nullable=False, ),

    sqlalchemy.Column("user\_id", sqlalchemy.ForeignKey("users.id"), nullable=False),

)

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

To Make Outer Join Between Posts Table And Likes Table:

select\_post\_and\_likes = (

    sqlalchemy.select(

        posts\_table,

        # .label('likes') --> Is Similar To Use: AS-Keyword

        sqlalchemy.func.count(likes\_table.c.id).labale("likes"),

    )\

    .select\_from(posts\_table.outerjoin(likes\_table))\

    .group\_by(posts\_table.c.id)

)

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

The Model Changes:

from pydantic import BaseModel

class PostLikeIn(BaseModel):

    post\_id: int

class PostLike(PostLikeIn):

    id: int

    user\_id: int

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

from pydantic import BaseModel, ConfigDict

# This Is For Our Request Content From User

class UserPostIn(BaseModel):

    body: str

# This Is For Our Output Response For User

class UserPost(UserPostIn):

    model\_config = ConfigDict(from\_attributes=True)

    id: int

    user\_id: int

class UserPostWithLikes(UserPost):

    model\_config = ConfigDict(from\_attributes=True)

    likes: int

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

class UserPostWithComments(BaseModel):

    post: UserPostWithLikes

    comments: list[UserComment]

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

In This Way We Can Select The Data From Posts, Comments, And Likes Table And Join Them.

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

@router.get("/{post\_id}/post-with-comments", response\_model=UserPostWithComments)

async def get\_post\_with\_comments(post\_id: int):

    # post = await find\_post(post\_id)

    query = select\_post\_and\_likes.where(posts\_table.c.id == post\_id)

    logger.debug(f"The Query For Getting Post, With Likes And Comments: {query}")

    post = await database.fetch\_one(query)

    if post is None:

        raise HTTPException(status\_code=404, detail=f"Post Not Found For Get Post With Its Comments With Id: {post\_id}")

    return {

        "post": post,

        "comments": await get\_post\_comments(post\_id=post\_id)

    }

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

In This Way, FastAPI Use Sorting As Query Parameter Because PostSorting Is Enum.

class PostSorting(str, Enum):

    new = "new"

    old = "old"

    most\_likes = "most\_likes"

@router.get("/", response\_model=list[UserPostWithLikes])

async def get\_all\_posts(sorting: PostSorting = PostSorting.new):

    # return post\_table.values()

    # OR We Can Use

    query = select\_post\_and\_likes.order\_by(sqlalchemy.desc("likes"))

    logger.debug(f"The Query For Get All Posts Is: {query}")

    return await database.fetch\_all(query)

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

If We Have Column Object Inside The Table, Then We Can Use Desc-Method Of It:

if sorting.new == PostSorting.new:

query = select\_post\_and\_likes.order\_by(posts\_table.c.id.desc())

elif sorting.old == PostSorting.old:

query = select\_post\_and\_likes.order\_by(posts\_table.c.id.asc())

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

If We Don’t Have Column Object, But We Know The Column Name, Then We Can use *sqlalchemy.desc("column-name-here")*

query = select\_post\_and\_likes.order\_by(sqlalchemy.desc("likes"))

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

If We Pass Any Other Value For Sorting (Right Values Is: New, Old, Most\_Likes), Then FastAPI Will Return Response With Status Code Is: 422



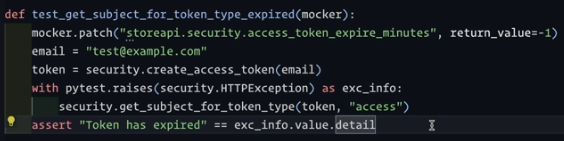
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create Multiple Tokens For Using In (Login, Confirmations, Roles, …etc) We Can Add type-Key To Data Of Token

**Note 1**: The *type: access* Means It is Used For Login

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

In This Way, We Can Getting The Details Of Exception That Raises From Test:



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

In This Way, We Can Build The URL For Specific Point:

**Note 1**: The *confirm\_user\_email* Is The Method Name For The Route.

return {

        "msg": "User Created Successfully",

        "id": result,

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

            "confirm\_user\_email",

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

        )

    }

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

@router.get("/confirm/{token}")

async def confirm\_user\_email(token: str):

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

    query = users\_table.update().where(users\_table.c.email == email).values(confirmed=True)

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

    await database.execute(query)

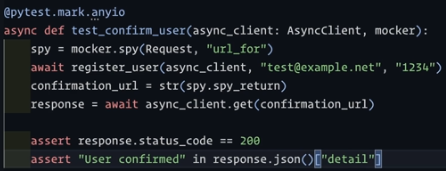
    return { "detail": "User Confirmed" }

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

In This Way We Can Spy On The Value Of Request-Class From FastAPI-Module

Here We Spy Only, Not Change The Value.

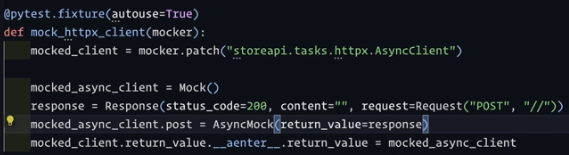
We Return The Value Using **spy\_return**



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







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

**Note 1**: Here We Must Set Empty Line Between ***From*** And The ***Body***

async def send\_simple\_email(to: str, subject: str, body: str, from\_: str):

    sender = f"Private Person <{from\_}>"

    receiver = f"A Test User <{to}>"

    message = f"""\

Subject: {subject}

To: {receiver}

From: {sender}

{body}

"""

    with smtp.SMTP(config.EMAIL\_HOST, config.EMAIL\_PORT) as server:

        server.ehlo()

        server.starttls()

        server.login(config.USERNAME, config.PASSWORD)

        result = server.sendmail(sender, receiver, message)

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