For Adding The Ability To Make Pydantic Read The Sqlalchemy Rows That Returns By databases-object, We Can Use ConfigDict-class From Pydantic With from\_attributes=True

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

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

If We Want To Select The Data Using Databases-Object:

**Note**: We Use where To Filter The Data, And We Use fetch\_one To Get Only The First Result

async def find\_post(post\_id: int):

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

    return await database.fetch\_one(query)

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

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

async def get\_all\_posts():

    # return post\_table.values()

    # OR We Can Use

    query = posts\_table.select()

    return await database.fetch\_all(query)

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

For Adding Rows Using Databases-Object:

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

async def create\_post(post: UserPostIn):

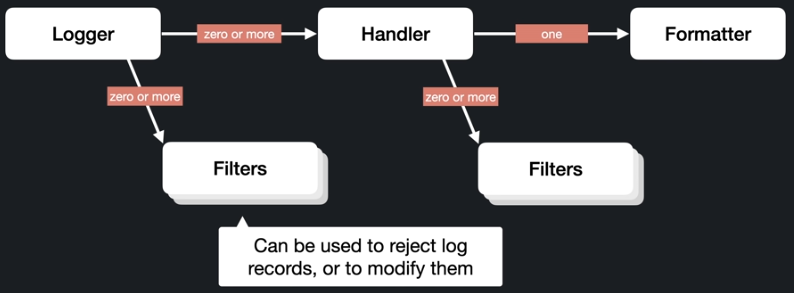
    data = post.model\_dump()

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

    last\_id = await database.execute(query)

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

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



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

For Adding Configuration To Our Project; For (Development, Testing, Production, ...etc.), Using Pydantic:

First We Install: **pip install pydantic-settings python-dotenv**

Then We Make Our .env File, And Exclude It From Repository, Ex:

ENV\_STATE=dev

DEV\_DATABASE\_URL=sqlite:///data.db

Then We Import The Important Modules:

from typing import Optional

from functools import lru\_cache # This For Caching The Results Of Function

# Depending On Parameters

from pydantic\_settings import BaseSettings, SettingsConfigDict

Then We Make The Class That Represent The Base For All Configuration With Their Parameters:

class BaseConfig(BaseSettings):

    ENV\_STATE: Optional[str] = None

    model\_config = SettingsConfigDict(env\_file='social\_network/.env', extra="ignore")

class GlobalConfig(BaseConfig):

    DATABASE\_URL: Optional[str] = None

    DB\_FORCE\_ROLL\_BACK: bool = False

Then We Create The Classes That ENV\_STATE Represent Them:

class DevConfig(GlobalConfig):

    model\_config = SettingsConfigDict(env\_prefix='DEV\_')

class ProdConfig(GlobalConfig):

    model\_config = SettingsConfigDict(env\_prefix='PROD\_')

class TestConfig(GlobalConfig):

    # In This Way We Override The Values In .env-File

    # For test-State

    DATABASE\_URL: Optional[str] = "sqlite:///test.db"

    DB\_FORCE\_ROLL\_BACK: bool = True

    model\_config = SettingsConfigDict(env\_prefix='TEST\_')

For Getting The Values Of Each Configuration In Easy Way:

@lru\_cache()

def get\_config(env\_state: str) -> GlobalConfig:

    configs = {"dev": DevConfig, "prod": ProdConfig, "test": TestConfig}

    if env\_state not in configs.keys():

        raise Exception(f"Invalid Value For env\_state Variable {env\_state}")

    return configs[env\_state]()

Then We Create The Configuration Variable For Our Project:

config = get\_config(BaseConfig().ENV\_STATE)

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

For Creating Logging Configuration For Our Project:

First, Import The Required Modules:

from logging.config import dictConfig

from social\_network.config import DevConfig, config

For Adding Better Formatting For Our *Console-Formatter*, We Install: *pip install rich*

For Adding Id For Each Request, So When We Log The Requests Of Different Users We Know The Related Ones, We Install: *pip install asgi-correlation-id*

Then We Create The Function That Configure Our Loggers With Inheritance For Other Files And Folders For Our Project:

def configure\_logging() -> None:

    dictConfig({

        "version": 1, # This Is For Using Specific Version Of Logging

        # Until Now, It is The Only Version

        "disable\_existing\_loggers": False,

        "filters": {

            "correlation\_id": {

                # In This Way We Reject Any Logging Msg

                # That Doesn't Contain CorrelationId-Value

                "()": "asgi\_correlation\_id.CorrelationIdFilter",

                # Here We Pass The Parameters To CorrelationIdFilter

                "uuid\_length": 8 if isinstance(config, DevConfig) else 32,

                "default\_value": "-"

            }

        },

        "formatters": {

            "console": {

                "class": "logging.Formatter",

                "datefmt": "%Y-%m-%dT%H:%M:%S",

                "format": "(%(correlation\_id)s) %(name)s:%(lineno)d - %(message)s"

            },

            "file": {

                "class": "logging.Formatter",

                "datefmt": "%Y-%m-%dT%H:%M:%S",

                "format": "%(asctime)s | %(levelname)-8s | (%(correlation\_id)s) %(name)s:%(lineno)d - %(message)s"

            },

        },

        "handlers": {

            "default": {

                "class": "rich.logging.RichHandler",

                "level": "DEBUG",

                "formatter": "console",

                "filters": ["correlation\_id"],

            },

            "rotating\_file": {

                "class": "logging.handlers.RotatingFileHandler",

                "level": "DEBUG",

                "formatter": "file",

                "filename": "j\_l\_social\_network.log",

                "maxBytes": 1024 \* 1024, # 1MB

                "backupCount": 2, # Only Save The Latest 2-Files Of Our Logs

                "encoding": "utf8",

                "filters": ["correlation\_id"],

            },

        },

        "loggers": {

            "social\_network": {

                "handlers": ["default", "rotating\_file"],

                "level": "DEBUG" if isinstance(config, DevConfig) else "INFO",

                "propagate": False, # This Will Prevent From Sending Logs To Parent

                # Note: The Main Parent For All Loggers Is Root

            },

            # In This Way We Override The Configuration Of

            # uvicorn, databases, aiosqlite-Modules

            # Note: Not All Logs Will Be Formatted

            "uvicorn": {

                "handlers": ["default", "rotating\_file"],

                "level": "INFO",

            },

            "databases": {

                "handlers": ["default", "rotating\_file"],

                "level": "INFO",

            },

            "aiosqlite": {

                "handlers": ["default", "rotating\_file"],

                "level": "INFO",

            }, }, })

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

To Use Our Configuration For Logging, In Our Project:

First, We Import The Required Functions And Modules:

Ex:

from contextlib import asynccontextmanager

from fastapi import FastAPI, HTTPException, Request

from fastapi.exception\_handlers import http\_exception\_handler

import logging

from asgi\_correlation\_id import CorrelationIdMiddleware

from social\_network.logging\_conf import configure\_logging

from social\_network.database import database

Then We Define The Name For Our Logger Based On Project Folder:

logger = logging.getLogger(\_\_name\_\_)

Then To Use CorrelationIdFilter, We Must Use CorrelationId Middleware:

app = FastAPI(lifespan=lifespan)

app.add\_middleware(CorrelationIdMiddleware)

Then We Can Use It In Our File:

@asynccontextmanager

async def lifespan(app: FastAPI):

    configure\_logging()

    await database.connect()

    logger.info("Database Connected Successfully")

    yield

    await database.disconnect()

For Better Handling Exceptions Depending On Type Of Exception OR Status Code:

@app.exception\_handler(HTTPException) # In This Way We Can Pass:

# The Exception Class OR The Status Code That We Want To Handle

# We Return Any Class That Inherit From Response-Class Like: JSONResponse

async def http\_exception\_handler\_logger(request: Request, exc: HTTPException):

    logger.error(f"HTTPException With Status Code: {exc.status\_code}, Details: {exc.detail}")

    return await http\_exception\_handler(request, exc)

For Checking The Way Of Our Exception Handler Function:

@router.get("/{post\_id}/comment", response\_model=list[UserComment])

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

    post = await find\_post(post\_id)

    if post is None:

        # logger.error(f"Post Not Found For Get Comments With Id: {post\_id}")

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

    query = comments\_table.select().where(comments\_table.c.post\_id == post\_id)

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

    return await database.fetch\_all(query)

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

To Make Our Custom Filter, That Hide The Sensitive Data From Records, Headers, Request Body, …etc

First, We Define The Class And inherit From logging.Filter

Second, We Define \_\_init\_\_-Method, And Give It Name As Parameter, And We Can Also Define Our Parameters.

Third, We Define The filter-Method, And Write The Logic Inside It:

* If We Return True, Then The Record Will Accepted.
* If We Return False, Then The Record Will Rejected.

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

import logging

def obfuscated(email: str, obfuscated\_length: int) -> str:

    characters = email[:obfuscated\_length]

    first, last = email.split('@')

    return characters + ('\*' \* (len(first) - obfuscated\_length)) + '@' + last

class EmailObfuscationFilter(logging.Filter):

    def \_\_init\_\_(self, name: str = "", obfuscated\_length: int = 2):

        super().\_\_init\_\_(name)

        self.obfuscated\_length = obfuscated\_length

    def filter(self, record: logging.LogRecord) -> bool:

        if "email" in record.\_\_dict\_\_:

            record.email = obfuscated(record.email, self.obfuscated\_length)

            # print("The Email Value Is: ", record.email) # For Debugging Only

        return True

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

Then We Add Our Filter To Filters-Dict Of dictConfig-Method:

"filters": {

            "correlation\_id": {

                # In This Way We Reject Any Logging Msg

                # That Doesn't Contain CorrelationId-Value

                "()": "asgi\_correlation\_id.CorrelationIdFilter",

                # Here We Pass The Parameters To CorrelationIdFilter

                "uuid\_length": 8 if isinstance(config, DevConfig) else 32,

                "default\_value": "-"

            },

            "email\_obfuscation": { # This Is The Name That We Want To Pass It

                # To The Init Method

                "()": EmailObfuscationFilter,

                "obfuscated\_length": 2

            },

        },

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

Then We Attach The Filter By Name To Handlers Filters’ Array:

"handlers": {

            "default": {

                "class": "rich.logging.RichHandler",

                "level": "DEBUG",

                "formatter": "console",

                "filters": ["correlation\_id", "email\_obfuscation"],

            },

            "rotating\_file": {

                "class": "logging.handlers.RotatingFileHandler",

                "level": "DEBUG",

                "formatter": "file",

                "filename": "j\_l\_social\_network.log",

                "maxBytes": 1024 \* 1024, # 1MB

                "backupCount": 2, # Only Save The Latest 2-Files Of Our Logs

                "encoding": "utf8",

                "filters": ["correlation\_id", "email\_obfuscation"],

            },

        },

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

**Note**: If We Want Our Specific Handler Like LogTail To Use Only In production Environment, Then We Can Define Our Function That Return Array Of Handlers Depending On The config-Object If It Is **ProdConfig** OR **DevConfig**

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

To Use Authentication In FastAPI: *pip install python-jose python-multipart passlib[bcrypt]*

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

The Modules Are For:

* *Module: python-jose*: For Using JWT With FastAPI.
* *Module: python-multipart*: For Handling Form-Data.
* *Module: passlib[bcrypt]*: For Hashing The Passwords.

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

To Use The Context Of Hashing From passlib[bcrypt], First We Import The Context:

from passlib.context import CryptContext

Then We Initialize It With Schemas As List Of Values:

pwd\_context = CryptContext(schemes=["bcrypt"])

Then We Define The Functions For Hashing, And Verifying The Passwords:

def get\_password\_hash(password: str):

    return pwd\_context.hash(password)

def verify\_password(password: str, hash: str):

    return pwd\_context.verify(password, hash)

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

To Create The JWT Authentication Token, First We Import jwt From *python-jose* Module:

from jose import jwt, ExpiredSignatureError, JWTError

Then We Create Functions To Create Token, Decode It, Verifying It:

def get\_token\_expire\_minutes():

    return config.EXPIRE\_MINUTES

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

def create\_access\_token(email: str) -> str:

    logger.debug("Creating Access Token", extra={"email": email})

expire = datetime.datetime.now(datetime.UTC) + datetime.timedelta(

        minutes=get\_token\_expire\_minutes(),

    )

    jwt\_data = {"sub": email, "exp": expire}

    encoded\_jwt = jwt.encode(jwt\_data, key=config.SECRET\_KEY, algorithm=config.ALGORITHM)

    return encoded\_jwt

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

async def authenticate\_user(email: str, password: str):

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

    if not user:

        raise credentials\_exception

    if not verify\_password(password, user.password):

        raise credentials\_exception

    return create\_access\_token(user.email)

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

credentials\_exception = HTTPException(

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

    detail="Couldn't Login",

    headers={

        "WWW-Authenticate": "Bearer",

    }

)

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

async def get\_current\_user(token: str):

    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 = 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

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