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)
```
****

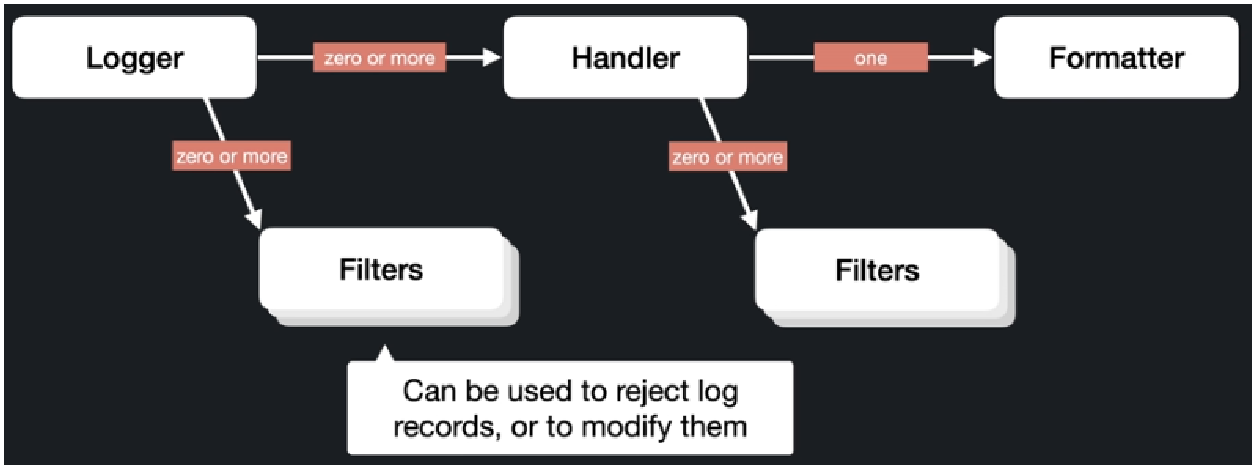
For Adding Rows Using Databases-Object:

```
@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:

```python
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:

```python
logger = logging.getLogger(__name__)
```

Then To Use CorrelationIdFilter, We Must Use CorrelationId Middleware:

```python
app = FastAPI(lifespan=lifespan)
app.add_middleware(CorrelationIdMiddleware)
```

Then We Can Use It In Our File:

```python
@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:

```python
@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
```
****