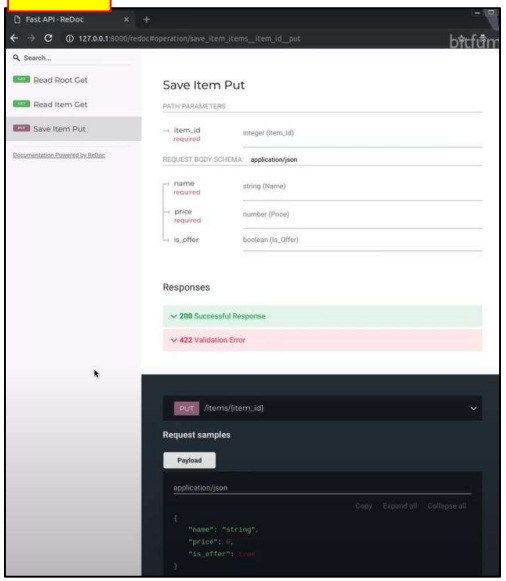
FAST API

Guizani Ahmed

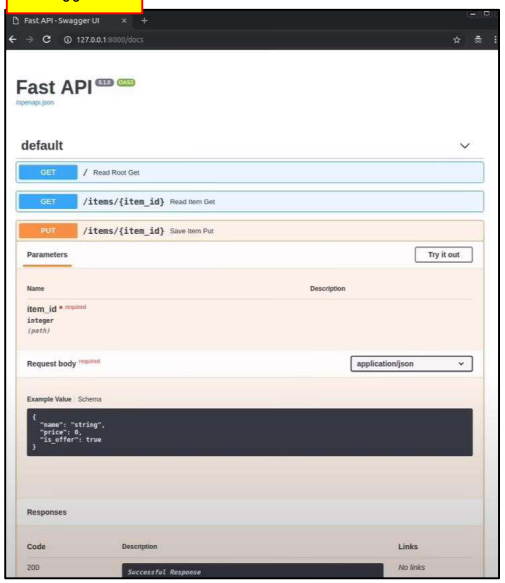
FAST API

- A super fast Python web framework: very modern and nice features of synchrounous programming which is still **lacking in django framework**.
- Fast API features:
  - **Automatic documentations:**
    - **swagger UI:** to check what are the routes we have created or api endpoints created or try them out (check what you are creating how it is going to respond).
    - **ReDoc:** same as swagger UI but more designed.
  - Uses python 3.6 and more: use also the pidentic library (provide type system like type hint variable such str or int...)

ReDoc



Swagger ui



FAST API

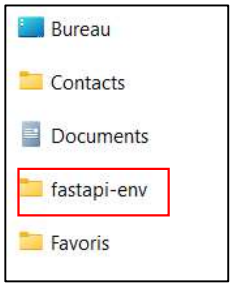
- Fast API features (continious):
  - **Totaly based in open API standards and json schema:**
    - **open API:** define how to create your api under linux foundation.
    - **Json schema:** by default return json which every api modern need to communicate with other things.
  - Code editors **auto-complete** in VScode or pycharm.
  - **Security and authentication** (http basic security, Oauth 2 with JWT token, API keys in header, query parameters, cookies).
  - **Dependancy injection unlimited plugins testing:** it uses **pytest** to test your applications.
  - **Starlette Features**: another framework for python:
    - Websocket support
    - Graphql support
    - In-process background tasks
    - Startup and shutdown events
    - Test client built on requests
    - CORS, gzip, Static files, Streaming responses
    - Session and Cookie support
  - Other Supports: SQL databases, nosql database.

INSTALL AND SETUP

- GitHub - tiangolo/fastapi: FastAPI framework, high performance, easy to learn, fast to code, ready for production
- FastAPI (tiangolo.com)

1- create virtual envirenment with python3 **venv**:

```
C:\Users\Ahmed>python -m venv fastapi-env
```



2- activate the venv:

```
C:\Users\Ahmed>fastapi-env\scripts\activate

(fastapi-env) C:\Users\Ahmed>
```

Rq (link for syntaxe: venv - Creation of virtual environments - Python 3.12.1 documentation)

3- Install fast api (installation dans le venv seulement):

```
(fastapi-env) C:\Users\Ahmed>pip3 install fastapi
Collecting fastapi
  Obtaining dependency information for fastapi from htt
pi-0.108.0-py3-none-any.whl.metadata
  Downloading fastapi-0.108.0-py3-none-any.whl.metadata
```

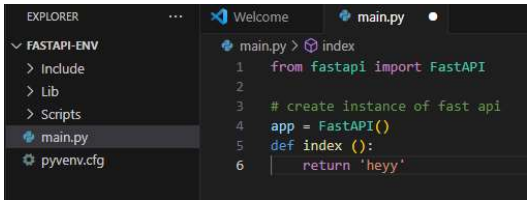
4- installation of ASGI server for production (Uvicorn or Hypercorn) only installed in virtual envirnment.

```
(fastapi-env) C:\Users\Ahmed>pip3 install uvicorn
Collecting uvicorn
  Downloading uvicorn-0.25.0-py3-none-any.whl.metadata (6.4 kB)
Collecting click>=7.0 (from uvicorn)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting h11>=0.8 (from uvicorn)
```

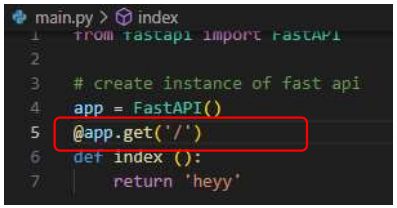
USES

5- in vscode after opening the folder fast api location: the activation of the venv run automatically:

6- now create the file main.py in vscode in the specific folder fastapi-env:



7- handle the path for the file to be executed from the server: @app.get ('/')
/: pour indiquer que defalt path is localhost



8- run the server in vs code terminal in the venv: uvicorn main:app --reload

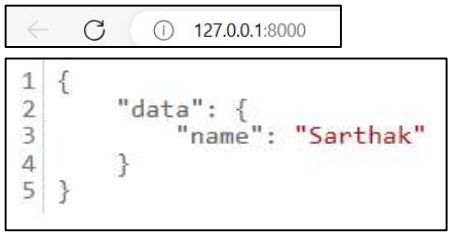
--reload: to tell the server to reload automatically when we change the file.

9- dans le navigateur 127.0.0.1:8000

USES

10- dectionnary use the return json format:

```python
# create instance of fast api
app = FastAPI()
# decorate
@app.get('/')
# function
def index ():
    #return 'comment va tu fast api'
    return  {'data': {'name': 'Sarthak'}}
```



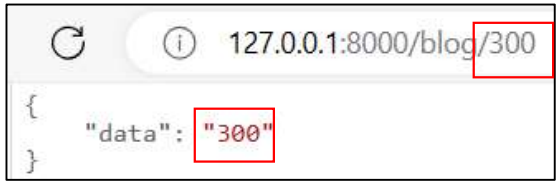
11- to add another path file to be use:

```python
@app.get('/about')
def about ():
    return {'data': {'about page'}}
```

12- initialise a Git repository: in terminal in venv

13- to use a blog you need the next syntaxe for routing easly with fastapi (Depending on the input of user, it will return the corresponding page to string as default)

```python
@app.get('/blog/{id}')
def about (id):
    return {'data': id}
```



USES

- 13- if the entred value is an integer you need to add the int

```python
# for specification of th
@app.get('/blog1/{id}')
def about (id : int):
    return {'data': id}
```

Case int

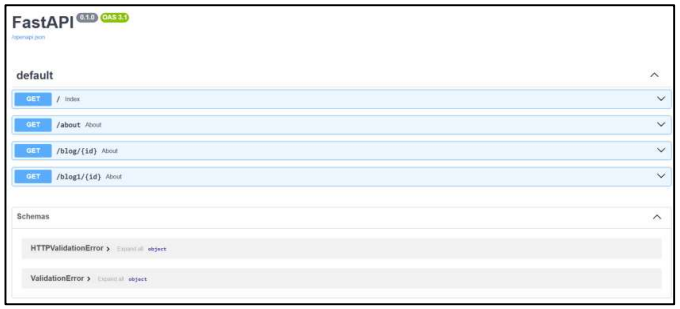
Case not int

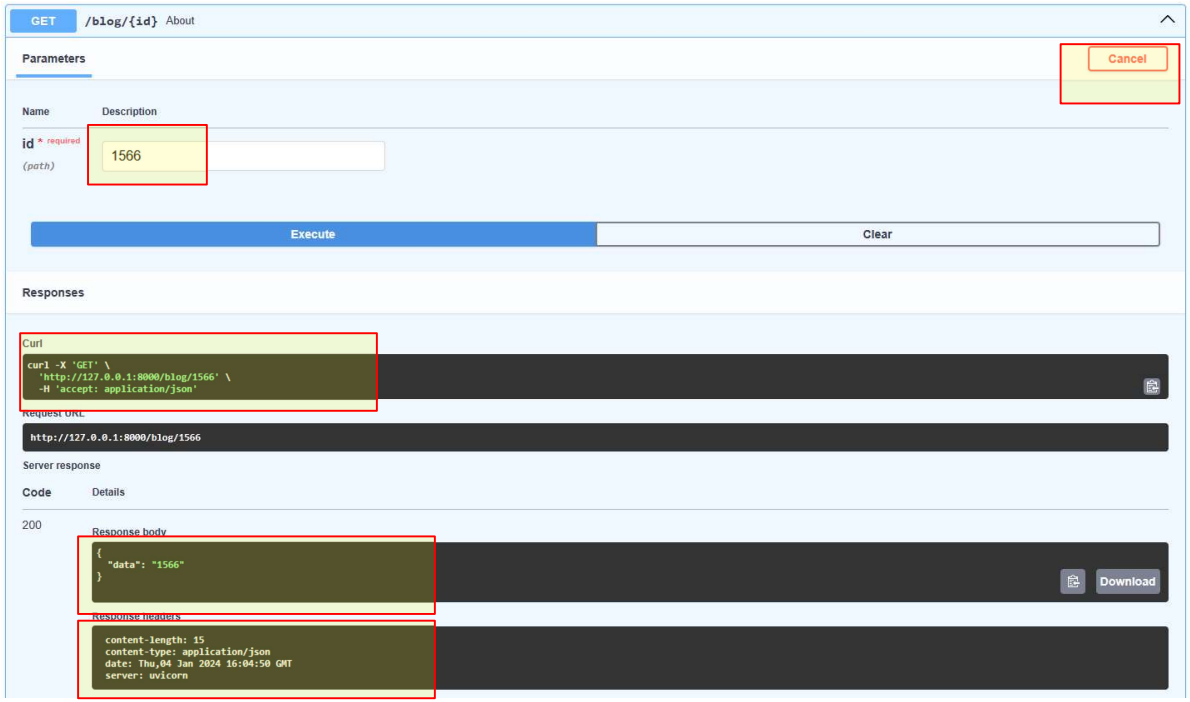
- 14- /blog/unpublished won't worked since dynamic rooting above which is {id} to correct it we need to to make all the function of unpulished above the dynamic {id}.

```python
@app.get('/blog/{id}')
def show(id: int):
    # fetch blog with id = id
    return {'data': id}


@app.get('/blog/unpublished')
def unpublished():
    return {'data': 'all unpublished blogs'}
```

API DOCS - SWAGGER/REDOCS

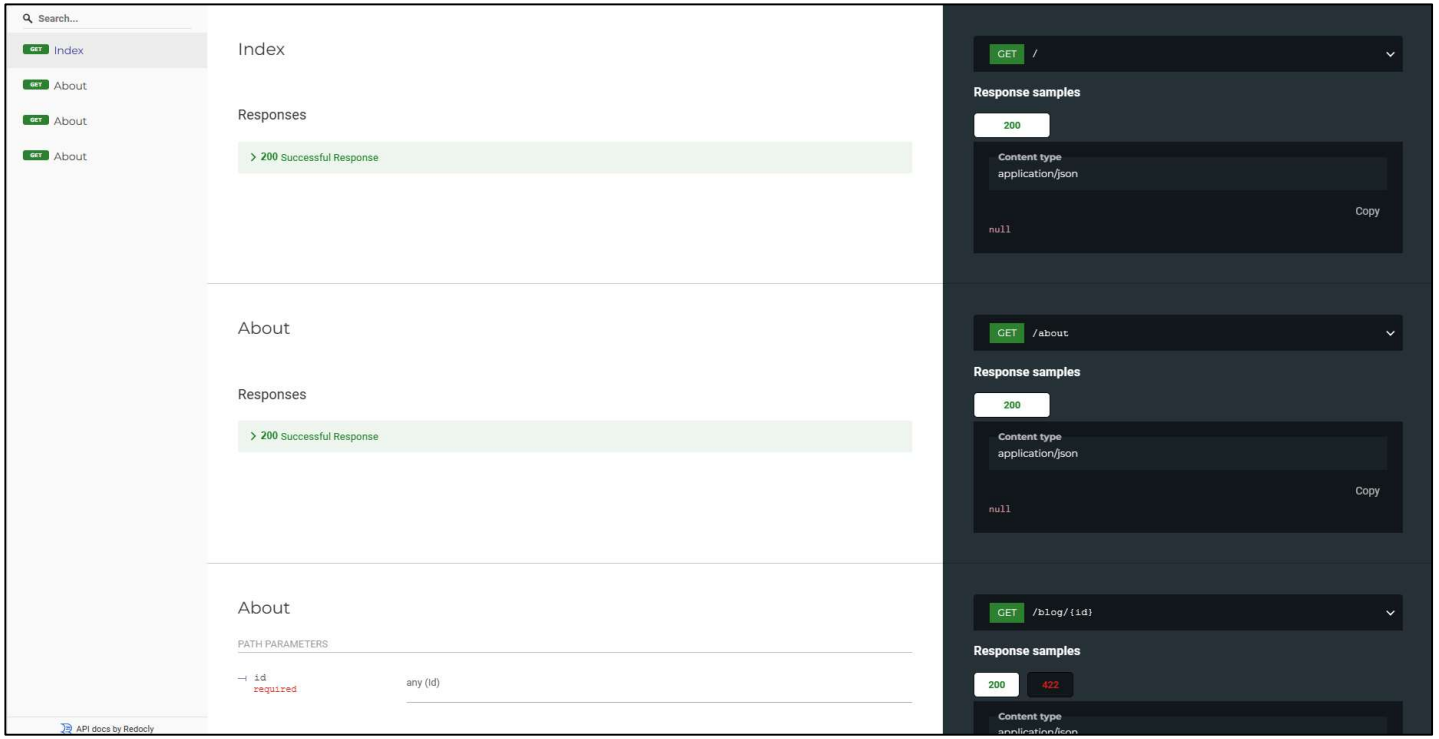
- To acces to the feature of documentations we need to put Localhost:8000/docs



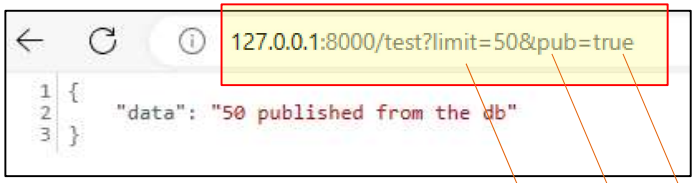


API DOCS - SWAGGER/REDOCS

- To acces to the feature of documentations we need to put Localhost:8000/redoc



QUERY PARAMETER




```python
@app.get('/test')
def test (limit, pub: bool):
    if pub == True:
        return {'data': f'{limit} published from the db'}
    else:
        return {'data': f'{limit} not existed from the db'}
```

```python
@app.get('/test')
def test (limit = 10, pub: bool = True):
    if pub == True:
        return {'data': f'{limit} published from the db'}
    else:
        return {'data': f'{limit} not existed from the db'}
```

For default value to be more pertinent when the user didn't enter the limit and the bool

QUERY PARAMETER

If it is not given by the user it will default to 10

```
def test (limit = 10, pub: bool = True, sort: Optional[str] = None):
```

```
from typing import Optional
```

To add optional value pour que l'utilisateur peut la entrer.

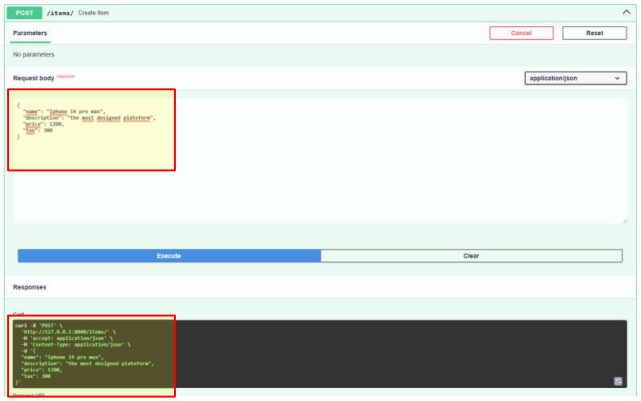
REQUEST BODY

- When you need to send data from a client (let's say, a browser) to your API, you send it as a request body.
- To declare a request body, you use Pydantic.

```
from pydantic import BaseModel
```

```
class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


@app.post("/items/")
async def create_item(item: Item):
    return item
```

http://127.0.0.1:8000/docs



HOW TO DEBUG

```python
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

To CHANGE THE PORT OF SERVER UVICORN

FastAPI doesn't require you to use a SQL (relational) database.

But you can use any relational database that you want.

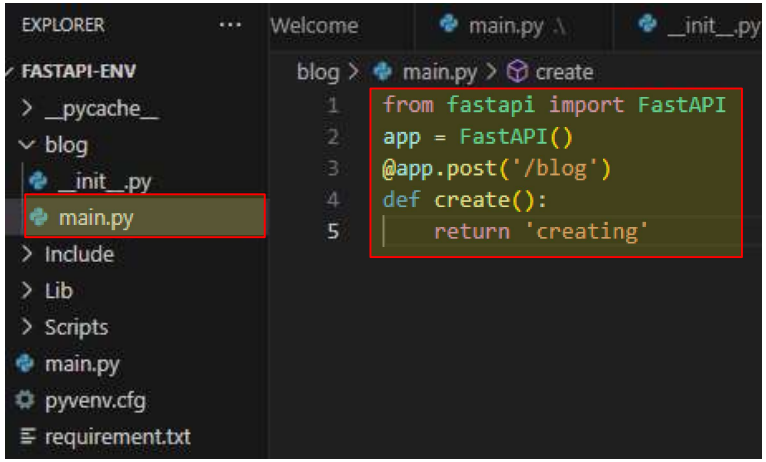
Here we'll see an example using SQLAlchemy [↪].

You can easily adapt it to any database supported by SQLAlchemy, like:

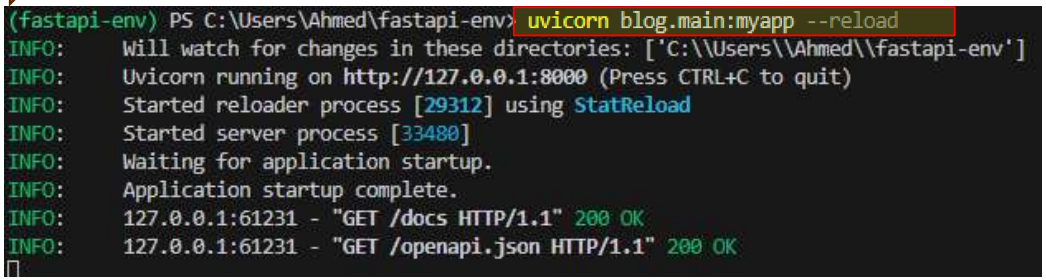
- PostgreSQL
- MySQL
- SQLite
- Oracle
- Microsoft SQL Server, etc.

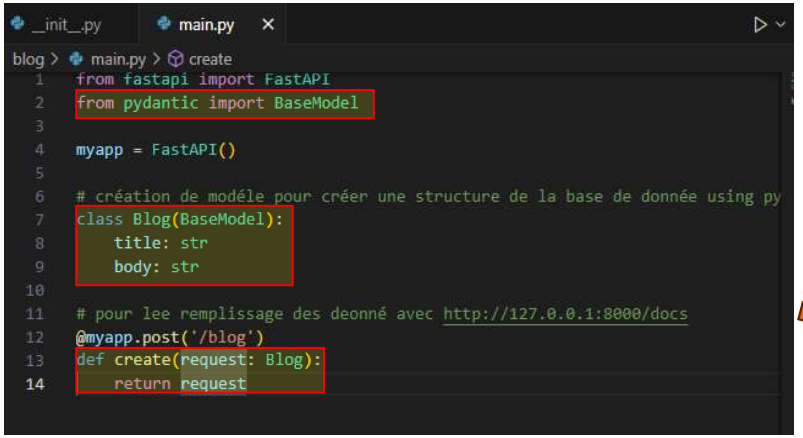
DATA BASE AND CONNECTIONS

Create the database



We need to put blog.main because the path is in a folder called blog



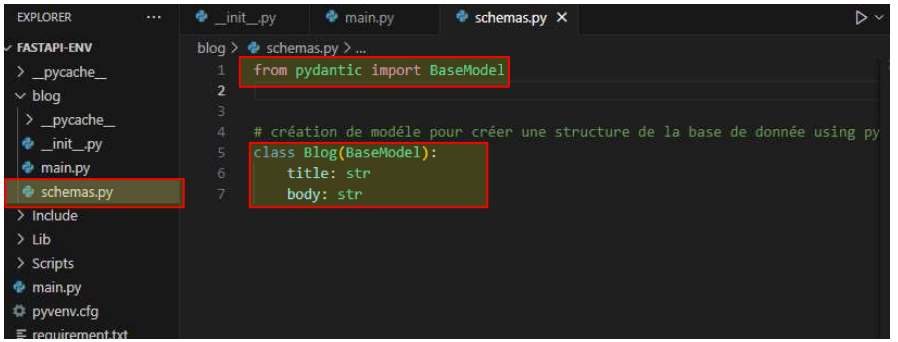


DATA BASE AND CONNECTIONS

To organize the giveaways:

1- Creation of diagrams. (for the model)

2- Call the main page template



Main.py

```python
from fastapi import FastAPI
from . import schemas

myapp = FastAPI()

# pour lee remplissage des deonné avec http://127.0.0.1:8000/docs
@myapp.post('/blog')
def create(request: schemas.Blog):
    return request
```

DATA BASE AND CONNECTIONS

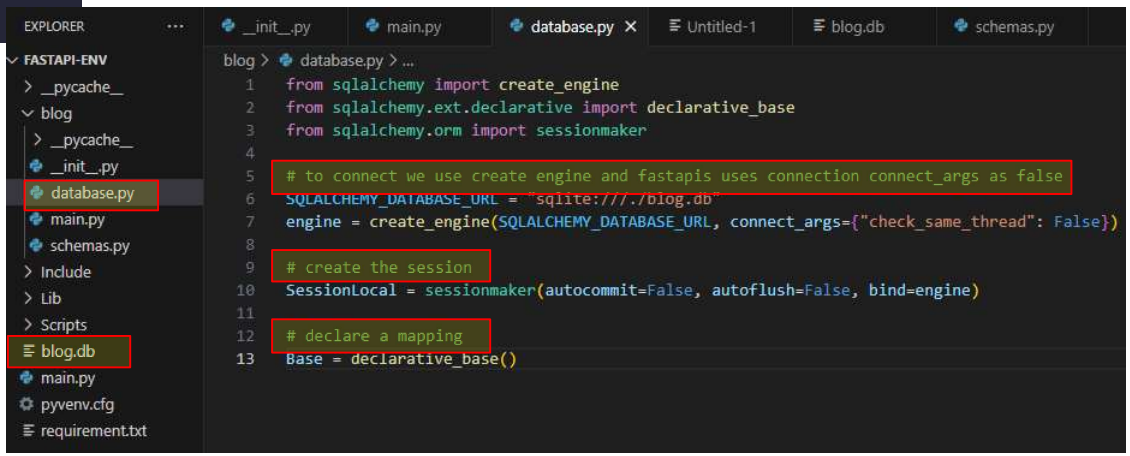
1- **FastAPI works with any database** and any style of library to talk to the database.
2- A **common pattern is to use an "ORM": an "object-relational mapping" library**.
3- An ORM has tools to **convert ("map") between objects in code and database tables** ("relations").
4- With an ORM, you normally create a class that represents a table in a SQL database, each attribute of the class represents a column, with a name and a type.

1- Install sqlalchemy

```
(fastapi-env) PS C:\Users\Ahmed\fastapi-env> pip install sqlalchemy
```

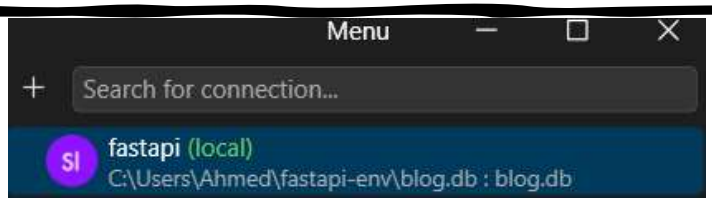
2- create blog.db under blog folder

3- create a database.py (where we're going to put the connections and map ORM)

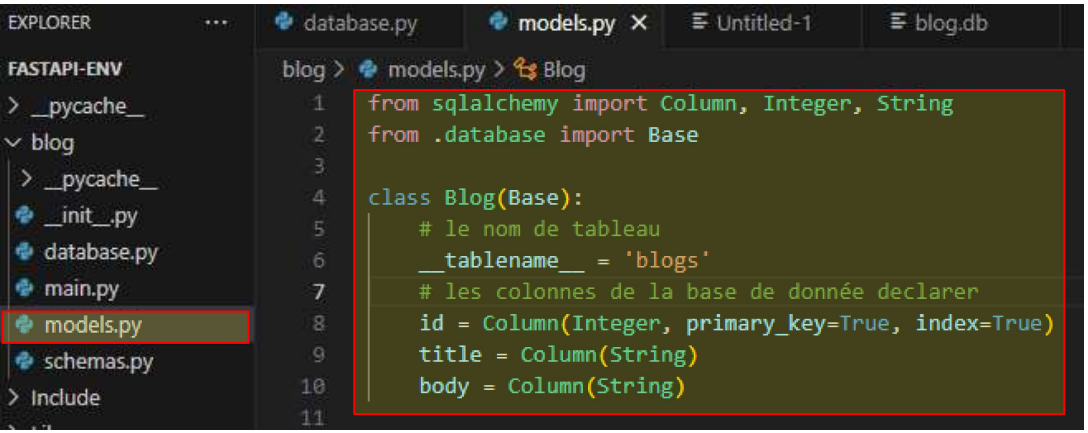


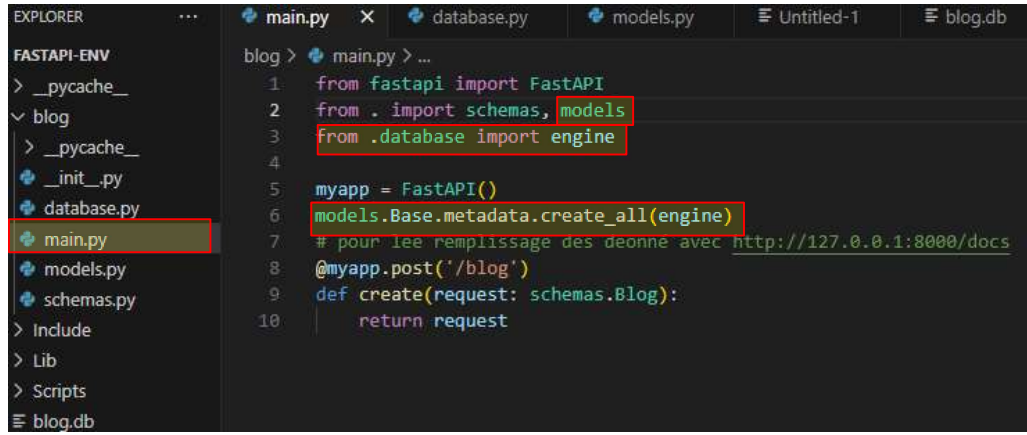
MODEL & TABLE

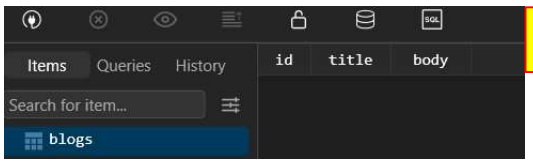
1- use of table plus to vizualise database and connect to blog database.



2- create schema of the data base which called models. File models.py and







results.py

STORE ELEMENT TO THE DATABASE WITH FASTAPI

- Next fastapi set will be used to add new contents. Main.py

```python
from fastapi import FastAPI, Depends
from . import schemas, models
from .database import engine, SessionLocal
from sqlalchemy.orm import Session

app = FastAPI()

models.Base.metadata.create_all(engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post('/blog')
def create(request: schemas.Blog, db: Session = Depends(get_db)):
    new_blog = models.Blog(title=request.title, body=request.body)
    db.add(new_blog)
    db.commit()
    db.refresh(new_blog)
    return new_blog
```

Database session use

For declaring variable db as a database

Use and set new arguments or user variable in the database

GET DATA FROM DABASE USING FASTAPI

- To get all blogs using fastapi we add the next code.

En main.py

```python
@myapp.get('/blog')
def all(db: Session=Depends(get_db)):
    blogs = db.query(models.Blog).all()
    return blogs
```

To say that run a query to return all the data in the db (.all)

For more exemples:
fastapi.tiangolo.com/tutorial/sql-databases/?h=sql#install-sqlalchemy

GET DATA FROM DABASE USING FASTAPI

- To get some specific result using fastapi we add the next code.

```python
@app.get('/blog/{id}')
def show(id, db: Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id).first()
    return blog
```

For the execution of database-specific commands to give according to identifiers

EXCEPTION & STATUS CODE

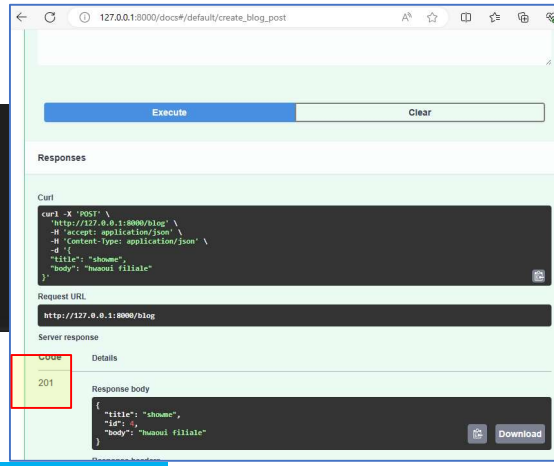
- When add a new data the respance code based on documentation shoud be 201

documentation

```python
from fastapi import FastAPI

app = FastAPI()


@app.post("/items/", status_code=201)
async def create_item(name: str):
    return {"name": name}
```

main.py

```python
@myapp.post('/blog', status_code=201)
def create(request: schemas.Blog, db: Session = Depends(get_db)):
    new_blog = models.Blog(title=request.title, body=request.body)
    db.add(new_blog)
    db.commit()
    db.refresh(new_blog)
    return new_blog
```



To avoid looking for code 201 .... and avoid waste of time the solutions is with Fastapi

```python
from fastapi import FastAPI, Depends, status
@myapp.post('/blog', status_code=status.HTTP_201_CREATED)
def create(request: schemas.Blog, db: Sessi
    new_blog = models.Blog(title=request.ti
    db.add(new_blog)
    db.commit()
    db.refresh(new_blog)
    return new_blog
```

```
HTTP_201_CREATED
WS_1000_NORMAL_CLOSURE
WS_1005_ABNORMAL_CLOSURE
```

EXCEPTION & STATUS CODE

- When we request some value that doesn't exist in the data base the message error generated will as follows

Main.py

```python
from fastapi import FastAPI, Depends, status, Response
@myapp.get('/blog/{id}', status_code=200)
def show(id, respance:Response, db : Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id).first()
    if not blog:
        respance.status_code = status.HTTP_404_NOT_FOUND
    return blog
```

- Adding this line the result will be more accurate:

```python
return {'detail': f'the blog with the id : {id} not existed yet'}
```

http://127.0.0.1:8000/docs

Code **Details**

404 *Undocumented*

Error: Not Found

Response body

```
{
  "detail": "the blog with the id : 10 not existed yet"
}
```

127.0.0.1:8000/blog/10

```
{
    "detail": "the blog with the id : 10 not existed yet"
}
```

EXCEPTION & STATUS CODE

- When we request some value that doesn't exist in the data base the message error generated will as follows 2nd method

Main.py

```
from fastapi import FastAPI, Depends, status, Response, HTTPException
```

```
@myapp.get('/blog/{id}', status_code=200)
def show(id, respance:Response, db : Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id).first()
    if not blog:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the blog with the id : {id} not existed yet')
        #respance.status_code = status.HTTP_404_NOT_FOUND
        #return {'detail': f'the blog with the id : {id} not existed yet'}
    return blog
```

DELETE A BLOG

Main.py

```python
@myapp.delete('/blog/{id}', status_code=status.HTTP_204_NO_CONTENT)
def destroy(id, db: Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id).delete(synchronize_session=False)
    db.commit()

    return 'done'
```

http://127001:8000/docs

UPDATE A BLOG (PUT)

Main.py

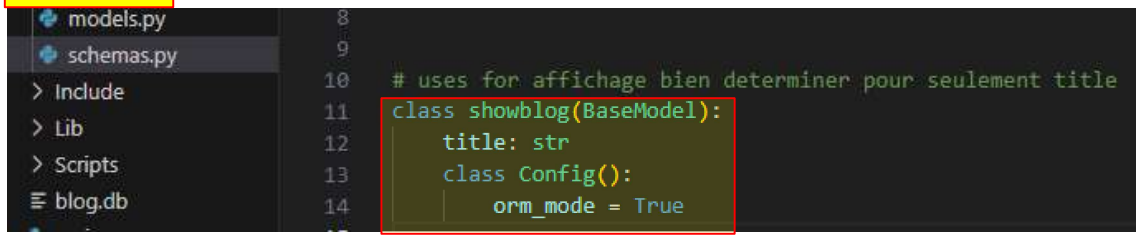
```python
@myapp.put('/blog/{id}', status_code=status.HTTP_202_ACCEPTED)
def updates(id, request: schemas.Blog, db : Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id)
    print (models.Blog.id)
    if not blog.first():
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail= f'the id: {id} not found')
    blog.update({models.Blog.title : request.title, models.Blog.body : request.body})
    db.commit()
    return 'updated'
```

- The schema is necessary for the blog content format.

http://127.0.0.1:8000/docs

RESPANSE MODEL: RESPANCE SCHEMA (PYNDANTIC MODEL)

Schemas.py




```python

# uses for affichage bien determiner pour seulement title
class showblog(BaseModel):
    title: str
    class Config():
        orm_mode = True
```

main.py

```python
# to get specific information from the database
@myapp.get('/blog/{id}', status_code=200, response_model=schemas.showblog)
def show(id, respance:Response, db : Session = Depends(get_db)):
    blog = db.query(models.Blog).filter(models.Blog.id == id).first()
    if not blog:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the blog with the id : {id} not existed yet')
        #respance.status_code = status.HTTP_404_NOT_FOUND
        #return {'detail': f'the blog with the id : {id} not existed yet'}
    return blog
```

result

CREATE USER

schemas.py

```
16  class User(BaseModel):
17      name: str
18      email: str
19      password: str
```

Main.py

```
@myapp.post('/user')
def create_user(request: schemas.User):
    return request
```

- Création de la table user

models.py

```
12  class User(Base):
13      __tablename__= 'users'
14
15      id = Column(Integer, primary_key=True, index=True)
16      name = Column(String)
17      email = Column(String)
18      password = Column(String)
```

Main.py

```
@myapp.post('/user')
def create_user(request: schemas.User, db : Session = Depends(get_db)):
    new_user = models.User(name=request.name, email=request.email, password=request.password)
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user
```

PASSWORD ENCRIPTION

- Install passlib bibleotheque

```
(fastapi-env) PS C:\Users\Ahmed\fastapi-env> pip install "passlib[bcrypt]"
```

Main.py

```python
from passlib.context import CryptContext
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

@myapp.post('/user')
def create_user(request: schemas.User, db : Session = Depends(get_db)):
    hashedpassword= pwd_context.hash(request.password)
    new_user = models.User(name=request.name, email=request.email, password=hashedpassword)
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user
```

http://127.0.0.1:8000/docs#

```
{
  "email": "Guizani",
  "id": 2,
  "name": "Ahmed",
  "password": "$2b$12$YOuJoBtRN1wUTUa.jn15tOp.zYPfcsgPnRVN4nfCiAD.0Wa2xNp4a"
}
```

sqlite

2	Ahmed	Guizani	$2b$12$YOuJoBtRN1wUTUa.jn15tO…

USER ID GET

Main.py

```python
@myapp.get('/user/{id}', status_code=200, response_model=schemas.showUser)
def show(id, db : Session = Depends(get_db)):
    user = db.query(models.User).filter(models.User.id == id).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the user with the id : {id} not existed')
        #respance.status_code = status.HTTP_404_NOT_FOUND
        #return {'detail': f'the blog with the id : {id} not existed yet'}
    return user
```

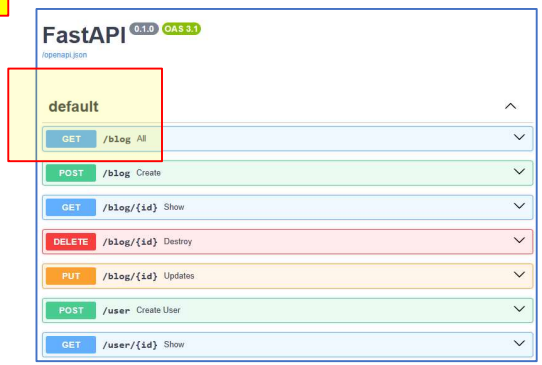
Shemas.py
Pour afficher que nom et email sans mdp

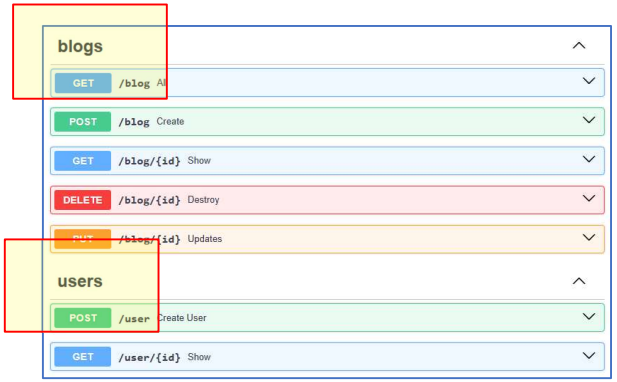
```python
class showUser(BaseModel):
    name: str
    email: str
```

http://127.0.0.1:8000/docs

USING DOCS TAGS

Main.py






```
@myapp.put('/blog/{id}', status_code=status.HTTP_202_ACCEPTED, tags=['blogs'])
```

```
@myapp.get('/user/{id}', status_code=200, response_model=schemas.showUser, tags=['users'])
```

RELATIONSHIP API (FOREIGN KEY)

Every blog should belong to user

models.py

```python
from sqlalchemy import Column, Integer, String,ForeignKey
from .database import Base
from sqlalchemy.orm import relationship

class Blog(Base):
    # le nom de tableau
    __tablename__ = 'blogs'
    # les colonnes de la base de donnée declarer
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String)
    body = Column(String)
    user_id = Column(Integer, ForeignKey("users.id"))
    creator = relationship("User", back_populates="blogs")

class User(Base):
    __tablename__= 'users'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    email = Column(String)
    password = Column(String)
    blogs = relationship('Blog', back_populates="creator")
```

Needs to delete table already created

main.py

```python
@myapp.post('/blog', status_code=status.HTTP_201_CREATED, tags=['blogs'])
def create(request: schemas.Blog, db: Session = Depends(get_db)):
    new_blog = models.Blog(title=request.title, body=request.body, user_id=1)
    db.add(new_blog)
    db.commit()
    db.refresh(new_blog)
```

http://127.0.0.1:8000/docs

201

Response body

```
{
  "body": "relationship is very important",
  "title": "iphone x",
  "id": 1,
  "user_id": 1
}
```

RELATIONSHIP API (FOREIGN KEY)

http://127.0.0.1:8000/docs

GET

200

Response body

```
{
  "title": "iphone x",
  "body": "relationship is very important"
}
```

Needs to show also relationships of the creator of the blog

Schemas.py

```
class showUser(BaseModel):
    name: str
    email: str
class showblog(BaseModel):
    title: str
    body: str
    creator: showUser
    class Config():
        orm_mode = True
```

In order for it to be called, showUser must be in front

GET

200

Response body

```
{
  "title": "iphone x",
  "body": "relationship is very important",
  "creator": {
    "name": "Ahmed",
    "email": "Ahmed@Whize.fr"
  }
}
```

RELATIONSHIP API (FOREIGN KEY)

Needs to show also the blogs for each user

Schemas.py

```python
class showUser(BaseModel):
    name: str
    email: str
    blogs : list[Blog] = []
    class Config():
        orm_mode = True
```

models.py

```python
class User(Base):
    __tablename__= 'users'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    email = Column(String)
    password = Column(String)
    blogs = relationship('Blog', back_populates="creator")
```

In order for the user display to be complete, need to call list

http://127.0.0.1:8000/docs

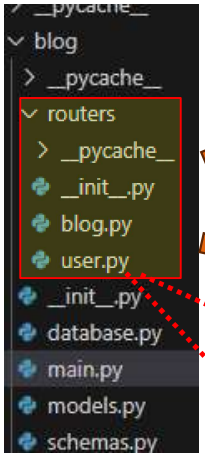
GET

200

Response body


```
{
  "name": "Ahmed",
  "email": "Ahmed@Whize.fr",
  "blogs": [
    {
      "title": "iphone x",
      "body": "relationship is very important"
    },
    {
      "title": "iphone 13 pro max",
      "body": "i wrote for this phone because of their composition and cameras"
    }
  ]
}
```

API ROUTER/ FOR BIGGER APPLICATIONS REFACTORING



database.py

```python
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

main.py

```python
from fastapi import FastAPI
from . import models
from .database import engine
from .routers import blog, user

myapp = FastAPI()
models.Base.metadata.create_all(engine)

# necessaire pour tout ce qui est affichage en swagerui
myapp.include_router(blog.router)
myapp.include_router(user.router)
```

user.py

```python
from fastapi import APIRouter, Depends, HTTPException, status
from .. import models, schemas
from ..database import get_db
from sqlalchemy.orm import Session
from passlib.context import CryptContext

router = APIRouter()
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

@router.post('/user', tags=['users'])
def create_user(request: schemas.User, db : Session = Depends(get_db)):
    hashedpassword= pwd_context.hash(request.password)
    new_user = models.User(name=request.name, email=request.email, password=hashedpass
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user

@router.get('/user/{id}', status_code=200, response_model=schemas.showUser, tags=['use
def show(id, db : Session = Depends(get_db)):
    user = db.query(models.User).filter(models.User.id == id).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the user with the id : {id} not existed')
    return user
```

Same for blog.py

API ROUTER OPERATOR

Blog.py

```python
@router.get('/blog', tags=['blogs'])
def all(db: Session=Depends(get_db)):
    blogs = db.query(models.Blog).all()
    return blogs
```

To avoid redendance of /blod or tags

Blog.py

```python
router = APIRouter(
    prefix='/blog',
    tags=['BLogs']
)
```

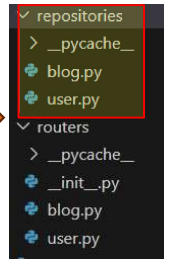
```python
@router.get('/')
```

Same for user.py

BLOG AND USER REPOSITORIES

Routers must be very well organized to differentiate that this directory is only for .py routing

```python
from fastapi import APIRouter, Depends, HTTPException, status
from .. import models, schemas
from ..database import get_db
from sqlalchemy.orm import Session
from passlib.context import CryptContext

router = APIRouter(
    prefix='/user',
    tags=['users']
)
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

@router.post('/')
def create_user(request: schemas.User, db : Session = Depends(get_db)):
    hashedpassword= pwd_context.hash(request.password)
    new_user = models.User(name=request.name, email=request.email, password=hashedpass
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user

@router.get('/{id}', status_code=200, response_model=schemas.showUser)
def show(id, db : Session = Depends(get_db)):
    user = db.query(models.User).filter(models.User.id == id).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the user with the id : {id} not existed')
    return user
```




```python
from fastapi import APIRouter, Depends, HTTPException, status
from .. import models, schemas
from ..database import get_db
from sqlalchemy.orm import Session
from passlib.context import CryptContext
from ..repositories import user

router = APIRouter(
    prefix='/user',
    tags=['users']
)
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

@router.post('/')
def create_user(request: schemas.User, db : Session = Depends(get_db)):
    return user.create_user(request, db)

@router.get('/{id}', status_code=200, response_model=schemas.showUser)
def show(id: int, db : Session = Depends(get_db)):
    return user.show(id, db)
```

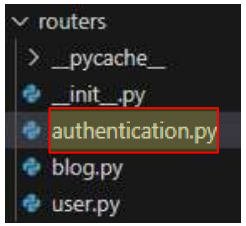
routers/user.py

Same for blog

```python
from fastapi import HTTPException, status
from .. import models, schemas
from sqlalchemy.orm import Session
from passlib.context import CryptContext


pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def create_user(request: schemas.User, db : Session):
    hashedpassword= pwd_context.hash(request.password)
    new_user = models.User(name=request.name, email=request.email, password=hashedpass
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user

def show(id: int, db : Session):
    user = db.query(models.User).filter(models.User.id == id).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'the user with the id : {id} not existed')
    return user
```

Repositories/user.py

LOGIN AND VERIFY PASSWORD



1- Creating an AUTH file in Routiers Repo

authentification.py

```python
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from .. import schemas, database, models, Hashing

router = APIRouter(
    prefix='/login',
    tags=['Login'])

@router.post('/')
def login(request: schemas.Login, db : Session = Depends(database.get_db)):
    user = db.query(models.User).filter(models.User.email == request.username).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'invalid credential')
    # verifying the password
    if not Hashing.Hash.verify(user.password, request.password):
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'Incorrect password')
    # generate the jwt token and return it
    return user
```

Hashing.py
(inside blog)

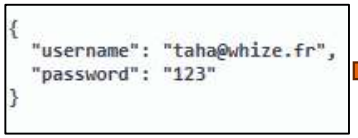
```python
class Hash():
    def bcrypt(password: str):
        return pwd_context.hash(password)
    def verify(hashedpwd, plainpwd):
        return pwd_context.verify(plainpwd, hashedpwd)
```

Main.py

```python
from .routers import authentication
myapp.include_router(authentication.router)
```

2- Code creation for the login file

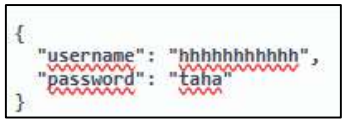
LOGIN AND VERIFY PASSWORD

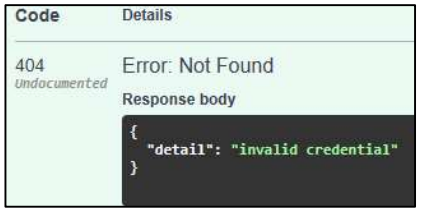


✗

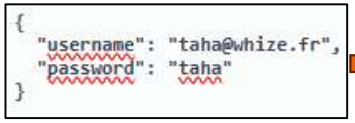
Le code

```python
@router.post('/')
def login(request: schemas.Login, db : Session = Depends(database.get_db)):
    user = db.query(models.User).filter(models.User.email == request.username).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'invalid credential')
    # verifying the password
    if not Hashing.Hash.verify(user.password, request.password):
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'Incorrect password')
    # generate the jwt token and return it
    return user
```





✗



✓

JWT ACCESS TOKEN (FASTAPI OAUTH2 WITH PASSWORD (AND HASHING), BEARER WITH JWT TOKENS)

fastapi.tiangolo.com/tutorial/security/oauth2-jwt/?h=oauth

1- install a librerie python-jose

```
(fastapi-env) PS C:\Users\Ahmed\fastapi-env> pip install python-jose
```

2- create file token (for JWT)

authentication.py 1 | token.py | schemas.py | Hashing.py | main.py

blog > token.py > create_access_token

```python
from datetime import datetime, timedelta, timezone
from jose import JWTError, jwt

SECRET_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30


def create_access_token(data: dict, expires_delta: timedelta | None = None):
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTE
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

3- add schemas basemodel

```python
class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    username: str | None = None
```

4- add the generate jwt token

```python
# generate the jwt token and return it
access_token = token.create_access_token(data={"sub": user.email})
return {"access_token": access_token, "token_type":"bearer"}
```


```
{
  "username": "taha@whize.fr",
  "password": "taha"
}
```

Code | Details

200

Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0YWhhQHdoaXplLmZyIiwiZXh
wIjoxNzA2MTA1MjYzfQ.tGcx2FeZLj-Ay7PCJki45_tBlnRLk48nQ8rAOGzfUgg",
  "token_type": "bearer"
}
```

Download

ROUTES BEHIND AUTHENTICATION

fastapi.tiangolo.com/tutorial/security/oauth2-jwt/?h=oauth
FastAPI - A python framework | Full Course (youtube.com)
SQL (Relational) Databases - FastAPI (tiangolo.com)

1- create file ouath2.py (we will use JWT)

```python
from fastapi import Depends, HTTPException, status
from typing import Annotated
from fastapi.security import OAuth2PasswordBearer
from . import token

# ROUTE OR URL FROM WHERE FASTAPI WILL FETCH THE TOKEN
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    return token.verify_token(token, credentials_exception)
```

2- add to token.py verify_token function

```python
def verify_token(token:str, credentials_exception):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if email is None:
            raise credentials_exception
        token_data = schemas.TokenData(email=email)
    except JWTError:
        raise credentials_exception
```

3- add to the blog verification of token

```python
@router.get('/')
def all(db: Session=Depends(get_db), get_current_user: schemas.User = Depends(ouath2.get_current_user)):
    return blog.all(db)
```

4- mofication of the schemas

```python
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    email: str | None = None
```

5- result