**DOCUMENTATION TECHNIQUE DU PROJET**

**THEME DU PROJET :** *API Rest & MicroServices «***BookIT***»*

**REALISATEUR :** *BOUGMA ABOUBAKARY*

**DESCRIPTION :**

Un projet permettant de de reconstruire une version allégée de la célèbre bibliothèque anglaise de leur ERP en Python avec FastAPI avec un client développé en Python.

**Les étapes de réalisation du projet :**

* Création du projet BookIT
* Création d’une base de données MySQL

BACK-END

* Création du fichier « Back.py » qui permettra de gérer la partie back-end :
* Création de la fonction permettant de créer un livre « save\_book »

def save\_book (name: str, release\_date: str, author\_name: str, number\_of\_pages: int):

    try:

        today = datetime.now().date()

        release\_date = datetime.strptime(release\_date, "%Y-%m-%d").date()

        if release\_date>today:

            raise ValueError("Release date is not AFTER TODAY")

        if number\_of\_pages < 2:

            raise ValueError("The number of pages is greater than 2")

        std = session.query(Book).filter\_by(name = name).first()

        if std:

            raise ValueError("The name of book is already use")

        new\_book = Book(name = name, release\_date= release\_date, author\_name= author\_name, number\_of\_pages = int(number\_of\_pages))

        session.add(new\_book)

        session.commit()

        return new\_book.id

    except ValueError as erreur:

        session.rollback()

        raise erreur

    except Exception as erreur:

        session.rollback()

        raise erreur

* Création de la fonction permettant de créer un utilisateur « create\_user »

def create\_user(pseudo: str, password: str):

    try:

        new\_user = user(pseudo = pseudo, password = password)

        session.add(new\_user)

        session.commit()

        return {"pseudo": pseudo}

    except Exception as error :

        session.rollback()

        raise HTTPException(status\_code=400, detail="Error") from error

* Création des différentes méthodes de l’API
* Route permettant de creer un utilisateur
* @app.post("/users", status\_code=201)
* async def save\_user(user: UserIn):
* result = create\_user(user.pseudo, user.password)
* return {"pseudo": user.pseudo}

# Route permettant de creer un nouveau livre

@app.post("/books/", status\_code=201)

async def create\_book(new\_book: NewBookInput):

    try:

        book\_id = save\_book(new\_book.name, new\_book.release\_date, new\_book.author\_name, new\_book.number\_of\_pages)

        return {"The book Id is": book\_id}

    except ValueError as erreur:

        raise HTTPException(status\_code=400, detail=str(erreur))

# Route pour editer un livre

@app.put("/books/{id\_book}", status\_code=200)

async def update\_book(id\_book : int, new\_book: NewBookInput):

    std = session.query(Book).filter\_by(id=id\_book).first()

    if std == None:

        raise HTTPException(status\_code=400, detail="The book not found")

    std.name = new\_book.name

    std.release\_date = new\_book.release\_date

    std.author\_name = new\_book.author\_name

    std.number\_of\_pages = new\_book.number\_of\_pages

    session.commit()

    return {"Reponse" : " The book has been updated"}

# Route pour supprimer un livre

@app.delete("/books/{id\_book}", status\_code=200)

async def delete\_book(id\_book: int):

    std = session.query(Book).filter\_by(id=id\_book).first()

    if std is None:

        raise HTTPException(status\_code=400, detail="The book was not found")

    session.delete(std)

    session.commit()

    return {"message": "The book has been deleted"}

# Route pour lister tous les livres

@app.get("/books", status\_code=200)

async def list\_books():

    result = session.query(Book).all()

    New\_list = []

    for std in result:

        New\_list.append({"Name" : std.name, "Release date" : std.release\_date, "Author name": std.author\_name, "Number of pages" : std.number\_of\_pages})

    return {"Reponse" : New\_list}

# Route pour obtenir un livre

@app.get("/books/{id\_book}", status\_code=200)

async def get\_book(id\_book : int):

    std = session.query(Book).filter\_by(id = id\_book).first()

    if std is None:

        raise HTTPException(status\_code=404, detail="The book not found")

    return{"Name" : std.name, "Release date" : std.release\_date, "Author name": std.author\_name, "Number of pages" : std.number\_of\_pages}

* L’API écoute sur le port TCP 8000
* if \_\_name\_\_ == '\_\_main\_\_':
* import uvicorn
* uvicorn.run(app, host="0.0.0.0", port=8000)

FRONT-END

* Développement d’une application python qui exécutera une requête sur l’API en écoutant  
  sur http://127.0.0.1:8000.
* Implémentation d’un menu permettant à l’utilisateur de faire un choix a partir d’une fonction « main »
* def main():
* while True:
* print("----------Menu--------")
* print("1. Listing the books")
* print("2. Getting a book")
* print("3. Creating a book")
* print("4. Deleting a book")
* print("5. Quit")
* choice = input("Enter your choice: ")
* if choice == "1":
* create\_user()
* list\_books()
* elif choice == "2":
* create\_user()
* get\_book()
* elif choice == "3":
* create\_book()
* elif choice == "4":
* delete\_book()
* elif choice == "5":
* break
* else:
* print("Invalid choice")
* Création des différentes fonctions permettant de lier notre Back-end a notre Front-end
* def create\_user():
* name = input("Enter your name : ")
* password = getpass.getpass("Enter your password : ")
* response = requests.post("http://127.0.0.1:8000/users", json={"pseudo" : name, "password": password})
* if response.status\_code == 201:
* print("Name : ",name)
* else:
* print("The name already exists")

def create\_book():

    name = input("Enter name: ")

    release\_date = input("Enter release date: ")

    author\_name = input("Enter author name: ")

    number\_of\_pages = input("Enter number of pages: ")

    response = requests.post("http://127.0.0.1:8000/books/", json={"name": name, "release\_date": release\_date, " author\_name": author\_name, "number\_of\_pages": number\_of\_pages})

    if response.status\_code == 201:

        print("Book created successfully")

    else:

        print("Error")

def list\_books():

    response = requests.get("http://127.0.0.1:8000/books")

    if response.status\_code == 200:

        books = response.json()["Reponse"]

        for book in books:

            bookInf = f"""

            Name: {book['Name']}

            Author: {book['Author name']}

            Date: {book['Release date']}

            Number of pages: {book['Number of pages']}

            """

            print(bookInf)

    else:

        print("Error")

def get\_book():

    id\_book = input("Enter book Id: ")

    response = requests.get(f"http://127.0.0.1:8000/books/{id\_book}")

    if response.status\_code == 200:

        book = response.json()

        bookInf = f"""

        Name: {book['Name']}

        Author: {book['Author name']}

        Date: {book['Release date']}

        Number of pages: {book['Number of pages']}

        """

        print(bookInf)

    else:

        print("Error")

def delete\_book():

    id\_book = input("Enter book Id: ")

    response = requests.delete(f"http://127.0.0.1:8000/books/{id\_book}")

    if response.status\_code == 200:

        print("Book deleted")

    else:

        print("Error")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

BONUS

* Masqué le mot de passe de l’utilisateur lors de sa saisie

import getpass

 password = getpass.getpass("Enter your password : ")