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)
```

```python
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"}
```

```python
# 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"}
```

```python
# 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}
```

```python
# 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")
```

```python
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")
```

```python
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")
```

```python
if __name__ == "__main__":
    main()
```

BONUS

- Masqué le mot de passe de l'utilisateur lors de sa saisie

```python
import getpass
 password = getpass.getpass("Enter your password : ")
```