TUGAS BESAR TAHAP - 5 IMPLEMENTASI LANJUTAN *BOOK WISE*
TEKNOLOGI SISTEM TERINTEGRASI

Mata Kuliah : II3160 Teknologi Sistem Terintegrasi
Dosen : Daniel Wiyogo Dwiputro, S.T., M.T.

Disiapkan oleh
Nama : Laras Hati Mahendra
NIM : 18223118

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
FAKULTAS SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

Daftar Isi

Daftar Tabel

1. Overview

Pada tahapan sebelumnya, sistem **BookWise** telah memiliki pondasi arsitektur domain, dimulai dari *capability business, bounded context,* hingga perancangan model yang membentuk *agregat Loan* sebagai pusat pengelolaan proses peminjaman buku digital. Implementasi awal Milestone 4 diterjemahkan rancangan tersebut sehingga menjadi struktur yang utuh.

Pada Milestone 5, berfokus pada mekanisme autentikasi sederhana agar API bisa lebih jelas dan bisa diakses oleh pengguna yang mempunyai izin. Mekanisme autentikasi ini menggunakan *JWT(JSON Web Token)*, tujuannya untuk membuat token saat *login* dan validasi token. Seluruh endpoint terkait *Loan* akan dilindungi setelah pengguna *login*.

2. Tujuan Implementasi Lanjutan

Tabel 2.1 Tabel Tujuan

No	Tujuan
1.	Menambahkan autentikasi sederhana dengan menggunakan JWT
2.	Membuat *endpoint* login/logout
3.	Menambahkan *middleware protection* pada API tertentu
4.	Memperbarui struktur folder dengan komponen keamanan
5.	Menguji *endpoint* melalui Swagger UI

3. Penjelasan JWT Sederhana

Tabel 3.1. Implementasi BookWise

No.	Penjelasan
1	JWT SEDERHANA
	JWT (JSON Web Token) adalah token berbentuk string yang mengklaim pengguna, ditandatangani menggunakan *secret key* tertentu. Setelah pengguna melakukan login, sistem membuat token yang berisi informasi dasar seperti userId. Token ini dikirimkan pada setiap request melalui header *Authorization*. Struktur JWT terdiri dari tiga bagian utama, yang dipisahkan oleh titik(.): ● ***Header*** → menentukan tipe token dan algoritma signing yang digunakan ● ***Payload*** → berisi data pengguna. ● ***Signature*** → hasil hashing yang memastikan token tidak dimodifikasi Dengan mekanisme ini, server tidak menyimpan *session* karena verifikasi dilakukan langsung.
2	*Flow Login*

1. User mengirim *username* & *password* ke endpoint **POST /auth/login.** Sesuaikan pada database yang telah disusun pada kode user.py.
2. Sistem memverifikasi kecocokan *username* dan *password* sederhana dengan hash.
3. Jika valid → sistem membuat JWT dan mengembalikannya kepada pengguna/peminjam.
4. Pengguna mengirim token untuk setiap permintaan dengan format: *Authorization*: Bearer <token>
5. Server memverifikasi token pada setiap request ke *endpoint* yang dilindungi
6. Jika token *valid* → akses diberikan
7. Jika token tidak *valid* atau hilang → server mengembalikan 401 Unauthorized

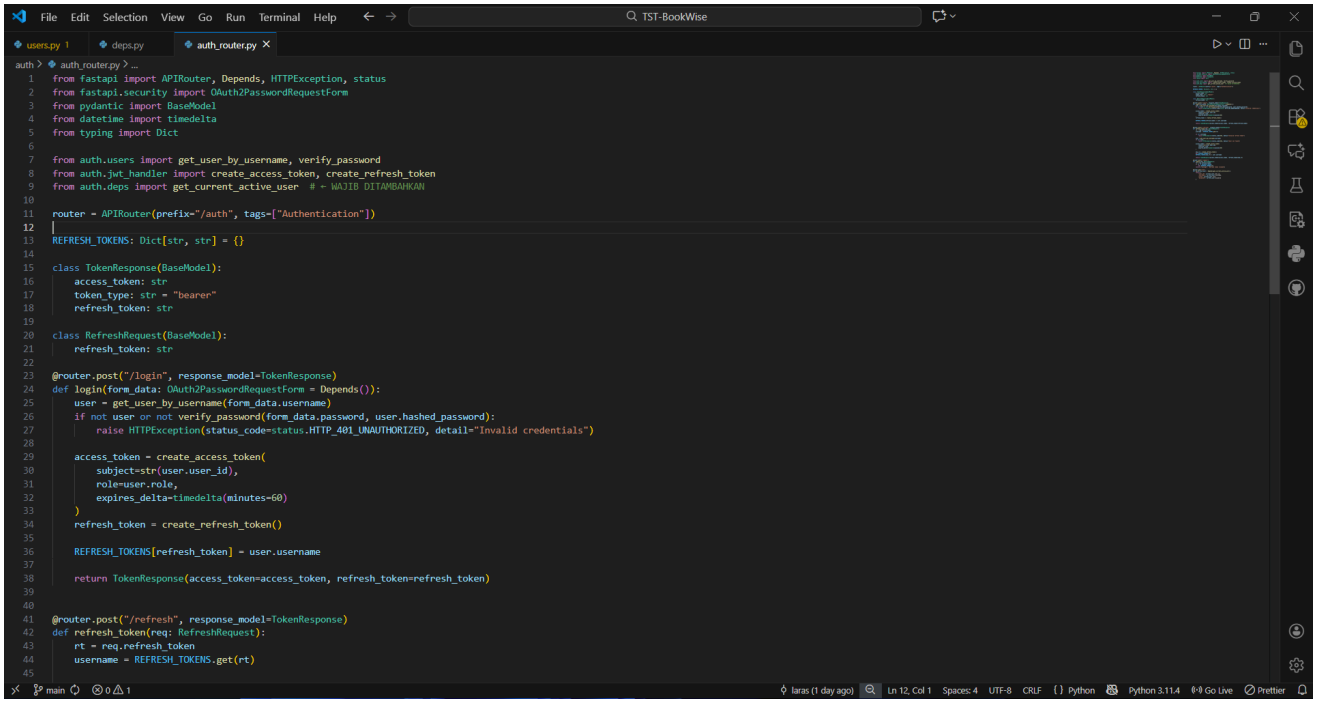
3	Penambahan Folder Implementasi

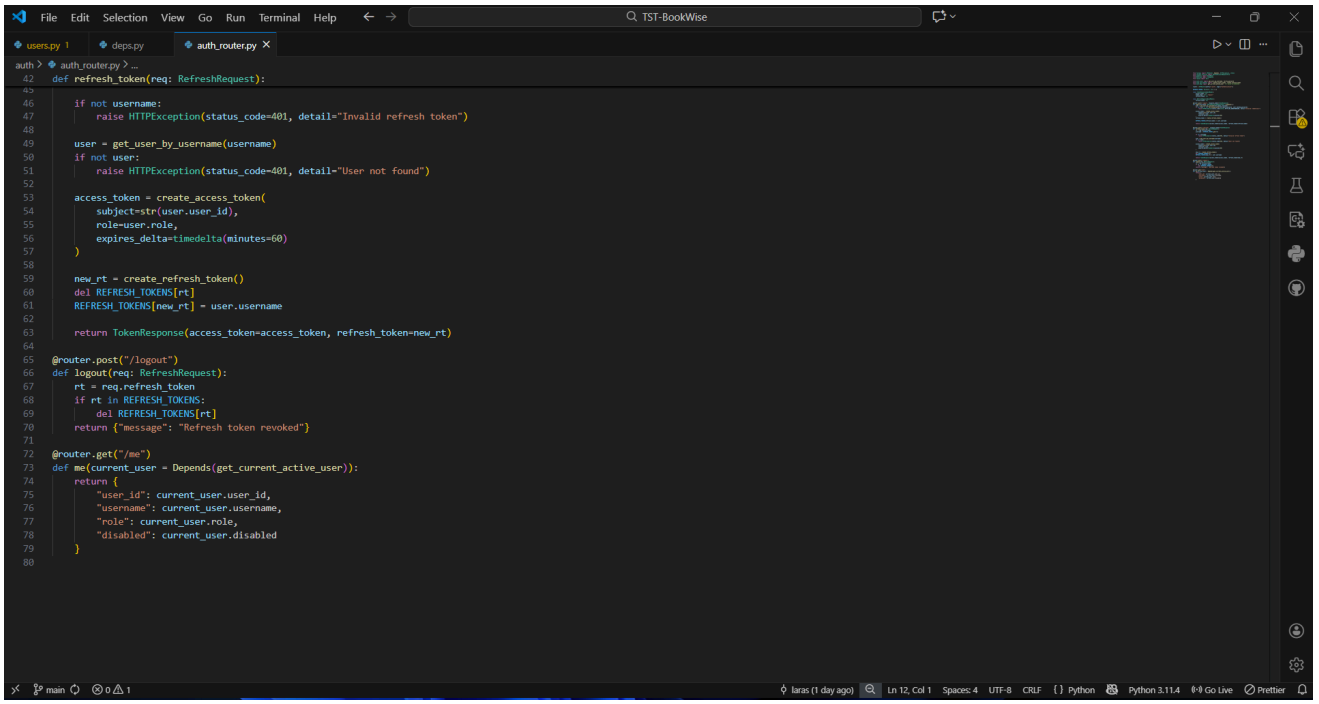
```
auth/
│
├── auth_router.py
├── deps.py
├── jwt_handler.py
└── users.py
```

Folder **auth/** ditambahkan karena pada *Milestone* 5 **BookWise** membutuhkan autentikasi untuk melindungi API. Semua dikumpulkan dalam folder ini supaya rapi dan terpisah dari domain utama. Di dalamnya, **auth_router.py**, **jwt_handler.py**, **deps.py**, dan **users.py**. Keempat file ini bekerja bersama untuk memastikan hanya pengguna yang berhak yang dapat mengakses fitur **BookWise.**

Implementasi	
4	File dan Penjelasan Kode
a.	auth/auth_router.py

File **auth_router.py** adalah tempat semua proses login dan keamanan di BookWise diatur. Di dalamnya, ada endpoint **/login** yang bertugas mengecek username dan password, lalu memberikan access token dan refresh token supaya pengguna bisa masuk ke sistem. Refresh token ini disimpan sementara sehingga bisa dipakai lagi di endpoint **/refresh** ketika access token sudah kedaluwarsa. Kalau pengguna ingin keluar, endpoint **/logout** akan menghapus refresh token-nya. Terakhir, endpoint **/me** dipakai untuk melihat siapa yang sedang login, dengan mengecek token yang dikirim di request.

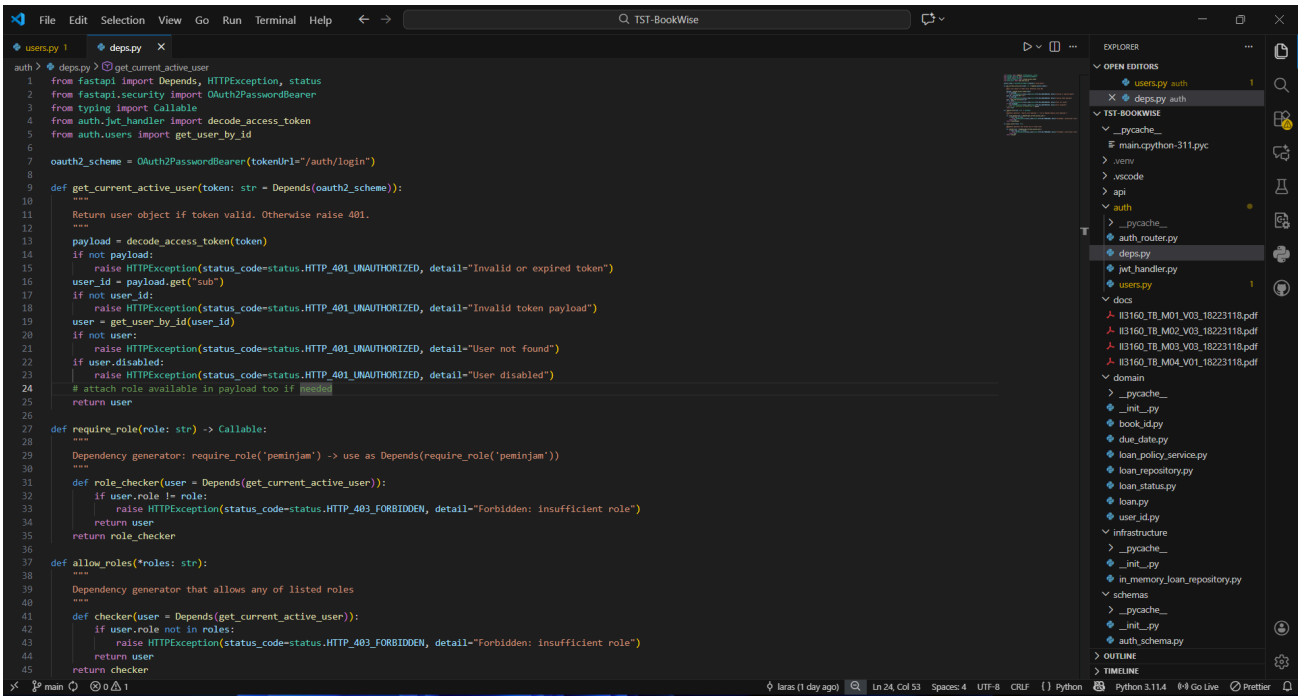




b.	auth/deps.py

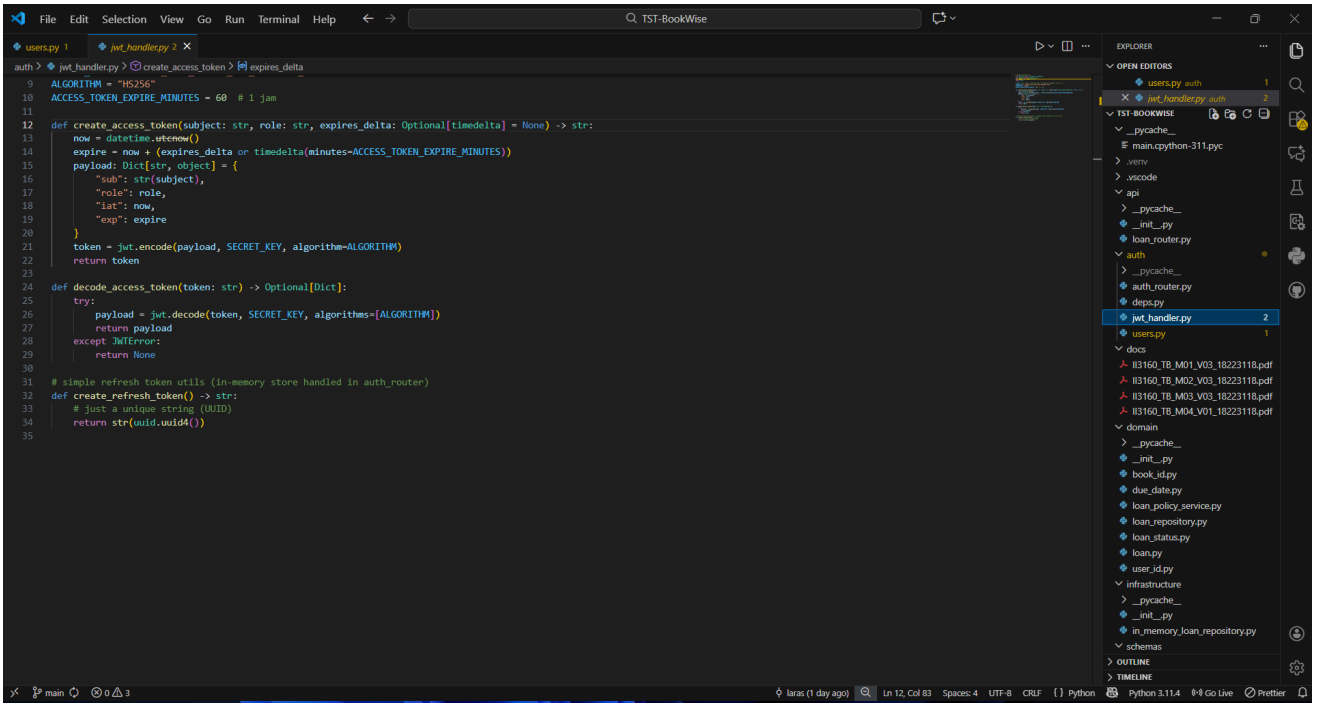
File **deps.py** berisi kumpulan dependency untuk memeriksa token dan mengatur hak akses setiap pengguna sebelum mereka boleh masuk ke endpoint tertentu. Di dalamnya ada fungsi **get_current_active_user**, yang bertugas membaca access token, memverifikasi apakah token valid, mengecek apakah user ada dan tidak dinonaktifkan, lalu mengembalikan objek user tersebut jika semuanya aman. Selain itu, ada juga fungsi **require_role** dan **allow_roles**, yang membantu membatasi akses berdasarkan peran pengguna, misalnya hanya peminjam atau hanya admin yang boleh mengakses

endpoint tertentu.



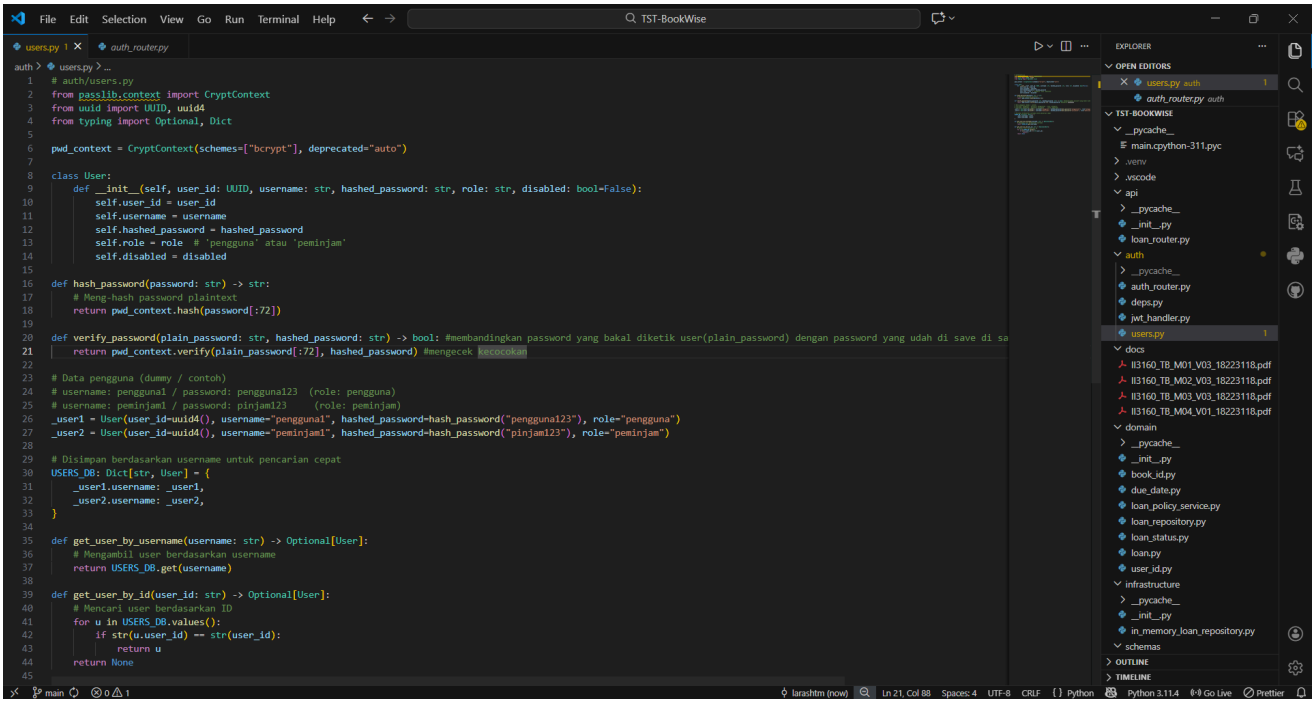
c.	auth/jwt_handler.py

File **jwt_handler.py** digunakan untuk membuat dan memverifikasi token JWT sistem autentikasi di BookWise. Di dalamnya terdapat fungsi **create_access_token**, yang membentuk token berisi informasi penting seperti user ID, role, waktu pembuatan, dan waktu kedaluwarsa, kemudian menandatanganinya menggunakan secret key. Fungsi **decode_access_token** digunakan untuk membaca token dan memastikan token valid, jika tidak valid atau sudah kedaluwarsa, fungsi ini mengembalikan nilai None sehingga request ditolak. Selain itu, file ini juga menyediakan fungsi **create_refresh_token**, yaitu generator token sederhana berbentuk UUID yang dipakai untuk memperbarui access token tanpa perlu login ulang.



d.	user.py

File **users.py** berfungsi sebagai tempat penyimpanan data pengguna sederhana yang digunakan untuk proses login di BookWise. Di dalamnya terdapat class **User**, yang menyimpan informasi dasar seperti user ID, username, password yang sudah di-hash, role (*pengguna* atau *peminjam*), serta status aktif. File ini juga menyediakan fungsi **hash_password** dan **verify_password** untuk mengubah password menjadi bentuk terenkripsi dan memeriksa kecocokannya saat pengguna login. Untuk keperluan tugas, dua akun contoh disiapkan secara hard-coded, lalu disimpan dalam dictionary **USERS_DB**, sehingga pencarian user bisa dilakukan dengan cepat melalui **get_user_by_username** atau **get_user_by_id**. Secara keseluruhan, file ini menyediakan data pengguna dan fungsi pendukung yang diperlukan agar autentikasi JWT di BookWise dapat berjalan.



5.	Payload Endpoint Autentication
a.	Endpoint Login (POST /auth/login)

1. Potongan kode terkait

```
@router.post("/login", response_model=TokenResponse)
def login(form_data: OAuth2PasswordRequestForm = Depends()):
```

Fungsi baris ini: menerima username & password via form-data, lalu mengembalikan access token + refresh token.

2. Request Payload

Tipe: `application/x-www-form-urlencoded`
Field: **`username`** → username pengguna/peminjam, **`password`** → password plaintext

3. Response Payload

Tipe: JSON
Field:

- **`access_token`** → token JWT utama (dipakai untuk semua endpoint protected)
- **`token_type`** → selalu "bearer"

- **`refresh_token`** → token untuk request `/auth/refresh`

Contoh Response:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer",
  "refresh_token": "d5a340b5-b8cb-4756-8d24-08d7a9011c2f"
}
```

4. **Penjelasan Teknis**
   - OAuth2PasswordRequestForm **memaksa client** mengirim username/password lewat form-data, standar OAuth2.
   - Server mengecek username → lalu verify password terhadap hash di database dummy.
   - Kalau valid:
     → **`generate access_token`** (berisi user_id + role + expired 60 menit)
     → **`generate refresh_token`** acak UUID
   - Refresh token disimpan sementara pada dictionary REFRESH_TOKENS.

b.	Endpoint Refresh Token (POST /auth/refresh)

1. **Potongan kode terkait**

```
@router.post("/refresh", response_model=TokenResponse)
def refresh_token(req: RefreshRequest):
```

Fungsi: menerima **`refresh_token`** lalu mengeluarkan access_token baru + refresh token baru.

2. **Request Payload**

Tipe: JSON
Field:

- **`refresh_token`** → token yang sebelumnya diberikan saat login

Contoh Request:

```
{
  "refresh_token": "d5a340b5-b8cb-4756-8d24-08d7a9011c2f"
}
```

3. **Response Payload**

   Field:

   - `access_token` → token JWT baru
   - `refresh_token` → refresh token baru (yang lama DIHAPUS)

   Contoh Response:

```
{
  "access_token": "eyJhbGc...baru...",
  "token_type": "bearer",
  "refresh_token": "fa22b8d1-c84e-47b3-93d9-8c84c3410bbd"
}
```

4. **Penjelasan Teknis**
   - Endpoint mengecek apakah refresh token valid (ada di REFRESH_TOKENS)
   - Kalau valid:
     → buat access token baru
     → buat refresh token baru
     → hapus refresh token lama
     → simpan refresh token baru di dictionary
   - Mekanisme ini memastikan refresh token **hanya bisa dipakai sekali**, meningkatkan keamanan.

c.	Endpoint LOGOUT (POST /auth/logout)

1. **Potongan kode terkait**

```
@router.post("/logout")
```

```
def logout(req: RefreshRequest):
```

2. **Request Payload**

   Field:

   - `refresh_token` → token yang ingin dicabut

   Contoh Request:

```
{
  "refresh_token": "fa22b8d1-c84e-47b3-93d9-8c84c3410bbd"
}
```

3. **Response Payload**

```
{
  "message": "Refresh token revoked"
}
```

4. **Penjelasan Teknis**
   - Logout dalam JWT **tidak menghapus access token** (karena JWT sifatnya stateless).
   - Jadi yang bisa dilakukan adalah:
     → menghapus refresh token dari dictionary
   - Tanpa refresh token, user tidak bisa memperbarui access token lagi → otomatis "keluar" setelah token expired.

d.	Endpoint GET PROFILE (POST /auth/me)

1. **Potongan kode terkait**

```
@router.get("/me")
def me(current_user = Depends(get_current_active_user)):
```

2. **Request Payload**

Tidak ada body.
Tapi butuh header:

Authorization: **`Bearer <access_token>`**

3. **Response Payload**

   Field:

   - user_id
   - username
   - role
   - disabled

   Contoh Response:

```
{
  "user_id": "ce93dd1d-30f5-48dd-8129-ac1e512f9306",
  "username": "pengguna1",
  "role": "pengguna",
  "disabled": false
}
```

4. **Penjelasan Teknis**
   - **`get_current_active_user`** memverifikasi access token:
     - valid atau tidak
     - expired atau tidak
     - user ditemukan atau tidak
   - Hanya access token valid yang bisa mengakses endpoint ini.
   - Berguna untuk mengecek siapa user aktif saat ini.

6.	Test Case

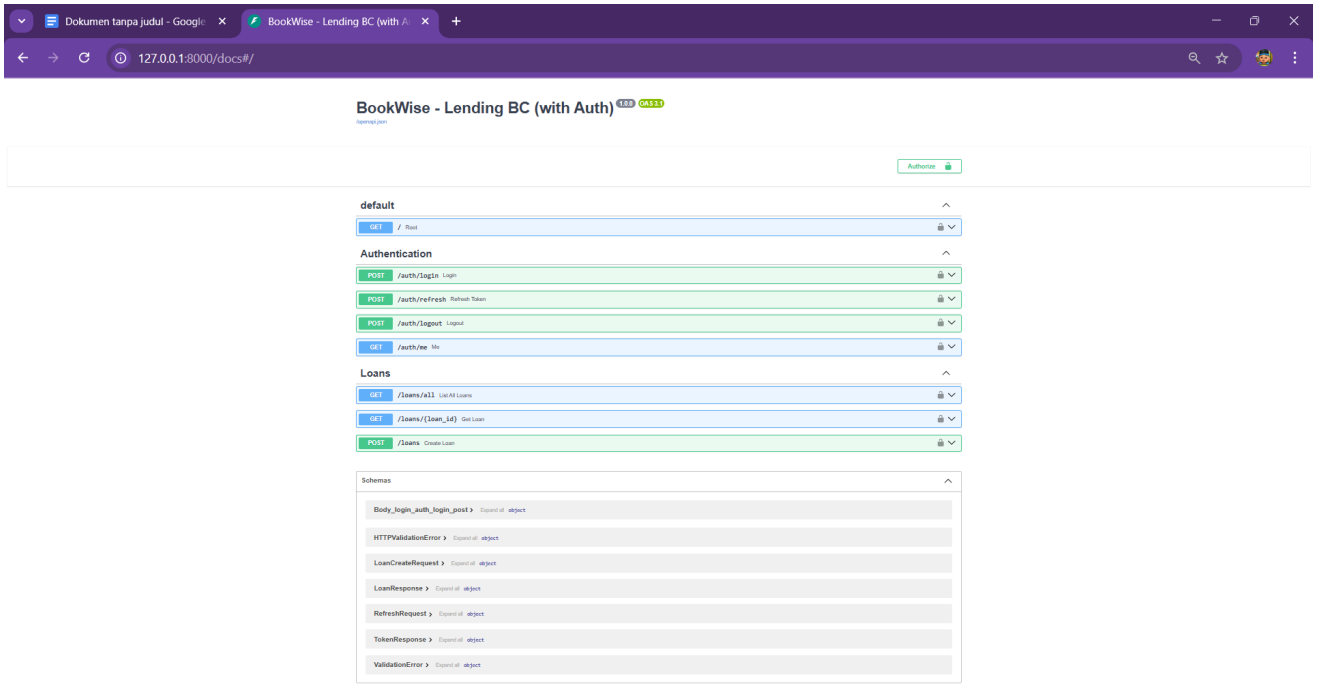
Cara Menampilkan Endpoint + Testing POST & GET di Swagger

Eksekusi Server

```
PS C:\Users\laras\Bahasa C\SEMESTER 5\II3160_TST\TST-BookWise> Uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\laras\\Bahasa C\\SEMESTER 5\\II3160_TST\\TST-BookWise']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [16444] using StatReload
INFO:     Started server process [5340]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

- Perintah: `uvicorn main:app --reload`
- Pastikan terlihat: “Uvicorn running on http://127.0.0.1:8000”

Saat membuka link tersebut akan muncul daftar endpoint yang memuat:



a.	**Test Case 1: Login (POST /auth/login) (Peminjam)**

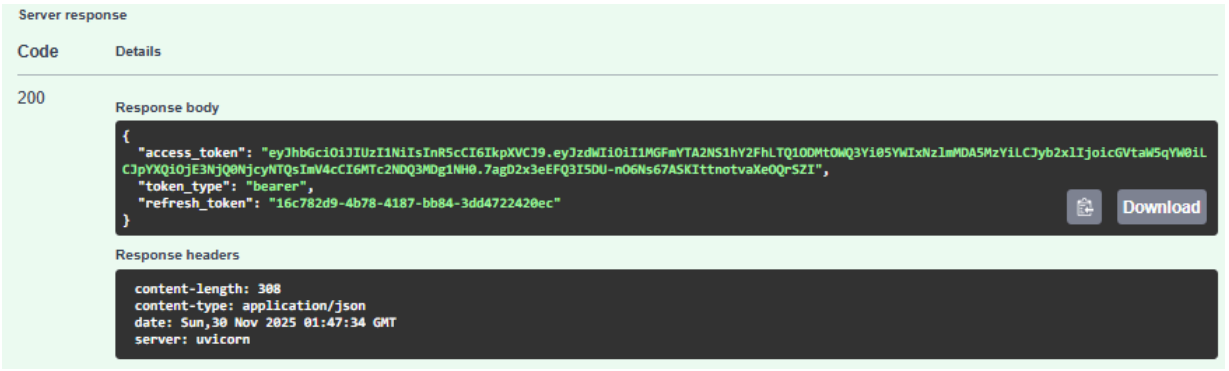
Terdapat 2 role yang sudah di setting dalam bookWise yang mencakup role pengguna dan role peminjam. Singkatnya, role peminjam akan membuat loan dan mengecek loan_id.

Tujuan: mendapatkan access_token dan refresh_token

Klik **POST /loans ->** Tekan tombol **Try it out ->** Isi JSON request body -> Klik **Execute**

Request:

```
{
  username: peminjam1
  password: pinjam123
}
```



b.	**Test Case 2: Login (POST /auth/login) (Pengguna)**

Role pengguna memiliki akses melihat data loan dalam penggunaan. Tes ini dilakukan dengan tujuan untuk mendapatkan access_token dan refresh_token untuk mengakses endpoint yang dipakai autentikasi.

Klik **POST /loans ->** Tekan tombol **Try it out ->** Isi JSON request body -> Klik **Execute**

Request:

```
{
  username: pengguna1
  password: pengguna123
}
```

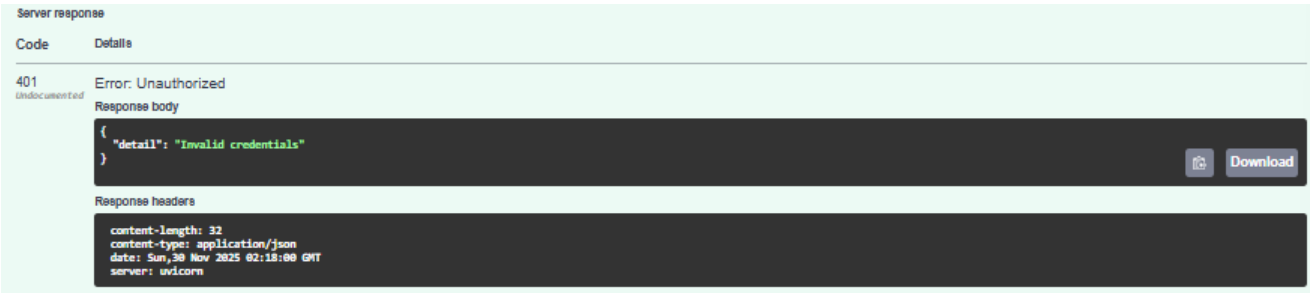
c.	**Test Case 3: Login (POST /auth/login) (Password salah)**

Tes ini dilakukan untuk memastikan sistem dapat menolak kredensial yang tidak valid. Jika password yang dimasukkan tidak sesuai, server harus memberikan respons error **401 Unauthorized** tanpa menghasilkan token apa pun. Dengan begitu, autentikasi BookWise tetap aman dan tidak memberikan akses kepada pengguna yang salah memasukkan password. Tujuan: mendapatkan access_token dan refresh_token

Klik **POST /loans ->** Tekan tombol **Try it out ->** Isi JSON request body -> Klik **Execute**

Request:

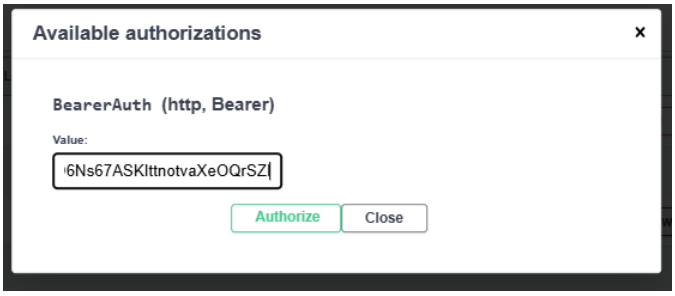
```
{
  username: peminjam1
  password: salah123
}
```

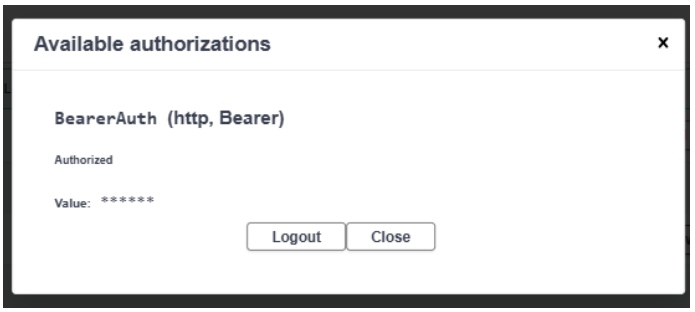


d.	Test Case 4: Authorize

Klik tombol **Authorize (ikon gembok. Terdapat di pojok kanan Swagger)** ->**Masukkan: Type dan Token** -> Klik **Authorize** -> Klik **Close**

Isi:
`Bearer <access_token>`





e.	Test Case 5: Create Loan

Fitur ini hanya boleh dijalankan oleh pengguna dengan role **peminjam**, karena hanya mereka yang berhak membuat pinjaman buku. Setelah tombol Authorize diaktifkan dan token peminjam dimasukkan, kamu bisa langsung mengetes endpoint ini.

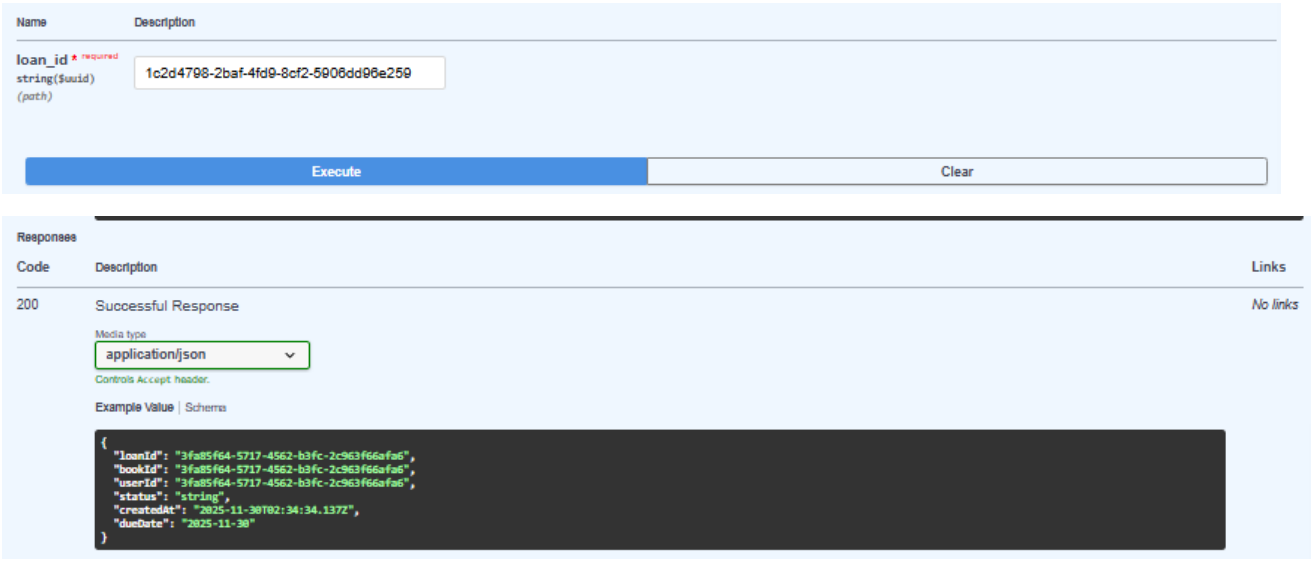
Klik **POST /loans ->** Tekan tombol **Try it out ->** Isi JSON request body -> Klik **Execute**

Request:

```
{
  "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "userId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
```

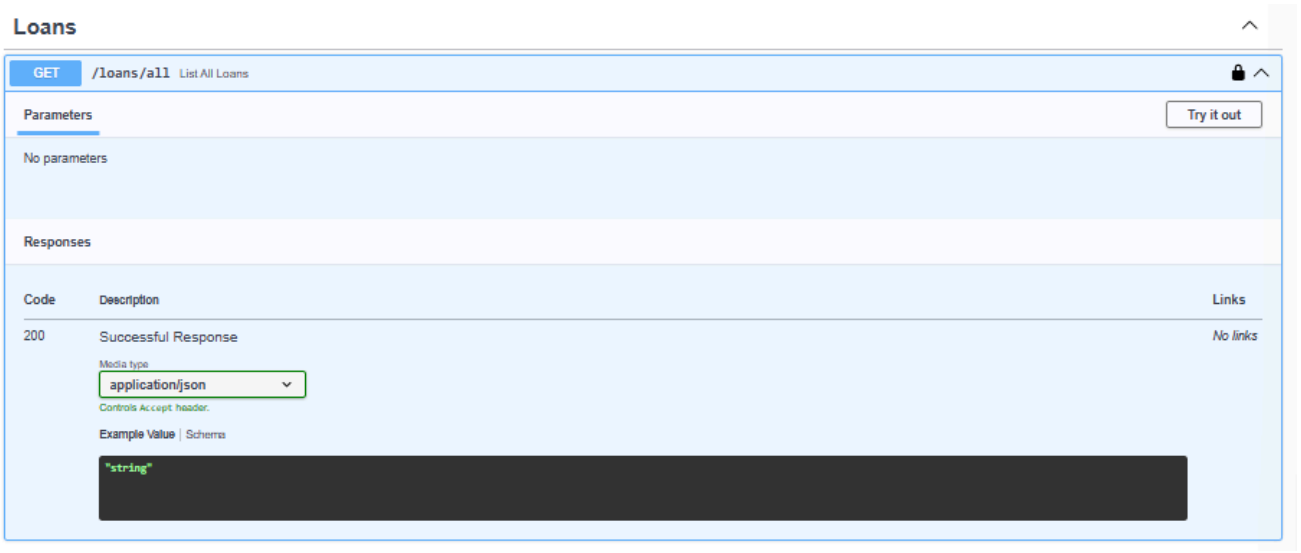
f.	Test Case 6: Get Loan

Login terlebih dahulu menggunakan akun peminjam. Fitur ini hanya dapat dijalankan oleh peminjam karena hanya mereka yang berhak membuat peminjaman buku. Setelah login, buka endpoint **POST /loans**, kemudian tekan tombol **Try it out**. Isi **JSON request body** dengan informasi buku dan pengguna, lalu klik **Execute**. Output yang diharapkan adalah sebuah objek yang berisi **loan_id**. Untuk menghindari konflik dengan path lain yang menggunakan method berbeda, posisi endpoint **Create Loan** ditempatkan di bagian paling bawah.



g.	Test Case 7: List Loans

Fitur **List Loans** merupakan pengembangan dari milestone sebelumnya dan hanya bisa diakses oleh pengguna. Jika dijalankan sebagai pengguna, response yang diterima akan sukses, sedangkan jika dijalankan sebagai peminjam, akses akan ditolak.
Karena fitur ini hanya menampilkan daftar pinjaman dan mengambil data, tidak usah menginput tambahan. Cukup lakukan **GET** untuk mendapatkan seluruh data pinjaman.
-> Dalam pengurutan endpoint, **loans/all** diletakkan di bagian atas agar FastAPI memilih path ini terlebih dahulu, sehingga path ini tidak terbaca sebagai UUID dengan salah.



h.	Test Case 8: Refresh Token (normal)

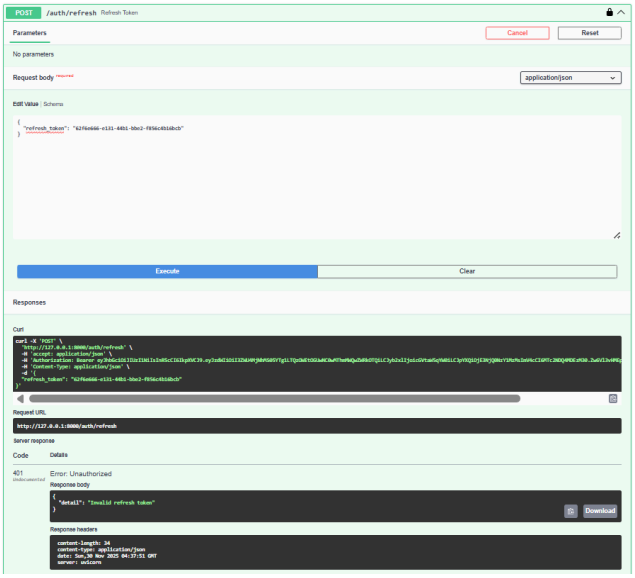
Fitur **Refresh Loans** dipakai untuk memperbarui token yang kadaluarsa. Cukup kirim JSON request body.

```
{
  "refresh_token": "<refresh_token>"
}
```

Setelah request dipakai, server akan ambil token baru.

i.	Test Case 9: Refresh Token setelah logout

Jika pengguna mencoba melakukan **Refresh Token** setelah melakukan **logout**, akses akan ditolak dan server akan mengembalikan status **401 Forbidden**. Hal ini menegaskan bahwa token refresh yang telah digunakan untuk logout tidak dapat dipakai kembali untuk mendapatkan token baru.



j.	Test Case 10: Logout

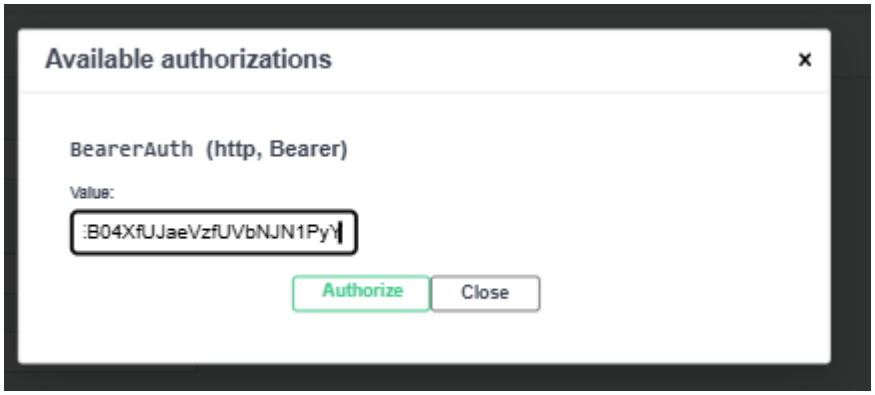
Fitur **Logout** untuk menonaktifkan token refresh jadi pengguna tidak lagi menggunakan token tersebut untuk memperbarui. Untuk menjalankannya, kirim:

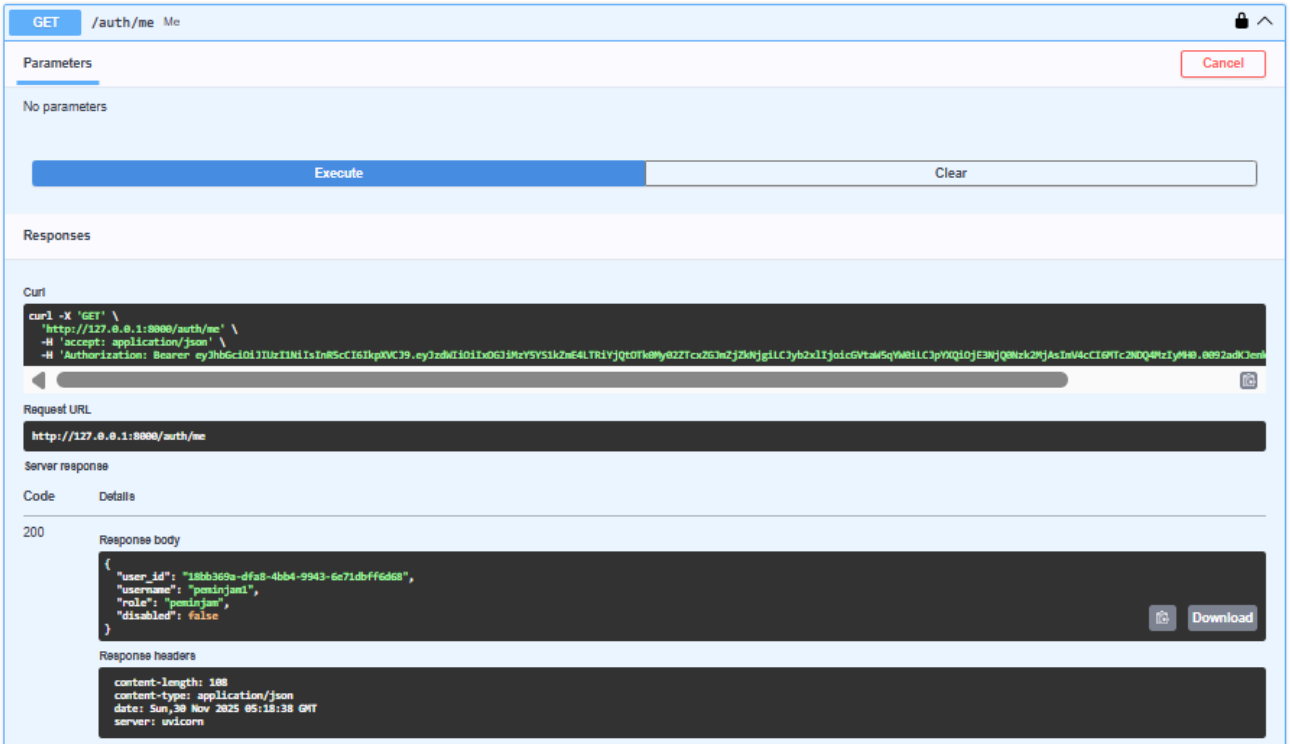
```
{
  "refresh_token": "<refresh_token>"
}
```

Setelah request dijalankan, token refresh akan dibatalkan dan pengguna harus melakukan login ulang.

k.	Get Profile (Authorized)

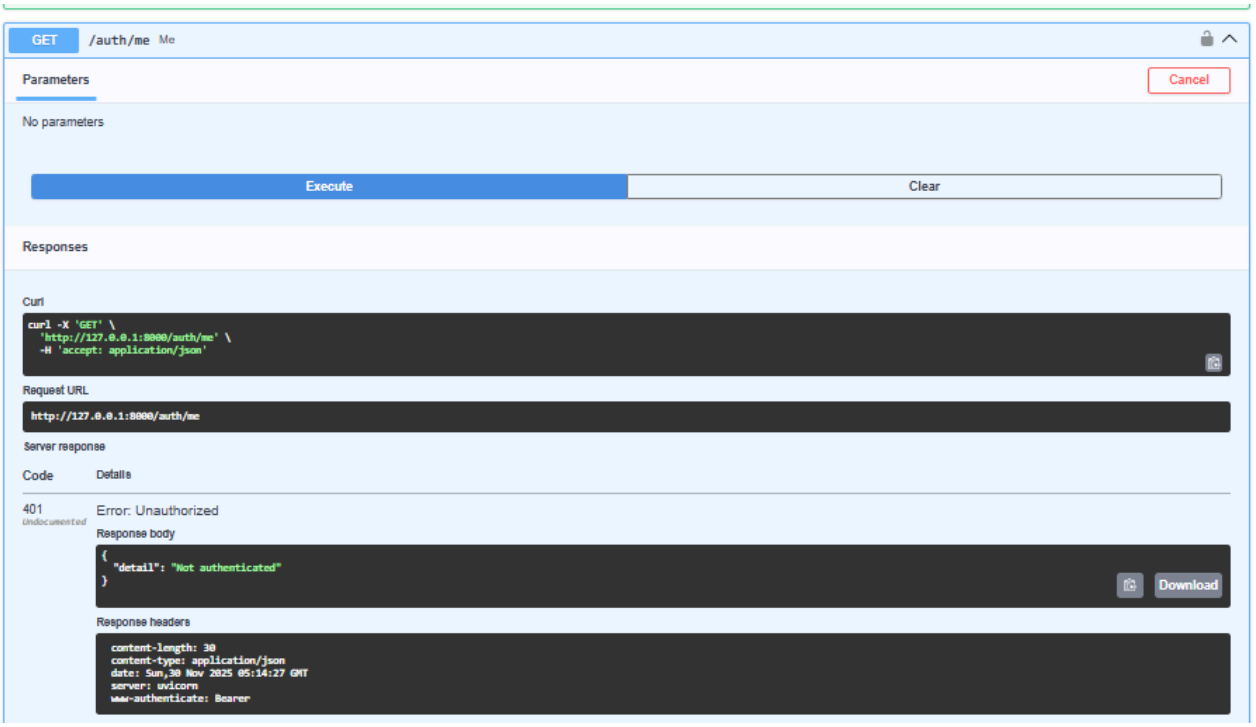
Endpoint **Get Profile** digunakan untuk mengambil informasi profil pengguna yang sedang login. Akses hanya diperbolehkan jika pengguna telah **authorized** (memiliki token yang valid). Jika request berhasil, server akan mengembalikan data profil pengguna dengan status **OK.**





l.	Get Profile tanpa Bearer Token

Jika endpoint **Get Profile** diakses tanpa menyertakan **Bearer Token** di header, server akan menolak akses dan mengembalikan status **401 Unauthorized**. Hal ini menegaskan bahwa semua request untuk mengambil data profil harus disertai token autentikasi yang valid.



7	Repository Git Hub

• URL repo:
https://github.com/larashtm/TST-BookWise