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

Implementasi Model Aggregate menjadi API

Bagian ini merujuk ke M03, fokus dokumen ini membahas mengenai implementasi aggregate Loan menjadi API. Implementasi dilakukan dengan membangun domain model pembentukan Aggregate Loan, pengerjaan ini memuat pembuatan repository dasar, pembangunan skema data yang disertai routing API dengan FastAPI.

Untuk memberikan gambaran ringkas mengenai struktur teknis yang dibangun, berikut disajikan pemetaan komponen sistem berdasarkan layer yang dipakai.

Tabel 1.1. Implementasi berdasarkan layer

Layer	Komponen	Implementasi (File/Folder)	Deskripsi Singkat
Domain Layer	Aggregate Root	`domain/loan.py`	Entity utama objek pinjam buku.
	Value Objects	`domain/book_id.py`, `domain/user_id.py`, `domain/loan_status.py`, `domain/due_date.py`	Value object yang digunakan untuk menyimpan identitas dan atribut.
	Domain Service	`domain/loan_policy_service.py`	Proses loan dasar.

Application Layer	API Router	`api/loan_router.py`	Router belum aktif sepenuhnya
	Schema	`schemas/loan_schema.py`	Struktur request/response untuk API.
Infrastructure Layer	Repository Implementation	`infrastructure/in_memory_loan_repository.py`	Implementasi penyimpanan.
	App Initialization	`main.py`	Menggabungkan router & menjalankan aplikasi.

Setelah memahami struktur komponen pada setiap layer, bagian berikut merangkum keseluruhan endpoint yang disediakan oleh sistem melalui API Surface.

Tabel 1.2. API Surface

Endpoint	Method	Deskripsi	Contoh Request Body
/loans	POST	Membuat permintaan peminjaman baru	{ "bookId": "uuid", "userId": "uuid" }
/loans/{loan_id}/verify	POST	Admin memverifikasi kelayakan peminjaman	*Tidak membutuhkan body*
/loans/{loan_id}/approve	POST	Admin menyetujui & mendistribusikan buku	*Tidak membutuhkan body*
/loans/{loan_id}	GET	Melihat detail loan termasuk status	*Tidak membutuhkan body*
/loans/my	GET	Peminjam melihat daftar pinjaman miliknya	*Tidak membutuhkan body*

/loans	GET	Admin melihat seluruh data pinjaman	*Tidak membutuhkan body*
/loans/{loan_id}/return	POST	Peminjam memulai proses pengembalian	*Tidak membutuhkan body*
/loans/{loan_id}/finalize	POST	Admin mengecek & menyelesaikan proses pengembalian	*Tidak membutuhkan body*
/loans/{loan_id}/extend	POST	Peminjam mengajukan perpanjangan durasi pinjam	{ "extra_days": 3 }

Selain daftar endpoint, sistem juga memiliki alur perubahan status peminjaman yang mengikuti aturan domain BookWise. Ringkasan perubahan status tersebut disajikan pada tabel berikut.

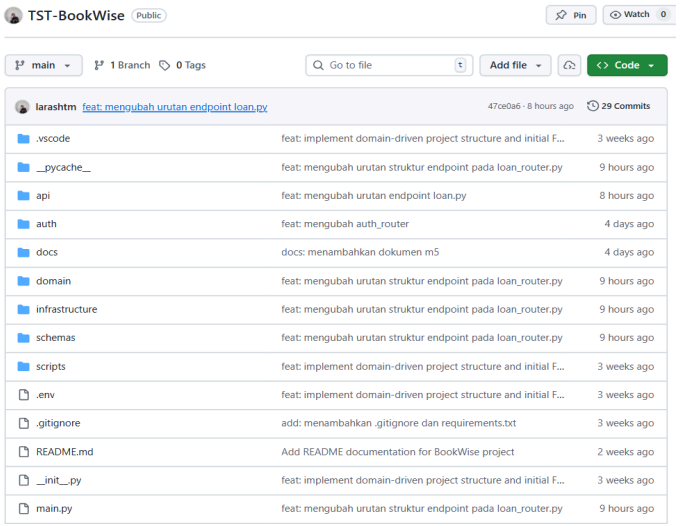
Tabel 1.3. Status Transition Matrix

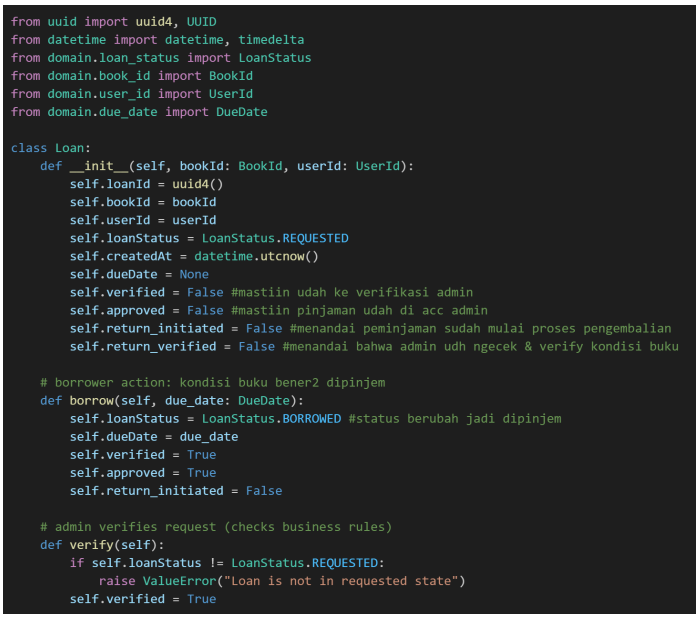
Endpoint	Method	Peminjam	Pengguna	Keterangan
/auth/login	POST	✔️	✔️	Dipakai oleh dua role untuk masuk
/auth/refresh	POST	✔️	✔️	Keduanya bisa refresh token
/auth/logout	POST	✔️	✔️	Keduanya bisa logout
/auth/me	GET	✔️	✔️	Ambil profil berdasarkan token
/loans	POST	✔️	×	Hanya peminjam yang bisa membuat pengajuan
/loans/my	GET	✔️	×	Peminjam melihat daftar pinjamannya sendiri
/loans/all / GET /loans	GET	×	✔️	Admin melihat seluruh loan

/loans/{loan_id}	GET	✔	✔	Admin bisa lihat semua, peminjam hanya miliknya
/loans/{loan_id}/verify	POST	×	✔	Verifikasi kelayakan peminjaman
/loans/{loan_id}/approve	POST	×	✔	Menyetujui & mendistribusikan buku
/loans/{loan_id}/return	POST	✔	×	Peminjam memulai pengembalian
/loans/{loan_id}/finalize	POST	×	✔	Admin finalisasi pengembalian
/loans/{loan_id}/extend	POST	✔	×	Peminjam meminta perpanjangan pinjaman

Setelah melihat API serta alur statusnya, bagian berikut menjelaskan implementasi BookWise secara lebih terperinci.

Tabel 1.4. Implementasi BookWise

No	Dokumentasi	Penjelasan
Struktur Folder pada Sistem Bookwise		
1.		a. **api/** memegang *use case level*, tempat request masuk dan response dikembalikan. b. **auth/** mengatur sistem autentikasi dan otorisasi serta memastikan hanya pengguna dengan token valid dan role yang tepat yang boleh mengakses fitur tertentu. c. **schemas/** memastikan data request valid dan konsisten dari router ke frontend dan sebaliknya. d. **domain/** representasi utama BookWise sesuai DDD. e. **infrastructure/** menjadi jembatan penyimpanan data. f. **scripts/** dipakai untuk tes

		pengujian.
Implementasi Domain Model		
1.	Aggregate Root: Loan	
	 	File: `domain/loan.py` Penjelasan: Loan berfungsi sebagai Aggregate Root yang menyimpan identitas peminjaman (LoanId) serta informasi penting lainnya seperti BookId, UserId, tanggal mulai dan tanggal jatuh tempo.
2.	Value Objects	

```python
from uuid import uuid4, UUID
from datetime import datetime, timedelta
from domain.loan_status import LoanStatus
from domain.book_id import BookId
from domain.user_id import UserId
from domain.due_date import DueDate

class Loan:
    def __init__(self, bookId: BookId, userId: UserId):
        self.loanId = uuid4()
        self.bookId = bookId
        self.userId = userId
        self.loanStatus = LoanStatus.REQUESTED
        self.createdAt = datetime.utcnow()
        self.dueDate = None
        self.verified = False #mastiin udah ke verifikasi admin
        self.approved = False #mastiin pinjaman udah di acc admin
        self.return_initiated = False #menandai peminjaman sudah mulai proses pengembalian
        self.return_verified = False #menandai bahwa admin udh ngecek & verify kondisi buku

    # borrower action: kondisi buku bener2 dipinjem
    def borrow(self, due_date: DueDate):
        self.loanStatus = LoanStatus.BORROWED #status berubah jadi dipinjem
        self.dueDate = due_date
        self.verified = True
        self.approved = True
        self.return_initiated = False

    # admin verifies request (checks business rules)
    def verify(self):
        if self.loanStatus != LoanStatus.REQUESTED:
            raise ValueError("Loan is not in requested state")
        self.verified = True
```

```python
# admin approves (distribute)
def approve(self, due_date: DueDate):
    if not self.verified:
        raise ValueError("Loan must be verified before approval")
    self.approved = True
    self.loanStatus = LoanStatus.BORROWED
    self.dueDate = due_date
    self.createdAt = datetime.utcnow()

# borrower mulai proses pengembalian
def initiate_return(self):
    if self.loanStatus != LoanStatus.BORROWED:
        raise ValueError("Loan is not currently borrowed")
    self.return_initiated = True #masuk ke step pengembalian
    # status tetap borowed sampai admin verifikasi

# admin memfinalisasi pengembalian
def finalize_return(self):
    if not self.return_initiated:
        raise ValueError("Return not initiated")
    self.return_verified = True
    self.loanStatus = LoanStatus.RETURNED
    self.return_initiated = False
    self.dueDate = None

def mark_overdue(self):
    #menandai pinjaman sebagai terlambat
    self.loanStatus = LoanStatus.OVERDUE

def extend_loan(self, extra_days: int):
    if not self.dueDate:
        raise ValueError("No due date to extend")
    #menghitung tanggal jatuh tempo
    new_date = self.dueDate.value + timedelta(days=extra_days)
    self.dueDate = DueDate(new_date)
    return self.dueDate
```

<table>
<tr>
<td>a</td>
<td>

```python
from uuid import UUID

class BookId:
    def __init__(self, value: UUID):
        self.value = value

```

</td>
<td>File: domain/book_id.py (Value Objects (VO)).
Dipakai supaya buku lebih aman dan sesuai dalam format. Misal kirim req:
{
"bookId": "e7d3d1dd-8fd8-4503-a4fa-b9f2407228c4"
}
Di Api router, input akan diubah jadi: BookId(UUID("e7d3d1dd-8fd8-4503-a4fa-b9f2407228c4"))</td>
</tr>
<tr>
<td>b.</td>
<td>

```python
from uuid import UUID

class UserId:
    def __init__(self, value: UUID):
        self.value = value

```

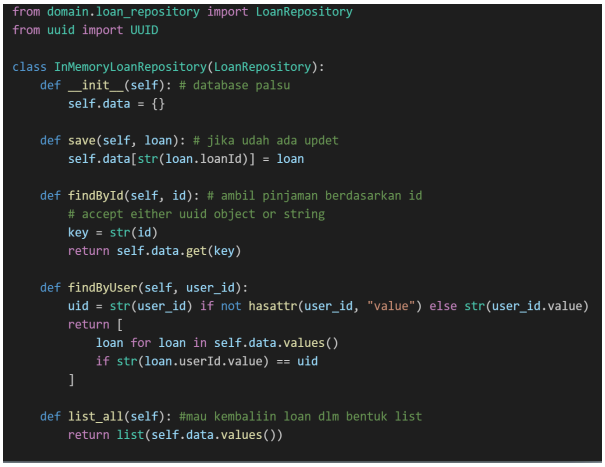
</td>
<td>File: domain/user_id.py (Value Objects (VO)).
Value Object yang mewakili identitas unik seorang pengguna.</td>
</tr>
<tr>
<td>c.</td>
<td>

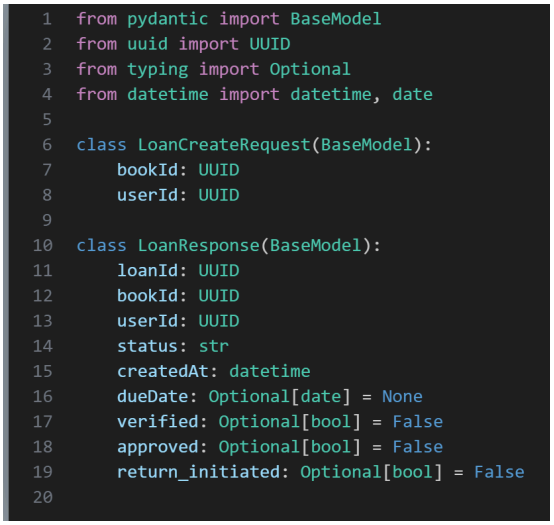
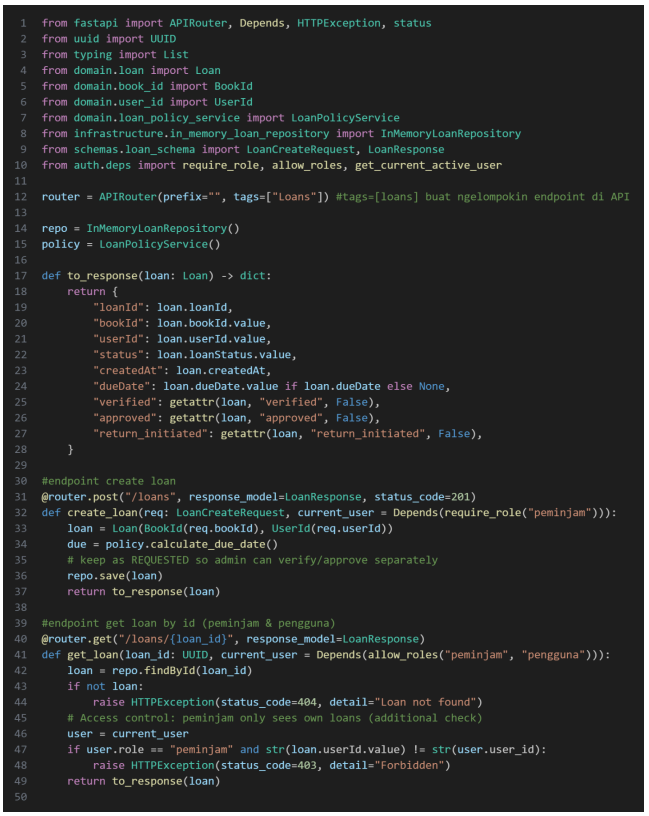
```python
from datetime import date

class DueDate:
    def __init__(self, value: date):
        self.value = value

    def is_overdue(self):
        return date.today() > self.value
```

</td>
<td>File: domain/due_date.py</td>
</tr>
<tr>
<td>d.</td>
<td>

```python
from enum import Enum

class LoanStatus(str, Enum):
    REQUESTED = "requested"
    BORROWED = "borrowed"
    RETURNED = "returned"
    OVERDUE = "overdue"

```

</td>
<td>File: domain/loan_status.py
- REQUESTED → peminjam baru mengajukan pinjaman
- VERIFIED → admin/pengguna memeriksa & menyetujui kelayakan
- BORROWED → pinjaman disetujui + diberikan ke peminjam
- RETURN_INITIATED → peminjam mulai proses pengembalian
- RETURNED → admin finalisasi pengembalian</td>
</tr>
<tr>
<td>3.</td>
<td colspan="2">Domain Server: LoanPolicyService</td>
</tr>
</table>

		File: domain/loan_policy_service.py
4.	Implementasi Repository	
a.		File: domain/loan_repository.py
b.		File: infrastructure/in_memory_loan_repository.py
5.	Implementasi API	

a.		File: `schemas/loan_schema.py`
b.		File: `api/loan_router.py`

```python
from pydantic import BaseModel
from uuid import UUID
from typing import Optional
from datetime import datetime, date

class LoanCreateRequest(BaseModel):
    bookId: UUID
    userId: UUID

class LoanResponse(BaseModel):
    loanId: UUID
    bookId: UUID
    userId: UUID
    status: str
    createdAt: datetime
    dueDate: Optional[date] = None
    verified: Optional[bool] = False
    approved: Optional[bool] = False
    return_initiated: Optional[bool] = False

```

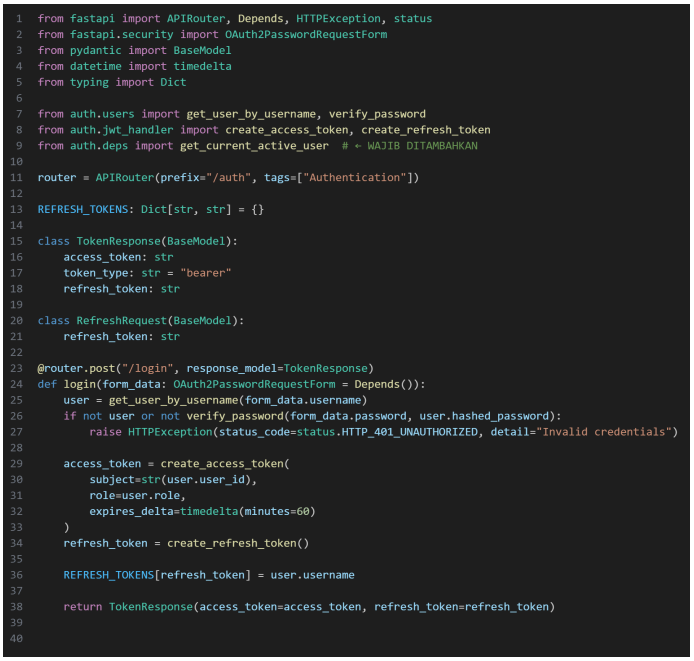
```python
from fastapi import APIRouter, Depends, HTTPException, status
from uuid import UUID
from typing import List
from domain.loan import Loan
from domain.book_id import BookId
from domain.user_id import UserId
from domain.loan_policy_service import LoanPolicyService
from infrastructure.in_memory_loan_repository import InMemoryLoanRepository
from schemas.loan_schema import LoanCreateRequest, LoanResponse
from auth.deps import require_role, allow_roles, get_current_active_user

router = APIRouter(prefix="", tags=["Loans"]) #tags=[loans] buat ngelompokin endpoint di API

repo = InMemoryLoanRepository()
policy = LoanPolicyService()

def to_response(loan: Loan) -> dict:
    return {
        "loanId": loan.loanId,
        "bookId": loan.bookId.value,
        "userId": loan.userId.value,
        "status": loan.loanStatus.value,
        "createdAt": loan.createdAt,
        "dueDate": loan.dueDate.value if loan.dueDate else None,
        "verified": getattr(loan, "verified", False),
        "approved": getattr(loan, "approved", False),
        "return_initiated": getattr(loan, "return_initiated", False),
    }

#endpoint create loan
@router.post("/loans", response_model=LoanResponse, status_code=201)
def create_loan(req: LoanCreateRequest, current_user = Depends(require_role("peminjam"))):
    loan = Loan(BookId(req.bookId), UserId(req.userId))
    due = policy.calculate_due_date()
    # keep as REQUESTED so admin can verify/approve separately
    repo.save(loan)
    return to_response(loan)

#endpoint get loan by id (peminjam & pengguna)
@router.get("/loans/{loan_id}", response_model=LoanResponse)
def get_loan(loan_id: UUID, current_user = Depends(allow_roles("peminjam", "pengguna"))):
    loan = repo.findById(loan_id)
    if not loan:
        raise HTTPException(status_code=404, detail="Loan not found")
    # Access control: peminjam only sees own loans (additional check)
    user = current_user
    if user.role == "peminjam" and str(loan.userId.value) != str(user.user_id):
        raise HTTPException(status_code=403, detail="Forbidden")
    return to_response(loan)

```

```python
#endpoint list loans punya peminjam
@router.get("/loans/my", response_model=List[LoanResponse])
def list_my_loans(current_user = Depends(require_role("peminjam"))):
    user = current_user
    loans = repo.findByUser(user.user_id)
    return [to_response(l) for l in loans]

#endpoint list semua loan(pengguna)
@router.get("/loans/all", response_model=List[LoanResponse])
def list_all_loans(current_user = Depends(require_role("pengguna"))):
    loans = repo.list_all()
    return [to_response(l) for l in loans]

#admin aksi: verifikasi loan
@router.post("/loans/{loan_id}/verify")
def verify_loan(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
    loan = repo.findById(loan_id)
    if not loan:
        raise HTTPException(status_code=404, detail="Loan not found")
    try:
        loan.verify()
        repo.save(loan)
        return {"detail": "Loan verified"}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

#admin apporve loan (nentuin due date dan status borrowed)
@router.post("/loans/{loan_id}/approve")
def approve_loan(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
    loan = repo.findById(loan_id)
    if not loan:
        raise HTTPException(status_code=404, detail="Loan not found")
    try:
        # Admin picks due date per policy
        due = policy.calculate_due_date()
        loan.approve(due)
        repo.save(loan)
        return {"detail": "Loan approved and distributed", "dueDate": due.value}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

#peminjam mulai proses pengembalian
@router.post("/loans/{loan_id}/return")
def initiate_return(loan_id: UUID, current_user = Depends(require_role("peminjam"))): #require_role: memastikan user harus punya role tertentu
    loan = repo.findById(loan_id)
    if not loan:
        raise HTTPException(status_code=404, detail="Loan not found")
    user = current_user
    if str(loan.userId.value) != str(user.user_id):
        raise HTTPException(status_code=403, detail="Forbidden")
    try:
        loan.initiate_return()
        repo.save(loan)
        return {"detail": "Return initiated"}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
```

```python
#aksi admin: finalisasi pengembalian
@router.post("/loans/{loan_id}/finalize-return")
def finalize_return(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
    loan = repo.findById(loan_id)
    if not loan:

        raise HTTPException(status_code=404, detail="Loan not found")
    try:
        loan.finalize_return()
        repo.save(loan)
        return {"detail": "Return finalized"}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

#endpoint perpanjangan masa pinjam
from pydantic import BaseModel
class ExtendRequest(BaseModel):
    extra_days: int

@router.post("/loans/{loan_id}/extend")
def extend_loan(loan_id: UUID, req: ExtendRequest, current_user = Depends(require_role("peminjam"))):
    loan = repo.findById(loan_id)
    if not loan:
        raise HTTPException(status_code=404, detail="Loan not found")
    # only peminjam can request extension
    user = current_user
    if str(loan.userId.value) != str(user.user_id):
        raise HTTPException(status_code=403, detail="Forbidden")
    try:
        new_due = loan.extend_loan(req.extra_days)
        repo.save(loan)
        return {"detail": "Extension applied", "newDueDate": new_due.value}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

```

c.		File: main.py
OVERVIEW JWT SEDERHANA		
1.	Penjelasan 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	
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.	

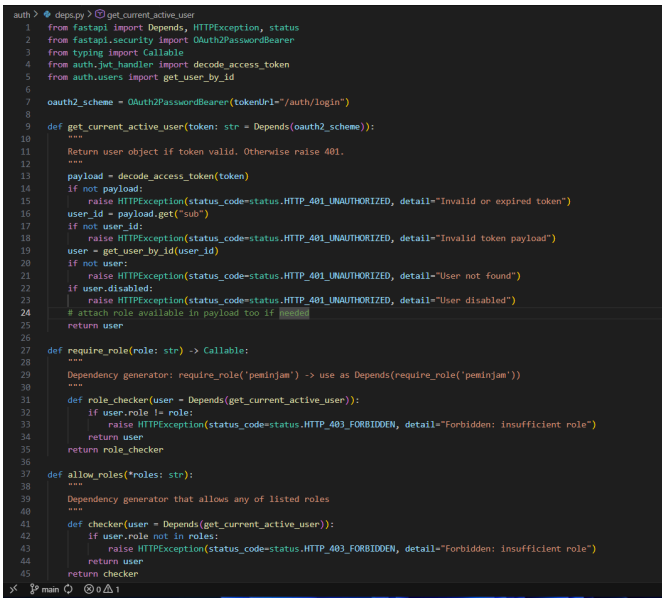
	3. Jika valid → sistem membuat JWT dan mengembalikannya kepada pengguna/peminjam. 4. Pengguna mengirim token untuk setiap permintaan dengan format: *Authorization*: Bearer <token> 5. Server memverifikasi token pada setiap request ke *endpoint* yang dilindungi 6. Jika token *valid* → akses diberikan 7. Jika token tidak *valid* atau hilang → server mengembalikan 401 Unauthorized

IMPLEMENTASI JWT AUTHENTICATION		
1.		File **auth_router.py** adalah tempat semua proses login dan keamanan di BookWise diatur. Beberapa endpoint: - POST /login → Mengecek username/password & memberi access + refresh token - POST /refresh → Memperbarui access token - POST /logout → Menghapus refresh token, logout - GET /me → Menampilkan info pengguna yang sedang login

```python
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from pydantic import BaseModel
from datetime import timedelta
from typing import Dict

from auth.users import get_user_by_username, verify_password
from auth.jwt_handler import create_access_token, create_refresh_token
from auth.deps import get_current_active_user  # ← WAJIB DITAMBAHKAN

router = APIRouter(prefix="/auth", tags=["Authentication"])

REFRESH_TOKENS: Dict[str, str] = {}

class TokenResponse(BaseModel):
    access_token: str
    token_type: str = "bearer"
    refresh_token: str

class RefreshRequest(BaseModel):
    refresh_token: str

@router.post("/login", response_model=TokenResponse)
def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user = get_user_by_username(form_data.username)
    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid credentials")

    access_token = create_access_token(
        subject=str(user.user_id),
        role=user.role,
        expires_delta=timedelta(minutes=60)
    )
    refresh_token = create_refresh_token()

    REFRESH_TOKENS[refresh_token] = user.username

    return TokenResponse(access_token=access_token, refresh_token=refresh_token)


```

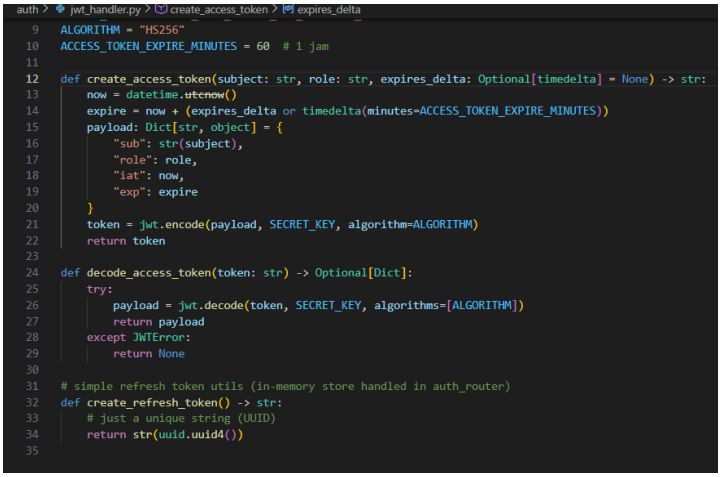
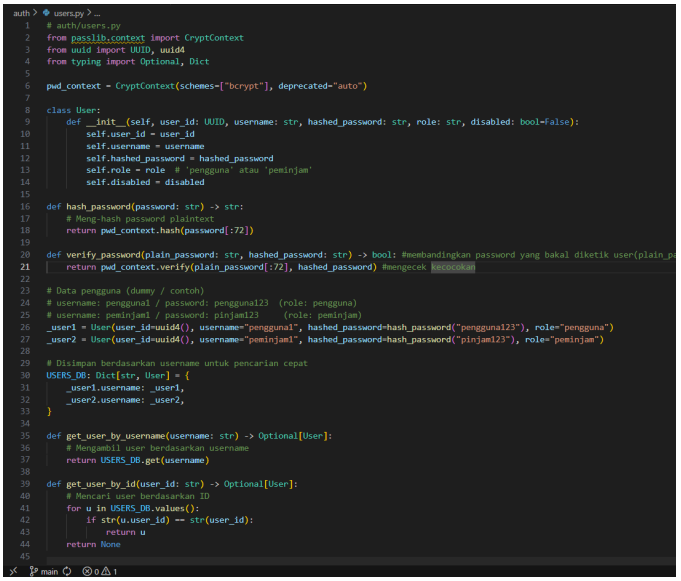
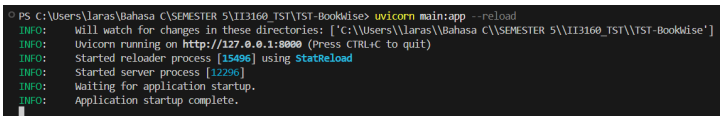
```python
@router.post("/refresh", response_model=TokenResponse)
def refresh_token(req: RefreshRequest):
    rt = req.refresh_token
    username = REFRESH_TOKENS.get(rt)

    if not username:
        raise HTTPException(status_code=401, detail="Invalid refresh token")

    user = get_user_by_username(username)
    if not user:
        raise HTTPException(status_code=401, detail="User not found")

    access_token = create_access_token(
        subject=str(user.user_id),
        role=user.role,
        expires_delta=timedelta(minutes=60)
    )

    new_rt = create_refresh_token()
    del REFRESH_TOKENS[rt]
    REFRESH_TOKENS[new_rt] = user.username

    return TokenResponse(access_token=access_token, refresh_token=new_rt)

@router.post("/logout")
def logout(req: RefreshRequest):
    rt = req.refresh_token
    if rt in REFRESH_TOKENS:
        del REFRESH_TOKENS[rt]
    return {"message": "Refresh token revoked"}

@router.get("/me")
def me(current_user = Depends(get_current_active_user)):
    return {
        "user_id": current_user.user_id,
        "username": current_user.username,
        "role": current_user.role,
        "disabled": current_user.disabled
    }
```

2.

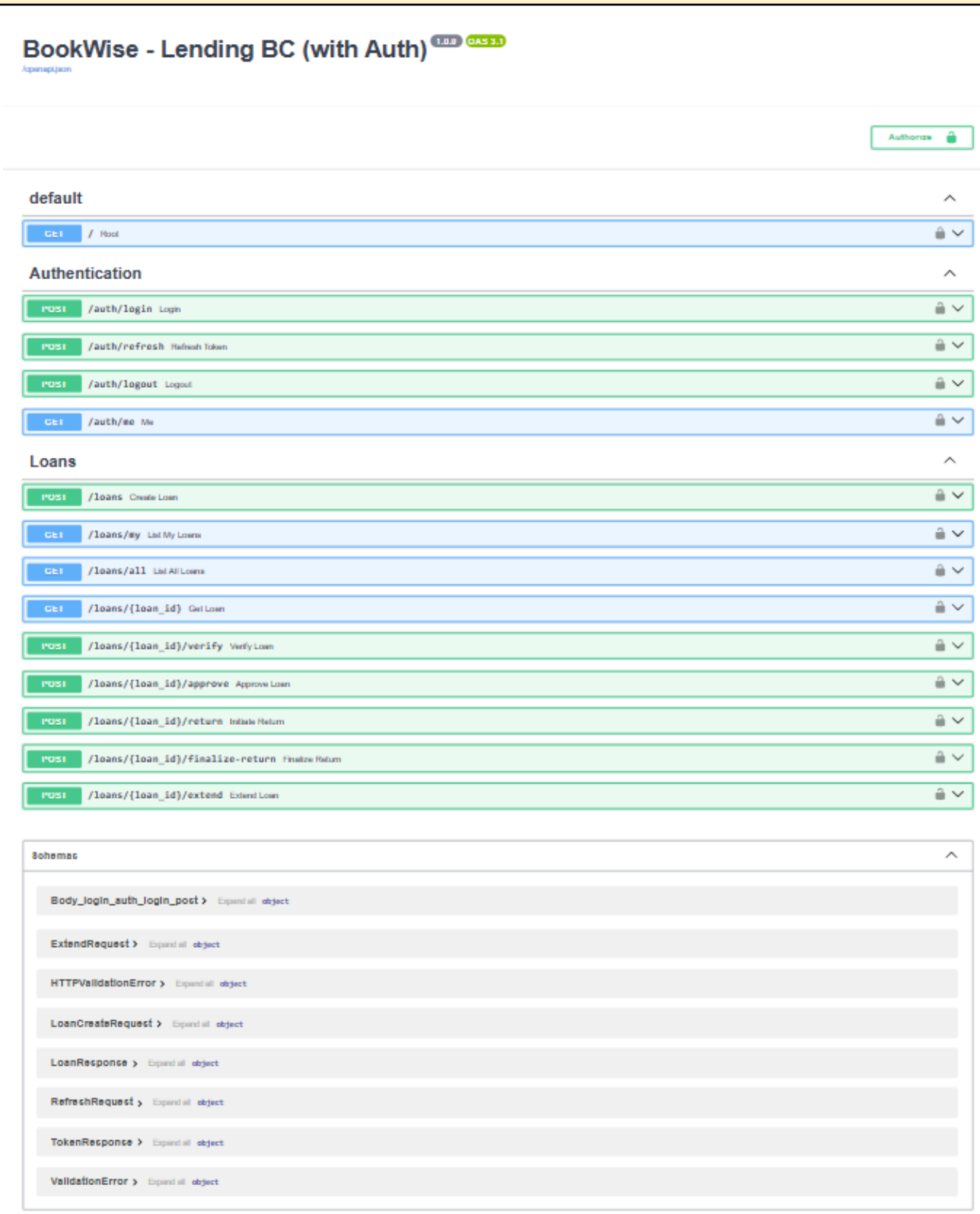


File **deps.py** berisi kumpulan dependency untuk memeriksa token dan mengatur hak akses setiap pengguna sebelum mereka boleh masuk ke endpoint tertentu. Di dalamnya ada fungsi:

- get_current_active_user → Membaca access token, memverifikasi validitas token, mengecek keberadaan dan status aktif user, lalu mengembalikan objek user.
- require_role → Membatasi akses endpoint hanya untuk peran tertentu.
- allow_roles → Membatasi akses endpoint untuk beberapa peran yang diizinkan.

3.		File **jwt_handler.py** dipakai untuk mengatur pembuatan dan verifikasi token JWT sistem autentikasi di BookWise. Di dalamnya terdapat fungsi: - create_access_token → Membuat token JWT berisi user info & kedaluwarsa. - decode_access_token → Verifikasi token, None kalau tidak valid/kadaluarsa. - create_refresh_token → Buat token UUID untuk perbarui access token.
4.		File **users.py** berfungsi sebagai tempat penyimpanan data pengguna sederhana yang digunakan untuk proses login di BookWise. Di dalamnya terdapat: - hash_password → Mengenkripsi password. - verify_password → Memeriksa kecocokan password login. - USERS_DB → Dictionary berisi akun contoh hard-coded. - get_user_by_username / get_user_by_id → Mencari user dengan cepat berdasarkan username atau ID.
PENGUJIAN SWAGGER UI		
1.	Eksekusi Server 	• Perintah: `uvicorn main:app --reload` • Pastikan terlihat: “Uvicorn running on http://127.0.0.1:8000” • Lalu pergi ke link berikut: http://127.0.0.1:8000/docs

Disclaimer: Apabila belum muncul maka: loan_router.py belum di-include di main.py.



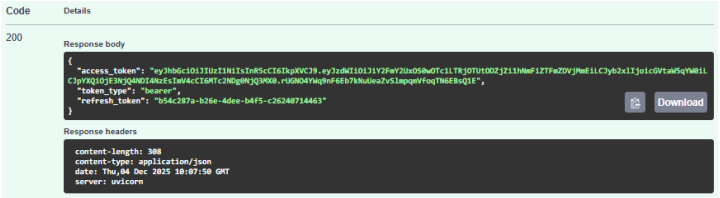
	Respon Swagger	Alur Ringkas

<table>
<tr><th colspan="3">Test Case 1: Authentication</th></tr>
<tr>
<td>1.</td>
<td>Login sebagai Peminjam
 
Login sebagai Pengguna
  </td>
<td>1. Kirim Kredensial
Peminjam/Pengguna login dengan username/password ke <code>/auth/login.</code>
2. Verifikasi
Server cek kredensial di database. Jika valid, lanjut ke token.
3. Generate Token
Server membentuk:
- <code>access_token</code> → JWT berisi sub (user ID), <code>role</code> (<code>peminjam</code>), <code>iat</code>, exp
- <code>refresh_token</code> → UUID unik
- <code>token_type</code> → <code>"bearer"</code>
4. Response ke Client
<pre>{

  "access_token": "...JWT...",

  "token_type": "bearer",

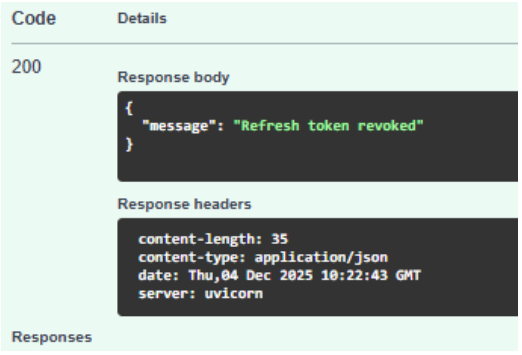
  "refresh_token": "...UUID..."

}</pre>
5. Penggunaan
Client pakai <code>access_token</code> di header Authorization untuk akses endpoint yang butuh login.</td>
</tr>
<tr>
<td>2.</td>
<td>Refresh token
  </td>
<td>1. Client Kirim Refresh Token
- Client (aplikasi) mengirim request ke endpoint refresh, <code>/auth/refresh.</code>
- Body request hanya berisi:
<pre>{

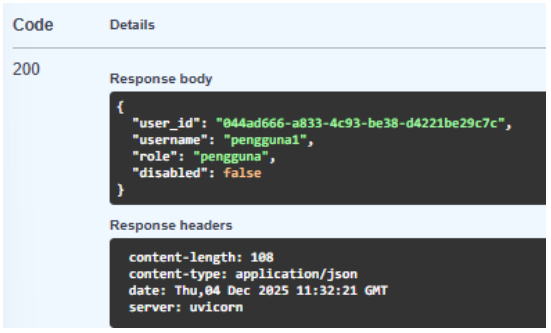
  "refresh_token": "string"</pre></td>
</tr>
</table>

		}
		
		c. "string" diganti dengan refresh token yang sebelumnya diterima saat login.
		
		2. Server Verifikasi Refresh Token
		
		- Server cek apakah refresh token valid dan belum kadaluarsa.
		- Jika tidak valid → request ditolak.
		- Jika valid → server lanjut buat access token baru.
		
		3. Server Buat Token Baru
		
		- Server buat access token baru (JWT) dengan informasi yang sama seperti saat login.
		- Server juga bisa buat refresh token baru (opsional, tapi biasanya diganti supaya lebih aman).
		
		4. Server Kirim Response
		
		contoh:
		
		{
		
		"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJiY2FmY2UxOS0wOTc1LTRjOTUtODZjZi1hNmFiZTFmZDVjMmEiLCJyb2xlIjoicGVtaW5qYW0iLCJpYXQiOjE3NjQ4NDI4NzEsImV4cCI6MTc2NDg0NjQ3MX0.rUGNO4YWq9nF6Eb7kNuUeaZvSlmpqmVfoqTN6EBsQ1E",
		
		"token_type": "bearer",
		
		"refresh_token": "b54c287a-b26e-4dee-b4f5-c26240714463"
		
		}
		
		Penjelasan payload:
		
		- **access_token** → JWT baru,

		dipakai untuk request API selanjutnya. - **`token_type`** → `"bearer"`, sama seperti login. - **`refresh_token`** → refresh token baru, disimpan untuk permintaan refresh berikutnya. **5. Penggunaan** - Client simpan access token baru dan pakai untuk akses API. - Refresh token lama biasanya tidak berlaku lagi jika server mengeluarkan token baru.

3.	Logout
	 
	Code Details
	200
	Response body
	{ "message": "Refresh token revoked" }
	Response headers
	content-length: 35
	content-type: application/json
	date: Thu,04 Dec 2025 10:22:43 GMT
	server: uvicorn
	Responses
		1. Client Kirim Refresh Token
		- Endpoint: misal /auth/logout.
		- Body request berisi refresh token yang tersimpan:
		
		`{`
		
		`"refresh_token": "string"`
		
		`}`
		
		"string" diganti dengan refresh token.
		
		2. Server Revoke Refresh Token
		- Server mengecek refresh token:
		
		○ Jika valid → token dicabut (revoked), tidak bisa dipakai lagi untuk generate access token baru.
		○ Jika tidak valid → bisa return error (misal 401 atau 400).
		3. Response
		- Server memberi konfirmasi logout:
		
		`{`
		
		`"message": "Refresh token revoked"`

		} - Status code: 200 OK - Menandakan refresh token berhasil dicabut dan user tidak bisa refresh lagi tanpa login ulang.

4.	Informasi Profil
	 
	
		1. Client kirim request ke `/auth/me` dengan header Authorization:
		
		Authorization: Bearer <access_token>
		
		2. Server membaca token dan memverifikasi:
		○ Apakah token valid?
		○ Apakah belum kadaluarsa?
		
		3. Jika valid, server mengembalikan data user terkait token itu, misal:
		
		{
		
		"user_id": "044ad666-a833-4c93-be38-d4221be29c7c",
		
		"username": "pengguna1",
		
		"role": "pengguna",
		
		"disabled": false
		
		}

Test Case 2: Loans		
1.	Create Loan Sesuai(Peminjam)	**1. Pastikan User Sudah Login dan Terautentikasi** - Hanya peminjam yang bisa membuat Loan Endpoint: misal /auth/logout. - Client harus pakai access token dari login di header:

Code Details

201

Response body


```
{
  "loanId": "ab439f55-e681-4525-8403-e42bc0ef9fd8",
  "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af78",
  "userId": "bcafce19-0975-4c95-86cf-a6abe1fd5c2a",
  "status": "requested",
  "createdAt": "2025-12-04T10:50:25.610158",
  "dueDate": null,
  "verified": false,
  "approved": false,
  "return_initiated": false
}
```


Response headers

```
content-length: 281
content-type: application/json
date: Thu,04 Dec 2025 10:50:25 GMT
server: uvicorn
```

Create Loan Ga Sesuai

Code Details

422

Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "type": "uuid_parsing",
      "loc": [
        "body",
        "userId"
      ],
      "msg": "Input should be a valid UUID, invalid length: expected length 32 for simple format,
      "input": "000",
      "ctx": {
        "error": "invalid length: expected length 32 for simple format, found 3"
      }
    }
  ]
}
```


Response headers

```
content-length: 254
content-type: application/json
date: Thu,04 Dec 2025 10:55:56 GMT
server: uvicorn
```

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

- Jalankan /auth/me untuk mengetahui userId yang sedang login. Ini memastikan user yang membuat loan memang sesuai.

2. Buat Request ke Create Loan

- Endpoint: misal /loans (POST)
- Body request berisi:

```
{

  "bookId": "<UUID buku>",

  "userId": "<UUID peminjam dari /auth/me>"

}
```

3. Validasi Server

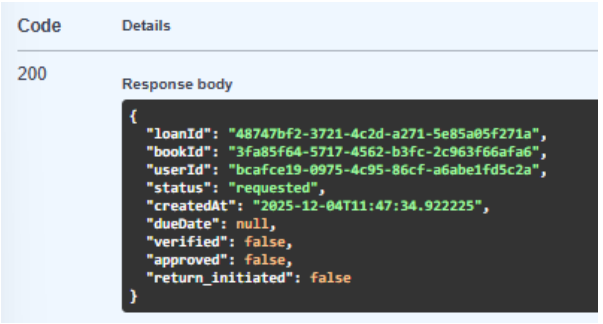
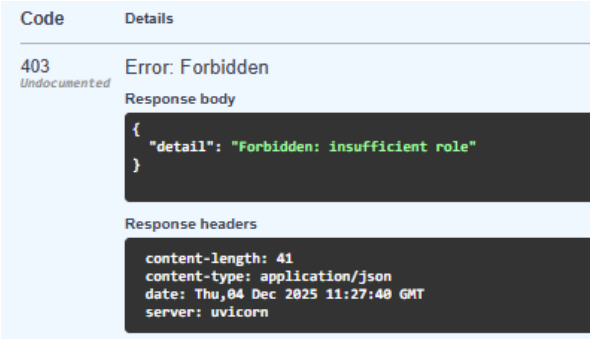
- Server cek:
  - **Role** → pastikan "peminjam"
  - **UUID valid** → bookId dan userId harus format UUID
  - **userId sesuai token** → kalau tidak sesuai muncul **422** dengan detail error:

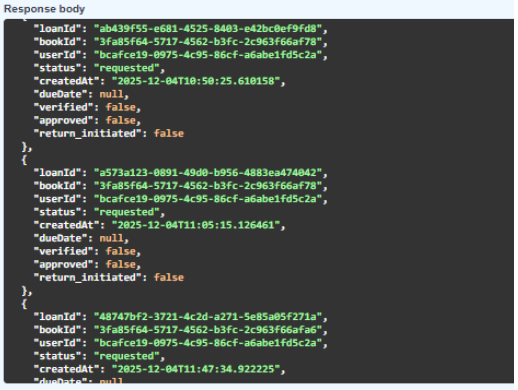
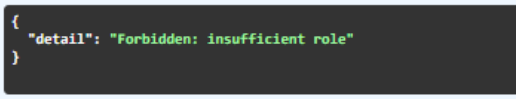
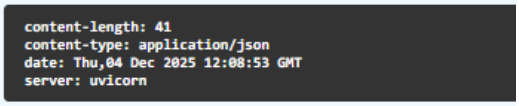
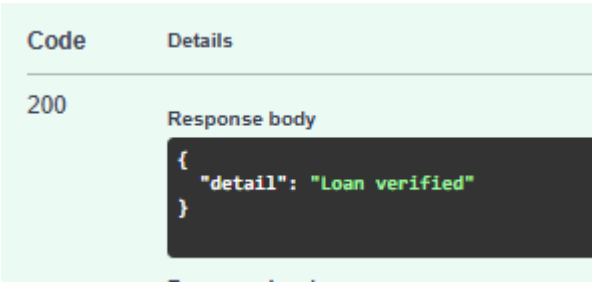
```
{
  "detail": [
    {
      "type": "uuid_parsing",
      "loc": ["body", "userId"],
      "msg": "Input should be a valid UUID, invalid length",
      "input": "000"
    }
  ]
}
```

4. Jika Valid

- Server buat loan baru dengan status awal “requested”, belum diverifikasi atau disetujui:

```
{
```

		"loanId": "a573a123-0891-49d0-b956-4883ea474042", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af78", "userId": "bcafce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:05:15.126461", "dueDate": null, "verified": false, "approved": false, "return_initiated": false } - **loanId** → ID pinjaman baru - **status** → "requested" - **verified / approved / return_initiated** → false karena belum
2.	Get Loan (My) (Peminjam)  Get Loan (All) (Peminjam)  Get Loan (All) (Pengguna)	1. Get Loan Detail • Endpoint: GET /loans/{loan_id} • **Input:** loan_id yang diperoleh saat **create loan**. • **Role:** harus **peminjam**. • **Output:** 200 OK dengan payload detail pinjaman: { "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "userId": "bcafce19-0975-4c95-86cf-a6abe1fd5c2a", "status":

	200 Response body  Get Loan (My) (Pengguna) 403 *Undocumented* Error: Forbidden Response body  Response headers 	"requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false } • **Server cek:** - Access token valid - Role = "peminjam" - userId di token sama dengan userId loan 2. Get All Loans • Endpoint: GET /loans/all • **Input:** tidak perlu input apapun. • **Role: peminjam tidak diperbolehkan**, hanya admin atau role tertentu bisa akses. • **Output:** 403 Forbidden
3.	Verfiy Loan(Pengguna) 	1. Login sebagai Pengguna - Pengguna harus login dahulu lalu akan memperoleh access token. - Gunakan token untuk akses endpoint. 2. Verify Loan - Endpoint: POST /loans/{loan_id}/verify - Input: loan_id pinjaman yang ingin diverifikasi.

	Code Details 404 Undocumented Error: Not Found Response body { "detail": "Loan not found" } Response headers content-length: 27 content-type: application/json date: Thu,04 Dec 2025 12:23:40 GMT server: uvicorn	Syarat server: - Loan harus dalam status "REQUESTED" - Hanya admin yang boleh memverifikasi 3. Respons: - Jika sukses akan menunjukan 200 ok - Jika loan dalam status REQUESTED artinya 400 Bad - Jika loan_id tidak ada maka akan menampilkan 404 not found.
4.	Approve Loan (Pengguna) Code Details 200 Response body { "detail": "Loan approved", "dueDate": "2025-12-11" }	1. Login sebagai Pengguna - Pengguna harus login dahulu lalu akan memperoleh access token. 2. Aprove Loan - Endpoint: POST `/loans/{loan_id}/approve` - **Input:** `loan_id` dari pinjaman yang ingin disetujui 3. Respons - **Jika sukses:** - Loan approved = true - Status loan berubah menjadi "BORROWED" - Response code: 200 OK
5.	Initiate Return (peminjam) Code Details 200 Response body { "detail": "Return initiated" } Response headers content-length: 29 content-type: application/json date: Thu,04 Dec 2025 02:12:49 GMT server: uvicorn	- **Peminjam** memulai pengembalian (POST /loans/{loan_id}/return): - Loan miliknya sendiri, status BORROWED → return_initiated = True. - Loan bukan miliknya →Forbidden 403. - Loan ID tidak ada → 404.
6.	Finalize Return(Admin)	- **Admin** memfinalisasi pengembalian (POST `/loans/{loan_id}/finalize-return`):

		- Loan sudah di-initiated → status = RETURNED, return_initiated = False. - Loan belum di-initiated → error 400. - Loan ID tidak ada → 404.
Test Case 3: Extend Loan		
1.	Extend Loan (Peminjam) Code Details 200 Response body { "detail": "Extension applied", "newDueDate": "2025-12-14" } Download Response headers content-length: 56 content-type: application/json date: Thu,04 Dec 2025 02:23:04 GMT server: uvicorn	- **Peminjam** meminta perpanjangan (POST /loans/{loan_id}/extend): ○ Loan miliknya → dueDate bertambah. ○ Loan bukan miliknya → Forbidden 403. ○ Loan ID tidak ada → 404.
Repository Github		
• URL repo: https://github.com/larashtm/TST-BookWise