

**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 Isi.....	2
Daftar Tabel.....	3
Implementasi Model Aggregate menjadi API.....	4

Daftar Tabel

Tabel 1.1. Implementasi berdasarkan layer	4
Tabel 1.2. API Surface	5
Tabel 1.3. Status Transition Matrix	5
Tabel 1.4. Implementasi BookWise	6

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, penggerjaan 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	<code>api/loan_router.py</code>	Router belum aktif sepenuhnya
	Schema	<code>schemas/loan_schema.py</code>	Struktur request/response untuk API.
Infrastructure Layer	Repository Implementation	<code>infrastructure/in_memory_loan_repository.py</code>	Implementasi penyimpanan.
	App Initialization	<code>main.py</code>	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
<code>/loans</code>	POST	Membuat permintaan peminjaman baru	{ "bookId": "uuid", "userId": "uuid" }
<code>/loans/{loan_id}/verify</code>	POST	Admin memverifikasi kelayakan peminjaman	<i>Tidak membutuhkan body</i>
<code>/loans/{loan_id}/approve</code>	POST	Admin menyetujui & mendistribusikan buku	<i>Tidak membutuhkan body</i>
<code>/loans/{loan_id}</code>	GET	Melihat detail loan termasuk status	<i>Tidak membutuhkan body</i>
<code>/loans/my</code>	GET	Peminjam melihat daftar pinjaman miliknya	<i>Tidak membutuhkan body</i>

/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.

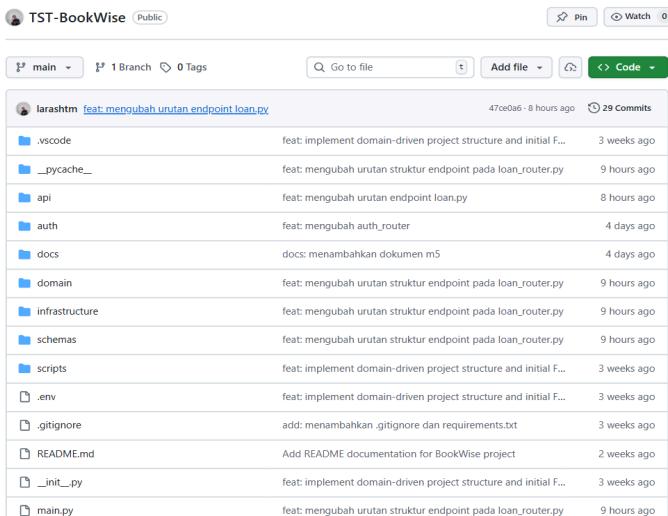
Tabel 1.3. Status Transition Matrix

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.		<ul style="list-style-type: none"> a. api/ memegang <i>use case level</i>, 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	<p>File: domain/loan.py</p> <p>Penjelasan:</p> <p>Loan berfungsi sebagai Aggregate Root yang menyimpan identitas peminjaman (LoanId) serta informasi penting lainnya seperti BookId, UserId, tanggal mulai dan tanggal jatuh tempo.</p> <pre> 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 dipinjam def borrow(self, due_date: DueDate): self.loanStatus = LoanStatus.BORROWED #status berubah jadi dipinjam 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 # 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 borrowed 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 </pre>
2.	Value Objects	

a	<pre> 1 from uuid import UUID 2 3 class BookId: 4 def __init__(self, value: UUID): 5 self.value = value 6 </pre>	<p>File: domain/book_id.py (Value Objects (VO)). Dipakai supaya buku lebih aman dan sesuai dalam format. Misal kirim req:</p> <pre> { "bookId": "e7d3d1dd-8fd8-4503-a4fa-b9f2407228c4" } </pre> <p>Di Api router, input akan diubah jadi: BookId(UUID("e7d3d1dd-8fd8-4503-a4fa-b9f2407228c4"))</p>
b.	<pre> 1 from uuid import UUID 2 3 class UserId: 4 def __init__(self, value: UUID): 5 self.value = value 6 </pre>	<p>File: domain/user_id.py (Value Objects (VO)). Value Object yang mewakili identitas unik seorang pengguna.</p>
c.	<pre> 1 from datetime import date 2 3 class DueDate: 4 def __init__(self, value: date): 5 self.value = value 6 7 def is_overdue(self): 8 return date.today() > self.value </pre>	<p>File: domain/due_date.py</p>
d.	<pre> 1 from enum import Enum 2 3 class LoanStatus(str, Enum): 4 REQUESTED = "requested" 5 BORROWED = "borrowed" 6 RETURNED = "returned" 7 OVERDUE = "overdue" 8 </pre>	<p>File: domain/loan_status.py</p> <ul style="list-style-type: none"> - 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
3.	Domain Server: LoanPolicyService	

	<pre> 1 from datetime import date, timedelta 2 from domain.due_date import DueDate 3 4 class LoanPolicyService: 5 def calculate_due_date(self): 6 return DueDate(date.today() + timedelta(days=7)) 7 </pre>	File: domain/loan_policy_service.py
4.	Implementasi Repository	
a.	<pre> 1 from abc import ABC, abstractmethod 2 from uuid import UUID 3 4 class LoanRepository(ABC): 5 6 @abstractmethod 7 def save(self, loan): 8 pass 9 10 @abstractmethod 11 def findById(self, id: UUID): 12 pass 13 14 @abstractmethod 15 def findByUser(self, user_id): 16 pass 17 </pre>	File: domain/loan_repository.py
b.	<pre> from domain.loan_repository import LoanRepository from uuid import UUID class InMemoryLoanRepository(LoanRepository): def __init__(self): # database palsu self.data = {} def save(self, loan): # jika udah ada update self.data[str(loan.loansId)] = loan def findById(self, id): # ambil pinjaman berdasarkan id # accept either uuid object or string key = str(id) return self.data.get(key) def findByUser(self, user_id): uid = str(user_id) if not hasattr(user_id, "value") else str(user_id.value) return [loan for loan in self.data.values() if str(loan.userId.value) == uid] def list_all(self): #mau kembalikan loan dlm bentuk list return list(self.data.values()) </pre>	File: infrastructure/in_memory_loan_repository.py
5.	Implementasi API	

a.	<pre> 1 from pydantic import BaseModel 2 from uuid import UUID 3 from typing import Optional 4 from datetime import datetime, date 5 6 class LoanCreateRequest(BaseModel): 7 bookId: UUID 8 userId: UUID 9 10 class LoanResponse(BaseModel): 11 loanId: UUID 12 bookId: UUID 13 userId: UUID 14 status: str 15 createdAt: datetime 16 dueDate: Optional[date] = None 17 verified: Optional[bool] = False 18 approved: Optional[bool] = False 19 return_initiated: Optional[bool] = False 20 </pre>	File: schemas/loan_schema.py
b.	<pre> 1 from fastapi import APIRouter, Depends, HTTPException, status 2 from uuid import UUID 3 from typing import List 4 from domain.loan import Loan 5 from domain.book_id import BookId 6 from domain.user_id import UserId 7 from domain.loan_policy_service import LoanPolicyService 8 from infrastructure.in_memory_loan_repository import InMemoryLoanRepository 9 from schemas.loan_schema import LoanCreateRequest, LoanResponse 10 from auth.deps import require_role, allow_roles, get_current_active_user 11 12 router = APIRouter(prefix="", tags=["Loans"]) #tags=[loans] buat ngelompokin endpoint di API 13 14 repo = InMemoryLoanRepository() 15 policy = LoanPolicyService() 16 17 def to_response(loan: Loan) -> dict: 18 return { 19 "loanId": loan.loanId, 20 "bookId": loan.bookId.value, 21 "userId": loan.userId.value, 22 "status": loan.loanStatus.value, 23 "createdAt": loan.createdAt, 24 "dueDate": loan.dueDate.value if loan.dueDate else None, 25 "verified": getattr(loan, "verified", False), 26 "approved": getattr(loan, "approved", False), 27 "return_initiated": getattr(loan, "return_initiated", False), 28 } 29 30 #endpoint create loan 31 @router.post("/loans", response_model=LoanResponse, status_code=201) 32 def create_loan(req: LoanCreateRequest, current_user = Depends(require_role("peminjam"))): 33 loan = Loan(bookId=req.bookId, UserId=req.userId) 34 due = policy.calculate_due_date() 35 # keep as REQUESTED so admin can verify/approve separately 36 repo.save(loan) 37 return to_response(loan) 38 39 #endpoint get loan by id (peminjam & pengguna) 40 @router.get("/loans/{loan_id}", response_model=LoanResponse) 41 def get_loan(loan_id: UUID, current_user = Depends(allow_roles("peminjam", "pengguna"))): 42 loan = repo.findById(loan_id) 43 if not loan: 44 raise HTTPException(status_code=404, detail="Loan not found") 45 # Access control: peminjam only sees own loans (additional check) 46 user = current_user 47 if user.role == "peminjam" and str(loan.userId.value) != str(user.user_id): 48 raise HTTPException(status_code=403, detail="Forbidden") 49 return to_response(loan) 50 </pre>	File: api/loan_router.py

```

    #endpoint list loans milik pengguna
    @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("/loan/{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()
            return {"detail": "loan verified"}
        except ValueError as e:
            raise HTTPException(status_code=400, detail=str(e))

    #admin approve loan (mewajibkan due date dan status borrowed)
    @router.post("/loan/{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))

    #endpoint mulai proses pengembalian
    @router.post("/loan/{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()
            return {"detail": "Return initiated"}
        except ValueError as e:
            raise HTTPException(status_code=400, detail=str(e))

    #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.	<pre> 1 from fastapi import FastAPI 2 from fastapi.openapi.models import APIKey, APIKeyIn, SecuritySchemeType 3 from fastapi.openapi.utils import get_openapi 4 from fastapi.security import HTTPBearer 5 from api.loan_router import router as loan_router 6 from auth.auth_router import router as auth_router 7 8 app = FastAPI(title="BookWise - Lending BC (with Auth)") 9 bearer_scheme = HTTPBearer() 10 11 @app.get("/") 12 def root(): 13 return {"message": "BookWise API is running"} 14 15 # Register routers 16 app.include_router(auth_router) 17 app.include_router(loan_router) 18 19 def custom_openapi(): 20 if app.openapi_schema: 21 return app.openapi_schema 22 23 openapi_schema = get_openapi(24 title="Bookwise - Lending BC (with Auth)", 25 version="1.0.0", 26 routes=app.routes, 27) 28 29 openapi_schema["components"]["securitySchemes"] = { 30 "BearerAuth": { 31 "type": "http", 32 "scheme": "bearer", 33 "bearerFormat": "JWT" 34 } 35 } 36 37 for path in openapi_schema["paths"]: 38 for method in openapi_schema["paths"][path]: 39 openapi_schema["paths"][path][method]["security"] = [40 {"BearerAuth": []} 41] 42 43 app.openapi_schema = openapi_schema 44 return app.openapi_schema 45 46 47 app.openapi = custom_openapi 48 </pre>	File: main.py
----	---	-------------------------------

OVERVIEW JWT SEDERHANA

1.	<p>Penjelasan JWT Sederhana</p> <p>JWT (<i>JSON Web Token</i>) adalah token berbentuk string yang mengklaim pengguna, ditandatangi menggunakan <i>secret key</i> tertentu. Setelah pengguna melakukan login, sistem membuat token yang berisi informasi dasar seperti <i>userId</i>. Token ini dikirimkan pada setiap request melalui header <i>Authorization</i>.</p> <p>Struktur JWT terdiri dari tiga bagian utama, yang dipisahkan oleh titik(.):</p> <ul style="list-style-type: none"> • Header → menentukan tipe token dan algoritma signing yang digunakan • Payload → berisi data pengguna. • Signature → hasil hashing yang memastikan token tidak dimodifikasi
2.	<p><i>Flow Login</i></p> <ol style="list-style-type: none"> 1. User mengirim <i>username & password</i> ke endpoint POST /auth/login. Sesuaikan pada database yang telah disusun pada kode <i>user.py</i>. 2. Sistem memverifikasi kecocokan <i>username</i> dan <i>password</i> sederhana dengan hash.

	<ol style="list-style-type: none"> 3. Jika valid → sistem membuat JWT dan mengembalikannya kepada pengguna/peminjam. 4. Pengguna mengirim token untuk setiap permintaan dengan format: <i>Authorization: Bearer <token></i> 5. Server memverifikasi token pada setiap request ke <i>endpoint</i> yang dilindungi 6. Jika token <i>valid</i> → akses diberikan 7. Jika token tidak <i>valid</i> atau hilang → server mengembalikan 401 Unauthorized
--	---

IMPLEMENTASI JWT AUTHENTICATION

1.	<pre> 1 from fastapi import APIRouter, Depends, HTTPException, status 2 from fastapi.security import OAuth2PasswordRequestForm 3 from pydantic import BaseModel 4 from datetime import timedelta 5 from typing import Dict 6 7 from auth.users import get_user_by_username, verify_password 8 from auth.jwt_handler import create_access_token, create_refresh_token 9 from auth.deps import get_current_active_user # ← WAJIB DITAMBAHKAN 10 11 router = APIRouter(prefix="/auth", tags=["Authentication"]) 12 13 REFRESH_TOKENS: Dict[str, str] = {} 14 15 class TokenResponse(BaseModel): 16 access_token: str 17 token_type: str = "bearer" 18 refresh_token: str 19 20 class RefreshRequest(BaseModel): 21 refresh_token: str 22 23 @router.post("/login", response_model=TokenResponse) 24 def login(form_data: OAuth2PasswordRequestForm = Depends()): 25 user = get_user_by_username(form_data.username) 26 if not user or not verify_password(form_data.password, user.hashed_password): 27 raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid credentials") 28 29 access_token = create_access_token(30 subject=str(user.user_id), 31 role=user.role, 32 expires_delta=timedelta(minutes=60) 33) 34 refresh_token = create_refresh_token() 35 36 REFRESH_TOKENS[refresh_token] = user.username 37 38 return TokenResponse(access_token=access_token, refresh_token=refresh_token) 39 40 </pre>	<p>File auth_router.py adalah tempat semua proses login dan keamanan di BookWise diatur. Beberapa endpoint:</p> <ul style="list-style-type: none"> - 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
----	---	---

```

@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.

```

auth > * deps.py > ⌂ get_current_active_user
1 from fastapi import Depends, HTTPException, status
2 from fastapi.security import OAuth2PasswordBearer
3 from typing import Callable
4 from auth_jwt_handler import decode_access_token
5 from auth.users import get_user_by_id
6
7 auth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")
8
9 def get_current_active_user(token: str = Depends(auth2_scheme)):
10     """
11     Return user object if token valid. Otherwise raise 401.
12     """
13     payload = decode_access_token(token)
14     if not payload:
15         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid or expired token")
16     user_id = payload.get("sub")
17     if not user_id:
18         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid token payload")
19     user = get_user_by_id(user_id)
20     if not user:
21         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="User not found")
22     if user.disabled:
23         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="User disabled")
24     # attach role available in payload too if needed
25     return user
26
27 def require_role(role: str) -> Callable:
28     """
29     Dependency generator: require_role('peminjam') -> use as Depends(require_role('peminjam'))
30     """
31     def role_checker(user = Depends(get_current_active_user)):
32         if user.role != role:
33             raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden: insufficient role")
34         return user
35     return role_checker
36
37 def allow_roles(*roles: str):
38     """
39     Dependency generator that allows any of listed roles
40     """
41     def checker(user = Depends(get_current_active_user)):
42         if user.role not in roles:
43             raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Forbidden: insufficient role")
44         return user
45     return checker

```

File **deps.py** berisi kumpulan dependency untuk memeriksa token dan mengatur hak akses setiap pengguna sebelum mereka boleh masuk ke endpoint tertentu. Didalamnya 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. auth > ⚡ jwt_handler.py > 🐄 create_access_token > 📁 expires_delta
  9 ALGORITHM = "HS256"
10 ACCESS_TOKEN_EXPIRE_MINUTES = 60 # 1 jam
11
12 def create_access_token(subject: str, role: str, expires_delta: Optional[timedelta] = None) -> str:
13     now = datetime.utcnow()
14     expire = now + (expires_delta or timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
15     payload: Dict[str, object] = {
16         "sub": str(subject),
17         "role": role,
18         "iat": now,
19         "exp": expire
20     }
21     token = jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
22     return token
23
24 def decode_access_token(token: str) -> Optional[Dict]:
25     try:
26         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
27         return payload
28     except JWTError:
29         return None
30
31 # simple refresh token utils (in-memory store handled in auth_router)
32 def create_refresh_token() -> str:
33     # just a unique string (UUID)
34     return str(uuid.uuid4())
35

```

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/kadaluwarsa.
- **create_refresh_token** → Buat token UUID untuk perbarui access token.

```

4. auth > ⚡ users.py > ...
  1 # auth/users.py
  2 from passlib.context import CryptContext
  3 from uuid import UUID, uuid4
  4 from typing import Optional, Dict
  5
  6 pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
  7
  8 class User:
  9     def __init__(self, user_id: UUID, username: str, hashed_password: str, role: str, disabled: bool=False):
10         self.user_id = user_id
11         self.username = username
12         self.hashed_password = hashed_password
13         self.role = role
14         self.disabled = disabled
15
16     def hash_password(password: str) -> str:
17         # Meng-hash password plaintext
18         return pwd_context.hash(password[:72])
19
20     def verify_password(plain_password: str, hashed_password: str) -> bool: # membandingkan password yang bakal diketik user(plain_password) dengan password yang sudah dihash (hashed_password)
21         return pwd_context.verify(plain_password[:72], hashed_password) #mengcek kecocokan
22
23     # Data pengguna (dummy / contoh)
24     # username: pengguna / password: pengguna123 (role: pengguna)
25     # username: peminjam / password: pinjam123 (role: peminjam)
26     _user1 = User(user_id=uuid4(), username="pengguna", hashed_password=hash_password("pengguna123"), role="pengguna")
27     _user2 = User(user_id=uuid4(), username="peminjam", hashed_password=hash_password("pinjam123"), role="peminjam")
28
29     # Disimpan berdasarkan username untuk pencarian cepat
30     USERS_DB: Dict[str, User] = {
31         _user1.username: _user1,
32         _user2.username: _user2,
33     }
34
35     def get_user_by_username(username: str) -> Optional[User]:
36         # Mengambil user berdasarkan username
37         return USERS_DB.get(username)
38
39     def get_user_by_id(user_id: str) -> Optional[User]:
40         # Mencari user berdasarkan ID
41         for u in USERS_DB.values():
42             if str(u.user_id) == str(user_id):
43                 return u
44         return None
45

```

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	<pre> 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 [15496] using StatReload INFO: Started server process [12296] INFO: Waiting for application startup. INFO: Application startup complete. </pre>	<ul style="list-style-type: none"> • 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.

BookWise - Lending BC (with Auth) 1.0.0 DAS 3.1

OpenAPI.json

Authorize

default

GET / Root

Authentication

POST /auth/login Login

POST /auth/refresh Refresh Token

POST /auth/logout Logout

GET /auth/me Me

Loans

POST /loans Create Loan

GET /loans/my List My Loans

GET /loans/all List All Loans

GET /loans/{loan_id} Get Loan

POST /loans/{loan_id}/verify Verify Loan

POST /loans/{loan_id}/approve Approve Loan

POST /loans/{loan_id}/return Initiate Return

POST /loans/{loan_id}/Finalize-return Finalize Return

POST /loans/{loan_id}/extend Extend Loan

Schemas

Body_Login_auth_login_post > Expand all object

ExtendRequest > Expand all object

HTTPValidationError > Expand all object

LoanCreateRequest > Expand all object

LoanResponse > Expand all object

RefreshRequest > Expand all object

TokenResponse > Expand all object

ValidationError > Expand all object

	Respon Swagger	Alur Ringkas
--	----------------	--------------

Test Case 1: Authentication

1. Login sebagai Peminjam

Code Details

200 Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInRcCjE6IkpXVCJ9.eyJzdWJlIDJ1V2FwZ2x0Sbw0TcL1TRjOTU0D2Z1iNmF1ZTFmZDVjM6EiIjybzx1IjicGvtawSqW8LLCjpxYQloIENjQ4NDEsM02s1mAv4cC18Ht2NDc0Bqgk6g-f1k81X1gb7sA-wT3uDYs1hs58CQzy88a7-35nH",
  "token_type": "bearer",
  "refresh_token": "53fe4472-33ba-45c2-b179-8e61fefe307b"
}
```

[Copy](#) [Download](#)

Login sebagai Pengguna

Code Details

200 Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInRcCjE6IkpXVCJ9.eyJzdWJlIDJ1V2FwZ2x0Sbw0TcL1TRjOTU0D2Z1iNmF1ZTFmZDVjM6EiIjybzx1IjicGvtawSqW8LLCjpxYQloIENjQ4NDEsM02s1mAv4cC18Ht2NDc0Bqgk6g-f1k81X1gb7sA-wT3uDYs1hs58CQzy88a7-35nH",
  "token_type": "bearer",
  "refresh_token": "77957d8-64b0-4ec3-cf52-246f29fc"
}
```

[Copy](#) [Download](#)

1. Kirim Kredensial

Peminjam/Pengguna login dengan username/password ke /auth/login.

2. Verifikasi

Server cek kredensial di database. Jika valid, lanjut ke token.

3. Generate Token

Server membentuk:

- **access_token** → JWT berisi sub (user ID), role (peminjam), iat, exp
- **refresh_token** → UUID unik
- **token_type** → "bearer"

4. Response ke Client

```
{
  "access_token": "...JWT...",
  "token_type": "bearer",
  "refresh_token": "...UUID..."
}
```

5. Penggunaan

Client pakai access_token di header Authorization untuk akses endpoint yang butuh login.

2. Refresh token

Code Details

200 Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInRcCjE6IkpXVCJ9.eyJzdWJlIDJ1V2FwZ2x0Sbw0TcL1TRjOTU0D2Z1iNmF1ZTFmZDVjM6EiIjybzx1IjicGvtawSqW8LLCjpxYQloIENjQ4NDEsM02s1mAv4cC18Ht2NDc0Bqgk6g-f1k81X1gb7sA-wT3uDYs1hs58CQzy88a7-35nH",
  "token_type": "bearer",
  "refresh_token": "b54c287a-b26e-4dee-b4f5-c262a0714463"
}
```

[Copy](#) [Download](#)

Response headers

```
content-length: 388
content-type: application/json
date: Thu, 04 Dec 2025 10:07:50 GMT
server: uvicorn
```

1. Client Kirim Refresh Token

- Client (aplikasi) mengirim request ke endpoint refresh, /auth/refresh.
- Body request hanya berisi:

```
{
  "refresh_token": "string"
}
```

```
}
```

- 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":  
        "eyJhbGciOiJIUzI1NiIsInR  
        5cCI6IkpXVCJ9.eyJzdWIoI  
        JiY2FmY2UxOS0wOTc1LTrjOT  
        UtODZjZi1hNmFIZTFmZDVjMm  
        EiLCJyb2xlijoicGVtaW5qYW  
        0iLCJpYXQiOjE3NjQ4NDI4Nz  
        EsImV4cCI6MTc2NDg0NjQ3MX  
        0.rUGNO4YWq9nF6Eb7kNuUea  
        ZvSlmpqmVfoqTN6EBsQ1E",
```

```
    "token_type": "bearer",
```

```
    "refresh_token":  
        "b54c287a-b26e-4dee-b4f5  
        -c26240714463"
```

```
}
```

Penjelasan payload:

- **access_token** → JWT baru,

		<p>dipakai untuk request API selanjutnya.</p> <ul style="list-style-type: none"> - token_type → "bearer", sama seperti login. - refresh_token → refresh token baru, disimpan untuk permintaan refresh berikutnya. <p>5. Penggunaan</p> <ul style="list-style-type: none"> - Client simpan access token baru dan pakai untuk akses API. - Refresh token lama biasanya tidak berlaku lagi jika server mengeluarkan token baru. 				
3.	Logout	<p>Code Details</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">200</td> <td> Response body <pre>{ "message": "Refresh token revoked" }</pre> Response headers <pre>content-length: 35 content-type: application/json date: Thu, 04 Dec 2025 10:22:43 GMT server: unicorn</pre> </td> </tr> <tr> <td colspan="2">Responses</td> </tr> </table> <p>1. Client Kirim Refresh Token</p> <ul style="list-style-type: none"> - Endpoint: misal /auth/logout. - Body request berisi refresh token yang tersimpan: <pre>{ "refresh_token": "string" }</pre> <p>"string" diganti dengan refresh token.</p> <p>2. Server Revoke Refresh Token</p> <ul style="list-style-type: none"> - Server mengecek refresh token: <ul style="list-style-type: none"> o Jika valid → token dicabut (revoked), tidak bisa dipakai lagi untuk generate access token baru. o Jika tidak valid → bisa return error (misal 401 atau 400). <p>3. Response</p> <ul style="list-style-type: none"> - Server memberi konfirmasi logout: <pre>{ "message": "Refresh token revoked"</pre>	200	Response body <pre>{ "message": "Refresh token revoked" }</pre> Response headers <pre>content-length: 35 content-type: application/json date: Thu, 04 Dec 2025 10:22:43 GMT server: unicorn</pre>	Responses	
200	Response body <pre>{ "message": "Refresh token revoked" }</pre> Response headers <pre>content-length: 35 content-type: application/json date: Thu, 04 Dec 2025 10:22:43 GMT server: unicorn</pre>					
Responses						

		<pre> } - Status code: 200 OK - Menandakan refresh token berhasil dicabut dan user tidak bisa refresh lagi tanpa login ulang. </pre>
4.	Informasi Profil	<p>Client kirim request ke /auth/me dengan header Authorization: Authorization: Bearer <access_token></p> <ol style="list-style-type: none"> Client kirim request ke /auth/me dengan header Authorization: Authorization: Bearer <access_token> Server membaca token dan memverifikasi: <ul style="list-style-type: none"> Apakah token valid? Apakah belum kadaluarsa? Jika valid, server mengembalikan data user terkait token itu, misal: <pre> { "user_id": "044ad666-a833-4c93-be38-d4221be29c7c", "username": "pengguna1", "role": "pengguna", "disabled": false } </pre>
Test Case 2: Loans		
1.	Create Loan Sesuai(Peminjam)	<p>1. Pastikan User Sudah Login dan Terautentikasi</p> <ul style="list-style-type: none"> Hanya peminjam yang bisa membuat Loan Endpoint: misal /auth/logout. Client harus pakai access token dari login di header:

Code	Details
201	<p>Response body</p> <pre>{ "loanId": "ab439f55-e681-4525-8403-e42bc0ef9fd8", "bookId": "3fa85ff64-5717-4562-b3fc-2c963f66af78", "userId": "bcacf19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T10:50:25.610158", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre> <p>Response headers</p> <pre>content-length: 281 content-type: application/json date: Thu, 04 Dec 2025 10:50:25 GMT server: unicorn</pre>

Create Loan Ga Sesuai

Code	Details
422	<p>Error: Unprocessable Entity</p> <p>Response body</p> <pre>{ "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" }] }] }</pre> <p>Response headers</p> <pre>content-length: 254 content-type: application/json date: Thu, 04 Dec 2025 10:55:56 GMT server: unicorn</pre>

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"
              }
            ]
          }
```
- Jika Valid
 - Server buat loan baru dengan status awal "requested", belum diverifikasi atau disetujui:


```
{
```

		<pre> "loanId": "a573a123-0891-49d0-b956-4883ea474042", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af78", "userId": "bcacfce19-0975-4c95-86cf-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 </pre>								
2.	<p>Get Loan (My) (Peminjam)</p> <table border="1"> <thead> <tr> <th>Code</th><th>Details</th></tr> </thead> <tbody> <tr> <td>200</td><td> <p>Response body</p> <pre>{ "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre> </td></tr> </tbody> </table> <p>Get Loan (All) (Peminjam)</p> <table border="1"> <thead> <tr> <th>Code</th><th>Details</th></tr> </thead> <tbody> <tr> <td>403 <small>Undocumented</small></td><td> <p>Error: Forbidden</p> <p>Response body</p> <pre>{ "detail": "Forbidden: insufficient role" }</pre> <p>Response headers</p> <pre>content-length: 41 content-type: application/json date: Thu, 04 Dec 2025 11:27:40 GMT server: unicorn</pre> </td></tr> </tbody> </table> <p>Get Loan (All) (Pengguna)</p>	Code	Details	200	<p>Response body</p> <pre>{ "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>	Code	Details	403 <small>Undocumented</small>	<p>Error: Forbidden</p> <p>Response body</p> <pre>{ "detail": "Forbidden: insufficient role" }</pre> <p>Response headers</p> <pre>content-length: 41 content-type: application/json date: Thu, 04 Dec 2025 11:27:40 GMT server: unicorn</pre>	<p>1. Get Loan Detail</p> <ul style="list-style-type: none"> • Endpoint: GET /loans/{loan_id} • Input: loan_id yang diperoleh saat create loan. • Role: harus peminjam. • Output: 200 OK dengan payload detail pinjaman: <pre>{ "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>
Code	Details									
200	<p>Response body</p> <pre>{ "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>									
Code	Details									
403 <small>Undocumented</small>	<p>Error: Forbidden</p> <p>Response body</p> <pre>{ "detail": "Forbidden: insufficient role" }</pre> <p>Response headers</p> <pre>content-length: 41 content-type: application/json date: Thu, 04 Dec 2025 11:27:40 GMT server: unicorn</pre>									

	<p>200 Response body</p> <pre> "loanId": "ab439f55-e681-4525-8403-e42bc0ef9fd8", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af78", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T10:50:25.610158", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }, { "loanId": "a573a123-0891-49d0-b956-4883ea474042", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af78", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:05:12.6461", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }, { "loanId": "48747bf2-3721-4c2d-a271-5e85a05f271a", "bookId": "3fa85f64-5717-4562-b3fc-2c963f66af6", "userId": "bcacfce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:05:15.126461", "dueDate": null }] </pre>	<pre> "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 </pre>				
	<p>403 Error: Forbidden Undocumented</p> <p>Response body</p> <pre> { "detail": "Forbidden: insufficient role" } </pre> <p>Response headers</p> <pre> content-length: 41 content-type: application/json date: Thu, 04 Dec 2025 12:08:53 GMT server: uvicorn </pre>	<p>2. Get All Loans</p> <ul style="list-style-type: none"> Endpoint: GET /loans/all Input: tidak perlu input apapun. Role: peminjam tidak diperbolehkan, hanya admin atau role tertentu bisa akses. Output: 403 Forbidden 				
3.	<p>Verify Loan(Pengguna)</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Details</th> </tr> </thead> <tbody> <tr> <td>200</td> <td> <p>Response body</p> <pre> { "detail": "Loan verified" } </pre> </td></tr> </tbody> </table>	Code	Details	200	<p>Response body</p> <pre> { "detail": "Loan verified" } </pre>	<ol style="list-style-type: none"> Login sebagai Pengguna <ul style="list-style-type: none"> Pengguna harus login dahulu lalu akan memperoleh access token. Gunakan token untuk akses endpoint. Verify Loan <ul style="list-style-type: none"> Endpoint: POST /loans/{loan_id}/verify Input: loan_id pinjaman yang ingin diverifikasi.
Code	Details					
200	<p>Response body</p> <pre> { "detail": "Loan verified" } </pre>					

	<table border="1"> <thead> <tr> <th>Code</th><th>Details</th></tr> </thead> <tbody> <tr> <td>404 <i>Undocumented</i></td><td> Error: Not Found Response body <pre>{ "detail": "Loan not found" }</pre> Response headers <pre>content-length: 27 content-type: application/json date: Thu,04 Dec 2025 12:23:40 GMT server: uvicorn</pre> </td></tr> </tbody> </table>	Code	Details	404 <i>Undocumented</i>	Error: Not Found Response body <pre>{ "detail": "Loan not found" }</pre> Response headers <pre>content-length: 27 content-type: application/json date: Thu,04 Dec 2025 12:23:40 GMT server: uvicorn</pre>	<p>Syarat server:</p> <ul style="list-style-type: none"> - Loan harus dalam status "REQUESTED" - Hanya admin yang boleh memverifikasi <p>3. Respons:</p> <ul style="list-style-type: none"> - Jika sukses akan menunjukkan 200 ok - Jika loan dalam status REQUESTED artinya 400 Bad - Jika loan_id tidak ada maka akan menampilkan 404 not found.
Code	Details					
404 <i>Undocumented</i>	Error: Not Found Response body <pre>{ "detail": "Loan not found" }</pre> Response headers <pre>content-length: 27 content-type: application/json date: Thu,04 Dec 2025 12:23:40 GMT server: uvicorn</pre>					
4.	Approve Loan (Pengguna) <table border="1"> <thead> <tr> <th>Code</th><th>Details</th></tr> </thead> <tbody> <tr> <td>200</td><td> Response body <pre>{ "detail": "Loan approved", "dueDate": "2025-12-11" }</pre> </td></tr> </tbody> </table>	Code	Details	200	Response body <pre>{ "detail": "Loan approved", "dueDate": "2025-12-11" }</pre>	<ol style="list-style-type: none"> 1. Login sebagai Pengguna <ul style="list-style-type: none"> - Pengguna harus login dahulu lalu akan memperoleh access token. 2. Aprove Loan <ul style="list-style-type: none"> - Endpoint: POST /loans/{loan_id}/approve - Input: loan_id dari pinjaman yang ingin disetujui <p>3. Respons</p> <ul style="list-style-type: none"> - Jika sukses: <ul style="list-style-type: none"> - Loan approved = true - Status loan berubah menjadi "BORROWED" - Response code: 200 OK
Code	Details					
200	Response body <pre>{ "detail": "Loan approved", "dueDate": "2025-12-11" }</pre>					
5.	Initiate Return (peminjam) <table border="1"> <thead> <tr> <th>Code</th><th>Details</th></tr> </thead> <tbody> <tr> <td>200</td><td> Response body <pre>{ "detail": "Return initiated" }</pre> Response headers <pre>content-length: 29 content-type: application/json date: Thu,04 Dec 2025 02:12:49 GMT server: uvicorn</pre> </td></tr> </tbody> </table>	Code	Details	200	Response body <pre>{ "detail": "Return initiated" }</pre> Response headers <pre>content-length: 29 content-type: application/json date: Thu,04 Dec 2025 02:12:49 GMT server: uvicorn</pre>	<ul style="list-style-type: none"> - Peminjam memulai pengembalian (POST /loans/{loan_id}/return): <ul style="list-style-type: none"> - Loan miliknya sendiri, status BORROWED → return_initiated = True. - Loan bukan miliknya →Forbidden 403. - Loan ID tidak ada → 404.
Code	Details					
200	Response body <pre>{ "detail": "Return initiated" }</pre> Response headers <pre>content-length: 29 content-type: application/json date: Thu,04 Dec 2025 02:12:49 GMT server: uvicorn</pre>					
6.	Finalize Return(Admin)	<ul style="list-style-type: none"> - Admin memfinalisasi pengembalian (POST /loans/{loan_id}/finalize-return): 				

		<ul style="list-style-type: none"> - 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: 54
Content-Type: application/json
date: Thu, 04 Dec 2025 02:23:04 GMT
server: unicorn
```

- **Peminjam** meminta perpanjangan (POST /loans/{loan_id}/extend):

- o Loan miliknya → dueDate bertambah.
- o Loan bukan miliknya → Forbidden 403.
- o Loan ID tidak ada → 404.

Repository Github

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

The screenshot shows the GitHub repository page for 'TST-BookWise'. The repository has 17 commits, 0 stars, and 0 forks. The commit history includes updates to the auth_router, domain structure, and endpoint implementation. The repository has 17 commits, 0 stars, and 0 forks.