

**TUGAS BESAR TAHAP - 6 LAPORAN FINAL *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 Gambar.....	3
Daftar Tabel.....	4
1. Deskripsi Umum.....	5
1.1. Latar Belakang.....	5
1.2. Deskripsi Sistem.....	5
1.3. Tujuan Sistem.....	5
2. Business Capability.....	6
2.1.1 Lending Management.....	6
2.1.2 Notification System.....	7
2.1.3 Book Management.....	7
2.1.4 User Management.....	7
3. Domain.....	7
4. Bounded Context.....	10
5. Arsitektur Aplikasi dan Implementasi.....	13
6. Implementasi Model Aggregate menjadi API.....	17

Daftar Gambar

Gambar 2.1 Business Capability BookWise	6
Gambar 4.1 Bounded Context BookWise	10
Gambar 5.4 Class Diagram Book Wise	16

Daftar Tabel

Tabel 3.1. Domain	7
Tabel 4.1 Lingkup (Boundary) Setiap Bounded Context)	10
Tabel 4.2 Pola Integrasi Antar Bounded Context	11
Tabel 5.1 Identifikasi Class Diagram	13
Tabel 5.2 Glosarium Ubiquitous Language (Loan BC)	14
Tabel 5.3 Relasi antar Entity/Value Object	15
Tabel 6.1. Implementasi berdasarkan layer	17
Tabel 6.2. API Surface	18
Tabel 6.3. Implementasi BookWise	18

1. Deskripsi Umum

1.1. Latar Belakang

Seiring dengan pesatnya perkembangan teknologi digital di era modern, kebutuhan masyarakat terhadap akses literatur yang cepat, mudah, dan efisien semakin meningkat. Aktivitas membaca dan mencari referensi kini tidak lagi terbatas pada ruang fisik perpustakaan, melainkan telah beralih menuju ekosistem digital yang lebih *fleksibel* dan dinamis. Namun, perpustakaan tradisional masih menghadapi berbagai kendala, seperti keterbatasan kapasitas penyimpanan, jam operasional yang terbatas, serta biaya perawatan yang tinggi. Kondisi tersebut seringkali menghambat proses pembelajaran dan pencarian informasi, terutama bagi pengguna yang memiliki keterbatasan waktu dan lokasi.

BookWise hadir sebagai solusi inovatif berupa platform perpustakaan digital yang memungkinkan pengguna untuk meminjam dan mengelola buku secara daring. Sistem ini dirancang dengan prinsip kemudahan akses dan efisiensi, sehingga pengguna dapat menikmati pengalaman pinjaman tanpa batasan waktu maupun tempat.

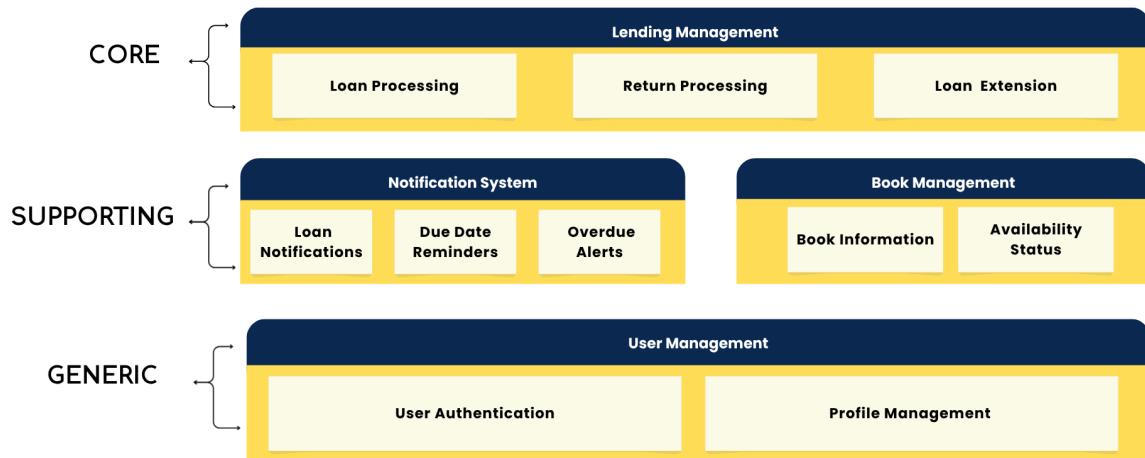
1.2. Deskripsi Sistem

BookWise merupakan platform peminjaman buku digital yang memungkinkan pengguna untuk mempermudah pengajuan dan menerima buku sesuai prosedur. Sistem ini juga mendukung proses pengembalian buku bisa dikelola dengan baik dan memberikan layanan perpanjangan peminjaman yang sah dan tercatat. Dalam sistemnya, terdapat dua peran utama dalam sistem yaitu peminjam dan pengguna, yang setiap perannya mempunyai akses dan fungsi sesuai kebutuhan.

1.3. Tujuan Sistem

1. Mempermudah peminjam dalam pinjaman.
2. Mampu memverifikasi kelayakannya, dan menerima buku sesuai prosedur.
3. Memastikan proses pengembalian berjalan lancar.
4. Memberikan perpanjangan peminjaman yang sah.

2. Business Capability



Gambar 2.1 Business Capability BookWise

Sistem BookWise terbagi menjadi empat kapabilitas utama yang mewakili area fungsional berbeda di dalam arsitektur. Setiap kapabilitas memiliki tanggung jawab jelas dan bekerja melalui integrasi antar bounded context. Pada tahap desain taktis (M3), hanya kapabilitas *Lending Management* yang dimodelkan dalam bentuk *class diagram*, sedangkan kapabilitas lainnya berada di bounded context eksternal.

2.1.1 *Lending Management*

Singkatnya, *Lending Management* adalah kapabilitas inti untuk proses peminjaman buku. Fokusnya mencakup seluruh alur mulai dari peminjaman, di mana peminjam mengajukan pinjaman dan sistem memverifikasi kelayakan serta validitas data, kemudian menyalurkan buku ke peminjam. Selain itu, *Lending Management* juga menangani pengembalian buku, termasuk memulai proses pengembalian, memeriksa kondisi buku dan status pembayaran, serta memperbarui status resmi untuk menutup proses. Kapabilitas ini juga mendukung loan extension, dengan memastikan perpanjangan peminjaman sah dan tercatat secara resmi.

2.1.2 *Notification System*

Notification System adalah kapabilitas pendukung yang mengirimkan notifikasi kepada pengguna terkait aktivitas peminjaman. Aktivitas utamanya meliputi notifikasi pinjaman baru, pengingat tanggal jatuh tempo, dan pemberitahuan keterlambatan. Notification System berada pada bounded context terpisah dan hanya mengonsumsi event yang dipublikasikan oleh Lending Management, misalnya *LoanCreated*, *LoanReturned*, atau *LoanOverdue*.

2.1.3 Book Management

Sementara itu, kapabilitas pendukung lainnya yaitu Book Management dipakai untuk pengelolaan informasi buku, mulai dari detail buku (judul, pengarang, deskripsi), pengelolaan ketersediaan buku, perubahan metadata buku oleh admin. Informasi buku ini ditampilkan kepada pengguna sebelum mereka melakukan peminjaman.

2.1.4 User Management

Sebagai subdomain terakhir, terdapat User Management yang berfungsi untuk pengelola data dan akses pengguna. User Management hanya berinteraksi melalui *UserId*, sehingga tidak dimodelkan pada desain taktis. Seluruh detail pengguna disimpan pada *bounded context* eksternal ini.

Berdasarkan pembagian kapabilitas di atas, tahap desain dan implementasi sistem BookWise berfokus pada Lending Management sebagai *bounded context* utama yang dimodelkan secara detail. Sementara, Notification System, Book Management, dan User Management sebagai layanan eksternal yang berinteraksi dengan Lending Management melalui referensi atau event, sehingga tidak dimodelkan secara internal dalam tahap desain taktis.

3. Domain

Tabel 3.1. Domain

Jenis Domain	Capability	Subdomain	Deskripsi
Core Domain	<i>Lending Management</i>	<i>Loan Processing</i>	Mengatur proses peminjaman buku digital, termasuk pembuatan pinjaman baru, validasi, dan

			pencatatan tanggal jatuh tempo. Subdomain ini adalah pusat logika inti aplikasi.
		<i>Return Processing</i>	Mengelola proses pengembalian buku, mengecek status buku, memeriksa buku yang dipinjam, dan mengupdate kondisi peminjaman.
		<i>Loan Extension</i>	Menangani permintaan perpanjangan masa pinjam.
Supporting Domain	<i>Notification System</i>	<i>Loan Notifications</i>	Mengirimkan notifikasi ketika pinjaman dibuat, diperpanjang, atau dikembalikan. Mengonsumsi <i>event</i> dari <i>Lending</i> .
		<i>Due Date Reminders</i>	Mengirim pengingat otomatis kepada pengguna sebelum jatuh tempo.
		<i>Overdue Alerts</i>	Mengirim peringatan kepada pengguna saat pinjaman melewati tanggal jatuh tempo.
	<i>Book Management</i>	<i>Book Information</i>	Mengelola detail buku seperti judul, penulis, kategori, deskripsi, dan metadata lainnya.

		<i>Availability Status</i>	Mengatur status ketersediaan buku (tersedia, sedang dipinjam, tidak aktif), digunakan sebelum proses peminjaman.
Generic Domain	<i>User Management</i>	<i>User Authentication</i>	Mengatur login, akses pengguna, dan validasi identitas. Tidak terhubung langsung ke logika <i>Lending</i> .
		<i>Profile Management</i>	Menyimpan dan memperbarui data profil pengguna. Lending hanya memakai <i>UserId</i> sebagai referensi.

Berdasarkan tabel diatas, komponen - komponen utama dalam domain BookWise. dapat diuraikan sebagai berikut. Pendekatan ini menerapkan prinsip Domain-Driven Design, di mana subdomain menjadi acuan utama dalam menyelesaikan masalah, sedangkan bounded context menentukan batasan dan model yang digunakan dalam implementasi.

Loan Processing, sebagai *core domain*, menjadi *bounded context* utama, yaitu *Lending / Loan Processing BC*, yang menjadi pusat dari desain dan implementasi *final project*. Sementara itu, tiga subdomain lainnya berperan sebagai *bounded context* eksternal:

1. *Catalog / Book Management BC* menyediakan model buku yang digunakan *Lending BC*.
2. *User Management BC* menyediakan identitas dan data pengguna.
3. *Notification BC* mengonsumsi *event domain* dari *Lending BC* untuk mengirimkan pemberitahuan otomatis.

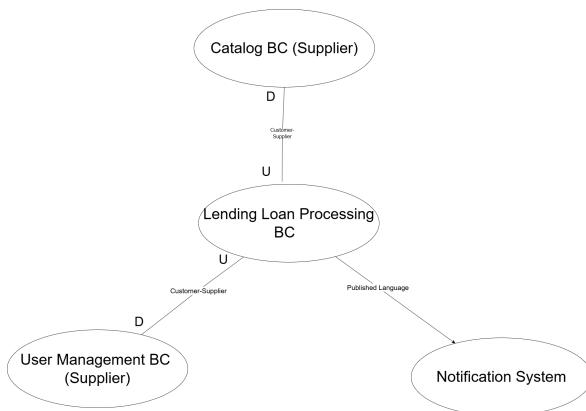
4. Bounded Context

Dalam pendekatan Domain-Driven Design (DDD), konsep Bounded Context dipakai untuk menegaskan setiap domain memiliki cakupan operasional, aturan, dan bahasa internal yang tidak bercampur dengan konteks lain. Pemisahan ini

memudahkan pengelolaan arsitektur, karena setiap konteks bisa berkembang tanpa beririsan.

Diagram *Bounded Context* memperlihatkan bagaimana domain-domain tersebut saling terhubung. Beberapa konteks berperan sebagai penyedia informasi dasar, beberapa menangani proses transaksi, dan yang lain memberikan layanan pendukung. Dengan menyusun semua hubungan disusun menggunakan pola integrasi yang tepat, terbentuk alur yang konsisten untuk mendukung layanan utama BookWise.

Untuk memudahkan pemahaman, berikut ini digambarkan Bounded Context dari BookWise, yang menunjukkan bagaimana masing-masing domain berinteraksi dan memiliki batas tanggung jawabnya.



Gambar 4.1 Bounded Context BookWise

Berdasarkan ilustrasi di atas, Tabel 1.1 merinci setiap Bounded Context, menjelaskan apa yang di-handle di masing-masing BC serta apa yang dipegang oleh BC lain.

Tabel 4.1 Lingkup (*Boundary*) Setiap *Bounded Context*)

Bounded Context	Apa yang di-handle BC ini	Apa yang dipegang BC lain
Lending Loan Processing BC (Core)	<i>loanStatus, dueDate, borrowedId, bookId; behavior (createLoan, verifyLoan, distributeBook, initiateReturn, verifyReturn, finalizeReturn, verifyLoanExtension)</i>	- <i>bookId, availabilityStatus</i> dari Book Management BC - <i>userId</i> dari User Management BC

		- Event LoanCreated, LoanReturned, LoanOverdue dikonsumsi Notification BC
Catalog BC (Supplier)	bookId, title, author, category, description, metadata, availabilityStatus; behavior (updateMetadata, checkAvailability, markAsBorrowed, markAsReturned, deactivateBook)	- Digunakan oleh Lending BC (bookId, availabilityStatus)
Notification System	notificationId, userId, eventType, message, status, createdAt; behavior (sendNotification, formatMessage, markAsRead, retrySend)	- Mengonsumsi event dari Lending BC (LoanCreated, LoanReturned, LoanOverdue) - Menggunakan userId dari User Management BC
User Management BC (Supplier)	userId, username, passwordHash, role, profile, lastLogin; behavior (authenticate, updateProfile, changePassword, getUserId)	- Referensi userId digunakan oleh Lending BC dan Notification BC

Setelah memahami batas tanggung jawab tiap BC, Tabel 4.2 menjelaskan pola integrasi antar Bounded Context, termasuk siapa yang menjadi sumber dan target, jenis pola integrasi, serta makna hubungan antar BC.

Tabel 4.2 Pola Integrasi Antar *Bounded Context*

Source BC	Target BC	Pattern	Makna Relasi
Lending Loan Processing BC (Core)	<i>Catalog BC (Supplier)</i>	Customer–Supplier (Reference Model)	Lending BC membutuhkan data buku (bookId, availabilityStatus) untuk peminjaman, tapi tidak

			menyimpan detail buku. Book Management menyediakan data dan struktur buku.
Lending Loan Processing BC (Core)	<i>User Management BC (Supplier)</i>	Customer–Supplier (Reference Model)	Lending BC membutuhkan userId untuk memverifikasi peminjam. User Management menyediakan identitas dan data profil pengguna, Lending hanya menggunakan ID sebagai referensi.
Lending Loan Processing BC (Core)	<i>Notification System</i>	<i>Published Language / Domain Event</i>	Lending BC mem-publish event seperti LoanCreated, LoanReturned, LoanOverdue. Notification BC mendengarkan event ini dan memprosesnya untuk mengirim notifikasi kepada pengguna.

Pemisahan konteks menegaskan batas tanggung jawab masing-masing domain, sekaligus memperjelas peran tiap BC dalam sistem BookWise. Catalog / Book Management dan User Management berfungsi sebagai penyedia informasi,

masing-masing menyimpan data buku dan identitas pengguna. Lending menggunakan informasi untuk memproses peminjaman, pengembalian, dan perpanjangan, namun hanya memakai referensi berupa BookId dan UserId, sehingga tidak perlu mengetahui detail data oleh kedua konteks.

Berbeda dengan itu, hubungan antara Lending dan Notification tidak menggunakan model bersama. Lending menerbitkan event sebagai bahasa komunikasi, yang kemudian dipahami dan diproses oleh Notification.

Seperti yang dibahas pada milestone sebelumnya, tahap pengembangan saat ini masih berfokus pada penguatan logika domain inti, yaitu Lending, sebelum memperluas integrasi dengan BC lainnya.

5. Arsitektur Aplikasi dan Implementasi

Class diagram BookWise disusun berdasarkan prinsip *Domain-Driven Design (DDD)* dengan mengidentifikasi entitas utama, *value object*, serta hubungan antar kelas yang mencerminkan proses bisnis pada sistem BookWise. Berikut adalah identifikasi kelas-kelas yang terdapat dalam sistem BookWise beserta jenis, atribut, dan methodnya:

Tabel 5.1 *Identifikasi Class Diagram*

Nama Kelas	Jenis Kelas	Atribut (Tipe Data)	Method (Acces)
Loan	Entity	borrowedId: String (private) bookId: String (private) userId: String (private) loanStatus: Enum (private) dueDate: Date (private)	-
LoanProcessingService	Service	-	createLoan() (public) verifyLoan() (public) distributeBook() (public)
ReturnProcessingService	Service	-	initiateReturn() (public) verifyReturn() (public) finalizeReturn() (public)
LoanExtensionService	Service	-	verifyLoanExtension()

Berdasarkan Tabel 5.1, struktur domain Loan menjadi jelas. Loan sebagai aggregate root menjaga state peminjaman melalui behavior yang ada, memastikan proses peminjaman, pengembalian, dan perpanjangan sesuai alur BookWise. Atribut entity dibuat private supaya data internal terlindungi, dan setiap perubahan hanya bisa dilakukan melalui method resmi dari Lending Loan Processing BC. Hal ini menjamin konsistensi data dan memastikan status pinjaman selalu valid.

Tabel 5.2 *Glosarium Ubiquitous Language (Loan BC)*

Istilah	Definisi / Penjelasan
Loan	Entitas yang merepresentasikan pinjaman buku oleh pengguna.
BorrowedId	Identifier unik transaksi pinjaman.
BookId	Identifier unik buku yang dipinjam.
UserId	Identifier unik pengguna yang meminjam buku.
LoanStatus	Status pinjaman, misal PENDING, APPROVED, RETURNED, OVERDUE.
DueDate	Tanggal jatuh tempo pengembalian buku.
LoanProcessingService	Service yang mengatur proses pengajuan, verifikasi, dan penyaluran pinjaman.
ReturnProcessingService	Service yang mengatur proses pengembalian buku dan penyelesaian pinjaman.
LoanExtensionService	Service yang mengatur proses perpanjangan peminjaman buku.
CreateLoan	Method untuk mengajukan pinjaman baru.
VerifyLoan	Method untuk memverifikasi kelayakan pinjaman.
DistributeBook	Method untuk menyalurkan buku ke peminjam setelah disetujui.

InitiateReturn	Method untuk memulai proses pengembalian buku.
VerifyReturn	Method untuk memeriksa kondisi buku dan status pengembalian.
FinalizeReturn	Method untuk menyelesaikan dan mencatat pengembalian buku.
VerifyLoanExtension	Method untuk memverifikasi dan menyetujui perpanjangan pinjaman.

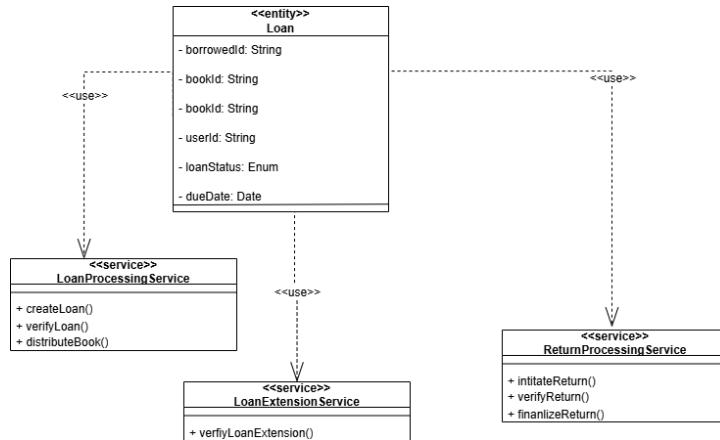
Tabel 5.2 menampilkan glosarium Ubiquitous Language dalam Loan Bounded Context, yang menjamin istilah konsisten dan class diagram sesuai proses bisnis. Setelah istilah utama dijelaskan, langkah berikutnya adalah memeriksa relasi antar entitas, yang dirangkum pada tabel berikut.

Tabel 5.3 Relasi antar *Entity/Value Object*

Relasi	Tipe Relasi UML	Penjelasan
<i>Loan</i> → <i>LoanProcessingService</i>	Usage(Dependency)	Service bergantung pada <i>Loan</i> untuk proses pinjaman.
<i>Loan</i> → <i>ReturnProcessingService</i>	Usage(Dependency)	Service bergantung pada <i>Loan</i> untuk proses pengembalian.
<i>Loan</i> → <i>LoanExtensionService</i>	Usage(Dependency)	Service bergantung pada <i>Loan</i> untuk memverifikasi perpanjangan.

Dari relasi pada Tabel 5.3 terlihat bahwa semua service bergantung pada entity *Loan* melalui dependency. *Loan* menjadi pusat yang mengendalikan state, sementara service hanya menggunakan entity untuk menjalankan behavior, tanpa komposisi, agregasi, atau pewarisan.

Berdasarkan identifikasi kelas dan relasi yang telah dijelaskan berikut adalah visualisasi lengkap *class diagram* sistem BookWise.



Gambar 5.4 *Class Diagram* Book Wise

Class diagram ini menampilkan struktur Lending Loan Processing BC pada BookWise. Loan berperan sebagai aggregate root yang mengendalikan state peminjaman, termasuk status, tanggal jatuh tempo, dan identitas peminjam dan buku. Semua service (LoanProcessingService, ReturnProcessingService, LoanExtensionService) memiliki dependency terhadap Loan, artinya mereka menggunakan entity Loan untuk menjalankan behavior masing-masing (pengajuan, pengembalian, dan perpanjangan pinjaman). Tidak terdapat relasi agregasi, komposisi, pewarisan, atau asosiasi lain, sehingga alur domain tetap sederhana dan terfokus pada pengelolaan pinjaman.

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, pengrajaan 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 6.1. 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 6.2. 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	<i>Tidak membutuhkan body</i>
/loans/{loan_id}/approve	POST	Admin menyetujui & mendistribusikan buku	<i>Tidak membutuhkan body</i>
/loans/{loan_id}	GET	Melihat detail loan termasuk status	<i>Tidak membutuhkan body</i>
/loans/my	GET	Peminjam melihat daftar pinjaman miliknya	<i>Tidak membutuhkan body</i>
/loans	GET	Admin melihat seluruh data pinjaman	<i>Tidak membutuhkan body</i>
/loans/{loan_id}/return	POST	Peminjam memulai proses pengembalian	<i>Tidak membutuhkan body</i>
/loans/{loan_id}/finalize	POST	Admin mengecek & menyelesaikan proses pengembalian	<i>Tidak membutuhkan body</i>
/loans/{loan_id}/extend	POST	Peminjam mengajukan perpanjangan durasi pinjam	{ "extra_days": 3 }

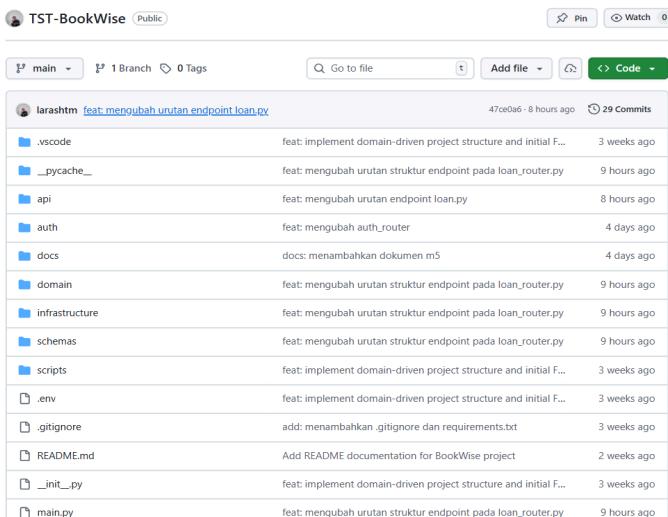
Selain daftar endpoint, sistem juga memiliki alur perubahan status peminjaman yang mengikuti aturan domain BookWise. Ringkasan perubahan status tersebut disajikan pada tabel berikut.

Tabel 6.3. 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 6.4. 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	

```

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 #masihin udah ke verifikasi admin
        self.approved = False #masihin 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

```

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

a

```

1  from uuid import UUID
2
3  class BookId:
4      def __init__(self, value: UUID):
5          self.value = value
6

```

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"))
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)).</p> <p>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>	File: domain/due_date.py
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>	<p>File: domain/loan_policy_service.py</p>
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.id)] = 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.

```
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.userId):
48         raise HTTPException(status_code=403, detail="Forbidden")
49     return to_response(loan)
50
```

```
1 #endpoint list loans punya peminjam
2 @router.get("/loans/my", response_model=list[LoanResponse])
3 def list_my_loans(current_user = Depends(require_role("peminjam"))):
4     user = current_user
5     loans = repo.findByUser(user.userId)
6     return [to_response(l) for l in loans]
7
8 #endpoint list semua loan(pengguna)
9 @router.get("/loans/all", response_model=list[LoanResponse])
10 def list_all_loans(current_user = Depends(require_role("pengguna"))):
11     loans = repo.list_all()
12     return [to_response(l) for l in loans]
13
14 #admin aksi: verifikasi loan
15 @router.put("/loans/{loan_id}/verify")
16 def verify_loan(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
17     loan = repo.findById(loan_id)
18     if not loan:
19         raise HTTPException(status_code=404, detail="Loan not found")
20     try:
21         loan.verify()
22         repo.save(loan)
23         return {"detail": "Loan verified"}
24     except ValueError as e:
25         raise HTTPException(status_code=400, detail=str(e))
26
27 #admin approve loan (menutin due date dan status borrowed)
28 @router.post("/loans/{loan_id}/approve")
29 def approve_loan(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
30     loan = repo.findById(loan_id)
31     if not loan:
32         raise HTTPException(status_code=404, detail="Loan not found")
33     try:
34         # Admin nirkas due date per policy
35         due = policy.calculate_due_date()
36         loan.approve(due)
37         repo.save(loan)
38         return {"detail": "Loan approved and distributed", "dueDate": due.value}
39     except ValueError as e:
40         raise HTTPException(status_code=400, detail=str(e))
41
42 #peminjam mulai proses pengembalian
43 @router.post("/loans/{loan_id}/return")
44 def initiate_return(loan_id: UUID, current_user = Depends(require_role("peminjam"))): #require_role: memastikan user harus punya role tertentu
45     loan = repo.findById(loan_id)
46     if not loan:
47         raise HTTPException(status_code=404, detail="Loan not found")
48     user = current_user
49     if str(loan.userId.value) != str(user.userId):
50         raise HTTPException(status_code=403, detail="Forbidden")
51     try:
52         loan.initiate_return()
53         repo.save(loan)
54         return {"detail": "Return initiated"}
55     except ValueError as e:
56         raise HTTPException(status_code=400, detail=str(e))
```

File: api/loan_router.py

```

1 #aksi admin: finalisasi pengembalian
2 @router.post("/loans/{loan_id}/finalize-return")
3 def finalize_return(loan_id: UUID, current_user = Depends(require_role("pengguna"))):
4     loan = repo.findById(loan_id)
5     if not loan:
6         raise HTTPException(status_code=404, detail="Loan not found")
7     try:
8         loan.finalize_return()
9         repo.save(loan)
10        return {"detail": "Return finalized"}
11    except ValueError as e:
12        raise HTTPException(status_code=400, detail=str(e))
13
14 #endpoint perpanjangan masa pinjam
15 from pydantic import BaseModel
16 class ExtendRequest(BaseModel):
17     extra_days: int
18
19 @router.post("/loans/{loan_id}/extend")
20 def extend_loan(loan_id: UUID, req: ExtendRequest, current_user = Depends(require_role("peminjam"))):
21     loan = repo.findById(loan_id)
22     if not loan:
23         raise HTTPException(status_code=404, detail="Loan not found")
24     # only peminjam can request extension
25     user = current_user
26     if str(loan.userId.value) != str(user.user_id):
27         raise HTTPException(status_code=403, detail="Forbidden")
28     try:
29         new_due = loan.extend_loan(req.extra_days)
30         repo.save(loan)
31         return {"detail": "Extension applied", "newDueDate": new_due.value}
32     except ValueError as e:
33         raise HTTPException(status_code=400, detail=str(e))
34
35

```

C.

```

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

```

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

	<p>membuat token yang berisi informasi dasar seperti userId. 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 user.py. 2. Sistem memverifikasi kecocokan <i>username</i> dan <i>password</i> sederhana dengan hash. 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

```

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 **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.

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

```

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.

```

- 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 OAS 3.1

OpenAPI.json

[Authorize](#)

default

Authentication

- POST** /auth/login Login
- POST** /auth/refresh Refresh Token
- POST** /auth/logout Logout
- GET** /auth/me Me

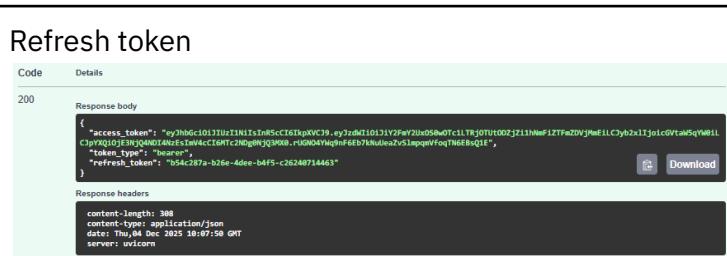
Loans

- POST** /loans Create Loan
- GET** /loans/me 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 Inside 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		

<p>1. Login sebagai Peminjam</p> 	<ol style="list-style-type: none"> Kirim Kredensial Peminjam/Pengguna login dengan username/password ke /auth/login. Verifikasi Server cek kredensial di database. Jika valid, lanjut ke token. Generate Token Server membentuk: <ul style="list-style-type: none"> - access_token → JWT berisi sub (user ID), role (peminjam), iat, exp - refresh_token → UUID unik - token_type → "bearer" Response ke Client <pre>{ "access_token": "...JWT...", "token_type": "bearer", "refresh_token": "...UUID..." }</pre> <ol style="list-style-type: none"> Penggunaan Client pakai access_token di header Authorization untuk akses endpoint yang butuh login.
<p>2. Refresh token</p> 	<ol style="list-style-type: none"> Client Kirim Refresh Token <ul style="list-style-type: none"> - Client (aplikasi) mengirim request ke endpoint refresh, /auth/refresh. - Body request hanya berisi: <pre>{ "refresh_token": "string" }</pre>

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).

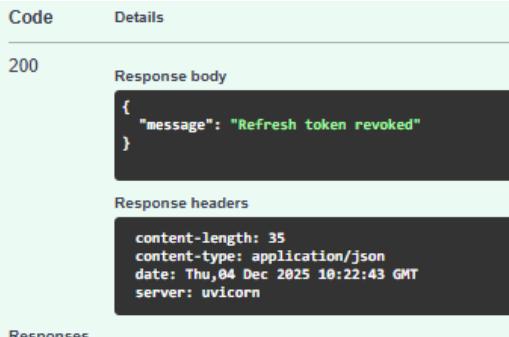
4. Server Kirim Response

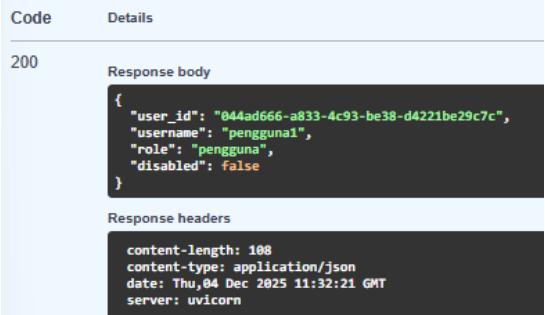
contoh:

```
{  
    "access_token":  
        "eyJhbGciOiJIUzI1NiIsInR  
        5cCI6IkpXVCJ9.eyJzdWIiOi  
        JiY2FmY2UxOS0wOTc1LTrjOT  
        UtODZjZi1hNmFizTFmZDVjMm  
        EiLCJyb2xlIjoicGVtaW5qYW  
        OiLCJpYXQiOjE3NjQ4NDI4Nz  
        EsImV4cCI6MTc2NDg0NjQ3MX  
        0.rUGNO4YWq9nF6Eb7kNuUea  
        ZvSlmpqmVfoqTN6EBsQ1E",  
    "token_type": "bearer",  
  
    "refresh_token":  
        "b54c287a-b26e-4dee-b4f5  
        -c26240714463"  
}
```

Penjelasan payload:

- **access_token** → JWT baru, dipakai untuk request API selanjutnya.

		<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.	<p>Logout</p>  <p>Responses</p>	<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> <ul style="list-style-type: none"> - Status code: 200 OK

		<ul style="list-style-type: none"> - Menandakan refresh token berhasil dicabut dan user tidak bisa refresh lagi tanpa login ulang.
4.	<p>Informasi Profil</p>  <pre> Code Details 200 Response body { "user_id": "044ad666-a833-4c93-be38-d4221be29c7c", "username": "pengguna1", "role": "pengguna", "disabled": false } Response headers content-length: 108 content-type: application/json date: Thu, 04 Dec 2025 11:32:21 GMT server: uvicorn </pre>	<ol style="list-style-type: none"> 1. Client kirim request ke /auth/me dengan header Authorization: Authorization: Bearer <access_token> 2. Server membaca token dan memverifikasi: <ul style="list-style-type: none"> o Apakah token valid? o Apakah belum kadaluarsa? 3. 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)	<ol style="list-style-type: none"> 1. Pastikan User Sudah Login dan Terautentikasi <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: <pre> Authorization: Bearer <access_token> </pre>
----	------------------------------	--

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>

- 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:

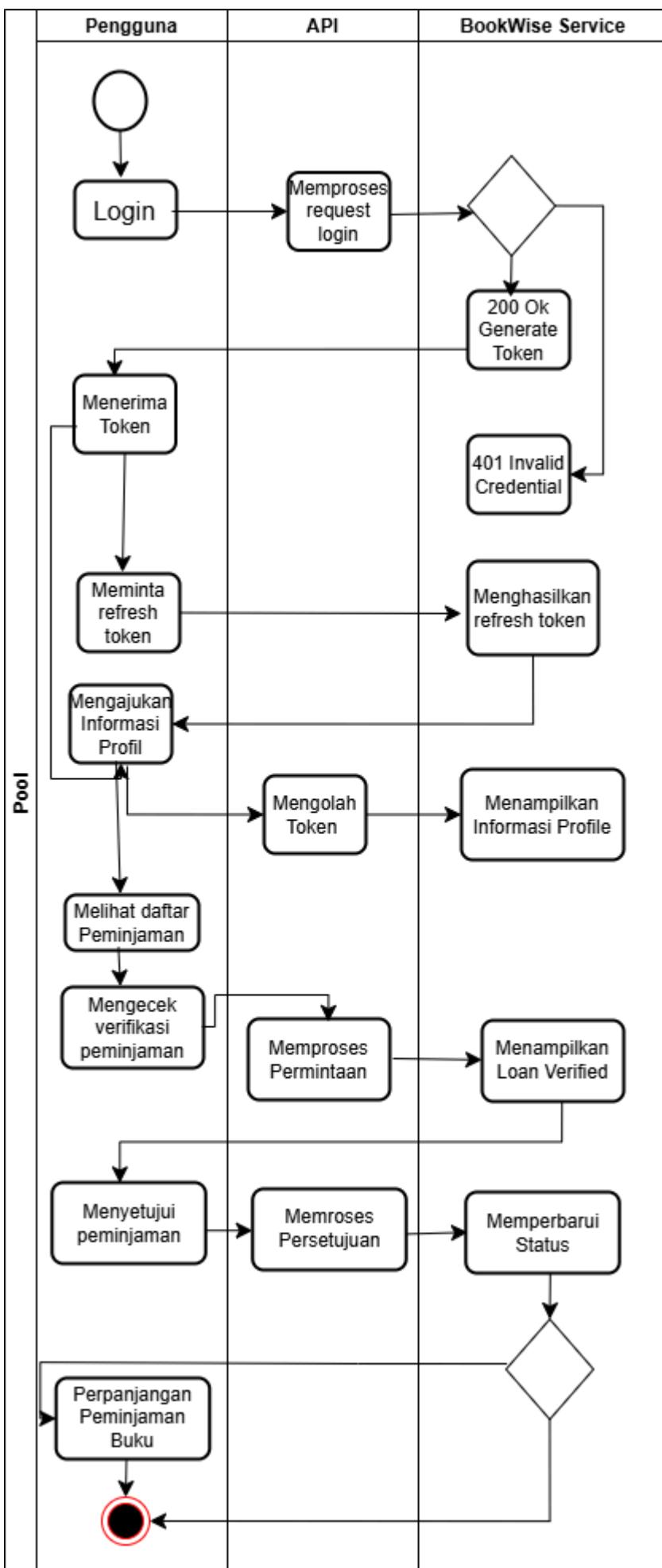

```
{
  "loanId": "a573a123-0891-49d0-b956-4883ea474042",
```

		<pre> "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 } </pre> <ul style="list-style-type: none"> - loanId → ID pinjaman baru - status → "requested" - verified / approved / return_initiated → false karena belum 								
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-2c963f66af78", "userId": "bcafce19-0975-4c95-86cf-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 <i>Undocumented</i></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: uvicorn</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-2c963f66af78", "userId": "bcafce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>	Code	Details	403 <i>Undocumented</i>	<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: uvicorn</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-2c963f66af78", "userId": "bcafce19-0975-4c95-86cf-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-2c963f66af78", "userId": "bcafce19-0975-4c95-86cf-a6abe1fd5c2a", "status": "requested", "createdAt": "2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>									
Code	Details									
403 <i>Undocumented</i>	<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: uvicorn</pre>									

	<p>200</p> <p>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": "2025-12-04T11:05:15.126461", "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": "2025-12-04T11:47:34.922225", "dueDate": null }</pre>	<pre>"2025-12-04T11:47:34.922225", "dueDate": null, "verified": false, "approved": false, "return_initiated": false }</pre>				
	<p>Get Loan (My) (Pengguna)</p> <p>403</p> <p>Error: Forbidden</p> <p>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>	<ul style="list-style-type: none"> Server cek: <ul style="list-style-type: none"> - Access token valid - Role = "peminjam" - userId di token sama dengan userId loan 				
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>	<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 <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. - Gunakan token untuk akses endpoint. 2. Verify Loan <ul style="list-style-type: none"> - Endpoint: POST /loans/{loan_id}/verify - Input: loan_id pinjaman yang ingin diverifikasi. <p>Syarat server:</p> <ul style="list-style-type: none"> - Loan harus dalam status "REQUESTED"
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>	<ul style="list-style-type: none"> - 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 3. Respons <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 					

		<ul style="list-style-type: none"> - → error 400. - Loan ID tidak ada → 404.
Test Case 3: Extend Loan		
1.	Extend Loan (Peminjam)	<ul style="list-style-type: none"> - Peminjam meminta perpanjangan (POST /loans/{loan_id}/extend): <ul style="list-style-type: none"> o Loan miliknya → dueDate bertambah. o Loan bukan miliknya → Forbidden 403. o Loan ID tidak ada → 404.
SPEK TAMBAHAN		
1.	Activity Diagram	



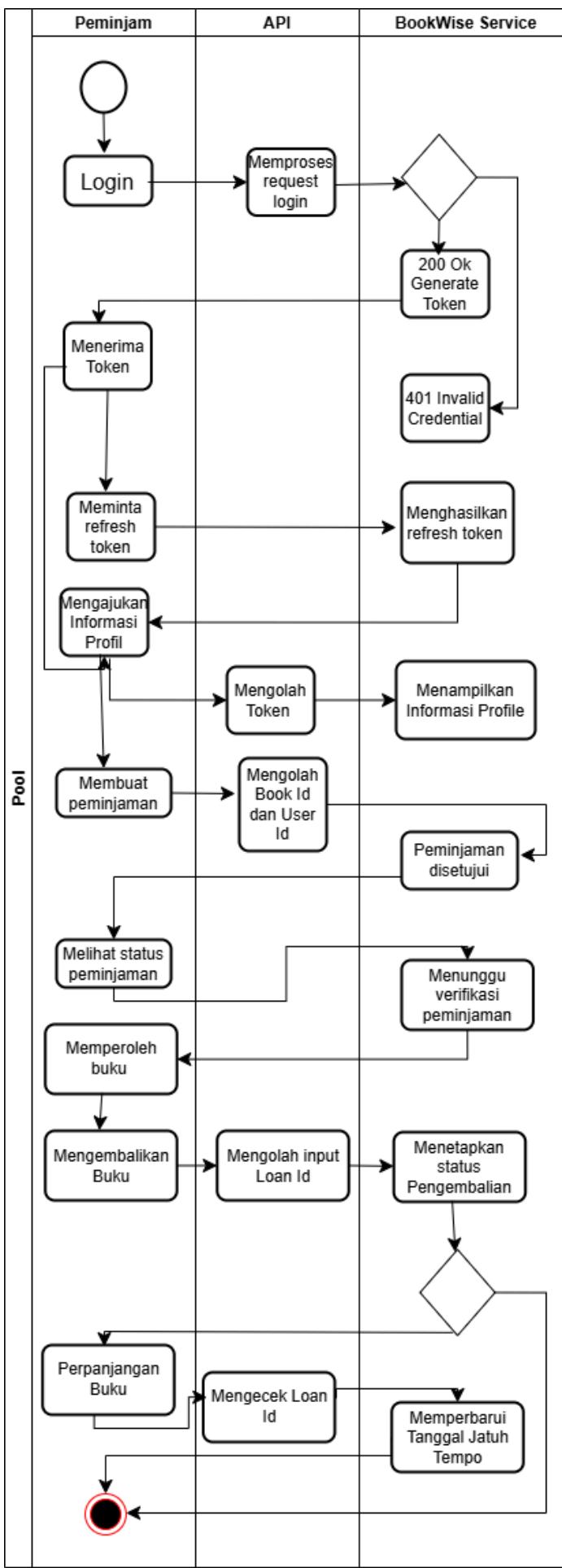


Diagram aktivitas (*activity diagram*) merupakan salah satu diagram UML yang digunakan untuk memodelkan alur aktivitas atau proses dalam suatu sistem dari awal hingga akhir. Menurut Booch dkk. (2005), diagram aktivitas digunakan untuk menggambarkan *workflow* suatu proses bisnis atau sistem, termasuk urutan aktivitas dan percabangan kondisi. Berangkat dari definisi tersebut, diagram aktivitas pada proyek BookWise digunakan untuk memvisualisasikan alur proses peminjaman buku dari sisi pengguna hingga sistem.

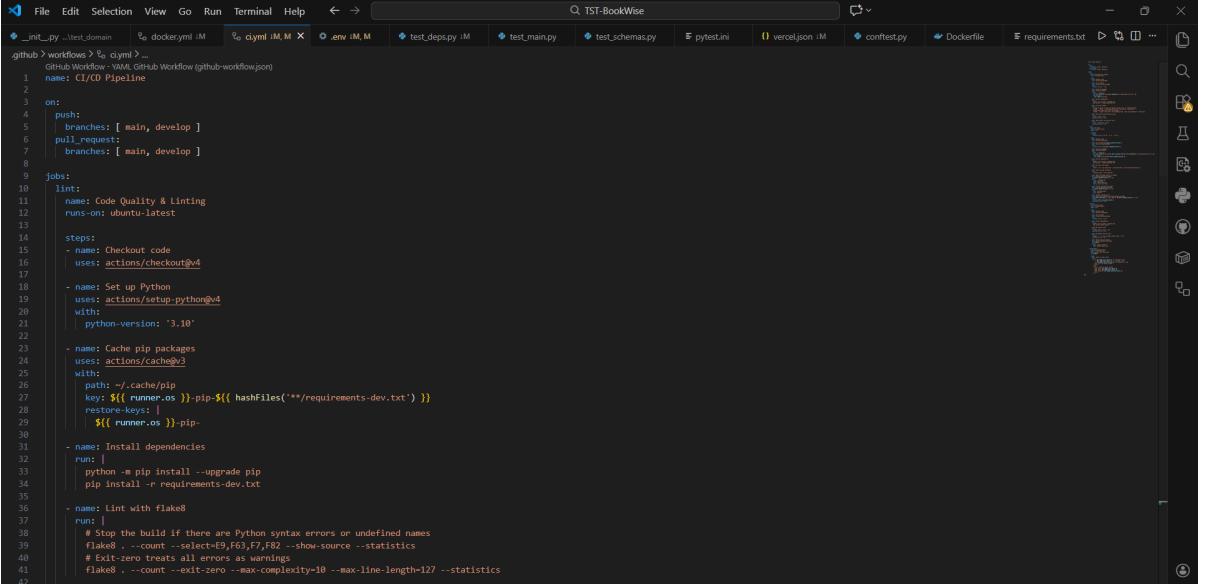
Diagram aktivitas BookWise disusun berdasarkan peran (*role*) dalam sistem, yaitu peminjam dan pengguna. Peminjam memiliki akses untuk melakukan peminjaman buku, melihat status peminjaman (*my loan*), melakukan pengembalian, serta mengajukan perpanjangan. Sementara itu, pengguna berperan dalam melihat daftar peminjaman, melakukan verifikasi, serta menyetujui permintaan peminjaman. Perbedaan hak akses ini direpresentasikan secara jelas pada alur aktivitas masing-masing peran di dalam diagram. Selain itu, diagram juga memodelkan interaksi antara pengguna, API, dan BookWise Service sebagai sistem backend. Percabangan alur ditunjukkan melalui simbol belah ketupat yang merepresentasikan kondisi keputusan.

2. Pengujian Unit Testing

The screenshot shows a VS Code interface with the following details:

- Terminal:** Displays test results for `test_infrastructure/test_in_memory_loan_repository.py`. It includes test cases like `testRepositoryListAll`, `test_repository_isolation`, and `test_save_multiple_loans`, all passing.
- Output:** Shows coverage statistics for platform `win32` and Python `3.11.4-final-0`. Coverage is at 96% with 2182 statements tested.
- Problems:** Shows 3 issues.
- Explorer:** Shows the project structure with files like `main.py`, `vercel.json`, and `coverage.xml`.
- Status Bar:** Shows file count (2446), column count (Col 1), tab size (4), and encoding (UTF-8).

- Pengujian pada sistem BookWise dilakukan dengan menerapkan prinsip Test Driven Development (TDD).
 - Alur pengujian meliputi: menulis unit test, menjalankan test, mengimplementasikan fitur, melakukan refactoring kode, dan menjalankan ulang seluruh test untuk memastikan tidak terjadi regresi.
 - Cakupan pengujian yang diterapkan mencakup:
 - Access testing, untuk memvalidasi hak akses endpoint berdasarkan role pengguna.

	<ul style="list-style-type: none"> • Error handling testing, untuk memastikan sistem menangani kondisi error seperti input tidak valid, status pinjaman tidak sesuai, dan data tidak ditemukan. • Security testing, untuk memverifikasi autentikasi dan otorisasi berbasis JWT. • Workflow testing, untuk menguji alur perubahan status pinjaman sesuai aturan domain.
	Berdasarkan hasil eksekusi pengujian unit, sistem BookWise berhasil mencapai test coverage sebesar 95,65%, melebihi target minimum yang ditetapkan. Dengan ringkasan hasil pengujian menunjukkan 2.182 baris kode diuji, 95 baris tidak teruji, dengan total test coverage sebesar 95,65%, yang menandakan kualitas dan stabilitas sistem sebelum tahap deployment.
3.	CI/CD (GitHub Workflow)
	 <pre> 1 name: CI/CD Pipeline 2 3 on: 4 push: 5 branches: [main, develop] 6 pull_request: 7 branches: [main, develop] 8 9 jobs: 10 lint: 11 name: Code Quality & Linting 12 runs-on: ubuntu-latest 13 14 steps: 15 - name: Checkout code 16 uses: actions/checkout@v4 17 18 - name: Set up Python 19 uses: actions/setup-python@v4 20 with: 21 python-version: '3.10' 22 23 - name: Cache pip packages 24 uses: actions/cache@v3 25 with: 26 path: ~/.cache/pip 27 key: \${runner.os}-pip-\${{ hashFiles('**/requirements-dev.txt') }} 28 restore-keys: 29 - \${runner.os}-pip- 30 31 - name: Install dependencies 32 run: 33 - python -m pip install --upgrade pip 34 - pip install -r requirements-dev.txt 35 36 - name: Lint with flake8 37 run: 38 # Stop the build if there are Python syntax errors or undefined names 39 flake8 . --count --select=E9,F63,F82 --show-source --statistics 40 # Exit-zero treats all errors as warnings 41 flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics 42 </pre>

```
File Edit Selection View Go Run Terminal Help ← → ⌘ TST-BookWise
github > workflows > cimy.iM > cimy.iM
jobs:
  lint:
    steps:
      - name: Check code formatting with black
        run:
          - black --check --diff .
          continue-on-error: true
      - name: Check import sorting with isort
        run:
          - isort --check-only --diff .
          continue-on-error: true
    test:
      name: Run Tests
      runs-on: ubuntu-latest
      needs: lint
    strategy:
      matrix:
        python-version: ['3.10', '3.11', '3.12']
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      - name: Cache pip packages
        uses: actions/cache@v3
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ matrix.python-version }}-${{ hashFiles('**/requirements-dev.txt') }}
          restore-keys:
            - ${{ runner.os }}-pip-${{ matrix.python-version }}
      - name: Install dependencies
        run:
          - python -m pip install --upgrade pip
          - pip install -r requirements-dev.txt
      - name: Run tests with pytest
        run:
          - pytest --cov --cov-report=xml --cov-report=html --cov-report-term-missing -v
      - name: Check coverage threshold
        run:
          - coverage report --fail-under=95
      - name: Upload coverage reports to Codecov
        uses: codecov/codecov-action@v3
        if: matrix.python-version == '3.10'
        with:
          file: ./coverage.xml
          flags: unittests
          name: codecov-umbrella
          fail_ci_if_error: false
      - name: Archive coverage HTML report
        uses: actions/upload-artifact@v2
        if: matrix.python-version == '3.10'
        with:
          name: coverage-report
          path: htmlcov
      - name: Comment coverage on PR
        uses: py-cov-action/python-coverage-comment.action@v3
        if: github.event_name == 'pull_request' && matrix.python-version == '3.10'
        with:
          GITHUB_TOKEN: ${{ github.token }}
          continue-on-error: true
    security:
      name: Security Scan
      runs-on: ubuntu-latest
      needs: lint
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - name: Install dependencies
        run:
          - python -m pip install --upgrade pip
          - pip install safety bandit
      - name: Run safety check
        run:
          - safety check --json || true
        continue-on-error: true
      - name: Run bandit security scan
        run:
          - bandit -r . -f json -o bandit-report.json || true
        continue-on-error: true
      - name: Upload security reports
        uses: actions/upload-artifact@v3
        if: always()
        with:
          name: security-reports
          path: bandit-report.json
    build-status:
      name: Build Status Check
      runs-on: ubuntu-latest
      needs: [lint, test, security]
      if: always()

```

```

157   steps:
158     - name: Check all jobs status
159       run:
160         if [ "${{ needs.lint.result }}" == "success" ] && \
161           [ "${{ needs.test.result }}" == "success" ] && \
162             [ "${{ needs.security.result }}" == "success" ]; then
163               echo "All CI checks passed!"
164               exit 0
165             else
166               echo "Some CI checks failed"
167               echo "Lint: ${{ needs.lint.result }}"
168               echo "Test: ${{ needs.test.result }}"
169               echo "Security: ${{ needs.security.result }}"
170               exit 1
171         fi
172

```



Continuous Integration (CI) pada sistem BookWise bertujuan untuk memastikan setiap perubahan kode yang di-push ke repository selalu melalui proses pengecekan kualitas, pengujian, dan validasi sebelum digabungkan ke branch utama, sehingga stabilitas sistem tetap terjaga.

Tahapan pipeline CI yang diterapkan meliputi:

- Testing, untuk menjalankan unit test menggunakan *pytest* pada beberapa versi Python (3.10, 3.11, dan 3.12) guna memastikan kompatibilitas sistem.
- Linting, untuk memastikan kualitas kode dan konsistensi penulisan menggunakan *flake8*, *black*, dan *isort*.
- Coverage Check, untuk memverifikasi bahwa tingkat pengujian memenuhi ambang batas minimum yang ditetapkan, yaitu 95% test coverage, dan menghentikan pipeline jika syarat tidak terpenuhi.
- Security Scan (opsional), untuk melakukan pemeriksaan kerentanan dependensi dan potensi isu keamanan menggunakan *safety* dan *bandit*.

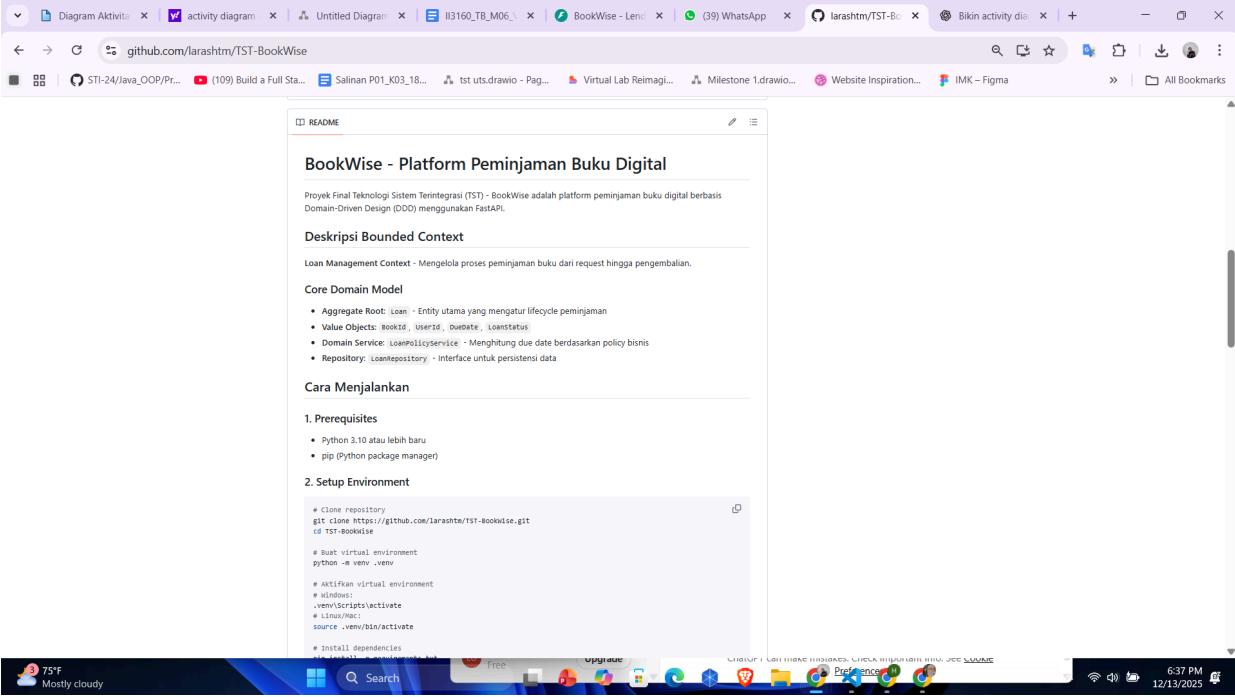
Sebagai hasil akhir, pipeline CI menampilkan status checklist hijau (passing) pada repository GitHub apabila seluruh tahapan berhasil dijalankan, yang menandakan bahwa kode telah memenuhi standar kualitas, pengujian, dan keamanan, serta siap untuk dilanjutkan ke tahap deployment.

4. Dockerfile dan Deploy

Link deploy: <https://tst-book-wise-theta.vercel.app/docs>

Sistem BookWise dikemas menggunakan Docker untuk memastikan konsistensi lingkungan antara tahap pengembangan, pengujian, dan deployment. Dockerfile digunakan untuk mendefinisikan proses pembuatan image aplikasi BookWise, termasuk konfigurasi environment dan dependensi, sehingga aplikasi dapat dijalankan secara konsisten sebagai sebuah container. Pada tahap deployment, sistem awalnya direncanakan untuk dideploy menggunakan Railway, namun proses implementasi tidak dapat dilanjutkan karena keterbatasan kuota dan batas penggunaan layanan.

Deployment kemudian dicoba menggunakan Vercel, tetapi ditemukan kendala pada konfigurasi environment variable serta ketidaksesuaian versi Python yang menyebabkan aplikasi tidak dapat berjalan dengan baik. Selain itu, pada proyek Vercel, proses deployment telah dilakukan beberapa kali, namun aplikasi sempat mengalami kegagalan dengan status 500 Internal Server Error meskipun telah melalui berbagai upaya perbaikan. Setelah melalui proses penyesuaian konfigurasi dan perbaikan kode secara bertahap, sistem akhirnya berhasil dideploy dengan konfigurasi yang sesuai. Dengan demikian, service BookWise dapat dijalankan baik secara lokal maupun melalui platform deployment berbasis container, dan berada pada kondisi ready to use.

5.	Readme
	
	<p>Pada projek ini, README disusun sebagai dokumentasi agar pengguna dapat memahami dan menjalankan sistem BookWise dengan lebih mudah. Di dalam README dijelaskan gambaran domain dan arsitektur sistem secara singkat, dilanjutkan dengan langkah menjalankan aplikasi, dokumentasi endpoint API, struktur proyek, serta panduan pengujian dasar. Dengan adanya README ini, pengguna tidak perlu membaca kode secara langsung untuk dapat mencoba dan menggunakan service yang disediakan.</p>
6.	Demo Tubes
	<p>link demo youtube: https://youtu.be/YMml5GDfgoQ?si=AvJ67kndUmu9V1TF</p>
Repository Github	
<ul style="list-style-type: none"> • URL repo: https://github.com/larashtm/TST-BookWise 	

The screenshot shows a Microsoft Edge browser window with multiple tabs open at the top. The active tab is 'github.com/larashtm/TST-BookWise'. The page displays the repository's main branch, which has 1 branch and 0 tags. The commit history is listed, showing 59 commits. The most recent commit is 'feat: commit lagiiii dan lagiii' by 'larashtm' at 30 minutes ago. Other commits include 'feat: implement domain-driven project structure and initial F...' and 'fix: optimize vercel deployment config'. The repository has 1 fork and 0 stars. On the right side, there is an 'About' section with a brief description of the BookWise platform, which is a digital lending system based on DDD and FastAPI. It also lists 'Readme', 'Activity', '0 stars', '0 watching', and '1 fork'. Below that are sections for 'Releases', 'Packages', and 'Deployments'. The system status is shown as 'Production - bookwise' at 29 minutes ago. The bottom of the screen shows the Windows taskbar with various pinned icons.