

TUGAS BESAR TAHAP - 4 IMPLEMENTASI AWAL *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.....	6
2.1.3 Book Management.....	7
2.1.4 User Management.....	7
3. Domain.....	7
4. Bounded Context.....	9
5. Arsitektur Aplikasi dan Implementasi.....	12
6. Implementasi Model Aggregate menjadi API.....	16

Daftar Gambar

Gambar 2.1 Business Capability BookWise	6
Gambar 4.1 Bounded Context BookWise	10
Gambar 5.4 Class Diagram Book Wise	15

Daftar Tabel

Tabel 3.1. Domain	7
Tabel 4.1 Lingkup (Boundary) Setiap Bounded Context)	10
Tabel 4.2 Pola Integrasi Antar Bounded Context	10
Tabel 5.1 Identifikasi Class Diagram	12
Tabel 5.2 Glosarium Ubiquitous Language (Loan BC)	13
Tabel 5.3 Relasi antar Entity/Value Object	14
Tabel 6.1. Implementasi berdasarkan layer	16
Tabel 6.2. API Surface	17
Tabel 6.3. Implementasi BookWise	17

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 bergeser 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.

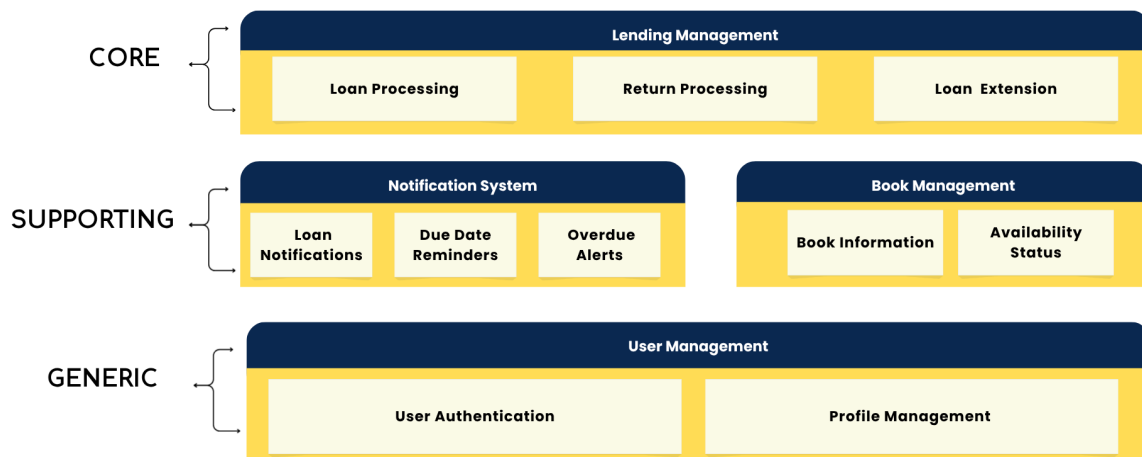
1.2. Deskripsi Sistem

BookWise merupakan platform peminjaman buku digital yang memungkinkan pengguna untuk meminjam dan mengelola buku secara daring. Sistem ini mendukung proses peminjaman dan pengembalian buku, pengelolaan informasi buku oleh admin, serta pengiriman notifikasi terkait status peminjaman.

1.3. Tujuan Sistem

1. Mempermudah pengguna dalam meminjam buku digital tanpa perlu kunjungan fisik ke perpustakaan
2. Memastikan pengelolaan buku yang efisien bagi admin
3. Memberikan notifikasi otomatis kepada pengguna terkait status peminjaman

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 dan hanya berinteraksi melalui referensi atau event.

2.1.1 Lending Management

Kapabilitas inti yang mengatur seluruh proses peminjaman buku digital. Area ini bertanggung jawab atas pembuatan pinjaman, pencatatan tanggal jatuh tempo, pengembalian buku, perpanjangan pinjaman, serta penandaan pinjaman yang melewati batas waktu. *Lending Management* juga memelihara riwayat peminjaman pengguna. *Lending Management* tidak menyimpan detail buku maupun data pengguna, ia hanya menyimpan sebagai referensi (BookId, UserID).

2.1.2 Notification System

Kapabilitas pendukung yang mengirimkan notifikasi kepada pengguna terkait aktivitas peminjaman. aktivitas utama meliputi: notifikasi pinjaman baru, pengingat tanggal jatuh tempo, dan pemberitahuan keterlambatan.

Notification berada pada *bounded context* terpisah. Ia hanya mengonsumsi event yang dipublikasikan oleh *Lending Management*, misalnya *LoanCreated*, *LoanReturned*, atau *LoanOverdue*.

2.1.3 Book Management

Kapabilitas pendukung untuk pengelolaan informasi buku, mulai dari detail buku (judul, pengarang, deskripsi), pengelolaan ketersediaan buku, perubahan metadata buku oleh admin. **BookWise** menampilkan informasi buku kepada pengguna sebelum mereka meminjam. Namun dalam domain *Lending*, konteks ini tidak dimodelkan. *Lending* hanya memanfaatkan **BookId** sebagai referensi tanpa mengetahui isi atau struktur data buku.

2.1.4 User Management

Kapabilitas dasar yang mengelola data dan akses pengguna ke dalam platform. Kapabilitas ini tidak dimodelkan pada *class diagram*. *Lending Management* tidak menyimpan informasi profil. Ia hanya berinteraksi melalui *UserId*, sehingga *User Management* tidak perlu dimodelkan pada desain taktis. Seluruh detail pengguna disimpan pada *bounded context* eksternal ini.

3. Domain

Tabel 3.1. Domain

Jenis Domain	Capability	Subdomain	Deskripsi
Core Domain	Lending Management	Loan Processing	Mengatur proses peminjaman buku digital, termasuk pembuatan pinjaman baru, validasi, dan pencatatan tanggal jatuh tempo. Subdomain ini adalah pusat logika inti aplikasi.
		Return Processing	Mengelola proses pengembalian buku, memperbarui status pinjaman, dan memastikan konsistensi data setelah buku dikembalikan.

Jenis Domain	Capability	Subdomain	Deskripsi
		<i>Loan Extension</i>	Menangani permintaan perpanjangan masa pinjam dan mengatur penyesuaian tanggal jatuh tempo sesuai kebijakan.
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	Manajemen Pengguna	<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. <i>Lending</i> hanya memakai <i>UserId</i> sebagai referensi.

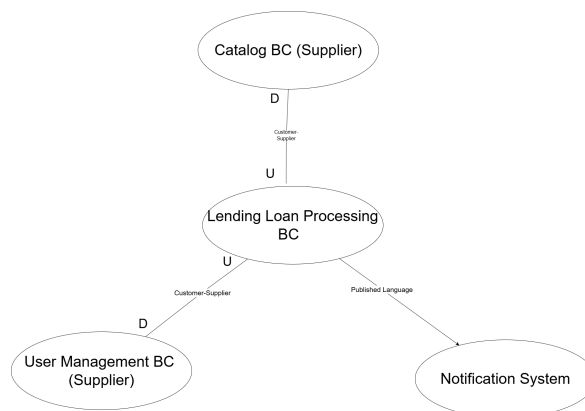
Pembagian ini menggambarkan area-area utama dalam domain bisnis peminjaman buku. Hal ini sesuai prinsip *Domain-Driven Design*, dimana subdomain mewakili area masalah bisnis, sedangkan *bounded context* menentukan batasan linguistik dan model yang digunakan dalam implementasi.

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**

Bounded Context merupakan bagian penting dalam pendekatan *Domain-Driven Design* (DDD) yang digunakan untuk memetakan batas tanggung jawab dari setiap domain di dalam sistem **Bookwise**. Melalui diagram *bounded context*, dapat diketahui bagaimana setiap domain saling berhubungan dan berinteraksi untuk mewujudkan fungsi utama sistem, yaitu peminjaman serta pembacaan buku digital.



Gambar 4.1 Bounded Context BookWise

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

Bounded Context	In-scope (Core Domain)	Out-of-scope
Lending BC (Core)	<i>Loan, LoanStatus, DueDate</i> , aturan peminjaman, <i>behavior</i> peminjaman (<i>borrow, returnBook, extendLoan, markOverdue</i>), <i>LoanPolicyService, LoanRepository</i>	Detail buku, detail user, notifikasi, autentikasi
Book Management BC (Catalog)	Informasi buku (judul, penulis, ISBN), metadata buku, status ketersediaan	Peminjaman, pengembalian, <i>due date</i> , identitas pengguna
Notification BC	<i>Event listener</i> , pengirim notifikasi, template notifikasi	Peminjaman, manajemen buku, autentikasi
User Management BC (Identity & Access)	Data pengguna, <i>autentikasi, otorisasi, profile management</i>	Detail peminjaman, detail buku, pengelolaan notifikasi

Tabel 4.2 Pola Integrasi Antar *Bounded Context*

Source BC	Target BC	Pattern	Makna Relasi
Lending BC (Core)	<i>Book Management BC</i>	<i>Customer–Supplier (Reference Model)</i>	<i>Book Management</i> menyediakan struktur dan data buku. <i>Lending</i> hanya menggunakan BookId tanpa mengetahui detail buku.
	<i>User Management BC</i>	<i>Customer–Supplier (Reference Model)</i>	<i>User Management</i> menyediakan data identitas

Source BC	Target BC	Pattern	Makna Relasi
			pengguna. <i>Lending</i> hanya menggunakan <i>UserId</i> sebagai referensi.
	<i>Notification BC</i>	<i>Published Language / Domain Event</i>	<i>Lending BC</i> menerbitkan event seperti <i>LoanCreated</i> , <i>LoanReturned</i> , <i>LoanOverdue</i> . <i>Notification BC</i> mendengarkan dan memproses <i>event</i> tersebut untuk mengirim notifikasi.

Dalam pendekatan Domain-Driven Design (DDD), **bounded context** digunakan untuk memetakan batas tanggung jawab setiap domain di sistem **BookWise**. Setiap bounded context memiliki peran spesifik, dan interaksi antar konteks sering mengikuti pola tertentu. Salah satu konsep penting adalah **upstream dan downstream**, yang menunjukkan arah ketergantungan antar domain. **Upstream** adalah bounded context yang menyediakan model atau informasi, sementara **downstream** adalah bounded context yang mengonsumsi model tersebut tanpa mengubahnya. Dalam **BookWise**, **Catalog / Book Management BC** dan **User Management BC** berperan sebagai upstream karena mereka menetapkan struktur utama buku dan pengguna. Sebaliknya, **Lending BC** menjadi downstream karena menggunakan model buku dan identitas pengguna untuk proses peminjaman, pengembalian, dan perpanjangan pinjaman, namun tidak memiliki otoritas untuk mengubah model tersebut. Pola integrasi yang tepat untuk hubungan ini adalah **Customer–Supplier**, di mana upstream bertindak sebagai supplier model dan downstream sebagai customer yang mengonsumsinya.

Berbeda dengan pola *Customer–Supplier*, hubungan antara **Lending BC** dan **Notification BC** menggunakan pola **Published Language**, karena yang dibagikan bukan model domain, melainkan *event*. *Notification BC* hanya mendengarkan *event*

untuk mengirim notifikasi, tanpa memerlukan penyesuaian model atau bahasa domain. Dalam pola ini, konsep *upstream downstream* tidak relevan, dan aliran informasi cukup direpresentasikan sebagai event. Dengan demikian, kombinasi *pola Customer–Supplier* dan *Published Language* memungkinkan sistem *BookWise* menjaga konsistensi data dan aliran logika, sekaligus memisahkan tanggung jawab inti dan layanan pendukung secara jelas.

5. Arsitektur Aplikasi dan Implementasi

Tabel 5.1 *Identifikasi Class Diagram*

Nama Kelas	Jenis Kelas	Deskripsi
Loan	<i>Aggregate Root / Entity</i>	Entitas utama peminjaman yang mengelola seluruh state pinjaman.
LoanStatus	<i>Value Object</i>	Merepresentasikan status peminjaman (<i>Requested, Borrowed, Returned, Overdue</i>).
DueDate	<i>Value Object</i>	Batas waktu pengembalian serta logika <i>overdue</i> .
BookId	<i>Value Object</i>	Identitas buku yang dipinjam, sebagai referensi ke <i>Catalog BC</i> .
UserId	<i>Value Object</i>	Identitas pengguna yang meminjam, sebagai referensi ke <i>User Management BC</i> .
LoanRepository	<i>Repository</i>	Antarmuka untuk menyimpan dan mengambil <i>Loan</i> .
LoanPolicyService	<i>Domain Service</i>	Mengelola aturan peminjaman seperti perhitungan <i>due date</i> atau <i>validasi policy</i> .

Berdasarkan Tabel 5.1, struktur domain *Loan* menjadi jelas. **Loan** sebagai *aggregate root* menjaga *state* peminjaman melalui *behavior*. *Behavior* ini memastikan aturan domain tetap valid, misalnya status awal harus **Requested**, transisi status harus sesuai urutan, **DueDate** tidak boleh di masa lalu, dan pengembalian hanya dapat dilakukan jika status sudah **Borrowed**. Value object seperti **LoanStatus**, **DueDate**, **BookId**, dan **UserId** menjaga konsistensi data, sementara **LoanRepository** dan **LoanPolicyService** memisahkan logika penyimpanan dan bisnis.

Tabel 5.2 *Glosarium Ubiquitous Language (Loan BC)*

Istilah	Deskripsi
<i>Loan</i>	Transaksi peminjaman buku.
<i>BookId</i>	Identitas unik buku.
<i>UserId</i>	Identitas unik pengguna.
<i>LoanStatus</i>	Status pinjaman.
<i>DueDate</i>	Tenggat pengembalian.
<i>Borrow</i>	Proses membuat pinjaman aktif.
<i>ReturnBook</i>	Proses mengembalikan buku.
<i>Overdue</i>	Kondisi lewat batas waktu.
<i>LoanRepository</i>	Tempat simpan Loan.
<i>LoanPolicy</i>	Kebijakan peminjaman.

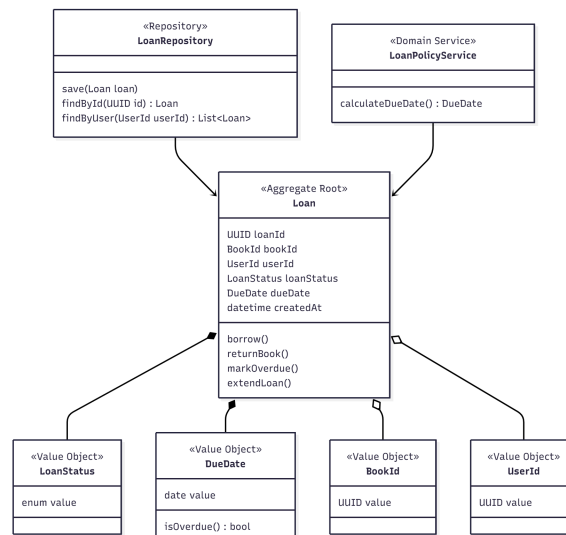
Tabel 5.2 menyajikan *glosarium Ubiquitous Language* yang digunakan dalam *Loan Bounded Context*. *Glosarium* ini berfungsi sebagai panduan istilah yang konsisten antara tim pengembang, analis, dan stakeholder, sehingga setiap konsep dalam domain memiliki makna yang sama dan tidak menimbulkan ambiguitas. Kehadiran *Ubiquitous Language* ini juga memastikan bahwa *class diagram* yang dikembangkan benar-benar mencerminkan proses bisnis. Tahap berikutnya adalah melihat bagaimana tiap entitas saling berhubungan dalam domain. Relasi tersebut dirangkum pada tabel berikut.

Tabel 5.3 Relasi antar *Entity/Value Object*

Entity 1	Entity 2	Kardinalitas	Tipe Relasi UML	Penjelasan
<i>Loan</i>	<i>LoanStatus</i>	1 : 1	Komposisi	<i>Loan</i> memiliki status, dan status tidak hidup terpisah.
<i>Loan</i>	<i>DueDate</i>	1 : 1	Komposisi	<i>DueDate</i> bagian dari <i>Loan</i> dan tidak berdiri sendiri.
<i>Loan</i>	<i>BookId</i>	1 : 1	Agregasi	<i>Loan</i> menggunakan <i>BookId</i> , tapi <i>BookId</i> berasal dari BC lain.
<i>Loan</i>	<i>UserId</i>	1 : 1	Agregasi	Sama: <i>UserId</i> adalah identitas dari BC eksternal.
<i>LoanRepository</i>	<i>Loan</i>	1 : N	Asosiasi	<i>Repository</i> menyimpan banyak <i>Loan</i> .
<i>LoanPolicyService</i>	<i>Loan</i>	1 : N	Asosiasi	<i>Service</i> tidak memiliki <i>Loan</i> , hanya memakai.

Dari relasi pada Tabel 5.3 terlihat bahwa domain *Loan* memisahkan dengan jelas mana komponen yang menjadi bagian inti entitas dan mana yang hanya digunakan sebagai referensi. Asosiasi dipakai untuk komponen yang hanya berinteraksi tanpa memiliki entitas tersebut.

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 pada Gambar 5.4 menunjukkan struktur lengkap sistem **BookWise** dengan atribut dan method pada setiap kelas, serta garis relasi yang merepresentasikan interaksi antar kelas sesuai dengan kardinalitas dan tipe relasi yang telah didefinisikan. Diagram ini menjadi blueprint teknis untuk implementasi kode program dan memastikan bahwa setiap komponen sistem memiliki tanggung jawab yang jelas sesuai dengan domain masing-masing. Dengan desain *class diagram* yang modular dan mengikuti prinsip DDD, sistem **BookWise** dapat dikembangkan secara terstruktur, mudah dipelihara, dan siap untuk dikembangkan lebih lanjut sesuai kebutuhan bisnis di masa mendatang.

6. Implementasi Model Aggregate menjadi API

Bagian ini menjelaskan implementasi awal **BookWise** berdasarkan model aggregate. Implementasi dilakukan dengan membangun domain model (entity, value object, domain service), *repository in-memory*, serta API dasar menggunakan FastAPI. Seluruh implementasi difokuskan pada aggregate **Loan** sebagai *core domain*. Kode dijalankan secara lokal menggunakan *virtual environment*.

Tabel 6.1. Implementasi berdasarkan layer

Layer	Komponen	Implementasi (File/Folder)	Deskripsi Singkat
Domain Layer	Aggregate Root	domain/loan.py	Entity utama yang mengatur lifecycle peminjaman.
	Value Objects	book_id.py, user_id.py, loan_status.py, due_date.py	Representasi identitas & atribut immutable.
	Domain Service	loan_policy_service.py	Menghitung due date & kebijakan peminjaman.
	Repository Interface	loan_repository.py	Abstraksi penyimpanan loan.
Application Layer	API Router	api/loan_router.py	Endpoint untuk membuat & mengambil loan.

	Schema	schemas/loan_schema.py	Struktur request/response untuk API.
Infrastructure Layer	Repository Implementation	infrastructure/in_memory_loan_repository.py	Implementasi penyimpanan berbasis memori.
	App Initialization	main.py	Menggabungkan router & menjalankan aplikasi.

Tabel 6.2. API Surface

Endpoint	Method	Deskripsi	Contoh Request Body
/loans	POST	Membuat peminjaman baru	{ "bookId": "...", "userId": "..." }
/loans/{loan_id}	GET	Mengambil data loan berdasarkan ID	-

Tabel 6.3. Implementasi BookWise

No.	Penjelasan
1	Struktur Folder
Struktur proyek dirancang mengikuti prinsip <i>layered architecture</i> .	
<pre> TST-BookWise/ ├── .gitignore ├── README.md ├── requirements.txt ├── main.py ├── ├── api/ │ ├── __init__.py │ └── loan_router.py ├── ├── domain/ │ ├── __init__.py │ └── loan.py </pre>	

```

├── book_id.py
├── user_id.py
├── due_date.py
├── loan_status.py
├── loan_repository.py
├── loan_policy_service.py
├── infrastructure/
│   ├── __init__.py
│   └── in_memory_loan_repository.py
├── schemas/
│   ├── __init__.py
│   └── loan_schema.py

```

Implementasi Domain Model

2 Aggregate Root: Loan

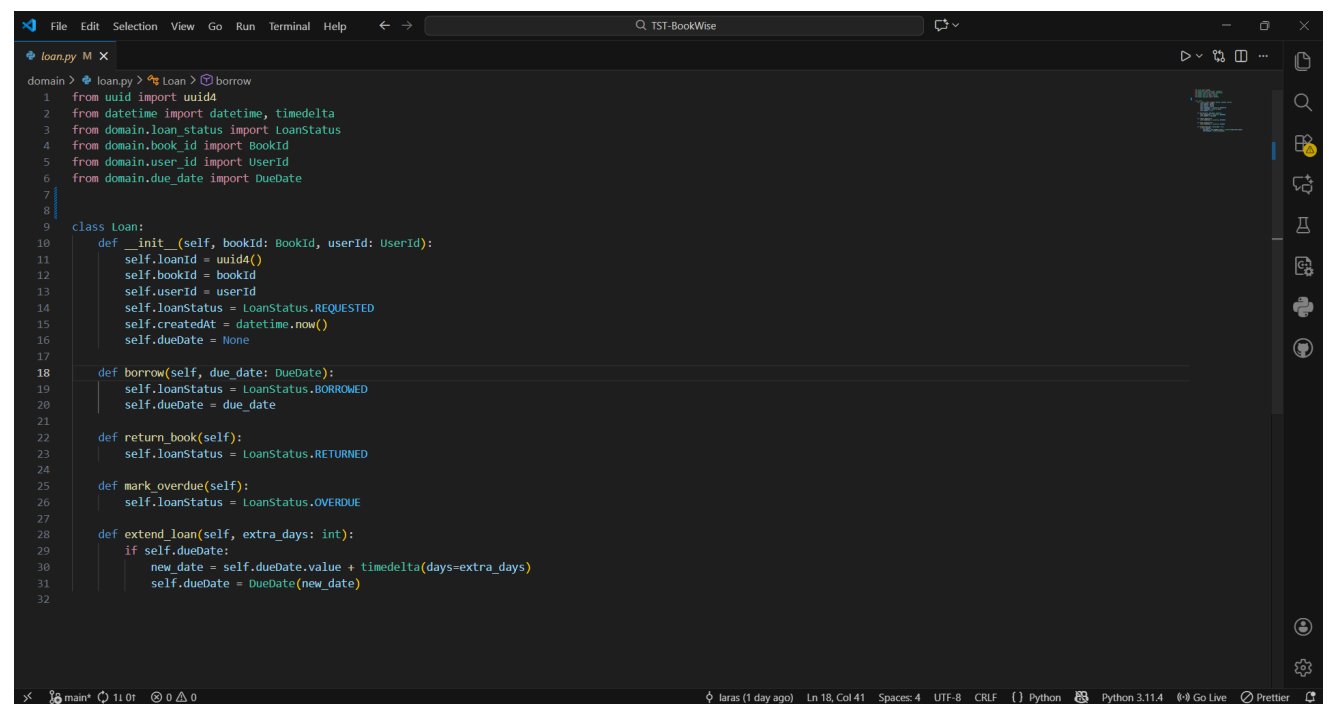
File: domain/loan.py

Penjelasan:

Loan adalah **Aggregate Root** dengan identity loanId

Memiliki **behaviors** yang meng-encapsulate business logic

Menjaga **consistency boundary** - tidak ada yang bisa mengubah state loan dari luar kecuali melalui methods



```

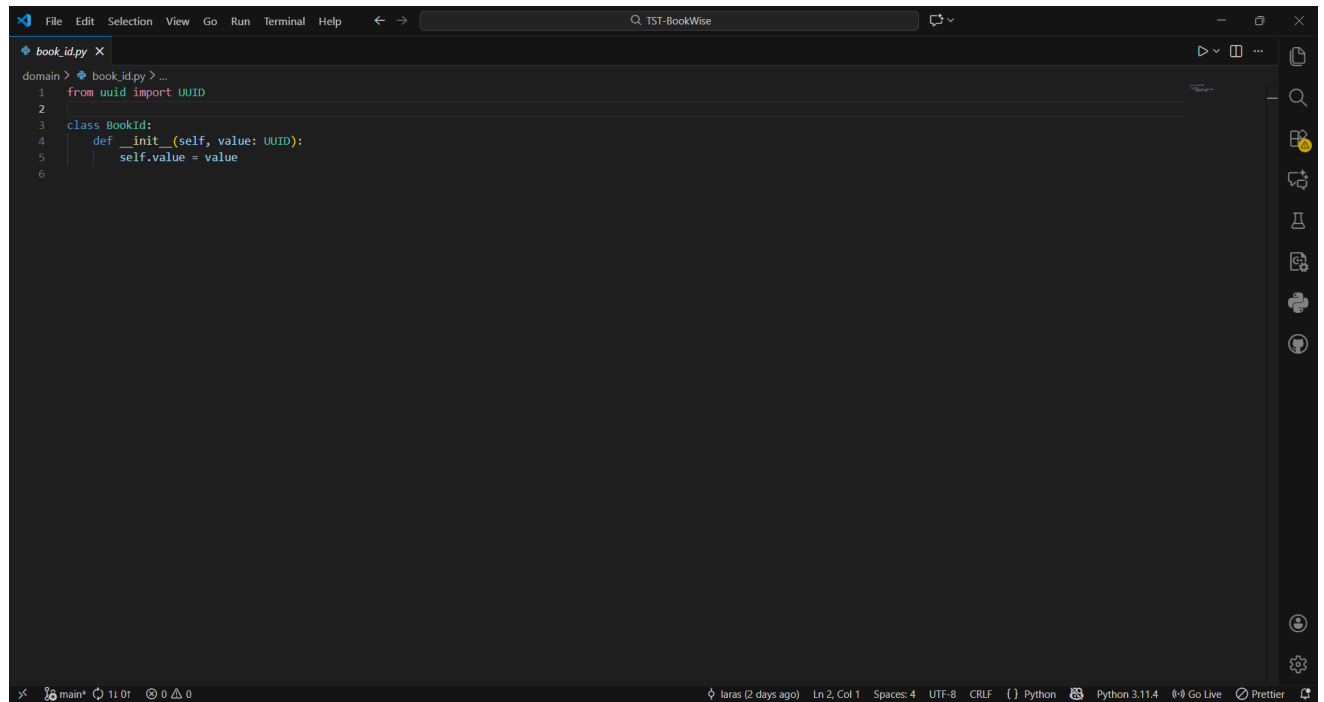
domain > loan.py > Loan > borrow
1 from uuid import uuid4
2 from datetime import datetime, timedelta
3 from domain.loan_status import LoanStatus
4 from domain.book_id import BookId
5 from domain.user_id import UserId
6 from domain.due_date import DueDate
7
8
9 class Loan:
10     def __init__(self, bookId: BookId, userId: UserId):
11         self.loanId = uuid4()
12         self.bookId = bookId
13         self.userId = userId
14         self.loanStatus = LoanStatus.REQUESTED
15         self.createdAt = datetime.now()
16         self.dueDate = None
17
18     def borrow(self, due_date: DueDate):
19         self.loanStatus = LoanStatus.BORROWED
20         self.dueDate = due_date
21
22     def return_book(self):
23         self.loanStatus = LoanStatus.RETURNED
24
25     def mark_overdue(self):
26         self.loanStatus = LoanStatus.OVERDUE
27
28     def extend_loan(self, extra_days: int):
29         if self.dueDate:
30             new_date = self.dueDate.value + timedelta(days=extra_days)
31             self.dueDate = DueDate(new_date)
32

```

3 Value Objects

a. BookId

File: domain/book_id.py

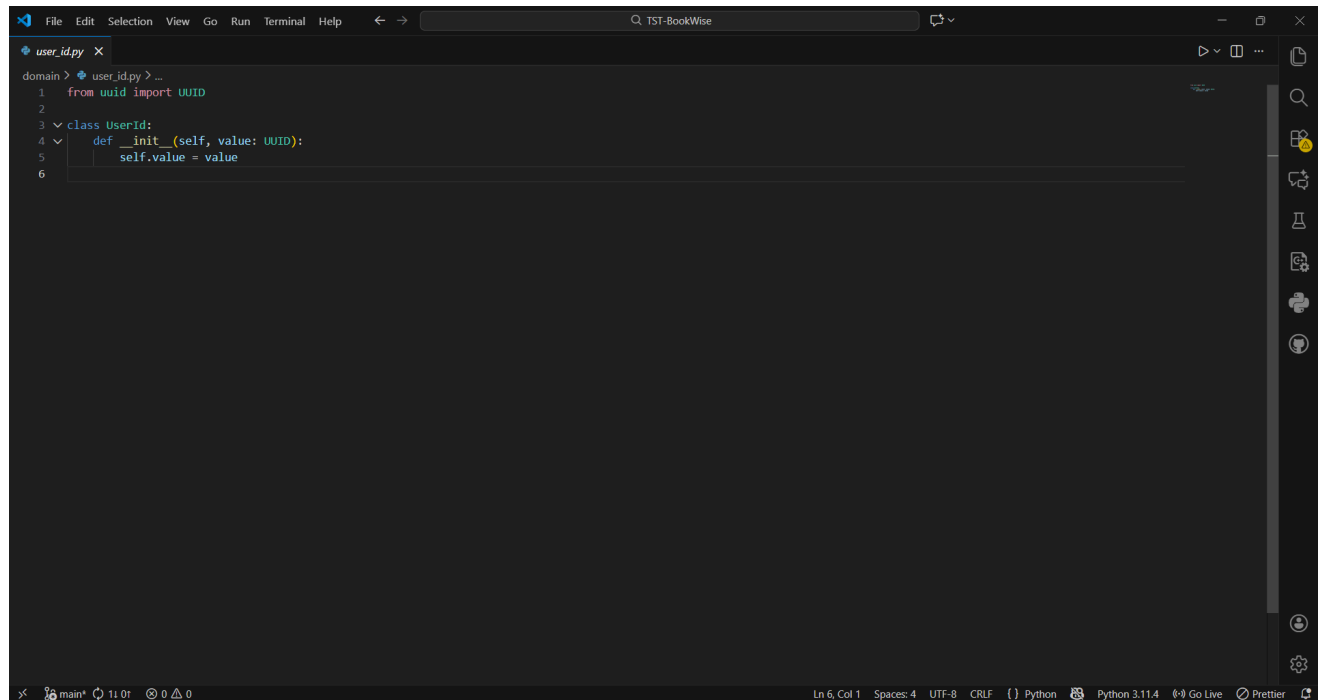


The screenshot shows a code editor with a dark theme. The file is named 'book_id.py'. The code defines a class 'BookId' with an '__init__' method that takes 'self' and 'value' as arguments and assigns 'self.value = value'. The editor has a menu bar with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The status bar at the bottom shows 'main*', '11:07', '0 0 0', 'laras (2 days ago)', 'Ln 2, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', 'Python 3.11.4', 'Go Live', and 'Prettier'.

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

b. UserId

File: domain/user_id.py

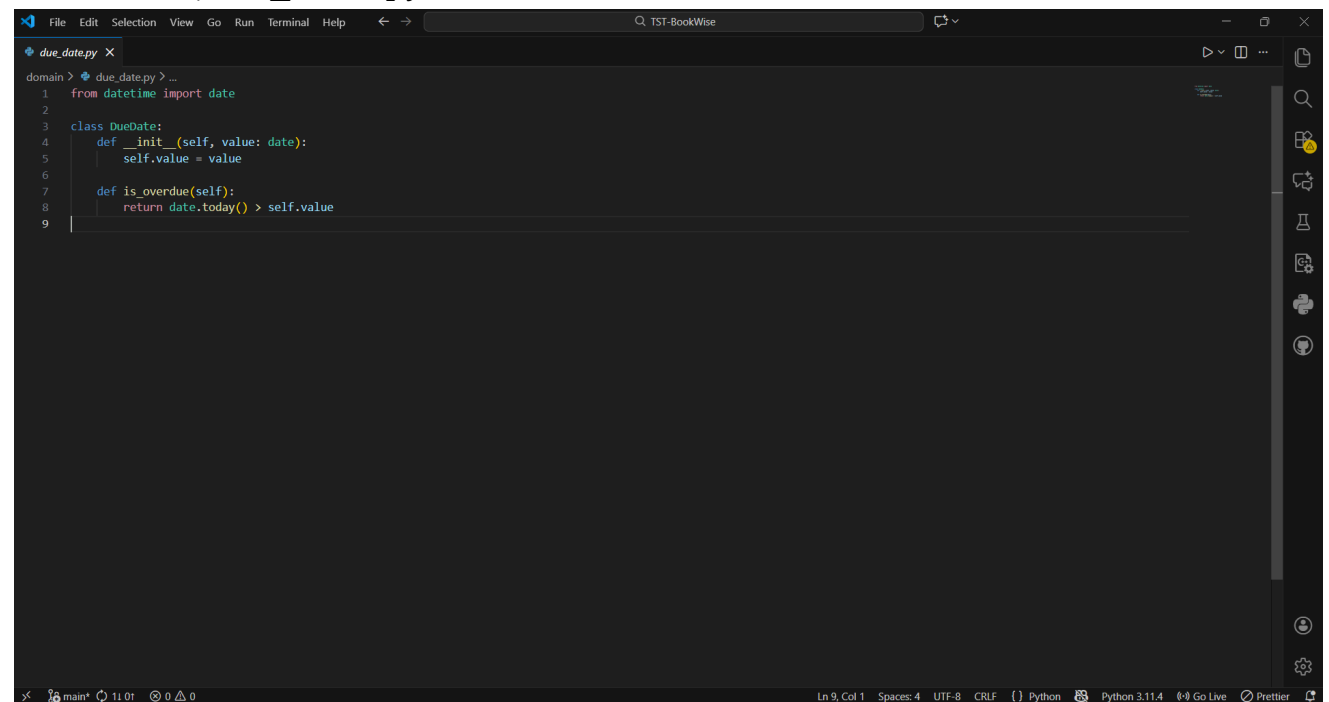


The screenshot shows a code editor with a dark theme. The file is named 'user_id.py'. The code defines a class 'UserId' with an '__init__' method that takes 'self' and 'value' as arguments and assigns 'self.value = value'. The editor has a menu bar with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The status bar at the bottom shows 'main*', '11:07', '0 0 0', 'Ln 6, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', 'Python 3.11.4', 'Go Live', and 'Prettier'.

```
domain > user_id.py > ...
1  from uuid import UUID
2
3  class UserId:
4      def __init__(self, value: UUID):
5          self.value = value
6
```

c. DueDate

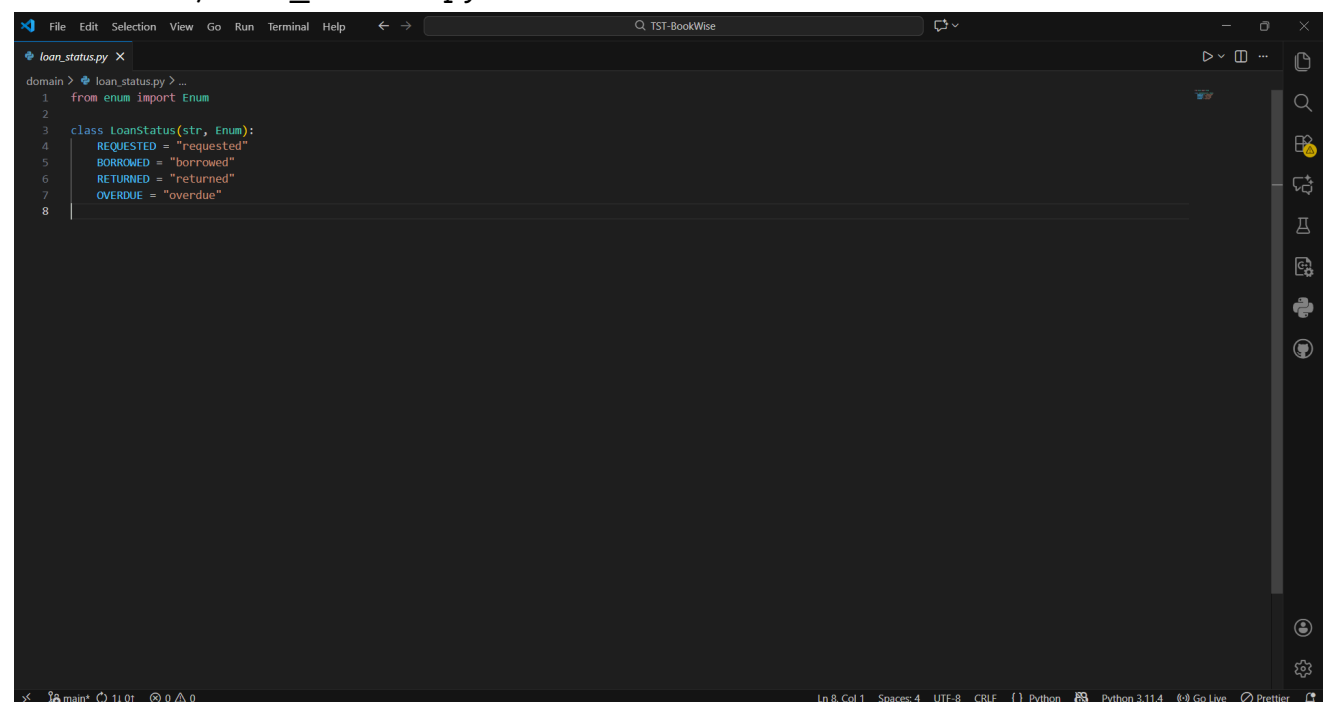
File: domain/due_date.py



```
domain > due_date.py > ...
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
9
```

d. LoanStatus

File: domain/loan_status.py



```
domain > loan_status.py > ...
1  from enum import Enum
2
3  class LoanStatus(str, Enum):
4      REQUESTED = "requested"
5      BORROWED = "borrowed"
6      RETURNED = "returned"
7      OVERDUE = "overdue"
8
```

4 Domain Server: LoanPolicyService

File: domain/loan_policy_service.py

```
File Edit Selection View Go Run Terminal Help
TST-BookWise

loan_policy_service.py
domain > loan_policy_service.py > ...
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
```

5 Implementasi Repository

a. Repository Interface

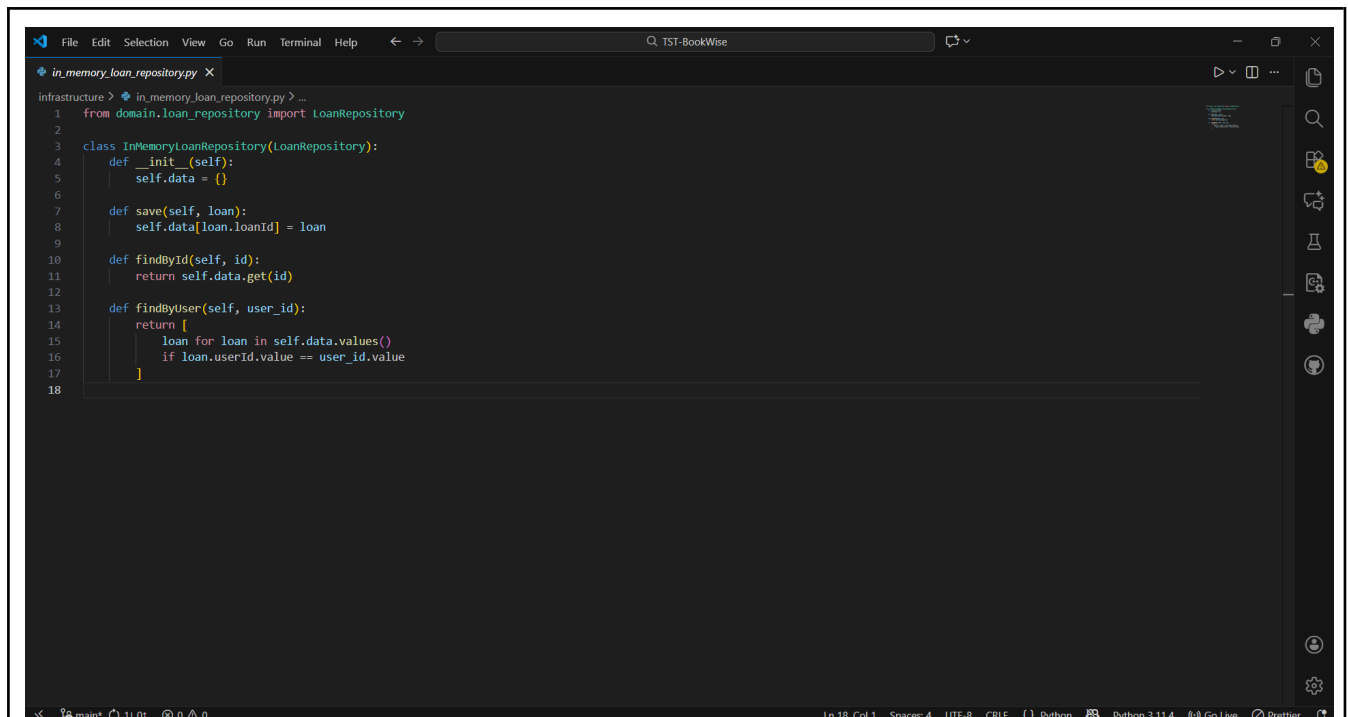
File: domain/loan_repository.py

```
File Edit Selection View Go Run Terminal Help
TST-BookWise

loan_repository.py
domain > loan_repository.py > ...
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
```

b. Repository Implementation

File: infrastructure/in_memory_loan_repository.py

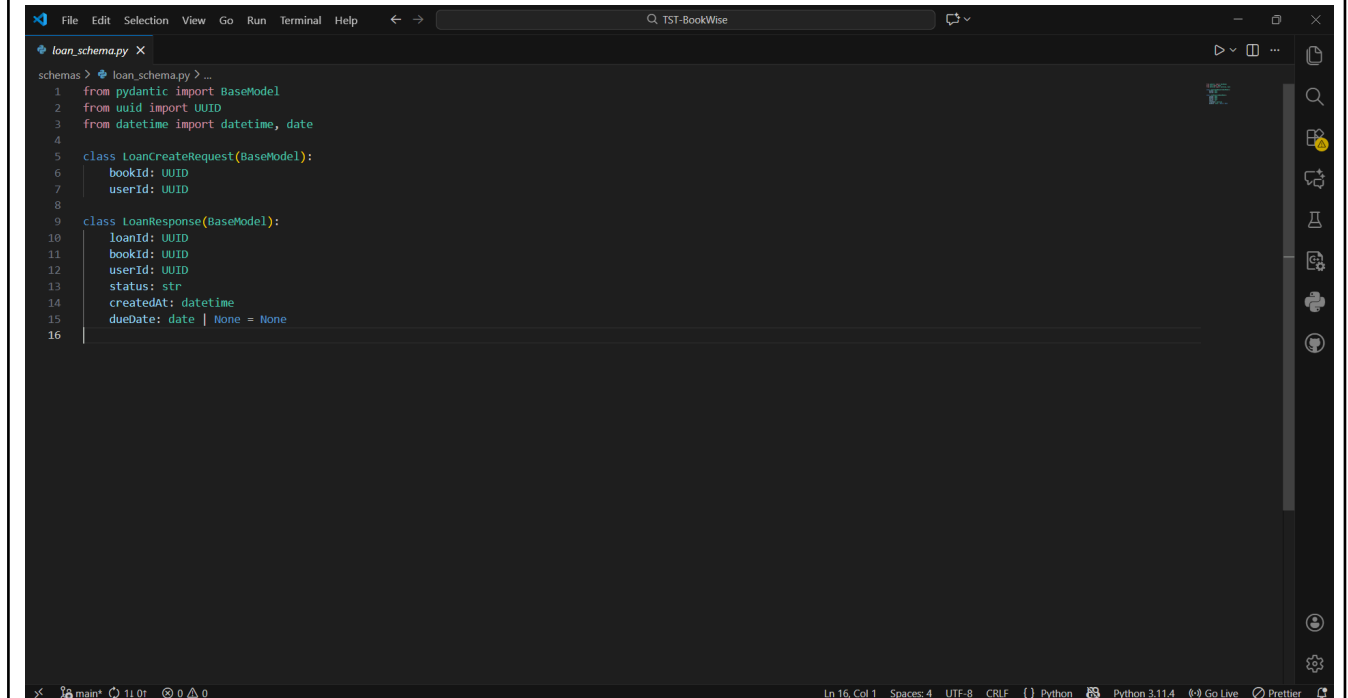


```
1 from domain.loan_repository import LoanRepository
2
3 class InMemoryLoanRepository(LoanRepository):
4     def __init__(self):
5         self.data = {}
6
7     def save(self, loan):
8         self.data[loan.loanId] = loan
9
10    def findById(self, id):
11        return self.data.get(id)
12
13    def findByUser(self, user_id):
14        return [
15            loan for loan in self.data.values()
16            if loan.userId.value == user_id.value
17        ]
18
```

6 Implementasi API

a. Request/Response Schemas

File: schemas/loan_schema.py



```
1 from pydantic import BaseModel
2 from uuid import UUID
3 from datetime import datetime, date
4
5 class LoanCreateRequest(BaseModel):
6     bookId: UUID
7     userId: UUID
8
9 class LoanResponse(BaseModel):
10     loanId: UUID
11     bookId: UUID
12     userId: UUID
13     status: str
14     createdAt: datetime
15     dueDate: date | None = None
16
```

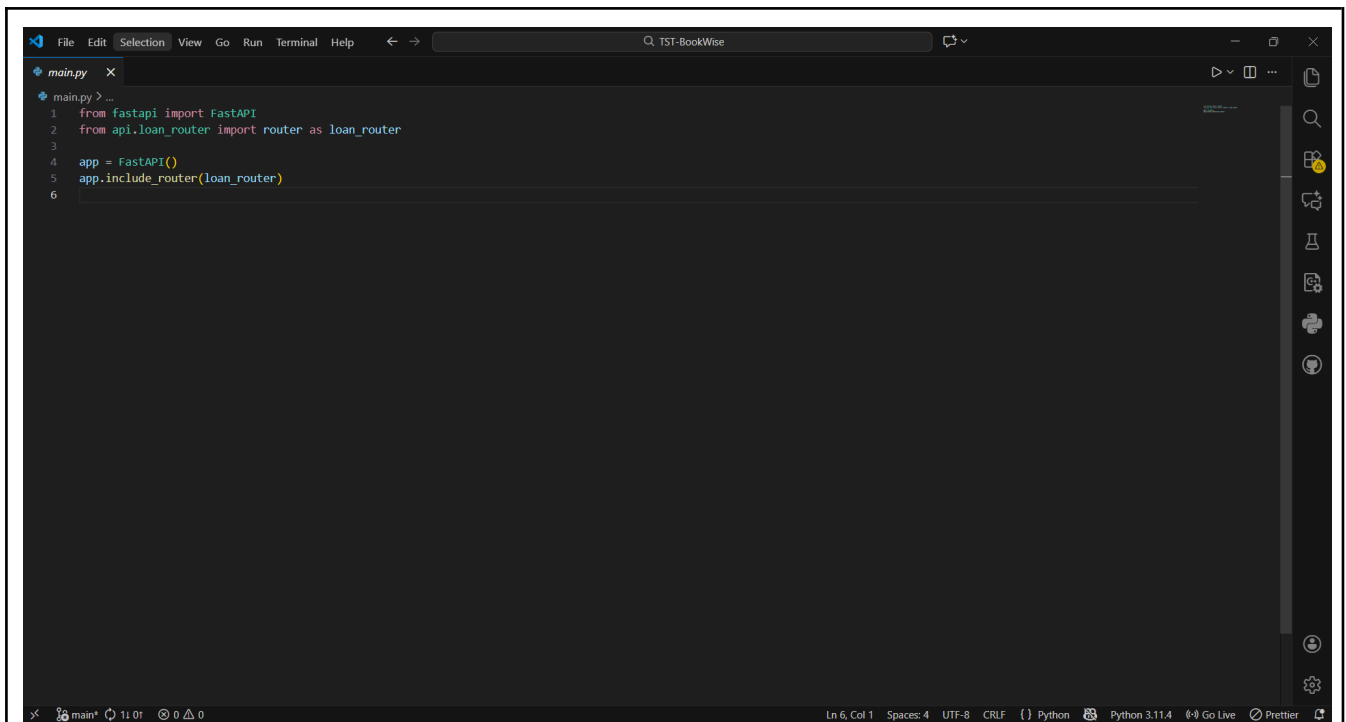
b. API Router

File: api/loan_router.py

```
loan_router.py X
api > loan_router.py > create_loan
1 from fastapi import APIRouter, HTTPException
2 from uuid import UUID
3
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
11 router = APIRouter()
12 repo = InMemoryLoanRepository()
13 policy = LoanPolicyService()
14
15
16 @router.post("/loans", response_model=LoanResponse)
17 def create_loan(req: LoanCreateRequest):
18     loan = Loan(BookId(req.bookId), UserId(req.userId))
19     due = policy.calculate_due_date()
20     loan.borrow(due)
21
22     repo.save(loan)
23
24     return LoanResponse(
25         loanId=loan.loanId,
26         bookId=loan.bookId.value,
27         userId=loan.userId.value,
28         status=loan.loanStatus.value, # <- FIX
29         createdAt=loan.createdAt,
30         dueDate=loan.dueDate.value
31     )
32
33
34 loan_router.py X
api > loan_router.py > create_loan
34 @router.get("/loans/{loan_id}", response_model=LoanResponse)
35 def get_loan(loan_id: UUID): # <- FIX (UUID type)
36     loan = repo.findById(loan_id)
37
38     if not loan:
39         raise HTTPException(status_code=404, detail="Loan not found")
40
41     return LoanResponse(
42         loanId=loan.loanId,
43         bookId=loan.bookId.value,
44         userId=loan.userId.value,
45         status=loan.loanStatus.value, # <- FIX
46         createdAt=loan.createdAt,
47         dueDate=loan.dueDate.value
48     )
49
50
51 main* 11:01 0 0 0
laras (2 days ago) Ln 17, Col 5 Spaces: 4 UTF-8 CRLF Python Python 3.11.4 Go Live Prettier
```

c. Main Application

File: main.py



```
1 from fastapi import FastAPI
2 from api.router import router as loan_router
3
4 app = FastAPI()
5 app.include_router(loan_router)
6
```

7 Test Case

Cara Menampilkan Endpoint + Testing POST & GET di Swagger

Eksekusi Server

```
PS C:\Users\laras\Bahasa C\SEMESTER 5\II3160_TST\TST-BookWise> uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['C:\Users\laras\Bahasa C\SEMESTER 5\II3160_TST\TST-BookWise']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [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>”

Saat membuka link tersebut akan muncul daftar endpoint yang memuat:

- POST /loans
- GET /loans/{loan_id}

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

FastAPI 0.1.0 OAS 3.1
/openapi.json

default

POST /loans Create Loan

GET /loans/{loan_id} Get Loan

Schemas

- HTTPValidationError** > Expand all object
- LoanCreateRequest** > Expand all object
- LoanResponse** > Expand all object
- ValidationError** > Expand all object

a. Test Case 1: Create Loan (POST /loans)

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

Request:

```
{
  "bookId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6",
  "userId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6"
}
```

POST /loans Create Loan

Parameters Cancel Reset

No parameters

Request body **required** application/json

Edit Value | Schema

```
{
  "bookId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6",
  "userId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6"
}
```

Execute

Output yang diharapkan:

Expected Response (200 OK):

```
{
  "loanId": "6e9e46db-62d3-4c1c-aef0-d3bc0b2ceef1",
  "bookId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6",
  "userId": "3fa80f64-5717-42e2-b3fc-2c983f6befa6",
}
```

```
"status": "borrowed",
"createdAt": "2025-11-16T20:18:51.179017",
"dueDate": "2025-11-24"
}
```

Code

Details

200

Response body

```
{
  "loanId": "76c61719-bd94-43e5-bd39-7055c943e140",
  "bookId": "3fa80f64-5717-42e2-b3fc-2c983f6bfa6",
  "userId": "3fa80f64-5717-42e2-b3fc-2c983f6bfa6",
  "status": "borrowed",
  "createdAt": "2025-11-17T19:34:38.688675",
  "dueDate": "2025-11-24"
}
```

Download

Response headers

```
content-length: 229
content-type: application/json
date: Mon, 17 Nov 2025 12:34:37 GMT
server: uvicorn
```

b. Test Case 2: Create Loan (GET /loans)

Klik **GET /loans/{loan_id}** -> Tekan tombol **Try it out** -> Masukkan **loanId** -> Klik **Execute**

loanId:
{afca1b5f-37d4-40f5-aa97-89f527d2e385}

GET

/loans/{loan_id}

Get Loan

^

Parameters

Cancel

Name	Description
loan_id * required	
string(\$uuid)	afca1b5f-37d4-40f5-aa97-89f527d2e385
(path)	

Execute

Clear

Output yang diharapkan:

Expected Response (200 OK):

```
{
  "loanId": "ded9d08b-5020-4a3c-ae02-d3bc0b2caaef",
  "bookId": "3fa80f64-5717-42e2-b3fc-2c983f6bfa6",
  "userId": "3fa80f64-5717-42e2-b3fc-2c983f6bfa6",
  "status": "borrowed",
  "createdAt": "2025-11-16T20:18:51.179017",
  "dueDate": "2025-11-24"
}
```

loan_id * required
string(\$uuid)
(path)

afca1b5f-37d4-40f5-aa97-89f527d2e385

ExecuteClear

Responses

Curl

curl -X 'GET' \
'http://127.0.0.1:8000/loans/afca1b5f-37d4-40f5-aa97-89f527d2e385' \
-H 'accept: application/json'

Request URL

http://127.0.0.1:8000/loans/afca1b5f-37d4-40f5-aa97-89f527d2e385

Server response

CodeDetails

200

Response body

{
 "loanId": "afca1b5f-37d4-40f5-aa97-89f527d2e385",
 "bookId": "3fa85f64-5717-4562-b3fc-2c963f66afae",
 "userId": "3fa85f64-5717-4562-b3fc-2c963f66afae",
 "status": "borrowed",
 "createdAt": "2025-11-17T20:07:11.628756",
 "dueDate": "2025-11-24"
}

Download

Response headers

content-length: 229
content-type: application/json
date: Mon, 17 Nov 2025 13:12:02 GMT
server: uvicorn

c. Test Case 3: Invalid (GET /loans)

Klik **GET /loans/{loan_id}** -> Tekan tombol **Try it out** -> Masukkan **ID random** -> Klik **Execute**
loanId:
11111111-1111-1111-1111-111111111111

Output yang diharapkan:
Response **404 Not Found**

Detail:

```
{ "detail": "Loan not found" }
```

CodeDetails

404
Undocumented

Error: Not Found

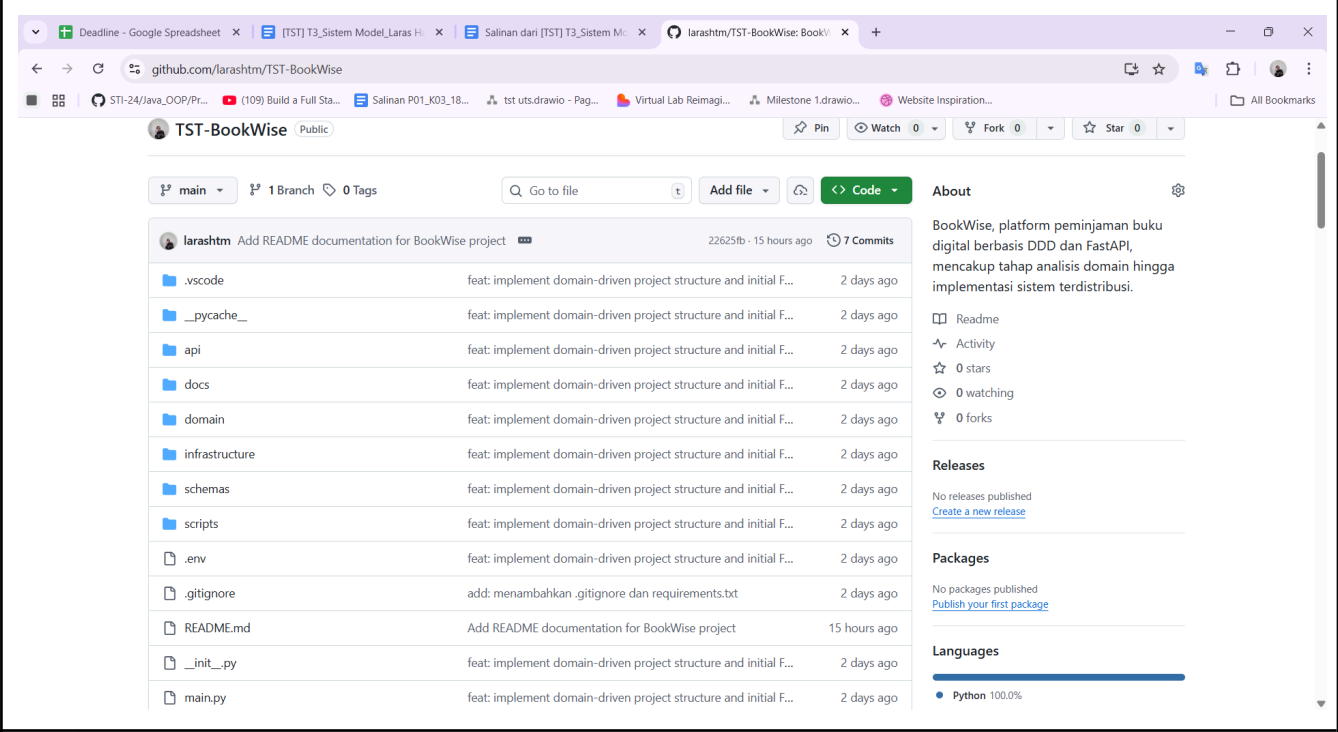
Response body

{
 "detail": "Loan not found"
}

Download

Response headers

content-length: 27
content-type: application/json
date: Mon, 17 Nov 2025 12:59:26 GMT
server: uvicorn

422	<div>Validation Error</div> <div>Media type application/json</div> <div>Example Value Schema</div> <pre>{ "detail": [{ "loc": ["string",], "msg": "string", "type": "string" }] }</pre>
8	Testing API (Swagger)
<p>Swagger UI (Interactive): http://127.0.0.1:8000/docs ReDoc (Alternative): http://127.0.0.1:8000/redoc</p> <ul style="list-style-type: none"> • Test Case dilakukan: <ul style="list-style-type: none"> - POST /loans → berhasil membuat loan - GET /loans/{id} → berhasil mengambil data - GET invalid ID → return 404 (sesuai ekspektasi) 	
9	Repository Git Hub
<ul style="list-style-type: none"> • URL repo: https://github.com/larashtm/TST-BookWise 	
	
10	Link Youtube
<ul style="list-style-type: none"> • URL repo: https://youtu.be/NU0c0JwMM70 	