Tugas Besar Milestone 4
Implementasi Awal
II3160 - Integrated Systems Technology

Diampu oleh:
Daniel Wiyogo Dwiputro, S.T., M.T.

Disusun oleh :
Nakeisha Valya Shakila
18223133

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

DAFTAR ISI

BAB I
PENDAHULUAN

1.1. Latar Belakang Masalah

Pada tahap analisis perancangan sistem yang telah dilakukan sebelumnya, berbagai model domain seperti Aggregate Root, Entity, dan Value Object telah berhasil diidentifikasi melalui pendekatan Domain-Driven Design (DDD). Model tersebut menggambarkan struktur logis dari domain bisnis, lengkap dengan aturan, batasan, serta proses yang terjadi di dalamnya. Namun, seluruh model tersebut masih berada pada tingkat konseptual dan belum dapat digunakan secara fungsional dalam sebuah sistem nyata. Agar model tersebut dapat dimanfaatkan oleh aplikasi maupun layanan lain, diperlukan langkah lanjutan berupa penerjemahan model abstrak tersebut ke dalam bentuk implementasi kode yang dapat dijalankan. Tantangan muncul ketika proses transformasi ini tidak hanya sekadar memindahkan struktur diagram ke dalam kode, tetapi harus memastikan bahwa setiap aturan domain, invariant, serta alur kerja yang telah dirancang tetap konsisten saat diimplementasikan. Di sisi lain, sistem juga memerlukan antarmuka yang memungkinkan operasi-operasi seperti pembuatan trip, proses booking, hingga transaksi pembayaran dapat dilakukan melalui permintaan API. FastAPI dipilih sebagai framework utama karena kemampuannya menyediakan API yang cepat, modern, dan mudah diintegrasikan dengan struktur arsitektur berbasis DDD. Dengan demikian, tahap implementasi ini menjadi krusial untuk menjembatani model konseptual yang sudah ada menjadi sebuah sistem backend yang fungsional, terstruktur, dan dapat diakses melalui API dasar sebagai fondasi pengembangan di tahapan selanjutnya.

1.2. Rumusan Masalah

Dari latar belakang yang ada, terdapat beberapa rumusan masalah yang diharapkan dapat terjawab setelah membaca laporan ini, antara lain sebagai berikut:

- Bagaimana menerjemahkan Aggregate Root, Entity, dan Value Object yang telah dirancang sebelumnya menjadi struktur kode Python yang sesuai prinsip DDD?
- Bagaimana memastikan bahwa aturan domain (invariant) tetap terjaga saat diterapkan ke dalam kode?
- Bagaimana membangun API dasar menggunakan FastAPI yang dapat menangani operasi utama seperti pembuatan booking, pengelolaan trip, transaksi pembayaran, dan perubahan status domain lainnya?
- Bagaimana memastikan API yang diimplementasikan dapat dijalankan, diuji, dan digunakan oleh sistem lain melalui Postman?

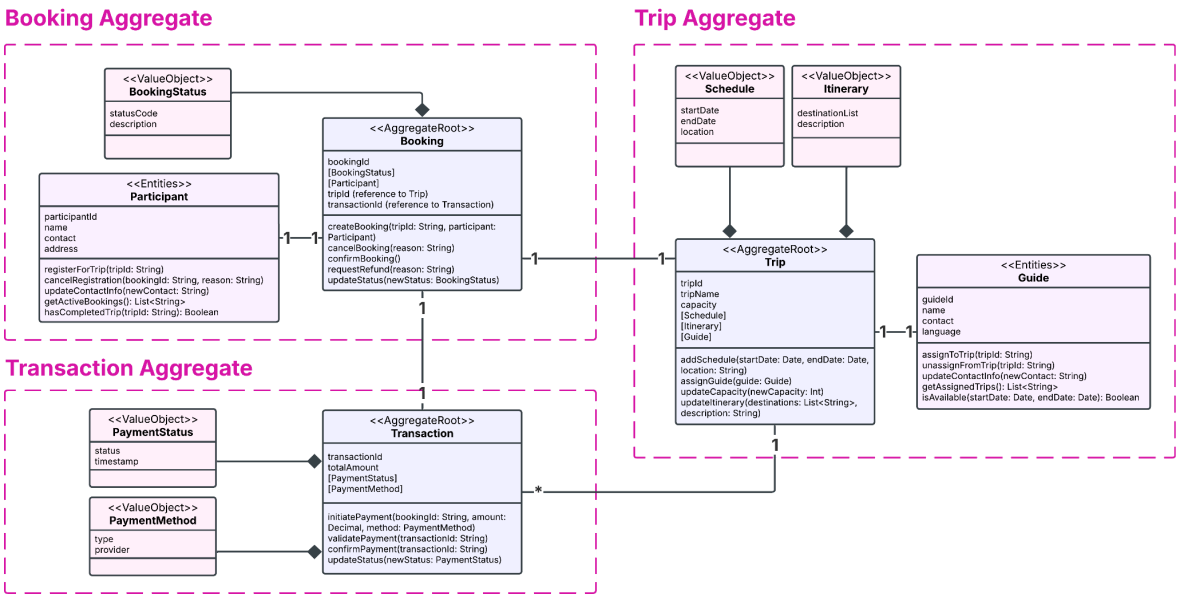
1.3. Tujuan

- Mengimplementasikan model Aggregate ke dalam kode yang sesuai aturan domain.
- Membangun API dasar dengan FastAPI untuk memungkinkan proses trip, booking, dan payment berjalan melalui layanan terstruktur.

BAB II

PEMBAHASAN

2.1. Analisis Model Domain



Gambar 2.1 Core Context Class Diagram DDD Principle

Berdasarkan milestone sebelumnya, telah ditetapkan tiga aggregate utama yang membentuk proses inti dalam *Booking Context*. Setiap aggregate memiliki *aggregate root*, *value object*, serta *entities* yang nantinya akan direalisasikan sebagai class dalam implementasi kode program. Penentuan aturan bisnis (*invariants*) dalam sebuah **Aggregate** memastikan bahwa setiap objek domain selalu berada dalam keadaan yang valid, konsisten, dan tidak dapat masuk ke kondisi yang melanggar aturan bisnis inti. Invariants ini dicek dan ditegakkan di dalam Aggregate Root melalui metode atau logika yang mengontrol perubahan state.

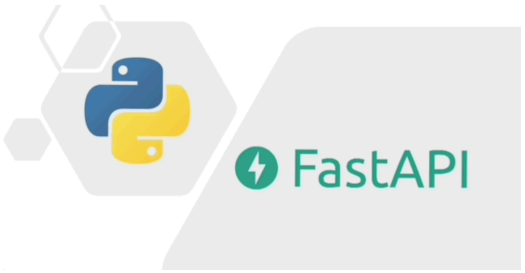
Aggregate	Invariant	Penjelasan Singkat
Trip	`end_date > start_date`	Trip tidak boleh berakhir sebelum mulai.
	`capacity.min_capacity >= 1` dan `capacity.max_capacity >= capacity.min_capacity`	Kapasitas minimum harus positif dan maksimum tidak boleh lebih kecil dari minimum.
	`current_bookings ∈ [0, capacity.max_capacity]`	Jumlah booking tidak boleh melebihi kapasitas.
	Status harus mengikuti transisi valid sdengan status PUBLISHED	Mencegah perubahan status yang tidak sah.
Booking	Tidak boleh mengonfirmasi booking yang bukan PENDING.	Menjaga transisi status tetap valid.

	Tidak boleh menandai PAID sebelum status CONFIRMED.	Pembayaran hanya bisa dilakukan setelah booking disetujui.
Transaction	`amount > 0`	Nominal transaksi tidak boleh nol atau negatif.
	Status transisi valid (SUCCESS/PENDING/FAILED).	Mengontrol alur status transaksi.

2.2. Lingkungan Sistem

Berdasarkan milestone sebelumnya, implementasi pada tahap ini difokuskan pada tiga aggregate utama yang membentuk proses inti dalam Booking Context, yaitu Trip, Booking, dan Transaction. Ketiganya memiliki *aggregate root*, *value object*, serta *entities* yang akan direalisasikan sebagai class dalam implementasi kode program, sementara aggregate lain seperti *Feedback*, *Notification*, *SupportTicket*, dan *FinancialReport* masih berada pada tahap desain dan akan menjadi ruang lingkup pengembangan di masa depan.

2.2.1. Teknologi yang digunakan



Gambar 2.2 Logo Python dan FastAPI

Project *open-trip-system* menggunakan Python 3.13.5 dengan **FastAPI** sebagai framework utama untuk membangun API yang cepat, lengkap dengan validasi otomatis melalui Pydantic serta dokumentasi OpenAPI. Objek domain seperti Aggregate, Entities, dan Value Objects ditulis menggunakan dataclasses agar struktur tetap sederhana, sementara **Uvicorn** digunakan sebagai ASGI server. Selama tahap pengembangan, data disimpan melalui in-memory storage pada `storage.py` yang bertindak sebagai repository sementara sebelum terhubung ke database sebenarnya, dan CORS diaktifkan untuk memungkinkan akses dari frontend. Pemasangan FastAPI dan Uvicorn diperlukan agar sistem dapat dibangun dan dijalankan melalui command line.

```
pip install fastapi uvicorn
```

2.2.2. Struktur File

Struktur sistem dipisahkan menjadi **frontend** dan **backend** agar arsitekturnya tetap fleksibel dan mudah dikembangkan di masa depan. Pemisahan ini memastikan bahwa jika nantinya da penambahan *interface*, maka seluruh fungsi backend tetap dapat digunakan ulang tanpa perubahan besar.

```
open-trip-system/
├── backend/
│   ├── booking/
│   │   ├── aggregate_root.py # Booking Aggregate
│   │   ├── entities.py
│   │   ├── booking_api.py
│   │   └── value_objects.py
│   │
│   ├── transaction/
│   │   ├── aggregate_root.py # Transaction Aggregate
│   │   ├── transaction_api.py
│   │   └── value_objects.py
│   │
│   ├── trip/
│   │   ├── aggregate_root.py # Trip Aggregate
│   │   ├── entities.py
│   │   ├── trip_api.py
│   │   └── value_objects.py
│   │
│   ├── main.py
│   └── storage.py
│
└── frontend/
```

.Arsitektur backend mengikuti prinsip Domain-Driven Design dengan tiga aggregate utama dalam Booking Context, yaitu Booking, Trip, dan Transaction, yang masing-masing menegakkan aturan bisnis melalui Aggregate Root, Entities, dan Value Objects sesuai kebutuhan, di mana Trip direalisasikan sebagai `class Trip` pada `trip/aggregate_root.py`, Booking direpresentasikan sebagai `class Booking` pada `booking/aggregate_root.py`, dan Transaction diwujudkan sebagai `class Transaction` pada `transaction/aggregate_root.py`. Lapisan API berfungsi sebagai Application Service yang menerjemahkan Data Transfer Object (DTO) ke objek domain serta mengeksekusi logika bisnis melalui endpoint yang menjadi antarmuka HTTP tiap aggregate. Seluruh API kemudian digabungkan melalui `main.py` yang menjadi titik masuk aplikasi FastAPI, menginisialisasi konfigurasi seperti CORS, serta mendaftarkan router agar backend berjalan sebagai layanan terpadu. Proses penyimpanan data ditangani oleh `storage.py` sebagai *in-memory* repository yang menyediakan operasi CRUD sederhana tanpa ketergantungan pada teknologi database tertentu, sehingga memudahkan migrasi ke penyimpanan ke *database* tanpa mengubah struktur domain yang sudah dibangun.

2.3. Implementasi Logika Domain melalui Aggregate Root dalam API

Sebelum melihat detail endpoint, perlu dipahami bahwa setiap aggregate dalam backend diekspos melalui API terpisah agar aturan bisnis di dalam aggregate root dapat diakses secara terstruktur dan konsisten, dengan tiap endpoint mewakili operasi domain yang telah divalidasi melalui logika masing-masing aggregate. Gunakan base url menggunakan url yang dihasillkan dari hasil run

server dan pastikan setiap ingin mengecek api maka server harus sudah dijalankan menggunakan uvicorn di path letak main.py

2.3.1. Endpoint Trip

Tabel berikut menyajikan daftar endpoint yang tersedia pada Aggregate Trip melalui file `trip_api.py`.

Endpoint	Method	Deskripsi Singkat
`/trips`	POST	Membuat trip baru
`/trips/{trip_id}/schedule`		Membuat jadwal baru untuk trip
`/trips/{trip_id}/guide`		Meng-*assign Guide* untuk trip
`/trips/{trip_id}/capacity`	PUT	Memperbarui kapasitas trip
`/trips/{trip_id}/itinerary`		Memperbarui itinerary trip
`/trips/`	GET	Mengambil seluruh data trip
`/trips/{trip_id}`		Mengambil detail trip berdasarkan ID

2.3.2. Endpoint Booking

Tabel berikut menyajikan daftar endpoint yang tersedia pada Aggregate Booking melalui file `booking_api.py`.

Endpoint	Method	Deskripsi Singkat
`bookings/`	POST	Membuat booking baru.
`bookings/{booking_id}/confirm`		Mengonfirmasi booking (PENDING → CONFIRMED).
`bookings/{booking_id}/cancel`		Membatalkan booking.
`bookings/`	GET	Mengambil seluruh booking.
`bookings/{booking_id}`		Mengambil detail booking berdasarkan ID.

2.3.3. Endpoint Transaction

Tabel berikut menyajikan daftar endpoint yang tersedia pada Aggregate Transaction melalui file `transaction_api.py`.

Endpoint	Method	Deskripsi Singkat
`transactions/`	POST	Membuat transaksi baru.
`transactions/{transaction`		Memulai pemrosesan transaksi.

`_id}/validate`		
`transactions/{transaction_id}/confirm`		Menyelesaikan transaksi.
`transactions/{transaction_id}/refund`		Menandai transaksi gagal.
`transactions/`	GET	Mengambil seluruh transaksi.
`transactions/{transaction_id}`		Mengambil detail transaksi berdasarkan ID.

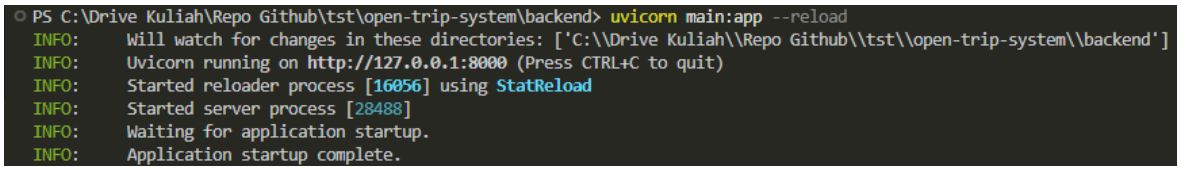
Alur dimulai dari Trip, yaitu paket perjalanan yang dibuat dan dipublikasikan oleh penyedia layanan. Setelah Trip berstatus Published dan tersedia untuk dipesan, pengguna dapat membuat Booking dengan memilih trip tertentu dan mengisi jumlah peserta. Booking yang dibuat akan divalidasi sesuai aturan domain seperti kapasitas serta status Trip, kemudian masuk ke status Pending. Untuk menyelesaikan pesanan, sistem membuat Transaction yang merepresentasikan proses pembayaran terhadap booking tersebut. Transaction akan diproses hingga berhasil (Success) atau gagal (Failed). Jika pembayaran berhasil, Booking diperbarui menjadi Paid atau Confirmed sesuai aturan bisnis. Dengan demikian, Trip menyediakan data perjalanan, Booking mengatur pemesanan dan kapasitas peserta, dan Transaction menangani proses pembayaran sebagai tahap akhir dari alur pemesanan.

2.4. Pengujian Sistem

Gambar 2.3 Logo Postman

Untuk memastikan setiap endpoint pada 2.3 berjalan sesuai aturan bisnis dan alur domain, pengujian dilakukan menggunakan Postman sebagai alat untuk mengirim request HTTP ke API. Melalui Postman, setiap operasi pada aggregate seperti Trip, Booking, dan Transaction dapat diuji secara terstruktur dengan mencoba berbagai kombinasi input, memvalidasi respons, serta memastikan setiap transisi status dan aturan bisnis diproses dengan benar oleh backend. Gunakan base URL yang diambil dari server FastAPI setelah dijalankan, dan pastikan setiap pengujian endpoint dilakukan ketika server sudah aktif melalui perintah uvicorn pada direktori tempat file `main.py` berada.

```
PS C:\Drive Kuliah\Repo Github\tst\open-trip-system\backend>
uvicorn main:app --reload
```



Gambar 2.4 Server Berhasil Dijalankan pada CLI

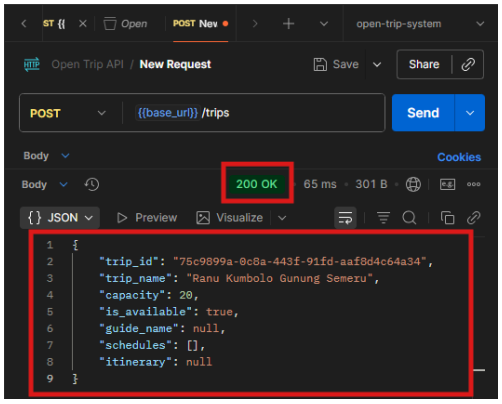
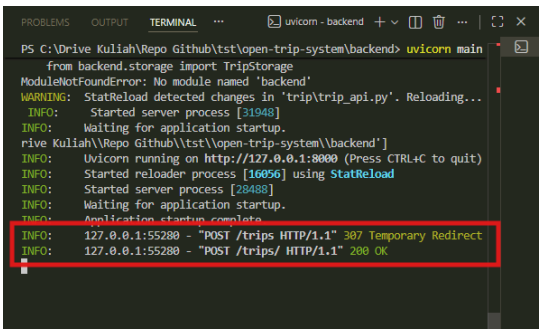
Setelah server berjalan pada alamat http://127.0.0.1:8000, tambahkan variabel baru bernama **base_url** di environment Postman dengan *value* URL tersebut. Setelah itu, pengujian endpoint bisa dilakukan melalui *request* yang sudah dibuat di dalam collection. Untuk memperjelas ruang lingkup, dokumentasi dibawah ini menegaskan bahwa API yang dibuat saat ini merealisasikan Booking Context sebagaimana didefinisikan pada milestone 2 dan 3.

2.4.1. Uji Coba Endpoint Trip

Tabel-tabel berikut berisi dokumentasi uji coba yang dilakukan pada postman

Nama Endpoint	`/trips`	**Method**	POST
URL	`http://127.0.0.1:8000/trips`		
Body			

<pre>{
"trip_name": "Ranu Kumbolo Gunung Semeru",
"capacity": 20
}</pre>			

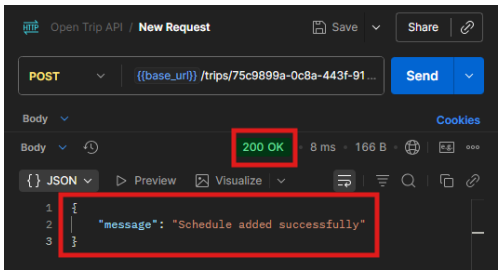
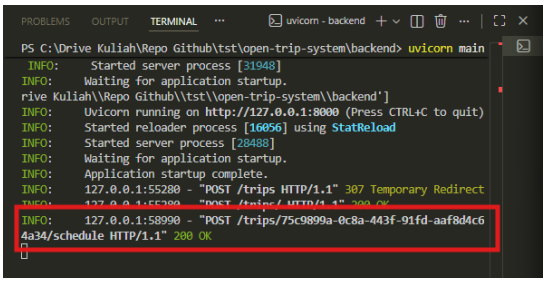
Response di Postman		**Response di Server**	
			
trip_id = 75c9899a-0c8a-443f-91fd-aaf8d4c64a34 digunakan untuk uji coba endpoint selanjutnya			

Nama Endpoint	`/trips/{trip_id}/schedule`	**Method**	POST
URL	`http://127.0.0.1:8000/trips/75c9899a-0c8a-44`		

	3f-91fd-aaf8d4c64a34/schedule

Body

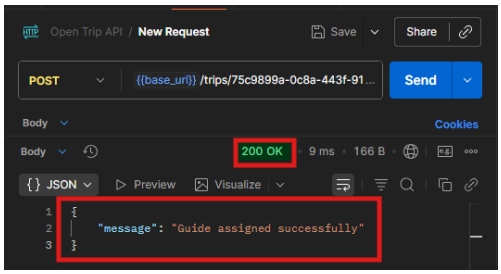
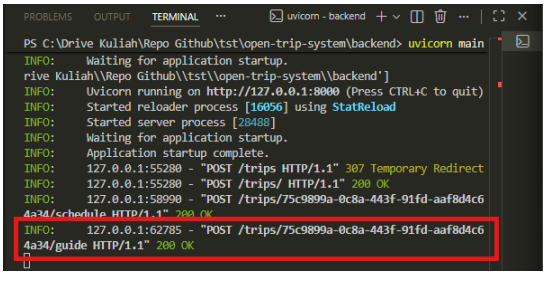
```
{
  "start_date": "2025-12-12",
  "end_date": "2025-12-14",
  "location": "Kabupaten Lumajang, Jawa Timur, Indonesia"
}
```

Response di Postman	Response di Server
	

Nama Endpoint	/trips/{trip_id}/guide	Method	POST
URL	http://127.0.0.1:8000/trips/*75c9899a-0c8a-443f-91fd-aaf8d4c64a34*/guide		

Body

```
{
  "guide_name": "Mike Wheeler",
  "contact": "08123456789",
  "language": "English"
}
```

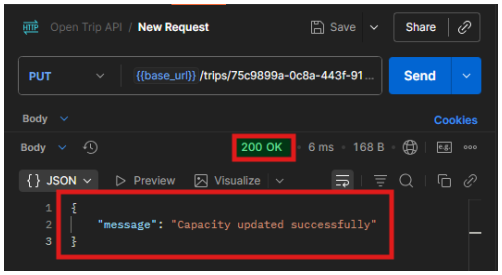
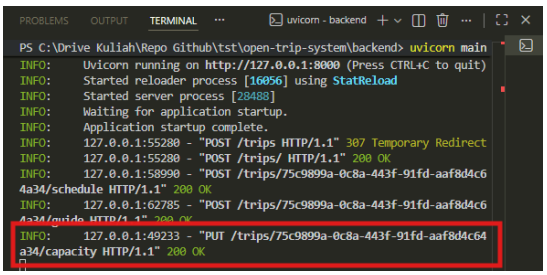
Response di Postman	Response di Server
	

Nama Endpoint	/trips/{trip_id}/capacity	Method	PUT

URL	http://127.0.0.1:8000/trips/*75c9899a-0c8a-443f-91fd-aaf8d4c64a34*/capacity

Body

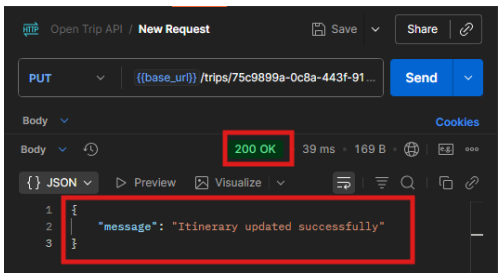
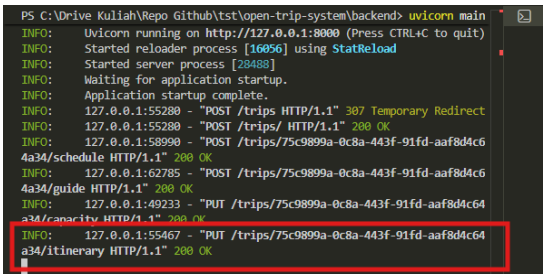
```
{
  "new_capacity": 15
}
```

Response di Postman	Response di Server
	

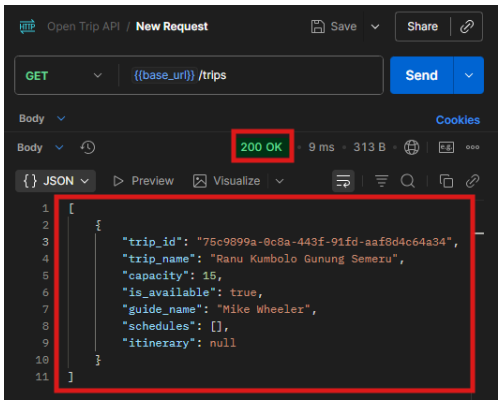
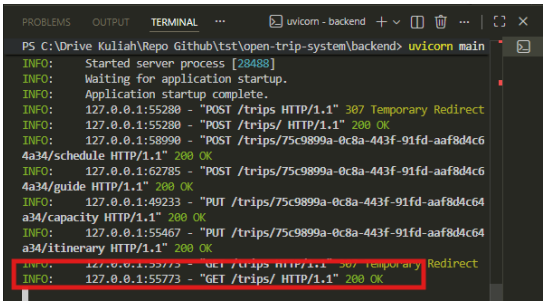
Nama Endpoint	/trips/{trip_id}/itinerary	Method	PUT
URL	http://127.0.0.1:8000/trips/*75c9899a-0c8a-443f-91fd-aaf8d4c64a34*/itinerary		

Body

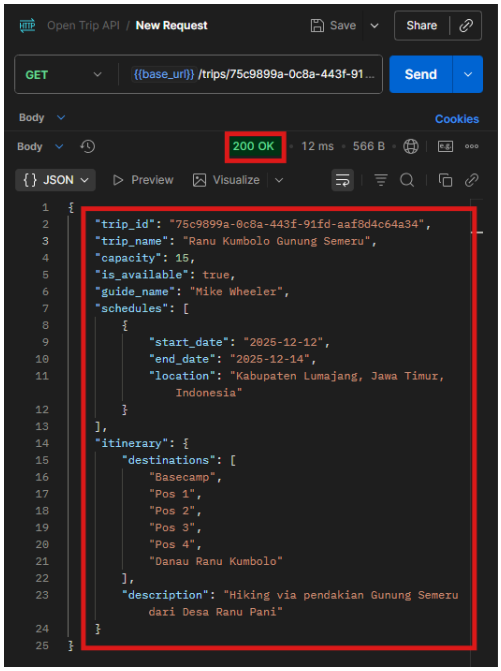
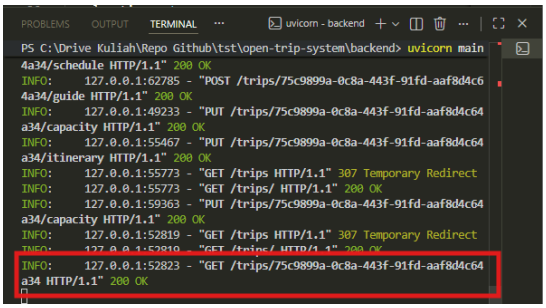
```
{
  "destinations": ["Basecamp", "Pos 1", "Pos 2", "Pos 3",
"Pos 4", "Danau Ranu Kumbolo"],
  "description": "Hiking via pendakian Gunung Semeru dari
Desa Ranu Pani"
}
```

Response di Postman	Response di Server
	

Nama Endpoint	/trips	Method	GET
URL	http://127.0.0.1:8000/trips		

Response di Postman	Response di Server
	

Nama Endpoint	/trips/	Method	GET
URL	http://127.0.0.1:8000/trips/*75c9899a-0c8a-443f-91fd-aaf8d4c64a34*		

Response di Postman	Response di Server
	

2.4.2. Uji Coba Endpoint Booking

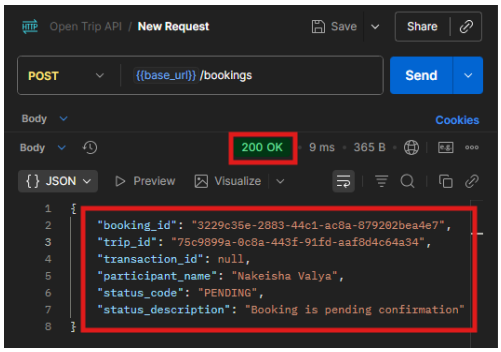
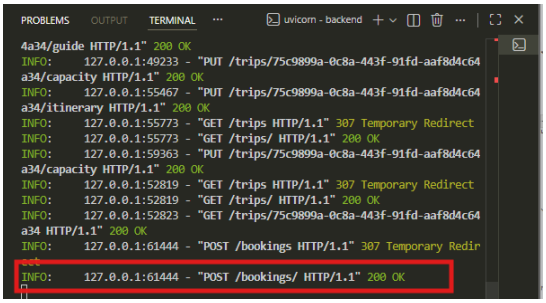
Tabel-tabel berikut berisi dokumentasi uji coba yang dilakukan pada postman

Nama Endpoint	/bookings	Method	POST

URL	http://127.0.0.1:8000/bookings

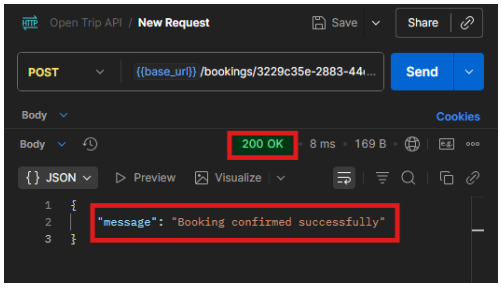
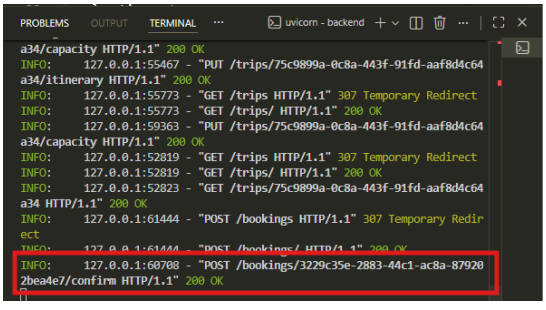
Body

```
{
  "trip_id": "75c9899a-0c8a-443f-91fd-aaf8d4c64a34",
  "participant": {
    "name": "Nakeisha Valya",
    "contact": "081234567890",
    "address": "Bandung"
  }
}
```

Response di Postman	Response di Server
	

booking_id = 3229c35e-2883-44c1-ac8a-879202bea4e7
digunakan untuk uji coba endpoint selanjutnya

Nama Endpoint	/bookings/{booking_id}/confirm	Method	POST
URL	http://127.0.0.1:8000/bookings/*3229c35e-2883-44c1-ac8a-879202bea4e7*/confirm		

Response di Postman	Response di Server
	

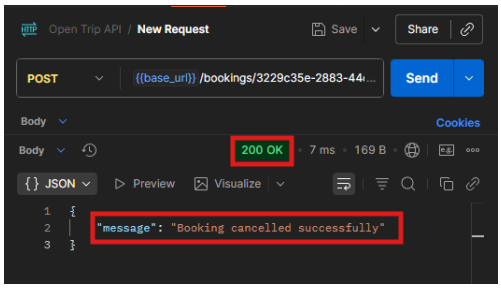
Nama Endpoint	/bookings/{booking_id}/cancel	Method	POST

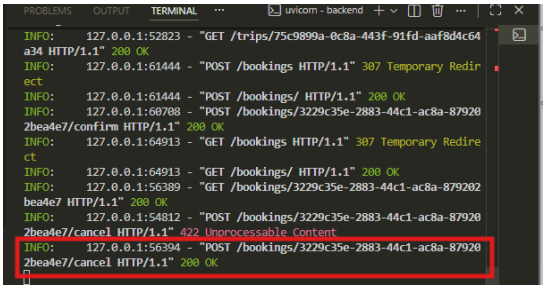
URL	http://127.0.0.1:8000/bookings/*3229c35e-2883-44c1-ac8a-879202bea4e7*/cancel

Body

```
{
  "reason": "Customer changed plan"
}
```

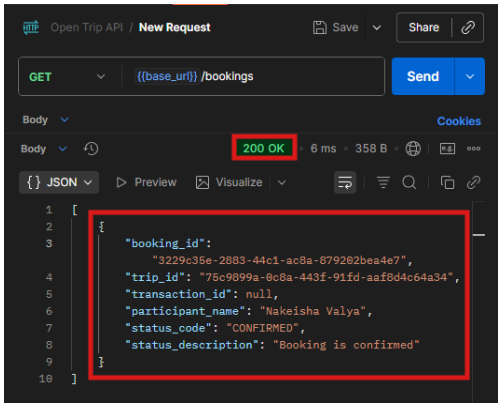
Response di Postman	Response di Server

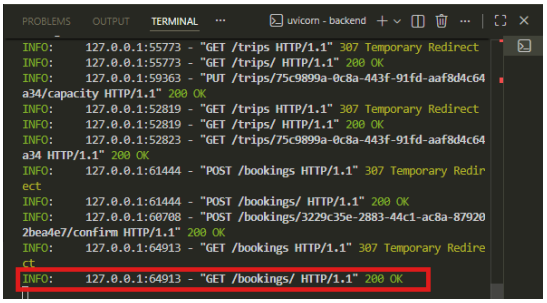




Nama Endpoint	/bookings	Method	GET
URL	http://127.0.0.1:8000/bookings		

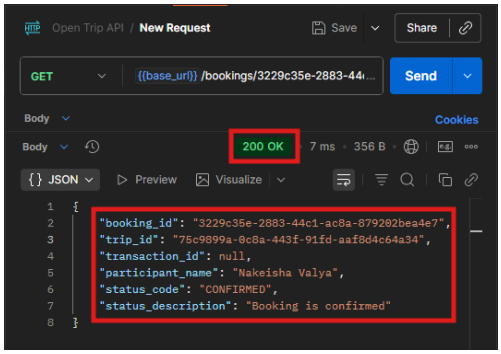
Response di Postman	Response di Server

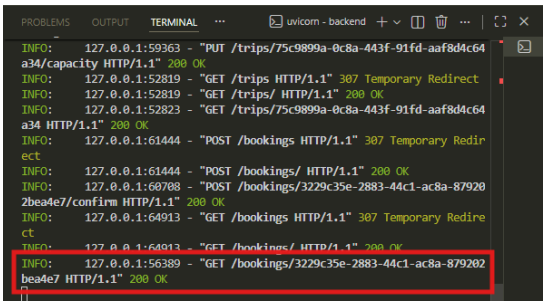




Nama Endpoint	/bookings/{booking_id}	Method	GET
URL	http://127.0.0.1:8000/bookings/*3229c35e-2883-44c1-ac8a-879202bea4e7*		

Response di Postman	Response di Server





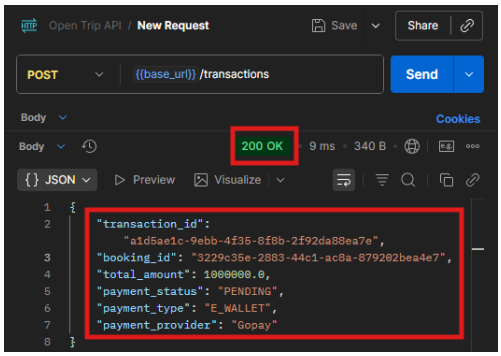
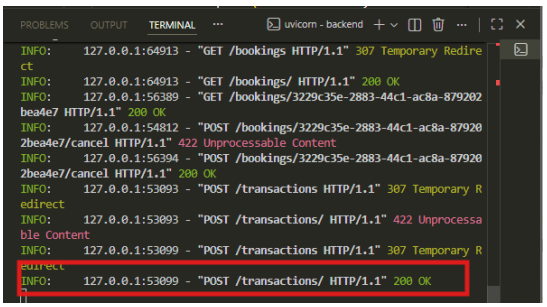
2.4.3. Uji Coba Endpoint Transactions

Tabel-tabel berikut berisi dokumentasi uji coba yang dilakukan pada postman

Nama Endpoint	`/transactions`	**Method**	POST
URL	`http://127.0.0.1:8000/transactions`		

Body

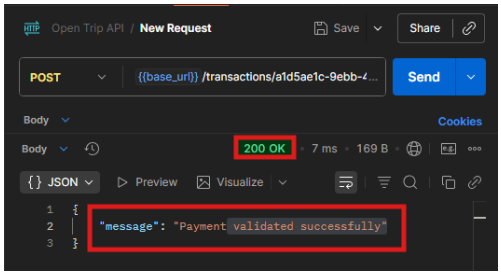
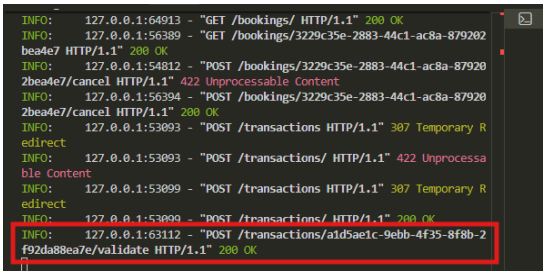
```
{
  "booking_id": "3229c35e-2883-44c1-ac8a-879202bea4e7",
  "amount": 1000000,
  "payment_type": "E_WALLET",
  "provider": "Gopay"
}
```

Response di Postman	**Response di Server**
	

`transactions_id = a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e` digunakan untuk uji coba endpoint selanjutnya

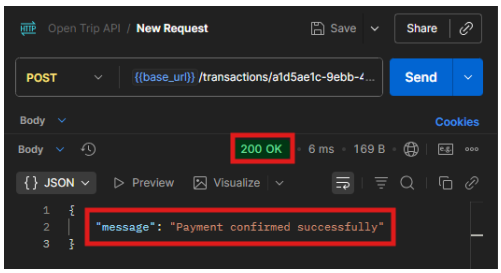
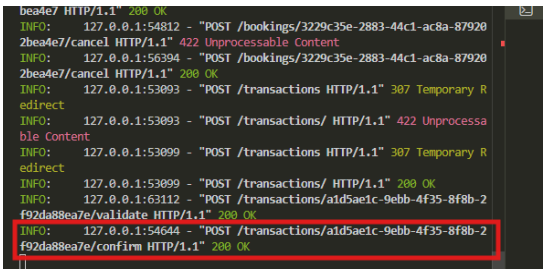
Nama Endpoint	`/transactions/{transaction_id}/validate`	**Method**	POST
URL	`http://127.0.0.1:8000/transactions/a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e/validate`		

Body
–

Response di Postman	Response di Server
	

Nama Endpoint	/transactions/{transaction_id}/confirm	Method	POST
URL	http://127.0.0.1:8000/transactions/*a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e*/confirm		

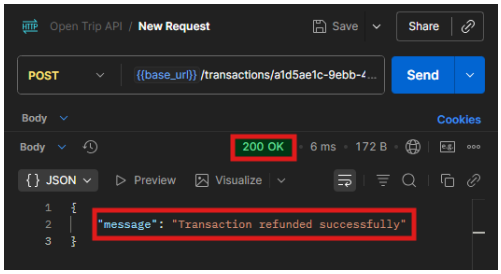
Body
–

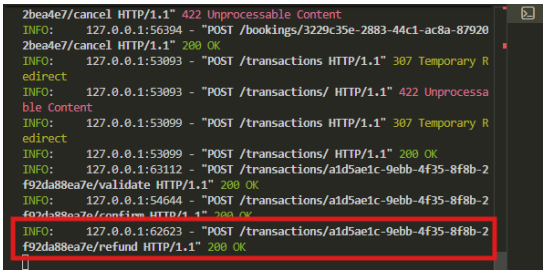
Response di Postman	Response di Server
	

Nama Endpoint	/transactions/{transaction_id}/refund	Method	POST
URL	http://127.0.0.1:8000/transactions/*a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e*/refund		

Body
–

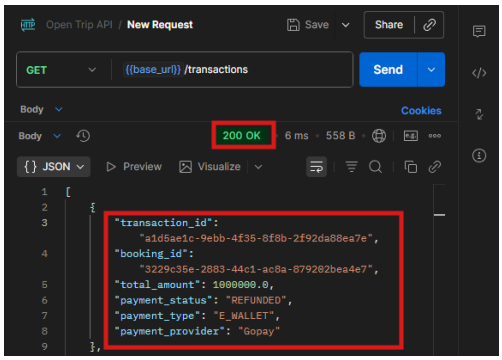
Response di Postman	Response di Server

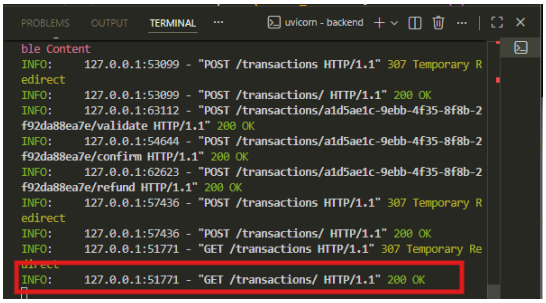




Nama Endpoint	/transactions	Method	GET
URL	http://127.0.0.1:8000/transactions		

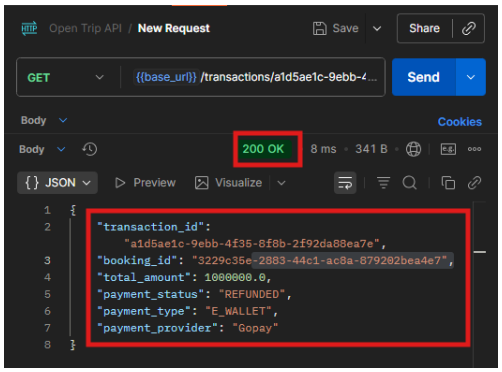
Response di Postman	Response di Server

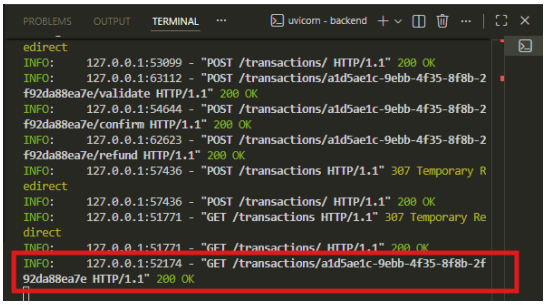




Nama Endpoint	/transactions/{transaction_id}	Method	GET
URL	http://127.0.0.1:8000/transactions/*a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e*		

Response di Postman	Response di Server





BAB III
PENUTUP

3.1. Kesimpulan

Penerjemahan model Aggregate ke dalam kode dan pembangunan API dasar menggunakan FastAPI membuktikan bahwa konsep Domain-Driven Design (DDD) dapat diimplementasikan secara efektif untuk menciptakan sistem yang terstruktur dan mudah dikembangkan. Dengan memisahkan domain menjadi aggregate seperti Trip, Booking, dan Transaction, setiap proses bisnis dapat dikelola secara konsisten melalui aturan domain yang jelas. Implementasi API berbasis FastAPI memberikan fondasi layanan yang cepat, modular, dan mudah diuji. Melalui integrasi antara aggregate dan endpoint, alur utama seperti pembuatan trip, pemesanan, hingga pembayaran dapat berjalan secara terstandardisasi. Tahapan ini menjadi pondasi penting untuk pengembangan fitur lebih lanjut serta memastikan sistem dapat dikembangkan secara berkelanjutan dan terukur.

Link Video Demo : [Video Demo] M04_II3160
Link Source Code : https://github.com/NakeishaValya/open-trip-system

DAFTAR PUSTAKA

GeeksforGeeks. (2025, September 3). Introduction to Postman for API development. GeeksforGeeks. Retrieved November 17, 2025, from https://www.geeksforgeeks.org/web-tech/introduction-postman-api-development/

Montgomery, M. (2020, June 24). API Testing with Postman — Getting Started. Medium. https://medium.com/assertqualityassurance/rest-api-test-automation-with-postman-jenkins-1-of-3-860edf3c2a45

Io, L. (2025, January 10). FastAPI + Uvicorn = Blazing Speed: The Tech Behind the Hype. Medium. Retrieved November 17, 2025, from https://leapcell.medium.com/fastapi-uvicorn-blazing-speed-the-tech-behind-the-hype-a205af1d9ee4

First steps - FastAPI. (n.d.). Retrieved November 17, 2025, from https://fastapi.tiangolo.com/tutorial/first-steps/