Tugas Besar Milestone 6
Finalisasi
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

Dalam pengembangan perangkat lunak modern, arsitektur yang baik saja tidak cukup untuk menjamin kualitas sistem; diperlukan mekanisme verifikasi yang ketat dan otomatisasi alur kerja untuk memastikan keandalan kode. Pada tahap pengembangan sebelumnya, yaitu Milestone 4 dan 5, desain taktis *Domain-Driven Design* (DDD) telah direalisasikan ke dalam struktur kode *backend*. Fokus utama implementasi ini terletak pada pembentukan tiga *aggregate* krusial dalam *Booking Context*, yaitu **Trip Aggregate** (mengelola data perjalanan, jadwal, dan *guide*), **Booking Aggregate** (pusat interaksi pemesanan), dan **Transaction Aggregate** (menangani validasi pembayaran). Keberadaan *aggregate root* tersebut membawa logika bisnis yang kompleks dan saling berketergantungan untuk menjalankan alur operasional secara *end-to-end*. Akibatnya, tantangan pengembangan kini bergeser dari sekadar perancangan domain menjadi pemastian bahwa setiap *aggregate* yang telah dibangun berjalan sesuai spesifikasi tanpa *bug* yang tersembunyi. Untuk menjawab tantangan kompleksitas tersebut, pendekatan *Test-Driven Development* (TDD) menjadi sangat krusial. TDD mengharuskan pengujian ditulis sebelum implementasi kode, memastikan bahwa setiap fungsi dan logika dalam *aggregate* memiliki tujuan yang jelas serta terverifikasi sejak awal. Guna menjamin stabilitas sistem yang telah terbentuk, standar kualitas tinggi ditetapkan dengan target *test coverage* minimal 95%, yang mencakup pengujian terhadap *edge cases*, penanganan *error*, aspek keamanan, hingga alur kerja (*workflow*) utama sistem. Selain aspek pengujian, efisiensi pengembangan juga perlu ditingkatkan melalui *Continuous Integration* (CI). Tanpa CI, proses verifikasi kode terhadap *aggregate* yang kompleks sering kali dilakukan secara manual dan rentan terlewat. Oleh karena itu, penerapan GitHub Workflows diperlukan untuk mengotomatisasi proses *linting* dan *testing* setiap kali terdapat perubahan kode, sehingga integrasi antar-modul tetap terjaga kualitasnya.

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 menerapkan TDD untuk memverifikasi fungsionalitas sistem sejak awal?
- Bagaimana menyusun unit testing komprehensif hingga coverage minimal 95%, mencakup skenario positif, error, dan edge cases?
- Bagaimana mengimplementasikan CI dengan GitHub Workflows serta melakukan deployment agar sistem dapat diakses publik?

1.3. Tujuan

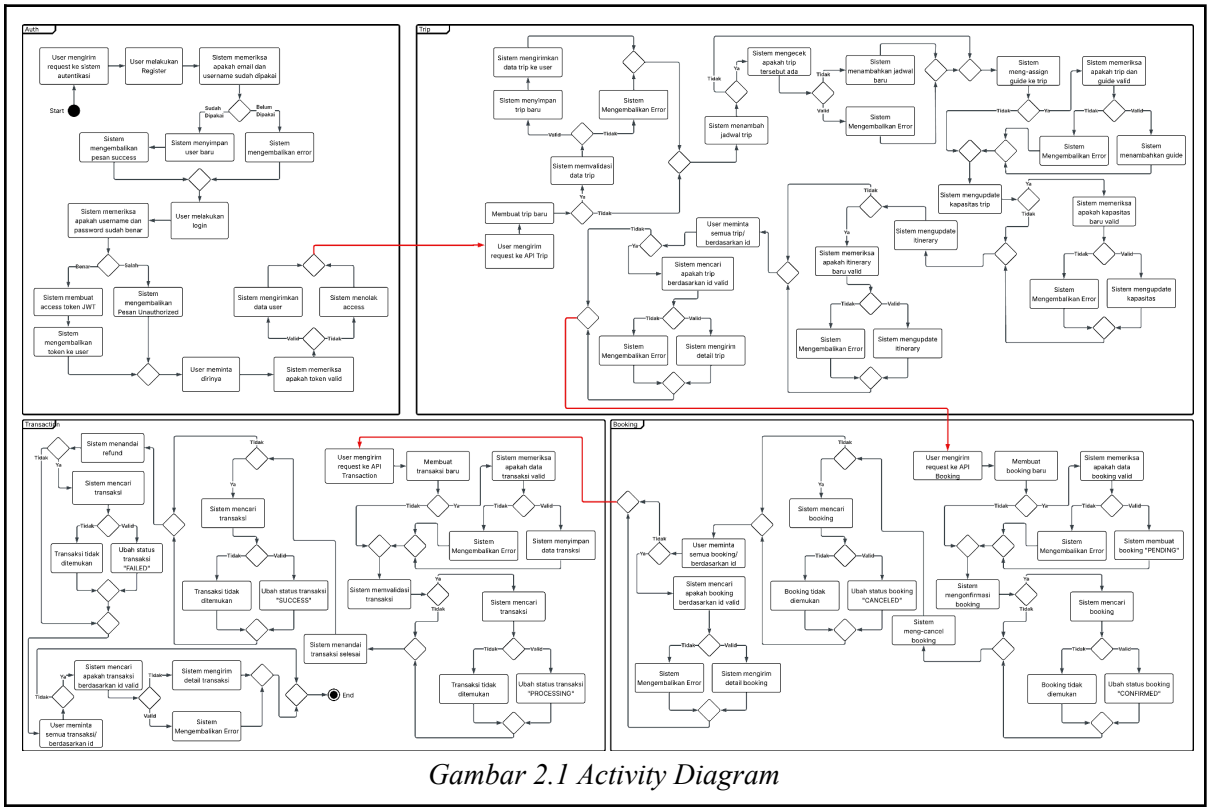
- Mengimplementasikan fitur sistem menggunakan siklus TDD (Red-Green-Refactor) untuk menjamin validitas logika bisnis.
- Mencapai code coverage di atas 95% dan pengujian menyeluruh pada seluruh modul utama.
- Membangun CI pipeline otomatis dan men-deploy aplikasi agar API dapat diakses secara real-time dengan dokumentasi lengkap.

BAB II

PEMBAHASAN

2.1. Rancangan Awal Sistem

Rancangan sistem Manajemen Booking Open Trip ini telah berevolusi dari pemetaan kapabilitas bisnis dan identifikasi domain strategis menjadi arsitektur yang terstruktur melalui *Context Mapping*, di mana *Booking System* ditetapkan sebagai *Core Domain*. Untuk tahap implementasi yang akan datang, ruang lingkup desain taktis dipersempit (dengan mengecualikan fitur pendukung seperti *Feedback* dan *Notification*) agar berfokus sepenuhnya pada tiga *aggregate* utama dalam *Booking Context*, yaitu **Trip Aggregate** (mengelola data perjalanan, jadwal, dan *guide*), **Booking Aggregate** (pusat interaksi pemesanan peserta), dan **Transaction Aggregate** (menangani validasi pembayaran), karena ketiganya merupakan fondasi operasional yang paling krusial untuk menjalankan alur bisnis utama secara *end-to-end*



Gambar 2.1 Activity Diagram

Berikut merupakan activity diagram dari tiga aggregate utama yang ditambahkan dengan proses autentikasi berbasis JWT, sehingga terdapat empat alur aktivitas dalam diagram tersebut. Setiap aktivitas memiliki beberapa kondisi dan percabangan logika yang menggambarkan alur proses secara lengkap.

2.2. Struktur File dan Penjelasan Modul

Pada pengembangan milestone ini, seluruh proses masih menggunakan folder yang sama seperti pada implementasi di milestone 5 dan 6, namun terdapat beberapa penyesuaian tertentu.

```
open-trip-system/
│
├── .coverage
├── .git/
├── .github/
│   └── workflows/
│       └── ci.yml
├── .venv/
├── backend/
│   ├── auth.py
│   ├── main.py
│   ├── requirements.txt
│   ├── storage.py
│   ├── booking/
│   │   ├── aggregate_root.py
│   │   ├── booking_api.py
│   │   ├── entities.py
│   │   └── value_objects.py
│   ├── transaction/
│   │   ├── aggregate_root.py
│   │   ├── transaction_api.py
│   │   └── value_objects.py
│   ├── trip/
│   │   ├── aggregate_root.py
│   │   ├── entities.py
│   │   ├── trip_api.py
│   │   └── value_objects.py
│   └── tests/
│       ├── test_auth.py
│       ├── test_booking.py
│       ├── test_main_edge.py
│       ├── test_storage.py
│       ├── test_transaction.py
│       └── test_trip.py
├── doc/
├── frontend/
├── pyproject.toml
└── README.md
```

Penyesuaian tersebut dilakukan pada struktur folder untuk memastikan proyek lebih modular. Struktur pengujian disempurnakan dengan **memusatkan seluruh unit test** dan **API test** di dalam *backend/tests/*, menggunakan satu file test untuk tiap domain, yaitu *test_auth.py*, *test_booking.py*, *test_transaction.py*, *test_trip.py*, *test_storage.py*, dan *test_main_edge.py*. Seluruh file test duplikat maupun yang sudah tidak relevan telah dihapus untuk mencegah redundansi dan menjaga efisiensi. Berdasarkan file pada folder tests/, terdapat enam testing dari masing masing modul yang mewakili pengujian autentikasi, booking, transaksi, trip, storage, serta pengujian edge-case pada sistem utama. Konteks modul disini berisi *aggregate* yang telah diidentifikasi pada milestone sebelumnya, dan beberapa komponen tambahan lainnya yang mendukung proses *aggregate* bekerja.

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 Autentikasi

Tabel berikut merupakan daftar endpoint yang digunakan dalam sistem autentikasi JWT beserta hak akses dan fungsinya melalui file `auth.py`.

Endpoint	Method	Akses	Deskripsi Singkat
`/auth/register`	POST	Publik	Registrasi user baru dengan validasi email dan username unik
`/auth/login`			Login dan mendapatkan access token
`/auth/me`	GET	Perlu Autentikasi (Bearer Token)	Mendapatkan informasi user yang sedang login

2.3.2. 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

2.3.3. 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.

2.3.4. 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_id}/validate`		Memulai pemrosesan transaksi.
`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 sistem dimulai dari Trip, yaitu paket perjalanan yang dibuat dan dipublikasikan oleh penyedia layanan. Setelah Trip berstatus *Published* dan tersedia untuk dipesan, pengguna yang ingin melakukan pemesanan harus melalui proses autentikasi terlebih dahulu. Pengguna baru dapat melakukan registrasi melalui endpoint `/auth/register`, kemudian login melalui endpoint `/auth/login`. Jika kredensial benar, sistem akan memberikan access token JWT yang digunakan sebagai Bearer Token untuk mengakses endpoint terproteksi seperti `/auth/me`.

Setelah pengguna terverifikasi, pemesanan dapat dilakukan melalui proses Booking dengan memilih Trip tertentu dan menentukan jumlah peserta. Booking yang dibuat akan divalidasi sesuai aturan domain, seperti kapasitas Trip serta status publikasinya, sebelum masuk dalam status *Pending*. Untuk menyelesaikan pemesanan, sistem membuat Transaction sebagai representasi proses pembayaran. Transaction kemudian diproses hingga menghasilkan status *Success* atau *Failed*. Jika pembayaran berhasil, Booking diperbarui menjadi *Paid* atau *Confirmed* sesuai aturan bisnis. Dengan demikian, autentikasi memastikan hanya pengguna sah yang dapat mengakses layanan, Trip menyediakan informasi perjalanan yang dapat dipesan, Booking mengelola proses pemesanan dan kapasitas peserta, sementara Transaction menjadi tahap akhir yang menangani pembayaran hingga pemesanan dinyatakan berhasil.

2.4. Implementasi Pengujian Sistem Menggunakan Postman

Gambar 2.2 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
```

```
PS C:\Drive Kuliah\Repo Github\tst\open-trip-system\backend> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Drive Kuliah\\Repo Github\\tst\\open-trip-system\\backend']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [16056] using StatReload
INFO:     Started server process [28488]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Gambar 2.3 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.

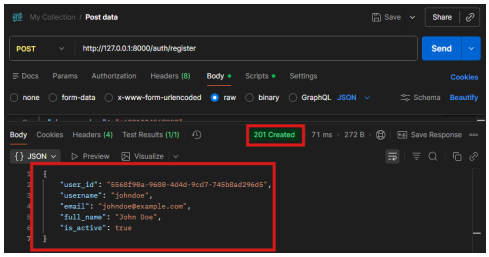
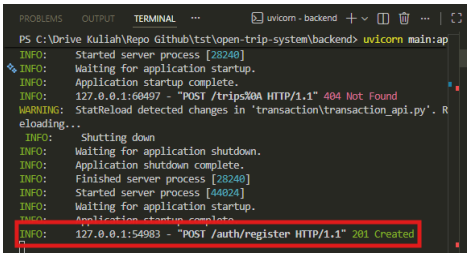
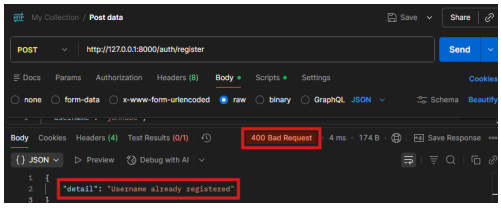
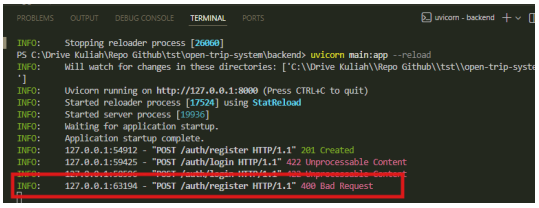
2.4.1. Uji Coba Endpoint Autentikasi

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

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

```
{
  "username": "johndoe",
  "email": "johndoe@example.com",
  "password": "SecurePassword123!",
  "confirm_password": "SecurePassword123!",
  "full_name": "John Doe",
  "phone_number": "+6281234567890",
  "address": "Jl. Sudirman No. 123, Jakarta",
  "date_of_birth": "1990-01-15"
```

}

Response di Postman	Response di Server
Berhasil login	
	
Gagal login akibat username telah digunakan sebelumnya	
	

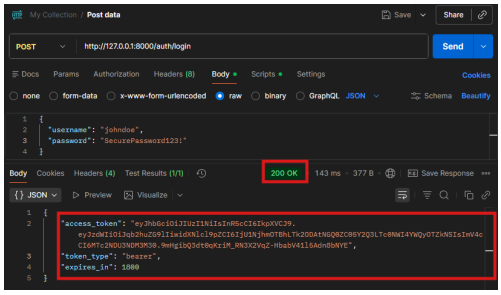
Informasi ini digunakan untuk uji coba endpoint selanjutnya
- Email : "johndoe@example.com"
- password": "SecurePassword123!"

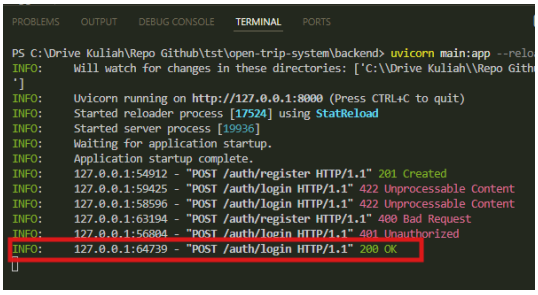
Nama Endpoint	/auth/login	Method	POST
URL	http://127.0.0.1:8000/auth/login		

Body

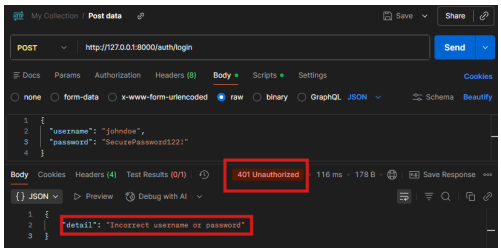
```
{
  "username": "johndoe",
  "password": "SecurePassword123!"
}
```

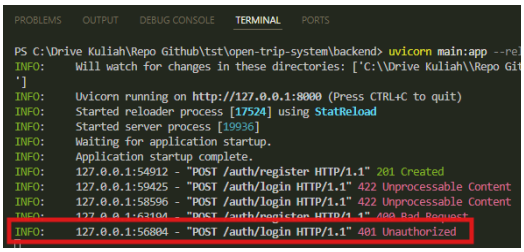
Response di Postman	Response di Server
Password dan Username benar (berhasil login)	





Password/Username salah (gagal login)

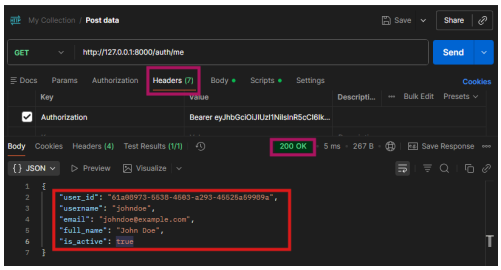


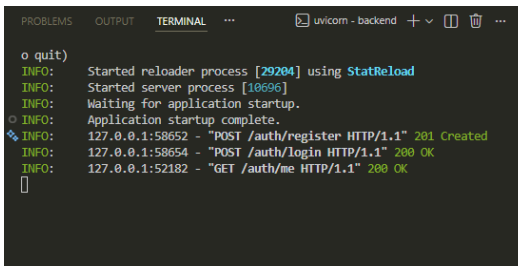


Informasi ini digunakan untuk uji coba endpoint selanjutnya

- access_token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2huZG9lIiwidXNlcl9pZCI6IjYxYTA4OTczLTU1MzgtNDUwMy1hMjkzLTQ1NTI1YTU5OTg5YSIsImV4cCI6MTc2NDU3NTMyNn0.d1IHMgP4YwTovqDQJCfkLpNY5TiJdnwF6gq7VGrIDtA”

Nama Endpoint	/auth/me	**Method**	GET
URL	http://127.0.0.1:8000/me		
Headers			
Authorization	**Bearer** eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2huZG9lIiwidXNlcl9pZCI6IjYxYTA4OTczLTU1MzgtNDUwMy1hMjkzLTQ1NTI1YTU5OTg5YSIsImV4cCI6MTc2NDU3NTMyNn0.d1IHMgP4YwTovqDQJCfkLpNY5TiJdnwF6gq7VGrIDtA		
Response di Postman		**Response di Server**	



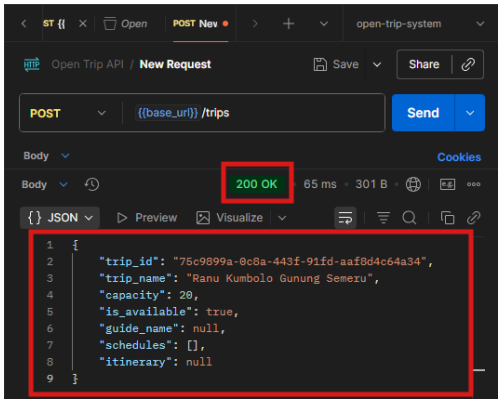
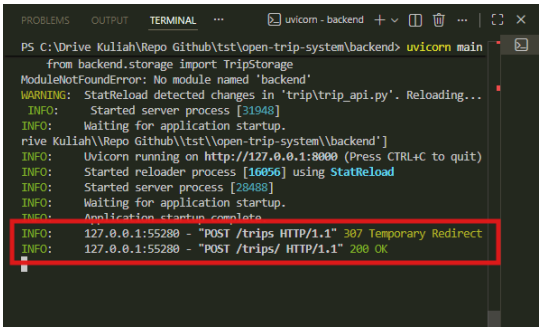


2.4.2. Uji Coba Endpoint Trip

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

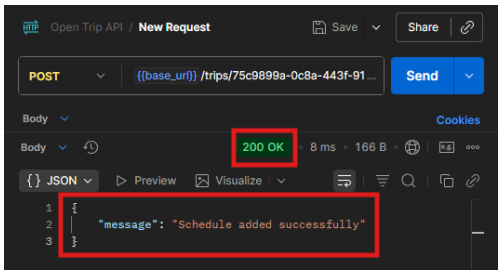
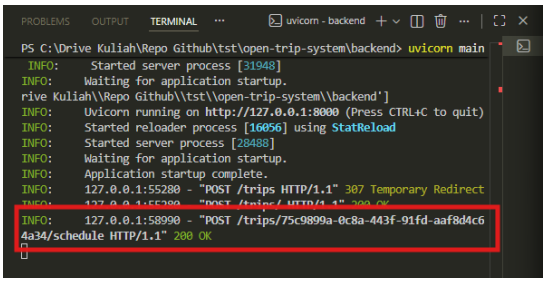
Nama Endpoint	`/trips`	**Method**	POST
URL	`http://127.0.0.1:8000/trips`		
Body			

<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-443f-91fd-aaf8d4c64a34/schedule`		
Body			

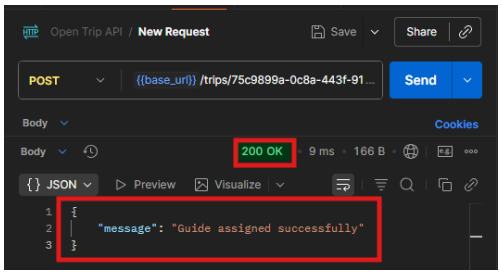
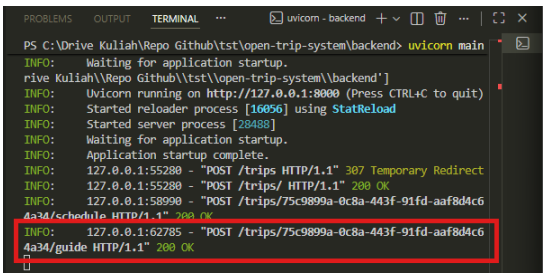
```
{
  "start_date": "2025-12-12",
  "end_date": "2025-12-14",
  "location": "Kabupaten Lumajang, Jawa Timur, Indonesia"
}
```

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

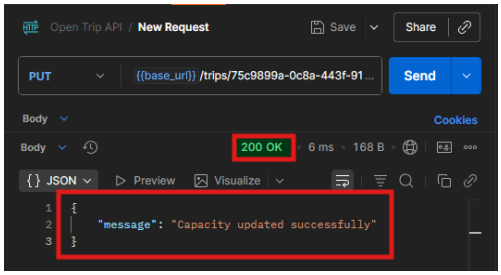
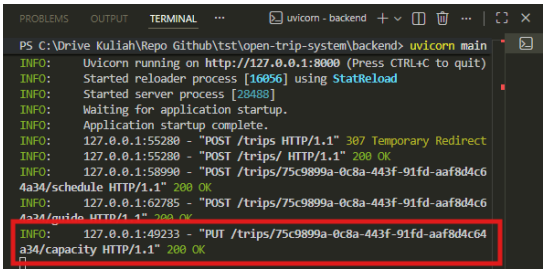
```
{
  "guide_name": "Mike Wheeler",
  "contact": "08123456789",
  "language": "English"
}
```

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

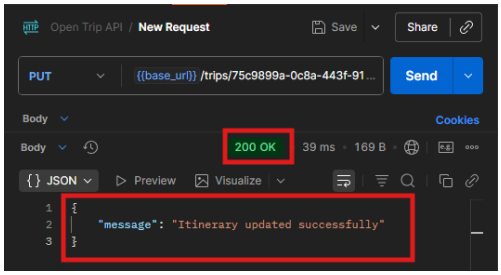
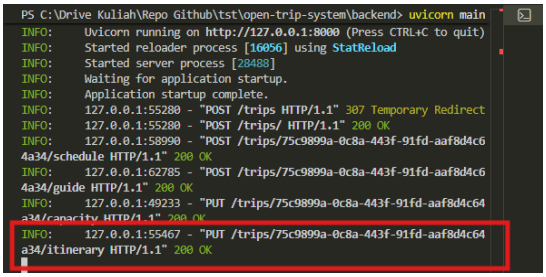
```
{
  "new_capacity": 15
}
```

Response di Postman	Response di Server
	

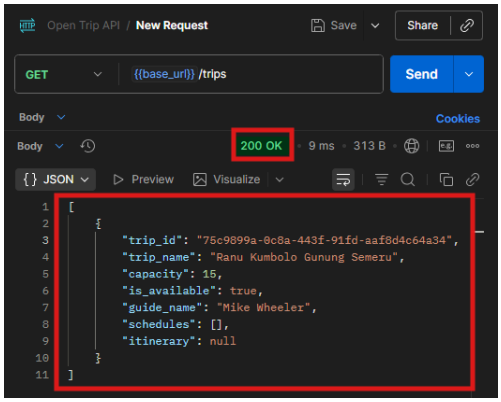
Nama Endpoint	/trips/{trip_id}/itinerary	Method	PUT
URL	http://127.0.0.1:8000/trips/*75c9899a-0c8a-443f-91fd-aaf8d4c64a34*/itinerary		

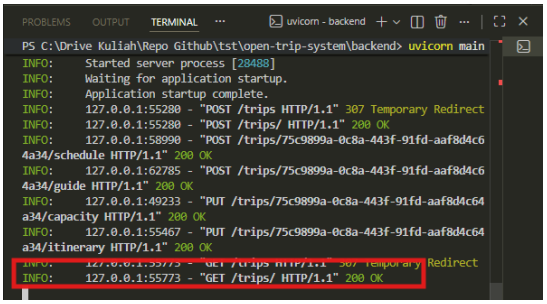
Body

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

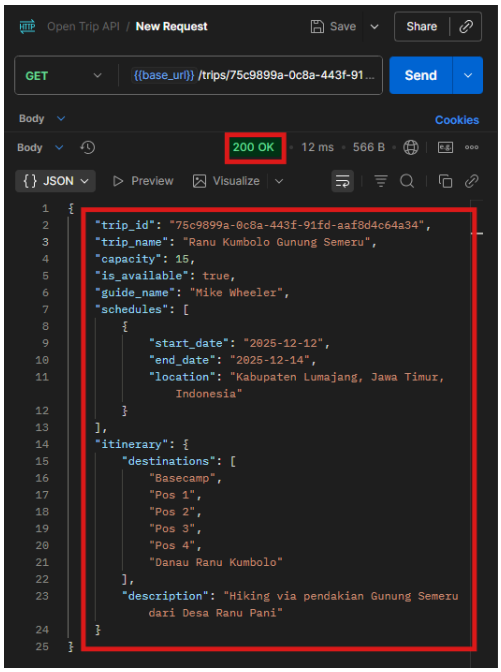
Response di Postman	Response di Server
	

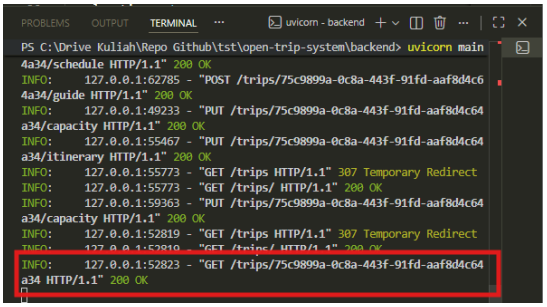
Nama Endpoint	/trips	Method	GET
URL	http://127.0.0.1:8000/trips		
Response di Postman		Response di Server	





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





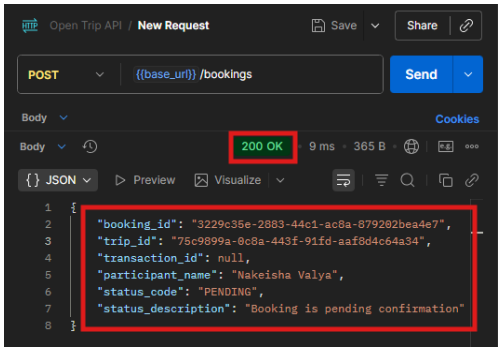
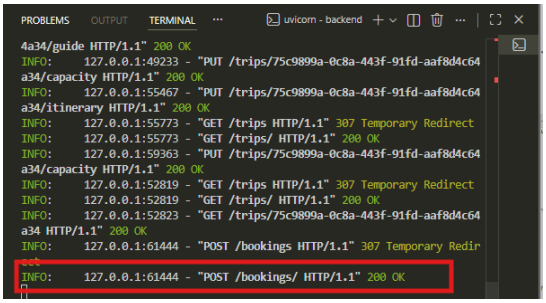
2.4.3. Uji Coba Endpoint Booking

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

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

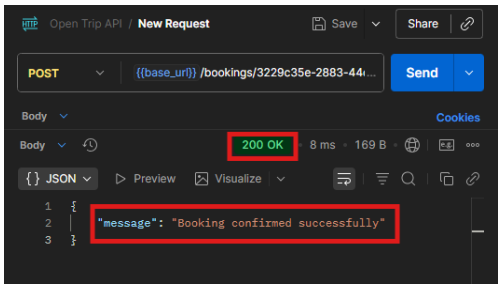
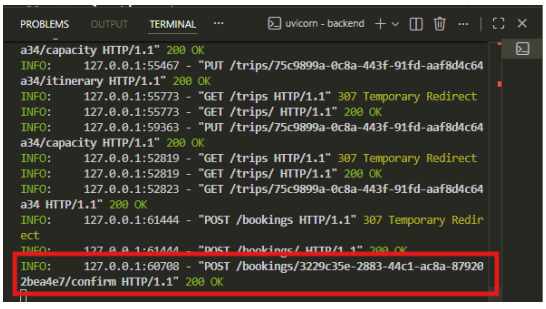
Body

```
{
  "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		

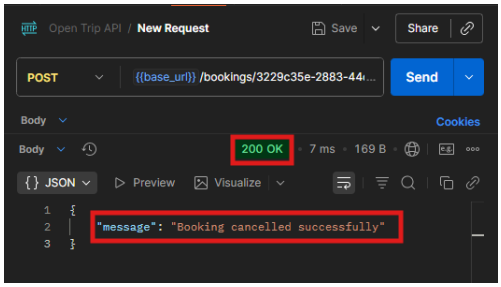
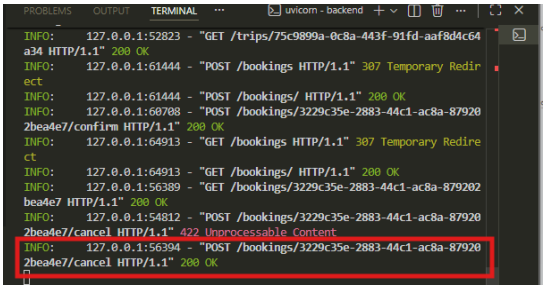
Response di Postman	Response di Server
	

Nama Endpoint	/bookings/{booking_id}/cancel	Method	POST
URL	http://127.0.0.1:8000/bookings/*3229c35e-2883*		

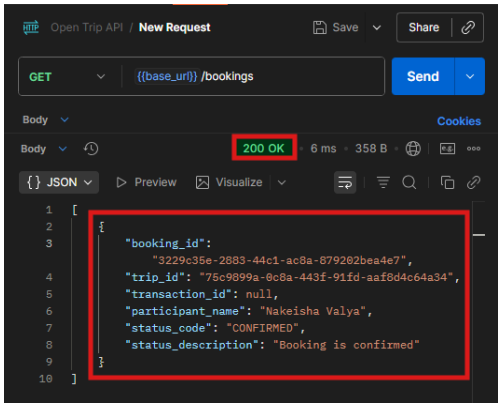
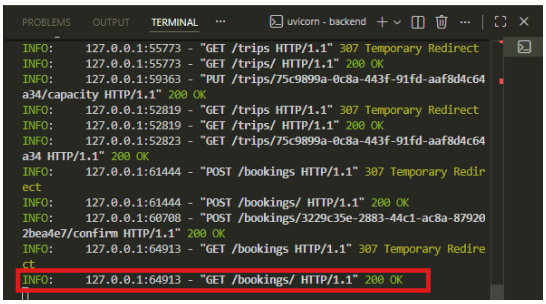
	-44c1-ac8a-879202bea4e7/cancel

Body

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

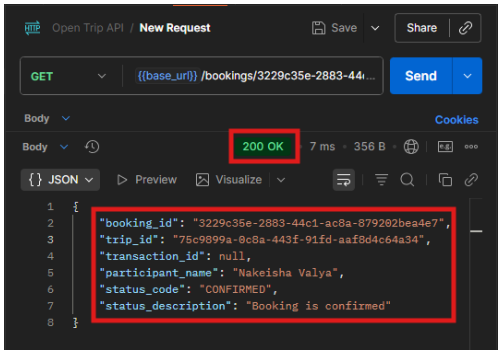
Response di Postman	Response di Server
	

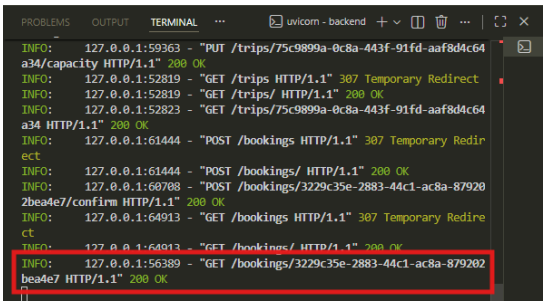
Nama Endpoint	/bookings	Method	GET
URL	http://127.0.0.1:8000/bookings		

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





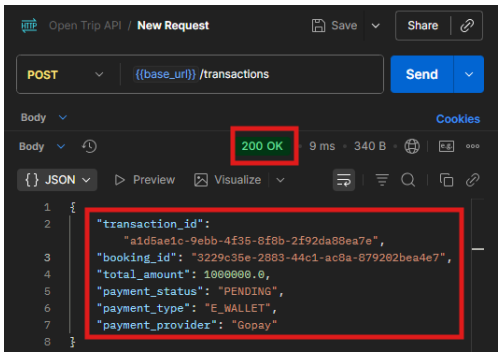
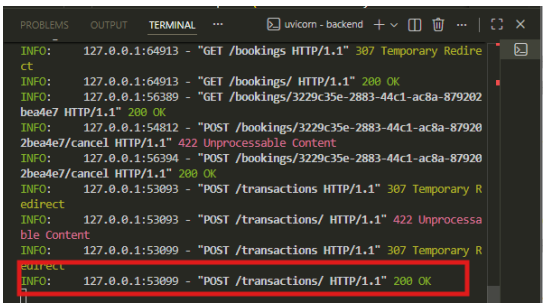
2.4.4. Uji Coba Endpoint Transactions

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

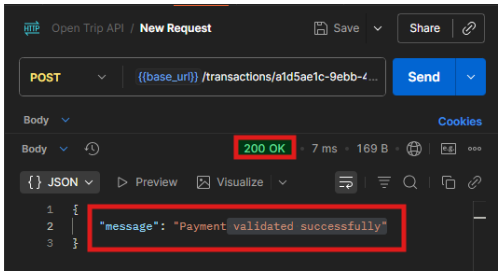
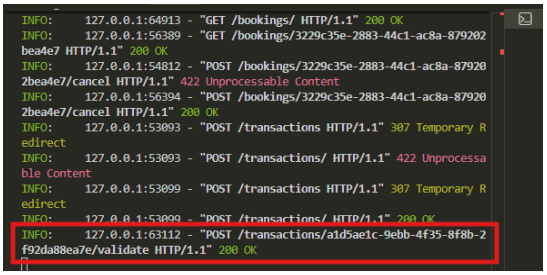
```
{
  "booking_id": "3229c35e-2883-44c1-ac8a-879202bea4e7",
  "amount": 1000000,
  "payment_type": "E_WALLET",
  "provider": "Gopay"
}
```

Response di Postman	**Response di Server**
	

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

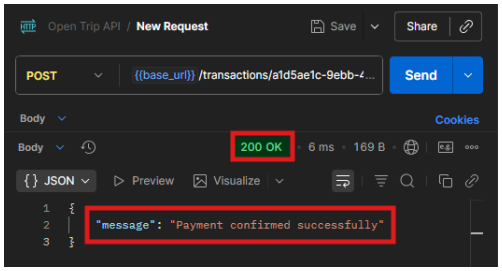
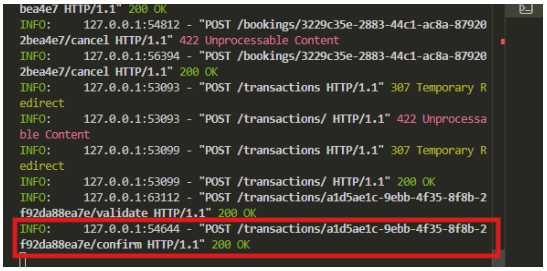
Nama Endpoint	`/transactions/{transaction_id}/validate`	**Method**	POST
URL	`http://127.0.0.1:8000/transactions/a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e/validate`		

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		

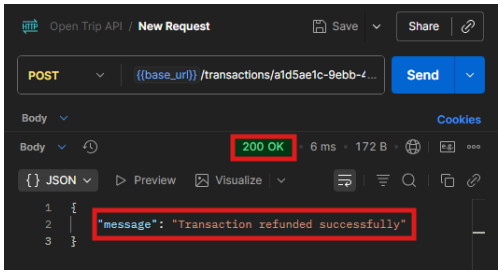
Body
–

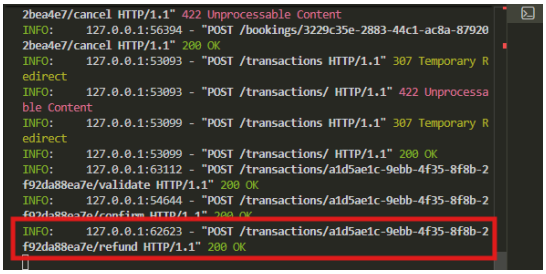
Response di Postman	Response di Server
	

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

Body
–

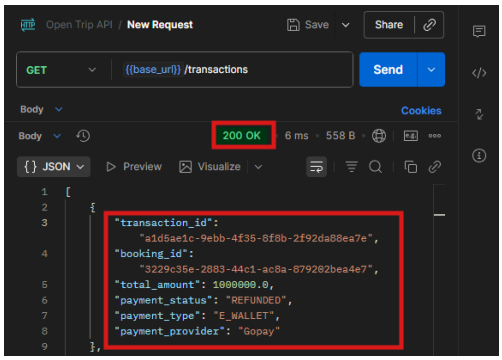
Response di Postman	Response di Server

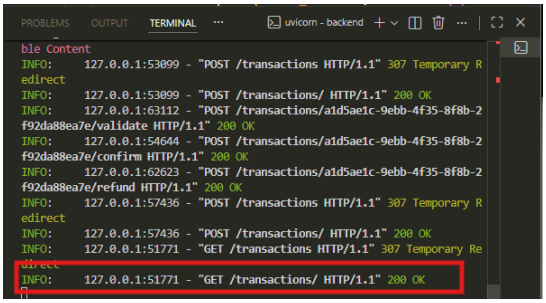




Nama Endpoint	/transactions	Method	GET
URL	http://127.0.0.1:8000/transactions		

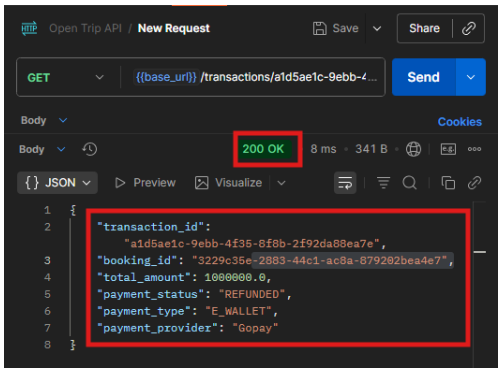
Response di Postman	Response di Server

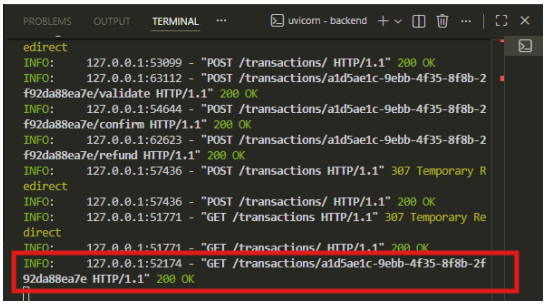




Nama Endpoint	/transactions/{transaction_id}	Method	GET
URL	http://127.0.0.1:8000/transactions/*a1d5ae1c-9ebb-4f35-8f8b-2f92da88ea7e*		

Response di Postman	Response di Server





2.5. Implementasi Test Drive Development dan Unit Testing

Penerapan Test-Driven Development (TDD) dan penyusunan unit testing dilakukan untuk memastikan seluruh fungsionalitas service terverifikasi sejak awal proses pengembangan. Setiap fitur dibangun melalui siklus TDD yang ketat sehingga kode yang ditulis benar-benar sesuai kebutuhan dan bebas dari perilaku tak terduga. Seluruh pengujian disusun berdasarkan domain service dan mencakup berbagai skenario validasi, error branch, serta edge-case. Selain itu, proses testing diarahkan untuk mencapai minimal 95% coverage, sehingga hampir seluruh alur logika backend teruji secara menyeluruh. Detail implementasinya dijelaskan sebagai berikut.

2.1.1. Alur Test Drive Development

Siklus TDD diterapkan dalam pengembangan setiap fitur melalui tiga tahap utama berikut:

Red	Menulis test yang gagal karena implementasi belum dibuat.
Green	Mengimplementasikan kode minimal agar test tersebut lulus.
Refactor	Merapikan struktur kode maupun test tanpa mengubah fungsionalitas.

Siklus ini diulang secara konsisten pada setiap endpoint atau modul/*aggregate* baru sehingga setiap bagian sistem telah teruji sejak awal sebelum dikembangkan lebih lanjut.

2.1.2. Penyusunan Unit Test

Penyusunan unit test dilakukan secara terstruktur berdasarkan modul/ atau domain service, seperti `booking`, `transaction`, `trip`, `storage`, dan `auth`.

Beberapa poin penyusunannya meliputi:

- Menggunakan **pytest** dan **FastAPI TestClient** untuk menguji endpoint secara langsung.\
- Menuliskan test yang mencakup validasi input, error branch, edge-case, serta response normal.
- Mengorganisasi test dalam satu file per domain agar mudah dibaca dan bebas duplikasi.

Dengan struktur ini, setiap domain memiliki cakupan pengujian yang konsisten dan mudah dirawat.

2.1.3. Cakupan Pengujian

Pengujian diarahkan untuk mencakup seluruh fungsi inti sistem beserta jalur error-nya. Cakupan utamanya meliputi:

No	Modul	Nama File Test	Cakupan Pengujian
1	Autentikasi	test_auth.py	Login, verifikasi token, hash password, endpoint /login, /me, error handling
2	Booking	test_booking.py	Pembuatan booking, konfirmasi, pembatalan, validasi input, error branch, endpoint
3	Transaksi	test_transaction.py	Inisiasi pembayaran, validasi, konfirmasi, refund, validasi input, error branch

4	Trip	test_trip.py	Pembuatan trip, jadwal, itinerary, assign guide, update kapasitas, error branch
5	Storage	test_storage.py	Operasi CRUD storage in-memory, integrasi storage dengan domain
6	Main/Edge Case	test_main_edge.py	Pengujian main, dokumentasi, edge-case sistem utama

Secara keseluruhan, struktur cakupan pengujian ini memastikan bahwa setiap domain terverifikasi secara menyeluruh, mulai dari alur normal hingga skenario edge-case dan jalur error yang berpotensi muncul dalam operasional sistem.

2.1.4. Analisis Test Coverage

Berdasarkan struktur cakupan pengujian pada poin 2.1.3 yang telah memastikan setiap domain sistem terverifikasi secara menyeluruh untuk mencapai target minimal 95 persen coverage, pencapaian target ini dilakukan dengan menguji seluruh alur logika backend secara konsisten menggunakan pytest-cov serta menambahkan test tambahan jika terjadi penurunan coverage akibat perubahan kode. Untuk mengujinya menggunakan perintah dibawah ini

```
pytest --cov=backend --cov-report=term-missing
backend/tests
```

Pengujian juga difokuskan pada edge-case dan jalur error sehingga setiap skenario yang berpotensi memicu kegagalan sistem dapat teridentifikasi dan tervalidasi dengan baik.

```
============================================================ tests coverage ============================================================
_____________________________________ coverage: platform win32, python 3.13.5-final-0 _____________________________________

Name                                        Stmts   Miss  Cover   Missing
-------------------------------------------------------------------------
backend\auth.py                               135      6    96%   87, 98, 159, 166, 202, 225
backend\booking\aggregate_root.py              32      0   100%
backend\booking\booking_api.py                 67      7    90%   40-53, 68
backend\booking\entities.py                    26      0   100%
backend\booking\value_objects.py               29      0   100%
backend\main.py                                21      4    81%   32, 40, 43-44
backend\storage.py                             67      1    99%   49
backend\tests\test_auth.py                     62      0   100%
backend\tests\test_booking.py                 112      3    97%   87-89
backend\tests\test_main_edge.py                20      0   100%
backend\tests\test_storage.py                  49      0   100%
backend\tests\test_transaction.py             189      5    97%   99-103
backend\tests\test_trip.py                    159      0   100%
backend\transaction\aggregate_root.py          39      0   100%
backend\transaction\transaction_api.py         78     17    78%   32-54, 94-95, 107-108, 120-121
backend\transaction\value_objects.py           57      0   100%
backend\trip\aggregate_root.py                 54      0   100%
backend\trip\entities.py                       29      0   100%
backend\trip\trip_api.py                      109     18    83%   48-61, 69-85, 97-98, 122-123, 154-155
backend\trip\value_objects.py                  28      0   100%
-------------------------------------------------------------------------
TOTAL                                        1362     61    96%
========================================================== 78 passed, 3 warnings in 2.81s ==========================================================
```

Gambar 2.4 Pengujian Berhasil pada CLI

Hasil pengujian menunjukkan bahwa seluruh 78 test berhasil dijalankan tanpa error dengan total coverage mencapai **96%**, melampaui target minimal yang ditetapkan dan mencerminkan bahwa sebagian besar domain seperti booking, transaction, trip, dan storage telah terverifikasi secara menyeluruh. Meskipun demikian, beberapa file masih berada di bawah 90%coverage, yaitu booking_api.py sebesar 90%, transaction_api.py sebesar 78%, trip_api.py sebesar 83%, dan backend/main.py sebesar 81%, dengan sisa branch yang belum tercover umumnya berasal dari skenario khusus seperti error branch, validasi input ekstrem, atau response API

tertentu yang jarang dipicu. Secara keseluruhan, hasil ini menunjukkan bahwa sistem telah memiliki tingkat keandalan yang tinggi, dengan cakupan pengujian yang komprehensif dan hanya membutuhkan sedikit penambahan test jika ingin mencapai coverage optimal pada seluruh modul.

2.6. Continuous Integration (CI) dengan GitHub Workflows

Continuous Integration (CI) adalah praktik otomatisasi penggabungan kode untuk memastikan setiap perubahan tetap stabil, bebas error, dan sesuai standar kualitas. Pada open-trip-system, CI dijalankan melalui GitHub Workflows yang secara otomatis melakukan linting, testing, dan persiapan environment setiap kali kode diubah, sehingga proses validasi berjalan konsisten tanpa intervensi manual.

2.1.1. Desain dan Job Pipeline

File **ci.yml** mendefinisikan pipeline **Continuous Integration (CI)** untuk proyek Python menggunakan GitHub Actions. Pipeline ini dijalankan otomatis setiap ada **push** atau **pull request** ke branch `main`.

```yaml
name: CI
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
jobs:
  lint-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r backend/requirements.txt
          pip install ruff pytest pytest-cov
      - name: Lint with ruff
        run: |
          ruff check backend
      - name: Run tests with pytest
        run: |
          PYTHONPATH=. pytest --cov=backend --cov-report=term-missing
--cov-report=xml backend/tests
      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v4
```

```
        with:
          files: ./coverage.xml
        continue-on-error: true
```

Terdapat satu job utama bernama `lint-and-test` yang berjalan di environment `ubuntu-latest` dan meliputi beberapa langkah:

1. **Checkout code:** Mengambil kode terbaru dari repository.
2. **Setup Python:** Menyiapkan environment Python versi 3.11.
3. **Install dependencies:** Menginstall semua dependensi dari `requirements.txt` serta tools tambahan seperti `ruff`, `pytest`, dan `pytest-cov`.
4. **Linting:** Menjalankan linter `ruff` pada folder `backend` untuk memeriksa kesesuaian style dan error sintaks.
5. **Testing + Coverage:** Menjalankan `pytest` dengan coverage report (`--cov=backend --cov-report=term-missing --cov-report=xml`) pada seluruh test di `backend/tests`.
6. **Upload Coverage:** Mengupload hasil coverage ke Codecov jika opsi ini diaktifkan.

2.1.2. Integrasi Linting, Testing, dan Build

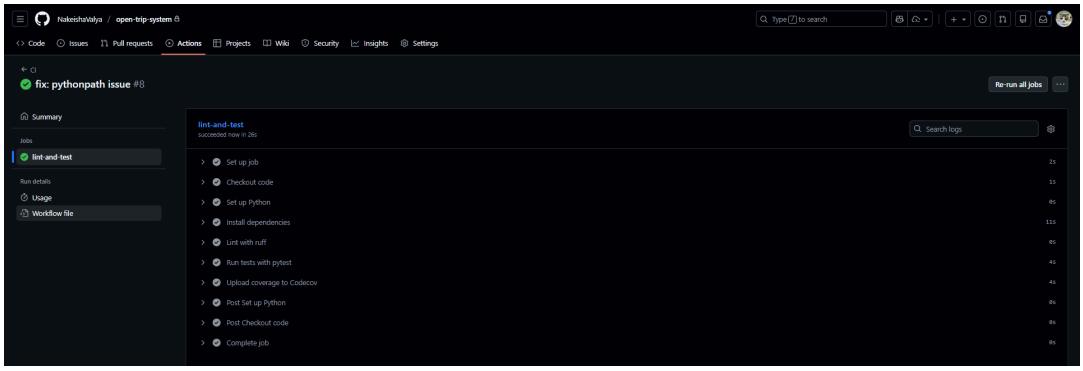
Pipeline mengintegrasikan linting, testing, dan persiapan environment sehingga seluruh proses dijalankan secara berurutan.

- **Linting:** `ruff` memastikan kode backend bebas dari error dan mengikuti standar style yang ditentukan. Pemilihan `ruff` untuk linting dilakukan karena tool ini cepat, ringan, dan mampu memastikan kode backend bebas dari error serta sesuai standar style
- **Testing:** `pytest` menjalankan semua unit test dan menghasilkan laporan coverage untuk mengevaluasi cakupan pengujian. Pemilihan `pytest` dipilih untuk testing karena fleksibel, mendukung pengujian unit secara menyeluruh, dan mampu menghasilkan laporan coverage yang detail untuk mengevaluasi cakupan pengujian
- **Build:** Meskipun tidak ada build khusus karena proyek ini berbasis Python, pipeline secara otomatis menyiapkan environment dan menginstall seluruh dependensi.

Dengan menggabungkan linting dan testing dalam satu pipeline, setiap push atau pull request langsung dapat diketahui status kelulusan kode, sehingga meminimalkan risiko error masuk ke branch utama.

2.1.3. Indikator Keberhasilan CI

Untuk mengecek keberhasilan CI, kita dapat melihat status pipeline setelah melakukan push ke GitHub pada menu **Actions** yang terdapat di bagian atas repository.



Gambar 2.5 CI Berhasil pada Github

2.7. Deployment

Deployment merupakan proses memindahkan dan menjalankan aplikasi dari lingkungan pengembangan ke lingkungan produksi, dan ini perlu dilakukan agar API dari sistem yang telah dirancang di milestone sebelumnya dapat diakses publik, bekerja secara stabil, serta dapat digunakan oleh pengguna dan layanan lain secara real-time.

Gambar 2.6 Logo Platform Railway

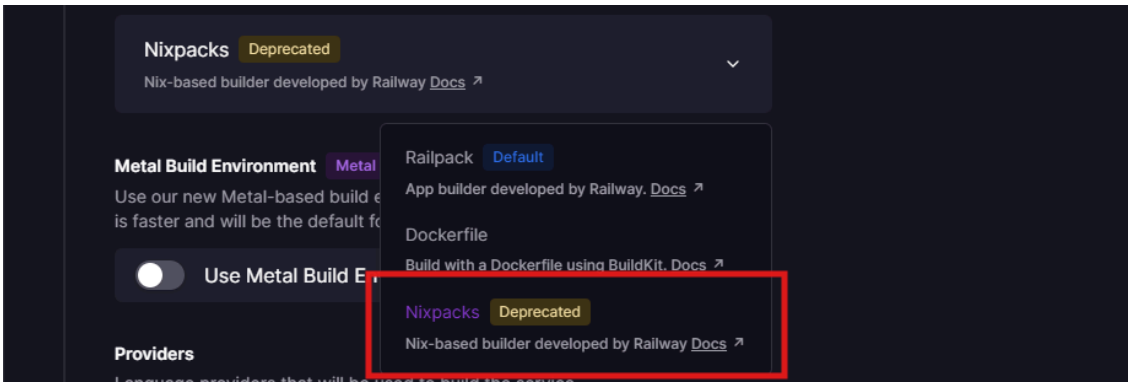
Proses deployment dilakukan menggunakan Railway, sebuah platform yang terintegrasi langsung dengan GitHub. Dengan integrasi ini, setiap perubahan kode pada repository akan otomatis diterapkan ke server, disertai beberapa pengaturan tambahan yang perlu dikonfigurasi. Berikut merupakan pengaturan konfigurasi yang dilakukan pada sistem:

1. Mengubah Deploy - Custom Start Command

```
python -m uvicorn backend.main:app --host 0.0.0.0 --port $PORT
```

Perubahan tersebut dilakukan agar aplikasi dijalankan dengan perintah Uvicorn yang benar dan menggunakan port dinamis dari Railway sehingga server dapat berjalan tanpa error.

2. Mengubah Build - Builder dan Custom Build Command



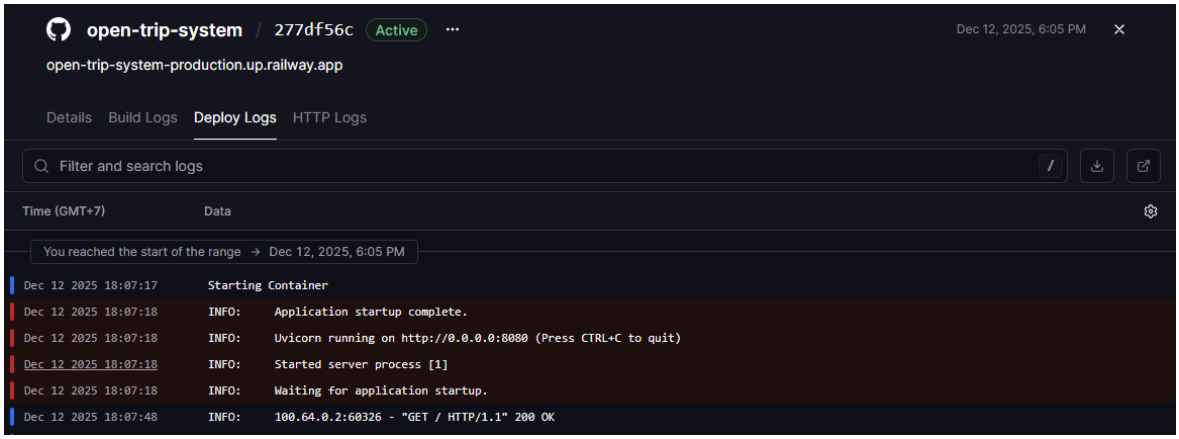
Gambar 2.7 CI Opsi Builder saat Build

Pengaturan builder diubah karena Nixpacks secara otomatis mendeteksi lingkungan Python dan memastikan semua dependency dapat dibangun sesuai struktur proyek.

```
pip install -r backend/requirements.txt
```

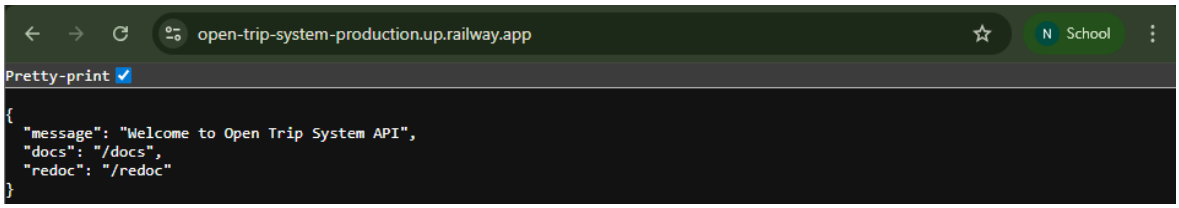
Custom build command ditambahkan untuk menjamin bahwa Railway meng-install semua library dari `requirements.txt` sehingga aplikasi dapat berjalan lengkap tanpa missing dependencies.

Setelah semua konfigurasi selesai, proses deployment akan berjalan secara otomatis dan Railway akan membangun serta menjalankan sistem berdasarkan pengaturan yang telah ditetapkan. Untuk memastikan server telah berjalan maka kita bisa langsung mengeceknya pada **Deploy Logs**



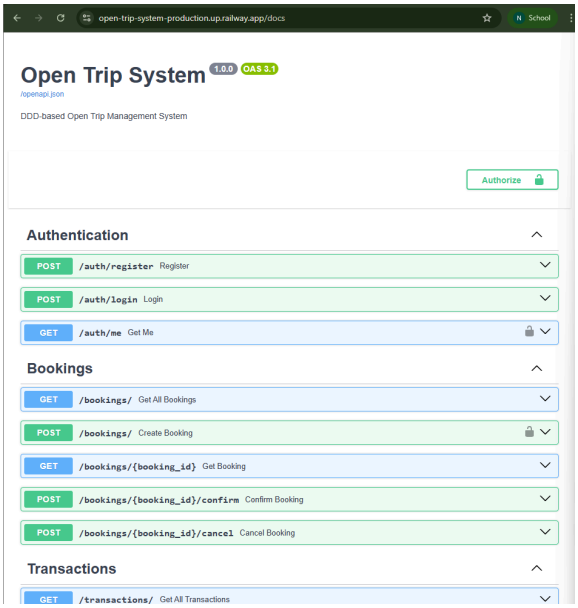
Gambar 2.8 Status Deploy Logs dan Server Berjalan

Apabila server telah berjalan, kita bisa langsung mengakses melalui URL yang otomatis dibuat Railway saat proses deployment selesai, URL yang digunakan pada sistem ini yaitu https://open-trip-system-production.up.railway.app/

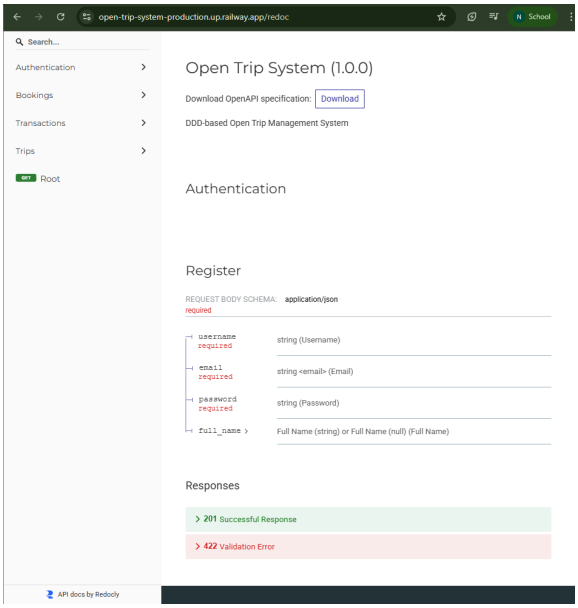


Gambar 2.9 Tampilan URL Hasil Deployment

Terdapat dua tampilan dokumentasi API yang tersedia, yaitu Swagger UI melalui `/docs` dan ReDoc melalui `/redoc`.



Gambar 2.10 Tampilan URL Melalui Swagger



Gambar 2.11 Tampilan URL Melalui ReDoc

BAB III

PENUTUP

3.1. Kesimpulan

Implementasi Domain dan Struktur Modular Sistem berhasil mengimplementasikan *Booking Context* dengan memusatkan logika bisnis pada tiga *aggregate* utama, yaitu *Trip Aggregate*, *Booking Aggregate*, dan *Transaction Aggregate*, yang didukung oleh sistem autentikasi berbasis JWT. Struktur folder proyek telah disesuaikan agar lebih modular dengan memisahkan *logic* domain dan memusatkan seluruh pengujian pada direktori `backend/tests/`.

Keandalan Sistem Melalui TDD dan Pengujian Penerapan *Test-Driven Development* (TDD) berjalan efektif dengan hasil pengujian otomatis menunjukkan bahwa 78 test berhasil dijalankan tanpa *error*. Sistem mencapai tingkat *coverage* keseluruhan sebesar 96%, melampaui target minimal 95%, yang menandakan bahwa sebagian besar logika bisnis, validasi input, hingga skenario *edge-case* telah terverifikasi dengan baik. Validasi tambahan menggunakan Postman juga mengonfirmasi bahwa seluruh *endpoint* berjalan sesuai alur bisnis yang diharapkan.

Otomasi dan Kualitas Kode (CI) Integrasi *Continuous Integration* (CI) melalui GitHub Workflows berhasil diterapkan. Pipeline `lint-and-test` secara otomatis menjalankan pemeriksaan gaya kode (*linting*) menggunakan `ruff` dan pengujian unit menggunakan `pytest` setiap kali terdapat perubahan kode (*push/pull request*), memastikan stabilitas sistem tetap terjaga sebelum kode digabungkan ke *branch* utama.

Ketersediaan Layanan (Deployment) Sistem sukses di-*deploy* ke lingkungan produksi menggunakan platform Railway dengan konfigurasi *custom build command* untuk menangani dependensi Python. API Open Trip System kini dapat diakses secara publik melalui URL yang disediakan, lengkap dengan dokumentasi interaktif berbasis Swagger UI dan ReDoc untuk memudahkan penggunaan oleh pihak ketiga.

Lampiran Link Deployment : open-trip-system-production.up.railway.app
Lampiran Link Repository Github: github.com/NakeishaValya/open-trip-system

DAFTAR PUSTAKA

Douglas, B., Douglas, B., & Douglas, B. (2022, February 2). How to build a CI/CD pipeline with GitHub Actions in four simple steps. The GitHub Blog. https://github.blog/enterprise-software/ci-cd/build-ci-cd-pipeline-github-actions-four-steps/

Indonesia, M. (n.d.). Activity Diagram: Pengertian, Tujuan, dan Cara Membuatnya. MyEduSolve Indonesia. https://myedusolve.com/id/blog/activity-diagram-pengertian-tujuan-dan-cara-membuatnya

SonarSource. (n.d.). Python Static Code Analysis & Quality Code | Sonar. Sonar. https://www.sonarsource.com/knowledge/languages/python/?utm_source=google&utm_medium=cpc&utm_campaign=SQ-APJ-2-All-Nonbrand-Language&utm_content=Python&utm_term=python%20tools&s_campaign=SQ-APJ-2-All-Nonbrand-Language&s_content=191036018791&s_category=Paid&s_source=Paid%20Search&s_origin=Google&cq_src=google_ads&cq_cmp=21465802938&cq_con=191036018791&cq_term=python%20tools&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gad_campaignid=21465802938&gbraid=0AAAAAC0fKmoTPtBJGHc-Hh1D5AZwz_R1_&gclid=CjwKCAiAl-_JBhBjEiwAn3rN7ShtaxobxekNPBW36ICFrjhsP4AJLlyxyKOQoIqJjGTi2hZ-k3_5bhoCb3gQAvD_BwE

Coverage.py — Coverage.py 7.13.0 documentation. (n.d.). Retrieved December 12, 2025, from https://coverage.readthedocs.io/en/7.13.0/