Tugas Finalisasi
II3160 Teknologi Sistem Terintegrasi

Dosen Pembimbing:
Daniel Wiyogo Dwiputro, S.T., M.T

Dibuat oleh:
Rasyid Rizky Susilo Nurdwiputro (18223114)

Program Studi Sistem dan Teknologi Informasi
STEI - ITB

Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat 40132

Daftar Isi

BAB I Pendahuluan

1.1 Deskripsi Dokumen

Pada tahap pengembangan sebelumnya, telah dirancang sebuah sistem "Sistem Reservasi & Manajemen Restoran Terintegrasi" menggunakan kerangka kerja FastAPI dengan pendekatan Domain-Driven Design (DDD). Pendekatan ini dipilih untuk memastikan bahwa kompleksitas logika bisnis, seperti siklus hidup reservasi, validasi waktu, dan alokasi meja, dapat dipetakan dengan jelas ke dalam kode program. Implementasi awal juga telah mengimplementasi basis data relasional PostgreSQL untuk menjamin persistensi dan integritas data.

Namun, dalam ekosistem pengembangan perangkat lunak modern tingkat enterprise, sekadar membangun fitur fungsional tidaklah cukup. Keandalan (reliability) dan kemudahan pemeliharaan (maintainability) sistem menjadi parameter utama keberhasilan. Tanpa mekanisme pengujian yang ketat, perubahan kode sekecil apa pun berpotensi menimbulkan bug yang merusak logika bisnis yang sudah ada (regression). Selain itu, proses deployment manual sering kali menimbulkan inkonsistensi antara lingkungan pengembangan lokal dan lingkungan produksi.

Untuk menjawab tantangan kualitas tersebut, tahap finalisasi pengembangan ini berfokus pada penerapan metodologi Test-Driven Development (TDD) dan otomasi infrastruktur. Penerapan TDD mewajibkan pembuatan skenario pengujian sebelum atau bersamaan dengan penulisan kode, dengan target cakupan kode (code coverage) minimal 95%. Hal ini mencakup pengujian unit untuk logika domain, edge cases, keamanan, hingga alur kerja bisnis utama.

Selanjutnya, untuk memastikan bahwa setiap perubahan kode memenuhi standar kualitas secara otomatis, diimplementasikan sistem Continuous Integration (CI) menggunakan GitHub Workflows. Sistem ini secara otomatis melakukan pemeriksaan kualitas kode (linting) dan menjalankan rangkaian pengujian setiap kali terjadi pembaruan pada repositori. Akhirnya, layanan dikemas menggunakan teknologi kontainerisasi Docker dan di-deploy ke layanan awan (seperti Railway) untuk membuktikan kesiapan sistem dalam melayani permintaan pengguna secara nyata.

Laporan akhir ini mendokumentasikan seluruh proses tersebut, mulai dari desain arsitektur, implementasi fitur, penjaminan mutu melalui pengujian intensif, hingga otomasi penyebaran sistem.

1.2 Tujuan Implementasi

Berdasarkan latar belakang yang telah dipaparkan, tujuan utama dari tahap finalisasi pengembangan sistem ini adalah:

1. **Menjamin Kualitas Kode melalui TDD (Test-Driven Development)**
   Mengimplementasikan rangkaian pengujian unit (unit testing) yang komprehensif dengan cakupan kode (code coverage) minimal 95%. Pengujian ini mencakup skenario sukses (happy paths), penanganan kesalahan (error handling/edge cases), validasi keamanan (security), serta alur kerja bisnis utama (business workflows) untuk meminimalkan risiko bug pada produksi.

2. **Membangun Otomasi Integrasi (Continuous Integration)**
   Mengonfigurasi GitHub Workflows untuk menciptakan pipa saluran (pipeline) CI yang otomatis. Hal ini bertujuan untuk memastikan setiap kode yang didorong (push) ke repositori telah melalui proses pemeriksaan standar penulisan kode (linting) dan lulus seluruh rangkaian pengujian sebelum dianggap valid.

3. **Implementasi Kontainerisasi dan Deployment**
   Mengemas aplikasi beserta dependensi dan konfigurasi basis datanya menggunakan teknologi Docker untuk menjamin konsistensi lingkungan. Selanjutnya, melakukan deployment layanan ke penyedia layanan awan (cloud provider) agar sistem dapat diakses dan digunakan secara langsung oleh pengguna akhir.

4. **Menyediakan Dokumentasi Teknis yang Lengkap**
   Menyusun dokumentasi penggunaan (README) yang informatif serta laporan akhir yang mencakup diagram aktivitas (Activity Diagram), spesifikasi API, dan bukti pengujian untuk memudahkan pengembang lain atau pemangku kepentingan dalam memahami dan menggunakan sistem.

1.3 Ruang Lingkup

Ruang lingkup pengembangan dan pembahasan dalam laporan ini difokuskan pada pembangunan layanan backend (API) untuk Reservation Management Context sebagai domain inti (Core Domain). Sistem tidak mencakup pengembangan antarmuka pengguna (frontend) berbasis web maupun seluler.

Secara spesifik, ruang lingkup sistem mencakup aspek-aspek berikut:

1. **Fungsionalitas Bisnis dan Logika Domain**
   - **Siklus Hidup Reservasi Penuh**: Implementasi alur bisnis lengkap mulai dari pembuatan reservasi (Create), konfirmasi (Confirm), kedatangan tamu (Check-in), penyelesaian layanan (Complete), hingga pembatalan (Cancel).
   - **Manajemen Meja**: Pengalokasian meja (Table Assignment) dan validasi kapasitas.

- **Dashboard Analitik**: Penyediaan endpoint statistik untuk memantau performa operasional (total reservasi, status terkini, dan estimasi pendapatan).
- **Autentikasi & Keamanan**: Penerapan mekanisme login berbasis JWT (JSON Web Token) untuk membatasi akses ke endpoint sensitif (Protected Routes) dan hashing kata sandi menggunakan Bcrypt.

2. **Arsitektur dan Teknologi**
    - **Framework**: Penggunaan FastAPI dengan arsitektur berlapis (Layered Architecture) yang memisahkan Domain, Service, Infrastructure, dan Interface sesuai prinsip DDD.
    - **Persistensi Data**: Migrasi dari penyimpanan memori sementara ke basis data relasional PostgreSQL untuk menjamin durabilitas data.
    - **Object-Relational Mapping (ORM)**: Penggunaan SQLAlchemy dengan strategi flattening untuk memetakan objek domain (Value Objects) ke dalam skema tabel relasional secara efisien.
    - **Data Enrichment**: Penyajian respons API dalam format JSON terstruktur (Nested JSON) yang kaya informasi untuk kemudahan konsumsi oleh klien.

3. **Penjaminan Mutu dan Otomasi (Quality Assurance & DevOps)**
    - **Test-Driven Development (TDD)**: Penerapan metodologi TDD secara ketat dengan cakupan pengujian (test coverage) mencapai minimal 95%, meliputi Unit Testing untuk happy paths, edge cases, keamanan, dan alur bisnis.
    - **Continuous Integration (CI)**: Implementasi GitHub Workflows untuk melakukan pemeriksaan otomatis (Linting dan Testing) pada setiap commit atau pull request.
    - **Kontainerisasi**: Pengemasan aplikasi dan lingkungan basis data menggunakan Docker dan Docker Compose untuk memastikan portabilitas sistem.
    - **Deployment**: Penyebaran layanan ke infrastruktur awan (Cloud Provider) agar API dapat diakses secara publik.

Sistem ini tidak mencakup integrasi langsung dengan gerbang pembayaran (Payment Gateway) nyata atau layanan SMS/Email eksternal. Fitur-fitur tersebut diimplementasikan dalam bentuk simulasi logika atau stub untuk menjaga fokus pada integritas model domain reservasi.

1.4 Definisi dan Istilah

Untuk menghindari ambiguitas dalam pemahaman dokumen ini, berikut adalah definisi dari istilah-istilah kunci yang digunakan dalam proses pengembangan dan pengujian sistem:

Istilah	Definisi
Aggregate	Struktur domain yang terdiri dari satu atau lebih Entity dan Value Object yang diperlakukan sebagai satu kesatuan logis. Aggregate memiliki satu Aggregate Root yang bertanggung jawab menjaga konsistensi data dan mengatur aturan bisnis internal.
Entity	Objek dalam domain yang memiliki identitas unik (ID) dan dapat mengalami perubahan keadaan sepanjang siklus hidupnya. Dalam konteks ini, Reservation, TableAssignment, dan Waitlist adalah contoh entity yang relevan.
Value Object	Objek tanpa identitas khusus yang merepresentasikan nilai atau karakteristik tertentu. Value Object bersifat immutabel dan digunakan untuk memperkuat makna atribut, seperti ReservationTime, ReservationPolicy, dan ContactInfo.
Aggregate Root	Entitas utama dalam sebuah aggregate yang menjadi titik akses bagi entitas lainnya. Dalam konteks ini, Reservation bertindak sebagai aggregate root yang mengoordinasikan struktur internal dan aturan bisnis reservasi.
Domain Event	Objek yang merepresentasikan kejadian penting dalam domain dan dipublikasikan setelah aggregate melakukan perubahan signifikan. Contohnya adalah event seperti ReservationCreated, ReservationConfirmed, dan PaymentCompleted, yang dapat digunakan untuk berkomunikasi dengan bounded context lain seperti Payment atau Notification.
Repository	Abstraksi yang menyediakan mekanisme penyimpanan dan pengambilan aggregate dari sumber data. Pada implementasi ini, repository menggunakan struktur penyimpanan database untuk mempermudah eksekusi dan pengujian.
API (Application Programming Interface)	Antarmuka yang memungkinkan interaksi pengguna atau sistem lain dengan layanan yang dibangun. API FastAPI digunakan untuk menerima request dari klien, memanggil service layer, dan mengembalikan response sesuai kebutuhan.
FastAPI	Framework modern berbasis Python yang digunakan untuk membangun API dengan performa tinggi. FastAPI mendukung validasi otomatis menggunakan Pydantic dan menghasilkan dokumentasi otomatis melalui Swagger UI.
Swagger UI	Dokumentasi antarmuka interaktif untuk API yang dihasilkan otomatis oleh FastAPI, dapat diakses melalui endpoint /docs.
Bounded Context	Batasan yang mendefinisikan ruang lingkup model yang konsisten dalam DDD. Setiap bounded context memiliki model sendiri, istilah sendiri, dan aturan bisnis sendiri. Dalam proyek ini, fokus implementasi adalah pada Reservation Management Context sebagai core context.
JWT (JSON Web	Standar terbuka (RFC 7519) untuk mentransmisikan informasi secara

Token)	aman antara pihak sebagai objek JSON. Dalam sistem ini, JWT digunakan sebagai Access Token untuk autentikasi stateless.
Docker	Platform perangkat lunak yang memungkinkan pembuatan, pengujian, dan penerapan aplikasi dengan cepat menggunakan konsep kontainerisasi.
Container	Unit standar perangkat lunak yang mengemas kode dan semua dependensinya agar aplikasi berjalan cepat dan andal di lingkungan komputasi apa pun.
Dockerfile	Dokumen teks berisi perintah-perintah yang digunakan Docker untuk membangun image kontainer aplikasi secara otomatis.
Bcrypt	Algoritma password hashing yang dirancang untuk keamanan tinggi, digunakan dalam sistem ini untuk menyimpan kata sandi pengguna secara terenkripsi.
Dependency Injection	Pola desain di mana objek menerima dependensi yang diperlukannya dari sumber eksternal. Di FastAPI (Depends), ini digunakan untuk menyuntikkan logika verifikasi token ke dalam setiap endpoint.
TDD (Test-Driven Development)	Metodologi pengembangan perangkat lunak di mana skenario pengujian (test cases) dibuat sebelum kode fungsional ditulis. Siklusnya dikenal dengan istilah Red-Green-Refactor.
Code Coverage	Ukuran metrik yang menyatakan persentase baris kode program yang telah dieksekusi dan divalidasi oleh rangkaian pengujian unit (unit tests). Target dalam proyek ini adalah minimal 95%.
CI (Continuous Integration)	Praktik pengembangan di mana perubahan kode secara otomatis digabungkan, diperiksa kualitasnya (linting), dan diuji oleh sistem otomatis setiap kali ada pembaruan ke repositori pusat.
CD (Continuous Deployment)	Praktik otomatisasi peluncuran aplikasi yang telah lolos tahap pengujian ke lingkungan produksi (production environment) agar dapat segera digunakan oleh pengguna.
GitHub Workflows	Fitur otomatisasi dari GitHub yang digunakan untuk mendefinisikan alur kerja CI/CD, seperti menjalankan tes otomatis dan deployment saat terjadi push atau pull request.
Linting	Proses analisis statis pada kode sumber untuk menemukan kesalahan pemrograman, bug, kesalahan gaya penulisan (style errors), dan konstruksi kode yang mencurigakan.
ORM (Object-Relational Mapping)	Teknik pemrograman untuk mengonversi data antara sistem tipe data berorientasi objek (Python) dan sistem basis data relasional (PostgreSQL).

BAB II Arsitektur dan Desain Sistem

2.1 Arsitektur Aplikasi

Sistem ini dirancang menggunakan pendekatan Layered Architecture yang diadopsi dari prinsip Domain-Driven Design (DDD). Tujuan utama dari arsitektur ini adalah memisahkan logika bisnis inti (Domain) dari detail teknis implementasi seperti kerangka kerja web, basis data, atau antarmuka pengguna. Pemisahan ini memastikan bahwa sistem mudah diuji (testable), mudah dipelihara (maintainable), dan fleksibel terhadap perubahan teknologi.

Secara garis besar, aplikasi terbagi menjadi empat lapisan logis utama yang berjalan di dalam lingkungan terisolasi (kontainer). Berikut adalah rincian arsitektur aplikasi:

1. **Interface Layer (Lapisan Antarmuka)**
   Lapisan terluar yang bertanggung jawab menangani interaksi dengan klien (aplikasi frontend atau layanan lain) melalui protokol HTTP.
   - **API Routes (src/api/)**: Mendefinisikan endpoint RESTful, menerima permintaan (request), dan mengembalikan respons standar. Lapisan ini tidak mengandung logika bisnis yang kompleks, melainkan mendelegasikannya ke lapisan domain.
   - **Data Transfer Objects (DTO) (src/schemas/)**: Menggunakan pustaka Pydantic untuk memvalidasi format data yang masuk (input sanitization) dan memformat data yang keluar (output serialization), memastikan data yang mengalir ke sistem selalu valid.

2. **Security Layer (Lapisan Keamanan)**
   Lapisan khusus yang menangani aspek autentikasi dan otorisasi secara terpusat (Cross-Cutting Concern).
   - **Authentication Service**: Menangani verifikasi kredensial pengguna dan penerbitan JWT (JSON Web Token).
   - **Dependency Injection**: Menyediakan mekanisme untuk menyuntikkan logika verifikasi token ke dalam setiap endpoint yang diproteksi (Protected Routes), memastikan hanya pengguna sah yang dapat mengakses fitur kritis.

3. **Domain Layer (Lapisan Domain)**
   Inti dari perangkat lunak yang merepresentasikan logika bisnis dan aturan operasional restoran. Lapisan ini bersifat murni (pure Python) dan tidak memiliki ketergantungan pada kerangka kerja eksternal.

- **Aggregate Root (src/domain/models.py)**: Entitas utama (Reservation) yang menjaga konsistensi data transaksi dan menegakkan invarian bisnis (contoh: tidak bisa check-in jika belum confirmed).
- **Value Objects (src/domain/value_objects.py)**: Objek nilai immutable yang mendeskripsikan atribut domain (seperti ReservationStatus, ReservationTime) dan divalidasi secara intrinsik.

4. **Infrastructure Layer (Lapisan Infrastruktur)**
   Lapisan ini menyediakan implementasi teknis untuk persistensi data dan konfigurasi lingkungan.
   - **ORM Models (src/infrastructure/orm_models.py)**: Mendefinisikan pemetaan (mapping) antara objek domain Python dengan tabel basis data relasional. Strategi flattening digunakan di sini untuk menyimpan atribut Value Object ke dalam kolom tabel fisik secara efisien.
   - **Database Configuration**: Mengelola koneksi session ke server PostgreSQL menggunakan SQLAlchemy.

5. **Deployment Architecture (Kontainerisasi)**
   Seluruh arsitektur aplikasi di atas dikemas menggunakan Docker.
   - **Application Container**: Menjalankan layanan FastAPI menggunakan server ASGI (Uvicorn).
   - **Database Container**: Menjalankan image resmi PostgreSQL. Kedua kontainer tersebut diorkestrasi menggunakan Docker Compose, menciptakan jaringan virtual privat yang memungkinkan komunikasi antar-layanan yang aman dan konsisten, baik di lingkungan pengembangan lokal (Local Development) maupun di infrastruktur cloud (Production).

2.2 Desain Database & Skema ORM

Sistem ini menggunakan basis data relasional PostgreSQL untuk penyimpanan data persisten. Interaksi antara kode aplikasi (Python) dan basis data tidak dilakukan melalui kueri SQL mentah (raw SQL), melainkan dikelola oleh SQLAlchemy ORM. Pendekatan ini memungkinkan manipulasi data dilakukan menggunakan paradigma berorientasi objek sambil tetap menjaga efisiensi kueri relasional.

2.2.1 Strategi Pemetaan (Mapping Strategy)

Untuk menjaga performa dan kesederhanaan skema, sistem menerapkan strategi Flattening (Peleburan). Dalam desain domain, konsep seperti ContactInfo, ReservationTime, dan PaymentDetail

dipisahkan sebagai Value Objects. Namun, pada lapisan infrastruktur basis data, atribut-atribut dari objek tersebut dilebur menjadi kolom-kolom datar di dalam satu tabel utama yaitu reservations. Strategi ini dipilih untuk:

- Menghindari join tabel yang berlebihan untuk data yang selalu diakses bersamaan.
- Mempercepat operasi baca/tulis untuk transaksi reservasi tunggal.
- Memastikan seluruh data reservasi disimpan atau dibatalkan dalam satu satuan transaksi atomik.

2.2.1 Strategi Pemetaan (Mapping Strategy)

Berikut adalah detail desain tabel reservations yang telah diimplementasikan dalam sistem:

Nama Tabel: reservations

Primary Key: reservation_id (UUID)

Nama Kolom	Tipe Data	Constraints	Keterangan & Mapping Domain
reservation_id	UUID	PK, Not Null	Identitas unik reservasi.
customer_id	UUID	Not Null	Referensi ID pelanggan (dari sistem eksternal/token).
status	VARCHAR	Not Null	Menyimpan status alur bisnis (PENDING, CONFIRMED, CHECKED_IN, COMPLETED, CANCELLED).
start_time	TIMESTAMP	Not Null	Waktu mulai reservasi. (Mapping dari VO ReservationTime).
duration_minutes	INTEGER	Not Null	Durasi reservasi dalam menit. (Mapping dari VO ReservationTime).
contact_name	VARCHAR	Not Null	Nama pemesan. (Mapping dari VO ContactInfo).
contact_phone	VARCHAR	Not Null	Nomor telepon pemesan. (Mapping dari VO ContactInfo).
contact_email	VARCHAR	Not Null	Email pemesan. (Mapping dari VO ContactInfo).
table_id	UUID	Nullable	ID meja fisik. Diisi saat proses Table Assignment.
table_area	VARCHAR	Nullable	Area meja (misal: "Indoor"). Diisi saat Table Assignment.
payment_status	VARCHAR	Default: 'UNPAID'	Status pembayaran deposit. (Mapping dari VO PaymentDetail).
payment_amount	INTEGER	Default: 0	Jumlah deposit. (Mapping dari VO

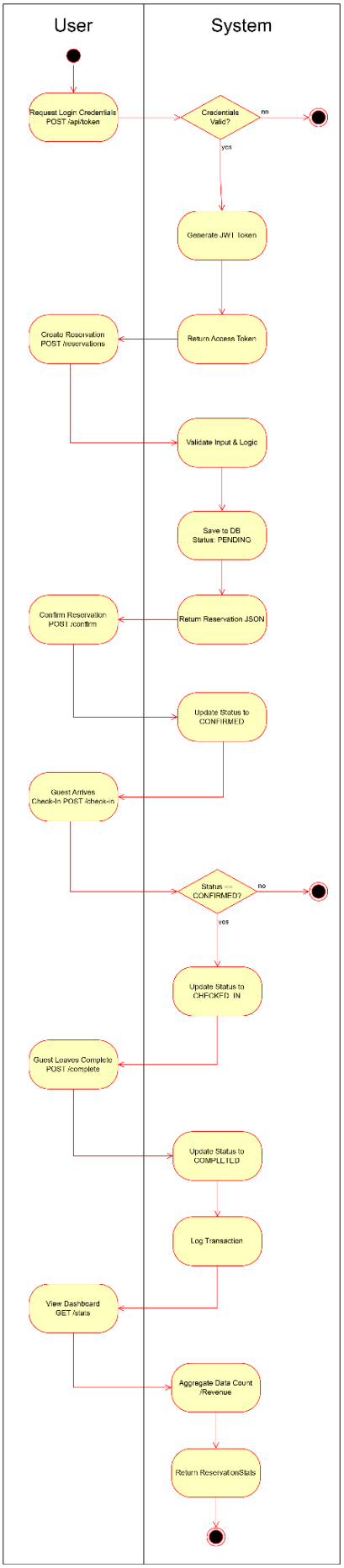
			PaymentDetail).

2.3 Activity Diagram Sistem

Diagram aktivitas berikut menggambarkan alur kerja sistem secara menyeluruh (End-to-End Service Flow) dari sisi pengguna (Klien API) dan respons sistem (Backend). Alur ini mencakup proses autentikasi, siklus hidup reservasi, hingga pemantauan statistik operasional.

Diagram ini dibagi menjadi dua swimlane utama:

- **User (Client)**: Pihak yang mengirimkan permintaan HTTP (misalnya Admin atau Sistem Frontend).
- **System (API & Database)**: Layanan backend yang memproses logika bisnis dan menyimpan data.

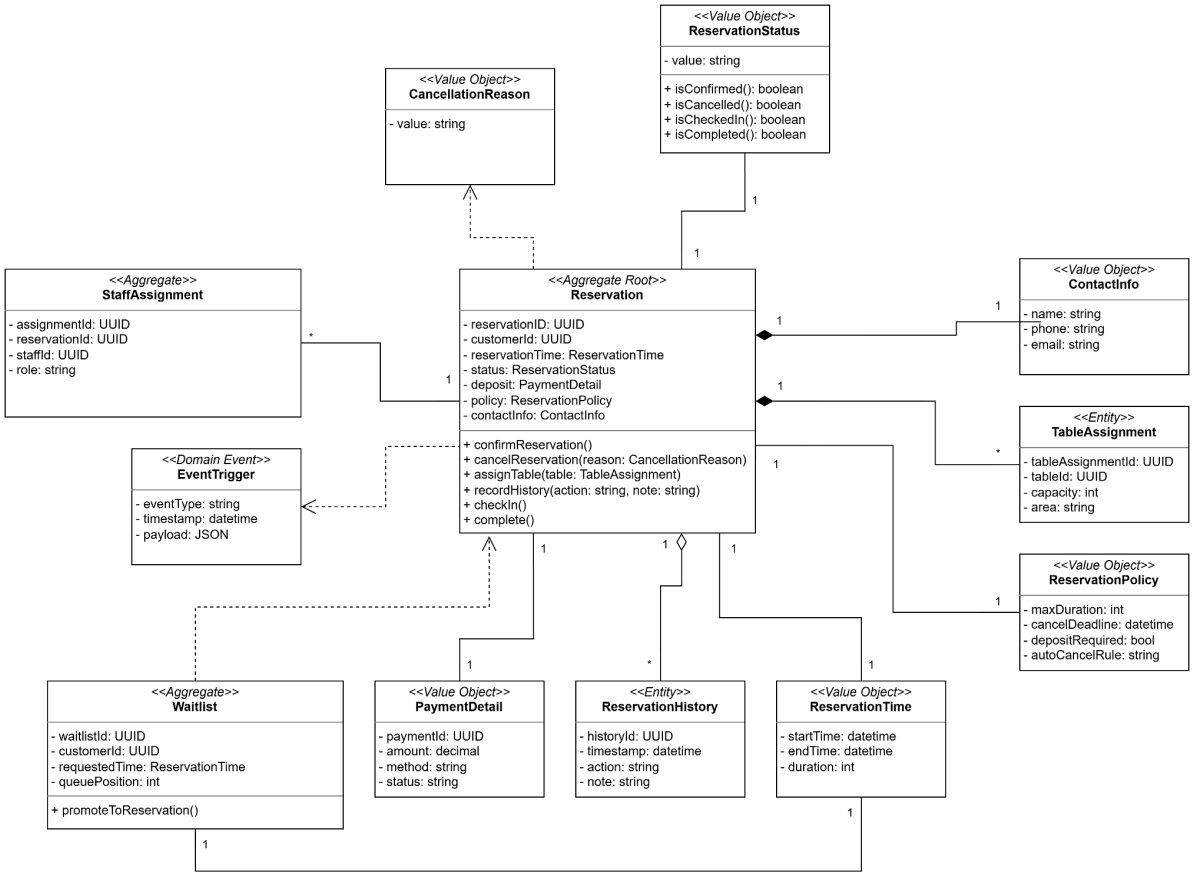

User
System
Request Login Credentials
POST /api/token
Credentials Valid?
no
yes
Generate JWT Token
Return Access Token
Create Reservation
POST /reservations
Validate Input & Logic
Save to DB
Status: PENDING
Return Reservation JSON
Confirm Reservation
POST /confirm
Update Status to
CONFIRMED
Guest Arrives
Check-In POST /check-in
Status == CONFIRMED?
no
yes
Update Status to
CHECKED_IN
Guest Leaves Complete
POST /complete
Update Status to
COMPLETED
Log Transaction
View Dashboard
GET /stats
Aggregate Data Count
/Revenue
Return ReservationStats

Penjelasan Alur Aktivitas secara garis besar:

1. **Autentikasi (Authentication Phase)**: Proses dimulai dengan pengguna mengirimkan kredensial (username dan password). Sistem memvalidasi data tersebut; jika valid, sistem menerbitkan Access Token (JWT). Token ini wajib digunakan untuk seluruh aktivitas selanjutnya.
2. **Pembuatan Reservasi (Reservation Creation)**: Pengguna mengirimkan data reservasi baru (nama, kontak, waktu). Sistem memvalidasi ketersediaan dan format data. Jika sukses, reservasi tersimpan dengan status awal PENDING.
3. **Konfirmasi & Alokasi (Confirmation Phase)**: Setelah reservasi dibuat, staf (pengguna) melakukan konfirmasi (/confirm) yang mengubah status menjadi CONFIRMED. Opsional, pengguna dapat mengalokasikan nomor meja spesifik (/assign-table).
4. **Operasional Restoran (Operational Phase)**:
    - **Check-In**: Saat tamu tiba, pengguna memanggil endpoint /check-in. Sistem memvalidasi status reservasi dan memperbaruinya menjadi CHECKED_IN.
    - **Completion**: Setelah layanan selesai, pengguna memanggil endpoint /complete. Sistem menandai reservasi sebagai COMPLETED dan membebaskan meja.
5. **Pemantauan (Analytics Phase)**: Di akhir alur atau sewaktu-waktu, pengguna dapat meminta data statistik (/stats). Sistem melakukan agregasi data dari basis data dan mengembalikan ringkasan kinerja operasional.

2.4 Class Diagram Domain Model

Diagram kelas berikut merepresentasikan struktur logis dari Reservation Management Domain pada tahap finalisasi. Berbeda dengan desain awal yang hanya berfokus pada struktur data, diagram ini telah diperbarui untuk mencerminkan perilaku bisnis (business behaviors) yang diimplementasikan secara nyata dalam kode Python (src/domain/models.py).



Tabel Definisi *Class Diagram*

Relasi	Jenis Relasi UML	Keterangan
Reservation – TableAssignment	*Composition (1–n)*	Satu reservasi dapat mencakup beberapa meja yang dialokasikan.
Reservation – ReservationPolicy	*Association (1–1)*	Setiap reservasi mengikuti satu kebijakan reservasi tertentu.
Reservation – ReservationTime	*Association (1–1)*	Waktu reservasi unik untuk setiap pemesanan.
Reservation – ReservationHistory	*Aggregation (1–n)*	Setiap reservasi memiliki riwayat aktivitas (log).
Reservation – PaymentDetail	*Association (1–1)*	Setiap reservasi memiliki detail pembayaran deposit.
Reservation – ContactInfo	*Composition (1–1)*	Informasi kontak pelanggan melekat sepenuhnya pada reservasi dan tidak berdiri sendiri.
Reservation – ReservationStatus	*Association (1–1)*	Penentuan status hasil reservasi untuk satu reservasi.
Reservation – CancellationReason	*Dependency*	Pembatalan reservasi tertentu dengan melibatkan alasan.

Waitlist – Reservation	*Dependency*	Pelanggan di Waitlist dapat dipromosikan menjadi Reservation.
Reservation – EventTrigger	*Dependency (event-based)*	Perubahan status pada reservasi memicu event ke Notification Context.
Reservation – StaffAssignment	*Association (1–n)*	Setiap reservasi dapat memiliki satu atau lebih staf yang ditugaskan.
Waitlist – ReservationTime	*Association (1–1)*	Waitlist menyimpan waktu pertama kalinya reservasi/waitlist dibuat.

Penjelasan Komponen Utama:

1. **Reservation (Aggregate Root)**: Sebagai pusat kendali, kelas ini tidak hanya menyimpan data, tetapi juga memiliki method transaksional penting:
    - **confirm()**: Memvalidasi perubahan dari Pending ke Confirmed.
    - **assignTable()**: Mengelola relasi dengan entitas TableAssignment.
    - **checkIn()** dan **complete()**: Menangani siklus hidup operasional tamu di restoran, memastikan status berubah sesuai urutan logis.
2. **Value Objects (Immutability)**: Komponen seperti ReservationTime, ContactInfo, dan ReservationPolicy digambarkan sebagai Value Objects. Dalam diagram, mereka terlihat terpisah untuk menjaga kejelasan konsep (High Cohesion). Namun, dalam implementasi fisik di basis data (seperti dijelaskan pada sub-bab 2.2), atribut-atribut dari objek ini disimpan menggunakan teknik Flattening (dilebur ke tabel induk) untuk efisiensi performa tanpa mengorbankan integritas model domain.
3. **ReservationStatus**: Status reservasi dimodelkan sebagai Value Object dengan logika validasi internal, memastikan transisi status hanya terjadi pada alur yang valid (misalnya: tidak bisa melakukan Check-in jika status masih Pending).

Diagram ini menjadi cetak biru (blueprint) yang dijaga konsistensinya dengan kode melalui penerapan Test-Driven Development (TDD), di mana setiap metode pada diagram memiliki pasangan unit test yang memvalidasi perilakunya.

BAB III Implementasi Layanan

3.1 Struktur Proyek dan Konfigurasi Lingkungan

Implementasi layanan backend disusun menggunakan struktur direktori yang modular untuk memisahkan tanggung jawab antar komponen sistem (Separation of Concerns). Struktur ini dirancang untuk mendukung skalabilitas, kemudahan pengujian (testability), dan alur kerja integrasi berkelanjutan (CI/CD).

3.1.1 Struktur Direktori

Untuk menjaga performa dan kesederhanaan skema, sistem menerapkan strategi Flattening (Peleburan). Dalam desain domain, konsep seperti ContactInfo, ReservationTime, dan PaymentDetail dipisahkan sebagai Value Objects. Namun, pada lapisan infrastruktur basis data, atribut-atribut dari objek tersebut dilebur menjadi kolom-kolom datar di dalam satu tabel utama yaitu reservations. Strategi ini dipilih untuk:

```
Restaurant-Reservation/
├── .github/
│   └── workflows/
│       └── ci.yml
├── src/
│   ├── api/
│   │   ├── auth.py
│   │   └── routes.py
│   ├── core/
│   │   ├── config.py
│   │   └── security.py
│   ├── domain/
│   │   ├── events.py
│   │   ├── models.py
│   │   └── value_objects.py
│   ├── infrastructure/
│   │   ├── database.py
│   │   └── orm_models.py
│   └── schemas/
│       └── reservation.py
├── tests/
│   ├── conftest.py
│   ├── test_api.py
│   ├── test_domain.py
│   └── test_security.py
├── docker-compose.yml
├── Dockerfile
├── main.py
├── pytest.ini
└── requirements.txt
```

3.1.2 Konfigurasi Lingkungan (Environment Configuration)

Sistem dirancang mengikuti metodologi The Twelve-Factor App, di mana konfigurasi dipisahkan dari kode sumber. Seluruh variabel sensitif dan konfigurasi infrastruktur dikelola melalui Environment Variables.

Pada lingkungan pengembangan lokal dan deployment (Docker), konfigurasi utama meliputi:

Variabel	Deskripsi	Contoh Nilai
DATABASE_URL	Connection string ke basis data PostgreSQL.	postgresql://user:pass@db:5432/restoran_db
SECRET_KEY	Kunci rahasia untuk menandatangani token JWT.	Tubes TST
ALGORITHM	Algoritma enkripsi token.	HS256
ACCESS_TOKEN_EXPIRE_MINUTES	Durasi masa berlaku token akses.	30

Pengaturan ini memungkinkan aplikasi dijalankan di berbagai lingkungan (Lokal, Testing, Staging, Produksi) tanpa perlu mengubah satu baris pun pada kode program, cukup dengan menyesuaikan nilai variabel lingkungan pada kontainer Docker atau platform cloud.

3.2 Implementasi Logika Bisnis (Aggregate & Entities)

Logika bisnis inti sistem diimplementasikan di dalam Domain Layer (src/domain/). Sesuai prinsip DDD, lapisan ini dibangun menggunakan Pure Python Objects tanpa ketergantungan langsung pada pustaka eksternal seperti FastAPI atau SQLAlchemy. Hal ini memastikan aturan bisnis tetap bersih, mudah dipahami, dan dapat diuji secara terisolasi (unit testing).

3.2.1 Reservation Aggregate Root

Kelas Reservation bertindak sebagai Aggregate Root. Ia tidak hanya berfungsi sebagai wadah data, tetapi juga sebagai penjaga konsistensi (consistency boundary). Setiap perubahan status pada reservasi harus melalui metode-metode yang ada di dalam kelas ini untuk memastikan tidak ada aturan bisnis yang dilanggar (invariants).

Berikut adalah implementasi logika transisi status pada kelas Reservation:

```
class Reservation:
    def __init__(
        self,
```

```
        customer_id: uuid.UUID,
        contact_info: ContactInfo,
        reservation_time: ReservationTime,
        policy: ReservationPolicy
    ):
        self.reservation_id = uuid.uuid4()
        self.customer_id = customer_id
        self.contact_info = contact_info
        self.reservation_time = reservation_time
        self.policy = policy

        self.status = ReservationStatus.PENDING
        self.payment_detail = PaymentDetail()

        self.table_assignment: Optional[TableAssignment] = None
        self.history: List[ReservationHistory] = []

        self.domain_events: List[DomainEvent] = []

        self._record_history("CREATED", "Reservation created")

        self.domain_events.append(ReservationCreated(
            reservation_id=self.reservation_id,
            customer_id=self.customer_id,
            start_time=self.reservation_time.start_time
        ))

    def _record_history(self, action: str, note: str):
        log = ReservationHistory(action, note)
        self.history.append(log)

    def confirm_reservation(self):
        if self.status == ReservationStatus.CANCELLED:
            raise ValueError("Cannot confirm a cancelled reservation.")

        self.status = ReservationStatus.CONFIRMED
        self._record_history("CONFIRMED", "Reservation confirmed by
system/staff")

        self.domain_events.append(ReservationConfirmed(
```

```
            reservation_id=self.reservation_id
        ))

    def cancel_reservation(self, reason: CancellationReason):
        if self.status == ReservationStatus.COMPLETED:
            raise ValueError("Cannot cancel a completed reservation.")

        self.status = ReservationStatus.CANCELLED
        self._record_history("CANCELLED", f"{reason.reason_code}:
{reason.description}")

        self.domain_events.append(ReservationCancelled(
            reservation_id=self.reservation_id,
            reason=reason.reason_code
        ))

    def assign_table(self, table_id: uuid.UUID, capacity: int, area: str):
        if self.status == ReservationStatus.CANCELLED:
            raise ValueError("Cannot assign table to cancelled
reservation.")

        self.table_assignment = TableAssignment(table_id, capacity, area)
        self._record_history("TABLE_ASSIGNED", f"Assigned to Table ID
{table_id} ({area})")

    def collect_domain_events(self) -> List[DomainEvent]:
        events = self.domain_events[:]
        self.domain_events.clear()
        return events
```

3.2.2 Manajemen Aturan Bisnis (Business Rules)

Sistem menerapkan beberapa aturan bisnis kunci yang dieksekusi oleh Aggregate Root di atas:

1. **Validasi Alur Status (State Flow Validation)**: Sistem memaksa perubahan status terjadi secara berurutan: PENDING → CONFIRMED → CHECKED_IN $\rightarrow$ COMPLETED. Lompatan status yang tidak logis (misalnya dari PENDING langsung COMPLETED) akan ditolak dengan DomainError.
2. **Immutability pada Value Objects**: Objek seperti ReservationTime dan ContactInfo diimplementasikan sebagai komponen yang tidak dapat diubah (immutable). Jika ada perubahan data (misalnya pelanggan ganti nomor HP), sistem akan mengganti keseluruhan

objek ContactInfo dengan yang baru, bukan mengubah atribut internalnya, untuk mencegah efek samping yang tidak diinginkan.

3. **Pemisahan Data dan Perilaku**: Meskipun penyimpanan data menggunakan ORM (Infrastructure Layer), logika di atas memastikan bahwa manipulasi status tidak dilakukan sembarangan melalui query database langsung, melainkan harus memanggil metode domain yang telah teruji.

3.3 Spesifikasi Endpoint API (Request & Response Payload)

Seluruh pertukaran data dalam sistem ini menggunakan format JSON (JavaScript Object Notation). Struktur respons dirancang mengikuti pola Data Enrichment, di mana data mentah dari basis data dikelompokkan ke dalam objek bersarang (nested objects) seperti customer_details dan booking_info untuk meningkatkan keterbacaan dan kemudahan konsumsi oleh klien.

Berikut adalah spesifikasi muatan (payload) untuk endpoint-endpoint:

3.3.1 POST /api/token

Endpoint ini digunakan untuk melakukan autentikasi pengguna. Klien mengirimkan username dan password, dan jika valid, server akan mengembalikan Access Token (JWT). Token ini selanjutnya wajib disertakan pada header setiap permintaan ke endpoint reservasi.

URL	`/api/token`	
Method	`POST`	
Content-Type	`application/x-www-form-urlencoded`	
Parameter		
Nama	**Tipe**	**Keterangan**
username	`string`	Nama pengguna.
password	`string`	Kata sandi pengguna.
Contoh Request (cURL)		

```
curl -X POST "http://localhost:8000/api/token" \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d "username=admin&password=password123"
```

Response	
Code	**Value**
200 (Successful	`{`

Response)	<pre>{
	  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
	  "token_type": "bearer"
	}</pre>

422 (Validation Error)	<pre>{
	  "detail": [
	  {
	  "loc": [
	  "string",
	  0
	  ],
	  "msg": "string",
	  "type": "string"
	  }
	  ]
	}</pre>

3.3.2 POST /api/reservations

Endpoint ini digunakan untuk membuat reservasi baru ke dalam sistem. Berbeda dengan implementasi awal, endpoint ini kini bersifat Protected. Klien wajib menyertakan token akses yang valid (didapatkan dari /api/token) untuk dapat melakukan transaksi ini.

URL	`/api/reservations`	
Method	`POST`	
Content-Type	`application/json`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
customer_id	`UUID`	ID unik pelanggan (format UUID v4).
contact_info	`Object`	Berisi detail kontak (name, phone, email).
start_time	`Datetime`	Waktu mulai reservasi (ISO 8601).
duration_minutes	`Integer`	Durasi reservasi dalam menit (Default: 90).
Request Body		

```
{
  "customer_id": "string",
  "contact_info": {
    "name": "string",
    "phone": "string",
    "email": "user@example.com"
  },
  "start_time": "2025-12-01T07:26:45.853Z",
  "duration_minutes": 90
}
```


Response

Code	Value
200 (Successful Response)	`{ "reservation_id": "string", "status": "PENDING", "table_area": "string", "customer_details": { "id": "string", "name": "string", "phone": "string", "email": "user@example.com" }, "booking_info": { "start_time": "2025-12-12T10:14:03.548Z", "end_time": "2025-12-12T10:14:03.548Z", "duration_minutes": 0, "is_peak_hour": false }, "payment_info": { "status": "string", "amount": 0, "currency": "IDR" }, "meta": { "api_version": "v1", "response_generated_at": "2025-12-12T10:14:03.548Z" } }`
422 (Validation Error)	`{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ]`

	}

3.3.3 GET /api/reservations/{reservation_id}

Endpoint ini digunakan untuk mengambil informasi lengkap mengenai satu reservasi spesifik berdasarkan ID-nya. Endpoint ini bersifat Protected, artinya hanya pengguna yang memiliki token akses valid yang diizinkan untuk melihat data reservasi.

URL	/api/reservations/{reservation_id}	
Method	GET	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
reservation_id	UUID	ID unik reservasi yang ingin dilihat.
Response		
Code	**Value**	

200 (Successful Response)	<pre>{
	  "reservation_id": "string",
	  "status": "PENDING",
	  "table_area": "string",
	  "customer_details": {
	  "id": "string",
	  "name": "string",
	  "phone": "string",
	  "email": "user@example.com"
	  },
	  "booking_info": {
	  "start_time": "2025-12-12T10:14:41.057Z",
	  "end_time": "2025-12-12T10:14:41.057Z",
	  "duration_minutes": 0,
	  "is_peak_hour": false
	  },
	  "payment_info": {
	  "status": "string",
	  "amount": 0,
	  "currency": "IDR"
	  },
	  "meta": {
	  "api_version": "v1",</pre>	

	<pre>"response_generated_at": "2025-12-12T10:14:41.057Z"
	}
	}</pre>

422 (Validation Error)	<pre>{
	"detail": [
	{
	"loc": [
	"string",
	0
	],
	"msg": "string",
	"type": "string"
	}
	]
	}</pre>

3.3.4 POST /api/reservations/{reservation_id}/confirm

Endpoint ini digunakan untuk mengubah status reservasi dari PENDING menjadi CONFIRMED. Tindakan ini bersifat Protected, yang berarti hanya pengguna terautentikasi (seperti staf restoran atau admin) yang memiliki izin untuk memvalidasi reservasi.

URL	`/api/reservations/{reservation_id}/confirm`	
Method	`POST`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
reservation_id	`UUID`	ID reservasi yang ingin dikonfirmasi.
Response		
Code	**Value**	
200 (Successful Response)	`"string"`	

422 (Validation Error)	<pre>{
	"detail": [
	{
	"loc": [
	"string",
	0</pre>	

	<pre>],
	      "msg": "string",
	      "type": "string"
	    }
	  ]
	}</pre>

3.3.5 POST /api/reservations/{reservation_id}/assign-table

Endpoint ini digunakan untuk mengalokasikan meja fisik ke dalam reservasi yang sudah ada. Informasi table_id dan area akan diperbarui secara persisten di basis data, menghubungkan entitas reservasi dengan manajemen meja.

URL	`/api/reservations/{reservation_id}/assign-table`	
Method	`POST`	
Content-Type	`application/json`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
reservation_id	`UUID`	ID reservasi yang akan dialokasikan meja.
Parameter		
Nama	**Tipe**	**Keterangan**
table_id	`UUID`	ID unik meja dari sistem manajemen meja.
capacity	`Integer`	Kapasitas kursi meja tersebut.
area	`String`	Area lokasi meja (misal: "Indoor", "Outdoor", "VIP").
Request Body		

<pre>{
  "table_id": "string",
  "capacity": 0,
  "area": "string"
}</pre>		

Response		
Code	**Value**	

200 (Successful Response)	`"string"`

422 (Validation Error)	`{`
	`  "detail": [`
	`  {`
	`  "loc": [`
	`  "string",`
	`  0`
	`  ],`
	`  "msg": "string",`
	`  "type": "string"`
	`  }`
	`  ]`
	`}`

3.3.6 POST /api/reservations/{reservation_id}/cancel

Endpoint ini menangani pembatalan reservasi. Sistem mewajibkan penyertaan alasan pembatalan (reason_code) dan menerapkan aturan bisnis untuk mencegah pembatalan pada reservasi yang sudah selesai (COMPLETED). Status akhir diubah menjadi CANCELLED.

URL	`/api/reservations/{reservation_id}/cancel`	
Method	`POST`	
Content-Type	`application/json`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
reservation_id	`UUID`	ID reservasi yang akan dibatalkan.
Parameter		
Nama	**Tipe**	**Keterangan**
reason_code	`String`	Kode singkat alasan (misal: "CUSTOMER_REQUEST", "NO_SHOW").
description	`String`	Penjelasan detail mengenai alasan pembatalan.
Request Body		
`{`		

<table>
<tr><td colspan="3"><pre> "reason_code": "string",
  "description": "string"
}</pre></td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td>Code</td><td colspan="2">Value</td></tr>
<tr><td>200 (Successful Response)</td><td colspan="2">"string"</td></tr>
</table>

3.3.7 GET /api/reservations

Endpoint ini digunakan untuk mengambil daftar koleksi reservasi. Mendukung fitur Pagination (skip, limit) dan Filtering berdasarkan status untuk efisiensi pengambilan data. Endpoint ini sangat berguna untuk menampilkan daftar antrean atau riwayat reservasi pada antarmuka dashboard admin.

<table>
<tr><td>URL</td><td colspan="2">/api/reservations</td></tr>
<tr><td>Method</td><td colspan="2">GET</td></tr>
<tr><td colspan="3">Header</td></tr>
<tr><td>Key</td><td colspan="2">Authorization</td></tr>
<tr><td>Value</td><td colspan="2">Bearer <access_token></td></tr>
<tr><td colspan="3">Parameter</td></tr>
<tr><td>Nama</td><td>Tipe</td><td>Keterangan</td></tr>
<tr><td>skip</td><td>integer</td><td>Jumlah data yang dilewati (Offset). Default: 0.</td></tr>
<tr><td>limit</td><td>integer</td><td>Jumlah maksimal data yang diambil. Default: 10.</td></tr>
<tr><td>status</td><td>String</td><td>(Opsional) Filter berdasarkan status reservasi (misal: "PENDING").</td></tr>
<tr><td colspan="3">Response</td></tr>
<tr><td>Code</td><td colspan="2">Value</td></tr>
<tr><td>200 (Successful Response)</td><td colspan="2"><pre>[
  {
    "reservation_id": "string",
    "status": "PENDING",
    "table_area": "string",
    "customer_details": {
      "id": "string",
      "name": "string",
      "phone": "string",
      "email": "user@example.com"</pre></td></tr>
</table>

	<pre> },
	    "booking_info": {
	      "start_time": "2025-12-12T10:21:27.409Z",
	      "end_time": "2025-12-12T10:21:27.409Z",
	      "duration_minutes": 0,
	      "is_peak_hour": false
	    },
	    "payment_info": {
	      "status": "string",
	      "amount": 0,
	      "currency": "IDR"
	    },
	    "meta": {
	      "api_version": "v1",
	      "response_generated_at": "2025-12-12T10:21:27.409Z"
	    }
	  }
	]</pre>

422 (Validation Error)	<pre>{
	  "detail": [
	    {
	      "loc": [
	        "string",
	        0
	      ],
	      "msg": "string",
	      "type": "string"
	    }
	  ]
	}</pre>

3.3.8 POST /api/reservations/{reservation_id}/check-in

Endpoint ini menandai kedatangan aktual pelanggan di restoran. Logika bisnis diterapkan secara ketat dengan sistem akan menolak permintaan ini jika status reservasi belum CONFIRMED. Jika valid, status akan diperbarui menjadi CHECKED_IN.

URL	`/api/reservations/{reservation_id}/check-in`	
Method	`POST`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**

reservation_id	`UUID`	ID reservasi pelanggan yang telah tiba di lokasi.
Response		
Code	**Value**	
200 (Successful Response)	`"string"`	
422 (Validation Error)	`{ "detail": [ { "loc": [ "string", 0 ], "msg": "string", "type": "string" } ] }`	

3.3.9 POST /api/reservations/{reservation_id}/complete

Endpoint ini menandai penyelesaian siklus layanan (tamu selesai makan dan meninggalkan meja). Status diubah menjadi COMPLETED dan meja dianggap kosong kembali. Validasi memastikan hanya reservasi yang sedang berjalan (CHECKED_IN) yang dapat diselesaikan.

URL	`/api/reservations/{reservation_id}/complete`	
Method	`POST`	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	**Tipe**	**Keterangan**
reservation_id	`UUID`	ID reservasi yang akan diselesaikan (tamu pulang).
Response		
Code	**Value**	
200 (Successful Response)	`"string"`	
422 (Validation	`{`	

Error)	<pre>{
	  "detail": [
	  {
	  "loc": [
	  "string",
	  0
	  ],
	  "msg": "string",
	  "type": "string"
	  }
	  ]
	}</pre>

3.3.10 GET /api/stats

Endpoint analitik ini menyediakan ringkasan kinerja operasional secara real-time. Menggunakan fungsi agregasi SQL, endpoint ini mengembalikan data statistik seperti total reservasi berdasarkan status dan estimasi total pendapatan (revenue) tanpa membebani aplikasi dengan perhitungan manual.

URL	`/api/stats`
Method	`GET`
Header	
Key	Authorization
Value	Bearer <access_token>
Response	
Code	**Value**

200 (Successful Response)	<pre>{
	  "total_reservations": 0,
	  "pending_count": 0,
	  "confirmed_count": 0,
	  "checked_in_count": 0,
	  "completed_count": 0,
	  "cancelled_count": 0,
	  "total_revenue": 0,
	  "generated_at": "2025-12-12T10:25:34.380Z"
	}</pre>

3.4 Implementasi Keamanan (Security Implementation)

Aspek keamanan merupakan komponen kritis dalam arsitektur sistem ini. Implementasi keamanan difokuskan pada dua area utama: perlindungan data kredensial pengguna (Data Protection) dan kontrol akses terhadap sumber daya API (Access Control). Seluruh logika keamanan dipusatkan pada modul src/core/security.py untuk memastikan konsistensi.

3.4.1 Enkripsi Kata Sandi (Password Hashing)

Sistem menerapkan kebijakan Zero-Trust terhadap penyimpanan kata sandi. Kata sandi pengguna tidak pernah disimpan dalam bentuk teks biasa (plain text) di dalam basis data. Sebagai gantinya, sistem menggunakan algoritma Bcrypt untuk melakukan hashing satu arah.

Proses ini menggunakan pustaka passlib dengan skema enkripsi yang kuat. Setiap kali pengguna mencoba masuk (login), kata sandi yang dimasukkan akan di-hash dan dicocokkan dengan hash yang tersimpan di basis data. Hal ini menjamin bahwa meskipun basis data bocor, kredensial asli pengguna tetap terlindungi dan sulit diretas melalui metode brute-force atau rainbow table.

3.4.2 Autentikasi Berbasis Token (JWT)

Untuk menangani sesi pengguna, sistem menggunakan mekanisme JSON Web Token (JWT) yang bersifat stateless. Pendekatan ini dipilih karena skalabilitasnya yang tinggi dibandingkan sesi berbasis server (server-side session), serta kemudahannya untuk digunakan lintas platform (Web/Mobile).

Alur Autentikasi:

1. **Login**: Pengguna mengirimkan kredensial ke /api/token.
2. **Issuance**: Jika valid, server menerbitkan token JWT yang ditandatangani secara digital menggunakan algoritma HS256 dan SECRET_KEY. Token ini berisi klaim identitas (sub) dan waktu kedaluwarsa (exp).
3. **Usage**: Pengguna menyimpan token ini dan mengirimkannya kembali pada header HTTP Authorization: Bearer <token> untuk setiap permintaan ke endpoint yang diproteksi.

3.4.3 Proteksi Endpoint (Dependency Injection)

FastAPI menyediakan mekanisme Dependency Injection yang dimanfaatkan untuk memproteksi endpoint. Fungsi dependensi get_current_user disuntikkan ke dalam rute API yang membutuhkan otorisasi (misalnya: /check-in atau /stats).

Dependensi ini bekerja dengan cara:

1. Mengekstrak token dari header permintaan.
2. Memvalidasi tanda tangan digital token.
3. Memastikan token belum kedaluwarsa.
4. Jika valid, permintaan diteruskan ke fungsi utama. Jika tidak, sistem secara otomatis mengembalikan respons 401 Unauthorized

BAB IV Quality Assurance & Test Driven Development

4.1 Metodologi TDD (Test-Driven Development)

Dalam pengembangan sistem ini, pendekatan Test-Driven Development (TDD) diterapkan secara ketat sebagai fondasi utama penjaminan mutu. Berbeda dengan pendekatan tradisional di mana pengujian dilakukan setelah kode selesai ditulis, TDD membalik urutan tersebut dengan mewajibkan penulisan skenario pengujian sebelum implementasi fitur dimulai.

Proses pengembangan mengikuti siklus standar Red-Green-Refactor yang diulang untuk setiap fitur baru:

1. **Red (Fase Gagal)**: Pengembang menulis unit test untuk spesifikasi fitur yang belum ada (misalnya: "Tes gagal jika pengguna melakukan check-in pada reservasi yang statusnya masih Pending"). Pada tahap ini, tes dipastikan gagal (Failed) karena logika bisnis belum diimplementasikan.
2. **Green (Fase Sukses)**: Pengembang menulis kode implementasi seminimal mungkin—hanya cukup untuk membuat tes tersebut lulus (Passed). Fokus pada tahap ini adalah fungsionalitas, bukan keindahan kode.
3. **Refactor (Fase Penyempurnaan)**: Setelah tes lulus, kode diperbaiki (clean up), dioptimalkan, atau distruktur ulang tanpa mengubah perilaku eksternalnya. Karena tes sudah hijau, pengembang memiliki jaring pengaman (safety net) untuk melakukan perubahan struktur kode tanpa takut merusak fitur yang sudah berjalan.

Penerapan TDD dalam proyek ini menggunakan kerangka kerja Pytest. Strategi pengujian dibagi menjadi dua lapisan utama:

- **Unit Testing (tests/test_domain.py)**: Menguji logika bisnis murni pada Domain Layer tanpa melibatkan basis data atau HTTP. Contoh: Memvalidasi transisi status reservasi.
- **Integration Testing (tests/test_api.py)**: Menguji interaksi antar komponen (API Routes, Service, dan Database) untuk memastikan data mengalir dengan benar dari request hingga tersimpan di PostgreSQL.

Pendekatan ini terbukti efektif dalam mencegah regresi (bugs yang muncul kembali) saat dilakukan migrasi infrastruktur dari in-memory ke PostgreSQL, serta menjamin stabilitas sistem saat penambahan fitur baru dilakukan.

4.2 Skenario Pengujian

Lingkup pengujian dalam proyek ini mencakup seluruh lapisan aplikasi, mulai dari unit terkecil (fungsi utilitas) hingga integrasi antar-komponen (API ke Database). Skenario pengujian dikelompokkan menjadi empat kategori utama: Logika Domain, Keamanan, Alur Bisnis (Happy Paths), dan Penanganan Kesalahan (Edge Cases).

4.2.1 Pengujian Unit Domain (Business Logic)

Pengujian ini berfokus pada validasi aturan bisnis di dalam src/domain/models.py. Tes ini dijalankan secara terisolasi tanpa koneksi ke basis data.

File: tests/test_domain.py

ID Skenario	Deskripsi Pengujian	Hasil yang Diharapkan
DOM-01	Transisi status normal (Pending → Confirmed).	Status berubah menjadi CONFIRMED.
DOM-02	Validasi aturan Check-In (hanya jika Confirmed).	Jika status awal PENDING, sistem melempar DomainError.
DOM-03	Validasi aturan Completion (hanya jika Checked-In).	Jika status awal CONFIRMED, sistem melempar DomainError.
DOM-04	Pencegahan pembatalan pada reservasi selesai.	Reservasi berstatus COMPLETED tidak boleh diubah menjadi CANCELLED.
DOM-05	Validasi Table Assignment.	Entitas TableAssignment berhasil terhubung dengan Reservation.

4.2.2 Pengujian Keamanan (Security & Auth)

Pengujian ini memverifikasi keandalan mekanisme enkripsi dan pembuatan token.

File: tests/test_security.py

ID Skenario	Deskripsi Pengujian	Hasil yang Diharapkan
SEC-01	Hashing kata sandi.	Teks asli tidak boleh sama dengan hasil hash. Verifikasi hash valid.
SEC-02	Pembuatan JWT Token.	Token yang dihasilkan dapat didekode kembali dan memuat data pengguna yang benar.
SEC-03	Validasi Token Kedaluwarsa.	Token yang masa berlakunya habis harus ditolak saat verifikasi.

4.2.3 Pengujian Integrasi API (Happy Paths)

Pengujian ini mensimulasikan alur penggunaan normal oleh pengguna, memastikan data mengalir dengan benar dari request HTTP hingga tersimpan di PostgreSQL.

File: tests/test_api.py

ID Skenario	Deskripsi Pengujian	Hasil yang Diharapkan
API-01	Login dengan kredensial valid.	Mengembalikan 200 OK dan access_token.
API-02	Membuat reservasi baru.	Mengembalikan 200 OK dan data tersimpan di DB dengan status PENDING.
API-03	Mengambil daftar reservasi (List).	Mengembalikan Array JSON berisi data reservasi.
API-04	Alur Bisnis Penuh (Full Cycle).	Reservasi sukses melewati status: Create → Confirm → Check-in → Complete.
API-05	Dashboard Statistik (/stats).	Mengembalikan angka agregasi yang akurat sesuai data di DB.

4.2.4 Pengujian Kasus Ekstrem & Kesalahan (Edge Cases)

Pengujian ini memastikan aplikasi tidak crash saat menerima input yang salah atau tidak valid.

File: tests/test_api.py

ID Skenario	Deskripsi Pengujian	Hasil yang Diharapkan
ERR-01	Akses endpoint tanpa token (Unauthorized).	Mengembalikan 401 Unauthorized.
ERR-02	Login dengan kata sandi salah.	Mengembalikan 401 Unauthorized.
ERR-03	Mengakses ID reservasi yang tidak ada (UUID acak).	Mengembalikan 404 Not Found.
ERR-04	Check-In prematur (Status masih Pending).	Mengembalikan 400 Bad Request dengan pesan kesalahan bisnis yang jelas.
ERR-05	Format UUID tidak valid pada URL.	Mengembalikan 422 Unprocessable Entity (Validasi Pydantic).

4.3 Laporan Coverage (Test Coverage Report)

Untuk mengukur efektivitas pengujian, sistem menggunakan alat bantu pytest-cov. Alat ini menganalisis baris kode mana saja yang dieksekusi selama rangkaian pengujian berjalan dan menghitung persentasenya. Target kualitas yang ditetapkan untuk proyek ini adalah minimal 95%.

Perintah yang dieksekusi untuk menghasilkan laporan ini adalah:

- `pytest --cov=src --cov-report=term-missing --cov-fail-under=95`, atau
- `docker-compose exec web pytest --cov=src --cov-report=term-missing --cov-fail-under=95`, jika menggunakan docker.

4.3.1 Hasil Eksekusi

Berdasarkan pengujian terakhir, sistem berhasil mencapai tingkat cakupan kode rata-rata (Total Coverage) sebesar 96%. Angka ini melampaui target minimum yang ditetapkan, menandakan bahwa hampir seluruh logika bisnis, percabangan kondisi (if-else), dan penanganan kesalahan (error handling) telah terverifikasi.

```
Name                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
src/api/auth.py                        12      0   100%
src/api/routes.py                      83      0   100%
src/core/security.py                   37      4    89%   47-49, 52
src/domain/events.py                   16      0   100%
src/domain/models.py                   55      3    95%   90-92
src/domain/value_objects.py            33      1    97%   26
src/infrastructure/database.py         13      4    69%   14-18
src/infrastructure/orm_models.py       18      0   100%
src/schemas/reservation.py             54      0   100%
-----------------------------------------------------------------
TOTAL                                 321     12    96%
Required test coverage of 95% reached. Total coverage: 96.26%
```

Tingginya angka coverage ini memberikan tingkat kepercayaan diri yang tinggi (high confidence) untuk melakukan deployment ke lingkungan produksi, karena risiko kegagalan fungsi (bugs) telah diminimalkan secara signifikan sejak tahap pengembangan.

BAB V Continuous Integration & Deployment (CI/CD)

5.1 Konfigurasi GitHub Workflows

Untuk menjamin kualitas kode yang berkelanjutan, sistem menerapkan praktik Continuous Integration (CI) menggunakan layanan GitHub Actions. Seluruh definisi alur kerja (workflow) otomasi ditulis dalam format YAML dan disimpan pada direktori .github/workflows/ci.yml. Konfigurasi ini bertindak sebagai instruksi bagi server GitHub untuk menyiapkan lingkungan, menjalankan pengujian, dan melaporkan hasilnya setiap kali ada perubahan kode.

5.1.1 Pemicu Alur Kerja (Workflow Triggers)

Pipa saluran (pipeline) CI dikonfigurasi untuk berjalan secara otomatis pada dua kondisi utama (events):

1. **Push**: Setiap kali pengembang mendorong kode (push) ke cabang utama (main).
2. **Pull Request**: Setiap kali ada permintaan penggabungan kode (pull request) yang ditujukan ke cabang utama.

Mekanisme ini berfungsi sebagai gerbang kualitas (quality gate), mencegah kode yang rusak atau tidak memenuhi standar masuk ke dalam basis kode utama (production codebase).

5.1.2 Lingkungan Eksekusi & Layanan Basis Data

Salah satu tantangan utama dalam pengujian aplikasi backend adalah ketergantungan pada basis data. Dalam konfigurasi ini, GitHub Actions tidak hanya menjalankan kode Python, tetapi juga memutar (spin up) layanan PostgreSQL sementara menggunakan fitur Service Containers.

Berikut adalah spesifikasi lingkungan eksekusi CI:

- **Operating System**: ubuntu-latest (Lingkungan Linux standar).
- **Python Version**: 3.10 (Sesuai dengan environment pengembangan).
- **Service Container**: Image postgres:15 yang berjalan di port 5432.
    - Sistem CI menunggu hingga layanan Postgres siap (health check) sebelum mulai menjalankan tes aplikasi. Hal ini menyimulasikan lingkungan produksi yang sesungguhnya di dalam runner CI.

5.1.3 Definisi Langkah (Jobs & Steps)

Alur kerja terdiri dari satu pekerjaan utama (job) bernama test yang berisi rangkaian langkah berurutan:

1. **Checkout Code**: Mengambil salinan kode terbaru dari repositori.
2. **Setup Python**: Menginstal interpreter Python versi 3.10.
3. **Install Dependencies**: Menginstal seluruh pustaka yang tercatat di requirements.txt (termasuk FastAPI, SQLAlchemy, Pytest, Flake8).
4. **Linting**: Memeriksa kualitas penulisan kode (gaya, sintaks, dan potensi bug) menggunakan flake8.
5. **Testing**: Menjalankan rangkaian uji pytest dengan menyuntikkan variabel lingkungan DATABASE_URL yang mengarah ke service container PostgreSQL.

Konfigurasi ini memastikan bahwa pengujian dilakukan dalam lingkungan yang bersih (clean slate) dan terisolasi, menghilangkan masalah tidak bisa running di device lain yang sering terjadi pada pengembangan manual.

5.2 Pipeline Tahapan (Stages)

Alur kerja CI/CD dirancang dengan pendekatan Fail Fast, di mana proses akan segera dihentikan jika ditemukan kesalahan pada tahapan awal. Hal ini mencegah pemborosan sumber daya dan memberikan umpan balik cepat kepada pengembang. Pipeline ini terdiri dari tiga tahapan kritis:

5.2.1 Pemeriksaan Kualitas Kode (Linting)

Tahap pertama adalah analisis statis kode menggunakan alat bantu Flake8. Pada tahap ini, kode sumber diperiksa tanpa dieksekusi untuk mendeteksi:

- Kesalahan sintaksis (Syntax Errors).
- Ketidakpatuhan terhadap standar gaya penulisan Python (PEP 8).
- Variabel yang tidak digunakan atau impor modul yang berlebihan.
- Kompleksitas kode yang terlalu tinggi (Cyclomatic Complexity).

Jika Flake8 menemukan pelanggaran standar (misalnya: baris terlalu panjang atau indentasi tidak konsisten), pipeline akan ditandai Gagal (Failed) dan pengembang wajib memperbaiki gaya penulisan kodenya sebelum melanjutkan.

5.2.2 Pengujian Otomatis (Automated Testing)

Setelah kode dinyatakan bersih secara sintaksis, tahap selanjutnya adalah menjalankan rangkaian pengujian fungsional menggunakan Pytest. Tahap ini adalah inti dari penjaminan mutu.

- **Integrasi Basis Data**: Runner CI menghubungkan aplikasi dengan kontainer layanan PostgreSQL yang telah disiapkan sebelumnya.
- **Eksekusi Test Suite**: Menjalankan seluruh skenario pengujian yang didefinisikan dalam folder tests/, mencakup unit test domain dan integrasi API.

- **Laporan Coverage**: Memastikan bahwa perubahan kode baru tidak menurunkan persentase cakupan tes (code coverage) di bawah ambang batas yang ditentukan.

Kegagalan satu saja dari puluhan skenario tes akan menyebabkan seluruh deployment dibatalkan, mencegah lolosnya bug ke lingkungan produksi.

5.2.3 Pembangungan Kontainer (Build & Containerization)

Tahap terakhir dalam pipeline integrasi adalah verifikasi pembangunan image aplikasi.

- Sistem menjalankan perintah docker build menggunakan Dockerfile yang tersedia.
- Tujuannya adalah memvalidasi bahwa seluruh dependensi sistem (requirements.txt) dapat diunduh dan diinstal dengan sukses, serta memastikan tidak ada kesalahan konfigurasi pada definisi kontainer.

Jika tahap ini berhasil, kode dianggap Valid dan siap untuk didistribusikan atau di-deploy ke server tujuan.

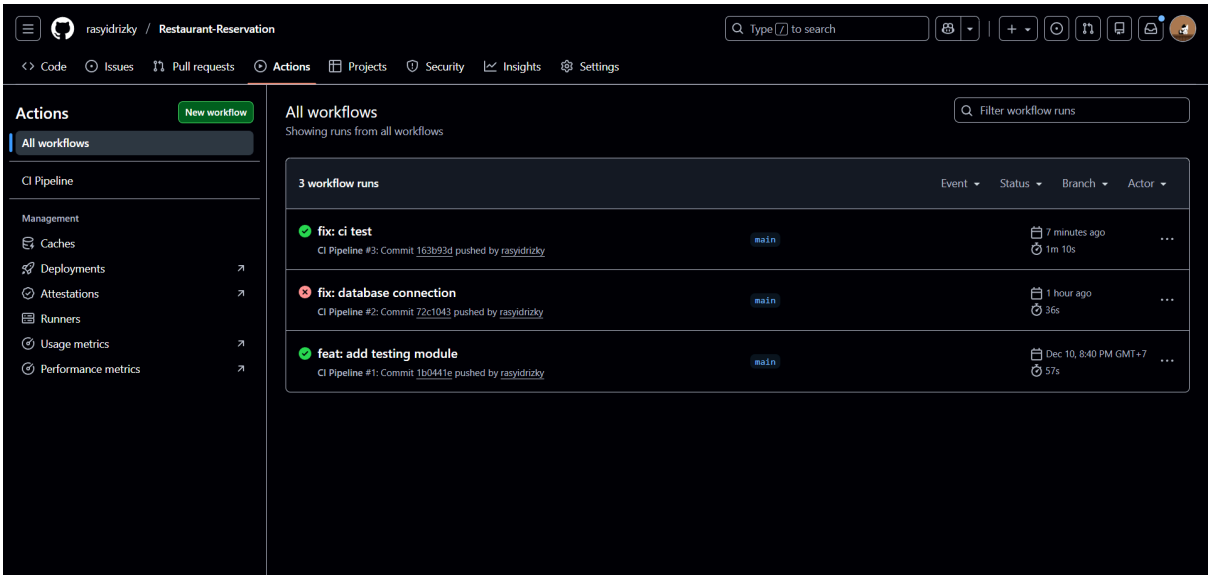
5.3 Bukti Keberhasilan CI

Keberhasilan konfigurasi Continuous Integration dibuktikan melalui riwayat eksekusi alur kerja pada repositori GitHub. Setiap kali kode baru didorong (push), GitHub Actions secara otomatis memicu alur kerja "CI Pipeline" dan melaporkan status akhirnya.

Tanda centang hijau (✓ Success) indikator mutlak bahwa kode tersebut telah lolos uji kualitas dan fungsionalitas.

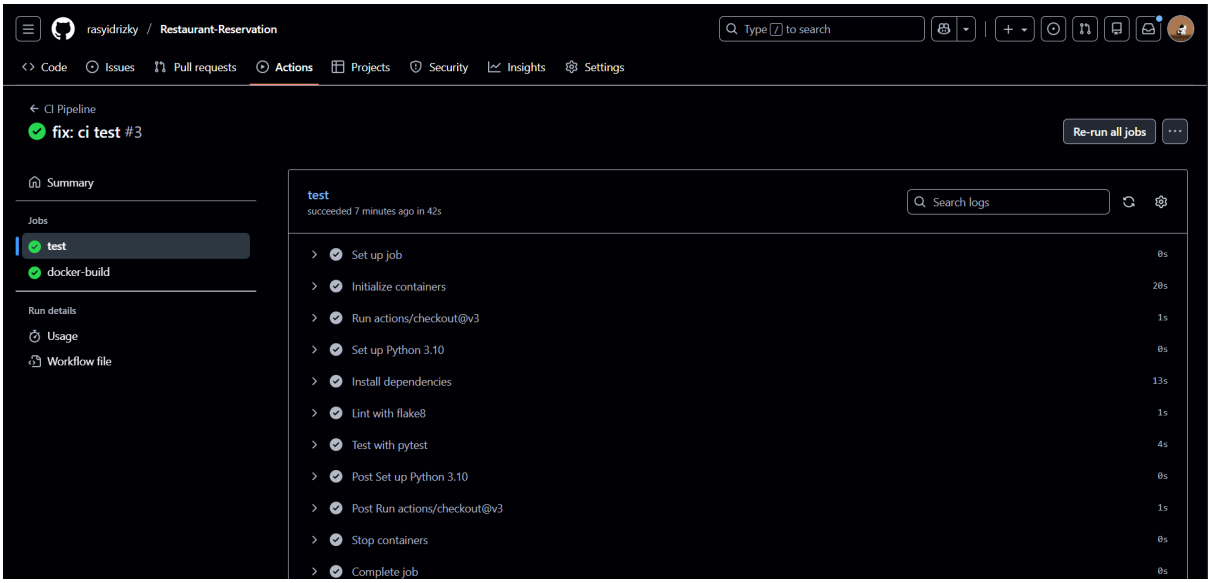
1. **Tampilan Riwayat Alur Kerja (Workflow Runs)**

   Gambar berikut menunjukkan daftar eksekusi CI terakhir pada tab Actions. Terlihat bahwa seluruh commit terakhir berhasil melewati proses pengujian tanpa kesalahan.



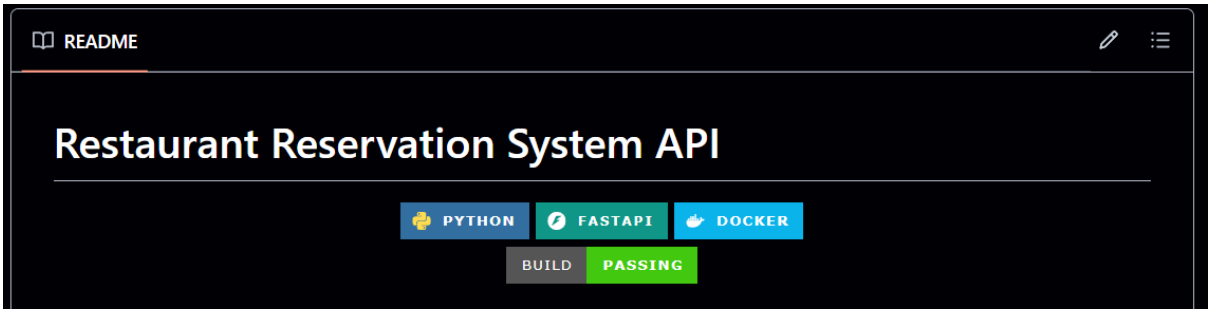
2. **Rincian Eksekusi Pekerjaan (Job Details)**

Gambar di bawah memperlihatkan rincian langkah-langkah yang dieksekusi dalam satu siklus CI. Dapat dilihat bahwa tahapan kritis seperti Set up Python, Lint with flake8, dan Run Tests telah selesai dengan status sukses, termasuk inisialisasi layanan basis data PostgreSQL.



3. **Lencana Status (Status Badge)**

Sebagai indikator cepat bagi pengguna atau pengembang lain yang mengunjungi repositori, lencana status "Passing" disematkan pada halaman utama (README). Lencana ini mencerminkan status kesehatan kode terkini dari cabang utama (main).



BAB VI Deployment & Dokumentasi Pengguna

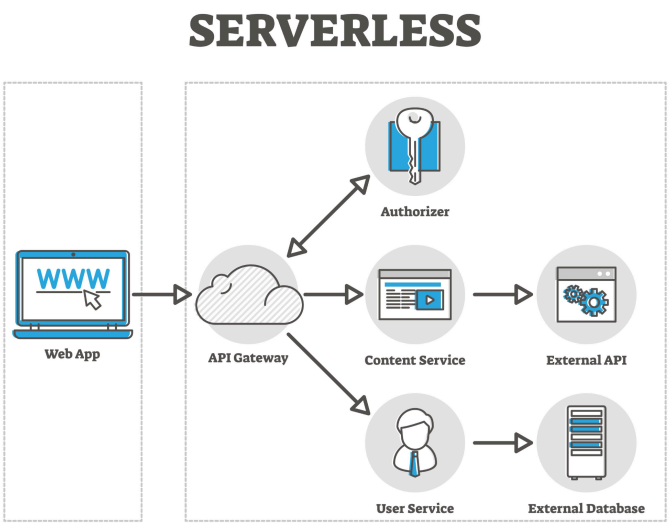
6.1 Arsitektur Deployment

Untuk kebutuhan lingkungan produksi (Production Environment), sistem ini menerapkan arsitektur berbasis Cloud-Native dan Serverless. Pendekatan ini dipilih karena menawarkan skalabilitas tinggi, biaya operasional yang efisien, dan kemudahan dalam manajemen infrastruktur tanpa perlu mengelola server fisik atau virtual machine secara manual.

6.1.1 Topologi Infrastruktur

Sistem di-host menggunakan kombinasi layanan Platform-as-a-Service (PaaS) modern. Berikut adalah komponen utama dalam arsitektur deployment:

1. **Application Layer (Vercel)**: Layanan backend FastAPI di-deploy ke Vercel. Di sini, aplikasi Python tidak berjalan sebagai server long-running (seperti dalam Docker), melainkan dikemas menjadi Serverless Functions. Setiap permintaan HTTP yang masuk akan memicu fungsi terisolasi yang menangani logika bisnis dan kemudian "tidur" kembali setelah respons dikirim. Hal ini meminimalkan penggunaan sumber daya saat sistem sedang idle.
2. **Database Layer (Supabase)**: Penyimpanan data persisten dikelola oleh Supabase, yang menyediakan layanan PostgreSQL terkelola (Managed Database). Karena sifat Serverless Function di Vercel yang dapat memicu banyak koneksi simultan dalam waktu singkat, sistem menggunakan Supavisor (Connection Pooler) pada port 6543 dengan mode Transaction. Ini mencegah kelebihan beban koneksi (connection exhaustion) pada basis data.
3. **Client Layer**: Pengguna mengakses sistem melalui internet menggunakan protokol aman HTTPS (TLS/SSL). Vercel menyediakan jaringan pengiriman konten (Edge Network) global yang memastikan latensi rendah dari berbagai lokasi akses.



6.1.2 Konfigurasi Keamanan Produksi

Berbeda dengan lingkungan lokal, konfigurasi di lingkungan produksi dikelola secara ketat melalui Environment Variables pada dasbor Vercel:

- **DATABASE_URL**: Mengarah ke alamat Connection Pooler Supabase, bukan alamat langsung basis data, untuk stabilitas koneksi IPv4 dan efisiensi pooling.
- **SECRET_KEY**: Menggunakan kunci kriptografi acak yang panjang dan kompleks untuk penandatanganan token JWT, berbeda dengan kunci yang digunakan saat pengembangan.

6.1.3 Perbedaan dengan Lingkungan Pengembangan

Penting untuk dicatat perbedaan mendasar antara lingkungan lokal dan produksi:

- **Lokal & CI:** Menggunakan Docker & Docker Compose untuk menjamin isolasi dan konsistensi lingkungan pengujian (reproducibility).
- **Produksi:** Menggunakan Serverless (Vercel) untuk skalabilitas otomatis (auto-scaling) dan ketersediaan tinggi (high availability) tanpa manajemen server manual.

6.2 Panduan Instalasi dan Penggunaan

Sistem ini dirancang agar mudah diinstal dan dijalankan menggunakan teknologi kontainerisasi. Berikut adalah panduan langkah demi langkah untuk menjalankan aplikasi di lingkungan lokal serta cara mengakses versi produksi yang telah di-deploy.

6.2.1 Instalasi Lingkungan Lokal (Local Development)

Untuk menjalankan aplikasi secara lokal guna keperluan pengembangan atau pengujian, pastikan perangkat telah terinstal Git dan Docker Desktop.

Langkah-langkah Instalasi:

1. Kloning Repositori: Unduh kode sumber dari repositori GitHub ke komputer lokal menggunakan terminal.
   ```
   git clone https://github.com/rasyidrizky/Restaurant-Reservation.git
   cd Restaurant-Reservation
   ```
2. Konfigurasi Lingkungan: Salin file contoh konfigurasi .env.example menjadi .env. Secara bawaan, konfigurasi ini sudah disesuaikan untuk berjalan di dalam jaringan Docker lokal tanpa perlu perubahan lebih lanjut.
   ```
   cp .env.example .env
   ```

3. Jalankan Kontainer: Gunakan Docker Compose untuk membangun (build) dan menjalankan layanan aplikasi beserta basis datanya. Perintah ini akan mengunduh image yang diperlukan dan menyalakan server.
   `docker-compose up --build -d`
4. Akses Aplikasi Lokal: Setelah kontainer berjalan, antarmuka dokumentasi API (Swagger UI) dapat diakses melalui peramban web pada alamat: http://localhost:8000/docs. Untuk menghentikan aplikasi, jalankan perintah: `docker-compose down`.

6.2.2 Akses Layanan Produksi (Live Demo)

Versi produksi sistem yang telah stabil dapat diakses secara publik tanpa perlu melakukan instalasi apa pun. Layanan ini terhubung dengan basis data cloud Supabase dan siap menerima permintaan HTTP.

- Base URL API: https://restaurant-reservation-alpha-eight.vercel.app
- Dokumentasi (Swagger UI): https://restaurant-reservation-alpha-eight.vercel.app/docs

Pengguna dapat langsung melakukan pengujian endpoint (seperti Login atau Create Reservation) melalui antarmuka Swagger UI tersebut menggunakan akun demo.

6.3 Panduan Instalasi dan Penggunaan

Sistem ini dirancang agar mudah diinstal dan dijalankan menggunakan teknologi kontainerisasi. Berikut adalah panduan langkah demi langkah untuk menjalankan aplikasi di lingkungan lokal serta cara mengakses versi produksi yang telah di-deploy.

6.3.1 Akses Dokumentasi

Dokumentasi antarmuka pengguna (Swagger UI) tersedia dan dapat diakses baik pada lingkungan pengembangan lokal maupun lingkungan produksi yang telah di-deploy.

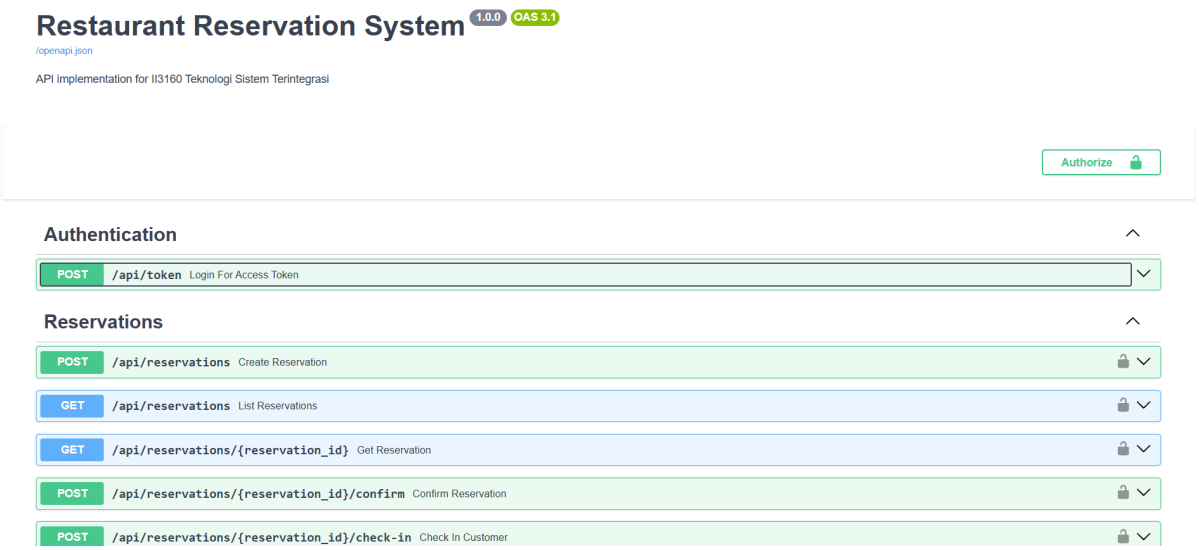
- Lingkungan Lokal: http://localhost:8000/docs
- Lingkungan Produksi (Live): https://restaurant-reservation-alpha-eight.vercel.app/docs

6.3.2 Akses Layanan Produksi (Live Demo)

Antarmuka ini menyediakan beberapa fitur kunci yang memudahkan pengujian dan eksplorasi API tanpa perlu menulis kode klien manual:

1. **Visualisasi Endpoint**: Seluruh endpoint dikelompokkan berdasarkan tags (misalnya: Auth, Reservations, Stats) dengan kode warna yang membedakan metode HTTP (GET biru, POST hijau).
2. **Uji Coba Interaktif**: Pengguna dapat mengirimkan permintaan HTTP langsung dari peramban dengan mengisi formulir parameter atau body JSON yang telah disediakan.

3. **Dukungan Autentikasi**: Tombol "Authorize" (ikon gembok) di pojok kanan atas memungkinkan pengguna memasukkan Access Token. Setelah terautentikasi, token akan otomatis disisipkan ke header setiap permintaan berikutnya.
4. **Skema Data (Schemas)**: Bagian bawah halaman menampilkan definisi struktur data (DTO) lengkap dengan tipe data, validasi (wajib/opsional), dan contoh nilai.



Gambar di atas menunjukkan halaman dokumentasi sistem yang telah aktif. Terlihat bahwa endpoint-endpoint kritis seperti /api/reservations telah tersedia dan dilindungi oleh mekanisme keamanan (ikon gembok terkunci), menandakan bahwa implementasi Security Layer berjalan dengan baik hingga ke lapisan dokumentasi.

BAB VII Kesimpulan dan Saran

7.1 Kesimpulan

Berdasarkan seluruh tahapan pengembangan yang telah dilakukan, mulai dari analisis domain hingga implementasi akhir dan penyebaran sistem (deployment), dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. **Transformasi Menjadi Sistem Siap Produksi (Production-Ready System)**: Pengembangan sistem telah berhasil melampaui fase prototipe dasar. Migrasi infrastruktur dari penyimpanan memori sementara ke basis data PostgreSQL yang dikelola oleh Supabase, serta penerapan arsitektur Cloud-Native menggunakan Vercel, membuktikan bahwa sistem ini siap menangani beban kerja nyata dengan skalabilitas dan reliabilitas tinggi.
2. **Jaminan Mutu Kode yang Tinggi**: Penerapan metodologi Test-Driven Development (TDD) secara disiplin terbukti efektif dalam menjaga integritas sistem. Dengan pencapaian Test Coverage sebesar 96%, risiko terjadinya regresi (bugs berulang) dapat diminimalkan secara drastis. Setiap baris logika bisnis kritis telah divalidasi oleh skenario pengujian otomatis, memberikan kepercayaan diri penuh bagi tim pengembang untuk melakukan iterasi fitur di masa depan.
3. **Otomasi Proses Pengembangan (DevOps Maturity)**: Implementasi Continuous Integration (CI) menggunakan GitHub Workflows berhasil menciptakan siklus umpan balik (feedback loop) yang cepat. Mekanisme ini secara otomatis memblokir kode yang tidak memenuhi standar kualitas atau gagal uji fungsi sebelum kode tersebut digabungkan ke cabang utama, sehingga stabilitas kode produksi (main branch) senantiasa terjaga.
4. **Keamanan dan Kepatuhan Standar**: Sistem telah memenuhi standar keamanan aplikasi modern melalui penerapan autentikasi berbasis JWT (Stateless) dan enkripsi kata sandi yang kuat (Bcrypt). Perlindungan pada endpoint API memastikan bahwa data sensitif pelanggan dan operasional restoran terlindungi dari akses yang tidak sah.

7.2 Saran

Meskipun sistem ini telah mencapai target fungsionalitas dan kualitas yang ditetapkan, terdapat beberapa peluang pengembangan lebih lanjut untuk meningkatkan nilai tambah sistem di masa depan:

1. **Integrasi Gerbang Pembayaran (Payment Gateway)**: Saat ini, status pembayaran masih dikelola secara manual atau simulasi. Pengembangan selanjutnya disarankan mengintegrasikan layanan pembayaran nyata (seperti Midtrans, Stripe, atau Xendit) untuk memproses pembayaran deposit secara otomatis dan memperbarui status reservasi secara real-time.

2. **Notifikasi Pengguna (Notification Service)**: Menambahkan fitur notifikasi otomatis melalui Email atau WhatsApp (misalnya menggunakan Twilio atau SendGrid) untuk memberikan konfirmasi reservasi dan pengingat jadwal kepada pelanggan, sehingga dapat mengurangi tingkat ketidakhadiran (no-show).
3. **Pengembangan Antarmuka Pengguna (Frontend)**: Membangun aplikasi antarmuka berbasis web (React/Vue.js) atau seluler (Flutter/React Native) yang terhubung dengan API ini. Hal ini akan memberikan pengalaman pengguna (User Experience) yang lebih intuitif bagi pelanggan maupun staf restoran dibandingkan berinteraksi langsung melalui Swagger UI.
4. **Implementasi Caching (Redis)**: Untuk meningkatkan performa pada skenario beban tinggi, implementasi lapisan caching menggunakan Redis dapat dipertimbangkan, terutama untuk endpoint yang sering diakses namun jarang berubah data dasarnya, seperti daftar menu atau konfigurasi meja restoran.

Daftar Pustaka

Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

Vernon, V. (2013). Implementing Domain-Driven Design. Addison-Wesley Professional.

Percival, H., & Gregory, B. (2020). Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly Media.

FastAPI. (2024). FastAPI Documentation: Modern, Fast (High-Performance) Web Framework for Building APIs with Python. Diakses dari https://fastapi.tiangolo.com/

SQLAlchemy. (2024). SQLAlchemy 2.0 Documentation: The Python SQL Toolkit and Object Relational Mapper. Diakses dari https://www.sqlalchemy.org/

Pydantic. (2024). Pydantic Documentation: Data Validation using Python Type Hints. Diakses dari https://docs.pydantic.dev/

Docker. (2024). Docker Documentation: Get Started with Docker. Diakses dari https://docs.docker.com/

GitHub. (2024). GitHub Actions Documentation: Automate your workflow from idea to production. Diakses dari https://docs.github.com/en/actions

Pytest. (2024). Pytest Documentation: Helps you write better programs. Diakses dari https://docs.pytest.org/

PostgreSQL Global Development Group. (2024). PostgreSQL 16 Documentation. Diakses dari https://www.postgresql.org/docs/

Supabase. (2024). Supabase Documentation: The Open Source Firebase Alternative. Diakses dari https://supabase.com/docs

Internet Engineering Task Force (IETF). (2015). RFC 7519: JSON Web Token (JWT). Diakses dari https://tools.ietf.org/html/rfc7519

Fowler, M. (2006). Continuous Integration. Diakses dari https://martinfowler.com/articles/continuousIntegration.html

Vercel. (2024). Vercel Documentation: Develop. Preview. Ship. Diakses dari https://vercel.com/docs

Lampiran

GitHub Repository: https://github.com/rasyidrizky/Restaurant-Reservation

Riwayat CI/CD (GitHub Actions): https://github.com/rasyidrizky/Restaurant-Reservation/actions

Base URL API: https://restaurant-reservation-alpha-eight.vercel.app

Dokumentasi Interaktif (Swagger UI): https://restaurant-reservation-alpha-eight.vercel.app/docs

Dokumentasi Alternatif (Redoc): https://restaurant-reservation-alpha-eight.vercel.app/redoc

Video Demo:https://bit.ly/18223114-TubesTST-Demo

Akun Pengujian:

- Username: admin
- Password: password123