

# **Tugas Implementasi Lanjutan**

## **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

<b>Daftar Isi.....</b>	<b>2</b>
<b>BAB I Pendahuluan.....</b>	<b>3</b>
1.1 Deskripsi Dokumen.....	3
1.2 Tujuan Penulisan Dokumen.....	3
1.3 Ruang Lingkup.....	4
1.4 Definisi dan Istilah.....	6
<b>BAB II Pemetaan Model Domain ke Kode.....</b>	<b>8</b>
2.1 Ringkasan Model Domain.....	8
2.2 Pemetaan Komponen Domain ke Kode.....	9
Tabel Pemetaan Komponen Domain ke Kode.....	9
2.3 Implementasi Endpoint.....	10
Tabel Implementasi Endpoint.....	10
<b>BAB III Desain dan Implementasi API (FastAPI).....</b>	<b>12</b>
3.1 Arsitektur Aplikasi.....	12
<b>BAB IV Pengujian API.....</b>	<b>15</b>
4.1 POST /api/token.....	15
4.2 POST /api/reservations.....	16
4.3 GET /api/reservations/{reservation_id}.....	18
4.4 POST /api/reservations/{reservation_id}/confirm.....	20
4.5 POST /api/reservations/{reservation_id}/assign-table.....	21
4.6 POST /api/reservations/{reservation_id}/cancel.....	23
<b>BAB V Kesimpulan.....</b>	<b>26</b>
<b>Daftar Pustaka.....</b>	<b>27</b>
<b>Lampiran.....</b>	<b>28</b>

# BAB I Pendahuluan

## 1.1 Deskripsi Dokumen

Dokumen ini berisi proses penerjemahan model domain, khususnya Reservation Management Aggregate, ke dalam bentuk implementasi perangkat lunak menggunakan kerangka kerja FastAPI. Dokumen ini menjembatani hasil perancangan domain pada tahap sebelumnya (analisis subdomain, bounded context, aggregate, dan class diagram) dengan tahap implementasi teknis yang diwujudkan dalam bentuk API operasional.

Secara khusus, dokumen ini menjelaskan bagaimana komponen-komponen domain seperti Entity, Value Object, Aggregate Root, dan Domain Event direpresentasikan dalam kode Python melalui pendekatan tactical design dari Domain-Driven Design (DDD).

Sebagai implementasi lanjutan, dokumen ini kini diperluas untuk mencakup aspek keamanan dan portabilitas sistem. Pemaparan meliputi penambahan mekanisme autentikasi berbasis JWT (JSON Web Token) untuk melindungi akses API, serta penerapan teknologi kontainer Docker untuk memastikan aplikasi dapat dijalankan secara konsisten di berbagai lingkungan. Selain itu, dilakukan juga penyempurnaan pada kode domain untuk merepresentasikan PaymentDetail dan DomainEvents secara eksplisit sesuai desain taktis.

Dokumen ini juga menyertakan instruksi teknis mengenai cara menjalankan aplikasi, baik secara manual maupun menggunakan Docker, dependensi yang diperlukan, serta tautan menuju repositori Git yang memuat seluruh riwayat kode implementasi. Dengan demikian, dokumen ini berfungsi sebagai panduan komprehensif yang menghubungkan analisis domain konseptual dengan implementasi perangkat lunak yang aman, modular, dan siap didistribusikan.

## 1.2 Tujuan Penulisan Dokumen

Dokumen ini disusun untuk menjelaskan secara sistematis proses implementasi model domain ke dalam bentuk layanan perangkat lunak berbasis API menggunakan pendekatan Domain-Driven Design (DDD) dan framework FastAPI, serta pengembangannya lebih lanjut untuk memenuhi standar keamanan dan deployment modern.

Adapun tujuan utama dari dokumen ini meliputi:

- 1. Menerjemahkan model domain ke dalam kode program**

Menyajikan bagaimana konsep-konsep domain seperti Entity, Value Object, Aggregate, dan Domain Event direalisasikan dalam kode Python. Implementasi ini memastikan bahwa

struktur dan perilaku domain tetap konsisten dengan aturan bisnis yang telah dirumuskan pada tahap analisis.

**2. Membangun API fungsional yang aman dan terintegrasi**

Mengembangkan serangkaian endpoint utama pada konteks Reservation Management yang tidak hanya mencakup operasi reservasi inti, tetapi juga dilindungi oleh mekanisme autentikasi berbasis JWT (JSON Web Token). Hal ini bertujuan untuk memastikan bahwa akses terhadap data sensitif dan operasi kritis hanya dapat dilakukan oleh pengguna yang terotorisasi.

**3. Menyediakan lingkungan eksekusi yang konsisten (Containerization)**

Menjelaskan proses pengemasan aplikasi menggunakan Docker, sehingga sistem dapat dijalankan dengan mudah dan konsisten di berbagai lingkungan tanpa kendala dependensi (dependency hell).

**4. Menyediakan dokumentasi struktur dan arsitektur aplikasi**

Menjelaskan desain arsitektur backend, struktur folder, dependensi internal, serta pemisahan tanggung jawab antar layer, termasuk penambahan modul keamanan (security module) dan konfigurasi infrastruktur. Dokumentasi ini memudahkan pengembangan, pemeliharaan, dan perluasan sistem ke depannya.

**5. Menyediakan bukti implementasi melalui repositori Git**

Mengumpulkan dan menyajikan tautan repositori Git sebagai bukti nyata implementasi kode, termasuk riwayat commit yang menunjukkan pembangunan API awal, integrasi autentikasi, serta penambahan konfigurasi Docker.

Dengan adanya tujuan-tujuan ini, dokumen ini diharapkan mampu menjadi referensi lengkap yang menghubungkan aspek konseptual domain model dengan implementasi teknis yang dapat dijalankan dan dievaluasi.

## **1.3 Ruang Lingkup**

Ruang lingkup dokumen ini difokuskan pada proses implementasi teknis dari model domain Reservation Management Context ke dalam bentuk layanan API berbasis FastAPI yang aman dan siap didistribusikan.

Lingkup pembahasan tidak mencakup seluruh fungsi operasional restoran, tetapi dibatasi pada konteks inti (core context) dan infrastruktur pendukung yang diperlukan. Adapun ruang lingkup yang dicakup dalam dokumen ini meliputi:

**1. Penerjemahan Model Domain (Aggregate) ke Implementasi Kode**

Dokumen ini memodelkan Reservation Aggregate beserta entitas, value object, dan domain event yang relevan. Lingkup ini kini diperluas dengan implementasi eksplisit untuk kelas PaymentDetail dan DomainEvents di lapisan domain, untuk merepresentasikan struktur data pembayaran dan kejadian bisnis secara lebih konkret sesuai desain taktis.

**2. Pengembangan API Inti dan Mekanisme Keamanan**

Selain membangun operasi dasar reservasi (pembuatan, validasi, konfirmasi), lingkup tugas ini mencakup implementasi sistem autentikasi sederhana menggunakan JWT (JSON Web Token). Hal ini meliputi pembuatan endpoint login (/api/token) serta penerapan dependency injection untuk memproteksi endpoint reservasi agar hanya dapat diakses oleh pengguna yang terautentikasi.

**3. Standarisasi Lingkungan Eksekusi dengan Docker**

Penerapan teknologi kontainerisasi menggunakan Docker untuk mengemas aplikasi beserta seluruh dependensinya (Dockerfile). Hal ini dilakukan untuk memastikan aplikasi dapat dijalankan secara konsisten (portabel) tanpa bergantung pada konfigurasi lokal pengembang, menggantikan metode eksekusi manual sebelumnya.

**4. Lingkup Operasional Terbatas pada In-Memory Repository**

Penyimpanan data tetap dilakukan menggunakan in-memory repository (struktur data Dictionary) untuk mempermudah evaluasi logika domain dan autentikasi tanpa kompleksitas konfigurasi basis data eksternal.

**5. Penyediaan Repositori Git sebagai Bukti Implementasi**

Hasil implementasi disertakan dalam bentuk tautan repositori Git yang memuat struktur kode lengkap, termasuk riwayat commit yang mencakup fitur autentikasi, konfigurasi Docker, dan penyempurnaan model domain.

Dengan batasan ruang lingkup tersebut, dokumen ini menyediakan fokus pada penerapan DDD Tactical Design secara nyata dalam pembangunan API inti.

## 1.4 Definisi dan Istilah

Bagian ini menyajikan istilah-istilah penting yang digunakan dalam dokumen ini. Tujuannya adalah untuk memastikan kesamaan pemahaman terkait konsep-konsep teknis dan domain yang menjadi dasar implementasi 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 in-memory 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 Token)	Standar terbuka (RFC 7519) untuk mentransmisikan informasi secara 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.

## **BAB II Pemetaan Model Domain ke Kode**

### **2.1 Ringkasan Model Domain**

Model domain dalam proyek ini berfokus pada Reservation Management Context sebagai core domain dalam keseluruhan sistem. Konteks ini dipilih karena memiliki peran strategis dalam mengatur seluruh proses reservasi pelanggan, yang menjadi pusat operasional restoran dan sumber utama nilai bisnis. Model domain yang dirancang sebelumnya menetapkan struktur logis berupa Entity, Value Object, Aggregate, serta Domain Event yang mencerminkan aturan bisnis inti dalam proses reservasi.

Pada tingkat konseptual, proses reservasi melibatkan beberapa komponen penting:

#### **1. Reservation (Aggregate Root)**

Merupakan entitas utama yang mewakili satu pemesanan meja oleh pelanggan. Reservation bertanggung jawab menjaga konsistensi seluruh aturan bisnis reservasi, mulai dari validasi waktu, alokasi meja, kebijakan pembatalan, hingga pelacakan status dan pembayaran deposit.

#### **2. TableAssignment (Entity)**

Entitas yang mewakili alokasi satu atau lebih meja untuk sebuah reservasi. Entitas ini berada di dalam Reservation Aggregate karena alokasi meja merupakan bagian dari aturan konsistensi reservasi.

#### **3. ReservationHistory (Entity)**

Entitas yang menampung riwayat pemesanan meja oleh pelanggan. ReservationHistory bertanggung jawab mencatat waktu pertama kali reservasi dibuat, aksi yang dilakukan, dan catatan khusus.

#### **4. ReservationStatus, ContactInfo, ReservationTime, CancellationReason, ReservationPolicy, dan PaymentDetail (Value Object)**

Kumpulan objek nilai yang digunakan untuk memperkuat makna bisnis atribut-atribut reservasi, seperti waktu, kebijakan durasi, aturan pembatalan, dan detail pembayaran deposit. Value object ini bersifat immutabel untuk menjaga konsistensi data.

#### **5. Domain Events**

Implementasi konkret dari kejadian bisnis (business events) seperti ReservationCreated, ReservationConfirmed, dan ReservationCancelled. Event ini kini direpresentasikan sebagai kelas objek yang dibuat dan disimpan sementara di dalam Aggregate Root saat terjadi

perubahan status, siap untuk diproses lebih lanjut atau dikirim ke sistem eksternal di masa depan.

Model domain ini dirancang menggunakan prinsip tactical design dari DDD, yang mengutamakan keselarasan antara konsep bisnis dan implementasi teknis. Struktur ini kemudian menjadi dasar untuk menyusun kode Python yang merepresentasikan perilaku domain secara eksplisit melalui aggregate logic, method behavior, dan domain invariants.

Pada bagian selanjutnya, setiap komponen dalam model domain akan dipetakan secara langsung ke implementasi kode di dalam struktur program FastAPI, sehingga hubungan antara model konseptual dan model operasional dapat terlihat dengan jelas.

## 2.2 Pemetaan Komponen Domain ke Kode

Pada tahap implementasi lanjutan ini, seluruh komponen domain diterjemahkan ke dalam kode Python melalui struktur folder yang telah ditetapkan pada konteks Reservation Management. Pemetaan ini kini lebih lengkap dibandingkan tahap awal, di mana komponen yang sebelumnya hanya bersifat konseptual (seperti pembayaran dan event) kini telah memiliki representasi kode yang nyata untuk mendukung integritas data dan potensi pengembangan fitur di masa depan. Tabel berikut merangkum seluruh komponen domain dan lokasi implementasinya:

**Tabel Pemetaan Komponen Domain ke Kode**

Komponen Domain	Implementasi Kode	Lokasi	Keterangan
Reservation (Aggregate Root)	class Reservation	src/domain/models.py	Kelas utama yang mengontrol konsistensi data. Memiliki daftar Entities dan Value Objects di dalamnya.
TableAssignment (Entity)	class TableAssignment	src/domain/models.py	Entitas anak yang hanya bisa diakses melalui Reservation. Memiliki ID unik (assignment_id).
ReservationHistory (Entity)	class ReservationHistory	src/domain/models.py	Mencatat log perubahan status (Audit Trail) di dalam agregat reservasi.
ReservationStatus	class ReservationStatus(str, Enum)	src/domain/value_objects.py	Menggunakan Enum untuk membatasi status valid (PENDING, CONFIRMED, dll).
ContactInfo	class ContactInfo(BaseModel)	src/domain/value_objects.py	Mengelompokkan Nama, HP, dan Email. Bersifat Immutable

			(menggunakan Pydantic).
ReservationTime	class ReservationTime(BaseModel)	src/domain/value_objects.py	Mengkapsulasi logika waktu mulai dan durasi.
ReservationPolicy	class ReservationPolicy(BaseModel)	src/domain/value_objects.py	Menyimpan aturan seperti auto_cancel_minutes.
CancellationReason	class CancellationReason(BaseModel)	src/domain/value_objects.py	Objek nilai untuk menampung alasan saat pembatalan terjadi.
Domain Event	class ReservationCreated, class ReservationConfirmed, class ReservationCancelled	src/domain/events.py	Objek immutable yang merepresentasikan kejadian bisnis penting (perubahan status).
confirmReservation()	def confirm_reservation(self)	class Reservation	Memvalidasi status saat ini sebelum mengubahnya menjadi CONFIRMED.
assignTable()	def assign_table(self, ...)	class Reservation	Membuat objek TableAssignment dan menghubungkannya ke reservasi.
cancelReservation()	def cancel_reservation(self, reason)	class Reservation	Menerima CancellationReason dan mengubah status menjadi CANCELLED.
recordHistory()	def _record_history(self, ...)	class Reservation	Method internal yang otomatis dipanggil saat status berubah untuk mencatat log.

## 2.3 Implementasi Endpoint

Implementasi API pada aplikasi ini dibangun berdasarkan model domain Reservation Management dan kini telah dilengkapi dengan lapisan keamanan (security layer). Endpoint-endpoint diorganisasi ke dalam dua modul utama, yaitu modul autentikasi dan modul reservasi. Setiap endpoint reservasi kini dilindungi oleh dependency injection yang memvalidasi keberadaan token akses valid sebelum meneruskan permintaan ke logika domain. Bagian ini menyajikan daftar lengkap endpoint dengan metode, URL, dan fungsinya.

**Tabel Implementasi Endpoint**

Metode HTTP	Endpoint URL	Fungsi / Deskripsi
-------------	--------------	--------------------

POST	/api/token	Login & Autentikasi. Menerima kredensial pengguna (username & password) dalam format form-data, memvalidasi kecocokan hash, dan mengembalikan Access Token (JWT).
POST	/api/reservations	<b>(Protected)</b> Membuat Reservasi Baru. Menerima data pelanggan dan waktu, mengembalikan ID reservasi baru dengan status PENDING.
GET	/api/reservations/{reservation_id}	<b>(Protected)</b> Melihat Detail Reservasi. Mengambil informasi lengkap reservasi berdasarkan ID (termasuk status dan meja jika ada).
POST	/api/reservations/{reservation_id}/confirm	<b>(Protected)</b> Konfirmasi Reservasi. Mengubah status reservasi dari PENDING menjadi CONFIRMED.
POST	/api/reservations/{reservation_id}/assign-table	<b>(Protected)</b> Alokasi Meja. Menetapkan nomor/area meja untuk reservasi tertentu.
POST	/api/reservations/{reservation_id}/cancel	<b>(Protected)</b> Batalkan Reservasi. Membatalkan reservasi dengan menyertakan alasan pembatalan.

Catatan:

- **Format Data:** Semua endpoint menerima dan mengirim data dalam format JSON.
- **Keamanan:** Endpoint bertanda (Protected) mewajibkan klien mengirimkan HTTP Header: Authorization: Bearer <access\_token>.
- **Parameter:** {reservation\_id} pada URL adalah UUID unik yang didapatkan saat membuat reservasi pertama kali.

## BAB III Desain dan Implementasi API (FastAPI)

### 3.1 Arsitektur Aplikasi

Arsitektur aplikasi yang digunakan pada implementasi ini dirancang berdasarkan prinsip Domain-Driven Design (DDD) dengan pendekatan Layered Architecture yang disederhanakan. Struktur ini bertujuan untuk memisahkan logika bisnis murni (Domain) dari detail teknis framework (FastAPI), keamanan, dan infrastruktur. Hal ini memastikan bahwa aturan bisnis tetap menjadi pusat dari pengembangan perangkat lunak.

Aplikasi ini kini terbagi menjadi lima komponen logis utama: Application Entry Point, Interface Layer, Core Security Layer, Domain Layer, dan Infrastructure Layer (Simulated). Berikut adalah rincian tanggung jawab setiap lapisan:

1. Application Entry Point (main.py)

File ini berfungsi sebagai gerbang utama aplikasi. Tanggung jawab utamanya meliputi:

- Inisialisasi Framework: Mengaktifkan instance aplikasi FastAPI.
- Konfigurasi Routing: Mendaftarkan router dari lapisan API, termasuk router autentikasi (/api/token) dan router reservasi.
- Server Bootstrapping: Menjalankan server aplikasi menggunakan Uvicorn (atau melalui Docker entrypoint).

2. Interface Layer (src/api/ & src/schemas/)

Lapisan ini bertanggung jawab menangani interaksi dengan dunia luar (klien HTTP). Lapisan ini bertindak sebagai penerjemah antara permintaan eksternal dan logika internal aplikasi.

- Auth Routes (src/api/auth.py):
  - Menangani proses login.
  - Menerima kredensial pengguna, memverifikasi hash password, dan menerbitkan Access Token (JWT).
- API Routes (src/api/routes.py):
  - Mendefinisikan endpoint HTTP (GET, POST) sesuai spesifikasi REST API.
  - Menerima input dari klien dan meneruskannya ke model domain.
  - Menangani respon HTTP dan kode status (misalnya: 200 OK, 404 Not Found, 422 Unprocessable Entity).
  - Pada implementasi dasar ini, routes juga bertindak sebagai pengendali alur aplikasi sederhana.

- Menggunakan mekanisme Dependency Injection (Depends) untuk memvalidasi token sebelum meneruskan permintaan ke domain.
- Schemas / DTO (src/schemas/reservation.py):
  - Berisi model Pydantic untuk Data Transfer Objects (DTO).
  - Melakukan validasi format data masuk (input sanitization) sebelum data menyentuh lapisan domain (misalnya: memastikan format UUID valid).
  - Mendefinisikan struktur data keluar (output serialization) untuk dikirimkan kembali ke klien.
- 3. Core Security Layer (src/core/)

Lapisan khusus yang menangani logika keamanan aplikasi yang bersifat lintas-sektoral (cross-cutting concern).

  - Security Logic (src/core/security.py):
    - Melakukan hashing dan verifikasi password menggunakan algoritma Bcrypt.
    - Membuat (encode) dan membaca (decode) JSON Web Token (JWT).
    - Menyediakan dependensi `get_current_user` untuk memproteksi endpoint.
- 4. Domain Layer (src/domain/)

Layer ini adalah inti dari perangkat lunak yang merepresentasikan Core Domain yaitu Reservation Management. Lapisan ini murni berisi logika Python tanpa ketergantungan pada framework eksternal atau basis data .

  - Aggregate & Entities (src/domain/models.py):
    - Berisi implementasi Reservation sebagai Aggregate Root yang menjaga konsistensi data transaksi.
    - Menyimpan logika perilaku bisnis (behavior), seperti metode `confirm_reservation()`, `assign_table()`, dan `cancel_reservation()`.
    - Mengelola entitas pendukung seperti `TableAssignment` dan `ReservationHistory`.
  - Value Objects (src/domain/value\_objects.py):
    - Mendefinisikan objek nilai yang bersifat immutable (tidak berubah) untuk merepresentasikan konsep deskriptif seperti `ReservationTime`, `ContactInfo`, dan `ReservationStatus`.
    - Mengandung aturan validasi intrinsik (misalnya: durasi reservasi default).
  - Domain Events (src/domain/events.py):
    - Mendefinisikan objek kejadian bisnis (business events) yang bersifat immutable, seperti `ReservationCreated` dan `ReservationConfirmed`.

- Merepresentasikan perubahan status penting (state change) yang terjadi di dalam Aggregate Root untuk keperluan audit trail atau pemrosesan lanjutan.

#### 5. Infrastructure Layer (src/infrastructure/ & Dockerfile)

Dalam implementasi prototipe API dasar ini, lapisan infrastruktur disimulasikan di dalam memori untuk memudahkan pengujian tanpa dependensi database fisik yang kompleks.

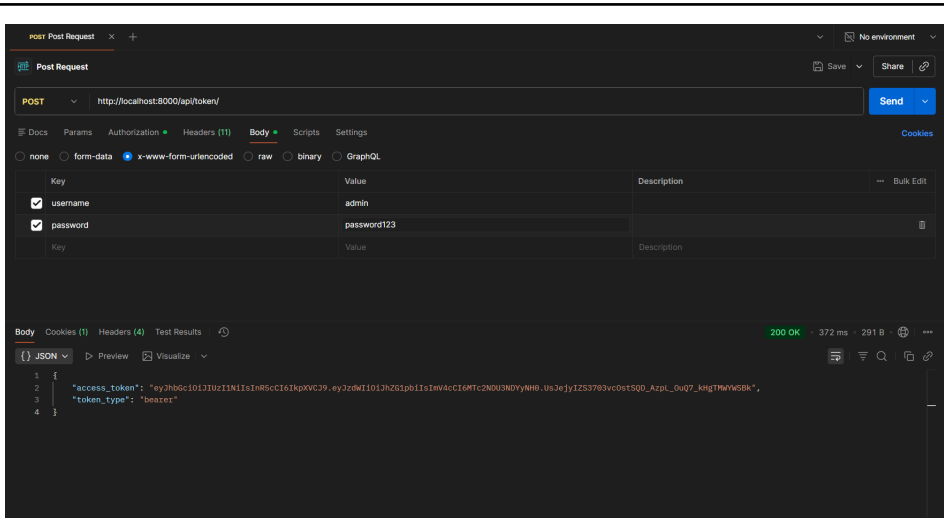
- In-Memory Repository:
  - Diwakili oleh variabel `reservation_db` (Dictionary) di dalam `routes.py`.
  - Bertanggung jawab untuk menyimpan dan mengambil kembali objek Reservation selama aplikasi berjalan.
- Containerization (Docker):
  - Konfigurasi Dockerfile yang membungkus aplikasi beserta seluruh dependensinya (`requirements.txt`) ke dalam image terisolasi, memastikan aplikasi dapat berjalan konsisten di lingkungan mana pun.

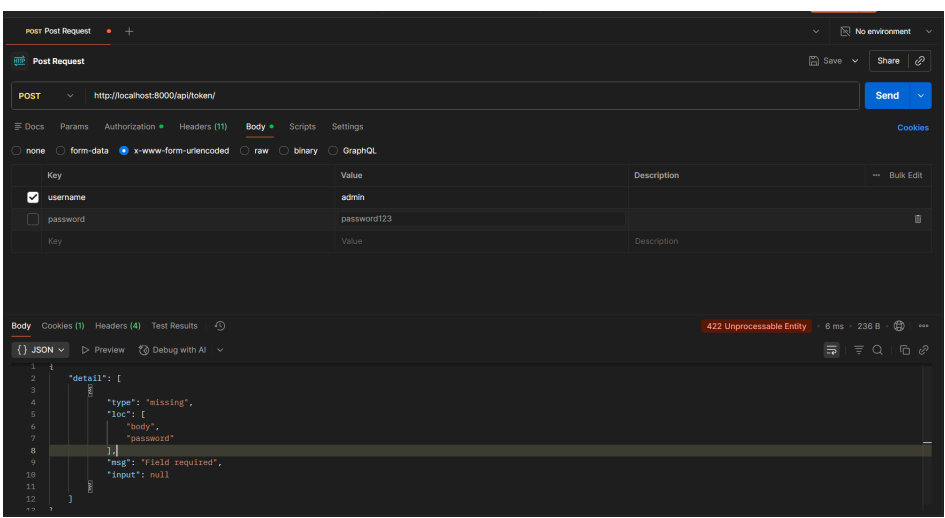
Struktur ini memastikan bahwa Domain Layer tetap bersih dan tidak terpolusi oleh detail teknis seperti HTTP request atau query database, sehingga memudahkan pengujian unit dan pemeliharaan logika bisnis di masa depan.

## BAB IV Pengujian API

### 4.1 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.

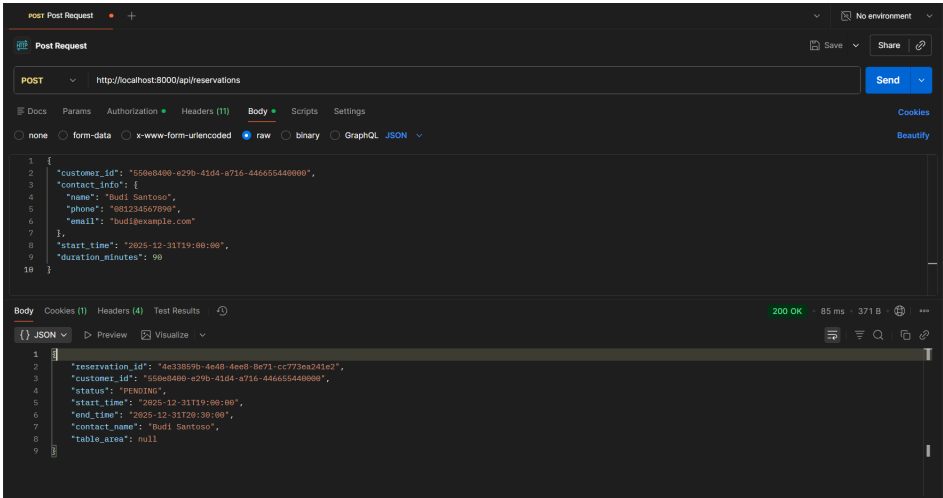
URL	/api/token	
Method	POST	
Content-Type	application/x-www-form-urlencoded	
Parameter		
Nama	Tipe	Keterangan
username	string	Nama pengguna.
password	string	Kata sandi pengguna.
Contoh Request (cURL)		
curl -X POST "http://localhost:8000/api/token" \		
-H "Content-Type: application/x-www-form-urlencoded" \		
-d "username=admin&password=password123"		
Response		
Code	Value	
200 (Successful Response)	"string"	
		
422 (Validation	{	

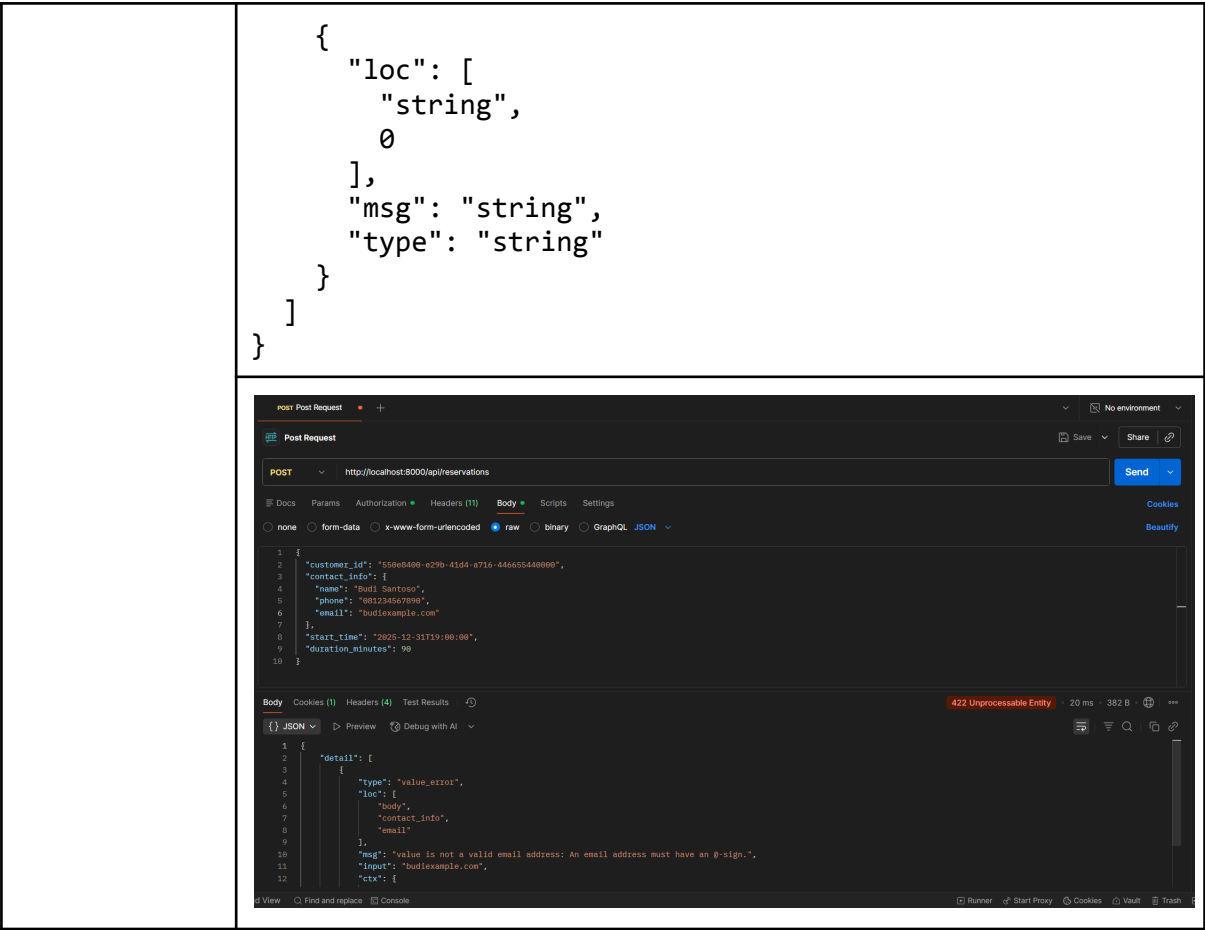
Error)	<pre>"detail": [   {     "loc": [       "string",       0     ],     "msg": "string",     "type": "string"   } ]</pre> 
--------	--

4.2 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		
<pre>{   "customer_id": "string",   "contact_info": {     "name": "string",     "phone": "string",     "email": "user@example.com"   },   "start_time": "2025-12-01T07:26:45.853Z",   "duration_minutes": 90 }</pre>		
Response		
Code	Value	
200 (Successful Response)	<pre>{   "reservation_id": "string",   "customer_id": "string",   "status": "PENDING",   "start_time": "2025-12-01T07:26:45.856Z",   "end_time": "2025-12-01T07:26:45.856Z",   "contact_name": "string",   "table_area": "string" }</pre> 	
422 (Validation Error)	<pre>{   "detail": [</pre>	

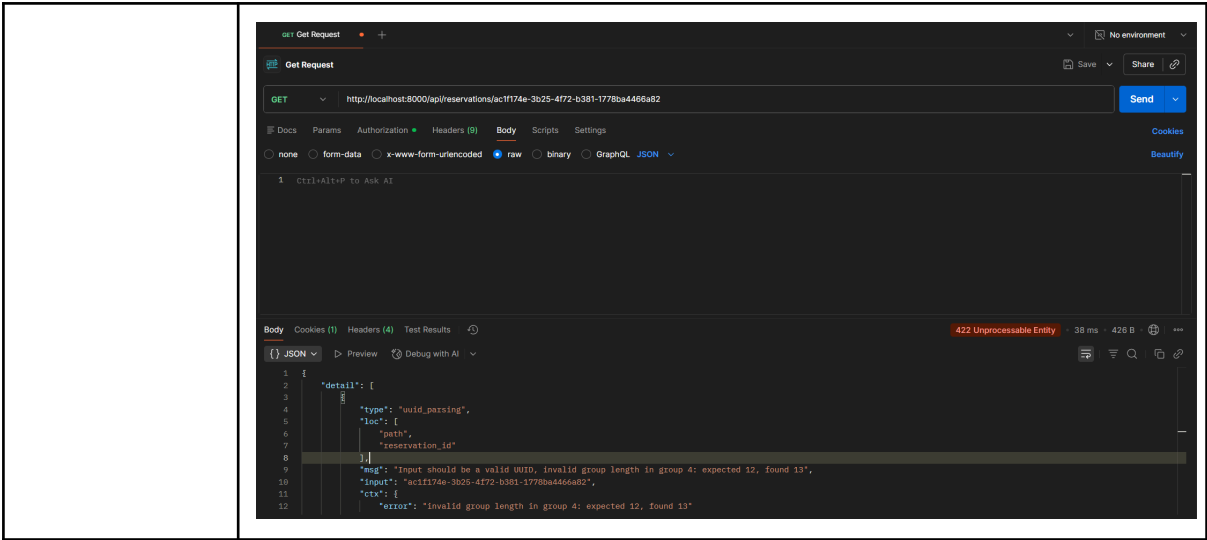


4.3 GET /api/reservations/{reservation\_id}

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		

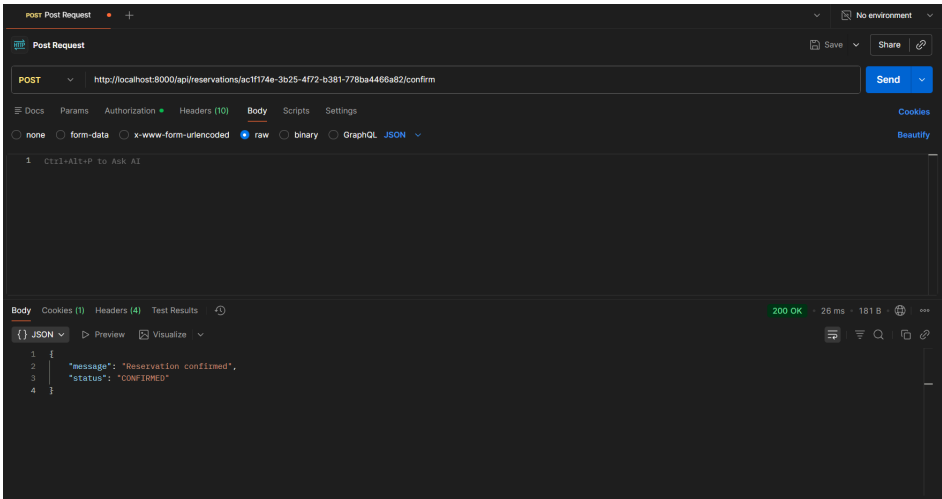
Code	Value
200 (Successful Response)	<div><pre>{   "reservation_id": "string",   "customer_id": "string",   "status": "PENDING",   "start_time": "2025-12-01T07:26:45.860Z",   "end_time": "2025-12-01T07:26:45.860Z",   "contact_name": "string",   "table_area": "string" }</pre></div> <div></div>
422 (Validation Error)	<pre>{   "detail": [     {       "loc": [         "string",         0       ],       "msg": "string",       "type": "string"     }   ] }</pre>



#### 4.4 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.

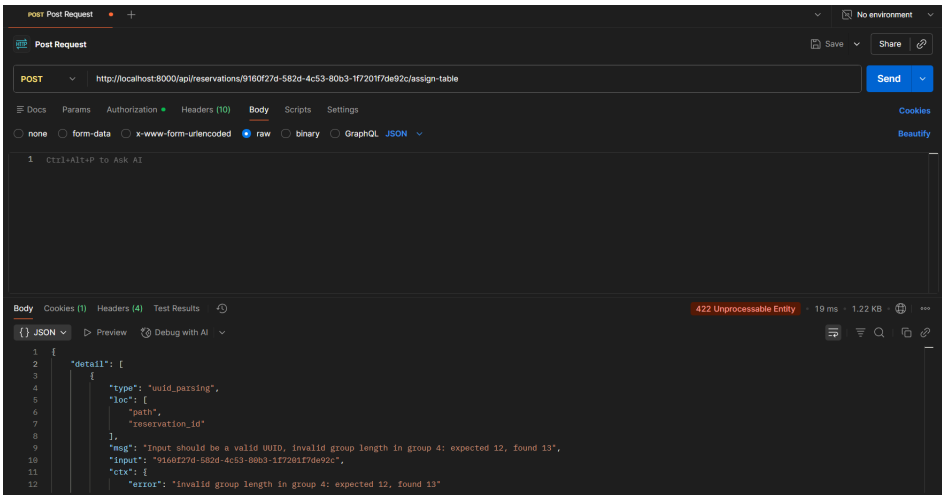
URL	/api/reservations/{reservation_id}/confirm	
Method	POST	
Header		
Key	Authorization	
Value	Bearer <access_token>	
Parameter		
Nama	Tipe	Keterangan
reservation_id	UUID	ID reservasi yang ingin dikonfirmasi.
Response		
Code	Value	
200 (Successful Response)	"string"	

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

#### 4.5 POST /api/reservations/{reservation\_id}/assign-table

Endpoint ini digunakan untuk menetapkan meja spesifik ke dalam reservasi yang sudah ada. Endpoint ini bersifat Protected, yang berarti klien harus menyertakan token akses yang valid pada header permintaan untuk membuktikan otorisasi.

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		
{ "table_id": "string", "capacity": 0, "area": "string" }		
Response		
Code	Value	
200 (Successful Response)	"string"	

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

4.6 POST /api/reservations/{reservation\_id}/cancel

Endpoint ini digunakan untuk membatalkan reservasi yang aktif. Karena dampak operasionalnya, endpoint ini bersifat Protected dan mewajibkan klien menyertakan token akses yang valid. Selain itu, sistem juga memvalidasi logika bisnis, misalnya mencegah pembatalan untuk reservasi yang statusnya sudah COMPLETED.

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
reason_code	String	Kode singkat alasan (misal: "CUSTOMER_REQUEST", "NO_SHOW").
description	String	Penjelasan detail mengenai alasan pembatalan.
Request Body		
{ "reason_code": "string", "description": "string" }		
Response		
Code	Value	
200 (Successful Response)	"string"	

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

## **BAB V Kesimpulan**

Dokumen ini telah menjelaskan proses perancangan dan implementasi sistem berbasis Domain-Driven Design (DDD) untuk membangun layanan inti Reservation Management pada Sistem Reservasi & Manajemen Restoran Terintegrasi. Pendekatan ini memastikan bahwa logika bisnis kompleks pada domain reservasi dapat dimodelkan dengan baik melalui struktur domain yang jelas, konsisten, dan mudah dikembangkan.

Pada tahap implementasi lanjutan ini, sistem telah mengalami peningkatan signifikan dari sisi keamanan dan operasional. Implementasi mekanisme autentikasi menggunakan JWT (JSON Web Token) dan enkripsi password (Bcrypt) berhasil ditambahkan untuk melindungi endpoint reservasi, memastikan bahwa operasi sensitif hanya dapat diakses oleh pengguna yang terotorisasi. Hal ini mengubah sistem dari sekadar prototipe fungsional menjadi layanan yang aman (secure service).

Selain itu, aspek deployment dan portabilitas sistem telah distandarisasi melalui penerapan teknologi kontainerisasi Docker. Dengan adanya Dockerfile dan konfigurasi infrastruktur, aplikasi kini dapat dijalankan secara konsisten di berbagai lingkungan tanpa kendala dependensi. Penyempurnaan juga dilakukan pada Domain Layer dengan mengimplementasikan objek PaymentDetail dan DomainEvents secara konkret, menegaskan komitmen terhadap desain taktis yang telah dirancang sebelumnya.

Meskipun penyimpanan data masih menggunakan in-memory repository, fondasi arsitektur yang telah dibangun, meliputi pemisahan layer, modul keamanan, dan kontainerisasi, memberikan dasar yang sangat kokoh. Sistem ini kini siap untuk melangkah ke tahap pengembangan selanjutnya, yaitu integrasi dengan basis data persisten (seperti PostgreSQL) dan penerapan mekanisme event-driven secara asinkron untuk komunikasi antar-layanan.

## 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.

Richardson, C. (2018). Microservices Patterns: With Examples in Java. Manning Publications.

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

Docker Documentation. (2024). Get Started with Docker: Containerization for All. Retrieved from <https://docs.docker.com/get-started/>

Internet Engineering Task Force (IETF). (2015). RFC 7519: JSON Web Token (JWT). Retrieved from <https://tools.ietf.org/html/rfc7519>

Internet Engineering Task Force (IETF). (2012). RFC 6749: The OAuth 2.0 Authorization Framework. Retrieved from <https://tools.ietf.org/html/rfc6749>

Python Software Foundation. (2024). Python 3.11 Documentation. Retrieved from <https://docs.python.org/3/>

Passlib Documentation. (2024). Password Hashing Library for Python. Retrieved from <https://passlib.readthedocs.io/>

Stripe API Documentation. (2024). Stripe Payments API Reference. Retrieved from <https://stripe.com/docs/api>

Twilio API Documentation. (2024). Twilio Messaging API Reference. Retrieved from <https://www.twilio.com/docs/messaging/api>

Fowler, M. (2004). Patterns of Enterprise Application Architecture. Addison-Wesley.

Microsoft Azure Architecture Center. (2024). Domain-Driven Design Fundamentals. Retrieved from <https://learn.microsoft.com/en-us/azure/architecture/microservices/model/domain-analysis>

OpenAPI Initiative. (2024). OpenAPI Specification v3.1. Retrieved from <https://spec.openapis.org/oas/latest.html>

## **Lampiran**

Link Repository: <https://github.com/rasyidrizky/Restaurant-Reservation>