PROPOSAL TUGAS BESAR

MATA KULIAH INTEGRASI APLIKASI ENTERPRISE

JUDUL:

SISTEM MANAJEMEN RESTORAN BERBASIS MICROSERVICES DENGAN ARSITEKTUR TERINTEGRASI MENGGUNAKAN GRAPHQL DAN DOCKER

Integrasi Internal (Auth–Course–Enrollment) dan Integrasi Eksternal (Payment Gateway Lintas Kelompok)

RINGKASAN

Proyek Anugerah Resto merupakan sistem manajemen restoran berbasis microservices yang menggunakan GraphQL sebagai mekanisme komunikasi antar layanan dan Docker untuk deployment terisolasi. Sistem ini terdiri dari beberapa service independen, yaitu Kitchen Service, Inventory Service, User Service, dan Order Service, yang masing-masing memiliki tanggung jawab dan database tersendiri.

Order Service berperan sebagai pusat integrasi dengan menghubungkan proses pemesanan, validasi pengguna, pengecekan stok, serta pengiriman pesanan ke dapur. Selain integrasi internal, sistem ini juga mendukung integrasi lintas kelompok dengan Toko Sembako untuk pengadaan bahan baku melalui endpoint GraphQL, sehingga arsitektur yang dibangun bersifat modular, terintegrasi, dan mudah dikembangkan.

1. PENDAHULUAN

1.1 Latar Belakang

Sistem manajemen restoran modern menuntut efisiensi operasional, skalabilitas, dan integrasi sistem yang baik seiring meningkatnya kompleksitas proses bisnis. Dalam konteks operasional restoran, kebutuhan utama meliputi pengelolaan pesanan pelanggan secara real-time, sinkronisasi antara dapur dan layanan pemesanan, manajemen stok bahan baku yang akurat, serta pengelolaan data pengguna dan transaksi yang konsisten.

Karena sistem bersifat terdistribusi dan melibatkan banyak proses yang saling bergantung, maka integrasi antar layanan harus dilakukan secara terstruktur dan konsisten. Oleh karena itu, proyek ini menerapkan arsitektur microservices dengan komunikasi berbasis GraphQL, di mana setiap layanan (Kitchen, Inventory, User, dan Order Service) berfokus pada domain bisnisnya masing-masing. Pendekatan ini memungkinkan pemisahan tanggung jawab yang jelas, mendukung interoperabilitas antar layanan internal maupun eksternal, serta meningkatkan fleksibilitas sistem dalam menghadapi pengembangan dan integrasi lanjutan.

1.2 Tujuan Pembelajaran (Learning Outcomes)

Tujuan pembelajaran dari pengembangan proyek Anugerah Resto ini adalah untuk membangun dan memahami penerapan arsitektur microservices dalam sistem manajemen restoran yang terintegrasi. Melalui proyek ini, mahasiswa diharapkan mampu merancang sistem modular dengan pemisahan tanggung jawab yang jelas antar layanan, serta mengimplementasikan GraphQL sebagai mekanisme komunikasi antar service.

Selain itu, proyek ini bertujuan untuk melatih kemampuan deployment berbasis container menggunakan Docker dan Docker Compose, serta membuktikan keberhasilan integrasi lintas kelompok melalui pertukaran data dan layanan antar sistem yang berbeda. Dengan demikian, mahasiswa dapat memahami konsep interoperabilitas, skalabilitas, dan integrasi sistem enterprise secara praktis.

1.3 Ruang Lingkup dan Batasan

Ruang lingkup:

- Sistem terdiri dari empat layanan internal berbasis microservices, yaitu Kitchen Service, Inventory Service, User Service, dan Order Service, di mana setiap layanan dikembangkan dan dijalankan secara independen.

- Order Service berperan sebagai pusat integrasi yang menghubungkan proses pemesanan dengan layanan dapur, inventori, dan pengguna.

- Sistem menerapkan integrasi eksternal wajib dengan sistem Toko Sembako (kelompok lain) melalui layanan GraphQL untuk pengadaan dan pengecekan stok bahan baku.

- Setiap service memiliki database terpisah sebagai penerapan prinsip *one service one database*, dengan penggunaan MySQL sesuai kebutuhan masing-masing layanan.

Batasan:

- Sistem Toko Sembako diperlakukan sebagai external dependency, sehingga fungsionalitas integrasi disesuaikan dengan kontrak API yang disediakan oleh kelompok lain.

- Implementasi sistem difokuskan pada fungsi inti manajemen restoran, dan tidak mencakup aspek non-fungsional lanjutan seperti keamanan tingkat produksi, load balancing, dan monitoring skala besar.

2. ARSITEKTUR SISTEM (OVERVIEW)

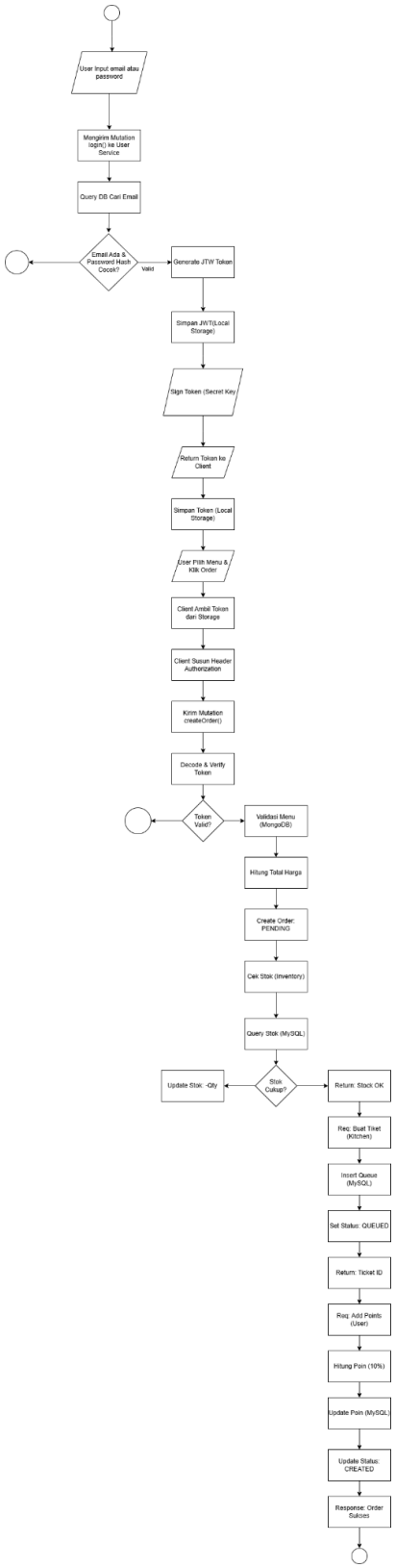
2.1 Prinsip Desain

1. **Separation of Concerns**
   Setiap layanan dalam sistem Anugerah Resto memiliki tanggung jawab domain yang jelas, dimana pengelolaan pesanan, dapur, inventori, dan pengguna dipisahkan ke dalam service yang berbeda untuk menghindari kompleksitas dan ketergantungan yang tidak perlu.
2. **Service Autonomy**
   Masing-masing microservice memiliki database tersendiri sesuai prinsip *one service one database*, sehingga setiap layanan dapat dikembangkan, diuji, dan diskalakan secara independen tanpa menimbulkan *tight coupling* antar service.
3. **Interoperability**
   Komunikasi antar layanan dilakukan menggunakan GraphQL, yang menyediakan kontrak data yang eksplisit dan terstruktur, serta memudahkan integrasi antar service internal maupun dengan sistem eksternal seperti Toko Sembako.
4. **Containerized Deployment**
   Seluruh layanan dijalankan secara terisolasi dalam container Docker dan dikelola menggunakan Docker Compose, sehingga proses deployment menjadi konsisten, terstandarisasi, dan memenuhi aspek portabilitas lingkungan.

2.2 Komponen dan Tanggung Jawab

1. **User Service (Authentication & Authorization / JWT Provider)**
   Mengelola autentikasi dan otorisasi pengguna sistem, termasuk manajemen akun staf dan pelanggan serta pengelolaan loyalty program.
2. **Order Service (Order Management & Orchestration)**
   Mengelola katalog menu, proses pemesanan, dan transaksi pelanggan, serta bertindak sebagai *orchestrator* yang mengintegrasikan Kitchen Service, Inventory Service, dan User Service.
3. **Kitchen Service (Kitchen Queue Management)**
   Bertanggung jawab atas pengelolaan antrian pesanan dapur dan pembaruan status pengerjaan makanan.
4. **Inventory Service (Inventory & Supplier Management)**
   Mengelola stok bahan baku, supplier, serta integrasi dengan sistem Toko Sembako (kelompok lain) sebagai external dependency wajib untuk pengadaan bahan baku.

2.3 Diagram FlowChart



1. Fase Autentikasi dan Keamanan (Langkah 1–9) Proses dimulai dengan interaksi pengguna pada antarmuka aplikasi (*Client*) untuk memasukkan kredensial email dan kata sandi. Aplikasi mengirimkan permintaan *mutation login* ke **User Service**, yang kemudian memverifikasi data tersebut dengan melakukan *query* ke database MySQL. Jika kredensial valid (password cocok setelah proses *hashing*), User Service akan membangkitkan token **JWT (JSON Web Token)** yang berisi identitas pengguna dan menandatanganinya dengan *secret key*. Token ini dikirimkan kembali ke *Client* dan disimpan secara lokal (*Local Storage*) sebagai "tiket akses" untuk transaksi selanjutnya, menjamin bahwa sistem beroperasi secara *stateless*.

2. Inisiasi dan Validasi Pesanan (Langkah 10–18) Setelah terautentikasi, pengguna memulai transaksi dengan memilih menu dan menekan tombol *checkout*. Aplikasi secara otomatis mengambil token JWT dari penyimpanan lokal dan menyematkannya ke dalam *Header Authorization* pada HTTP Request, lalu mengirimkan *mutation createOrder* ke **Order Service**. Di sisi server, Order Service pertama-tama melakukan *decoding* dan verifikasi validitas token melalui *middleware*. Jika token valid, layanan akan memvalidasi ketersediaan menu di database **Mysql**, menghitung total harga transaksi, dan membuat dokumen pesanan baru dengan status awal **"PENDING"**. Status ini menandakan bahwa pesanan telah dicatat namun belum melalui pengecekan stok.

3. Integrasi Pengecekan Inventaris (Langkah 19–23) Langkah selanjutnya melibatkan komunikasi antar-layanan (*inter-service communication*). **Order Service** mengirimkan permintaan HTTP internal ke **Inventory Service** untuk memastikan ketersediaan bahan baku. Inventory Service merespons dengan melakukan *query* ke database MySQL miliknya. Sistem menerapkan logika keputusan: jika stok di database mencukupi untuk jumlah yang diminta, Inventory Service akan segera melakukan pengurangan stok (*decrement*) secara *real-time* dan mengembalikan respon sukses ("OK") ke Order Service. Mekanisme ini mencegah terjadinya pesanan yang tidak bisa dipenuhi karena kehabisan bahan.

4. Integrasi Operasional Dapur (Langkah 24–27) Setelah stok terkonfirmasi aman, **Order Service** melanjutkan alur dengan mengirimkan permintaan pembuatan tiket ke **Kitchen Service**. Layanan ini bertanggung jawab mencatat pesanan ke dalam tabel antrian (*queue*) di database operasional dapur dan menetapkan status pesanan menjadi **"QUEUED"**. Kitchen Service kemudian mengembalikan sebuah *Ticket ID* unik ke Order Service, yang nantinya digunakan oleh koki untuk melacak dan memproses pesanan tersebut secara fisik.

5. Integrasi Program Loyalitas (Langkah 28–30) Secara paralel atau berurutan, sistem juga menjalankan logika bisnis untuk retensi pelanggan. **Order Service** menghubungi **User Service** kembali untuk memicu penambahan poin loyalitas. User Service akan mengkalkulasi jumlah poin bonus (misalnya 10% dari total transaksi) dan memperbarui saldo poin pengguna di database MySQL. Proses ini berjalan otomatis di latar belakang tanpa mengganggu pengalaman pengguna, memastikan setiap transaksi yang berhasil langsung memberikan *reward* kepada pelanggan.

6. Finalisasi Transaksi (Langkah 31–33) Setelah seluruh rangkaian integrasi (Inventory, Kitchen, dan User) berhasil merespons dengan sukses, **Order Service** melakukan langkah

terakhir yaitu memperbarui status pesanan di database Mysql dari "PENDING" menjadi **"CREATED"** (atau Sukses). Layanan kemudian menyusun seluruh data transaksi menjadi paket JSON lengkap dan mengirimkannya kembali ke aplikasi pengguna (*Client*). Aplikasi menerima respon tersebut dan menampilkan halaman "Transaksi Berhasil" atau nota digital kepada pengguna, menandai berakhirnya satu siklus hidup transaksi secara utuh.

3. PEMBAGIAN PERAN (1 MEMBER 1 SERVICE)

Struktur ini mengikuti ketentuan “1 orang 1 layanan per kelompok”.

Anggota 1 (102022330069 - Bimo Alfarizy Lukman) — User & Security Engineer

Service: User Service Fokus: Autentikasi JWT, Manajemen Data Staff & Pelanggan, Sistem Poin Loyalitas.

Ruang kerja teknis:

- **Authentication: Login menghasilkan JWT dengan *claims* (id, role, email).**
- **Loyalty System: Logika penambahan poin setiap kali transaksi sukses dan manajemen *tier* (Bronze, Silver, Gold).**
- **Database: MySQL (`user_db`).**
- **Tech Stack: Node.js (Apollo Server) / Python (Strawberry).**

Anggota 2 (102022330325 - Revaldo A.Nainggolan) — Order & Transaction Manager

Service: Order Service Fokus: Katalog Menu, Keranjang Belanja, Orkestrasi Pesanan.

Ruang kerja teknis:

- **Catalog Management: CRUD menu makanan (nama, harga, kategori).**
- **Transaction Flow: Membuat pesanan yang memicu aksi ke layanan lain (Kirim ke dapur, Potong stok).**
- **Database: Mysql (`order_db`) - *NoSQL dipilih untuk fleksibilitas struktur menu*.**
- **Tech Stack: Node.js (Mongoose) / Python (Motor).**

Anggota 3 (102022300021 - Mochamad Rizky Maulana Aviansyah) — Inventory & Integration Specialist

Service:Inventory Service Fokus: Manajemen Stok Bahan Baku dan Integrasi Lintas Kelompok.

Ruang kerja teknis:

- **Stock Management: Tracking masuk/keluar bahan baku.**
- **External Integration (Consumer): Mengkonsumsi API Toko Sembako untuk melakukan *restock* otomatis atau manual saat stok menipis.**
- **Database: MySQL (`inventory_db`).**
- **Tech Stack: Node.js / Python.**

Anggota 4 (102022300102 - Alvina Sulistina) — Kitchen Operations Engineer

Service: Kitchen Service **Fokus:** Manajemen Operasional Dapur.

Ruang kerja teknis:

- **Queue Management:** Menerima tiket pesanan dari Order Service.
- **Chef Assignment:** Menugaskan koki untuk pesanan tertentu.
- **Status Tracking:** Memantau status (*Pending -> Preparing -> Ready*).
- **Database:** MySQL (`kitchen_db`).
- **Tech Stack:** Node.js / Python.

4. MEKANISME JWT & MODEL KEAMANAN

Arsitektur keamanan sistem microservices pada proyek ini menerapkan autentikasi terpusat menggunakan JSON Web Token (JWT) dengan User Service sebagai penyedia autentikasi utama. Mekanisme keamanan dirancang untuk memastikan setiap layanan hanya dapat diakses oleh pengguna yang memiliki hak akses sesuai perannya.

Authentication:

Proses autentikasi sepenuhnya ditangani oleh User Service. Layanan ini menyimpan kredensial pengguna dalam database MySQL dengan password yang telah dienkripsi. Setelah proses login berhasil, sistem menghasilkan JWT yang berisi klaim identitas pengguna, yaitu id pengguna, email, dan role. Token ini menjadi identitas digital pengguna selama berinteraksi dengan sistem.

Token Distribution:
JWT yang dihasilkan oleh User Service dikirimkan kepada klien dan digunakan kembali pada setiap permintaan ke layanan lain. Token tidak disimpan di layanan selain User Service dan tidak dilakukan replikasi data kredensial antar service.

Token Usage:
Setiap request ke Order Service, Inventory Service, dan Kitchen Service wajib menyertakan JWT pada HTTP header dengan format Authorization Bearer Token. Request tanpa token atau dengan token tidak valid akan langsung ditolak oleh service tujuan.

Token Validation:
Masing-masing layanan selain User Service melakukan validasi JWT sebelum memproses request. Validasi mencakup pemeriksaan tanda tangan token, masa berlaku token, serta konsistensi klaim id, email, dan role yang dikandung di dalamnya.

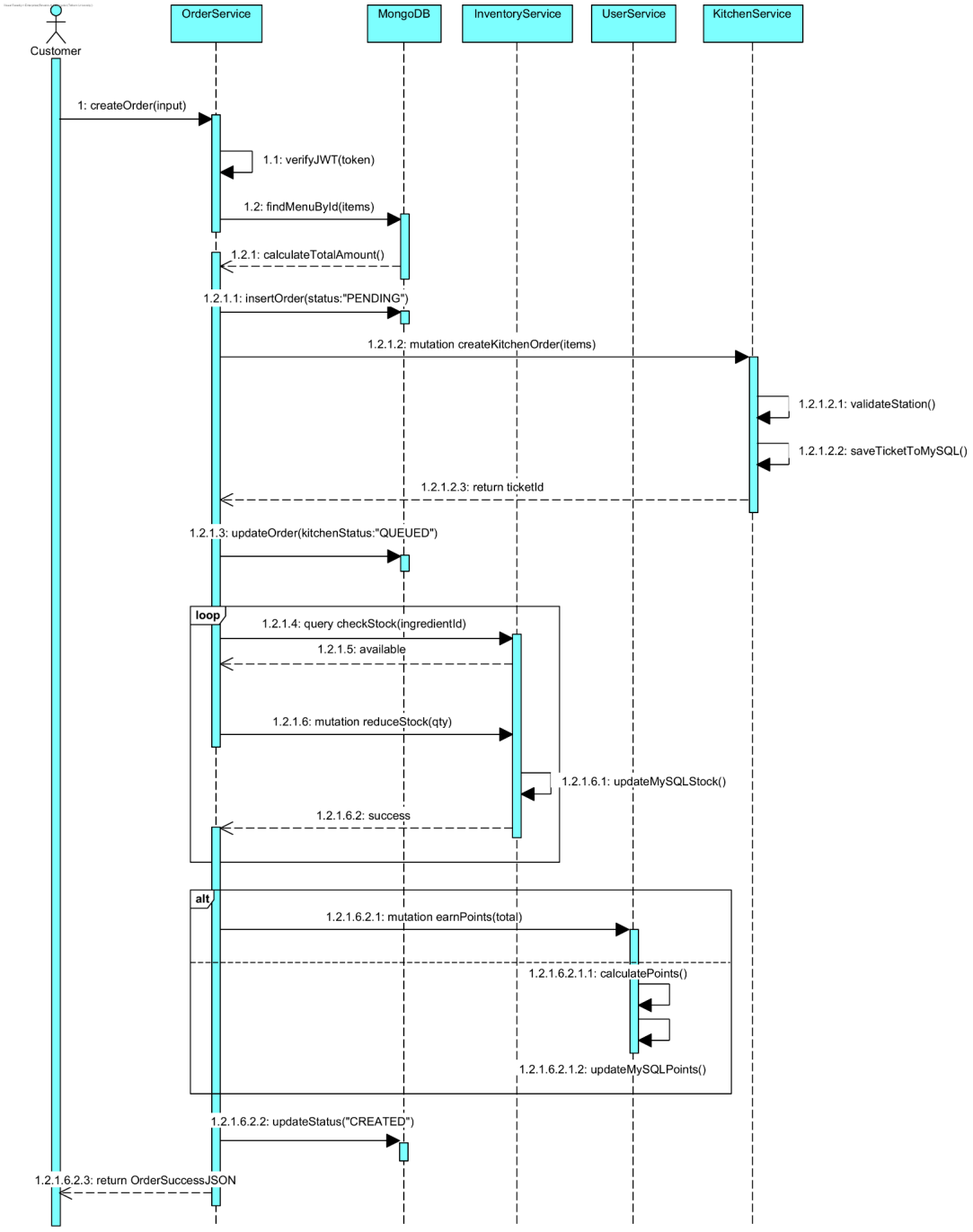
Role-Based Authorization:
Kontrol akses diterapkan berdasarkan nilai role pada JWT. Kitchen Service hanya mengizinkan perubahan status pesanan oleh pengguna dengan role Chef. Inventory Service membatasi akses pengelolaan stok dan proses restock hanya untuk role Admin. Order Service hanya dapat diakses oleh pengguna terautentikasi untuk melakukan pembuatan dan pengelolaan pesanan.

Service-to-Service Security:
Komunikasi antar layanan internal tetap menggunakan JWT sebagai mekanisme keamanan. Order Service hanya dapat mengirim permintaan ke Kitchen Service dan Inventory Service jika menyertakan token yang valid, sehingga setiap interaksi antar service tetap terkontrol dan terlindungi.

Security Consistency:
Dengan pendekatan ini, seluruh microservice bersifat independen dari sisi database dan implementasi, namun tetap berada dalam satu kerangka keamanan yang konsisten. User Service berperan sebagai pusat autentikasi, sementara layanan lain fokus pada validasi token dan otorisasi sesuai fungsi masing-masing, sebagaimana diimplementasikan pada repository TUBES-IAE.

5. ALUR INTEGRASI (INTERNAL & EKSTERNAL)

5.1 Sequence Diagram (Wajib Mengandung Integrasi Lintas Kelompok)



Alur dimulai ketika **Customer** mengirimkan permintaan pesanan melalui fungsi `createOrder` yang menyertakan data item dan token JWT ke **OrderService**. Secara internal, **OrderService** segera melakukan validasi keamanan menggunakan `verifyJWT` dan mengekstraksi identitas

pengguna. Langkah berikutnya adalah sinkronisasi dengan database **Mysql** untuk mencari data menu dan menghitung total harga secara akurat melalui proses `calculateTotalAmount`. Setelah validasi awal selesai, sistem mencatat pesanan tersebut ke dalam database dengan status awal **"PENDING"**.

Proses kemudian berlanjut ke integrasi layanan pertama, yaitu **KitchenService**. **OrderService** mengirimkan mutasi pesanan ke dapur, di mana **KitchenService** melakukan validasi internal dan menyimpan tiket masak ke dalam database **MySQL** miliknya sendiri sebelum mengembalikan `ticketId`. Setelah status dapur diperbarui menjadi **"QUEUED"**, sistem masuk ke dalam logika perulangan (*Loop*) untuk mengelola inventaris. Dalam setiap putaran loop, **OrderService** berkomunikasi dengan **InventoryService** untuk memeriksa ketersediaan bahan baku. Jika stok tersedia, perintah `reduceStock` dijalankan untuk memotong saldo stok di database **MySQL Inventory** secara otomatis.

Setelah urusan inventaris selesai, sistem menjalankan logika opsional (*Opt*) untuk program loyalitas. Jika pelanggan teridentifikasi sebagai member, **OrderService** memicu **UserService** untuk menghitung dan menambah poin loyalitas ke dalam database **MySQL User**. Sebagai tahap finalisasi, **OrderService** memperbarui status pesanan di **Mysql** menjadi **"CREATED"** yang menandakan transaksi telah berhasil sepenuhnya. Seluruh rangkaian proses ini ditutup dengan pengiriman respon JSON sukses kembali ke layar **Customer**, memberikan konfirmasi bahwa pesanan telah diterima dan siap diproses.

5.2 Aturan Bisnis Minimal Enrollment

Sistem Anugerah Resto menerapkan serangkaian aturan bisnis (business rules) yang ketat untuk menjaga integritas data dan kelancaran operasional antar-layanan. Sebuah pesanan (*Order*) hanya dinyatakan valid dan dapat diproses apabila memenuhi seluruh prasyarat berikut:

1. Validasi Autentikasi dan Otorisasi (User Service)

Sebelum transaksi dimulai, sistem harus memvalidasi identitas pengguna:

- **Token Validity:** Setiap permintaan pemesanan (`createOrder`) wajib menyertakan JSON Web Token (JWT) yang valid dan belum kadaluwarsa di *Header Authorization*.
- **Role Validation:** Hanya pengguna dengan peran (`role`) sebagai **'Customer'** yang diizinkan membuat pesanan baru. Pengguna dengan role 'Staff' atau 'Chef' dibatasi hanya pada fungsi manajemen operasional.
- **Account Status:** Akun pengguna harus dalam status aktif (tidak ter-suspend) di database *User Service*.

2. Validasi Ketersediaan Menu & Stok (Inventory Service)

Sistem menerapkan mekanisme pengecekan ganda untuk mencegah pesanan yang tidak dapat dipenuhi:

- **Menu Availability:** Menu yang dipilih harus terdaftar dan berstatus *available* di database *Order Service*.
- **Ingredient Sufficiency:** Untuk setiap menu yang dipesan, *Order Service* akan memicu pengecekan ke *Inventory Service*. Transaksi hanya dapat dilanjutkan jika **stok bahan baku lokal mencukupi** untuk jumlah porsi yang diminta.
    - *Aturan Gagal:* Jika salah satu bahan baku habis, sistem akan menolak seluruh pesanan (Atomic Transaction) dan mengembalikan pesan error "Out of Stock" kepada pelanggan.

3. Validasi Pembayaran Eksternal (Payment Integration)

Sesuai dengan skenario integrasi eksternal, validasi finansial menjadi syarat mutlak sebelum pesanan masuk ke dapur:

- **Payment Status Check:** *Order Service* melakukan query ke layanan eksternal (Payment Gateway/Kelompok Lain) menggunakan `checkPayment(orderId)`.
- **Settlement Rule:** Pesanan hanya akan diteruskan ke *Kitchen Service* jika status pembayaran dari layanan eksternal adalah **'PAID'** atau **'SUCCESS'**. Jika status 'PENDING' atau 'FAILED', pesanan akan tertahan di status 'AWAITING_PAYMENT' dan tidak akan diproses oleh koki.

4. Logika Operasional Dapur (Kitchen Service)

Untuk menjaga alur kerja dapur yang efisien:

- **Queue Entry:** Pesanan yang telah lolos validasi stok dan pembayaran akan dibuatkan **Tiket Masak** unik di *Kitchen Service*.
- **Chef Assignment:** Pesanan yang masuk ke antrian (*queue*) berstatus 'PENDING' dan belum dianggap diproses sampai seorang Koki (*Chef*) mengambil alih pesanan tersebut (status berubah menjadi 'PREPARING').

5. Aturan Program Loyalitas (Loyalty Logic)

Sebagai nilai tambah bagi pelanggan terdaftar:

- **Earning Rules:** Poin loyalitas hanya diberikan setelah transaksi selesai sepenuhnya (Status Order: 'COMPLETED').
- **Calculation Logic:** Sistem menghitung poin berdasarkan persentase total transaksi (misal: 10% dari Total Harga).
- **Tier Update:** Jika akumulasi poin pelanggan melampaui ambang batas tertentu (misal: >1000 poin), *User Service* secara otomatis menaikkan status member dari 'Bronze' ke 'Silver' atau 'Gold'.

6. KONTRAK GRAPHQL (RINGKASAN DESAIN SKEMA)

Untuk memastikan kualitas implementasi GraphQL, setiap microservice menyediakan definisi skema dalam file .graphql yang berisi tipe data, query, dan mutation utama sesuai tanggung jawab layanan masing-masing. Selain itu, setiap layanan menyediakan contoh penggunaan query dan mutation sebagai dokumentasi API.

User Service (Authentication & User Management)

```
type Query {
  me: User
  userById(id: ID!): User
}

type Mutation {
  login(username: String!, password: String!): AuthPayload
  registerUser(input: RegisterUserInput!): User
  updateUser(id: ID!, input: UpdateUserInput!): User
}

type User {
  id: ID!
  name: String!
  email: String!
  role: String!
  loyaltyPoint: Int
  loyaltyTier: String
}

type AuthPayload {
  token: String!
  user: User!
}
```

Skema ini digunakan sebagai pusat autentikasi dan penyedia JWT. Query me digunakan untuk mengambil data pengguna berdasarkan token aktif, sementara mutation login menghasilkan JWT yang digunakan oleh seluruh layanan lain.

Order Service (Order & Transaction Management)

```
type Query {
  menus: [Menu!]!
  menuById(id: ID!): Menu
  myOrders: [Order!]!
}

type Mutation {
  createOrder(input: CreateOrderInput!): Order
  updateOrderStatus(orderId: ID!, status: String!): Order
}

type Menu {
  id: ID!
  name: String!
  price: Float!
  category: String!
}

type Order {
  id: ID!
  userId: ID!
  totalPrice: Float!
  status: String!
  createdAt: String!
}
```

Order Service berfungsi sebagai orkestrator pemesanan. Query berfokus pada pengambilan menu dan riwayat pesanan pengguna, sementara mutation createOrder memicu proses lintas layanan ke Kitchen Service dan Inventory Service.

Inventory Service (Stock Management & Integration)

```
type Query {
  ingredients: [Ingredient!]!
  ingredientById(id: ID!): Ingredient
}

type Mutation {
  updateStock(id: ID!, quantity: Int!): Ingredient
  restockIngredient(id: ID!, quantity: Int!): Ingredient
}

type Ingredient {
```

```
  id: ID!
  name: String!
  stock: Int!
  unit: String!
}
```

Inventory Service bertanggung jawab atas ketersediaan bahan baku. Mutation updateStock dipanggil saat terjadi pemesanan, sedangkan restockIngredient digunakan untuk proses pengisian ulang stok, termasuk hasil integrasi dengan API eksternal toko sembako.

Kitchen Service (Kitchen Operations)

```
type Query {
  kitchenOrders: [KitchenOrder!]!
  kitchenOrderById(id: ID!): KitchenOrder
}

type Mutation {
  assignChef(orderId: ID!, chefId: ID!): KitchenOrder
  updateKitchenStatus(orderId: ID!, status: String!): KitchenOrder
}

type KitchenOrder {
  id: ID!
  orderId: ID!
  chefId: ID
  status: String!
}
```

Kitchen Service menerima tiket pesanan dari Order Service dan mengelola proses dapur. Mutation updateKitchenStatus hanya dapat diakses oleh pengguna dengan role Chef sesuai mekanisme otorisasi JWT.

External Service (Dikonsumsi – Payment / Supplier)

```
type Query {
  checkPayment(orderId: ID!): PaymentStatus
}

type PaymentStatus {
  orderId: ID!
  status: String!
  paidAt: String
}
```

Skema ini merepresentasikan layanan eksternal yang dikonsumsi, bukan dikembangkan oleh kelompok. Order Service menggunakan query checkPayment untuk memverifikasi status pembayaran sebelum melanjutkan proses pesanan ke tahap dapur dan inventori.

7. STRATEGI DEPLOYMENT DOCKER & TOPOLOGI JARINGAN

Sistem dijalankan via `docker-compose.yml` agar semua service berjalan pada network yang sama, dengan database masing-masing sebagai container terpisah. Rekomendasi struktur:

- `auth-service` + `auth-db`
- `course-service` + `catalog-db`
- `enrollment-service` + `enrollment-db`
- `payment-service` (external; bisa pakai URL dari kelompok lain, atau container mock saat pengujian lokal)

Target penilaian: semua service terpisah dalam container berbeda dan dapat dijalankan ulang dengan setup yang rapi.

8. RENCANA INTEGRASI LINTAS KELOMPOK (WAJIB)

Inventory Service berperan sebagai **Consumer** yang mengkonsumsi minimal 1 endpoint GraphQL dari Toko Sembako. Berikut adalah endpoint yang dikonsumsi:

8.1 Mengkonsumsi Product Service (Toko Sembako)

Endpoint: `http://toko-sembako-product:4001/graphql`

Query yang Dikonsumsi:

```
# Query 1: Get All Products
query GetProducts($category: String) {
  products(category: $category) {
    id
    name
    category
    price
    unit
```

```
    available
    description
  }
}

# Query 2: Get Product by ID
query GetProductById($productId: ID!) {
  product(id: $productId) {
    id
    name
    category
    price
    unit
    available
    description
  }
}
```

Implementasi di Inventory Service:

```
# inventory-service-python/src/services/toko_sembako_client.py
TOKO_SEMBAKO_PRODUCT_URL =
os.getenv("TOKO_SEMBAKO_PRODUCT_URL",

"http://host.docker.internal:4001/graphql")

async def get_products_from_toko_sembako(category:
Optional[str] = None):
    """Mengkonsumsi Product Service dari Toko Sembako"""
    query = """
    query GetProducts($category: String) {
        products(category: $category) {
          id
          name
          category
          price
          unit
          available
          description
        }
    }
    """
```

```
    variables = {"category": category} if category else {}

    async with httpx.AsyncClient() as client:
        response = await client.post(
          TOKO_SEMBAKO_PRODUCT_URL,
          json={"query": query, "variables": variables},
          headers={"Content-Type": "application/json"}
        )
        data = response.json()
        return data.get("data", {}).get("products", [])
```

Exposed di Inventory Service GraphQL:

```
# Inventory Service memaparkan query untuk frontend
type Query {
  tokoSembakoProducts(category: String): [TokoSembakoProduct!]!
}

type TokoSembakoProduct {
  id: String!
  name: String!
  category: String
  price: Float!
  unit: String!
  available: Boolean!
  description: String
}
```

8.1.2 Mengkonsumsi Inventory Service (Toko Sembako)

Endpoint: `http://toko-sembako-inventory:4000/graphql`

Query yang Dikonsumsi:

```
# Check Stock dari Toko Sembako
query CheckStock($productId: ID!, $quantity: Float!) {
  checkStock(productId: $productId, quantity: $quantity) {
    available
    currentStock
```

```
    requestedQuantity
    message
  }
}
```

Implementasi:

```
async def check_stock_from_toko_sembako(product_id: str,
quantity: float):
    """Mengkonsumsi Inventory Service dari Toko Sembako untuk
cek stok"""
    query = """
    query CheckStock($productId: ID!, $quantity: Float!) {
        checkStock(productId: $productId, quantity: $quantity)
{
          available
          currentStock
          requestedQuantity
          message
        }
    }
    """
    variables = {
        "productId": product_id,
        "quantity": quantity
    }

    async with httpx.AsyncClient() as client:
        response = await client.post(
          TOKO_SEMBAKO_INVENTORY_URL,
          json={"query": query, "variables": variables}
        )
        data = response.json()
        return data.get("data", {}).get("checkStock", {})
```

8.1.3 Mengkonsumsi Order Service (Toko Sembako)

Endpoint: `http://toko-sembako-order:4002/graphql`

Mutation yang Dikonsumsi:

```
# Create Order di Toko Sembako
mutation CreateOrder($input: CreateOrderInput!) {
```

```
  createOrder(input: $input) {
    id
    orderId
    status
    total
    items {
      productId
      name
      quantity
      price
    }
    createdAt
  }
}
```

Implementasi:

```
async def create_order_at_toko_sembako(order_number: str,
items: List[Dict], notes: Optional[str] = None):
    """Mengkonsumsi Order Service dari Toko Sembako untuk
membuat pesanan"""
    mutation = """
    mutation CreateOrder($input: CreateOrderInput!) {
        createOrder(input: $input) {
          id
          orderId
          status
          total
          items {
               productId
               name
               quantity
               price
          }
          createdAt
        }
    }
    """

    order_input = {
        "orderNumber": order_number,
        "items": items,
```

```
        "notes": notes
    }

    variables = {"input": order_input}

    async with httpx.AsyncClient() as client:
        response = await client.post(
          TOKO_SEMBAKO_ORDER_URL,
          json={"query": mutation, "variables": variables}
        )
        data = response.json()
        return data.get("data", {}).get("createOrder", {})
```

Alur Penggunaan:

1. Admin restoran melihat daftar produk dari Toko Sembako melalui query `tokoSembakoProducts`

2. Admin memilih produk dan mengecek ketersediaan stok melalui `checkTokoSembakoStock`

3. Admin membuat pesanan pembelian melalui mutation `purchaseFromTokoSembako`

4. Inventory Service memanggil Order Service Toko Sembako untuk membuat pesanan

5. Setelah pesanan berhasil, stok lokal di restoran ditambahkan secara otomatis

8.2 Sebagai Provider (Nilai Tambah)

Sebagai nilai tambah dan bukti integrasi nyata, sistem **Anugerah Resto** juga menyediakan endpoint GraphQL yang dapat dikonsumsi oleh kelompok lain. **User Service** dan **Order Service** menyediakan endpoint untuk integrasi eksternal.

8.2.1 User Service sebagai Provider

Endpoint yang Disediakan: `http://user-service:4003/graphql`

Query yang Disediakan untuk Kelompok Lain:

```
# Query 1: Get Customer by ID (untuk validasi customer dari
sistem eksternal)
query GetCustomerById($customerId: ID!) {
  customerById(id: $customerId) {
    id
```

```
    customerId
    name
    email
    phone
    loyaltyPoints
    status
  }
}

# Query 2: Get Staff by ID (untuk validasi staff dari sistem
eksternal)
query GetStaffById($staffId: ID!) {
  staffById(id: $staffId) {
    id
    staffId
    name
    email
    phone
    role
    status
  }
}

# Query 3: Validate Customer Loyalty Points
query ValidateLoyaltyPoints($customerId: ID!, $requiredPoints:
Float!) {
  customerById(id: $customerId) {
    id
    loyaltyPoints
  }
}
```

Contoh Penggunaan oleh Kelompok Lain:

```
# Kelompok lain (misalnya Payment Gateway) dapat memvalidasi
customer
query {
  customerById(customerId: "CUST001") {
    id
    name
    loyaltyPoints
    status
```

```
  }
}
```

8.2.2 Order Service sebagai Provider

Endpoint yang Disediakan: http://order-service:4004/graphql

Query yang Disediakan untuk Kelompok Lain:

```
# Query 1: Get Order by ID (untuk tracking pesanan dari sistem
eksternal)
query GetOrderById($orderId: ID!) {
  order(id: $orderId) {
    id
    orderId
    customerId
    total
    orderStatus
    paymentStatus
    items {
      menuId
      name
      quantity
      price
    }
    createdAt
  }
}

# Query 2: Get Orders by Customer (untuk riwayat pesanan)
query GetOrdersByCustomer($customerId: String!) {
  orders(customerId: $customerId) {
    id
    orderId
    total
    orderStatus
    createdAt
  }
}

# Query 3: Check Menu Availability
query CheckMenuAvailability($menuId: ID!, $quantity: Int!) {
```

```
  menu(id: $menuId) {
    id
    name
    available
    price
  }
  checkMenuStock(menuId: $menuId, quantity: $quantity) {
    available
    message
    ingredientId
    ingredientName
    required
    availableQuantity
  }
}
```

Contoh Penggunaan oleh Kelompok Lain:

```
# Kelompok lain (misalnya Delivery Service) dapat tracking
order
query {
  order(orderId: "ORD001") {
    id
    orderId
    orderStatus
    items {
      name
      quantity
    }
    total
  }
}
```

9. RENCANA PENGUJIAN & BUKTI (SCREENSHOT)

Agar memenuhi deliverables dokumentasi (diagram, desain skema, penjelasan integrasi, dan screenshot komunikasi), tim menyiapkan bukti berikut.

Bukti minimal yang akan dikumpulkan(:

1. Screenshot GraphQL Playground: login menghasilkan token.
2. Screenshot request enrollment membawa JWT dan sukses setelah cek payment.
3. Screenshot log antar container (Enrollment memanggil Auth, Course, dan Payment).
4. Screenshot `docker ps` dan service healthcheck berjalan.

---

10. TIMELINE PENGERJAAN (MINGGU 13–16)

Timeline mengikuti ketentuan pengerjaan minggu 13–16 dengan *weekly daily report* dan pengujian/presentasi minggu 16.

Minggu	Fokus Kegiatan	Output Terukur
13	Perancangan arsitektur microservices dan pembagian peran tim	Dokumen arsitektur sistem, diagram integrasi antar service, penetapan tanggung jawab masing-masing anggota
13	Perancangan skema GraphQL per service	File `.graphql` untuk User, Order, Inventory, dan Kitchen Service
14	Implementasi User Service dan mekanisme JWT	Endpoint login berjalan, JWT dapat dihasilkan dan divalidasi oleh service lain
14	Implementasi Order Service dan manajemen menu	Query dan mutation Order Service berfungsi, data pesanan tersimpan di database
15	Implementasi Inventory Service dan integrasi eksternal	Manajemen stok berjalan, pengurangan dan penambahan stok tervalidasi
15	Implementasi Kitchen Service dan alur dapur	Antrean pesanan, penugasan koki, dan pembaruan status pesanan berjalan
15	Integrasi antar service dan pengujian lintas layanan	Komunikasi Order–Inventory–Kitchen berhasil menggunakan JWT
16	Pengujian end-to-end dan perbaikan bug	Seluruh alur sistem berjalan tanpa error kritis
16	Finalisasi dokumentasi dan repository	Laporan PDF lengkap, repository GitHub rapi dan terdokumentasi
16	Presentasi dan demo sistem	Demo login, pemesanan, update stok, dan status dapur berhasil ditampilkan

11. DELIVERABLES (SESUSAI KETENTUAN)

Deliverables proyek disusun sesuai dengan ketentuan tugas, mencakup dokumentasi tertulis, kode program pada repository GitHub, serta presentasi dan demo sistem berbasis microservices dan GraphQL.

Dokumentasi Proyek (PDF)

Dokumen laporan berisi penjelasan menyeluruh mengenai sistem yang dibangun, meliputi:

- Deskripsi arsitektur microservices beserta diagram integrasi antar layanan.
- Desain skema GraphQL untuk setiap service (User, Order, Inventory, Kitchen) beserta contoh query dan mutation.
- Narasi integrasi lintas kelompok, khususnya peran Inventory Service sebagai consumer API eksternal (toko sembako) dan interaksi internal antar service.
- Lampiran screenshot komunikasi antar layanan, termasuk proses autentikasi JWT, pemesanan, pembaruan stok, dan status dapur.

Kode Program (GitHub Repository)

Kode program disimpan dalam repository GitHub dengan struktur folder yang terorganisir dan mencerminkan arsitektur microservices, yaitu:

- Struktur folder layanan: `/user-service`, `/order-service`, `/inventory-service`, `/kitchen-service`
- File `docker-compose.yml` sebagai orkestrator container untuk menjalankan seluruh service secara bersamaan.
- File `Dockerfile` pada masing-masing service.
- Folder `schemas/*.graphql` yang berisi definisi skema GraphQL per service.
- (Opsional) Folder `queries/*.graphql` yang berisi contoh query dan mutation sebagai dokumentasi penggunaan API.

Presentasi dan Demo

Presentasi dan demo sistem menampilkan alur utama penggunaan aplikasi secara end-to-end, meliputi:

- Proses login pada User Service hingga menghasilkan JWT.
- Penggunaan token untuk mengakses Order Service dan membuat pesanan.
- Proses orkestrasi pesanan yang memicu komunikasi ke Inventory Service (pengurangan stok) dan Kitchen Service (pengolahan pesanan).
- Penjelasan peran masing-masing service serta mekanisme komunikasi antar container dalam lingkungan Docker.

12. KRITERIA KEBERHASILAN (ACCEPTANCE CRITERIA)

- Semua layanan dapat dijalankan melalui Docker dan saling terhubung via network compose.
- GraphQL schema dapat diakses dan diuji; query berjalan konsisten.
- Integrasi lintas kelompok terbukti: Enrollment memanggil Payment eksternal dan hasilnya mempengaruhi keputusan bisnis.
- Dokumentasi lengkap dan dapat direplikasi oleh dosen/asisten.