

LAPORAN TUGAS BESAR INTEGRASI APLIKASI ENTERPRISE



Sistem Informasi Telkom University Surabaya

Disusun Oleh:

Dirga Eka Prasetya	1204220002
Azzahra Rizki Sulardi Trisnawati	1204220118
Ryan Firmansyah	1204220077
Anandito Satria Pradana	1204210068

Dosen Pengampu:
Purnama Anaking, S. Kom., M. Kom.

**PROGRAM STUDI S1 SISTEM INFORMASI
FAKULTAS REKAYASA INDUSTRI
TELKOM UNIVERSITY SURABAYA
2025**

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi informasi yang semakin pesat telah membawa perubahan signifikan dalam cara perusahaan mengelola dan menjalankan operasionalnya. Salah satu inovasi terbaru dalam dunia perangkat lunak adalah penerapan arsitektur *microservices*, yang memungkinkan pengembangan aplikasi lebih fleksibel, skalabel, dan mudah dikelola. *Microservices* adalah pendekatan di mana aplikasi besar dibagi menjadi layanan-layanan kecil yang dapat beroperasi secara independen, namun tetap dapat berinteraksi satu sama lain untuk mencapai tujuan bersama. Dalam mengintegrasikan berbagai *microservices*, *REST API* (*Representational State Transfer Application Programming Interface*) menjadi pilihan utama. *REST API* menawarkan cara komunikasi yang efisien antar layanan, memungkinkan pertukaran data dalam format standar seperti JSON atau XML. *REST API* memfasilitasi interaksi antar sistem dengan menggunakan HTTP, yang telah menjadi protokol komunikasi standar di dunia web. Namun, dalam implementasi *microservices* yang melibatkan berbagai layanan terdistribusi, tantangan terbesar adalah pengelolaan kontainerisasi layanan tersebut. *Docker* menyediakan solusi dengan menyediakan wadah (*container*) untuk menjalankan aplikasi secara terisolasi, sehingga memudahkan pengelolaan aplikasi yang dibangun dengan *microservices*. *Docker* juga memungkinkan deployment yang konsisten di berbagai lingkungan, dari pengembangan hingga produksi. Selain itu, untuk memastikan komunikasi yang efisien dan terjadwal antar *microservices*, *RabbitMQ*, sebagai sistem antrian pesan, menawarkan solusi pengelolaan pesan yang handal. *RabbitMQ* memastikan pengiriman pesan secara asinkron antar layanan, mengurangi ketergantungan antar layanan dan meningkatkan ketahanan aplikasi secara keseluruhan.

Di sisi lain, untuk meningkatkan fleksibilitas dalam pengambilan data antar *microservices*, *GraphQL* muncul sebagai alternatif dari *REST API*. *GraphQL* memungkinkan klien untuk meminta data dengan lebih tepat dan fleksibel, mengurangi jumlah permintaan yang diperlukan, serta memberikan kontrol yang lebih besar terhadap data yang diterima. Hal ini penting dalam sistem yang memiliki banyak

layanan dan data yang beragam. Dengan mengintegrasikan berbagai teknologi tersebut REST API, Docker Container, GraphQL, dan RabbitMQ aplikasi microservices dapat berjalan lebih efisien, lebih mudah di-deploy, dan lebih fleksibel dalam mengelola komunikasi antar layanan. Laporan ini akan membahas implementasi integrasi aplikasi *Enterprise Microservices* menggunakan teknologi-teknologi tersebut untuk meningkatkan kinerja, skalabilitas, dan ketahanan sistem aplikasi.

1.2. Rumusan Masalah

- a. Bagaimana cara mengintegrasikan berbagai microservices yang dikembangkan secara terpisah dengan menggunakan *REST API* untuk memastikan komunikasi antar layanan berjalan efektif dan efisien?
- b. Apa tantangan dalam penerapan *Docker Container* untuk mengelola berbagai microservices dalam lingkungan yang terisolasi, dan bagaimana cara mengoptimalkan penggunaan Docker untuk skalabilitas dan portabilitas aplikasi?
- c. Bagaimana cara mengimplementasikan *GraphQL* sebagai alternatif untuk REST API dalam mengelola permintaan data antar *microservices*, serta bagaimana pengaruhnya terhadap efisiensi dan fleksibilitas pengambilan data?

1.3. Tujuan

- a. Menganalisis dan mengimplementasikan integrasi antar microservices menggunakan REST API untuk memastikan komunikasi yang efisien dan efektif antara berbagai layanan yang terdistribusi dalam aplikasi Enterprise.
- b. Mengoptimalkan penggunaan Docker Container dalam pengelolaan dan orkestrasi microservices, dengan tujuan meningkatkan skalabilitas, portabilitas, dan konsistensi lingkungan pengembangan dan produksi.
- c. Mengeksplorasi penerapan GraphQL sebagai alternatif dari REST API untuk mengelola permintaan data, sehingga memberikan fleksibilitas lebih besar dalam pengambilan data dan mengurangi jumlah permintaan yang diperlukan dalam arsitektur microservices.

BAB 2

TINJAUAN PUSTAKA

2.1. Laravel Framework

Laravel merupakan seramework open-source untuk pengembangan aplikasi web berbasis PHP. Dikembangkan oleh Taylor Otwell, Laravel dirancang untuk menyederhanakan proses pengembangan aplikasi web dengan menyediakan struktur dan sintaksis yang bersih dan ekspresif. Laravel mengikuti pola arsitektur MVC (Model-View-Controller) yang memisahkan logika aplikasi, tampilan, dan data untuk mempermudah pengelolaan kode dan memfasilitasi pengembangan yang lebih cepat.

2.2. Microservices

Microservices adalah pendekatan arsitektur perangkat lunak yang membagi aplikasi besar dan kompleks menjadi serangkaian layanan kecil yang dapat beroperasi secara independen, masing-masing bertanggung jawab atas fungsi tertentu. Setiap layanan atau microservice ini biasanya memiliki basis data sendiri, dikembangkan, dan dikerjakan oleh tim yang terpisah, serta dapat berkomunikasi satu sama lain melalui protokol jaringan, seperti HTTP/REST atau message brokers.

2.3. Rest API

REST API adalah sebuah arsitektur yang digunakan untuk membangun layanan web yang dapat diakses melalui protokol HTTP. REST berfokus pada penggunaan metode HTTP seperti GET, POST, PUT, DELETE, untuk melakukan operasi pada sumber daya (resource) yang diwakili dalam format seperti JSON atau XML. Konsep utama dari REST adalah untuk memisahkan antara klien dan server, sehingga aplikasi dapat berinteraksi secara terpisah namun tetap berjalan secara efisien.

2.4. Docker Container

Docker Container adalah platform open-source yang memungkinkan pengembang untuk membangun, mengemas, dan menjalankan aplikasi di dalam container. Container Docker memberikan isolasi antara aplikasi dan sistem operasi di bawahnya, memastikan bahwa aplikasi berjalan dengan cara yang konsisten di berbagai lingkungan (development, testing, production). Container ini ringan dan cepat karena berbagi kernel host, berbeda dengan mesin virtual yang memiliki overhead lebih besar.

2.5. GraphQL

Docker Container merupakan platform open-source yang memungkinkan pengembang untuk membangun, mengemas, dan menjalankan aplikasi di dalam container.

Container Docker memberikan isolasi antara aplikasi dan sistem operasi di bawahnya, memastikan bahwa aplikasi berjalan dengan cara yang konsisten di berbagai lingkungan (development, testing, production). Container ini ringan dan cepat karena berbagi kernel host, berbeda dengan mesin virtual yang memiliki overhead lebih besar. GraphQL adalah sebuah query language untuk API dan runtime untuk mengeksekusi query tersebut dengan menggunakan data yang sudah ada. Dibuat oleh Facebook, GraphQL memungkinkan klien untuk meminta hanya data yang dibutuhkannya, mengurangi pemborosan data dibandingkan dengan REST API yang cenderung mengembalikan data lebih banyak dari yang diperlukan. Salah satu keuntungan utama GraphQL adalah kemampuannya untuk menggabungkan beberapa sumber data menjadi satu permintaan (request).

2.6. RabbitMQ

RabbitMQ message broker open-source yang mendukung protokol messaging seperti AMQP (Advanced Message Queuing Protocol). RabbitMQ memungkinkan sistem untuk mengirim dan menerima pesan antara aplikasi secara asinkron. Ini memungkinkan aplikasi untuk berkomunikasi tanpa tergantung langsung pada satu sama lain, memberikan lebih banyak fleksibilitas dan skalabilitas pada arsitektur sistem.

2.7. API

API (Application Programming Interface) adalah sekumpulan definisi dan protokol yang memungkinkan perangkat lunak atau aplikasi untuk berinteraksi satu sama lain. API dapat dianggap sebagai jembatan yang memungkinkan komunikasi antar sistem yang berbeda dengan cara yang terstruktur. API memungkinkan aplikasi untuk menggunakan fungsionalitas atau data dari aplikasi lain tanpa harus memahami implementasi internalnya. API bisa berbentuk pustaka, protokol, atau bahkan layanan web yang memungkinkan akses kepada berbagai sistem atau layanan.

BAB 3

HASIL DAN PEMBAHASAN

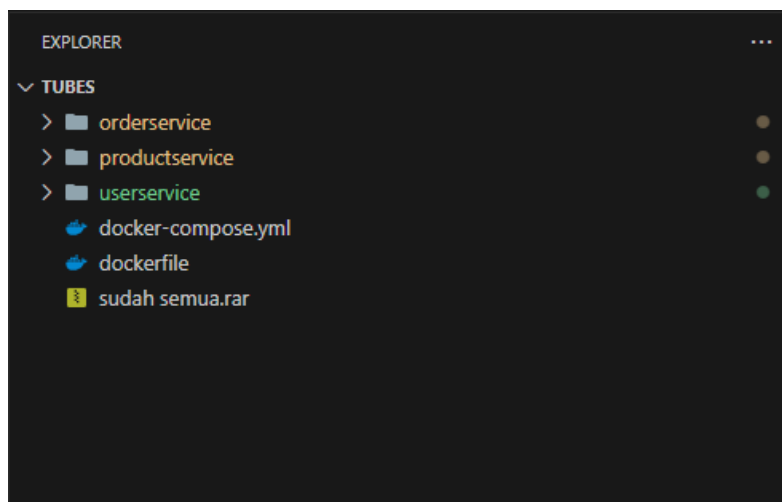
3.1. Dokumentasi Arsitektur

Berikut merupakan struktur dan desain sistem microservice secara keseluruhan.

A. Struktur Microservice

Microservice pada tugas besar ini terdiri dari tiga layanan utama yang terpisah:

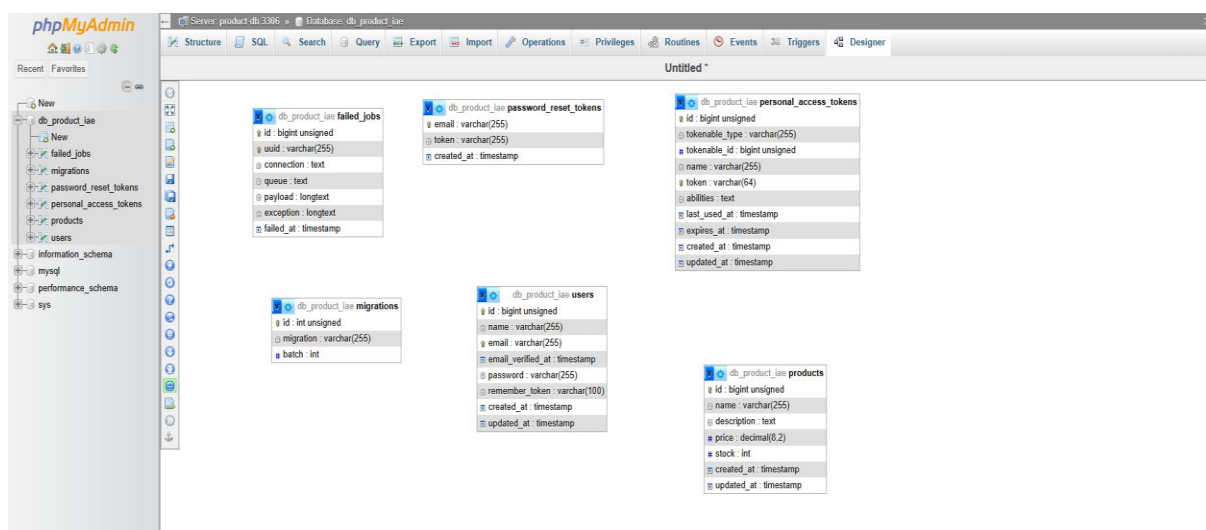
- User Service: Layanan yang menyimpan data pengguna (user).
- Product Service: Layanan yang menyimpan data produk (product).
- Order Service: Layanan yang menyimpan data pesanan (order).



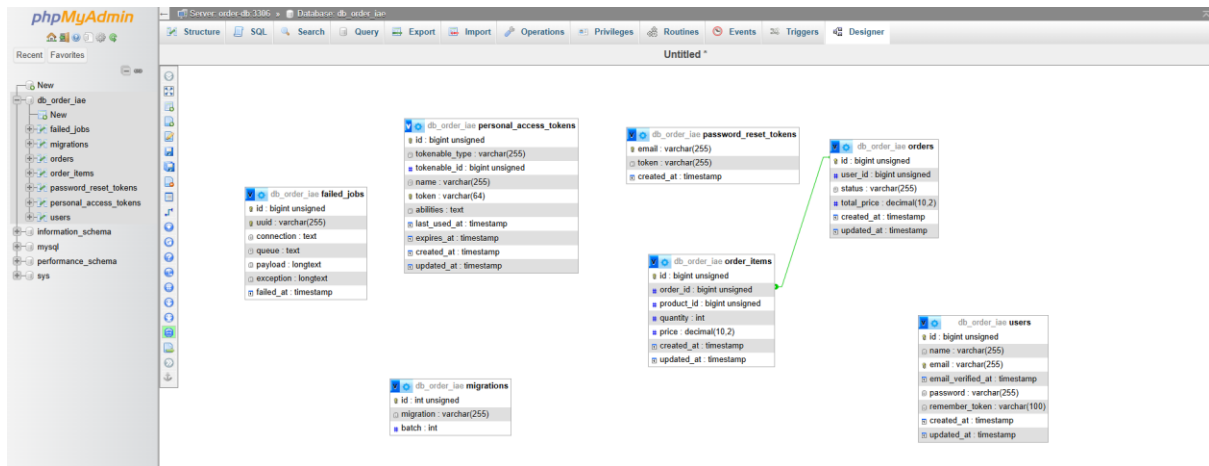
B. Database Setiap Layanan

Berikut merupakan data base setiap layanan microservice yang digunakan untuk mengakase dan menyimpan data.

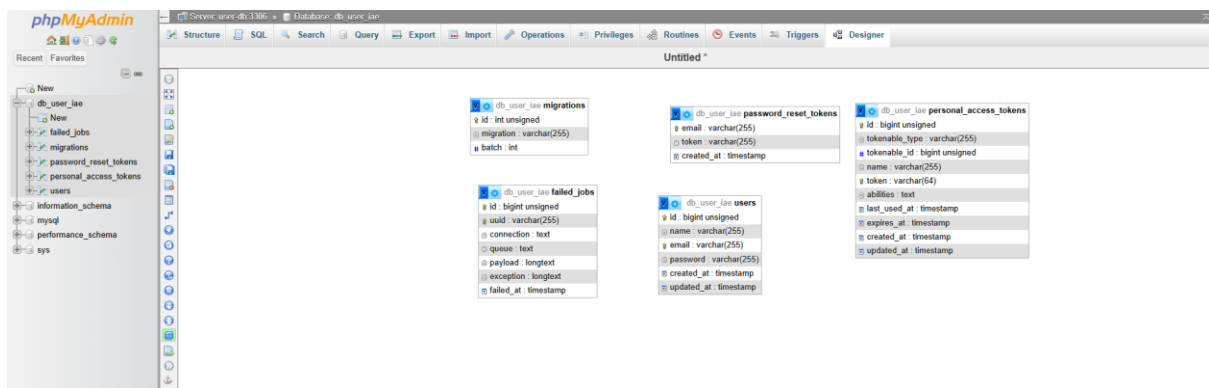
• Product service



- Order service



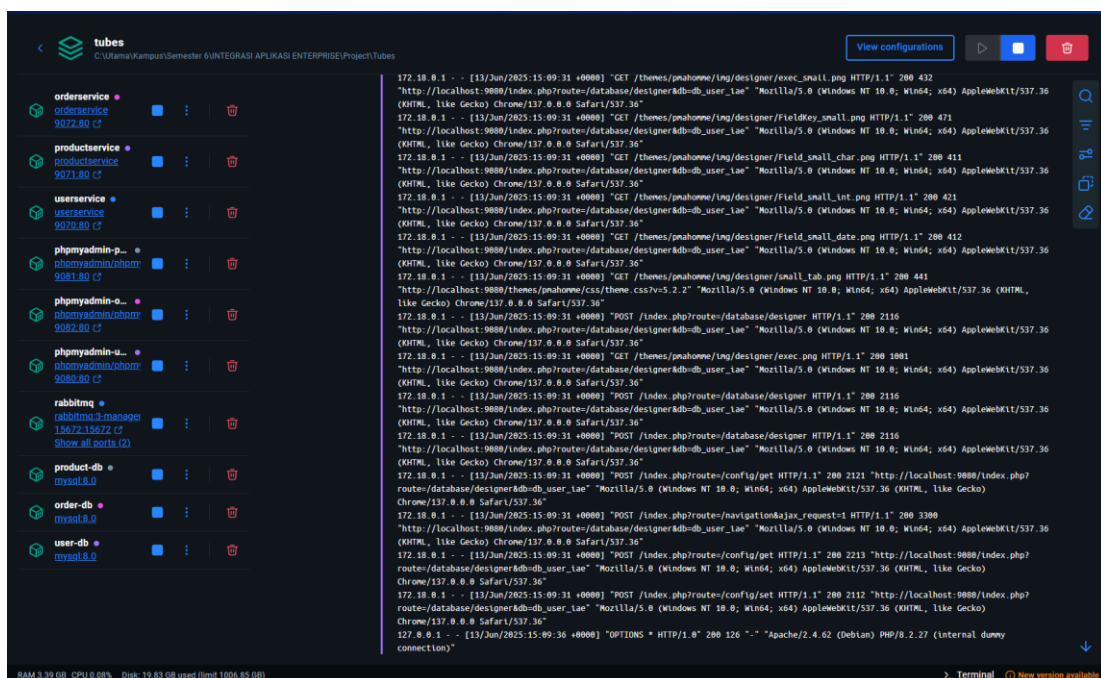
- User service



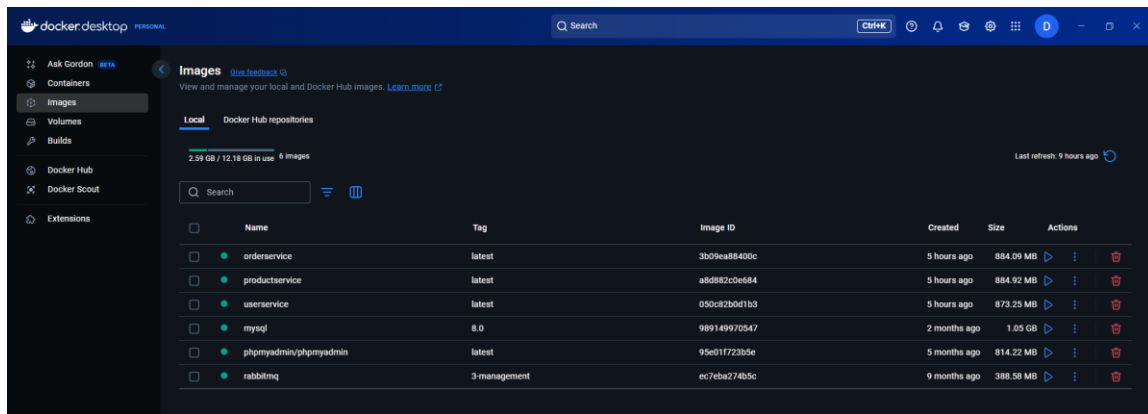
C. Implementasi Docker

Implementasi docker dibuat untuk membuat image dan membuat container untuk menjalankan aplikasi bersama semua dependensinya, termasuk pustaka, konfigurasi, dan file sistem yang dibutuhkan sistem

- Container



- Image



A. Docker File

Dockerfile ini menyatakan langkah-langkah yang diperlukan untuk mengonfigurasi lingkungan tempat aplikasi akan berjalan di dalam container Docker. Setiap baris dalam Dockerfile adalah perintah yang akan dijalankan saat membangun image, mulai dari memilih base image, menginstal dependensi, menyalin kode aplikasi, hingga menjalankan aplikasi.

- Semua layanan sama

```
orderservice > dockerfile > ...
1 FROM php:8.2-apache
2
3 RUN apt-get update && apt-get install -y \
4     curl \
5     libzip-dev zip unzip \
6     && docker-php-ext-install pdo pdo_mysql zip sockets
7
8
9 RUN a2enmod rewrite
10
11 ENV APACHE_DOCUMENT_ROOT=/var/www/html/public
12 RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/sites-available/*.conf
13 RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/apache2.conf /etc/apache2/conf-available/*.conf
14
15 WORKDIR /var/www/html
16
17 COPY . .
18
19 RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
20
21 RUN composer install
22
23 RUN chown -R www-data:www-data /var/www/html/storage /var/www/html/bootstrap/cache
24
25 EXPOSE 80
26
27 CMD ["apache2-foreground"]
```

B. Docker Compose: User Service

Docker Compose menggunakan file konfigurasi YAML yang untuk mendefinisikan pada layanan, jaringan, dan volume yang digunakan dalam pengembangan program

- Laravel


```

4    userservice:
5      build:
6        context: ./userservice
7        dockerfile: Dockerfile
8      image: userservice
9      container_name: laravel-userservice
10     restart: unless-stopped
11     working_dir: /var/www/html
12     volumes:
13       - ./userservice:/var/www/html
14     networks:
15       - laravel-net
16     ports:
17       - 9070:80
18     environment:
19       DB_HOST: user-db
20       DB_DATABASE: db_user_iae
21       DB_USERNAME: root
22       DB_PASSWORD: root
23       QUEUE_CONNECTION: database
24     depends_on:
25       - user-db
26     command: >
27       sh -c "php artisan migrate --force && apache2-foreground"

```

- Database

```

29    user-db:
30      image: mysql:8.0
31      container_name: mysql-user-db
32      environment:
33        MYSQL_DATABASE: db_user_iae
34        MYSQL_ROOT_PASSWORD: root
35      volumes:
36        - mysql-user:/var/lib/mysql
37      networks:
38        - laravel-net

```

C. Docker Compose: Product Service

menggunakan Docker Compose, kita dapat dengan mudah mengelola aplikasi berbasis mikroservis dan memastikan bahwa semua layanan yang saling bergantung berjalan dengan baik dalam satu perintah..

- Laravel

```

40 productservice:
41   build:
42     context: ./productservice
43     dockerfile: Dockerfile
44   image: productservice
45   container_name: laravel-productservice
46   restart: unless-stopped
47   working_dir: /var/www/html
48   volumes:
49     - ./productservice:/var/www/html
50   networks:
51     - laravel-net
52   ports:
53     - 9071:80
54   environment:
55     DB_HOST: product-db
56     DB_DATABASE: db_product_iae
57     DB_USERNAME: root
58     DB_PASSWORD: root
59     QUEUE_CONNECTION: rabbitmq
60   depends_on:
61     - product-db
62     - userservice
63   command: >
64     sh -c "php artisan migrate --force && apache2-foreground"

```

- Database

```

66 product-db:
67   image: mysql:8.0
68   container_name: mysql-product-db
69   environment:
70     MYSQL_DATABASE: db_product_iae
71     MYSQL_ROOT_PASSWORD: root
72   volumes:
73     - mysql-product:/var/lib/mysql
74   networks:
75     - laravel-net

```

D. Docker Compose: Order Service

Dalam implementasi Docker Compose, Order Service ini akan dihubungkan dengan layanan lain, seperti Product Service dan Database Service untuk menyimpan data pesanan.

- **Laravel**

```

  ▷ Run Service
77  orderservice:
78    build:
79      context: ./orderservice
80      dockerfile: Dockerfile
81    image: orderservice
82    container_name: laravel-orderservice
83    restart: unless-stopped
84    working_dir: /var/www/html
85    volumes:
86      - ./orderservice:/var/www/html
87    networks:
88      - laravel-net
89    ports:
90      - 9072:80
91    environment:
92      DB_HOST: order-db
93      DB_DATABASE: db_order_iae
94      DB_USERNAME: root
95      DB_PASSWORD: root
96      QUEUE_CONNECTION: rabbitmq
97    depends_on:
98      - order-db
99      - userservice
100     - productservice
101    command: >
102    sh -c "php artisan migrate --force && apache2-foreground"
```

- **Database**

```

  ▷ Run Service
104  order-db:
105    image: mysql:8.0
106    container_name: mysql-order-db
107    environment:
108      MYSQL_DATABASE: db_order_iae
109      MYSQL_ROOT_PASSWORD: root
110    volumes:
111      - mysql-order:/var/lib/mysql
112    networks:
113      - laravel-net
```

E. Docker Compose: RabbitMQ

Digunakan untuk mengirim pesan antara aplikasi atau komponen sistem yang terpisah, memastikan komunikasi yang asinkron dan terpisah antara pengirim dan penerima pesan.

```

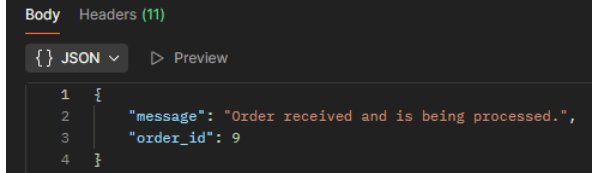
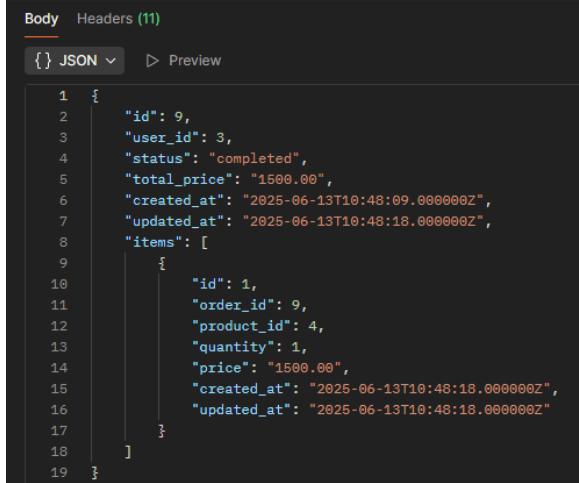
  ▷ Run Service
160  rabbitmq:
161    image: rabbitmq:3-management
162    container_name: rabbitmq
163    ports:
164      - "5672:5672" # AMQP port
165      - "15672:15672" # Management UI
166    environment:
167      RABBITMQ_DEFAULT_USER: guest
168      RABBITMQ_DEFAULT_PASS: guest
169    networks:
170      - laravel-net
```

3.3. Implementasi REST API (Postman)

No.	METHOD	Keterangan	Endpoint	Input	Output
PRODUCT SERVICE					
1	POST	Create a Product	http://localhost:9071/api/products	<pre>{ "name" : "Pensil", "description" : "buat nulis", "price" : "1500", "stock" : "100" }</pre>	<div>Body Headers (11)</div> <div>{ } JSON ▾ ▶ Preview</div> <div><pre>1 { 2 "name": "Pensil", 3 "description": "buat nulis", 4 "price": "1500", 5 "updated_at": "2025-06-13T06:26:35.000000Z", 6 "created_at": "2025-06-13T06:26:35.000000Z", 7 "id": 1 8 }</pre></div>

2	GET	Get All Product	http://localhost:9071/api/products		<div> <div>BodyHeaders (11)</div> <div> <div>{ }JSON ▾▶Preview</div> <pre> 1 { 2 "data": [3 { 4 "id": 1, 5 "name": "Penghapus", 6 "price": "1500.00", 7 "stock": 100 8 }, 9 { 10 "id": 2, 11 "name": "Pensil", 12 "price": "1500.00", 13 "stock": 100 14 } 15] 16 }</pre> </div> </div>
3	GET	Get a Product by ID	http://localhost:9071/api/products/1		<div> <div>BodyHeaders (11)</div> <div> <div>{ }JSON ▾▶Preview</div> <pre> 1 { 2 "data": { 3 "id": 1, 4 "name": "Penghapus", 5 "price": "1500.00", 6 "stock": 100 7 } 8 }</pre> </div> </div>

4	PUT	Update a Product by ID	http://localhost:9071/api/products/3	<pre>{ "name" : "Pensil", "description" : "buat nulis", "price" : "1500", "stock" : "100" }</pre>	<div>Body Headers (11)</div> <div>{ } JSON ▾ ▶ Preview</div> <pre> 1 { 2 "name": "Pensil", 3 "description": "buat nulis", 4 "price": "1500", 5 "updated_at": "2025-06-13T06:26:35.000000Z", 6 "created_at": "2025-06-13T06:26:35.000000Z", 7 "id": 1 8 }</pre>
5	DEL	Delete a Product by ID	http://localhost:9071/api/products/1	<pre>{ "name" : "Pensil", "description" : "buat nulis", "price" : "1500", "stock" : "100" }</pre>	<div>Body Headers (11)</div> <div>{ } JSON ▾ ▶ Preview</div> <pre> 1 { 2 "message": "Product deleted" 3 }</pre>

No.	METHOD	Keterangan	Endpoint	Input	Output
ORDER SERVICE					
1	POST	Make a Order	http://localhost:9072/api/orders	<pre>{ "user_id": 3, "products": [{ "product_id": 4, "quantity": 1 }] }</pre>	 <pre>{ "message": "Order received and is being processed.", "order_id": 9 }</pre>
2	GET	Get History by ID	http://localhost:9072/api/orders/9	<pre>{ "user_id": 1, "products": [{ "product_id": 3, "quantity": 5 }] }</pre>	 <pre>{ "id": 9, "user_id": 3, "status": "completed", "total_price": "1500.00", "created_at": "2025-06-13T10:48:09.000000Z", "updated_at": "2025-06-13T10:48:18.000000Z", "items": [{ "id": 1, "order_id": 9, "product_id": 4, "quantity": 1, "price": "1500.00", "created_at": "2025-06-13T10:48:18.000000Z", "updated_at": "2025-06-13T10:48:18.000000Z" }] }</pre>

3	GET	Get History Order by User ID	http://localhost:9072/api/orders/user/3	<pre>{ "user_id": 1, "products": [{ "product_id": 3, "quantity": 5 }] }</pre>	 <pre>Body Headers (11) {} JSON Preview 1 [2 { 3 "id": 1, 4 "user_id": 3, 5 "status": "completed", 6 "total_price": "75000.00", 7 "created_at": "2025-06-13T10:27:11.000000Z", 8 "updated_at": "2025-06-13T10:27:21.000000Z" 9 }, 10 { 11 "id": 2, 12 "user_id": 3, 13 "status": "completed", 14 "total_price": "75000.00", 15 "created_at": "2025-06-13T10:31:12.000000Z", 16 "updated_at": "2025-06-13T10:31:38.000000Z" 17 }, 18 { 19 "id": 3, 20 "user_id": 3, 21 "status": "completed", 22 "total_price": "75000.00", 23 "created_at": "2025-06-13T10:32:54.000000Z", 24 "updated_at": "2025-06-13T10:33:03.000000Z" 25 }, 26 { 27 "id": 4, 28 "user_id": 3, 29 "status": "completed", 30 "total_price": "30000.00", 31 "created_at": "2025-06-13T10:37:15.000000Z", 32 "updated_at": "2025-06-13T10:37:26.000000Z" 33 }, 34 { 35 "id": 5, 36 "user_id": 3, 37 "status": "completed", 38 "total_price": "7500.00", 39 "created_at": "2025-06-13T10:39:53.000000Z",</pre>
---	-----	------------------------------	---	---	--

Link dokumentasi REST API:

1. Product: <https://documenter.getpostman.com/view/35103814/2sB2x6mrsk>
2. Order: <https://documenter.getpostman.com/view/35103814/2sB2x6mrsj>

2.4. Implementasi GraphQL (Postman)

No.	METHOD	Keterangan	Endpoint	Input	Output
USER SERVICE					
1	POST	Create a User	http://localhost:9070/graphql	<p>Query :</p> <pre>mutation CreateUser(\$name: String!, \$email: String!, \$password: String!) { createUser(input: { name: \$name, email: \$email, password: \$password }) { id name email } }</pre> <p>GraphQL Variables :</p> <pre>{ "name": "Budi Setiawan",</pre>	<pre>1 { 2 "data": { 3 "createUser": { 4 "id": "7", 5 "name": "Budi Setiawan", 6 "email": "budi.setiawan@example.com" 7 } 8 } 9 }</pre>

				<pre> "email": "budi.setiawan@example.com", "password": "password123" } </pre>	
2	GET	Get All User	http://localhost:9070/graphql	Query : <pre> query { users { data { id name email } } } </pre>	<div> <div>Body</div> <div>Headers (8)</div> <div> <div>{}</div> <div>JSON</div> <div>Preview</div> </div> <pre> 1 { 2 "data": { 3 "users": { 4 "data": [5 { 6 "id": "1", 7 "name": "john", 8 "email": "john@gmail.com" 9 }, 10 { 11 "id": "3", 12 "name": "john", 13 "email": "john123@gmail.com" 14 }, 15 { 16 "id": "6", 17 "name": "john", 18 "email": "joh123@gmail.com" </pre> </div>
3	GET	Get a User by ID	http://localhost:9070/graphql	Query : <pre> query { user(id: 1) { id name email } } </pre>	<pre> 1 { 2 "data": { 3 "user": { 4 "id": "1", 5 "name": "john", 6 "email": "john@gmail.com" 7 } 8 } 9 } </pre>

				Graphql Variables :	
4	DEL	Delete a User by ID	http://localhost:9070/graphql	<p>Query:</p> <pre>mutation DeleteUser(\$id: ID!) { deleteUser(id: \$id) { id name } }</pre> <p>Graphql Variables :</p> <pre>{ "id": "7" }</pre>	<pre>1 { 2 "data": { 3 "deleteUser": { 4 "id": "7", 5 "name": "Budi Setiawan" 6 } 7 } 8 }</pre>

Link dokumentasi GraphQL:

1. User : <https://documenter.getpostman.com/view/35103814/2sB2x6mrsk>

3.5. Implementasi GraphQL (Backend)

Adapun contoh hasil penggunaan RabbitMQ, yakni sebagai berikut:

A. Konfigurasi Lighthouse

Melakukan instalasi melalui `composer require nuwave/lighthouse`

B. GraphQL Schema

Publish schema melalui artisan `php artisan vendor:publish --tag=lighthouse-schema`

```
1. User: schema.graphql
2. "A date-time string at UTC, such as 2019-12-03 10:15:30Z."
3. scalar DateTime @scalar(class:
  "Nuwave\\Lighthouse\\Schema\\Types\\Scalars\\DateTime")
4.
5. type Query {
6.   users: [User!]! @paginate(defaultCount: 10)
7.   user(id: ID! @eq): User @find
8. }
9.
10. type User {
11.   id: ID!
12.   name: String!
13.   email: String!
14.   created_at: DateTime!
15. }
16.
17. type Mutation {
18.   createUser(input: CreateUserInput! @spread): User
    @create
19.
20.   # PERBAIKAN: Tambahkan directive @eq pada argumen id
21.   deleteUser(id: ID! @eq): User @delete
22. }
23.
24. input CreateUserInput {
25.   name: String!
26.   email: String!
27.   password: String!
28. }
29.
```

Scalar Type	:	DateTime scalar
Type User	:	User dengan model ID, name, email, created_at
Query Operations	:	Users : Mencari daftar user dengan filter nama dan pagination (default 10 item) User : Mencari user menggunakan ID

Input Types	:	CreateUserInput : Input untuk membuat user baru menggunakan name, email, dan password wajib diisi
Mutation Operations	:	<pre> public function create(\$_, array \$args): User { return User::create(['name' => \$args['name'], 'email' => \$args['email'], 'password' => Hash::make(\$args['password']), // Enkripsi password di sini]); } </pre> <p>Create disini untuk membuat user.</p>

3.6. Implementasi RabbitMQ

A. Alur Konfigurasi

1. Mengatur konfigurasi environment (.env) pada layanan yang diterapkan RabbitMQ (Product Service dan Order Service), kemudian
2. Menambahkan konfigurasi queue pada config/queue.php (Product Service dan Order Service).
3. Membuat Job untuk unit kerja yang akan dieksekusi secara asinkronus.
4. Membuat order menggunakan dispatch pada OrderController.php untuk mengirim Job ke RabbitMQ queue dengan nama (product-stock-update). Dalam hal ini, dispatch hanya ada di OrderController.php karena 'product-stock-update' akan dikirim melalui queue dan diterima di database Product Service.
5. Menjalankan `php artisan queue:work` untuk mengaktifkan antrian.

B. Implementasi

1. Stock awal

```

1  {
2      "data": {
3          "id": 2,
4          "name": "Pensil",
5          "price": "1500.00",
6          "stock": 45
7      }
8  }

```

2. Buat order

```

1  {
2      "message": "Order received and is being processed.",
3      "order_id": 12
4  }

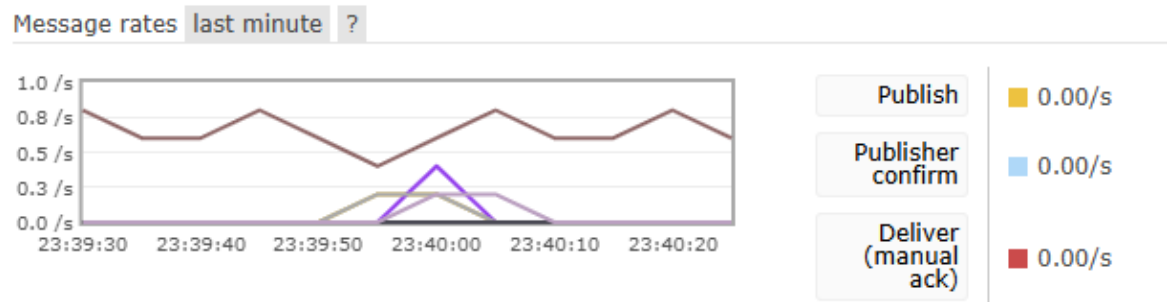
```

3. Stock setelah membuat order

```
Body  Cookies  Headers (11)  Test Results  ↻
[{} JSON]  ▶ Preview  ⚙ Visualize  ▼

1  {
2      "data": {
3          "id": 2,
4          "name": "Pensil",
5          "price": "1500.00",
6          "stock": 40
7      }
8  }
```

4. Antrian di RabbitMQ



3.7. Link GitHub

Adapun link GitHub dari proyek ini, yakni: https://github.com/dirqa41/Tubes-IAE_Microservices-IAE.git

BAB IV

KESIMPULAN DAN SARAN

Melalui implementasi arsitektur microservice yang dibangun di atas kerangka kerja Laravel 10, tim kami berhasil mendemonstrasikan secara efektif bagaimana berbagai teknologi berinteraksi untuk menciptakan sistem yang tangguh dan efisien. Kami secara khusus menunjukkan bahwa REST API adalah fondasi komunikasi sinkron yang vital, memfasilitasi pertukaran data yang mulus dan responsif antar setiap layanan mikro yang independen. Tidak hanya itu, kami juga berhasil membuktikan bahwa GraphQL menawarkan fleksibilitas dan efisiensi *endpoint* yang superior; mengambil data yang berlebihan, GraphQL memungkinkan klien untuk secara presisi menentukan data yang mereka butuhkan dalam satu permintaan saja, secara signifikan mengurangi *overhead* jaringan dan meningkatkan kinerja aplikasi.. Penggunaan RabbitMQ sebagai message broker asinkron secara signifikan meningkatkan performa sistem. Docker memainkan peran penting dalam kontainerisasi, memastikan setiap layanan dapat berjalan secara independen namun tetap terhubung lancar melalui network, yang pada akhirnya menyederhanakan proses deployment dan komunikasi antar microservice. Ke depan, disarankan untuk mengintegrasikan CI/CD pipeline guna mengotomatisasi proses deployment.