Rakamin Academy

API Architecture

Service Repository Pattern
Microservice Pattern

Raih Mimpi #TanpaBatas

Outline Pembelajaran

Rakamin Academy

API architecture,
Service Repository Pattern & Microservice Pattern

- [ ] Service Repository Pattern
- [ ] Microservice Pattern

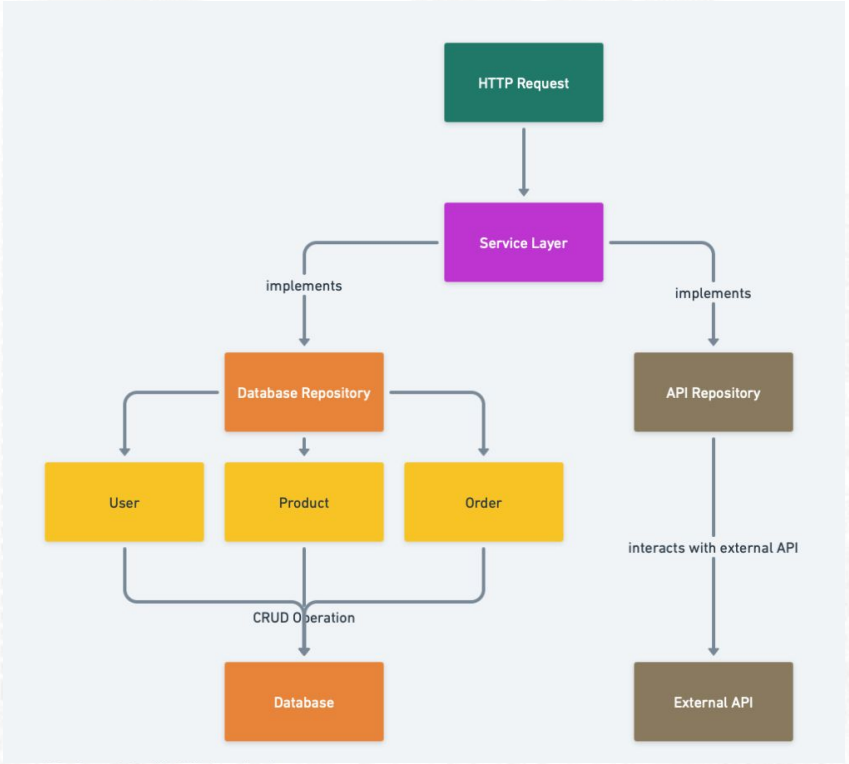
Objektif sesi

- Memahami Konsep Service Repository Pattern
- Memahami Penerapan Microservice Pattern

Service Repository Pattern

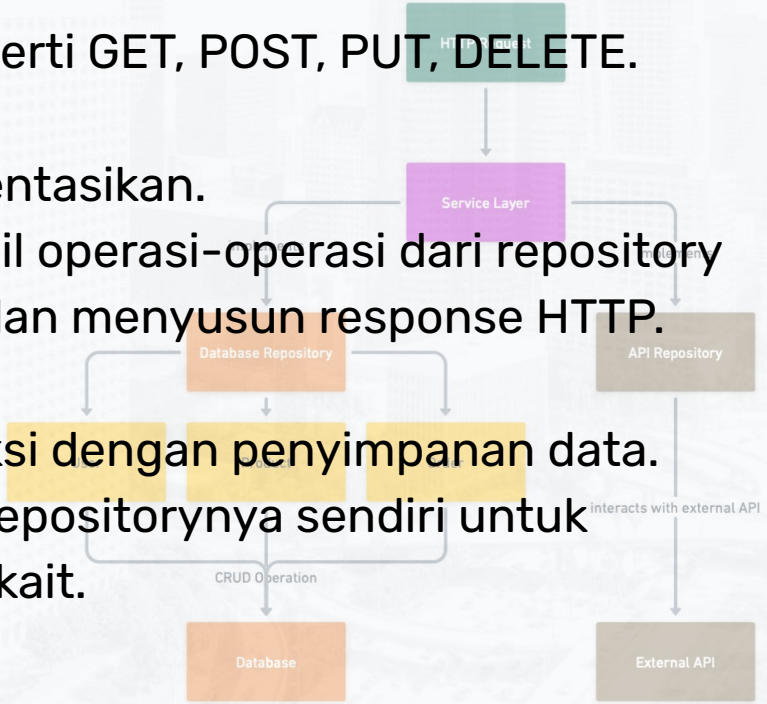
Service Repository Pattern adalah pola yang digunakan dalam pengembangan perangkat lunak untuk memisahkan logika bisnis dari akses data.

Dalam konteks pengembangan RESTful API, pola ini memungkinkan pemisahan antara logika bisnis aplikasi (layanan) dan operasi-operasi database (repository).



Service Repository Pattern: Penjelasan

- ➔ **HTTP Request:**
  Permintaan HTTP masuk dari klien, seperti GET, POST, PUT, DELETE.
- ➔ **Service Layer:**
  Tempat logika bisnis aplikasi diimplementasikan.
  Menerima permintaan HTTP, memanggil operasi-operasi dari repository yang sesuai, melakukan validasi data, dan menyusun response HTTP.
- ➔ **Repository Layer:**
  Berisi operasi-operasi untuk berinteraksi dengan penyimpanan data.
  Setiap entitas dalam aplikasi memiliki repositorynya sendiri untuk mengakses dan memanipulasi data terkait.



Outline Pembelajaran

Rakamin Academy

API architecture,
Service Repository Pattern & Microservice Pattern

- [x] Service Repository Pattern
- [ ] Microservice Pattern

Microservice Pattern: Penjelasan

Konsep utama pada sebuah microservice RESTful API mencakup beberapa aspek penting:

- **Pembagian Fungsionalitas:**
  Didesain untuk melayani satu fungsionalitas atau kebutuhan bisnis tertentu. Misalnya, layanan pengguna, layanan produk, atau layanan pembayaran.
- **Desentralisasi:**
  Tim yang berbeda dapat bekerja secara independen pada setiap layanan, mempercepat pengembangan dan memfasilitasi perubahan tanpa mempengaruhi keseluruhan sistem.
- **Skalabilitas Independen:**
  Dapat di-scale secara independen berdasarkan kebutuhan kinerja atau beban.
- **Resiliensi:**
  Dirancang untuk menjadi tahan terhadap kegagalan dan dapat menangani gangguan tanpa mengganggu operasi keseluruhan sistem. Mekanisme seperti retry otomatis, penanganan kesalahan yang baik, dan manajemen waktu tunggu perlu diterapkan.

Microservice Pattern: Instalasi

➔ Buatlah folder proyek dan instalasi

```bash
mkdir nodejs-microservice
cd nodejs-microservice
npm init -y
npm install express
```

Microservice Pattern: Project Structure

- **services/**: Ini adalah direktori utama yang berisi setiap service.
  - **user-service/**: Direktori layanan pengguna.
  - **product-service/**: Direktori layanan produk.
  - **order-service/**: Direktori layanan pesanan.
- **common/**: Direktori ini berisi kode yang dapat digunakan oleh banyak layanan.
  - **models/**: Berisi definisi skema untuk model MongoDB.
  - **middleware/**: Berisi middleware yang dapat digunakan oleh semua layanan.
  - **config.js**: Berisi konfigurasi yang bersifat global.
- **docker-compose.yml**: Berkas konfigurasi Docker Compose untuk mendefinisikan layanan dan mengelola container Docker.
- **package.json**: Berkas konfigurasi untuk proyek Node.js secara keseluruhan.

project structure

```
nodejs-microservice/
/
├── services/
/   ├── user-service/
/   /   ├── index.js
/   /   ├── package.json
/   /   └── Dockerfile
/   ├── product-service/
/   /   ├── index.js
/   /   ├── package.json
/   /   └── Dockerfile
/   └── order-service/
/       ├── index.js
/       ├── package.json
/       └── Dockerfile
/
├── common/
/   ├── models/
/   /   ├── User.js
/   /   ├── Product.js
/   /   └── Order.js
/   ├── middleware/
/   /   └── errorHandling.js
/   └── config.js
/
├── docker-compose.yml
└── package.json
```

Microservice Pattern: Implementasi

➔ Membuat User Service

index.js

```
// services/user-service/index.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(bodyParser.json());

mongoose.connect('mongodb://mongo:27017/userDB', { useNewUrlParser: true, useUnifiedTopology: true });

const User = mongoose.model('User', {
    name: String,
    email: String
});

app.get('/users', async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (err) {
        res.status(500).send(err);
    }
});

app.post('/users', async (req, res) => {
    try {
        const user = new User(req.body);
        await user.save();
        res.status(201).send(user);
    } catch (err) {
        res.status(400).send(err);
    }
});

app.listen(PORT, () => {
    console.log(`User service is running on port ${PORT}`);
});

```

Microservice Pattern: Implementasi

➔ Membuat Product Service

index.js

```
// services/product-service/index.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();
const PORT = process.env.PORT || 4000;

app.use(bodyParser.json());

mongoose.connect('mongodb://mongo:27017/productDB', { useNewUrlParser: true, useUnifiedTopology: true });

const Product = mongoose.model('Product', {
    name: String,
    price: Number
});

app.get('/products', async (req, res) => {
    try {
        const products = await Product.find();
        res.json(products);
    } catch (err) {
        res.status(500).send(err);
    }
});

app.post('/products', async (req, res) => {
    try {
        const product = new Product(req.body);
        await product.save();
        res.status(201).send(product);
    } catch (err) {
        res.status(400).send(err);
    }
});

app.listen(PORT, () => {
    console.log(`Product service is running on port ${PORT}`);
});

```

Microservice Pattern: Implementasi

➔ Membuat Order Service

index.js

```
// services/order-service/index.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();
const PORT = process.env.PORT || 5000;

app.use(bodyParser.json());

mongoose.connect('mongodb://mongo:27017/orderDB', { useNewUrlParser: true, useUnifiedTopology: true });

const Order = mongoose.model('Order', {
    userId: String,
    productId: String,
    quantity: Number
});

app.get('/orders', async (req, res) => {
    try {
        const orders = await Order.find();
        res.json(orders);
    } catch (err) {
        res.status(500).send(err);
    }
});

app.post('/orders', async (req, res) => {
    try {
        const order = new Order(req.body);
        await order.save();
        res.status(201).send(order);
    } catch (err) {
        res.status(400).send(err);
    }
});

app.listen(PORT, () => {
    console.log(`Order service is running on port ${PORT}`);
});

```

Microservice Pattern: Konfigurasi Model

➔ Konfigurasi scheme model

User.js

```
// common/models/User.js
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    }
});

const User = mongoose.model('User', userSchema);

module.exports = User;

```

Product.js

```
// common/models/Product.js
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    price: {
        type: Number,
        required: true
    }
});

const Product = mongoose.model('Product', productSchema);

module.exports = Product;

```

Order.js

```
// common/models/Order.js
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
    userId: {
        type: String,
        required: true
    },
    productId: {
        type: String,
        required: true
    },
    quantity: {
        type: Number,
        required: true
    }
});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;

```

Microservice Pattern: Error Handling

➔ Konfigurasi error handling

errorHandling.js

```
1  // common/middleware/errorHandling.js
2  function errorHandler(err, req, res, next) {
3      console.error(err.stack);
4      res.status(500).send('Something broke!');
5  }
6
7  module.exports = errorHandler;
8
```

Microservice Pattern: Implementasi

- ➔ Pastikan MongoDB telah berjalan di port default (27017).
- ➔ Menjalankan ketiga layanan ini secara terpisah dengan menjalankan file masing-masing
- ➔ *(node user-service.js, node product-service.js, dan node order-service.js).*
- ➔ Setelah itu, Anda dapat mengakses layanan masing-masing pada port yang sesuai (3000, 4000, dan 5000) untuk berinteraksi dengan basis data MongoDB.

Microservice Pattern: Common Config

➔ Konfigurasi common

config.js

```
// common/config.js
const config = {
    mongodb: {
        userDB: 'mongodb://mongo:27017/userDB',
        productDB: 'mongodb://mongo:27017/productDB',
        orderDB: 'mongodb://mongo:27017/orderDB'
    },
    server: {
        port: process.env.PORT || 3000
    }
};

module.exports = config;

```

Microservice Pattern: Tambahan, Penggunaan Docker

➔ Konfigurasi docker user, product, order

Dockerfile

```
# services/user-service/Dockerfile
FROM node:14

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD [ "node", "index.js" ]

```

Dockerfile

```
# services/product-service/Dockerfile
FROM node:14

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 4000

CMD [ "node", "index.js" ]

```

Dockerfile

```
# services/order-service/Dockerfile
FROM node:14

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5000

CMD [ "node", "index.js" ]

```

Microservice Pattern: Tambahan

➔ Konfigurasi package json, user, product, order

package.json

```
// services/user-service/package.json
{
  "name": "user-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "body-parser": "^1.19.1",
    "express": "^4.17.1",
    "mongoose": "^6.1.6"
  }
}
```

package.json

```
// services/product-service/package.json
{
  "name": "product-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "body-parser": "^1.19.1",
    "express": "^4.17.1",
    "mongoose": "^6.1.6"
  }
}
```

package.json

```
// services/order-service/package.json
{
  "name": "order-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "body-parser": "^1.19.1",
    "express": "^4.17.1",
    "mongoose": "^6.1.6"
  }
}
```

Pastikan penempatan Dockerfile dan package.json di direktori yang sesuai dengan struktur proyek yang telah dijelaskan sebelumnya.
*Dengan ini, Anda dapat menjalankan container Docker untuk masing-masing service menggunakan perintah **docker build** dan **docker run**, atau menggunakan **Docker Compose** untuk mengelola kontainer secara bersamaan.*

Outline Pembelajaran

Rakamin Academy

API architecture,
Service Repository Pattern & Microservice Pattern

- [x] Service Repository Pattern
- [x] Microservice Pattern


Rakamin
Academy
Thank you
Raih Mimpi #TanpaBatas