YUMMY – ONLINE FOOD ORDERING AND DELIVERY SYSTEM

LAB 3 REPORT – LAYERED ARCHITECTURE IMPLEMENTATION (CRUD)

I. Summary

This section presents the practical implementation of the Food Management feature of the Yummy system based on the Layered Architecture defined in Lab 2. Lab 3 focuses on translating the architectural design into actual source code by setting up a clear layered project structure and implementing the Create, Read, Update, and Delete (CRUD) operations for the Food entity, while strictly enforcing the downward flow of control between layers.

In this lab, the group implements the Layered Architecture for the backend of the Yummy application using Node.js and Express to provide RESTful APIs, and Sequelize ORM to interact with a relational database. Each architectural layer is clearly separated in terms of responsibilities: the Presentation Layer handles HTTP requests and responses, the Business Logic Layer encapsulates core business rules related to food items, and the Persistence Layer manages data access operations. This implementation demonstrates how the layered architectural pattern supports maintainability, scalability, and clear separation of concerns in the Yummy system.

II. Technology and tool installation

Tool	Purpose	Installation/Setup Instructions
Flutter SDK	Framework to build Presentation Layer (Mobile App).	Tải và cài đặt Flutter SDK từ trang chủ flutter.dev.
Node.js	Backend source code running environment (API layer, Service, Repository).	Download the LTS version from nodejs.org and install.
Express.js	Web framework để xây dựng Lớp Trình diễn (Backend API).	Run the `npm install express` command in the project directory.
MySQL	Database management system for storing physical data.	Install MySQL Community Server.

Table 3.1. List of supporting tools used in Lab 3

III. Lab Specific Section: Layered Architecture Implementation (CRUD)

1. Project Setup and Model Definition

The backend service is implemented using Node.js with the Express framework.
Sequelize ORM is used to define data models and interact with a MySQL relational database via the mysql2 driver.
Environment-specific configurations are managed using dotenv to ensure sensitive information is not hard-coded.

1.1. Project Initialization

The backend service is initialized using Node.js. Required dependencies for building a RESTful application and interacting with a relational database are installed.

```
bash
# Create the directory for the food service
mkdir -p src/backend/services/food-service

# Navigate to the newly created directory
cd src/backend/services/food-service

# Initialize a new Node.js project
npm init -y
```

```
# Install required dependencies
npm install express sequelize mysql2 dotenv
```

1.2. Layered Folder Structure

```
src/backend/services/food-service/
├── config/           # Database configuration
├── models/           # Sequelize models
├── repositories/     # Data access layer
├── services/         # Business logic layer
├── controllers/      # HTTP controllers
├── routes/           # API route definitions
└── index.js          # Application entry point
```

1.3. Food Model Definition

The database configuration is defined in the config/ folder, which establishes the connection to a MySQL database (`food_service_db`) running on `localhost:3306`.
Sequelize is configured to automatically synchronize models with the database schema, allowing tables to be created if they do not exist.
The Food entity is defined through the Sequelize Model.
File: `models/food.sequelize.js`

```
javascript
const { DataTypes } = require("sequelize");
const { sequelize } = require("../config/database");

const Food = sequelize.define("Food", {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  price: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  restaurantId: {
    type: DataTypes.INTEGER,
    allowNull: false,
    field: "restaurant_id",
  },
}, {
  tableName: "foods",
  timestamps: true,
});

module.exports = Food;
```

2. Persistance Layer (Repositoy)

This layer is implemented using repository classes located in the repositories/ directory.
File: `repositories/food.repository.js`

This repository encapsulates all CRUD operations (create, findById, findAll, update, delete) for the Food entity.

3. Business Logic Layer (Server)

This layer contains application-specific rules, validations, and workflows related to food management.
File: `services/food.service.js`

4. Presentation Layer (Controller/API)

The Presentation Layer exposes RESTful APIs using Express:
Controllers: `controllers/food.controller.js`: Handle HTTP requests and responses, extract parameters, and delegate processing to the Service layer.
Routes: `routes/food.routes.js`:Define REST endpoints and map them to controller methods.

Route Definition
File: `routes/food.routes.js`

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/food.controller');

router.post('/', controller.createFood);
router.get('/', controller.getFoods);
router.get('/:id', controller.getFood);
router.put('/:id', controller.updateFood);
router.delete('/:id', controller.deleteFood);


module.exports = router;
```

File: `index.js`

- The application entry point is defined in this file.
- The Food Service runs on port 3001. When accessed through the API Gateway, requests are forwarded via port 3000.