Ingeniería en Sistemas, Informática y Ciencias de la Computación
Curso: Programación Comercial
Ing. Arturo Monterroso

DOCUMETACION DEL PROYECTO BACKEND DE NODE.JS

Mario Bladimir Gálvez Velásquez

Carné: 202108024

Quetzaltenango, 30 de agosto de 2024

Introducción

Este proyecto utiliza un patrón de diseño MVC (Modelo-Vista-Controlador) con un enfoque en el uso del **Factory Pattern** para la creación dinámica de controladores y servicios. La aplicación está construida con Node.js y Express para el backend, Angular para el frontend y MySQL para la base de datos.

Objetivo

Capacitar al nuevo desarrollador en la implementación y utilización del patrón de diseño Factory Method dentro del proyecto MVC utilizando Node.js, para mejorar la flexibilidad y mantenibilidad del código.

Arquitectura del Proyecto

- **Frontend:** Angular para la interfaz de usuario.
- **Backend:** Node.js con Express para el manejo de rutas y lógica de negocios.
- **Base de Datos:** MySQL para el almacenamiento de datos.

Patrones de Diseño Implementados

- **Factory Pattern:** Utilizado para crear instancias de controladores y servicios de manera flexible y extensible.

El Factory Method define una interfaz para crear un objeto, pero permite a las subclases alterar el tipo de objetos que se crean. En nuestro caso, utilizaremos este patrón para gestionar la creación de servicios en el backend.

Beneficios del Patrón

- **Desacoplamiento:** Separa la lógica de creación de objetos de la lógica de negocio, facilitando la gestión de dependencias.

- **Facilidad de extensión:** Permite agregar nuevos servicios sin modificar el código existente.
- **Mantenibilidad:** Mejora la organización del código, haciéndolo más fácil de leer y mantener.

Estructura del Proyecto

- **/factories**: Contiene ControllerFactory.js que maneja la creación de controladores.
- **/services**: Contiene ServiceFactory.js que maneja la creación de servicios.
- **/controller**: Contiene UserController.js y ProductController.js.
- **index.js**: Punto de entrada de la aplicación que configura el servidor, la base de datos y las rutas.

Configuración del Proyecto

Instalación de Dependencias:

- npm install express mysql2 dotenv
- npm install --save-dev nodemon

Archivo .env de Configuración:

Crea un archivo db.env en la raíz del proyecto con las siguientes variables:

DB_HOST=localhost

DB_USER=root

DB_PASSWORD=

DB_NAME=backnodejs

PORT=3000

Uso del Factory Pattern

- **ControllerFactory.js**: Define un método estático createController que retorna una instancia de controlador basado en el nombre proporcionado.
- **ServiceFactory.js**: Define un método estático createService que retorna una instancia de servicio basado en el tipo proporcionado y la conexión a la base de datos.

Ejemplo de Implementación del ServiceFactory:

ServiceFactory.js

```
// services/ServiceFactory.js

const UserService = require('./UserService');
const ProductService = require('./ProductService');

class ServiceFactory {
    static createService(serviceType, db) {
        switch (serviceType) {
            case 'User':
                return new UserService(db);
            case 'Product':
                return new ProductService(db);
            default:
                throw new Error('Invalid service type');
        }
    }
}

module.exports = ServiceFactory;
```

Modificación del index.js para Usar el ServiceFactory:

index.js

```
require('dotenv').config({ path: './db.env' });
const express = require('express');
const mysql = require('mysql2');
const ControllerFactory = require('./factories/ControllerFactory');
const ServiceFactory = require('./services/ServiceFactory');
const app = express();
const PORT = process.env.PORT || 3000;

console.log('DB_USER:', process.env.DB_USER); // Para verificar que la variable se carga

const db = mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME
});

db.connect((err) => {
    if (err) {
        console.error('Error connecting to the database:', err);
        return;
    }
    console.log('Connected to the MySQL database');
});

// Middleware para pasar la conexión a la base de datos a los controladores
app.use((req, res, next) => {
    req.db = db;
    next();
});

app.use(express.json());

// Usar Factory Pattern para crear instancias de controladores
const userController = new (ControllerFactory.createController('User'))();
const productController = new (ControllerFactory.createController('Product'))();

// Usar Factory Pattern para crear instancias de servicios
const userService = ServiceFactory.createService('User', db);
const productService = ServiceFactory.createService('Product', db);

// Rutas de usuarios
app.get('/users', (req, res) => userController.getAllUsers(req, res));
app.get('/users/:id', (req, res) => userController.getUserById(req, res));

// Rutas de productos
app.get('/products', (req, res) => productController.getAllProducts(req, res));
app.get('/products/:id', (req, res) => productController.getProductById(req, res));
app.post('/products', (req, res) => productController.createProduct(req, res));
app.put('/products/:id', (req, res) => productController.updateProduct(req, res));
app.delete('/products/:id', (req, res) => productController.deleteProduct(req, res));

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

Ejemplo de Rutas y Consultas

- **GET /products**:

  GET http://localhost:3000/products

- **GET /products/**

  GET http://localhost:3000/products/1

- **POST /products**:

  POST http://localhost:3000/products

  Content-Type: application/json

  Body: {

  "name": "Product Name",

  "price": 100,

  "quantity": 10

  }

- **PUT /products/**

  PUT http://localhost:3000/products/1

  Content-Type: application/json

  Body: {

  "name": "Updated Product Name",

  "price": 150,

  "quantity": 20

  }

- **DELETE /products/**

  DELETE http://localhost:3000/products/1

Conclusión

Implementar el **Factory Method** en nuestra arquitectura MVC nos proporciona una manera flexible y escalable de gestionar la creación de servicios, alineada con las mejores prácticas de desarrollo de software. Este enfoque facilita la introducción de nuevos tipos de servicios en el futuro y reduce el acoplamiento entre las distintas partes de la aplicación.