**Architecture en couches**

Cette architecture est souvent appelée **architecture en couches** (ou **Layered Architecture**), utilisée couramment pour structurer des applications web modernes. Elle permet de séparer les responsabilités dans différentes couches pour mieux organiser le code et le rendre plus modulaire, maintenable et extensible.

Voici un aperçu de chaque couche dans cette architecture :

1. **Controller Layer** (Contrôleurs) : Cette couche reçoit les requêtes HTTP depuis les routes et utilise les services pour traiter la logique métier. Les contrôleurs agissent comme des intermédiaires entre la requête de l'utilisateur et le cœur de l'application.
2. **Service Layer** (Services) : Les services contiennent la logique métier. Ils orchestrent les appels vers les repositories et encapsulent les règles de l'application. La couche de service est responsable du traitement des données et applique les règles spécifiques avant de renvoyer les résultats au contrôleur.
3. **Repository Layer** (Dépôts) : Les repositories gèrent l'interaction avec la base de données, en utilisant Sequelize dans notre cas. Ils sont responsables de l'accès aux données, isolant ainsi les requêtes et opérations de la base de données de la logique métier.
4. **Entity Layer (Modèles)** : Les entités (ou modèles) représentent la structure de données, notamment les tables dans la base de données (avec Sequelize). Elles sont les classes qui définissent les objets métiers.
5. **DTO Layer** (Data Transfer Objects) : Les DTOs (objets de transfert de données) encapsulent les données que l'application envoie entre les couches. Ils permettent de standardiser les formats de données en isolant les entités internes des données exposées vers l'extérieur, ce qui peut améliorer la sécurité et la maintenabilité.
6. **View Layer (Vue)** : La couche vue ( avec EJS) gère l'affichage et la présentation des données dans le navigateur. Elle fournit une interface utilisateur pour interagir avec les produits, les catégories et d'autres informations.

## 2. Structure des Dossiers

La structure des fichiers et des dossiers pour ce TP pourrait être la suivante :

```
project/
│
├── config/
│   └── database.js      # Configuration de la base de données
├── controllers/
│   ├── ProductController.js
│   └── CategoryController.js
├── dtos/
│   ├── ProductDTO.js
│   └── CategoryDTO.js
├── entities/
│   ├── Product.js
│   └── Category.js
├── public/
│   └── js/
│       └── script.js    # Pour les appels AJAX
├── repositories/
│   ├── ProductRepository.js
│   └── CategoryRepository.js
├── services/
│   ├── ProductService.js
│   └── CategoryService.js
├── views/
│   ├── products/
│   │   ├── index.ejs
│   │   ├── add.ejs
│   └── categories/
│       ├── index.ejs
│       └── add.ejs
└── app.js               # Point d'entrée de l'application
```

**Étape 1 : Configuration de la base de données**
**npm install  sequelize mysql2 body-parser**
**Sequelize :  est un ORM**
Dans config/database.js :

const { Sequelize } = require('sequelize');

```
const sequelize = new Sequelize('db_name', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql',
});

module.exports = sequelize;
```

**Étape 2 : Définir les Entités (Product et Category)**

**Dans entities/Category.js :**

```
const { DataTypes } = require('sequelize'); const sequelize =
require('../config/database');
const Category = sequelize.define('Category',
{ name: { type: DataTypes.STRING, allowNull: false } });
module.exports = Category;
```

**Dans entities/Product.js :**

```
const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');
const Category = require('./Category');

const Product = sequelize.define('Product', {
  name: { type: DataTypes.STRING, allowNull: false },
  price: { type: DataTypes.FLOAT, allowNull: false }
});

// Relation
Product.belongsTo(Category, { foreignKey: 'categoryId', as: 'category' });
Category.hasMany(Product, { foreignKey: 'categoryId', as: 'products' });

module.exports = Product;
```

**Étape 3 : Définir les DTOs**

**Dans dtos/ProductDTO.js :**

```
class ProductDTO {
   constructor(id, name, price, category) {
      this.id = id;
      this.name = name;
      this.price = price;
      this.category = category ? { id: category.id, name: category.name } : null;
   }
}
module.exports = ProductDTO;
```

**Dans entities/CategoryDTO.js :**

```
class CategoryDTO {
    constructor(id, name) {
        this.id = id;
        this.name = name;
    }
}

module.exports = CategoryDTO;
```

**Étape 4 : Créer les Repositories**

Dans repositories/ProductRepository.js :

```
const Product = require('../entities/Product');

class ProductRepository {
    async createProduct(data) {
        return await Product.create(data);
    }

    async getAllProducts() {
        return await Product.findAll({ include: 'category' });
    }
}

module.exports = new ProductRepository();
```

*Dans repositories/CategoryRepository.js :*

```
const Category = require('../entities/Category');

class CategoryRepository {
    async createCategory(data) {
        return await Category.create(data);
    }

    async getAllCategories() {
        return await Category.findAll();
    }
}

module.exports = new CategoryRepository();
```

**Étape 5 : Créer les Services**

Dans services/ProductService.js :

```
const ProductRepository = require('../repositories/ProductRepository');
const ProductDTO = require('../dtos/ProductDTO');

class ProductService {
   async createProduct(name, price, categoryId) {
      const product = await ProductRepository.createProduct({ name, price, categoryId
});
      return new ProductDTO(product.id, product.name, product.price,
product.category);
   }

   async getAllProducts() {
      const products = await ProductRepository.getAllProducts();
      return products.map(product => new ProductDTO(product.id, product.name,
product.price, product.category));
   }
}

module.exports = new ProductService();
```

**Dans services/CategoryService.js :**

```
const CategoryRepository = require('../repositories/CategoryRepository');
const CategoryDTO = require('../dtos/CategoryDTO');

class CategoryService {
   async createCategory(name) {
      const category = await CategoryRepository.createCategory({ name });
      return new CategoryDTO(category.id, category.name);
   }

   async getAllCategories() {
      const categories = await CategoryRepository.getAllCategories();
      return categories.map(category => new CategoryDTO(category.id,
category.name));
   }
}

module.exports = new CategoryService();
```

**Étape 6 : Créer les Contrôleurs**

**Dans controllers/ProductController.js :**

```
const ProductService = require('../services/ProductService');
const CategoryService = require('../services/CategoryService');
```

```
class ProductController {
    async addProduct(req, res) {
        const { name, price, categoryId } = req.body;
        try {
            const product = await ProductService.createProduct(name, price, categoryId);
            res.status(201).json(product);
        } catch (error) {
            res.status(400).json({ error: error.message });
        }
    }

    async getProducts(req, res) {
        try {
            const products = await ProductService.getAllProducts();
            const categories = await CategoryService.getAllCategories();
            res.render('products/index', { products, categories });
        } catch (error) {
            res.status(500).json({ error: error.message });
        }
    }
}
```

**module.exports = new ProductController();**

**Dans controllers/CategoryController.js :**

```
const CategoryService = require('../services/CategoryService');

class CategoryController {
    async addCategory(req, res) {
        const { name } = req.body;
        try {
            const category = await CategoryService.createCategory(name);
            res.status(201).json(category);
        } catch (error) {
            res.status(400).json({ error: error.message });
        }
    }

    async getCategories(req, res) {
        try {
            const categories = await CategoryService.getAllCategories();
            res.render('categories/index', { categories });
        } catch (error) {
            res.status(500).json({ error: error.message });
        }
    }
}

module.exports = new CategoryController();
```

**Étape 7 : Créer les Contrôleurs**
**Dans routes/ProductRoutes.js :**

```
// routes/productRoutes.js
const express = require('express');
const router = express.Router();

// Exemple de route pour afficher tous les produits
// Route principale pour afficher les produits
router.get('/', async (req, res) => {
   try {
      const products = await productService.getAllProducts();
      const categories = await categoryService.getAllCategories();
      res.render('products/index', { products, categories });
   } catch (error) {
      console.error('Error fetching products:', error.message);
      res.status(500).send('Server Error');
   }
});

// Exemple de route pour ajouter un produit
router.post('/', (req, res) => {
   const { name, price } = req.body;
   // Ici, vous pouvez ajouter la logique pour enregistrer le produit dans la base de
données
   res.redirect('products/index');
});

module.exports = router;
```

**Dans routes/CategoryRoutes.js :**

```
// routes/categoryRoutes.js
const express = require('express');
const router = express.Router();

// Exemple de route pour afficher toutes les catégories
router.get('/', (req, res) => {
   res.render('categoryList');  // Vous pouvez changer 'categoryList' par le nom de
votre vue
});

// Exemple de route pour ajouter une catégorie
router.post('/', (req, res) => {
```

```
    const { name } = req.body;
    // Ajouter la logique pour enregistrer la catégorie
    res.redirect('/categories');
});

module.exports = router;
```

**Étape 8 : Créer les Vues (EJS)**

Dans views/products/index.ejs :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Products</title>
</head>
<body>
    <h1>Products</h1>
    <ul>
        <% products.forEach(product => { %>
            <li><%= product.name %> - $<%= product.price %> (<%= product.category ?
product.category.name : 'No category' %>)</li>
        <% }) %>
    </ul>
    <form action="/products" method="POST">
        <input type="text" name="name" placeholder="Product Name">
        <input type="number" name="price" placeholder="Product Price">
        <select name="categoryId">
            <% categories.forEach(category => { %>
                <option value="<%= category.id %>"><%= category.name %></option>
            <% }) %>
        </select>
        <button type="submit">Add Product</button>
    </form>
</body>
</html>
```

**Étape 8 : Configurer les Routes dans app.js**

```
const express = require('express');
const app = express();
const productRoutes = require('./routes/productRoutes');
const categoryRoutes = require('./routes/categoryRoutes');

app.set('view engine', 'ejs');
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static('public'));

app.use('/products', productRoutes);
app.use('/categories', categoryRoutes);
```

```
app.listen(3000, () => console.log('Server running on port 3000'));
```