Ministry of Higher Education and Scientific Research - University of Mila
Institute of Mathematics and Computer Science - Department of Computer Science
Master 1 STIC – Advanced Web Development - 2025/2026

Practical Works 6 - MVC Architecture & PostgreSQL

Objective: Flat routing is messy. You will restructure your app into the MVC pattern, install an ORM (Sequelize), and connect to a real PostgreSQL database.

Part 1: The MVC Folder Structure

Create three folders in your backend root: /config, /routes, and /controllers.

1. The Router Extraction:

Create /routes/productRoutes.js.

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

// We just map the URL to the Controller. No logic here!
router.get('/', productController.getAllProducts);
// TODO: Map the POST route to productController.createProduct

module.exports = router;
```

Update your server.js to import this router and app.use('/api/products', productRoutes).

Part 2: Connecting to PostgreSQL

1. Open pgAdmin and create a blank database named dzshop.
2. Install the ORM: npm install sequelize pg pg-hstore
3. Create .env and add:
   DATABASE_URL=postgres://postgres:YOUR_PASSWORD@localhost:5432/dzshop

The Database Config (config/database.js):

```
const { Sequelize } = require('sequelize');
require('dotenv').config();

const sequelize = new Sequelize(process.env.DATABASE_URL, {
  dialect: 'postgres',
  logging: false
});

module.exports = sequelize;
```

Part 3: Your Turn - The Product Model & Controller

Task A: Define the Model (/models/Product.js)

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

```
const Product = sequelize.define('Product', {
  title: {
    type: DataTypes.STRING,
    allowNull: false
  },
  // TODO: Add a 'price' field (FLOAT) that cannot be null
  // TODO: Add a 'description' field (TEXT)
});

module.exports = Product;
```

(In server.js, import sequelize and call sequelize.sync({ alter: true }) to create the table).

Task B: The Asynchronous Controller (/controllers/productController.js)

Now that we have a real database, queries take time. You must use async/await.

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

exports.createProduct = async (req, res) => {
  try {
    // 1. Use await Product.create(req.body) to insert the data into PostgreSQL.
    // 2. Save the result in a variable (e.g., newProduct)
    // 3. Send a 201 status with the newProduct
  } catch (error) {
    res.status(500).json({ error: "Failed to create product" });
  }
};

exports.getAllProducts = async (req, res) => {
  // TODO: Write a try/catch block.
  // Inside try, use await Product.findAll() and send the result with status 200.
};
```

Test: Use **Postman** to POST a new product. Then check pgAdmin to visually confirm the data is saved in your hard drive!