**# 13 Object-Relational Mapping (ORM): E-Commerce Back End**

**## Your Task**

Internet retail, also known as **\*\*e-commerce\*\***, is the largest sector of the electronics industry, generating an estimated $29 trillion in 2019. E-commerce platforms like Shopify and WooCommerce provide a suite of services to businesses of all sizes. Due to their prevalence, understanding the fundamental architecture of these platforms will benefit you as a full-stack web developer.

Your task is to build the back end for an e-commerce site by modifying starter code. You’ll configure a working Express.js API to use Sequelize to interact with a MySQL database.

Because this application won’t be deployed, you’ll also need to provide a link to a walkthrough video that demonstrates its functionality and all of the acceptance criteria being met. You’ll need to submit a link to the video and add it to the readme of your project.

**## User Story**

```md

AS A manager at an internet retail company

I WANT a back end for my e-commerce website that uses the latest technologies

SO THAT my company can compete with other e-commerce companies

```

**## Acceptance Criteria**

```md

GIVEN a functional Express.js API

WHEN I add my database name, MySQL username, and MySQL password to an environment variable file

THEN I am able to connect to a database using Sequelize

WHEN I enter schema and seed commands

THEN a development database is created and is seeded with test data

WHEN I enter the command to invoke the application

THEN my server is started and the Sequelize models are synced to the MySQL database

WHEN I open API GET routes in Insomnia Core for categories, products, or tags

THEN the data for each of these routes is displayed in a formatted JSON

WHEN I test API POST, PUT, and DELETE routes in Insomnia Core

THEN I am able to successfully create, update, and delete data in my database

```

**## Mock-Up**

The following animation shows the application's GET routes to return all categories, all products, and all tags being tested in Insomnia Core:

![In Insomnia Core, the user tests “GET tags,” “GET Categories,” and “GET All Products.”.](./Assets/13-orm-homework-demo-01.gif)

The following animation shows the application's GET routes to return a single category, a single product, and a single tag being tested in Insomnia Core:

![In Insomnia Core, the user tests “GET tag by id,” “GET Category by ID,” and “GET One Product.”](./Assets/13-orm-homework-demo-02.gif)

The following animation shows the application's POST, PUT, and DELETE routes for categories being tested in Insomnia Core:

![In Insomnia Core, the user tests “DELETE Category by ID,” “CREATE Category,” and “UPDATE Category.”](./Assets/13-orm-homework-demo-03.gif)

Your walkthrough video should also show the POST, PUT, and DELETE routes for products and tags being tested in Insomnia Core.

**## Getting Started**

You’ll need to use the [MySQL2](https://www.npmjs.com/package/mysql2) and [Sequelize](https://www.npmjs.com/package/sequelize) packages to connect your Express.js API to a MySQL database and the [dotenv](https://www.npmjs.com/package/dotenv) package to use environment variables to store sensitive data.

Use the `schema.sql` file in the `db` folder to create your database with MySQL shell commands. Use environment variables to store sensitive data like your MySQL username, password, and database name.

**### Database Models**

Your database should contain the following four models, including the requirements listed for each model:

\* `Category`

\* `id`

\* Integer.

\* Doesn't allow null values.

\* Set as primary key.

\* Uses auto increment.

\* `category\_name`

\* String.

\* Doesn't allow null values.

\* `Product`

\* `id`

\* Integer.

\* Doesn't allow null values.

\* Set as primary key.

\* Uses auto increment.

\* `product\_name`

\* String.

\* Doesn't allow null values.

\* `price`

\* Decimal.

\* Doesn't allow null values.

\* Validates that the value is a decimal.

\* `stock`

\* Integer.

\* Doesn't allow null values.

\* Set a default value of `10`.

\* Validates that the value is numeric.

\* `category\_id`

\* Integer.

\* References the `Category` model's `id`.

\* `Tag`

\* `id`

\* Integer.

\* Doesn't allow null values.

\* Set as primary key.

\* Uses auto increment.

\* `tag\_name`

\* String.

\* `ProductTag`

\* `id`

\* Integer.

\* Doesn't allow null values.

\* Set as primary key.

\* Uses auto increment.

\* `product\_id`

\* Integer.

\* References the `Product` model's `id`.

\* `tag\_id`

\* Integer.

\* References the `Tag` model's `id`.

**### Associations**

You'll need to execute association methods on your Sequelize models to create the following relationships between them:

\* `Product` belongs to `Category`, and `Category` has many `Product` models, as a category can have multiple products but a product can only belong to one category.

\* `Product` belongs to many `Tag` models, and `Tag` belongs to many `Product` models. Allow products to have multiple tags and tags to have many products by using the `ProductTag` through model.

> **\*\*Hint:\*\*** Make sure you set up foreign key relationships that match the column we created in the respective models.

**### Fill Out the API Routes to Perform RESTful CRUD Operations**

Fill out the unfinished routes in `product-routes.js`, `tag-routes.js`, and `category-routes.js` to perform create, read, update, and delete operations using your Sequelize models.

Note that the functionality for creating the many-to-many relationship for products has already been completed for you.

> **\*\*Hint\*\***: Be sure to look at the mini-project code for syntax help and use your model's column definitions to figure out what `req.body` will be for POST and PUT routes!

**### Seed the Database**

After creating the models and routes, run `npm run seed` to seed data to your database so that you can test your routes.

**### Sync Sequelize to the Database on Server Start**

Create the code needed in `server.js` to sync the Sequelize models to the MySQL database on server start.

**## Grading Requirements**

This homework is graded based on the following criteria:

**### Deliverables: 10%**

\* The GitHub repository containing your application code.

**### Walkthrough Video: 37%**

\* A walkthrough video that demonstrates the functionality of the e-commerce back end must be submitted, and a link to the video should be included in your readme file.

\* The walkthrough video must show all of the technical acceptance criteria being met.

\* The walkthrough video must demonstrate how to create the schema from the MySQL shell.

\* The walkthrough video must demonstrate how to seed the database from the command line.

\* The walkthrough video must demonstrate how to start the application’s server.

\* The walkthrough video must demonstrate GET routes for all categories, all products, and all tags being tested in Insomnia Core.

\* The walkthrough video must demonstrate GET routes for a single category, a single product, and a single tag being tested in Insomnia Core.

\* The walkthrough video must demonstrate POST, PUT, and DELETE routes for categories, products, and tags being tested in Insomnia Core.

**### Technical Acceptance Criteria: 40%**

\* Satisfies all of the preceding acceptance criteria plus the following:

\* Connects to a MySQL database using the [MySQL2](https://www.npmjs.com/package/mysql) and [Sequelize](https://www.npmjs.com/package/sequelize) packages.

\* Stores sensitive data, like a user’s MySQL username, password, and database name, using environment variables through the [dotenv](https://www.npmjs.com/package/dotenv) package.

\* Syncs Sequelize models to a MySQL database on the server start.

\* Includes column definitions for all four models outlined in the homework instructions.

\* Includes model associations outlined in the homework instructions.

**### Repository Quality: 13%**

\* Repository has a unique name.

\* Repository follows best practices for file structure and naming conventions.

\* Repository follows best practices for class/id naming conventions, indentation, quality comments, etc.

\* Repository contains multiple descriptive commit messages.

\* Repository contains quality readme with description and a link to a walkthrough video.

**## Review**

You are required to submit BOTH of the following for review:

\* A walkthrough video demonstrating the functionality of the application and all of the acceptance criteria being met.

\* The URL of the GitHub repository. Give the repository a unique name and include a readme describing the project.

---

© 2021 Trilogy Education Services, LLC, a 2U, Inc. brand. Confidential and Proprietary. All Rights Reserved.