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

Your Task

Internet retail, also known as **e-commerce**, plays a significant role within the electronics industry, as it empowers businesses and consumers alike to conveniently engage in online buying and selling of electronic products. In the latest available data from 2021, the industry in the United States alone was estimated to have generated the substantial amount of US$2.54 trillion, according to the United Nations Conference on Trade and Development. E-commerce platforms like Shopify and WooCommerce provide a suite of services to businesses of all sizes. Due to the prevalence of these platforms, developers should understand the fundamental architecture of e-commerce sites.

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

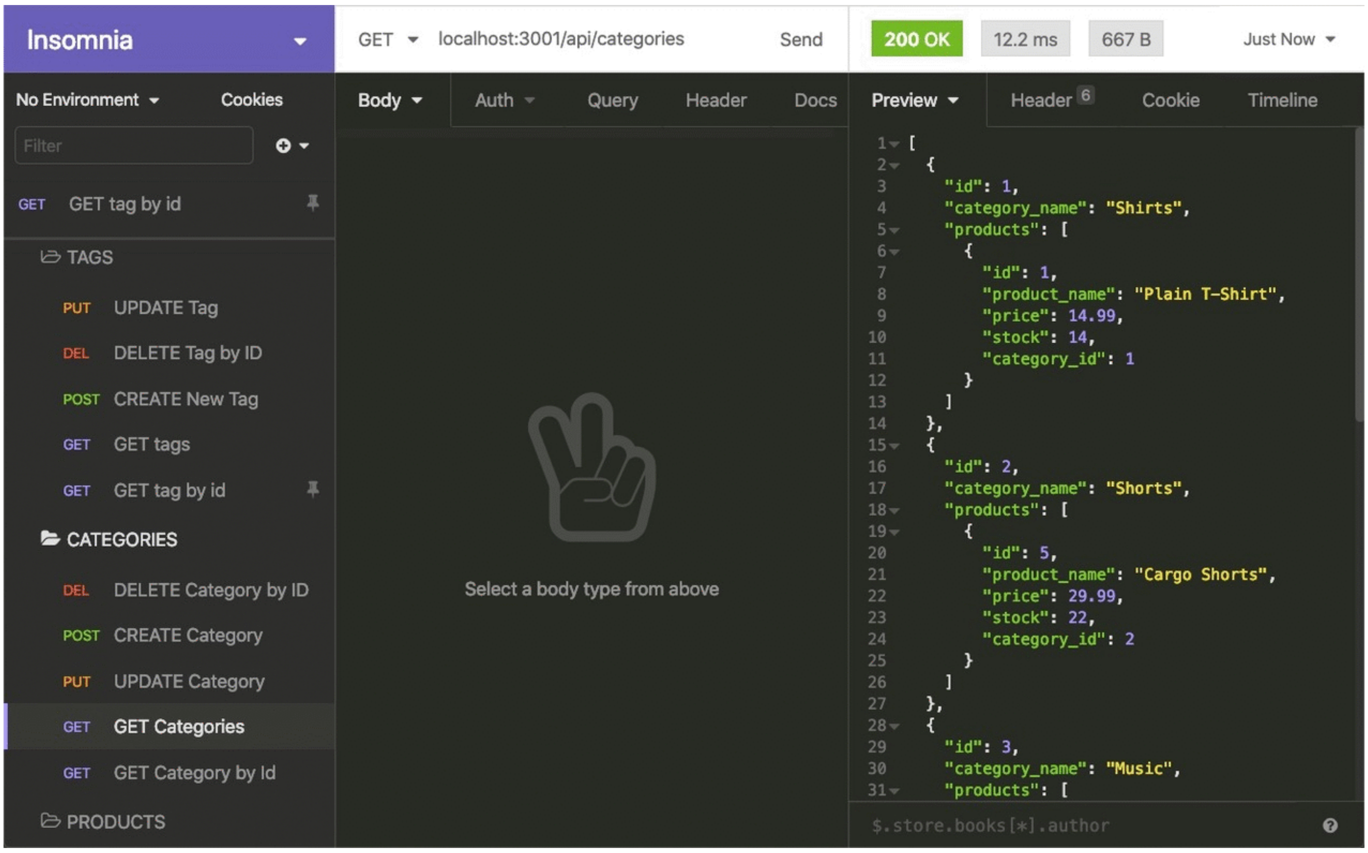
```
AS A manager at an internet retail company
I WANT a back end for my e-commerce website that uses the latest technologies
SO THAT my company can compete with other e-commerce companies
```

Acceptance Criteria

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

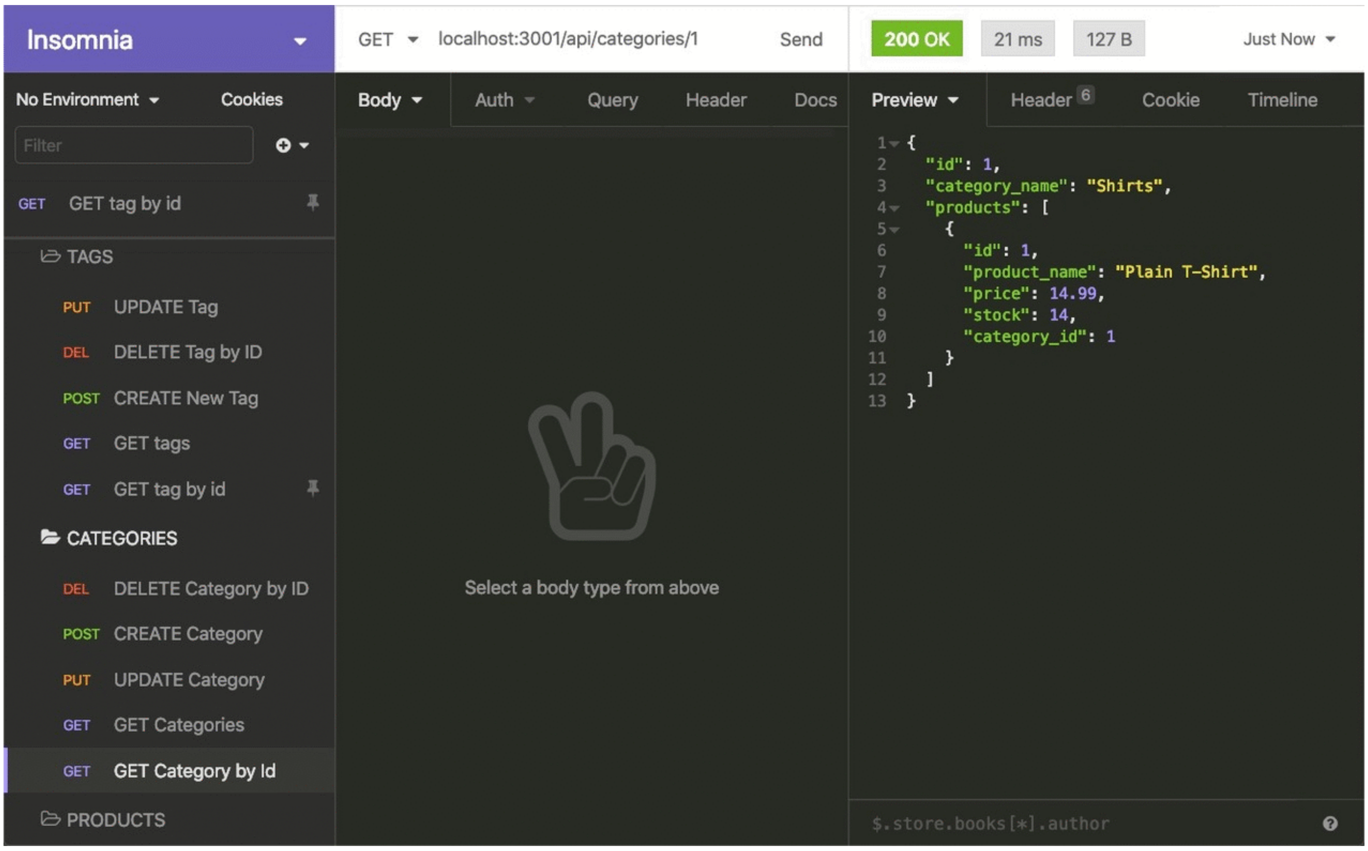
Mock-Up

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



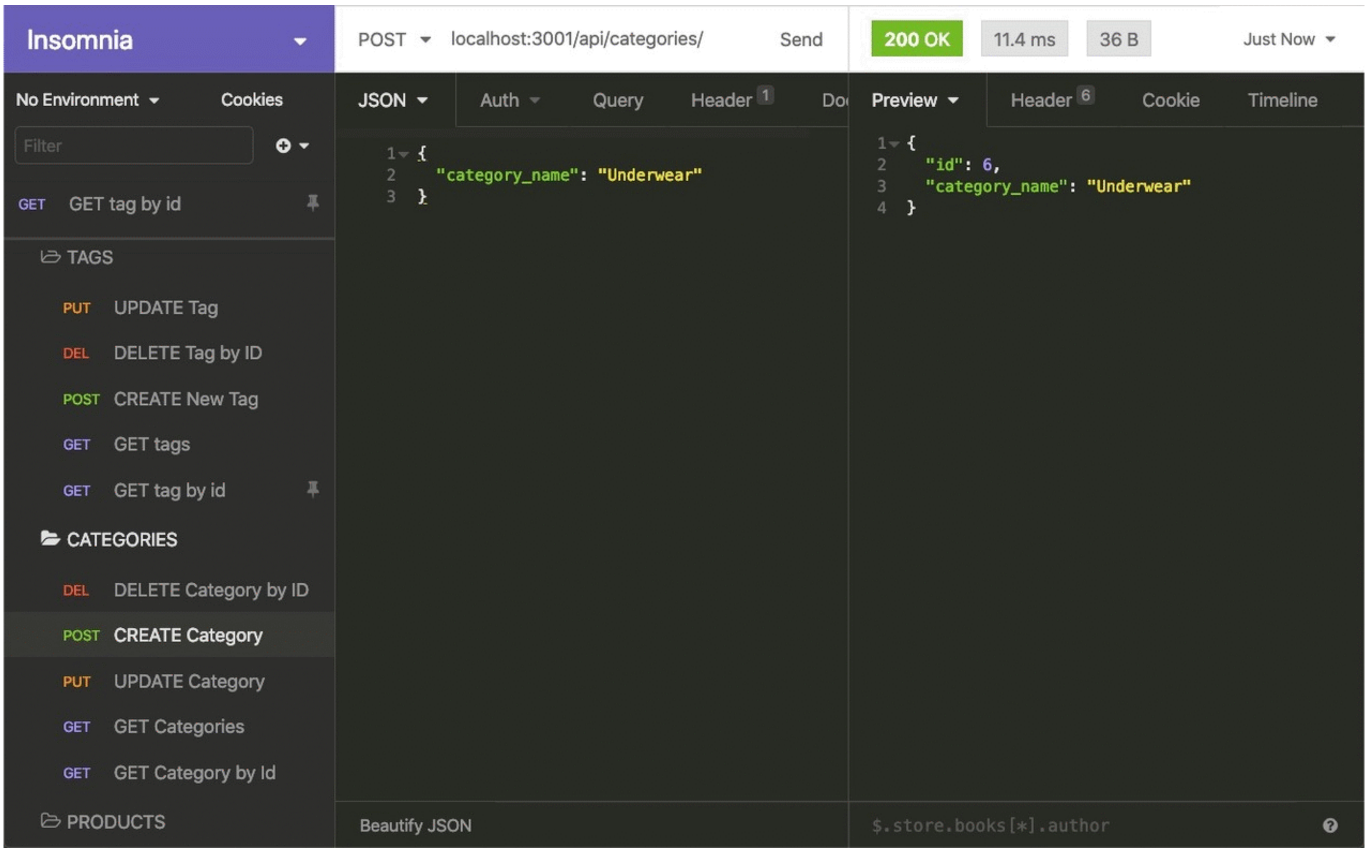
```
[
  {
    "id": 1,
    "category_name": "Shirts",
    "products": [
      {
        "id": 1,
        "product_name": "Plain T-Shirt",
        "price": 14.99,
        "stock": 14,
        "category_id": 1
      }
    ]
  },
  {
    "id": 2,
    "category_name": "Shorts",
    "products": [
      {
        "id": 5,
        "product_name": "Cargo Shorts",
        "price": 29.99,
        "stock": 22,
        "category_id": 2
      }
    ]
  },
  {
    "id": 3,
    "category_name": "Music",
    "products": [
```

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:



```
{
  "id": 1,
  "category_name": "Shirts",
  "products": [
    {
      "id": 1,
      "product_name": "Plain T-Shirt",
      "price": 14.99,
      "stock": 14,
      "category_id": 1
    }
  ]
}
```

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



POST localhost:3001/api/categories/

```
{
  "category_name": "Underwear"
}
```

```
{
  "id": 6,
  "category_name": "Underwear"
}
```

GET GET tag by id
TAGS
PUT UPDATE Tag
DEL DELETE Tag by ID
POST CREATE New Tag
GET GET tags
GET GET tag by id
CATEGORIES
DEL DELETE Category by ID
POST CREATE Category
PUT UPDATE Category
GET GET Categories
GET GET Category by Id
PRODUCTS

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

Getting Started

This Challenge will require a video submission. Refer to the Fullstack Blog Video Submission Guide for additional guidance on creating a video.

You'll need to use the MySQL2 and Sequelize packages to connect your Express.js API to a MySQL database and the 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

> **Note**: If a Challenge assignment submission is marked as "0", it is considered incomplete and will not count towards your graduation requirements. Examples of incomplete submissions include the following:
>
> - A repository that has no code
> - A repository that includes a unique name but nothing else
> - A repository that includes only a README file but nothing else
> - A repository that only includes starter code

This Challenge 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.
- The walkthrough video must demonstrate GET routes for a single category, a single product, and a single tag being tested in Insomnia.
- The walkthrough video must demonstrate POST, PUT, and DELETE routes for categories, products, and tags being tested in Insomnia.

Technical Acceptance Criteria: 40%

- Satisfies all of the preceding acceptance criteria plus the following:
    - Connects to a MySQL database using the MySQL2 and Sequelize packages.
    - Stores sensitive data, like a user’s MySQL username, password, and database name, using environment variables through the dotenv package.
    - Syncs Sequelize models to a MySQL database on the server start.
    - Includes column definitions for all four models outlined in the Challenge instructions.
    - Includes model associations outlined in the Challenge 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.
