NODE & EXPRESS LABS

This unit includes a stand-alone Node.js lab and a stand-alone SQL lab. It also includes a two-part full-stack Express lab.

Full-Stack Express Lab

For the full-stack lab, you will be building a shopping cart that persists to an SQL database. The user will be able to add to, remove from, and modify items in their cart. And all of it will be saved safely in a database.

It is built in two parts.

1. Build the API server with four CRUD endpoints. Add the base of the Angular client application that displays shopping cart items from the API.
2. Add a database to the API server and complete the Angular client application to handle all the add, remove, and modify operations.

NODE LAB

Task: Construct a Node server that will be able to send information from a module on your back-end to your front-end.

What does the application do?

1. When the user navigates to localhost:3000, it will display one random quote from a list.

Build Specifications:

1. The `server.js` file must require an external module that you have created.
2. The external module will consist of an array of quotes, which must be exported.
3. Has to send a random quote from the list to the front-end.
4. When testing in Chrome, you should see it giving you different random quotes when you refresh the page.

NOTE: You can use random quotes, or facts, or whatever you want. (Obviously should be work-appropriate.)

FULL-STACK EXPRESS LAB PART 1

Task: Build a REST API with an Express server. Create a module that contains routes for your front-end to communicate with. Test the endpoints with Postman. Add a front-end to display the shopping cart items from your back-end API.

What does the application do?

1. The back-end REST API provides access to an array of shopping cart items.
2. The back-end will now have routes for GET, POST, PUT, and DELETE which allows our front-end to communicate with our server. Each route will be handling the following functionality:
    a. `GET /cart-items`: returns a JSON array of all items
    b. `POST /cart-items`: for now, log the body to the console. (later, this will add a new item to the list)
    c. `PUT /cart-items/_ID_`: for now, log the _ID_ URL param and the body to the console. (later, this will replace an item in the list)
    d. `DELETE /cart-items/_ID_`: for now, log the _ID_ URL param to the console. (later, this will delete an item from the list.)
3. When the user visits /index.html, an AngularJS application is displayed, which fetches the shopping cart items from the API and displays them beautifully.

Build Specifications:

Server Side

1. Use Express to create your server.
2. Require the module that will contain the routes you have created.
3. Start your server out with a hard-coded array of cart items, each including `id`, `product`, `price`, and `quantity`.
4. Test your endpoints using Postman.
5. Create a `public` folder that will house your front-end files. Adjust the server.js file accordingly to ensure Express is going to be using the public folder.

Client (Angular) Side

6. Build an Angular app within the public folder.
7. Create a `CartService` in Angular. Give it a `getAllItems()` method that uses $http to make a GET request to your /cart-items API.
8. Display the cart items from the service on the page.
9. For this part of the lab, we do NOT yet need to handle POST, PUT, and DELETE on the Angular side.

Bonus:

1. Modify your POST endpoint to add an item to the array.
2. Modify your DELETE endpoint to remove an item from the array, based on the ID.
3. Modify your PUT endpoint to replace an item in the array, based on the ID.

SQL LAB

TASK:
Practice writing SQL statements on the Northwind database.

SETUP:

- In pgAdmin, create a database called northwind.
- Open up a SQL window. Copy-paste and run this file... https://raw.githubusercontent.com/pthom/northwind_psql/master/northwind.sql

DETAILS:
Write SQL queries to do the following tasks. Record these queries in a text document so you can repeat them in class.

1. Select all the records from the "Customers" table.
2. Get distinct countries from the Customers table.
3. Get all the records from the table Customers where the Customer's ID starts with "BL".
4. Get the first 100 records of the orders table.
5. Get all customers that live in the postal codes 1010, 3012, 12209, and 05023.
6. Get all orders where the ShipRegion is not equal to NULL.
7. Get all customers ordered by the country, then by the city.
8. Add a new customer to the customers table. You can use whatever values/
9. Update all ShipRegion to the value 'EuroZone' in the Orders table, where the ShipCountry is equal to France.
10. Delete all orders from `order_details` that have a quantity of 1.
11. Calculate the average, max, and min of the quantity at the `order details` table.
12. Calculate the average, max, and min of the quantity at the `order details` table, grouped by the orderid.
13. Find the CustomerID that placed order 10290 (orders table)

BONUS:

14. Do an inner join, left join, right join on orders and customers tables.
15. Get first names of all employees who report to no one.
16. Get first names of all employees who report to Andrew.

FULL-STACK EXPRESS LAB PART 2

Task: Create a database for your shopping cart. Build out the front end functionality.

What does the application do?

1. The application will now have a consistent database to hold and retrieve information from, allowing the user to keep their shopping cart.
2. The Angular application will allow users to add items, remove items, and update the quantity of an item.

Build Specifications:

Server Side

1. In pgAdmin, create a database called "ExpressShopDB" and a table called "ShoppingCart". The table will have columns: `id`, `product`, `price`, and `quantity`.
2. Construct the pg-connection-pool.js file that will contain all of the information allowing the server to communicate with the database.
3. Adjust your GET, POST, PUT, and DELETE requests in your routes module to include the appropriate queries for each of the four requests.
4. Test your endpoints with Postman to make sure the routing is set up.

Client (Angular) Side

5. Update the `CartService` to call all four of your API endpoints. Add the methods `addItem(item)`, `removeItem(id)`, and `updateItem(item)`.
6. Expand the UI to handle each of these new operations.
    a. Submit the form to add an item.
    b. Click the "x" to remove an item.
    c. Change the quantity.