**Capstone Project: Expresso**

**Build a full back-end CRUD API from scratch for Expresso, a local café.**

**Expresso**

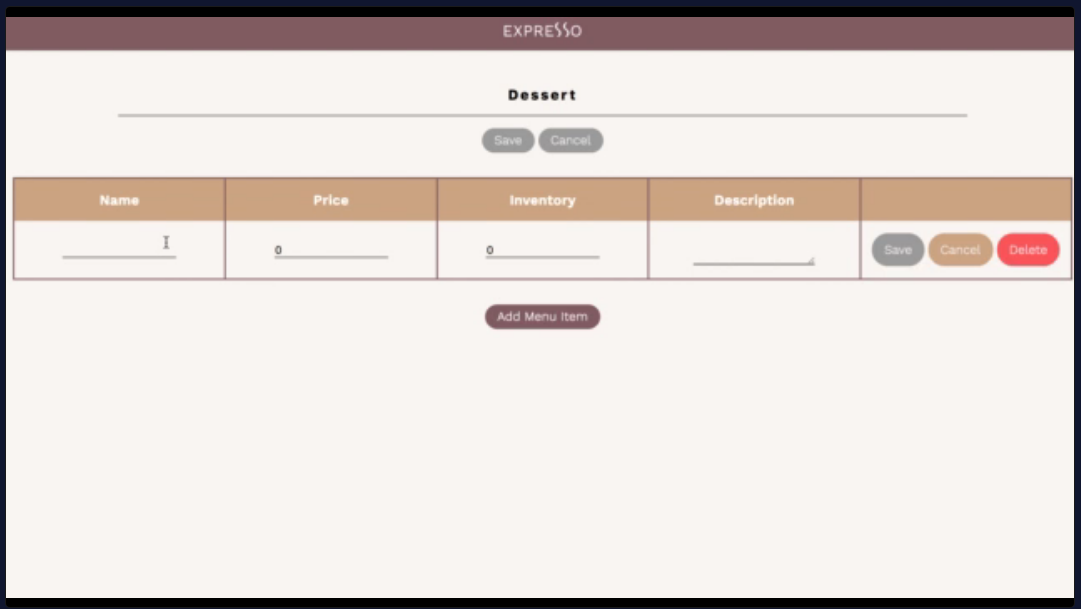
**Project Overview**

In this capstone project, you will build all of the routing and database logic for an internal tool for a coffee shop called Expresso.

The Expresso internal tool should allow users to:

* Create, view, update, and delete menus
* Create, view, update, and delete menu items
* Create, view, update, and delete employees
* Create, view, update, and delete employee’s timesheets

You can view all of this functionality in action in the video below:



**How To Begin**

To start, download the starting code for this project [here](https://content.codecademy.com/PRO/skill-paths/backend-javascript/projects/expresso/capstone-project-2-expresso-start.zip). After downloading the zip folder, double click it to uncompress it and access the contents of this project.

To view your local version of the site, open **index.html** in Google Chrome.

**Implementation Details**

To complete this project, you will need to create the database tables and API routes specified below.

To test this functionality you can run the testing suite and interact with the API via the provided front-end. If you want more data to interact with in the front-end, you can run the **seed.js** file to add data to your database.

We’ve provided an empty **migration.js** file for you to write table creation logic in.

In order for the tests and provided front-end to run properly, you will need to make sure to:

* Create and export your Express app from a root-level file called **server.js**
* Accept and set an optional port argument for your server to listen on from process.env.PORT
* If process.env.PORT is not set, server should run on port 4000 (this is where the provided front-end will make requests to)
* Accept and set an optional database file argument from process.env.TEST\_DATABASE in all Express route files that open and modify your database
* Use the root-level **database.sqlite** as your API’s database
* **Note:** When loading **database.sqlite** in your JavaScript files, sqlite3 will always try to load **database.sqlite** from the root directory path, ./database.sqlite, regardless of where the current file is located. Therefore your code will always be new sqlite3.Database(process.env.TEST\_DATABASE || './database.sqlite') regardless of the file you are writing in

**Database Table Properties**

* **Employee**
  + id - Integer, primary key, required
  + name - Text, required
  + position - Text, required
  + wage - Integer, required
  + is\_current\_employee - Integer, defaults to 1
* **Timesheet**
  + id - Integer, primary key, required
  + hours - Integer, required
  + rate - Integer, required
  + date - Integer, required
  + employee\_id - Integer, foreign key, required
* **Menu**
  + id - Integer, primary key, required
  + title - Text, required
* **MenuItem**
  + id - Integer, primary key, required
  + name - Text, required
  + description - Text, optional
  + inventory - Integer, required
  + price - Integer, required
  + menu\_id - Integer, foreign key, required

**Route Paths and Functionality**

**/api/employees**

* GET
  + Returns a 200 response containing all saved currently-employed employees (is\_current\_employee is equal to 1) on the employees property of the response body
* POST
  + Creates a new employee with the information from the employee property of the request body and saves it to the database. Returns a 201 response with the newly-created employee on the employee property of the response body
  + If any required fields are missing, returns a 400 response

**/api/employees/:employeeId**

* GET
  + Returns a 200 response containing the employee with the supplied employee ID on the employee property of the response body
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response
* PUT
  + Updates the employee with the specified employee ID using the information from the employee property of the request body and saves it to the database. Returns a 200 response with the updated employee on the employee property of the response body
  + If any required fields are missing, returns a 400 response
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response
* DELETE
  + Updates the employee with the specified employee ID to be unemployed (is\_current\_employee equal to 0). Returns a 200 response.
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response

**/api/employees/:employeeId/timesheets**

* GET
  + Returns a 200 response containing all saved timesheets related to the employee with the supplied employee ID on the timesheets property of the response body
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response
* POST
  + Creates a new timesheet, related to the employee with the supplied employee ID, with the information from the timesheet property of the request body and saves it to the database. Returns a 201 response with the newly-created timesheet on the timesheet property of the response body
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response

**/api/employees/:employeeId/timesheets/:timesheetId**

* PUT
  + Updates the timesheet with the specified timesheet ID using the information from the timesheet property of the request body and saves it to the database. Returns a 200 response with the updated timesheet on the timesheet property of the response body
  + If any required fields are missing, returns a 400 response
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response
  + If an timesheet with the supplied timesheet ID doesn’t exist, returns a 404 response
* DELETE
  + Deletes the timesheet with the supplied timesheet ID from the database. Returns a 204 response.
  + If an employee with the supplied employee ID doesn’t exist, returns a 404 response
  + If an timesheet with the supplied timesheet ID doesn’t exist, returns a 404 response

**/api/menus**

* GET
  + Returns a 200 response containing all saved menus on the menus property of the response body
* POST
  + Creates a new menu with the information from the menu property of the request body and saves it to the database. Returns a 201 response with the newly-created menu on the menu property of the response body
  + If any required fields are missing, returns a 400 response

**/api/menus/:menuId**

* GET
  + Returns a 200 response containing the menu with the supplied menu ID on the menu property of the response body
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response
* PUT
  + Updates the menu with the specified menu ID using the information from the menu property of the request body and saves it to the database. Returns a 200 response with the updated menu on the menu property of the response body
  + If any required fields are missing, returns a 400 response
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response
* DELETE
  + Deletes the menu with the supplied menu ID from the database if that menu has no related menu items. Returns a 204 response.
  + If the menu with the supplied menu ID has related menu items, returns a 400 response.
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response

**/api/menus/:menuId/menu-items**

* GET
  + Returns a 200 response containing all saved menu items related to the menu with the supplied menu ID on the menuItems property of the response body
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response
* POST
  + Creates a new menu item, related to the menu with the supplied menu ID, with the information from the menuItem property of the request body and saves it to the database. Returns a 201 response with the newly-created menu item on the menuItem property of the response body
  + If any required fields are missing, returns a 400 response
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response

**/api/menus/:menuId/menu-items/:menuItemId**

* PUT
  + Updates the menu item with the specified menu item ID using the information from the menuItem property of the request body and saves it to the database. Returns a 200 response with the updated menu item on the menuItem property of the response body
  + If any required fields are missing, returns a 400 response
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response
  + If a menu item with the supplied menu item ID doesn’t exist, returns a 404 response
* DELETE
  + Deletes the menu item with the supplied menu item ID from the database. Returns a 204 response.
  + If a menu with the supplied menu ID doesn’t exist, returns a 404 response
  + If a menu item with the supplied menu item ID doesn’t exist, returns a 404 response

**Testing**

A testing suite has been provided for you, checking for all essential functionality and edge cases.

To run these tests, first, open the root project directory in your terminal. Then run npm install to install all necessary testing dependencies (if you haven’t already). Finally, run npm test. You will see a list of tests that ran with information about whether or not each test passed. After this list, you will see more specific output about why each failing test failed.

As you implement functionality, run the tests to ensure you are creating correctly named variables and functions that return the proper values. The tests will additionally help you identify edge cases that you may not have anticipated when first writing the functions.