Hamburger Helper App

This application is a single page, full stack web app, in which a menu of available hamburgers is presented.

User can eat any of the burgers on the menu in the “Burgers Available To Eat” section. “Eating” is accomplished by clicking the “Devour It!” button below the burger’s name. When a burger has been eaten, it is moved to the “Burgers That Have Been Eaten” list.

The user is also able to Add A Burger. To do this, the user simply types in a free-form text description in the “Add A Burger” section of the App. The new burger will then appear in the “Burgers Available To Eat” section. The user may then “eat” this new burger.

Links

Heroku: Hamburger Helper Store
The project is also included in my portfolio at Steve Hulme’s Portfolio.

Motivation

This project is a full-stack web app, using the Model-View-Controller(MVC) design pattern.

File Overview – what does what?

This is a complex project. A short overview of the function and structure of each file will be helpful. We’ll look first at the file organization, then at the driver code, and then at how the MVC design pattern is implemented.

Project Directory/File Structure

```
├── config
│   ├── connection.js
│   └── orm.js
├── controllers
│   └── burgers_controller.js
├── db
│   ├── JAWSDB.sql
│   ├── schema.sql
│   └── seeds.sql
├── models
│   └── burger.js
├── package.json
├── public
│   └── assets
│       ├── css
│       │   └── styles.css
│       └── images
│           ├── Pilgrim_Logo_grey_background_tiny.png
│           ├── emburger.jpg
│           └── emburger_200.jpg
├── server.js
└── views
    ├── index.handlebars
    └── layouts
        └── main.handlebars
```

Driver Code

- server.js
  - Requires the npm modules express, body-parser, method-override.
  - Requires controllers/burgers_controller.js

server.js contains the code that retrieves the routes from burgers_controller, does basic initialization, summons handlebars to create the base html page, and starts listening on the default port (usually 8080). Note that it also establishes for the Express server the / public directory as the source for static information (e.g., images and css).

MVC Design Pattern Implementation

Files that constitute the “Model”

These are the files that interact with the database (in our case, the "burgers" table, as defined in db/schema.sql and seeded by db/seeds.sql).

"Model" files

- models/burger.js. Requires orm.js. This file contains the logic that actually drives the Model portion of our application, that is, it has the logic that implements selectAll records, insertOne record and updateOne record, by calling the "orm" functions in config/orm.js.
- config/orm.js. Requires config/connection.js. This file contains the functions that perform the Create, Retrieve, Update functions of the CRUD model (there is no "Delete" function in this project. It uses selectAll to retrieve all the records in the database; insertOne to insert a single record into the database; updateOne to update a single record in the database.
- config/connection.js controls the connection to the database to the database.

"View" files

- views/layouts/main.handlebars is the HTML shell of our app's homepage. It pulls in the materialize and google fonts stylesheets, as well as the /public/assets/css/styles.css stylesheet that provides custom formatting.
- views/index.handlebars builds the dynamic content of our homepage —- the available and devoured hamburgers read from the database when the app is started.

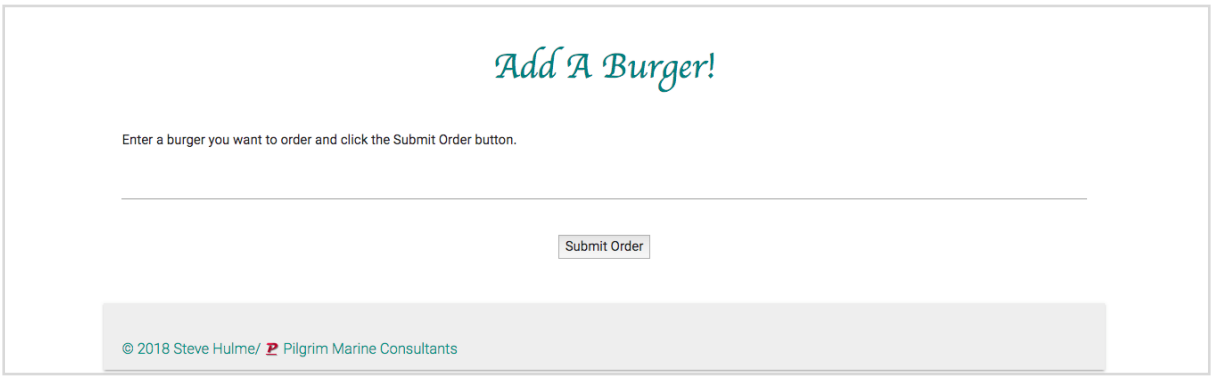
"Controller" files

- controllers/burgers_controller.js. Requires express so that it can use the Express router function. Requires models/burger.js in order to access its database functions in the route code.
    - Sets up a get route at the document root ("/") that retrieves all burger data from the database and then uses handlebars to render that index page.
    - Sets up a post route at the document root that implements the "add burger" capability. When the user types in a new burger and clicks the "Submit Order" button, this route is fired and its logic will insert the new burger into the database by means of the selectAll function provided by models/burger.js.
    - Sets up a post route at the document root ("/") that reads the id number of the burger associated with the "Devour It!" button that was clicked. It then updates the "devoured" field in the corresponding database record. The effect of this logic is to move the associated burger from the "Burgers Available To Eat" section to "Burgers That Have Been Eaten".

Build status

This is version 1.0 of the application. Version 2 will include the replacement of orm.js with Sequelize, as well as refactoring and clean up of all the code.

Screenshots



Tech/framework used

- Node
- Express
- MySql
- Handlebars
- MVC design pattern
- Materialize css framework

Features

This project is a complete, end-to-end implementation of a web app.

- Front End/Browser
  - Handlebars

    - CSS
    - Dynamic web page rendering
- Middleware
    - Express routing
    - Controller utilizes object - relational modeling to interface to the database.
- Server
    - Database

Code Example

The following is the “controller” logic in “controllers/burger_controller.js”.

```
/* require Express and instantiate a variable of type "Router"
*/
var express = require('express');
var router  = express.Router();

// Import the model (burger.js) so we can use its database functions.
var burger  = require('../models/burger.js');

// Create the routes and associated logic
router.get('/', function(req, res) {

  burger.selectAll(function(data) {
    var hbsObject = {
      burgers: data
    };
    res.render('index', hbsObject);
  });
});



router.post('/', function(req, res) {
  burger.insertOne([
    'burger_name',
  ], [
    req.body.burger_name
```

```
  ], function(data) {
    res.redirect('/');
  });
});


router.post('/:id', function(req, res) {
  var condition = 'id = ' + req.params.id;
  burger.updateOne({
    devoured: true
  }, condition, function(data) {
    res.redirect('/');
  });
});

// Export routes for the server.js to use.
module.exports = router;
```

Installation

1. git clone the repository.
2. From the command line,
    1. $ npm install package.json
        1. This will install node and the following required packages in the node_modules directory:
        - "body-parser": "^1.18.2"
        - "express": "^4.16.2",
        - "express-handlebars": "^3.0.0",
        - "method-override": "^2.3.10"
    - "mysql": "^2.15.0”
3. Set up burger database, burgers table, and initial rows in the burgers table. From the command line
    1. $ cd to the db/ subdirectory where the schema.sql and seeds.sql are stored.
    2. $ mysql -u root -p
    3. $ source schema.sql
    4. $ source seeds.sql
    5. $ exit
4. From the command line, cd to the burger directory. Start the server with the following

command:

   - $ node server

5. Point the browser to localhost:8080.

License