- ⏱ 4 hours

Express Application Practice Assessment Challenge

Note: To read this in a rendered view, open your VS Code Command Palatte (using Control+Shift+P on Windows, Command+Shift+P on macOS) and choose "Markdown: Open Preview" or "Markdown: Open Preview to Side".

In this assessment, you are asked to create an Express application. You will create an Express application that

- Has a page that shows a list of pasta dishes
- Has a page that shows a list of pasta dishes by sauce
- Has a page that shows a list of pasta dishes by noodle
- Has a page to create a pasta dish
- Is protected from Cross-Site Request Forgeries

Use the technologies you have used up to this point. They are all installed in the **package.json** for your convenience.

- Express.js
- pg (the library to connect to PostgreSQL), Sequelize, and Sequelize CLI
- CSURF middleware
- Pug.js
- cookie-parser middleware
- body-parser middleware
- nodemon (for development purposes)

A **package.json** file already exists with the dependencies. Please run `npm install` to install those before doing your development and running your tests. Do not remove any dependencies already listed in the **package.json**.

Running the application

You can run your application in "dev" mode. The **nodemon** package is installed and runnable from `npm run dev`.

App Requirements

These are the requirements for the application. Follow them closely.

Read all of the requirements. Determine the data needed to include in your data model.

Please use port 8081 for your Express.js server.

The database

Create a database user with `CREATEDB` priveleges:

- The login username that you must use is "express_practice_B_app"
- The login password that you must use is "password"

Initialize Sequelize in your assessment and replace the use
your `config/config.json` file with the following configuration:

```
{
  "development": {
    "username": "express_practice_b_app",
    "password": "password",
    "database": "express_practice_b_development",
    "host": "127.0.0.1",
    "dialect": "postgres",
    "seederStorage": "sequelize",
    "logging": false
  }
}
```

Remove `logging: false` if you want to see SQL output in your terminal.
Create the `development` database with those configurations.

You will need to generate and run the migrations, models, and seeders.

The data model

You will need to store "pasta" data, "noodle" data, and "sauce" data

You can name the tables and columns however you want but there are certain constraints on the columns:

A pasta dish should have:

- `label` - Text to label the dish but can only be restricted to 50 characters, required
- `description` - Text to describe the dish and can be any amount of characters
- `taste` - A numeric taste "grade" that is a scale with a lowest value of 0, a highest value of 10, and can only support numbers like XX.X

A noodle should have:

- `name` - A string field restricted to 30 characters, required
- `isStuffed` - A boolean field for keeping track of whether the noodle is stuffed or not (e.g. raviolli), default is `false`, required

A sauce should have:

- `name` - A string field restricted to 40 characters, required
- `color` - Text to describe the color of the sauce and restricted to 30 characters, required

Relationships between the data:

- A pasta dish should have a noodle and a sauce.
- A noodle can be in many different pasta dishes.
- A sauce can be in many different pasta dishes.

HINT: The order in which the data is organized now may not be the order you want to migrate the tables! Draw a SQL schema with relationships and foreign keys to make sure you know what table should have what foreign keys and how to structure your migrations!

Make seeds for the noodles and sauces:

Create the following noodles (can include more if you want):

name	isStuffed
Linguini	false
Fettucini	false
Tortellini	true
Ravioli	true
Udon	false

name	isStuffed
Ramen	false

Create the following sauces (can include more if you want):

name	color
Alfredo	white
Bolognese	red
Cheesy Bechamel	white
Garlic Soy	brown
Brown Butter Sage	brown
Red Chili Broth	red

Your main file

You must use the **app.js** file to create and configure your Express application. You must store the instance of your Express.js application in a variable named "app". That is what is exported at the bottom of the **app.js** file.

Add the following middleware to timeout the server if the server is left hanging on a request:

```
app.use((req, res, next) => {
  req.setTimeout(1000, () => {
    res.status(500).end();
  });
  res.setTimeout(1000, () => {
    res.status(500).end();
  });
  next();
});
```

Set up your CSRF middleware to use cookies.

The route "GET /pasta/create"

This page shows a form in which a visitor can add a new pasta. The form must have

- a method of "post"
- an action of "/pasta/create"

In the form, you should have these inputs with the provided name:

Field HTML name	Field type	Constraints	Default values
label	single-line text	required	
description	multi-line text		
taste	number		
noodleId	dropdown	required	One of the pre-defined noodles
sauceId	dropdown	required	One of the pre-defined sauces
_csrf	hidden		The value provided by the CSURF middleware

You should also have a submit button.

The route "POST /pasta/create"

The post handler should validate the data from the HTTP request. If everything is fine, then it should create a new pasta and redirect to the route "/".

Remember, all of the data constraints for this assessment can be handled by the database with the `allowNull` and `unique` flags in your migrations. You **do not** need to use form validations in this project. They are good to have, in real applications, but can require too much time for you to integrate them into this project. Again, you **do not** need to use a form validator, just use database constraints and let the errors turn into 500 status codes by Express.
If the data does not pass validation, then no new record should be created. It is ok to just let Express return an error code of 500 in this case. **Note**: you would not do this in a real application.

The route "GET /"

When someone accesses your application, they should see a list of pastas that are stored in your database. The list should contain:

- The pasta's label
- The pasta's noodle - also links that that go to `/noodle/:id` where `:id` is the id of the noodle
- The pasta's sauce - also links that that go to `/sauce/:id` where `:id` is the id of the sauce
- The pasta's taste factor
- The pasta's description

To create a table in a Pug.js template, you'll use something like the following code. You probably already know this, but it's included for your reference.

```
table
  thead
    tr
      th Header 1
      th Header 2
  tbody
    each thing in things
      tr
        td= thing.property1
        td= thing.property2
```

The route "GET /noodle/:id"

This page shows a list of pastas for that noodle that are stored in your database. The list should contain a similar table of pastas to that on the "GET /" route.

The route "GET /sauce/:id"

This page shows a list of pastas for that sauce that are stored in your database. The list should contain a similar table of pastas to that on the "GET /" route.

Did you find this lesson helpful?