Express Routing, Middleware

Springboard

Express Router

Download Demo Code

- Placing all routes in **app.js** gets messy quickly!
- Express provides feature to place routes elsewhere and use them in **app.js**!

A Router Outside of *app.js*

demo/routing-demo/routes.js

```
const express = require("express");
const router = new express.Router();

const users = [];

/** GET /users: get list of users */

router.get("/", function(req, res) {
  return res.json(users);
});

/** DELETE /users/[id]: delete user, return status */

router.delete("/:id", function(req, res) {
  const idx = users.findIndex(u => u.id === +req.params.id);
  users.splice(idx, 1);
  return res.json({ message: "Deleted" });
});


module.exports = router;
```

Using Our Router in *app.js*

We apply the router to all */users* routes with **app.use**:

demo/routing-demo/app.js

```
//  apply a prefix to every route in userRoutes
app.use("/users", userRoutes);
```

Benefits of the Express Router

- We can make our **app.js** file smaller and more readable.
- We can separate different RESTful resources into their own files:
  - /users has its own router inside **userRoutes.js**
  - /pets has its own router inside **petRoutes.js**

Middleware

What is Middleware?

- It is code that runs in the **middle** of the request / response cycle!
- In Express, middleware are functions that get access to the **req** and **res** objects and can also call the **next** function.
- **express.json()** is an example of middleware
- Our 404 and global error handler are example of middleware

When would you use it?

It opens up the door for separating our code into more logical groupings and providing more robust / abstracted error handling.

- Logging useful information on every request
- Adding a **current_user** for every request (like **g** in Flask!)
- Ensuring that users are authenticated
- Ensuring that a user is authorized to access an endpoint

What does it look like?

- In another file called **middleware.js**

demo/routing-demo/middleware.js

```
function logger(req, res, next) {
  console.log(`Sending ${req.method} request to ${req.path}.`);
  return next();
}
```

Why do we need *next*?

- If we do not include it, we will not make it to the *next* route!
- Notice here we are **not** passing anything to **next**.
- If argument are passed to **next**, Express **always** treats this as an error.

Using our middleware

demo/routing-demo/app.js

```
const middleware = require("./middleware");

app.use(express.json());

// this applies to all requests at all paths
app.use(middleware.logger);
```

Writing middleware to authorize

demo/routing-demo/middleware.js

```
const ExpressError = require("./expressError");

function onlyAllowElie(req, res, next) {
  try {
    if (req.params.name === "Elie") {
      return next();
    } else {
      throw new ExpressError("Unauthorized", 401);
    }
  } catch (err) {
    return next(err);
  }
}

module.exports = { logger, onlyAllowElie };
```

Using our middleware

demo/routing-demo/app.js

```
// route handler with middleware
app.get(
  "/hello/:name",
  middleware.onlyAllowElie,
  function(req, res, next) {
    return res.send("Hello " + req.params.name);
  }
);
```

Using external middleware

- Instead of writing our own logger, we will use a more robust one called **morgan**
- When using external middleware, we follow a simple process:
  - install it - **npm install morgan**
  - require it - **const morgan = require("morgan");**
  - use it - **app.use(morgan("dev"));**
- Once you have set up morgan, take a look at your terminal on each request and you will see the route requested, HTTP verb, and much more.

Summarizing Middleware

- We've already been using built in middleware like **express.json()**
- Middleware are functions that can intercept the request/response cycle
- When using external middleware, make sure to first install, require, and then use.

Integration Tests in Express: Setup

Integration Tests

- Making sure the parts work together
- Essential to have along with unit tests!
- More involved than unit tests

Integration Tests with Supertest

- A library for testing Express applications
- Our tool for integration testing
- Like Flask's test client; can make requests against app in tests
- Docs: https://github.com/visionmedia/supertest

Installing Supertest

```
! npm ! --save-dev supertest
```

Creating a server.js

- To create a test client, we are going to need our **app** variable from **app.js**
- Right now we are combining logic to create the **app** variable and start the server all in one file
- To ensure we don't start the server when we import our **app** variable in our tests, we're going to move out our **app.listen** code into a file called **server.js**
- We're also going to export our **app** variable in **app.js**

What our app.js looks like

demo/supertest-demo/app.js

```
/** general error handler */

app.use((err, req, res, next) => {
  res.status(err.status || 500);

  return res.json({
    error: err.message,
  });
});

module.exports = app;
```

What our server.js looks like

demo/supertest-demo/server.js

```
const app = require("./app")

app.listen(3000, function(){
  console.log("Server starting on port 3000")
})
```

The application we are going to be building

- A simple API for CRUD on cats!
- We're going to be using an array for storage
- We'll move that logic into a file called **fakeDb.js**

demo/supertest-demo/fakeDb.js

```
global.cats = [];

module.exports = cats;
```

What our test setup looks like

demo/supertest-demo/routes/cats-routes.test.js

```
process.env.NODE_ENV = "test";

const request = require("supertest");

const app = require("../app");
let cats = require("../fakeDb");

let pickles = { name: "Pickles" };

beforeEach(function() {
  cats.push(pickles);
});

afterEach(function() {
  // make sure this *mutates*, not redefines, `cats`
  cats.length = 0;
});
```

What should I test?

- Getting all cats
- Getting a single cat
  - What finding successfully looks like
  - What happens when it is not found
- Deleting a cat
  - What deleting successfully looks like
  - What happens when it is not found
- Adding a cat
  - What creating successfully looks like
  - What happens when you create a duplicate cat
  - What happens when you are missing required data

Testing Reading

demo/supertest-demo/routes/cats-routes.test.js

```
/** GET /cats - returns `{cats: [cat, ...]}` */

describe("GET /cats", function() {
  test("Gets a list of cats", async function() {
    const resp = await request(app).get(`/cats`);
    expect(resp.statusCode).toBe(200);

    expect(resp.body).toEqual({cats: [pickles]});
  });
});
```

Testing Creating

demo/supertest-demo/routes/cats-routes.test.js

```
/** POST /cats - create cat from data; return `{cat: cat}` */

describe("POST /cats", function() {
  test("Creates a new cat", async function() {
    const resp = await request(app)
      .post(`/cats`)
      .send({
        name: "Ezra"
      });
    expect(resp.statusCode).toBe(201);
    expect(resp.body).toEqual({
      cat: { name: "Ezra" }
    });
  });
});
```

Testing Updating

demo/supertest-demo/routes/cats-routes.test.js

```
/** PATCH /cats/[name] - update cat; return `{cat: cat}` */

describe("PATCH /cats/:name", function() {
  test("Updates a single cat", async function() {
    const resp = await request(app)
      .patch(`/cats/${pickles.name}`)
      .send({
        name: "Troll"
      });
    expect(resp.statusCode).toBe(200);
    expect(resp.body).toEqual({
      cat: { name: "Troll" }
    });
  });


  test("Responds with 404 if id invalid", async function() {
    const resp = await request(app).patch(`/cats/0`);
    expect(resp.statusCode).toBe(404);
  });
});
```

Testing Deleting

demo/supertest-demo/routes/cats-routes.test.js

```
/** DELETE /cats/[name] - delete cat,
 *  return `{message: "Cat deleted"}` */

describe("DELETE /cats/:name", function() {
  test("Deletes a single a cat", async function() {
    const resp = await request(app).delete(`/cats/${pickles.name}`);
    expect(resp.statusCode).toBe(200);
    expect(resp.body).toEqual({ message: "Deleted" });
  });
});
```

Debugging your tests

- You can always **console.log** inside of your test files
- If you'd like to use the chrome dev tools, write the following:
  - `node --inspect-brk $(which jest) --runInBand NAME_OF_FILE`

Coming Up

- Adding PostgreSQL to Express
- Testing using a Database