

Lesson 3: Learn Express Router



Express Router / POST / UPDATE / DELETE methods

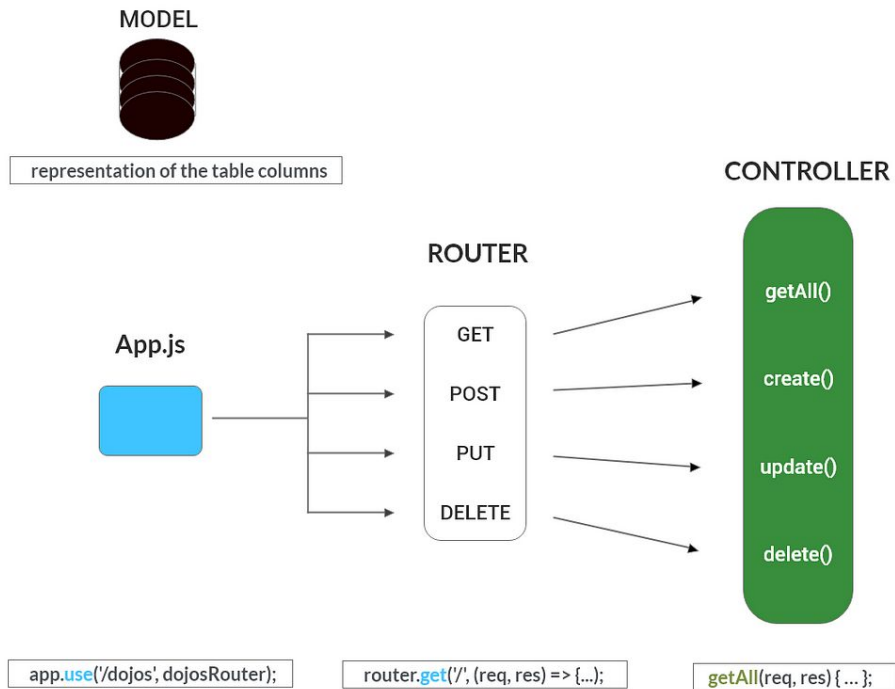
REST API's

REST is a widely adopted philosophy for building API's that can talk to each other in a structured way. It provides a standard way for applications to send and receive data using simple commands like "get," "post," "put," and "delete." Think of REST as a common language that we all unify around for building the structure of our API routes / endpoints.

HTTP Method	Route	Description	Database Method
GET	/dogs	Get all dogs	getAllDogs()
GET	/dogs/:id	Get a specific dog by ID	getDogById(id)
POST	/dogs	Create a new dog	createDog(dogData)
PUT	/dogs/:id	Update a specific dog by ID	updateDogById(id, updatedDogData)
DELETE	/dogs/:id	Delete a specific dog by ID	deleteDogById(id)
GET	/dogs/:id/knee_exam	Get knee exam results for a specific dog	getKneeExamByDogId(dogId)
POST	/dogs/:id/knee_exam	Create new knee exam results for a specific dog	createKneeExamForDog(dogId, examData)
PUT	/dogs/:id/knee_exam	Update knee exam results for a specific dog	updateKneeExamForDog(dogId, examData)
DELETE	/dogs/:id/knee_exam	Delete knee exam results for a specific dog	deleteKneeExamForDog(dogId)

What is Express Router

- Express Router is a built-in middleware function in Express that allows you to group and organize your routes into separate files and directories.
- Using Express Router can help keep your code modular and organized, making it easier to maintain and scale your application over time.
- With Express Router, you can define multiple routes that match a specific URL pattern and HTTP method, and specify a callback function to handle the request and response objects.



Building a route into your API (index.js)

```
const express = require('express');
const app = express();
const exampleRouter = require('./exampleRouter');

// Register the example router as a middleware
app.use('/example', exampleRouter);

// Start the server
app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

Inside the example router (exampleRouter.js)

```
const express = require('express');
const router = express.Router();

// Define the GET endpoint for /example
router.get('/', (req, res) => {
  res.send('This is the example endpoint');
});

// Define the POST endpoint for /example
router.post('/', (req, res) => {
  res.send('This is the example POST endpoint');
});


module.exports = router;
```

Why we use Routes in our API


By grouping similar routes together in separate files, you can reuse the same middleware functions and handlers for multiple routes, making your code more DRY (Don't Repeat Yourself) and easier to maintain.



app.get
HTTP GET



app.put
HTTP PUT



app.delete
HTTP
DELETE



app.post
HTTP POST

Planning our routes for pleb-wallet-backend

Laying out the structure of our API and asking ourselves the question: *“What will my frontend need, and how do I structure it”*

What pieces of data are we working with?

- Users
 - Give us some examples for signup/auth flows
 - allow us to have admin rights that only allow us to spend out of the wallet
 - allow authed users (non admins) to create invoices to pay into the wallet
 - allow non logged in users to see the data / balance but not initiate any actions on the wallet
- Lightning
 - Create an invoice with a certain amount so a user can pay into the wallet (AUTHED)
 - Pay an invoice that we paste into the wallet (ADMIN)
 - Get our lightning wallet balance (ANYONE)
 - Save our paid/received invoices

Our API routes

- Root '/'
 - GET / (welcome message)
- Users '/users'
 - GET / (get all Users)
 - POST /register (add a new user and save them in the database "http://localhost:5500/users/register")
 - POST /login (login an already existing user "http://localhost:5500/users/login")
 - PUT /:id (update a user by their id)
 - DELETE /:id (delete a user by their id)
 - GET /user (get a specific user by their username "http://localhost:5500/users/user")
- Lightning '/lightning'
 - GET / (get all invoices)
 - POST /createInvoice (create an invoice with the amount/memo provided in the request "http://localhost:5500/lightning/createInvoice")
 - POST /payInvoice (Pay the invoice that is sent in the request "http://localhost:5500/lightning/payInvoice")
 - GET /balance (fetch the lightning wallet balance "http://localhost:5500/lightning/balance")

Express “request parameters”

- In Express, we can define routes that include parameters by using a colon (:) followed by the parameter name. For example, /users/:id defines a route that expects an id parameter to be provided in the URL.
- The id parameter can be any string of characters, numbers, or special characters that are valid in a URL. For example, /users/123 or /users/john-doe.
- When a route with a parameter is requested, the value of the parameter is extracted from the URL and made available in the req.params object.

```
app.get('/users/:id', (req, res) => {  
  const userId = req.params.id;  
  // Do something with the user ID, such as retrieve user from database  
  res.send(`User with ID ${userId} was retrieved`);  
});
```

Building our Express routes



Lay it out / leave helpful comments / test each endpoint

Create our routers folder and router files

- Create a new folder called 'routers' in your code editor or by running `mkdir routers` in the root of your project
- Add a file called `usersRouter.js` and `lightningRouter.js` inside the routers folder

Your project structure should now look like this:

```
> node_modules
  routers
    JS lightningRouter.js
    JS usersRouter.js
  JS index.js
  {} package-lock.json
  {} package.json
```

Add our usersRouter endpoints

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

// GET all users
router.get("/", (req, res) => {
  res.status(200).json({ message: "I'm alive!" });
});

// GET user by their username
router.get("/user", (req, res) => {
  const id = req.params.id;

  console.log(id);

  res.status(200).json({ message: "I'm alive!" });
});

// POST a user to register
router.post("/register", (req, res) => {
  const user = req.body;

  console.log(user);

  res.status(201).json({ message: "I'm alive!" });
});
```


Add our lightningRouter endpoints

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

// GET lightning wallet balance
router.get("/balance", (req, res) => {
  res.status(200).json({ message: "I'm alive!" });
});

// GET all invoices from the database
router.get("/invoices", (req, res) => {
  res.status(200).json({ message: "I'm alive!" });
});
```

Add our lightningRouter endpoints

```
// POST required info to create an invoice

router.post("/invoice", (req, res) => {

  const { value, memo } = req.body;

  console.log(value, memo);

  res.status(200).json({ message: "I'm alive!" });

});

// POST an invoice to pay

router.post("/pay", (req, res) => {

  const { payment_request } = req.body;

  console.log(payment_request);

  res.status(200).json({ message: "I'm alive!" });

});

// export our router so we can initiate it in index.js
module.exports = router;
```


Add our new routers to our server in index.js

```
// Imports for our new routers
```

```
const usersRouter = require("./routers/usersRouter");
```

```
const lightningRouter = require("./routers/lightningRouter");
```



```
// Add our routers before server.listen()
```

```
server.use("/users", usersRouter);
```

```
server.use("/lightning", lightningRouter);
```

```
const express = require("express");

const usersRouter = require("./routers/usersRouter" );

const lightningRouter = require("./routers/lightningRouter" );


// Create a new instance of the Express server

const server = express();

// Use the built-in JSON middleware to parse incoming JSON requests

server.use(express.json());


// Set up a route to handle GET requests to the root path

server.get("/", (req, res) => {

  // Send a JSON response with a "message" property set to "I'm alive!"

  res.status(200).json({ message: "I'm alive!" });

});


// Add our routers before server.listen()

server.use("/users", usersRouter);

server.use("/lightning", lightningRouter);


// Set the server to listen on the provided port, or 5000 if no port is specified

const PORT = process.env.PORT || 5000;


server.listen(PORT, () => {

  // Log a message to the console when the server starts listening

  console.log(`Server listening on port ${PORT}`);

});
```

Test all of our new endpoints

- Get server - (GET) <http://localhost:5500>
- Get users - (GET) <http://localhost:5500/users>
- Register user - (POST) <http://localhost:5500/users/register>
- Login user - (POST) <http://localhost:5500/users/login>
- Update user - (PUT) <http://localhost:5500/users/1>
- Delete a user - (DELETE) <http://localhost:5500/users/1>
- Get user by their ID - (GET) <http://localhost:5500/users/1>
- Get all invoices - (GET) <http://localhost:5500/lightning>
- Create an invoice - (POST) <http://localhost:5500/lightning/createInvoice>
- Pay an invoice - (POST) <http://localhost:5500/lightning/payInvoice>
- Get wallet balance - (GET) <http://localhost:5500/lightning>

Resources

- Express routing official documentation -
<https://expressjs.com/en/guide/routing.html>
- Express Routing -
<https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4>
- RESTful Routing in ExpressJS -
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#RESTful_routing_in_Express
- REST APIs: How They Work and What You Need to Know -
<https://blog.hubspot.com/website/what-is-rest-api>
- What is req.params in Express.js? -
<https://www.educative.io/answers/what-is-reqparams-in-expressjs>

Review

- Express routers: We learned how to create and use routers in Express to organize routes for specific parts of our application. By using routers, we can keep our code modular and easy to maintain.
- RESTful APIs: We learned about RESTful APIs and how they allow us to create a standardized interface for interacting with our application's resources. With Express, we can easily create RESTful APIs by defining HTTP methods like GET, POST, PUT, and DELETE for different routes.
- Request parameters: We learned how to handle request parameters in our Express routes. Request parameters can be used to pass data to our server in the URL, allowing us to create dynamic routes that can handle different types of requests.