NodeJS

API BASICS

Learning Outcomes

By the end of this lesson, you should be able to do the following:

- Know what REST stands for.
- Explain the purpose of using REST when structuring an API.
- Detail the REST naming conventions for your API endpoints
- Have a reinforced understanding of the HTTP Methods/Verbs
- Describe the Same Origin Policy
- Explain the purpose of CORS
- Use CORS as middleware in Express (Globally and on a single route)
- Configure CORS to only allow certain origins to access our API
- Explain CORS headers

Introduction

In recent years, a new pattern for developing websites has been gaining popularity. Instead of creating an app that hosts both the database and view templates, many developers are separating these concerns into separate projects, hosting their database on a server (either on something like Heroku or on a VPS like Digital Ocean), then using a service such as GitHub Pages or Netlify to host their frontend.

Organizing your project this way can be beneficial because it allows your project to be more modular. This also allows you to use a single back-end source for multiple front-end applications, such as a website, a desktop app, or a mobile app. Other developers enjoy this pattern because they simply like using front-end frameworks such as React or Vue to create nice front-end-only, single-page applications.

Frontend and backend applications talk to each other using JSON, which you have already encountered if you've gone through our front-end javascript course. So at this point, all you really need to learn is how to get your express application to speak JSON instead of HTML, which fortunately for you is extremely simple! The assignment at the end of this lesson will take you through a tutorial, but essentially all you have to do is pass your information into `res.json()` instead of `res.send()` or `res.render()`. How easy is that?

It is also quite possible to have an express app that both serves views and JSON by using the Express router set up different routes. If you think back to the organization of the routes in our Library Tutorial (here's a link to

set up different routes. If you think back to the organization of the routes in our Library tutorial (here's a link to it). All of our routes were set up in a `catalog` module, so to get the view for our list of books you would access `/catalog/books`. Setting the Library project up to also serve JSON would be as easy as creating a different router module called `api` and then adjusting the controllers so that `/catalog/books` would serve up the HTML list of our books and `/api/books` would serve up the same information as JSON.

REST

Technically it doesn't really matter how you structure your API. Since you are designing it for your own use across your own apps, as long as you know how to get the information you need or document it well it doesn't matter if you have an endpoint named `/api/getAllPostComments/:postid` or `/api/posts/:postid/comments` or anything else. *However*, REST (an acronym for Representational State Transfer) is a popular and common organizational method for your APIs. It can be beneficial to follow a pattern such as REST when organizing your APIs because doing so will make using it that much easier for you in the long run and if you ever want to make your API public, other users will have a better idea of how to get the information they're seeking.

The actual technical definition of REST is a little complicated (you can read about it on wikipedia), but for our purposes, most of the elements (statelessness, cacheability, etc.) are covered by default just by using express to output JSON. The piece that we specifically want to think about is how to **organize our endpoint URIs** (Uniform Resource Identifier). REST APIs are resource based, which basically means that instead of having names like `/getPostComments` or `/saveNewItemInDatabase` we refer **directly to the resource** and then use the HTTP verbs to determine what action we are taking.

Typically this takes the form of 2 URI's per resource, one for the whole collection and one for a single object in that collection, for example, you might get a list of blog-posts from `/posts` and then get a specific post from `/posts/:postid`. You can also nest collections in this way. To get the list of comments on a single post you would access `/posts/:postid/comments` and then to get a single comment: `/posts/:postid/comments/:commentid`.

As mentioned above, you access and manipulate things using the HTTP verbs that you already know about, GET, POST, PUT and DELETE. The usage here is basically what you would expect, but to be clear, let's work through a few examples.

- To get a list of blog posts you would `GET /posts`, so in your express app you would respond to `app.get("/posts")` with a list of all posts.
- To add a new blog-post you would `POST` to `/posts`.
- To update the contents of a blog-post you would `PUT` to the URI of the specific post you're updating. `/posts/:postid`
- To delete a post you would `DELETE /posts/:postid`

CORS

The Same Origin Policy is an important security measure that basically says "Only requests from the same origin (the same IP address or URL) should be allowed to access this API". (Look at the link above for a couple of examples of what counts as the 'same origin'.) This is a big problem for us because we are specifically trying set up our API so that we can access it from different origins, so to enable that we need to set up Cross-origin

resource sharing, or CORS.

Setting up CORS in express is very easy, there's a middleware that does the work for us. This article explains everything you need to do, and the official docs can be found here.

For now, it is acceptable to just allow access from any origin. This makes development quite a bit easier but for any *real* project, once you deploy to a production environment you will probably want to specifically block access from any origin *except* your front-end website. The documentation above explains how to do this.

Assignment

1. This article is a good resource for organizing RESTful APIs.
2. Read and code along with this tutorial on setting up a REST API in Express. This is one of the best Express tutorials we've come across, it also talks about modular code organization, writing middleware, and links to some great extra info at the end.