33% Complete

Project: Mini Message Board

NodeJS Course

Introduction

Let’s take a quick break from the main Express tutorial to practice what we’ve already learned. At this point you should know enough to use Express to make some fun interactive web apps! We’re going to create a super simple message board.

Assignment

1. Use `express-generator` to set up a basic project using whichever templating language you prefer. If you want, you can set it all up manually – it doesn't really take that much longer.
    - Hint: here are links to some of the more popular templating language docs: PUG, EJS, Handlebars

2. Initialize a Git repo in your project directory with

    ```
    git init
    ```

    Create a `.gitignore` file in your project directory that includes `node_modules`.

3. We are going to have 2 routes, the index (`"/"`) and a new-message form (`"/new"`). The generator already created a router for our index, so find that file and open it up. It can be found at `routes/index.js`. There is already a `router.get()` method for `"/"` that should be rendering your index view, so lets add some messages to it.

4. Create an array at the top of your index router called `messages` and put a couple of sample messages inside of it like this:

    ```
    const messages = [
       {
         text: "Hi there!",
         user: "Amando",
         added: new Date()
       },
       {
         text: "Hello World!",
         user: "Charles",
         added: new Date()
       }
    ];
    ```

5. Next, in your index template (in the `"views"` folder) loop through the messages array using whichever templating language you selected and for each one, display the user, text and the date the message was added. Don't forget to make your messages available to your template by including it in the res.render 'locals' object (e.g. `res.render('index', { title: "Mini Messageboard", messages: messages })`).

6. Next let's set up the new message form. In the router add a `router.get()` for the `"/new"` route and point it to a template named `"form"`. In the views directory create your `form` template. Add a heading, 2 inputs (one for the author's name and one for the message text) and a submit button. To have the form make a network request you will need to define it with both a method and an action like so:

    ```
    <form method="POST" action="/new">
       put your inputs and buttons in here!
    </form>
    ```

7. With your form set up like this, when you click on the submit button it should send a POST request to the url specified by the action attribute, so go back to your index router and add a `router.post()` for `"/new"`.
8. In order to get and use the data from your form, you will need to access the contents of your form inside `router.post()` as an object called `req.body`. The individual fields inside the body object are named according to the `name` attribute on your inputs (the value of `<input name="messageText">` will show up as `req.body.messageText` inside the `router.post` function).

9. In your `router.post()` take the contents of the form submission and push them into the messages array as an object that looks something like this:

    ```
    messages.push({text: messageText, user: messageUser, added: new Date()});
    ```

10. At the end of the `router.post()` function use `res.redirect('/')` to send users back to the index page after submitting a new message.
11. At this point, you should be able to visit `/new` (it might be a good idea to add a link to that route on your index page), fill out the form, submit it and then see it show up on the index page!

12. Now you're almost ready to deploy your application on Heroku, but before doing that, you need to modify a few things just to make life easier for your deployment. First, you need to specify the exact version of Node that you're using in your `package.json` file; if you don't remember the version number, just find it using `node -v`. Then, add it to your `package.json` file, so that it will look similar to this:

    ```
    "engines": { "node": "10.x.y" },
    ```

13. Heroku usually requires a `Procfile`, which specifies all the commands that need to run on startup. With node.js, this file isn't obligatory since Heroku searches in the `package.json` file for a start script which is already defined in your app, but it's still good practice to add it to your project. Create it in your root directory, and add this single line to it:

    ```
    web: node ./bin/www
    ```

14. You're finally ready to deploy to Heroku! You can first try it on local, using

    ```
    heroku local web
    ```

    This will run your app locally using Heroku at http://localhost:5000/. Test it, and if everything works fine, you can finally create it:

    ```
    heroku create
    ```

    Stage and commit the changes that you've made since starting this project, and then push it to your Heroku repository with:

    ```
    git push heroku main
    ```

Lesson contents

Solutions:

NodeJS: (Mini Message Board)

Add Solution

34

ding-09

View CodeLive Preview

19

sher_s7