LAB Extra 1

HOSTING

What You Will Learn

- How to host a HTML+CSS+JavaScript site using GitHub Pages
- How to host a Node API using Heroku
- How to host a React+Node service using Heroku

Approximate Time

The exercises in this lab should take approximately 45 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: December 4, 2021

In this lab, you will be hosting three sample web applications on a variety of hosting environments. In the first section, you will host a front-end only site using GitHub Pages. You will then host a Node application using Heroku, a popular Platform-as-a-Service (PaaS). Finally, you will host a combined React front-end and Node back-end site again using Heroku. You do not need to work your way through all four examples: simply work your way through whichever one you are currently interested in.

Preparing Directories

The starting `labExtra1` folder has been provided for you (within the zip folder downloaded from Gumroad). It contains three subfolders: one for the front-end hosting, one for the Node hosting, and the third for the combined react-node hosting.

Front-End Hosting Using GitHub Pages

GitHub provides an easy mechanism for hosting a HTML/JavaScript/CSS site (also known as the JAM stack). You simply need to push your source code to a dedicated GitHub repo and then make the site live.

Exercise Extra1.1 — Hosting using GitHub Pages

1. Create a public repo for the front-end-only site using github.com. Do not that this repo must be Public.
2. Create a GitHub repo for the site within the front-end-only folder. You can do so via the following commands:

```
cd front-end-only
git init
git add *
git commit -m "first commit"
git branch -M main
git remote add origin your-git-url-goes-here
git push -u origin main
```

3. In the repo page on GitHub, click on the Settings link.
4. In the **Code and automation** sidebar, click on the Pages link.
5. Select the branch to be used for the site. If you used the example git commands in Step 2, then select `main` as your branch.
6. Click the Save button.

   You will eventually see a message about your site being ready to be published (see Figure Extra1.1). It can still take a minute or so after the appearance of this message before it actually is available. If you are getting a 404 error message, try waiting a few minutes. If still getting the error, double check that your repo is Public.

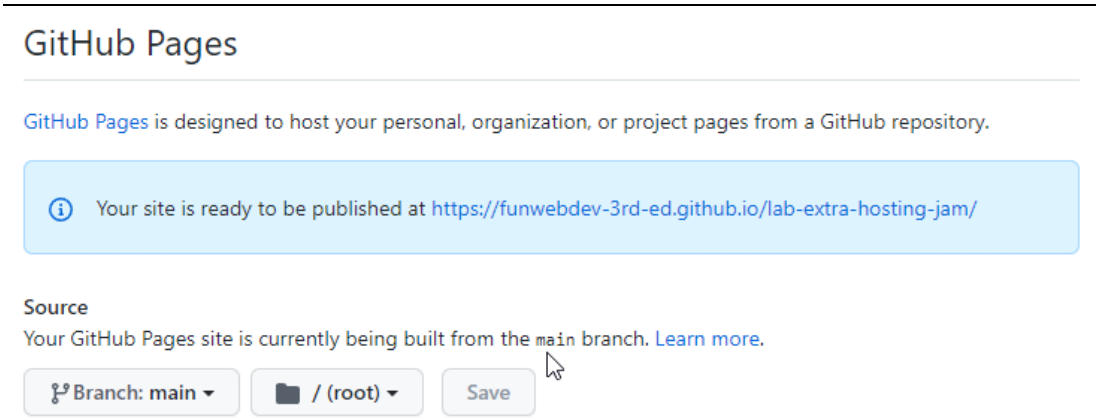

Figure Extra1.1 – Hosting using github pages

HOSTING USING HEROKU

Heroku focuses on the developer experience: it abstracts away many of the complexities involved in setting up a virtual server and the necessary services it requires. Applications are hosted within Heroku dynos, which are virtualized Linux containers. In this example, you will make use of a free shared dyno. While quite limited it is free and will illustrate the ease of deployment ease of this PaaS.

Note: using Heroku will require you create an account on heroku.com and install software on your development computer.

Exercise Extra1.2 — SETTING UP HEROKU

1. Navigate to `https://www.heroku.com` and create a free account.
2. You now need to install the Heroku CLI. Navigate to `https://devcenter.heroku.com/articles/heroku-cli` and choose the appropriate installer.

   You will also need to have git installed on your computer.
3. Verify your installation by entering the following command from the terminal/command window/powershell/etc.

   ```
   heroku -version
   ```
4. If heroku has been installed correctly, then you need to login into the CLI via:

   ```
   heroku login
   ```

   Depending on your installation, you will either login via the CLI or via the web browser.

You are now ready to begin creating applications hosted on Heroku. This typically involves the following starting process:

1. On development machine, `cd` to the root folder for the site.
2. Run: `git init`
3. Run: `heroku create`

This generates a random domain name for your site, and then links the local git repository with this new heroku domain.

NODE API HOSTING USING HEROKU

Exercise Extra1.3 — CREATE THE NODE EXAMPLE

1 Begin by creating a folder named `labExtra1-node` that will contain your project

2 Copy code from the starting `node` folder into `labExtra1-node`. You may want to examine the code to get a sense of what it does (i.e., it serves companies information extracted, for simplicity sake, from the data file `stocks-simple.json`).

3 Navigate to the `labExtra1-node` folder using a terminal and then run the following commands:

```
npm init
npm install -save express
```

4 Test by running node:

```
node app
```

5 Open a browser and try one of the following requests:

```
http://localhost:8080/
http://localhost:8080/public/venice.jpg
http://localhost:8080/api
```

This should work correctly. Notice that the node application also serves static file requests from a folder named public.

6 Stop the node application and add the following to the `package.json` file (some omitted):

```
{
  ...
  "scripts": {
    "start": "node app"
  },
  ...
}
```

This will tell our hosting environment that this command (node app) needs to be run in order to start the application.

7 Test via the following command:

```
npm start
```

Notice that this runs our application.

8 Stop the previous execution (Ctrl-C) and then test via the following command:

```
heroku local web
```

Notice that this also runs our application locally (but at a different port). To test this, you will need to run the same sample routes as step 5 but using a different port.

8 Stop the node application.

Exercise Extra1.4 — Setting Up Heroku

1 See if you are logged in via the following command:

```
heroku auth:whoami
```

2 If you are not already logged in, you will need to login into the Heroku CLI via:

```
heroku login
```

Depending on your installation, you will either login via the CLI or via the web browser.

3 Open and examine the `.gitignore` file. This file indicates the folders and files that will not be uploaded to your git repo.

3 Recall that the Heroku workflow requires you to first commit to git before uploading changes to Heroku. Set up your git with the following commands:

```
git init
git add *
git commit -m "First upload to Heroku"
```

4 You are now ready to create the app in Heroku. Enter the following command:

```
heroku create
```

Recall that this command will create a new project on Heroku and provide you with an auto-generated domain. Mine was https://pacific-caverns-83571.herokuapp.com. Yours will be something different: be sure to make note of it.

5 Now push up the content to Heroku via:

```
git push heroku master
```

6 You should be able to test your Node application on Heroku via:

```
heroku open
```

You could have instead opened a new browser tab and entered the url provided by the heroku create command.

Exercise Extra1.5 — Modifying Your Project

1. Make a small change to `stocks-api.js` (such as the console.log string).
2. Commit the changes to git via:

```
git add *
git commit -m "small sample change"
```

3. Now push up the content to Heroku via:

```
git push heroku master
```

4. You should be able to test your Node application on Heroku via:

```
heroku open
```

5. To view anything output via console.log, you will need to examine the logs on heroku via the command:

```
heroku logs -tail
```

6. You can also examine application logs via the heroku.com web dashboard. Visit heroku.com and log into your account. You should see your just-created heroku project in the dashboard.

NODE AND REACT HOSTING USING HEROKU

Exercise Extra1.6 — CREATE THE REACT CLIENT

1. Begin by creating a folder named `sample-react` that will contain your project
2. In the terminal, navigate to the `sample-react` folder and enter the following command to create the beginning project structure for a React application:

   ```
   npx create-react-app client
   ```

   This will take a few minutes. When it is complete, there will be a folder named `client` with the beginning code for a React application.

3. Verify this works, by entered the following:

   ```
   cd client
   npm start
   ```

   Eventually you will see a message about using http://localhost:3000/ to view the local React app.

4. View http://localhost:3000/ in the browser. This should display the standard default `create-react-app` starting page.

You now have created two applications: the Node back-end and the React application. In the next exercise, you will consume the Node API in the React client.

Exercise Extra1.7 — MODIFYING THE REACT CLIENT

1. Copy the `components` folder from the `react` starting files into the `src` folder generated in the previous exercise.
2. Replace the `app.css` file in `src` folder with the one provided in this lab's starting files.
4. Modify the top of the `App.js` file in the `src` folder as follows:

   ```
   import React, { Component } from 'react';
   import HeaderBar from './components/HeaderBar.js';
   import CompanyList from './components/CompanyList.js';
   import './App.css';
   ```

5 Modify the rest of the `App.js` file as follows:

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { companies: [] };
  }

  // retrieve data from our Node API
  async componentDidMount() {
    try {
      const url = "api";
      const response = await fetch(url);
      const jsonData = await response.json();
      this.setState( {companies: jsonData } );
    }
    catch (error) {
       console.error(error);
    }
  }

  render() {
    return (
      <div className="App">
        <HeaderBar />
        <CompanyList companies={this.state.companies} />
      </div>
    );
  }
}
export default App;
```

Notice that our `App` component consumes the API provided by our Node API application created earlier.

6 Add the following to the `package.json` file (the one in your `client` folder, not the one in the root folder).

```
{
  "name": "client",
  ...
  "dependencies": {
    ...
  },
  "scripts": {
    ...
  },
  "proxy": "http://localhost:8080",
  ...
}
```

This tells React to add this proxy url to any fetches when testing locally. In step 4 above, you added code to fetch from `api/`. When our code is eventually uploaded to the server, this route will make sense. But for testing on our local development machine, our API is `http://localhost:8080/api/`; this proxy settings ensures that this is the case.

8 Test the React app locally. This is going to require having two terminal sessions running.

In the first terminal, run the Node server by navigating to the `sample-react-node` folder and entering the command: **`node app`**.

In the second terminal, run the React app by navigating to the `client` folder and entering the command: **`npm start`**.

If everything worked, you should see something similar to that shown in Figure Extra1-2.

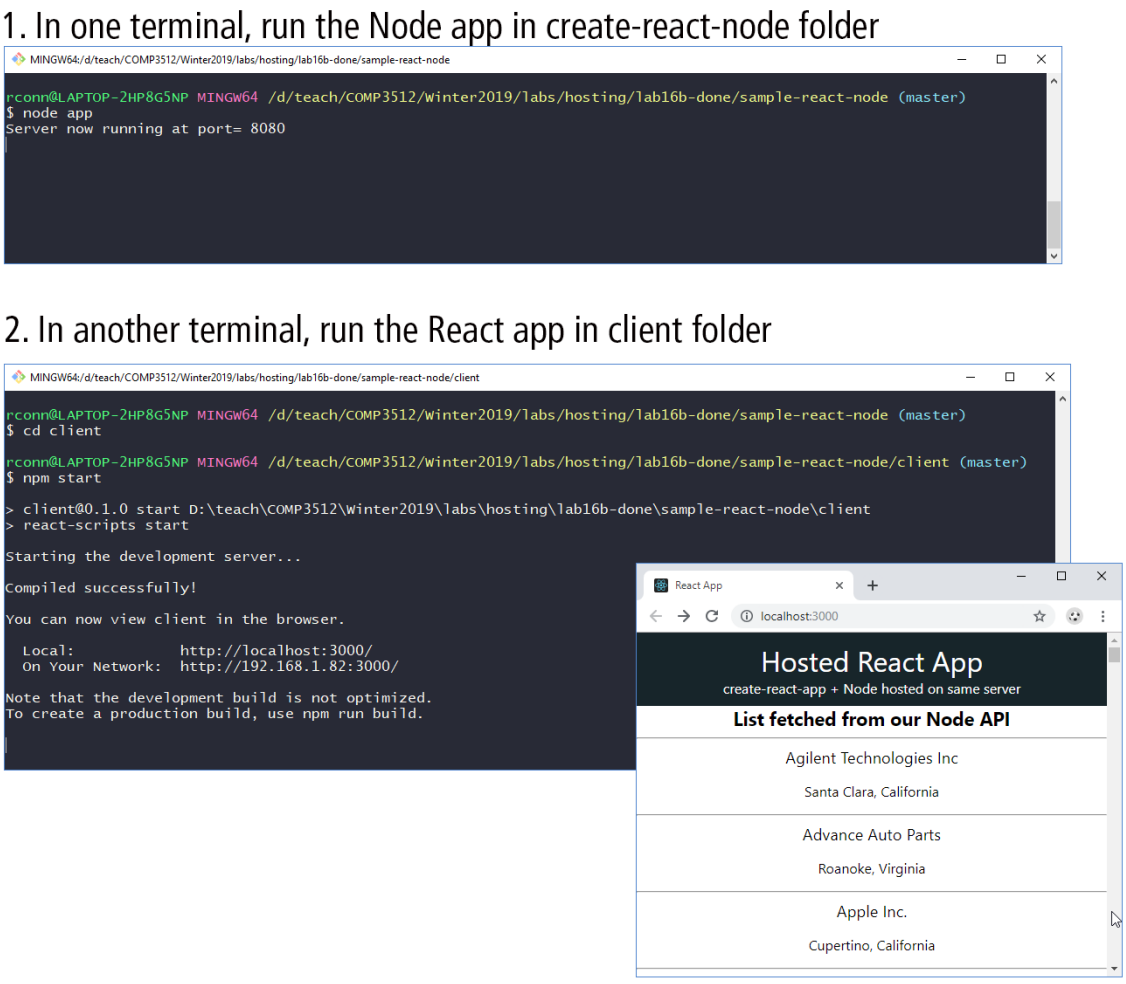

Figure Extra1.2 – Running the React app and Node app locally

Exercise Extra1.8 — Serving the Build Version

1. Navigate to the `sample-react` folder, and run the following command:

   ```
   npm run build
   ```

   This creates a build version of your application.

2. After the build process is finished, copy the `build` folder and paste it into your `labExtra1-node` folder.

3. Edit your `app.js` file as follows:

   ```
   const publicPath = path.join(__dirname,'build');
   ```

   This serves static file requests from your build folder.

4. Test by running `node app` and requesting `http://localhost:8080/`.

   This should display the React page and display the company data from the API.

5. Repeat the steps 1-4 from Exercise Extra1.4 (except use a different commit message).

   Your React site should now be served from your Node application running on Heroku.

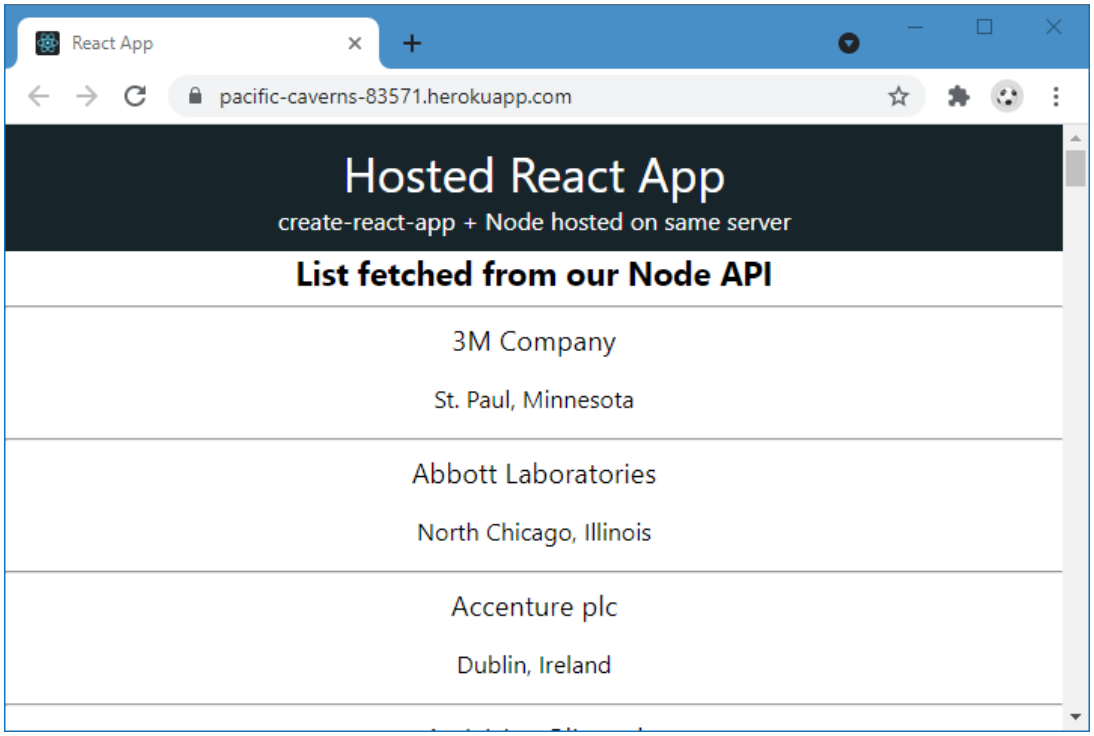

Figure Extra1.3 – Viewing the finished production build on the server