Phase 2 Project Guidelines

(https://github.com/learn-co-curriculum/react-hooks-phase-2-project) (https://github.com/learn-co-curriculum/react-hooks-phase-2-project/issues/new)

Learning Goals

- Build a React single page application from scratch
- Apply your knowledge of components, props and state management
- Incorporate client-side routing
- Use data from an API

Introduction

Now that you've learned the fundamentals of React, it's time to practice and expand your skills by making a React project from scratch!

This assignment is open-ended when it comes to the actual content. You are free to create whatever you'd like, as long as it incorporates the requirements listed in these instructions.

Requirements

1. You must make a single page application (only one `index.html` file) using `create-react-app`
2. Your app should use at least 5 components in a way that keeps your code well organized
3. There should be at least 3 client-side routes using **react-router (https://reactrouter.com/web/guides/quick-start)**
4. Use a `json-server` to create a RESTful API for your backend and make both a `GET` and a `POST` request to the json server. Additionally, you may choose to incorporate data from an external API but it is not required.
   - You should keep your `json-server` data simple: avoid nested data and associations. You'll learn how to work with more complex data in the next two phases. Focus on the frontend for this project.

Stretch Goals

Once you have met the minimum requirements, feel free to explore! These are only the basic requirements — you're free to add on as much stuff as you'd like.

Some ideas for stretch goals:

- Incorporate data from an external API. Use **this list of APIs** **(https://apilist.fun/)** if you need some inspiration!
- Add some styling: you're encouraged to write your CSS from scratch, either by using **styled-components** **(https://styled-components.com/)** or writing CSS files and using id/className

to style your elements. You can also incorporate a UI framework (like **react-bootstrap (https://react-bootstrap.github.io/)**, **semantic-ui (https://react.semantic-ui.com/)** **material-ui (https://material-ui.com/)**) if you prefer.

Setup

For this project, you will need two separate repositories: one for your frontend and one for your backend. This will make it easier to deploy your project later, should you choose to do so.

Frontend Setup

Use `create-react-app` to generate starter code for your your project. Follow the instructions on the **create-react-app (https://create-react-app.dev/docs/getting-started)** site to get started.

Backend Setup

You can use this **json-server template (https://github.com/learn-co-curriculum/json-server-template)** to generate your backend code. Using this template will make it easier to deploy your backend later on.

If you prefer, instead of using the template, you can create a `db.json` file with a structure in the root of your project that looks like this:

```
{
  "toys": [
    {
      "id": 1,
      "name": "Woody",
      "image": "http://www.pngmart.com/files/3/Toy-Story-Woody-PNG-Photos.png",
      "likes": 8
    },
    {
      "id": 2,
      "name": "Buzz Lightyear",
      "image": "http://www.pngmart.com/files/6/Buzz-Lightyear-PNG-Transparent-P
      "likes": 14
    }
  ]
}
```

Then, assuming you have `json-server` installed globally, you can run this command to run the server:

```
$ json-server --watch db.json
```

Whatever top-level keys exist in your `db.json` file will determine the routes available. In the example above, since we have a key of `toys` pointing to an array of toy objects, `json-server` will generate the following routes:

- `GET /toys`
- `POST /toys`
- `GET /toys/:id`
- `PATCH /toys/:id`
- `DELETE /toys/:id`

You can consult the **json-server docs (https://www.npmjs.com/package/json-server)** for more information.

Deploying

When your project is complete, you are encouraged to deploy it! You'll need to deploy your frontend and backend repos to their own standalone servers.

For your backend, if you are using `json-server`, you will need a service capable of running a Node.js server. We recommend using **Heroku (https://devcenter.heroku.com/articles/getting-started-with-nodejs)**.

For your frontend, we recommend using **Netlify (https://docs.netlify.com/#get-started)**; however, there are a number of free services you can use if you'd like to explore alternatives.

Deploying Tips & Tricks

Routing

If you're using React Router, you'll also need to set up a `_redirects` file as specified here:

- **Netlify Redirects (https://docs.netlify.com/routing/redirects/)**

Your redirects file should be placed in the public folder. It look like this:

```
/*    /index.html   200
```

Environment Variables

When working on your app, it's useful to consider which environment you're working on:

- Development: when working locally
- Test: when running tests
- Production: when deployed to server

You'll likely have some variables that change depending on what environment you're working in. For example, after deploying your site to production, you won't be able to access your backend on `localhost` anymore.

To handle these kind of **environment variables**, we can use `.env` files.

`create-react-app` has some tools for working with `.env` files that you can read about here:

Custom Environment Variables (https://create-react-app.dev/docs/adding-custom-environment-variables/)

You can make a `.env.development` and `.env.production` file to keep track of separate environment variables. Note that these files should be in the **root** of your application directory (not in `/src`). For example, you might set up a `.env.development` file with your local development server URL:

```
REACT_APP_API_URL=http://localhost:4000
```

And a `.env.production` file with your deployed backend URL:

```
REACT_APP_API_URL=https://my-awesome-project.herokuapp.com
```

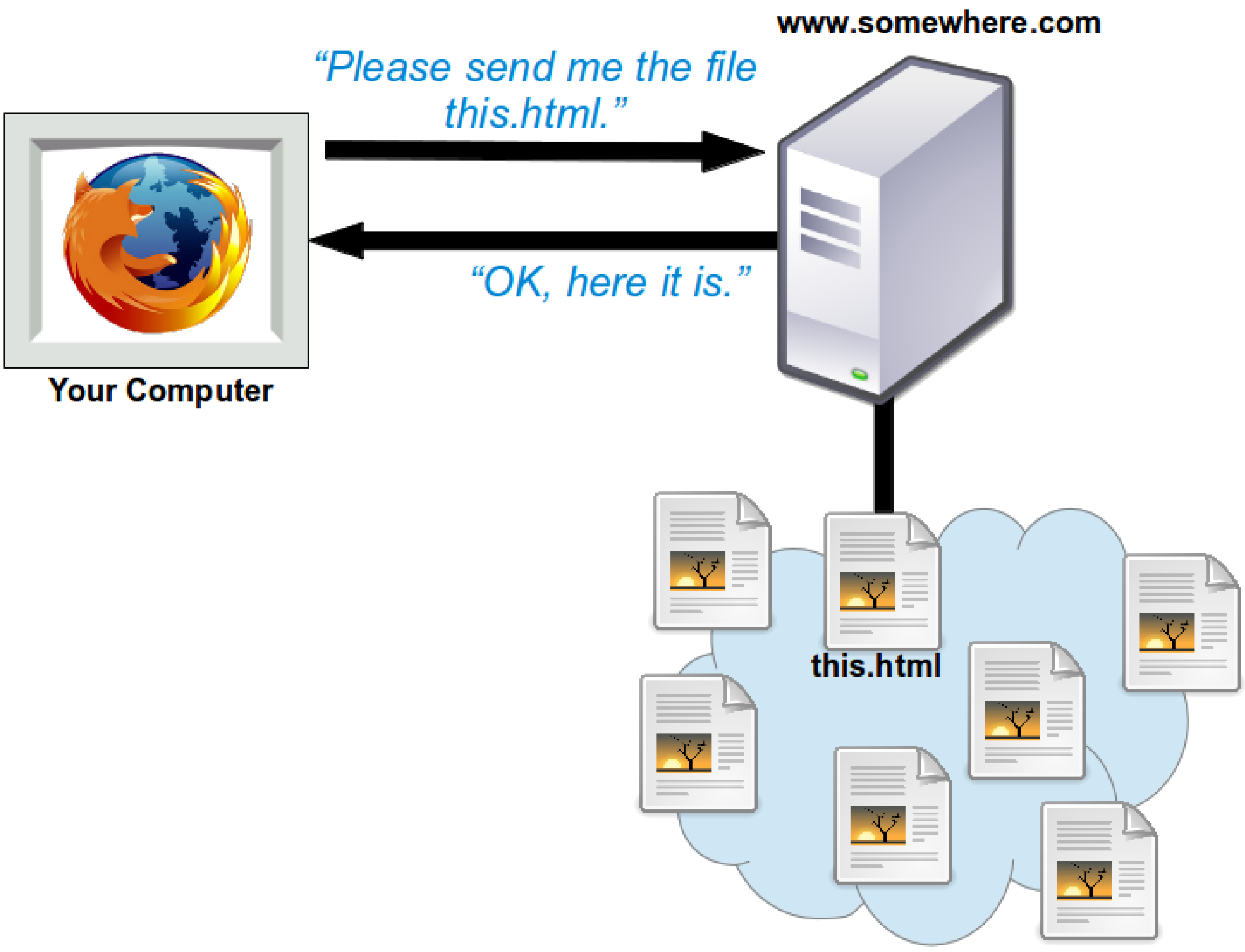
To use these environment variables in your code, you can access them at `process.env.REACT_APP_VARIABLE_NAME`:

```
fetch(`${process.env.REACT_APP_API_URL}/cats`)
  .then((r) => r.json())
  .then(setCats);
```

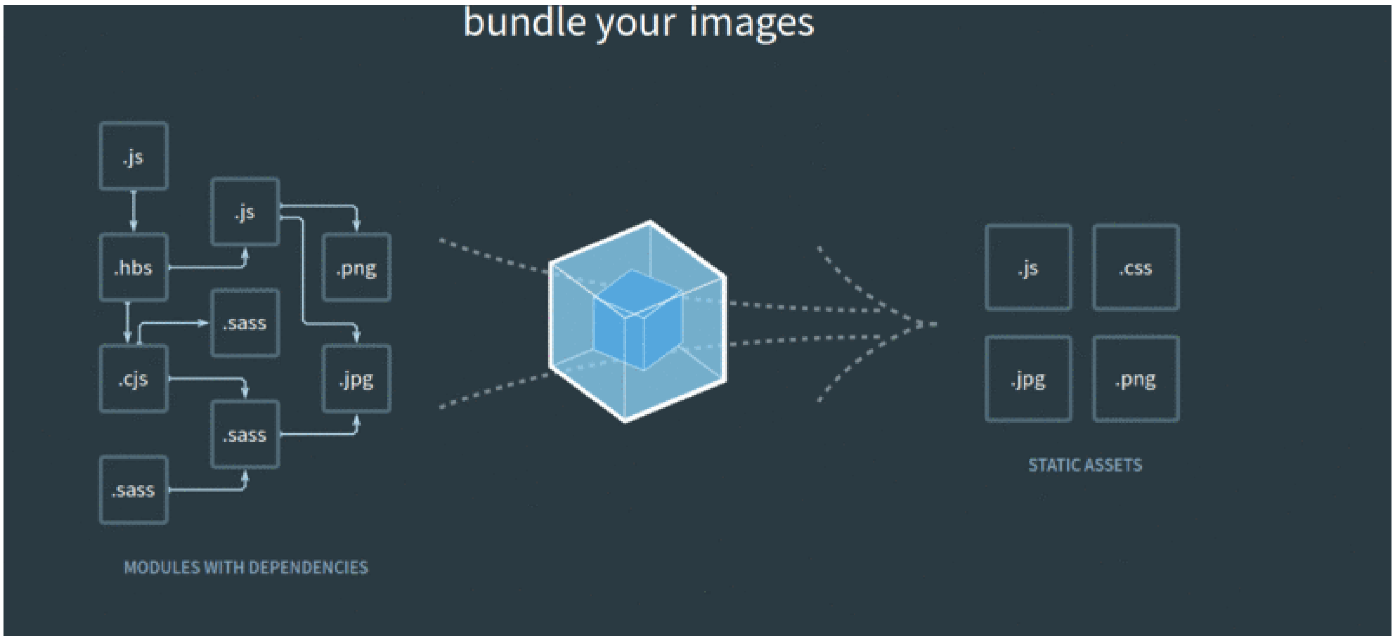
What Happens When I Deploy?

Glad you asked! Deploying your site involves taking the code that lives on your machine, and setting it up to run on someone else's machine.

As you'll recall, our frontend applications are a type of app known as a **Single Page Application**. What that means is that there is only **one** HTML file, along with a handful of JavaScript, CSS, fonts, images, and other static assets. So when our site is deployed, the main thing we need is a server to host all of those files and let other people from around the world access those files with their browsers.



In order to generate those files, `create-react-app` comes with a special `build` script that uses another tool, `webpack`, to take all of our JavaScript, CSS, and other assets from the `src` directory and optimize them by **bundling** (merging files together) and **minifying** (shortening the lines of code) so that the files load as fast as possible.



You can try this out on your own by running `npm run build`. This will create a new directory with your bundled and minified source code!

When you upload your project to Netlify, this `build` script will run automatically on Netlify's server, so that they can host the content for you. Any time you update your code and push the changes up to Netlify, the build script will run again and create a new bundle on the server.

Netlify can be configured to use **Continuous Deployment**, which typically works by connecting your Git repository with Netlify's build process. Then, any time you push up changes to your main branch, your deployed site will automatically update! This makes it very easy to add features even after you've deployed.