TASK

Full stack with React and Express

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

WELCOME TO THE FULL STACK WITH REACT AND EXPRESS TASK!

Now that you are able to create a front-end application using React and a back-end application using Express, the next step is to get your back-end and front-end to work together. This is the focus of this task. By the end of this task, you will be able to build a full stack web application!

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

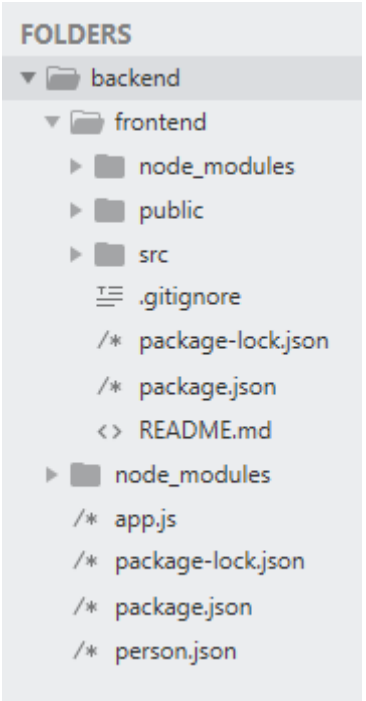
Copyright © 2021 HyperionDev. All rights reserved.

DIFFERENT APPROACHES

Historically, building a full stack web application always meant that your front-end application would be served up by your back-end application. With the advent of software as a service (SaaS), web services, Restful APIs and cloud computing, this is no longer always necessary. It is now possible to deploy your back-end application that exposes a restful API on one platform and your front-end application on another. Your front-end could then fetch data (using the Fetch API or similar) exposed by your custom API in the same way that it would any other API. However, sometimes it is still appropriate to keep your back-end and front-end together. Therefore, the decision that you face is basically whether to:

- Keep them together – Express and React applications sit on the same machine. The Express application then serves the React files and the API requests. Otherwise,
- Split them apart – Host the Express API on one machine, and the React app on another.

In this task, we will be keeping the React and Express apps together. Therefore, we recommend that you create the front-end React application within the back-end Express application project directory. Your back-end project directory would thus contain your front-end project directory:



Copyright © 2021 HyperionDev. All rights reserved.

SET UP PROXY

One of the most important things you need to do to get the front-end and back-end to work together is to make your Express app the proxy server for your React app. A proxy server is a server that works by intercepting connections between a sender and receiver. An HTTP proxy intercepts web access requests and handles them appropriately. From your React app, specify that the Express app is the proxy server (i.e. will intercept HTTP requests). When the user requests a resource that is not static through the React front-end, the request is forwarded to the proxy (Express app) which will then handle the request.

To specify that the Express app is the proxy, do the following:

1. Open the package.json file of your React app. Be careful here. Open the package.json file for your React app, not for your Express app.
2. Add the proxy information to the package.json file as shown below. Make sure that the port you specify is the same port number that you configured for the Express app. Also, make sure that the React and Express apps are working on different ports (you will have to modify at least one of them because both listen on Port 3000 by default).

```
"proxy": "http://localhost:3001"
```

USE THE FETCH API TO GET DATA FROM SERVER

You are already familiar with the Fetch API. It provides an interface for asynchronously fetching resources. The `fetch()` method takes one mandatory argument, the path to the resource you want to fetch. It returns a Promise that resolves to the Response to that request, whether it is successful or not.

Below is an example of using the Fetch API to get data from a custom API built using Express. You will notice that you use the Fetch API in nearly exactly the same manner whether you are getting data from a remote or local API (your Express app). The only difference if you are getting data from the Express server that you have configured as the proxy, is that you only provide the path to the resource, not the full URL (i.e. `fetch("/items")` instead of `fetch("https://api.example.com/items")`). The path that you pass as an argument to the `fetch()` method should have a corresponding route in the Express app. E.g. `fetch("/items")` in the React app would have a matching `app.get('/get', function(req, res) {...` route in the Express app.

Copyright © 2021 HyperionDev. All rights reserved.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      items: [] };}

  componentDidMount() {
    fetch("/items")
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            items: result.items
          });  },
        (error) => {
          this.setState({
            isLoaded: true,
            error
          }); }) }

  render() {
    const { error, isLoaded, items } = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
        <ul>
          {items.map(item => (
            <li key={item.name}>
              {item.name} {item.price}
            </li>
          ))}
        </ul>
      );}}}
```

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. As such, it can also be used to make HTTP PUT, POST or DELETE requests.

Make sure that you parse the response you get from the Express server appropriately. Remember that from the Express Server, **`res.send()`** can be used to

Copyright © 2021 HyperionDev. All rights reserved.

send various types of responses (in the body of the HTTP response) to the front-end including Buffer objects, Strings, objects, or Arrays. When the parameter is a String (e.g. `res.send('<p>some html</p>');`), the `res.send()` method sets the Content-Type to "text/html". When the parameter is an Array or Object, Express responds with the JSON representation. You can also use **`res.json()`** to send JSON data.

When you parse the data sent from the Express app in the `fetch()` method in your React app, you could use either `res.json()` (as shown in the example above) or **`res.text()`**.

USE FETCH API TO POST DATA TO SERVER

To use the `fetch()` method to make HTTP POST, PUT or DELETE requests, we pass the `fetch()` method an optional second argument, an init object that allows you to control a number of different settings. Two important settings that can be specified are:

- method: which can be GET, POST, PUT or DELETE
- headers: A headers object is a simple multi-map of names to values. Importantly we can modify the "Content-Type" to describe the type of data that is being passed with the request. The "Content-Type" can be "application/json" or "application/x-www-form-urlencoded".
- Body: Both requests and responses may contain body data which can be either text, JSON, form data or an array buffer. See more about the types of data that can be sent or received in the body **here**. Any data that should be passed from the front-end to the server or vice-a-versa is sent in the body. The body data type must match the "Content-Type" header.

Notice how the init object is passed as a second argument to the `fetch()` method below to make an HTTP post request that passes some data stored as JSON to the server:

```
postData(`/answer`, {answer: 42})
  .then(data => console.log(JSON.stringify(data))) // JSON-string from
`response.json()` call
  .catch(error => console.error(error));

function postData(url = ``, data = {name: "Sue"}) {
  // Default options are marked with *
    return fetch(url, {
        method: "POST",
```


Copyright © 2021 HyperionDev. All rights reserved.

```
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(data), // body data type must match
"Content-Type" header
    })
    .then(response => response.json()); // parses response to JSON
}
```

If we want our Express server to be able to access content that is passed in the body of the HTTP request, we need to include the **body-parser middleware**. The body-parser middleware extracts the entire body portion of an incoming request stream and exposes it on req.body. To use body-parser, do the following:

- Install body-parser: `npm install body-parser --save`
- In your Express code where you will be processing the Post request, include the following code:

  ```
  const bodyParser = require('body-parser');
  app.use(bodyParser.urlencoded({ extended: true }))
  app.use(bodyParser.json())
  ```
- You will now be able to get data passed through in the body of the HTTP POST or PUT request using req.body (e.g. `req.body.name`)

As you can see in the code example above, `JSON.stringify(...)` can be used to pass JSON in the body. You could also pass FormData in the body. See more about using FormData **here** and **here**.

PREPARING YOUR APP FOR DEPLOYMENT

Before you are ready to start deploying your app to Heroku, there are a few things that you have to change in your app first. Many of these may already have been done by default when you generated your app or you may already have changed them based on what you have learnt. Therefore, the following serves as a checklist of things to take into consideration before deploying your full stack application to Heroku:

- **Specify the version of Node.js and NPM you have used to build your app**

  If Heroku builds your app using a different version of Node.js than the one you have used to develop your app, it could lead to unexpected problems. It is, therefore, important that we specify the version of Node.js that we are using. To do this, follow these steps:

Copyright © 2021 HyperionDev. All rights reserved.

- Find out which version of Node.js you have been using to develop your app. Do this by typing `node -v` into the command line interface.

- Similarly, find out which version of NPM you have been using. `npm -v`

- Update your package.json file (for your Express app) to reflect this information. You do this by specifying an 'engines' field in your package.json file as shown below.

```
"engines": {
  "node": "10.14.1",
  "npm": "6.4.1"
},
```

- **Dynamically bind ports**

  Heroku specifies which port to listen for HTTP requests on. We, therefore, can't choose which port we want to use in the production environment. Consequently, we need to modify our code so that it gets a port number from Heroku and dynamically binds to the port in the production environment. We do this by modifying the code where we set up our server. We would usually have a line of code in this file that says something like, `app.listen(3001)`. We change this code as shown below:

```
const PORT = process.env.PORT || 3001;
app.listen(PORT);
```

  As shown in the code above, instead of hard-coding a port number, we create a variable that is assigned a number based on the process.env.PORT value. Process.env is used to access environment variables, i.e. variables that are set by the underlying runtime environment. The `process.env.PORT` value will only be assigned in the production environment (by Heroku) and not in the development environment (the PC you are using to develop your app). We, therefore, have an or (||) statement that sets the port value (to 3001 in this example) that we will use in the development environment.

- **Specify start scripts**

  Heroku will look in your package.json file (of your back-end app) to see how to start your server. Make sure that you specify the file that you want to use

Copyright © 2021 HyperionDev. All rights reserved.

to start your server in the scripts section of your package.json file. E.g. `"start": "app.js"`

- **Specify a heroku-postbuild script**

  If you want to add a React app to Heroku with your Express app, you must make sure that your React app is built and deployed on Heroku when the Express app is deployed. We ensure this by adding a heroku-postbuild script to our root Express package.json file, as shown in the previous image.

  An example of a full script is given below:

```
"heroku-postbuild": "NPM_CONFIG_PRODUCTION=false npm install --prefix frontend && npm run build --prefix frontend"
```

  In the script above, 'frontend' is the name of the React app you have created. Here Heroku is instructed to build the React app in the front-end directory. This results in resources like these:

```
build/static/js/main.e9c53ac3.js
build/static/css/main.c17080f1.css
```

  The environment variable NPM_CONFIG_PRODUCTION=false allows you to access packages declared under devDependencies in a different buildpack or at runtime. For more information about this, see **here**.

- **Change Express' App.js file to call React build assets**

  In production, Express needs to serve up resources that have been built from the React app. We allow this by adding the following code to App.js of the Express application.

```
if (process.env.NODE_ENV === 'production'){
  app.use(express.static(path.join(__dirname, 'frontend/build')));
      app.get('*',(req,res)=>   {res.sendFile(path.resolve(__dirname, 'frontend', 'build','index.html'));
});
}
```

  In the code above, 'frontend' is the name of the React app you have created. When this app is built, a directory called 'build' is created which contains the assets that the React app makes available. We, therefore, add the code:

```
app.use(express.static(path.join(__dirname,'frontend/build')));
```


Copyright © 2021 HyperionDev. All rights reserved.

For more information about how this statement is used to make the React app resources available, see here.

- **Create a .gitignore file**

  You would have noticed that every Express application that you have created so far has contained a node_modules directory. This directory is full of dependencies for your project. These dependencies are also specified in your package.json file.

  Heroku will automatically create a new node_modules directory for your app when your app is built on Heroku. To avoid conflicts, we don't want to include the node_modules directory in the source code that we deploy to Heroku. To ensure this, we create a .gitignore file. This file specifies which files and directories should not be committed to Git.

  Create a file called **.gitignore** in the root directory of your application. In this file simply type node_modules and save the file.

Copyright © 2021 HyperionDev. All rights reserved.

Compulsory Task 1

Follow these steps:

- Create an attractive React front-end that can be used to interact with the cars API you built with Express in the previous task.
    - You should be able to use your React front-end to get a list of cars, add additional cars to the list, modify the details about a specific car and delete a car from the list.
    - For a bonus challenge, make one of the car properties a URL to an image and display this image wherever you list a car item on your frontend.

Optional Bonus Task

Deploy this app to Heroku.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

Copyright © 2021 HyperionDev. All rights reserved.