TASK

Express - Overview

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

WELCOME TO THE EXPRESS - OVERVIEW TASK!

Now that you know the basics of how to use Node.js, you can start building server-side applications more quickly using Express. Express is a lightweight web framework. By the end of this task, you will be able to explain what Express is and what the benefits of using it are. You will also be able to use Express to create a back-end web application that handles routing and can respond to HTTP requests with both static and dynamic content.

WHAT IS EXPRESS.JS?

Express is the most popular Node web framework. It basically gives you access to a library of code (Node.js functions) that makes it easier for you to create web servers with Node. It is also the underlying library for several other popular Node web frameworks. Express is a minimal and flexible web framework that provides a robust set of features for web and mobile applications. It is open source and maintained by the Node.js foundation.

Although Express is minimalist, developers have created compatible middleware packages to address almost any web development problem. You can find a list of middleware packages maintained by the Express team at **Express Middleware**.

According to Express, “Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle.”

Unlike other frameworks, like Django, Express is an un-opinionated web framework. Unopinionated frameworks have fewer restrictions regarding the best way to glue components together to achieve a goal, or even what components should be used, than opinionated frameworks. These frameworks allow developers to use the best tools to complete a particular task, but you need to find these tools yourself. It is very flexible and pluggable and has no “best way” of doing something. You can use almost any compatible middleware, in almost any order and you can structure your app however you like.

WHY USE EXPRESS?

You use Express for the same reason that you would use any other web framework;

Copyright © 2021 HyperionDev. All rights reserved.

to speed up development and decrease the amount of boilerplate code you have to write. We have been using Node to create a web server. There are many aspects of creating a web server for your web application that will be very similar, regardless of the app you are building. We are going to use Express and later, the Express-generator to generate a lot of the boilerplate code and project structure for us to streamline the development process. We could write all the code we need without Express, but we will use Express to speed up development.

INSTALL EXPRESS

In the previous tasks, you have installed Node.js and learned about NPM. You will now install Express using NPM.

NPM is an extremely important tool for working with Node applications. It is used to fetch any packages (JavaScript libraries) that an application needs for development, testing, and/or production. It can also be used to run tests and Node modules used in the development process. Express is another package that you need to install using NPM.

1. Open your terminal or command prompt.

2. Create a directory using the mkdir command and then navigate into it using the cd command.
   ```
   mkdir myapp
   cd myapp
   ```

3. Use the **`init npm`** command to create a package.json file for your application. We manage dependencies using a plain-text definition file named **package.json**. All the dependencies for a specific JavaScript package are listed in this file. The package.json file should contain everything NPM needs to fetch and run your application.
   The command, **`npm init`**, will prompt you to enter several details, such as the package name, version, description, entry point, test command, etc.
   ```
   npm init
   ```

4. To display the package.json file enter **`cat package.json`** or **`type package.json`** (depending on the operating system you are using) into your terminal. You could also view the package.json file in Sublime or Visual Studio Code.

Copyright © 2021 HyperionDev. All rights reserved.

5. We will now install the Express library in the 'myapp' directory that you created and save it in the dependencies list of the package.json file by entering the following command into the terminal or command prompt:
   `npm install express`

6. Enter `cat package.json` or `type package.json` into your terminal again. The dependencies section of your package.json will now appear in the **package.json** file and will contain Express.

```
cat package.json
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.15.4"
  }
}
```

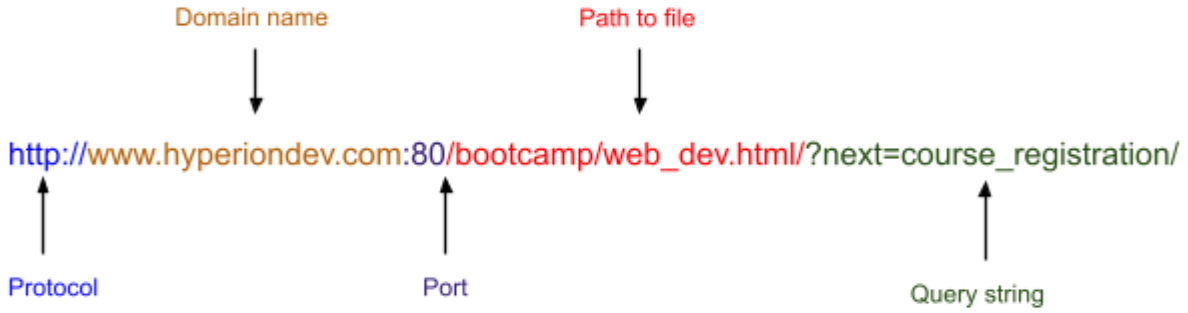
Now that we have Express installed, we can start creating the code for the backend of our web application. One of the most important aspects of server-side logic is routing.

ROUTING

One of the most important tasks of a server is to perform routing. Routing refers to determining how an application responds to a client's request to a particular endpoint. The request is made using a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Remember that, a URL contains a lot of information:

Copyright © 2021 HyperionDev. All rights reserved.



1. It identifies the protocol being used to send information. In the example above, the protocol being used is HTTP.
2. It identifies the domain name of the web server on which the resource can be found, e.g. www.hyperiondev.com.
3. It identifies the port on the server. In this example, the port number is given as port 80. In reality, if the default HTTP ports are used (port 80 is the default for HTTP, port 443 for HTTPS) they don't have to be given in the URL.
4. It gives the path to the resource on the web server, e.g. /bootcamp/web_dev.html
5. Data can be passed using parameters (as shown in the image below) or using a query string (as shown in the image above).

To perform routing, we are interested in the *path* section of the URL (you will see how to use the query string section of the URL in the next task). Routing involves identifying the path that was requested and providing a response based on that path.

With Express there are a few route methods used to perform routing: get, post, put and delete. In this task, we will only use the get method. In the next task, you will use the other routing methods. The get method responds to HTTP get requests. An HTTP get request is used to request a specific resource.

See a code example that uses the get method for routing:

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

The `app.get()` method takes two arguments:

- **The path**. In this example, the path is '/' the root route of our app. In development, the URL **http://localhost:3000/** (where the server is running on port 3000) will match the route specified in the `app.get('/', ...)` method in our code example. If you wanted to add a route handler that

Copyright © 2021 HyperionDev. All rights reserved.

would handle the HTTP request using the URL, **http://localhost:3000/about**, you would have to add an `app.get('/about', ...)` function.

- **A callback function**. The callback function that is passed as the second argument to the `app.get()` method acts as a *route handler*. In other words, in this example, when a get request is made to the homepage of the web app, a response object that simply contains the text 'Hello World!' will be sent from my server to the browser.

Now that we understand what routing is, let's create a server with Express!

CREATE A SERVER USING EXPRESS

In the root directory of your 'myapp' directory (the directory you created when you installed Express), create a file called 'app.js' and copy the code below into it:

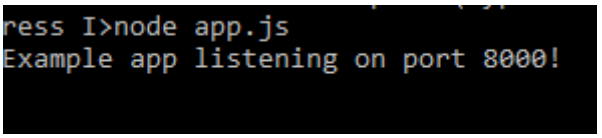
```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(8000, function () {
  console.log('Example app listening on port 8000!')
})
```

The code above is a simple "Hello World" Express web application. Let's analyse this code line by line:

- **Line 1:** `require()` is called to import the "express" module.
- **Line 2:** Create an object called `app` by calling the top level `express()` function. This object represents our Express application. The app object contains important methods that we use to create our server. Notice two of the methods in the code above: `get()` and `listen()`. You will also learn about `app.use()` in this task.
- **Line 4:** Create a route handler that will respond to requests using the `app.get()` routing method.
- **Line 8:** The app object also includes the `listen()` method. The `listen()` method specifies what port our app object (application server) will listen to HTTP requests on. The `app.listen()` method returns an `http.Server` object.

To start the server, call node with the script in your terminal or command prompt (remember to make sure that you navigate to the myapp folder first).

Copyright © 2021 HyperionDev. All rights reserved.



Now use your web browser to navigate to http://127.0.0.1:8000/. You should see the string “Hello World!” displayed in your browser!

SERVING STATIC FILES WITH EXPRESS

Even dynamic web applications may have certain static components to display. With Express it is easy to serve static resources by using the `express.static` built-in middleware function. Remember that a middleware function is a function that has access to the request and response objects.

To allow your app to serve static files, simply do the following:

1. Create a folder in your project directory to store the static files you want to serve. This folder is usually called ‘public’.
2. Add all the static resources that you want your app to make available (images, HTML files etc) to this folder. Make sure that these files have meaningful names because, by default, you will use the file name of the resource to access it. Don’t store any files that you don’t want users of your app to see in this directory!!
3. Add the following code to your app.js file: `app.use(express.static('public'));`
   We use `app.use()` to include any built-in middleware functions we need in our app. For a list of other built-in middleware functions for Express, see **here**.

Once you start your server (type `node app.js` in the terminal), you should be able to access these static resources from the browser. For example, if you added an HTML file called ‘example.html’ to the public folder, you could access it with this URL: **http://localhost:3000/example.html** (where you have created a server that listens for HTTP requests on port 3000).

ENVIRONMENT VARIABLES

When creating back-end apps, it is important to be able to access *environment variable*s. So far, the only variables we have used have been variables we have

Copyright © 2021 HyperionDev. All rights reserved.

created using the keywords var, const or let. However, backend code needs to be able to access variables created and set in the environment in which the code is run (i.e. variables set by the server). Node.js allows us to access these variables using **`process.env`**. To see some of the environment variables stored on your PC, try this: add the following line of code to your app.js file and run it in the terminal.

```
console.log('The value of process.env is:', process.env);
```

When your code is being executed on a web server (after deployment), an important environment variable that will be set on the server is the port number on which your application server will listen for HTTP requests. To get the port number from the environment variables instead of hardcoding it, we use the following code:

```
const PORT = process.env.PORT || 8000;
app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});
```

You should also use **`process.env`** for other values you don't want to hardcode or want in your codebase (like API keys).

Take note:

Notice how we use a string literal and an arrow function to write code more efficiently in the code example above. Remember, these are introduced as improvements to JavaScript with ES6.

Also notice that the variable, PORT is named with all capital letters. This is a common naming convention recommended by style guides for constant variables that can't change once a value has been assigned.

See more naming conventions recommended by Google for JavaScript **here**.

NODEMON

As you have probably experienced by now, it can be time-consuming to restart your server every time you make changes to your code! You can use Nodemon to enable server restarts on file changes. After typing the command shown below

Copyright © 2021 HyperionDev. All rights reserved.

into the command line interface, you should see that your package.json file now includes a reference to Nodemon in its `devDependencies` section.

```
npm install --save-dev nodemon
```

To run an app using Nodemon, type `nodemon name_of_file` in the terminal instead of `node name_of_file`.

ADD A START SCRIPT

It is recommended that you add a script to your **package.json** file for every application that you create that specifies how to start your app. This is done as shown in the image below:

To start the application, we then type `npm start`. This will use the instruction specified in the start script in the package.json file to run the code.

Copyright © 2021 HyperionDev. All rights reserved.

Compulsory Task 1

Follow these steps:

- Create an application project folder called ‘my_first_express_app’.
- Create another subdirectory called ‘public’ that contains two static html files called **terms_of_service.html** and **privacy_policy.html**. Feel free to reuse any html files you have created before.
- Create a file called **organisations.json** that describes at least 5 organisations. E.g. format: `{“name”: “New World Consulting”, “email”: “hello@newworld.org”, “pty_ltd”: true}`
- Create a server that will do the following:
    - Display “Organisation listing” and a list of organisations, and their properties in a table, loaded from the file **organisations.json** at URL http://localhost:3000/.
    - Display the static HTML page, **terms_of_service.html** at URL http://localhost:3000/terms_of_service.html
    - Display the static HTML page, **privacy_policy.html** at URL http://localhost:3000/privacy_policy.html
    - Displays the message “Sorry! Can’t find that resource. Please check your URL” if the user enters an unknown path. Help **here**.
- Enable the server to restart on file changes.
- You should be able to start the server using `npm start`.

Things to look out for:

Make sure that you delete ‘node_modules’ before submitting the code.

Copyright © 2021 HyperionDev. All rights reserved.

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.