**Using Express For Static HTML –**

[Ryan Schleck](https://ryan-m-schleck.medium.com/?source=post_page-----bc4235351473--------------------------------) Jun 26, 2022

**Introduction**

Previously I have written [a blog post](https://ryan-m-schleck.medium.com/an-introduction-to-nodejs-aeddbc7c6a91) discussing the JavaScript run time environment [NodeJS](https://nodejs.org/en/about/). Afterwards I wrote about the web framework built on top of it called [Express](https://expressjs.com/). Within [that blog post](https://ryan-m-schleck.medium.com/the-basics-of-the-express-framework-830e2c667ba4) I showed some examples of how you can use Express to serve a line or two of html. This post will expand on these and show how Express can be used to serve whole html files along with some very basic CSS. There are of course [templating engines](https://www.developerdrive.com/best-javascript-templating-engines/) that can be used in Express to create HTML a lot faster than writing the files by hand, but I will discuss that in a future blog post. This blog post assumes that you are familiar with Express or have at least read my two above linked blog posts as it continues on some of the information provided by them.

**Serving HTML Using Express**

Lets discuss how to serve HTML using Express by using an example. In this we will have a Views folder that contains our HTML files. We will use these files to create a home page along with a default 404 page. Lets begin!

First we will begin our project by using terminal to navigate to where you’d like your project to be and creating a folder for it. Do so by entering:

mkdir *project-name*

Then you will need to initialize your **package.json** file using **npm**. After navigating into your project and firing up your text editor. Open a terminal at the root of your new project and enter:

npm init

Once this has completed and you can now see your package.json file lets add express by entering:

npm install --save express

After this has completed we can set up our projects file structure. Please copy the following setup:

Graphical user interface

Description automatically generated with medium confidence

The following folders/files should be created automatically: **node\_modules**, **package-lock.json** and **package.json.** notice that **app.js** is at the topmost file of your project. You may therefore change the “main” property inside package.json to “app.js”,

Lets go to the top of our app which will coincidentally be **app.js**. We have covered the basics of what this file should contain previously within my previously mentioned [Express blog post](https://ryan-m-schleck.medium.com/the-basics-of-the-express-framework-830e2c667ba4). I am going to continue and assume you are familiar most of the contents of this file I’m about to show you. The **app.js** file should resemble the following (minus the comments):

Text

Description automatically generated

We first require the [path module](https://nodejs.org/en/knowledge/file-system/how-to-use-the-path-module/) in order to assist with file path manipulation. We then require express in order to create a valid request handler we will use for our middleware functions. We then require our middleware within home.js. We then allow access to our public folder from the browser by using [express.static](https://expressjs.com/en/starter/static-files.html" \t "_blank) and passing it our path module with the [join method](https://nodejs.org/api/path.html#pathjoinpaths). We then pass it [\_\_dirname](https://nodejs.org/api/modules.html#__dirname) along with the folder we want to share.

Now that we have the top of our app squared away lets look at our middleware within the routes folder **home.js**. Please edit this to resemble the following disregarding the comments:

Text

Description automatically generated

You will notice on line 7 that we are importing the path to the root of our project from **path.js** and using it on line 15 as the start of the file path to the html file we’d like to serve. We do this using **[sendFile](https://expressjs.com/en/4x/api.html" \l "res.sendFile" \t "_blank).**

On line 7 we are importing the contents of **path.js** lets take a look at this file now to see how it provides us the absolute path to our project. Our **path.js**file is just 2 lines of code resembling the following:

Text

Description automatically generated

on line 4 we export our absolute path by accessing it via the filename property on the [require.main](https://nodejs.org/api/modules.html" \l "requiremain" \t "_blank) object.

Now that we have our middleware both written and imported lets go inside **home.html** and create an html skeleton by typing *html* and selecting *html:5* and hitting enter. Add a link tag and a single h1 tag. The file will then resemble the following:

Text

Description automatically generated

please ignore the comments

We can now duplicate this process to create our default error page inside the **404.html**file. Also feel free to populate **main.css** inside the **public/css** folder with whatever styling you’d like! Once this is complete feel free to spin up your server by entering the following within terminal inside your projects root directory:

node app.js

At this point if you open a browser and navigate to **localhost:3000/home** you should see your page!

**Conclusion**

At this point you should have a working demo capable of serving up a message of “Hello Express World!” for the url of /home along with some css. As well as a default 404 error page for all of the urls that are not found. Now that you have the basic concept of serving whole html files you can expand much further than this simple example. The Express framework is a capable of a lot things outside of this minimal task but hopefully this provides a good foundation to build off of.