P

LEARNING PATHS

Setting up your project to use JavaScript

TABLE OF CONTENTS

The modern JavaScript ecosystem offers many options for setting up a JavaScript project. In this article, we'll examine a few of these options and outline a simple but effective setup that you can use to complete all the challenges in this learning path.

Linking a JavaScript file

You link a JavaScript file to your project in a very similar way to linking a stylesheet.

```
<head>
  <title>My First JavaScript Project</title>
  <link href="./styles.css" rel="stylesheet" />
  <script src="./main.js"></script> <!-- Load your JavaScript file here
</head>
<body>
  ...
</body>
```

As you can see, this is pretty straightforward, but you should be aware of a couple of things.

In the code snippet above, the script tag will load and execute the contents of `main.js` **before** the contents in the body have been rendered. Since there's every chance that you'll want to interact with the Document Object Model (DOM) in some way with your JavaScript file, you'll need a way to defer execution of the code until **after** the rest of the document has loaded.

Option 1: `DOMContentLoaded` event

Once the browser has finished loading the DOM, it fires a `DOMContentLoaded` event, which you can use to trigger the execution of your code. You would do that by adding an event listener to the code you write in your JavaScript file, something like:

```
// Approach 1: wrap your entire file in an event listener
document.addEventListener('DOMContentLoaded', function() {
  // all your code goes here
});


// Approach 2: create an initializer function that runs your code
function initialize() {
  // your event handlers and setup logic goes here
}

// call your initializer function when the DOM is loaded
document.addEventListener('DOMContentLoaded', initialize);
```

In this case, the browser will load the entire script before loading the rest of the DOM but will not execute its contents until the `DOMContentLoaded` event has fired.

Option 2: Place the script tag at the end of the HTML document

Alternatively, you can instead place the script at the end of your HTML document inside the body tag, like so:

```
<head>
  <title>My First JavaScript Project</title>
  <link href="./styles.css" rel="stylesheet" />
</head>
<body>
   ...
  <script src="./main.js"></script> <!-- place script at the end of the
</body>
```

This works because the browser processes each line of the HTML document at a time. Once it reaches the end of the document, the last node it finds will be the script tag, at which point it will download and execute the script. This effectively means that your which point it will download and execute the script. This effectively means that your script will run as expected without needing to use the `DOMContentLoaded` event.

Option 3: `defer` script attribute

You can achieve the same effect as option 2 by simply setting a defer attribute on the script like this:

```
<head>
  <title>My First JavaScript Project</title>
  <link href="./styles.css" rel="stylesheet" />
  <script defer src="./main.js"></script> <!-- add `defer` attribute -->
</head>
<body>
  ...
</body>
```

This means that the script will actually run *just before* the `DOMContentLoaded` event fires, but after the rest of the DOM has loaded. The other benefit is that the script will load *asynchronously* (i.e., in the background) while the browser performs other tasks. Because of this, unlike in option 1, the script will not block the rest of the page from being loaded while the script is downloaded.

Which option should I use?

You can use whichever option you like, but we recommend using option 3. It's probably the most performant and means you don't have to worry about setting up a `DOMContentLoaded` event in your JavaScript file. However, you will need to remember to add `defer` to your script tags.

When setting up your project, the best thing to do is to grab the body element and `console.log` it. If you don't see the body element in the console, you should check that you've linked the file correctly. You can get stuck into your project once you get the expected console log.

Taking it up a level: hot reloading

Once you've set up your project, you'll likely spend quite a bit of time making minor changes to your code and then refreshing the browser to see the changes. This can become tedious and time-consuming and wear out your F5 key (or cmd and R keys if you're using a Mac)!

We can help ourselves by running a small web server locally that watches for changes in our files and updates the browser accordingly. It's an excellent way to work on a web project, but it requires a little setup.

Option 1: Using VS Code

VS Code has a Live Server extension that takes care of this for you. It's straightforward to set up.

1. Click on the Extensions tab on the left-hand panel.
2. Search for Live Server.
3. Click Install to install the extension.
4. Open your project in VS Code and click the Go Live button at the bottom right of the screen.
5. VS Code will automatically open your project in your default browser. Any changes you make in any of your project's files will automatically update in the browser.

You can check out its GitHub repository for more information on the Live Server VS Code extension.

Option 2: Using npm (advanced)

If you prefer not to use VS Code, you can still replicate the effect of a Live Server by using a Command-Line Tool (CLI) that you can install with npm (The Node.js Package Manager).

This is a little more involved because you must first install Node and npm on your laptop. Node is a process that runs JavaScript on the command line (rather than in the browser) and allows you to write and execute JavaScript commands directly in your Terminal.

How you install Node depends on which operating system you use. Here are our recommended guides for each major operating system:

- MacOS: install Node.js using Homebrew.
- Windows: install Node.js using the Windows installer

- Windows: install Node.js using the Windows installer.
- Ubuntu: install Node.js using apt.

Once you have installed Node, you can install the Simple Hot Reload Server package like so:

1. `npm install -g simple-hot-reload-server`
2. Still in the Terminal, navigate to your project folder.
3. Type `hrs`. You should see something similar to the following output:

```
Server Address: <http://localhost:8082>
Map Data   →  <http://localhost:8082/__hrs__/map>
File View   →  <http://localhost:8082/__hrs__/file>
```

4. Enter the server address in your browser.
5. You should now see your project. Any changes to your project's files should now update the browser.

Project structure and internal links

There are a few things to note when using Live Server in VS Code or the Simple Hot Reload Server npm package.

Firstly, your `index.html` file should be at the root of your project folder. Otherwise, the web server will not be able to find your HTML document, and you won't be able to see it in the browser.

Secondly, now that your files are being served by a web server, you can use absolute paths when linking to resources like images and scripts. For example, if your images are located in an image folder, you can link to them like this:

```
<img src="/images/cats_on_hoovers.jpg" alt="Cute cats sitting on hoovers
```

Just like in your terminal, a leading slash indicates the root folder. In this case, the image is in the `images` folder in the root of your project.

The other approach is to use *relative paths*, whereby you indicate the location of the resource *relative to the current page*. This is normally done by either not using a leading slash or starting your links with a period to indicate the current folder:

```
<img src="./images/cats_on_hoovers.jpg" alt="Cute cats sitting on hoover
```

In this example, if this were on the home page `http://localhost:8082/`, the browser would request an image in an `images` folder at the root of your project, but if you were on say `http://localhost:8082/cats`, it would request an image located in `cats/images`.

Using relative paths for multipage sites can be a little confusing, so we would recommend switching to using absolute paths from here on out.

Beware GitHub pages

Unfortunately if you host your site with GitHub pages it will be hosted on its own sub-folder on your GitHub site, which means that absolute paths will behave unexpectedly.

If I had a repo called `project-one` and hosted it on GitHub pages, the url would be `https://mickyginger.github.io/project-one`, and an absolute path to `/images` would point to `https://mickyginger.github.io/images`, rather than `https://mickyginger.github.io/project-one/images`.

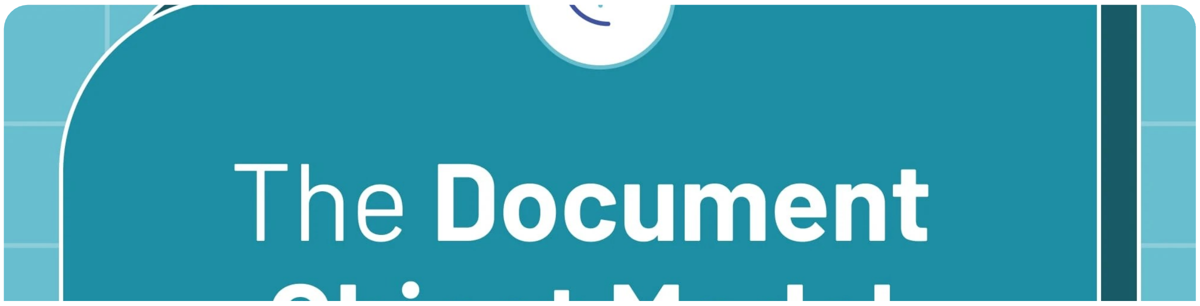
In the case where you want to host your project on GitHub pages, you should stick with relative paths when linking your images and scripts.

Conclusion

In this article, we've looked at different approaches to linking your scripts and running a web project locally as a web server. This setup should stand you in good stead for this learning path and help you focus on the job of coding.

You've also had your first taste of a more advanced project setup, which should make setting up projects for modern JavaScript frameworks less daunting.

Next step...



The Document Object Model

As the interface between HTML and JavaScript, the Document Object Model is a fundamental concept to working with JavaScript. This article gives you an overview of the DOM and how to interact with it.

START ARTICLE

Your path progress

- ✓ Welcome to the JS fundamentals path!
- ● **Setting up your project to use JavaScript**
- 🔒 The Document Object Model
- 🔒 Handling user events
- 🔒 Article preview component
- 🔒 Managing forms with JavaScript
- 🔒 Newsletter sign-up form with success message
- 🔒 Working with data
- 🔒 Time tracking dashboard
- 🔒 Refactoring your code
- 🔒 Tip calculator app

Password generator app

Frontend Quiz app

Stay up to date

with new challenges, featured solutions, selected articles, and our latest news

email@example.com

SUBSCRIBE

FRONTEND MENTOR

Unlock Pro
Contact us
FAQs
Become a partner

EXPLORE

Learning paths
Challenges
Solutions
Articles

COMMUNITY

Discord
Guidelines

FOR COMPANIES

Hire developers
Train developers

Terms Cookie Policy Privacy Policy License