Exploration — Linking HTML Documents and Serving Static Content

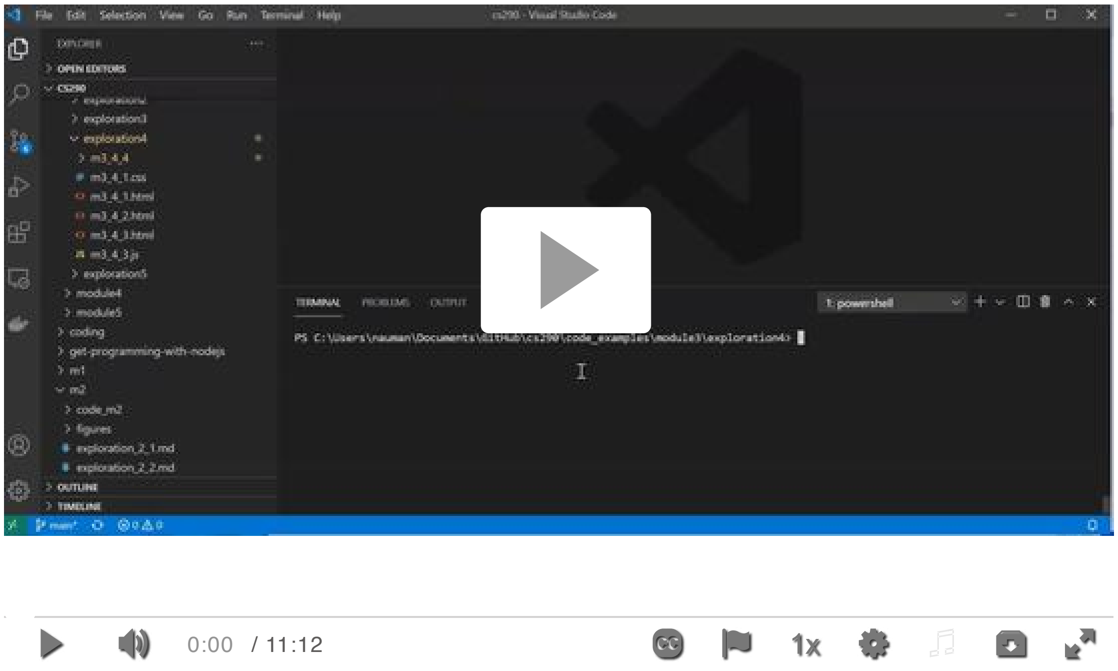
Introduction

Recall that the term routing means how an application is set to map HTTP requests to endpoints, where the term endpoint refers to the combination of a URL with an HTTP method. In a previous example, we had used Express's `app` API to define a route for the endpoint with URL `/` and HTTP method `GET` as follows:

```
app.get("/", (req, res) => { res.send("Hello World!");});
```

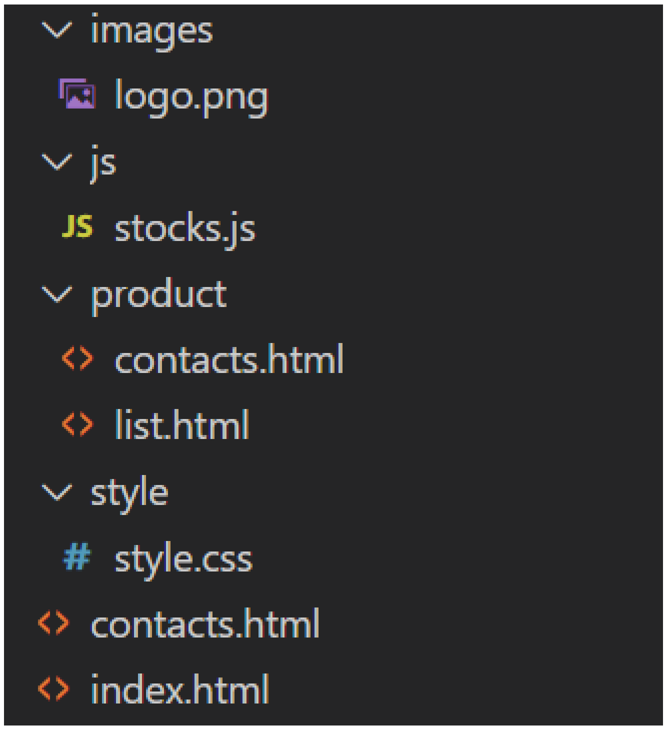
However, almost all web applications include a number of **static files** that rarely change. These includes HTML files, image files, CSS stylesheets, and files with JavaScript code. When a request is received for a static file, the requested file needs to be read from the filesystem and sent back in the HTTP response. Writing a separate route for every static file will be a waste of effort. Web application frameworks, such as Express, support a simple way to define routes for static files. In this exploration we will first look at how URLs in HTTP requests are interpreted to map to static files. We will also study how links within HTML documents are similarly interpreted. We will then write a simple web application using the API provided by Express that can serve static files.

Note: The files shown in the `public` directory in the following video are available for download as **public.zip (https://canvas.oregonstate.edu/courses/1879154/files/93831949?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93831949/download?download_frd=1) .


```
PS C:\Users\nauman\Documents\GitHub\cs290\code_examples\module3\exploration4>
```

URLs and Paths

Consider that we have a web application deployed at the following URL `https://module3.cs290.com/` and that the root directory for our application has the following structure.


```
images
  logo.png
js
  stocks.js
product
  contacts.html
  list.html
style
  style.css
contacts.html
index.html
```

The root directory has two files and four directories.

The files are `index.html` and `contacts.html`. The four directories are `images`, `js`, `product`, and `style`. The directories `images`, `js` and `style` each have one file in them, and the directory `product` has two files in it.

The Default Homepage

In almost all real-world web application, the file `index.html` in the root directory is called **the homepage** of the web application, and the web application is configured to return this file whenever a `GET` request is received for the URL `/`. This is in contrast with our "non real-world" example program where we had defined a route for the URL `/` that calls a function to return the string `Hello World` . Later in this exploration, we will see how we can configure a web app written using Express so that the app returns the homepage for `GET` requests for the URL `/`.

Absolute vs. Relative URLs

An **absolute URL** is a complete URL to a resource including the protocol and the domain name. For example:

- The URL `https://module3.cs290.com/index.html` and `https://module3.cs290.com/contacts.html` are both absolute URLs.

A **relative URL** points to a location relative to the file in which we use that URL.

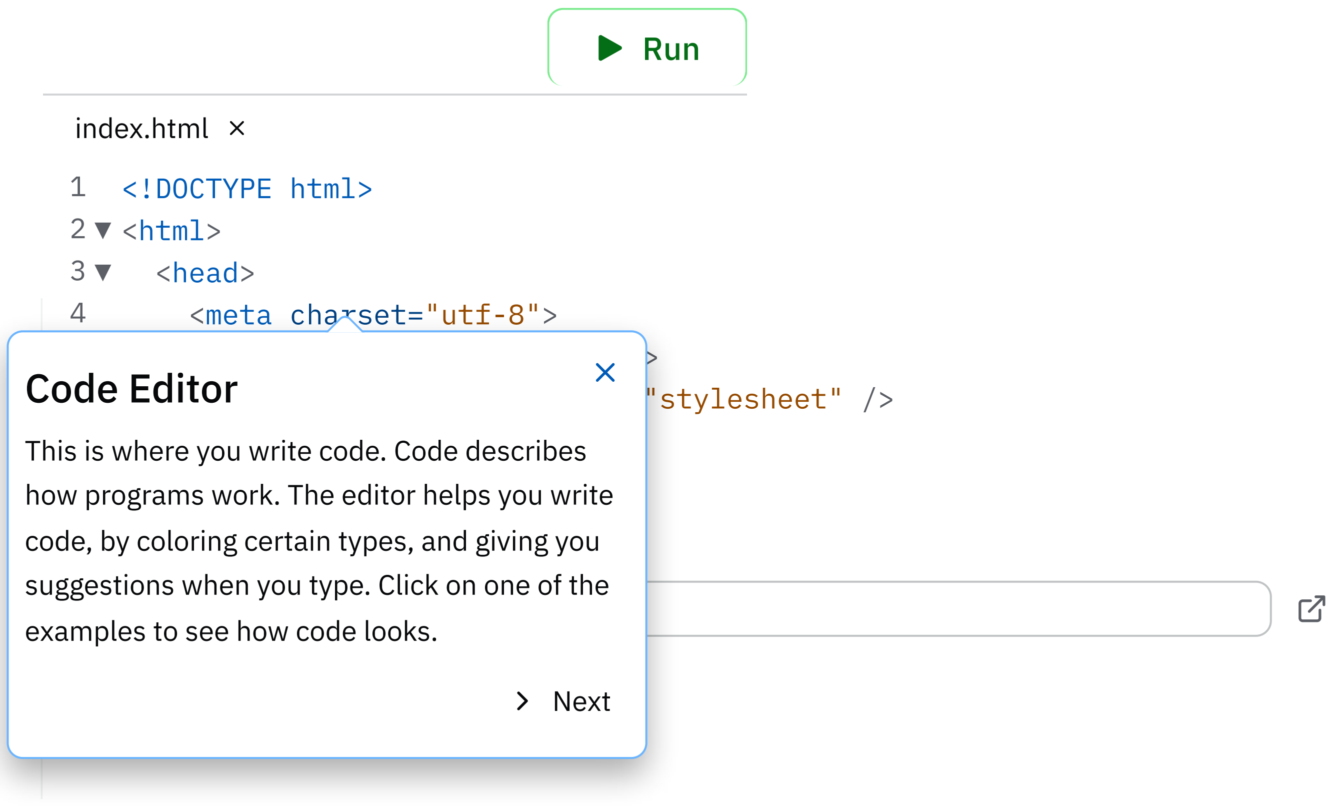
1. Same directory: We can specify a relative URL to a file in the same directory by using just the name of that file or by adding `./` before the name of the file. Using only the name of the file is the preferred syntax. For example:
    - In the file `index.html` , the URLs `contacts.html` and `./contacts.html` will both refer to the file `contacts.html` located in the root directory.
    - In the file `list.html` in the directory `product` , the URLs `contacts.html` and `./contacts.html` will both refer to the file `contacts.html` in the directory `product` .
2. Moving down the child directories: We can specify relative URLs for files down the directory structure using the names of the directories and files separated by a `/`. For example:
    - In the file `index.html`, the URL `images/logo.png` will refer to the file `logo.png` in the directory `images`, and the URL `style/style.css` will refer to the file `style.css` in the directory `style` .
3. Moving up the parent directories: We can specify relative URLs for files up the directory structure using `..` . Each `..` moves up one directory. Once we reach the needed ancestor directory, we can move down to the relevant file as done for moving down the child directories.. For example:
    - In the file `list.html` in the directory `product`, the URL of the file `logo.png` in the directory `images` is specified as `../images/logo.png`.

<link> element

We have seen two uses of URLs within HTML documents, linking to another document using the anchor element `<a>`, and specifying the source of an image in the `src` attribute of the image element `<img>`. Another use of URLs in HTML documents is using **the link element (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link)** `<link>` to specify a relationship between a document and an external resource. The `href` attribute is used to specify the URL of the external resource. The `rel` attribute, which stands for “relationship”, is used to specify the relationship between the external resource and the HTML document. Two commonly used values of `rel` are `stylesheet` to link stylesheets and `icon` to **link a website’s icon. (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link#providing_icons_for_different_usage_contexts)**

Example

In the following example, we use `<link href="style.css" rel="stylesheet"/>` to our HTML document. The `href` attribute has the value `style.css`, which means that this file `style.css` is in the same directory as the HTML document. Sometimes we will find stylesheets links that also include another attribute `type="text/css"`. However, specifying the `type` attribute for stylesheets is *optional.* Newer standards omit the type.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
                  >
                  "stylesheet" />
```

Code Editor

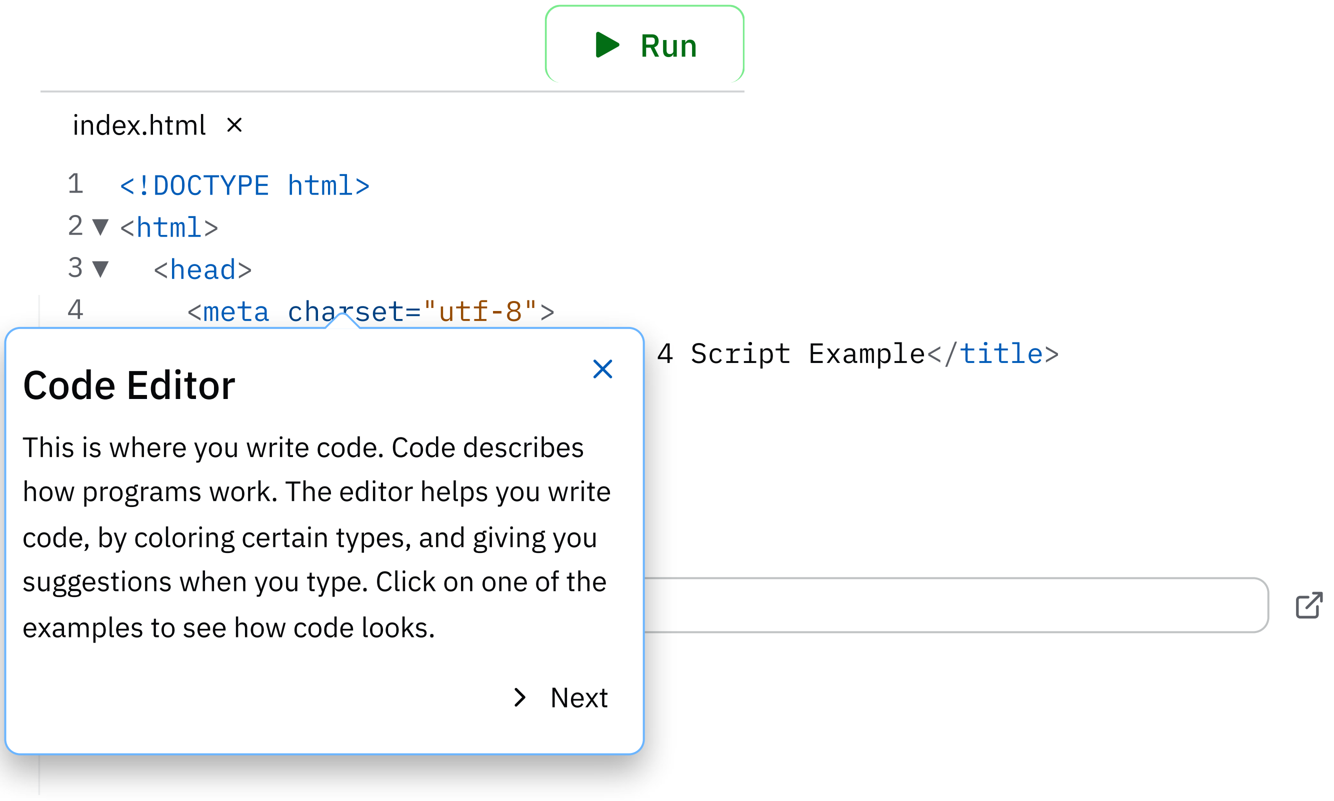
This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

<script> element

We add JavaScript code to an HTML document using the `script` element. We can either simply embed JavaScript code inside an opening and a closing `<script>` tag or use the `script` tag to refer to a JavaScript file using the `src` attribute to specify the URL for that file. Scripts are typically added below the content, so the content loads first. However, some scripts must be loaded first and as such will reside in the `<head>` area. In the following example, the script creates the only content on the page (a popup alert) and resides in the `<head>`. You saw the alert popup when you launched this Canvas page.

Examples

Notice the JavaScript code providing an alert in the following HTML file:

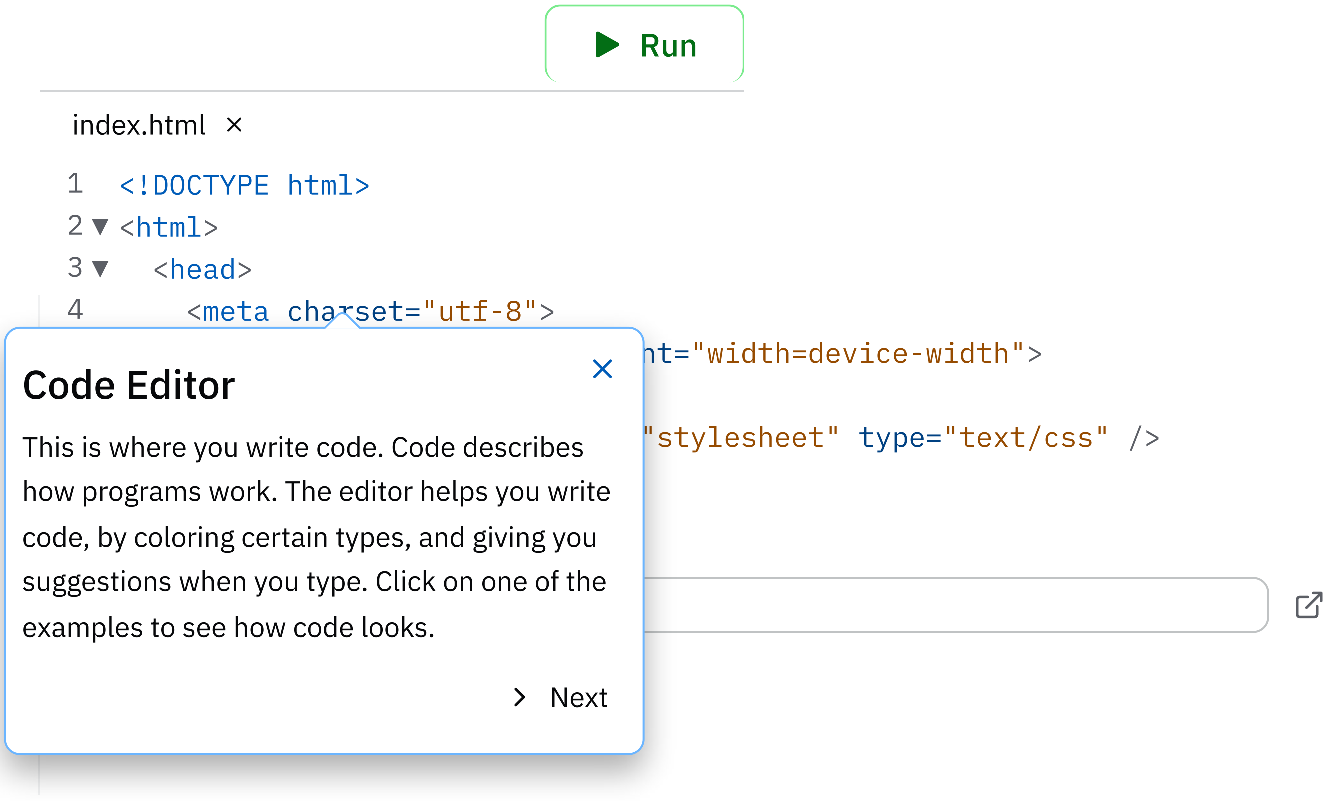


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
                             4 Script Example</title>
```

Code Editor

This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

Here is an example where we import an external JavaScript file using the `script` tag inside the body:



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
nt="width=device-width">
"stylesheet" type="text/css" />
```

This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

Serving Static Files with from a Web Application

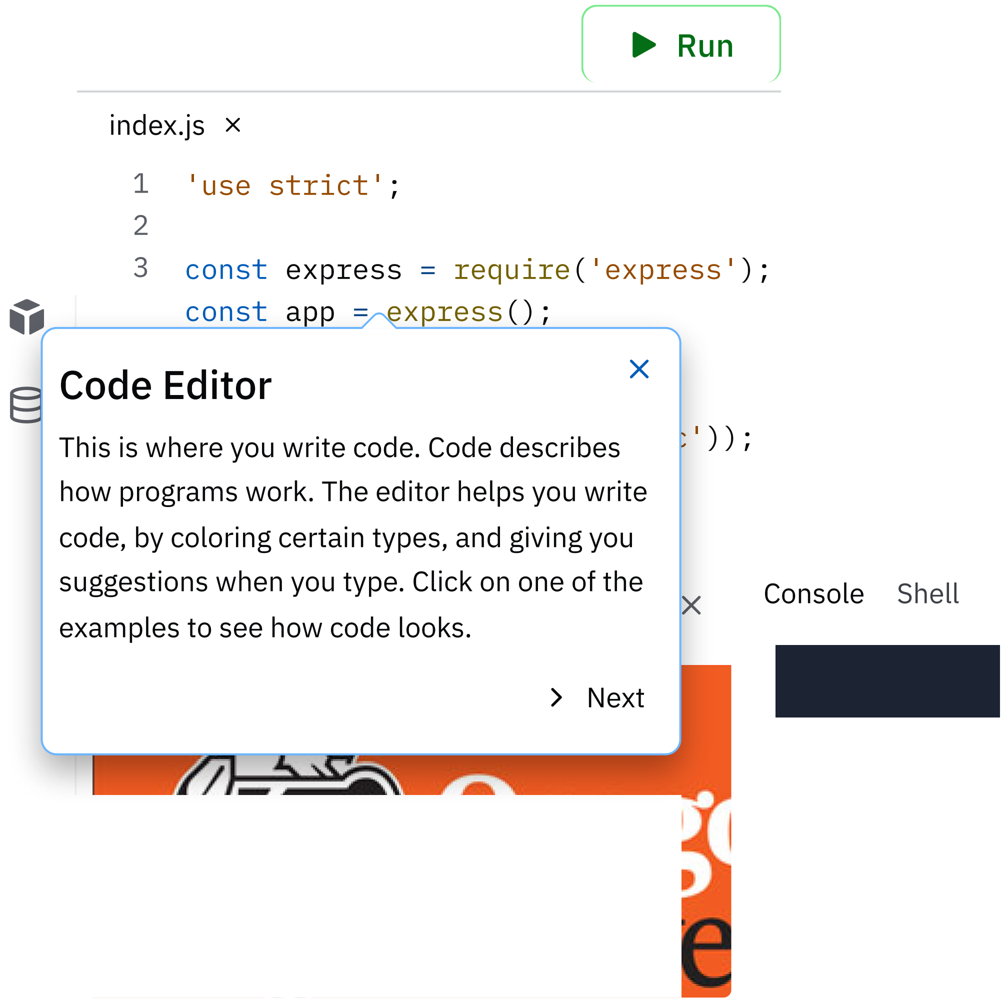
The `express` module (which we can access via Node and npm) provides a method `static` that we can use to serve static files. **This method (https://expressjs.com/en/4x/api.html#express.static)** has one required parameter named `root`. The value of `root` specifies the directory from which static files will be served.

Example

It is common to create a directory `public` in the root directory and place the static files underneath this directory, as presented in an earlier example in this Exploration. We can then enable serving static files from the `public` directory by adding the following line of code in our application:

```
app.use(express.static('public'));
```

A program that serves static files from the `public` directory is shown in the following example:



```
'use strict';

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

```
'));
```

Code Editor

This is where you write code. Code describes how programs work. The editor helps you write code, by coloring certain types, and giving you suggestions when you type. Click on one of the examples to see how code looks.

Note that the URL in the request should not include `public`. For example, if we deployed this application at the URL `https://module3.cs290.com`, then

- The URL `https://module3.cs290.com/` will map to the file `index.html` in the directory `public`, i.e., to the homepage.
  - Just by enabling our app to serve static files from the `public` directory and placing the file `index.html` in that `public` directory, our app is now configured to return the homepage whenever a `GET` request is received for the URL `/`.
- The URL `https://module3.cs290.com/index.html` will map to the file `index.html` in the directory `public`.
- The URL `https://module3.cs290.com/contacts.html` will map to the file `contacts.html` in the directory `public`.
- The URL `https://module3.cs290.com/images/logo.png` will map to the file `logo.png` in the directory `public/images`.

Summary

In this exploration, we took a detailed look at how paths in the URL are mapped to files. We also studied the use of the `link` and `script` elements. Finally, we learned how Express provides an easy way to serve static files from a web application.

Additional Resources

- MDN has a good discussion of **absolute and relative URLs** **(https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL#absolute_urls_vs_relative_urls)** and a **primer on URL and paths** **(https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Creating_hyperlinks#a_quick_primer_on_urls_and_paths)**.
- The **link element on MDN** **(https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link)**.
- Serving static files using Express is discussed **here** **(https://expressjs.com/en/starter/static-files.html)**.