WebMound

Perfect Ways to Serve Static Files in Express and Node.js

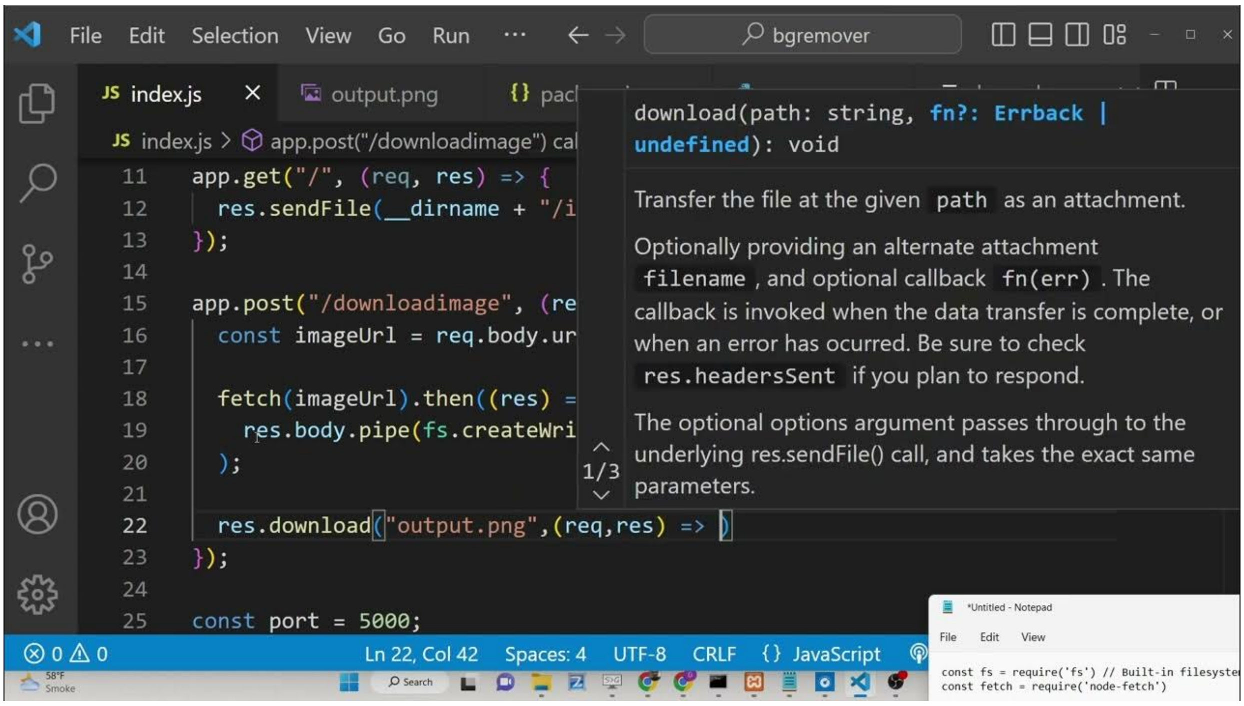
Robin Node.js Updated on April 30, 2022

Express has a built-in function to **serve static files from a Node.js server**. Using this function you can set a folder or multiple folders as public. This way anyone can access files present in those folders directly using the website URL.

In this article, you will learn multiple ways to serve static files in Express and Node.js. After reading this article you will be

- Serving static HTML files in Express for different routes.
- Serving a single-page application from an Express server.
- How to serve a static HTML Website in Express.

To get the best out of this lesson, you might need the following prerequisites. But don't worry you don't have to be an expert in all of these. I will explain every step in detail so that you don't face any problems. These prerequisites are:



```
app.get("/", (req, res) => {
  res.sendFile(__dirname + "/i
});

app.post("/downloadimage", (re
  const imageUrl = req.body.ur

  fetch(imageUrl).then((res) =
    res.body.pipe(fs.createWri
  );

  res.download("output.png",(req,res) => )
});

const port = 5000;
```

```
download(path: string, fn?: Errback |
undefined): void
```

Transfer the file at the given `path` as an attachment.

Optionally providing an alternate attachment `filename`, and optional callback `fn(err)`. The callback is invoked when the data transfer is complete, or when an error has ocurred. Be sure to check `res.headersSent` if you plan to respond.

The optional options argument passes through to the underlying res.sendFile() call, and takes the exact same parameters.

1/3

```
const fs = require('fs') // Built-in filesyste
const fetch = require('node-fetch')
```

- You must have Node.js version 12 or above installed on your computer.

setup using TypeScript guide.

What is Express Static?

In Express, the `static()` is a middleware function that is used to serve static files from an Express serve. It defines a folder as public so that any user can access files inside that folder.

You can use this middleware function with the `use()` method available in an Express application. The `express.static()` function accepts 2 parameters.

The first parameter is the path of the folder that you want to set as public. The second parameter is optional where you can set some additional options. Most of the time you will never need to set those options.

What are Static Files in Express and Node.js?

Static files refer to those files which browsers download while loading a web page. For example, JavaScript files, CSS files, images, etc.

When a browser displays a web page, it automatically downloads those files necessary for that page from the server. That's why these are serve as static files for your users. Then any browser will be able to access only those files directly on your server.

> ***Also Read: Easy Ways to Download Any File From Node.js Server Using Express***

Why Do We Need to Serve Static Files?

By default, Express servers don't allow browsers to download any files directly. So, if you don't explicitly serve static files necessary for your web pages then your website will break.

Without JavaScript files or CSS files, your website will not work. If your website displays images, you have to serve them statice. Otherwise, users can not see those images.

But interesting thing is that it is very easy to set it up. You can store all those files in a folder and set it as a static folder. That's it rest will be handled by your express server.

Let's see how you can set a static folder.

How to Serve Static Files in Express and Node.js

1. **Configure a Node.js project with Express:** You just have to configure a basic Node.js project by installing the necessary packages. You can also use **TypeScirpt with your Express server** if you prefer.
2. **Create a folder to store static files:** You have to create a folder at the root of your project. This folder will contain all of your static files.
3. **Create an Express server:** Now set up your Express server and listen to a port so that you can view it in a browser.
4. **Serve static files in Express:** In this step, you will use `express.static()` middleware function to define the created folder as static. Your express server will serve files inside that folder as static files.

Let's look into those steps in detail with some examples.

Step 1: Configure a Node.js Project with Express

First, you need to create a folder on your computer where you will store the Node.js project. I named my folder `node-express-static` but you can give any name for your folder. Run the following command in your terminal:

```
mkdir node-express-static
```

Go inside your newly created folder using the `cd` command.

```
cd node-express-static
```

Initialize a Node.js project with the default settings. It will create a `package.json` file at the root of your project. Open this folder in a text editor, you will find a `package.json` file.

This file will list all of your dependencies and dev-dependencies for this project.

```
npm init -y
```

Create a `app.js` file inside the `node-express-static` folder. It will be the entry file for your server.

```
touch app.js
```

Now install the `express` and `dotenv` packages as dependencies. Here `express` is the Node.js framework and the `dotenv` package is used to **set environment variables for Node.js and Express**.

To store environment variables, you have to create a `.env` file at the root of your project. Here you can keep sensitive information like port

```
npm install --save express dotenv
```

I will also install `nodemon` as a dev-dependency. This package will restart our server in the development environment when we update any code in our project.

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

Finally, I will add 2 new scripts to the `package.json` file. The `start` script will start our server using `node` command and the `dev` script will start our server using `nodemon` package.

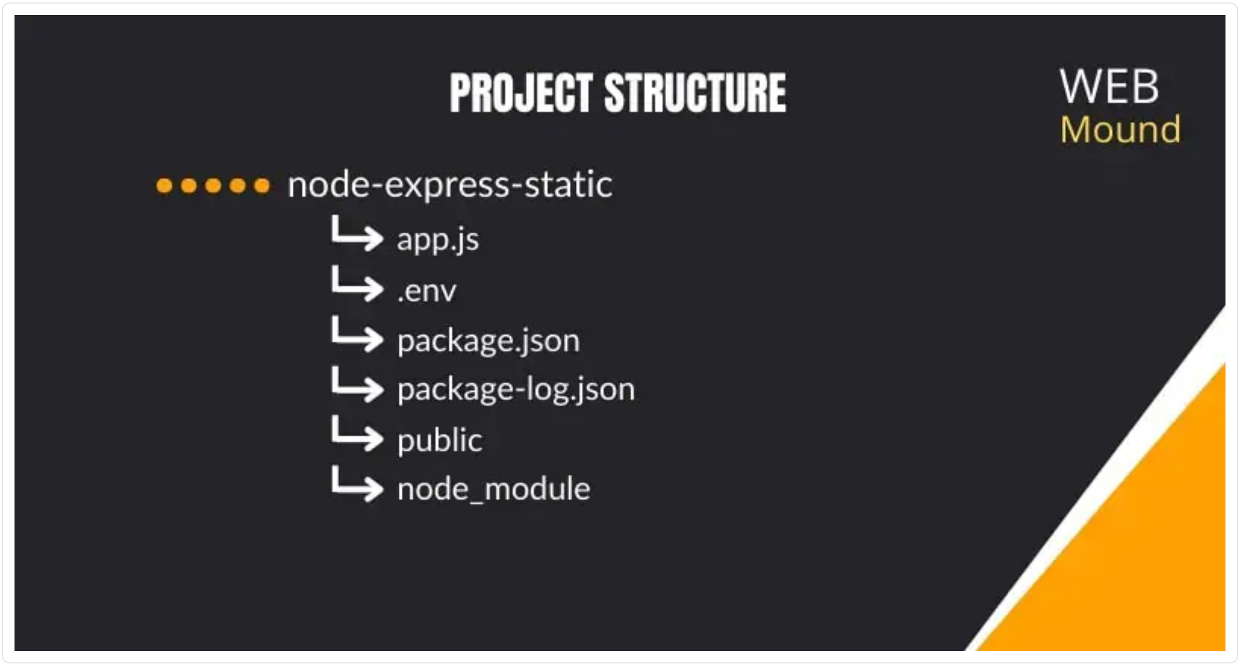
We will use the `start` script in a production environment and the `dev` script will be used during the time of development.

```
"scripts": {
  "start": "node app.js",
  "dev": "nodemon app.js"
}
```

Step 2: Create a Folder to Store Static Files

To store our static files create a folder called `public` inside the `node-express-static` folder. It's not necessary that you have to name your static folder as public. You can give any name you want.

Now in the public folder, I will create 3 new folders. These are `js`, `css`, and `images`. In the js folder, I will store all JavaScript files that are necessary for my website. In the `css` folder, I will store CSS files and the `images` folder will have the image files.



If we look at the project structure, we will see that there are 4 files and 2 folders in our project. These files are:

- package.json
- package-log.json
- app.js
- .env

And 2 folders are:

- public
- node_modules

Step 3: Create an Express Server

```
const { config } = require('dotenv');
const express = require('express');

const app = express();

// Load environment variables
config();

// Add routes
app.get('/', (req, res) => {
    res.send('Hello World');
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`);
});
```

Here, I am loading environment variables from `.env` file using the `config()` function. The `.env` file contains `PORT=5000` that I can access with `process.env.PORT` in our server.

I am also setting `5000` as a default value for the port number. If `process.env.PORT` returns `undefined` for any reason, our server will use `5000` as the port.

Now you can spin up the development server by running the `dev` script. Open your browser and visit `http://localhost:5000` to see the result.

Step 4: Serve Static Files in Express

Express has a built-in middleware function to set a folder as static. You have to use `express.static()` function inside `app.use()` function in our `app.js` file.

```
app.use(express.static(join(__dirname, 'public')));
```

With `app.use()` function, we are telling Express to execute the `express.static()` middleware function. It is good to use the absolute path in `express.static()` function for defining a static folder.

So, import `join()` function from path which is the **Node.js core module**. By using `join()` function, I am defining an absolute path for the `public` folder as the first argument in the `express.static()` function.

> ***Note:*** *You have to set your static folder before adding any routes.*

This is the complete look of our `app.js` file after serving static files:

```javascript
const { config } = require('dotenv');
const express = require('express');
const { join } = require('path');

const app = express();

// Load environment variables
config();

// Serve static files
app.use(express.static(join(__dirname, 'public')));

// Add routes
app.get('/', (req, res) => {
    res.send('Hello World');
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`);
});
```

That's it, you have successfully set a static folder. Therefore, Express will serve everything in the `public` folder as static files.

You can view any image by visiting `http://localhost:5000/images/logo.png`

```
<script src="/js/main.js"></script>
<link rel="stylesheet" href="/css/style.css">
```

Adding Virtual Prefix to Static File Path

If you want to add a prefix to all of your static file paths, you can do it. You have to pass the prefix in the `app.use()` function as the first argument.

```
app.use('/static', express.static(join(__dirname, 'public')));
```

In this example, I am passing **"/static"** as the path prefix. Now, to access images you have to visit `https://localhost:5000/static/images/logo.png` in the browser.

For JavaScript and CSS files, you also have to add `/static` in front of each file path. For example, instead of using `/js/main.js` you have to use `/static/js/main.js` in your HTML files.

```
<script src="/static/js/main.js"></script>
<link rel="stylesheet" href="/static/css/style.css">
```

differentiate between multiple folders.

Serve Static Files from Multiple Folders in Express

We can set multiple static files folders by using express.static() function multiple times. In this way, we can serve different types of files from different folders.

For example, we are serving our static files like images, JavaScript, and CSS files from the public folder. But you have an option on your website that individual user can upload their profile pictures.

Now, you want to store those profile pictures in a separate folder and serve from that folder. To achieve this functionality you have to set 2 folders as static.

```
app.use(express.static(join(__dirname, 'public')));
app.use(express.static(join(__dirname, 'uploads')));
```

Here, I am setting the `public` folder as static as I did previously. But I am also setting the `uploads` folder as static. Because when users will upload their profile pictures, I will store those inside this `uploads` folder.

By setting it as static, I am allowing my users to access their profile

Also Read: How to Get All Files in Directories (Recursively) with Node.js

But how can you distinguish the paths between those two static folders?

That's why you could use path prefixes for multiple static folders. For example, I can add `/public` prefix for the public folder and `/uploads` for the uploads folder. You can name these prefixes anything you want. It doesn't have to match the folder name.

```
app.use('/public', express.static(join(__dirname, 'public')));
app.use('/uploads', express.static(join(__dirname, 'uploads')));
```

When you see a path `/public/images/logo.png` like this, you can easily say that this is from the `public` folder. And when you see a path `/uploads/images/user-1.jpg` like this, starting with `/uploads` then you will know that it is from the `uploads` folder.

How to Serve a Single Static HTML File in Express

You can serve a single static HTML file from a route. Suppose you have a `views` folder at to root of your project. Inside this folder, you have 3 HTML files `home.html`, `about.html`, and `contact.html` respectively.

Now you want to send `home.html` file when a user visits the (`http://localhost:5000`) path. But when a user tries to visit your website with `/about` and `/contact` paths, you have to send `about.html` and `contact.html` files respectively.

```javascript
const { config } = require('dotenv');
const express = require('express');
const { join } = require('path');

const app = express();

// Load environment variables
config();

// Serve static files
app.use('/uploads', express.static(join(__dirname, 'uploads')));
app.use('/static', express.static(join(__dirname, 'public')));

app.get('/', (req, res) => {
    res.sendFile(join(__dirname, 'views', 'home.html'));
});

app.get('/about', (req, res) => {
    res.sendFile(join(__dirname, 'views', 'about.html'));
});

app.get('/contact', (req, res) => {
    res.sendFile(join(__dirname, 'views', 'contact.html'));
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`)
```

I have created an Express server with 3 separate routes for the home, about, and contact page. From each route, I can send a single HTML file using `res.sendFile()` function.

This function accepts an absolute path for any file. That's why you have to use `join()` function imported from the Node.js core `path` module.

For the home route, I am sending `home.html` file. For `/about` and `/contact` routes, I am sending `about.html` and `contact.html` files respectively.

Serving a Single Page Application in Express

Sometimes you might want to use a Node.js server to serve a single page application created with **React**, **Vue**, **Svelte**, or any other frontend framework.

You can serve your single-page application along with APIs that you can use in your application to get data from the backend.

For example, you have your single-page application in the `client` folder which is at the root of our project. To serve this single-page application for an Express server, you have to set this folder as a static folder using `express.static()` function.

```
const { config } = require('dotenv');
const express = require('express');
const { join } = require('path');

const app = express();

// Load environment variables
config();

// Serve static files
app.use('/uploads', express.static(join(__dirname, 'uploads')));
app.use('/static', express.static(join(__dirname, 'public')));

// API routes
app.get('/api', (req, res) => {
    res.json({
        success: true,
        message: 'Welcome to the API',
    });
});

// Serving a single page application
app.use(express.static(join(__dirname, 'client')));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`);
});
```

you can not use any path prefix for it. If you try to add a path prefix in the `app.use()` function, your application will not work.

I have also added a demo route to send JSON data for the frontend application. You are allowed to add as many API routes as you want.

> ***Important Note:*** *You have to serve your single-page application at the end of all routes and static folders. If you serve your application before any routes then those routes will not work. Because by default for those routes Express will try to serve the single-page application.*

That's it. Express will serve your single-page application for any routes that don't match with the paths of the static files or the API routes.

You can also serve a static HTML website from an Express server. Earlier we have seen how we can send a single static HTML file from a route. But using a static HTML website structure, you can serve a complete website without creating each route separately in the Express server.

For example, you have a `static-webite` folder that contains a static HTML website. My demo website has 3 pages home, about, and contact page. To serve it from an Express server, you have to set this folder as static.

```
const { join } = require('path');

const app = express();

// Load environment variables
config();

// Serve static files
app.use('/uploads', express.static(join(__dirname, 'uploads')));
app.use('/static', express.static(join(__dirname, 'public')));

app.get('/api', (req, res) => {
    res.json({
        success: true,
        message: 'Welcome to the API',
    });
});

// Serving static HTML website
app.use(express.static(join(__dirname, 'static-website')));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`);
});
```

Like a single-page application, you have to follow the same method. You can also add any other routes that are not available on your static HTML website.

For example, in my static HTML website, I have a route `/about` that shows the about page. But If I add a `/about` route in my Express server as well then Express will serve the content from this route when a user visits.

That means the routes in the express server that match with the static HTML website routes will overwrite them. You can add unique routes to

website at the end of all other routes and static files in the Express server.

Conclusion

Express doesn't support static file sharing by default. But you can set up your server to serve different types of files using a built-in function.

In this lesson, I have shown you multiple ways to serve static files using express.static() function. Not only that you have also learned to send single HTML files, and serve a single page application and a static HTML website.

Now you will be able to easily structure your Node.js and Express server with different types of static resources. I hope it was helpful for you.

Thank you so much for reading. Happy coding!

Related Posts

Robin Node.js

Easiest Way to Set & Use Environment Variables (.env) in Node.js

Robin Node.js

Getting Data From req.body in Node.js & Express Server

Robin Node.js

How to Delete Files in Node.js Safely: Step-by-Step Guide

Recent Posts

How to Remove All Duplicate Values From a Slice in Golang

Generate Type Declaration (.d.ts) Files Using TypeScript

How to Add Custom Types For NodeJS process.env in TypeScript

Add Any Number of Dynamic Properties to TypeScript Objects

Add Properties in TypeScript Based on Another Property Value

Magic of Empty Interface in Golang: Different Ways to Use It

Categories

JavaScript

CSS

HTML

Web Dev

Vue

Node.js

TypeScript

Golang

Blog About Contact Terms Privacy Policy Disclaimer