

How to work with Hardcoded data in your Backend.

There are, at least, three different ways of working with hardcoded data inside your server backend.

Hardcoded data refers to data that is directly embedded into the source code of a program, rather than being loaded from an external source or generated dynamically at runtime. Hardcoded data can include things like default configuration settings, fixed values for variables or constants, or even complete sets of data that are used as input for a program.

Hardcode Data inside your Express route

- This is an appropriate solution when your information is short so your data could be easy to read

In an Express application, you can use hardcoded data by defining it as a variable or constant within your code. This can be useful for things like default configuration settings or fixed data that is used by your application.

Here's an example of how to use hardcoded data in an Express application:

techtonica

```
.js index.js > ...
1 | import express from 'express';
2 | const app = express();
3 |
4 | // Define some hardcoded data
5 | const users = [
6 |   { id: 1, name: 'Alice' },
7 |   { id: 2, name: 'Bob' },
8 |   { id: 3, name: 'Charlie' }
9 | ];
10 |
11 | // Define a route that uses the hardcoded data
12 | app.get('/users', (req, res) => {
13 |   res.json(users);
14 | });
15 |
16 | // Start the server
17 | app.listen(3000, () => {
18 |   console.log('Server started on port 3000');
19 | });
```

You may also see this handled in the following way:

```
javascript Copy code
const express = require('express');
const app = express();

// Define some hardcoded data
const users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' },
  { id: 3, name: 'Charlie' }
];

// Define a route that uses the hardcoded data
app.get('/users', (req, res) => {
  res.json(users);
});

// Start the server
app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

In this example, we define an array of hardcoded data called `users`, which contains three user objects. We then define an Express route for the URL path `/users`, which simply returns the entire `users` array as a JSON response.

When the application is started, it will listen for incoming HTTP requests on port 3000. When a request is made to the `/users` URL, the application will respond with the `users` array as a JSON object.

Of course, in a real-world application, you would likely want to use some kind of database or external data source instead of hardcoded data, but this example should give you an idea of how to use hardcoded data in an Express application.

Hardcode Data from another JS file inside your Express route

- This is an appropriate solution when your information is vast, and your data is not easy to read

To add hardcoded data from another JavaScript file to an Express router, you can export the data from the other file and then import it into the router file.

For example, suppose you have a file called data.js with the following hardcoded data:

```
const data = [  
  { id: 1, name: 'John' },  
  { id: 2, name: 'Jane' },  
  { id: 3, name: 'Bob' }  
];  
  
module.exports = data;
```

You can then import this data in your router file and use it in a route handler like so:

techtonica

```
JS index.js > [⌕] default
1  import { Router } from 'express';
2  import data from './data';
3
4  const router = Router();
5
6  router.get('/users', (req, res) => {
7    res.json(data);
8  });
9
10 export default router;
```

You may also see this handled in the following way:

```
javascript Copy code
const express = require('express');
const data = require('./data');

const router = express.Router();

router.get('/users', (req, res) => {
  res.json(data);
});

module.exports = router;
```

In this example, the data variable is imported from the data.js file and used in the route handler for the /users endpoint to send a JSON response containing the hardcoded data.

Note that in order to import the data from the data.js file, both files must be located in the same directory.

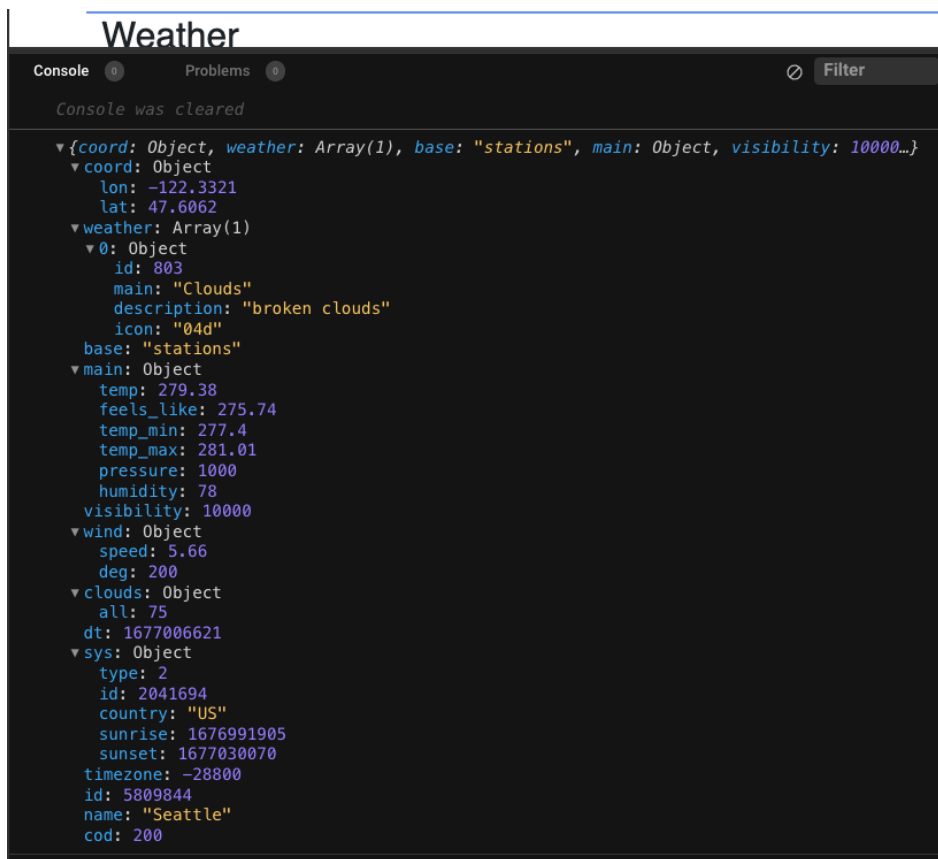
NOTE: Please note that we're simplifying the example in the code spinet for readability. If you want to see a real-life example of importing a vast file with information in a server, [please check this code](#).

Hardcode Data from an API inside your Express route

- This is an appropriate solution when you're trying a new or paid API. This will allow you to test that your connections are working properly before calling for information that will be updated programmatically

Imagine that you're doing a project with a paid API that will charge you for each request. You don't want to spend money setting up the connections while you make sure that your routes are working. If you are working with documentation from your API you will have access to examples of data that you will receive. You can use those examples in your Express router, then import them into the router file.

For example, suppose you are calling the paid services of the Weather API (<https://openweathermap.org/api>) and you already have read the documentation so you know that the API will response with an object like this one



```
Weather
Console Problems Filter
Console was cleared

▼ {coord: Object, weather: Array(1), base: "stations", main: Object, visibility: 10000...}
  ▼ coord: Object
    lon: -122.3321
    lat: 47.6062
  ▼ weather: Array(1)
    ▼ 0: Object
      id: 803
      main: "Clouds"
      description: "broken clouds"
      icon: "04d"
    base: "stations"
  ▼ main: Object
    temp: 279.38
    feels_like: 275.74
    temp_min: 277.4
    temp_max: 281.01
    pressure: 1000
    humidity: 78
    visibility: 10000
  ▼ wind: Object
    speed: 5.66
    deg: 200
  ▼ clouds: Object
    all: 75
    dt: 1677006621
  ▼ sys: Object
    type: 2
    id: 2041694
    country: "US"
    sunrise: 1676991905
    sunset: 1677030070
    timezone: -28800
    id: 5809844
    name: "Seattle"
    cod: 200
```

techtonica

You can then import this data in your router file and use it in the same form that you use data in a file in your system. Then, only when your test is done, and your front-end works correctly do you change your connections to bring real data.

```
JS index.js JS data.js X {} package.json
JS data.js > [e] coord > weather > main
1  const coord = {
2    lon: -122.3321,
3    lat: 47.6062,
4    weather: [{
5      id: 803,
6      main: "Clouds",
7      description: "broken clouds",
8      icon: "04d",
9      base: "stations",
10     main: {
11       temp: 279.38,
12       feels_like: 275.74,
13       temp_min: 277.4,
14       pressure: 1000,
15       humidity: 78
16     },
17     visibility: 10000,
18     wind: {
19       speed: 5.66,
20       deg: 200
21     },
22     clouds: {
23       all: 75
24     },
25     dt: 1677006621,
26     sys: {
27       type: 2,
28       id: 2041694,
29       country: "US",
30       sunrise: 1676991905,
31       sunset: 1677030070
32     },
33     timezone: -28800,
34     id: 5809844,
35     name: "Seattle",
36     cod: 200
37   }
38 ]
39 }
40
41 module.exports = coord;
```