Section

# How to Generate Fake Data in Node.js Using Faker.js

May 20, 2021

Topics: Node.js

During system development and testing, employment of Fake data can be very useful. This is because it prevents one from using ones real identity especially in for data like identification table. We shall run our process locally on our computer but fetch the information from an online server.

## Key takeaways

By the end of this tutorial, you will know the following:

- Main components of Faker.js.
- How to create a simple webpage structure.
- How to generate the fake data.
- Storing the fake data in a table.
- Linking the Fake data generated to the webpage.

## Pre-requisites

The basic requirements for this course include:

- Basic knowledge of Web development.
- Node.js basics.
- A basic IDE is installed on the machine. We shall preferably use Visual Studio Code.

## Table of contents

## What is Faker.js

Faker.js is a Node.js library that allows users to generate massive amounts of fake data for their project use. This can be generated either while running your program locally or remotely by deploying it in a web browser as a webpage. It does this generation by randomly choosing

Sect

## What type of data does Faker.js generate

Some examples of data that can be generated include, but not limited to the below categories:

- Address
- Animals
- Companies
- Commerce
- Dates
- Data types
- Images
- Names
- Time and many more…

You can choose a sub-category by first requiring “faker”, calling the main category, then adding the sub-category after a period. An example is `console.log('faker.animal.dog');`

Code.

Open the integrated terminal in Visual Studio Code and run `npm init -y` to quickly create a “package.json” file.

Create a file in the main directory named “app.js”. This will be the program’s starting point.

Once done, let us create two new folders in the main directory namely “views” and “models”. Inside the “views” directory, create “home.ejs” file. Create another file and name it “user.js” inside the “models” folder.

## Folder structure

The folder structure will be as shown below:

> Fakedata (Root Directory)
>
> > node_modules (folder)
> >
> > models (folder)
> >
> > > user.js (file)

> > app.js (file)

## Setting up our environment

Let’s do the following in sequence:

### Installing packages

The Node.js libraries required for this tutorial include:

- nodemon
- express
- ejs
- mongoose
- faker

In the integrated terminal, run the following command to install the packages above all at once:

```
npm i nodemon express ejs mongoose faker
```

Now open the "package.json" file and configure it so that you can start the application using nodemon. Under the scripts, add the “dev” and “start” configurations.

The code should look as follows:

```
{
    "name": "fakedata",
    "version": "1.0.0",
    "description": "",
    "main": "app.js",
    "scripts": {
        "start": "node app.js",
        "dev": "nodemon app.js",
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "",
    "license": "ISC",
    "dependencies": {
        "ejs": "^2.7.4",
        "express": "^4.17.1",
        "faker": "^4.1.0",
        "mongoose": "^5.12.5",
        "nodemon": "^2.0.7"
    }
}
```

twenty times due to the for loop used. You can see the data generated in the control panel.

```
var faker = require("faker");

// Initializing our variables with a different random data each time it
var randomName = faker.name.findName(); // Generates a random name
var randomEmail = faker.internet.email(); // Generates a random email
var randomProduct = faker.commerce.productName(); // Generates a random
var randomCompany = faker.company.companyName(); // Will give back a ra
var randomCard = faker.helpers.createCard(); // It's output is a random

// Iteration
// This code runs twenty times
// It produces each time different data
for (i = 0; i < 20; i++) {
    console.log(randomName); // Outputs a random name
    console.log(randomEmail); // Outputs a random email
    console.log(randomProduct); // Outputs the random product name gene
    console.log(randomCompany); // Produces a random company name
    console.log(randomCard); // Gives back a random card
    console.log(faker.date.past()); // Generates a random past date
}
```

Run the “app.js” file in the terminal using the command `node index.js`.

Open a new PowerShell terminal and run it as Admin.

Type in `get-executionpolicy` to check the current status.

If it returns as “Restricted”, run `set-executionpolicy remotesigned`.

Type “A” in the command line dialogue, to quickly accept all the changes, then press Enter.

For further errors, you can find help here or you can copy and paste the error and search for help in Stack overflow.

Now that we have seen our tests running successfully, we can get into the main application.

## How the project will run

The code when executed will use the faker.js library to generate the data needed several times. It will store each value generated in a local database. We’ll then see the results in a web browser as a webpage from the database.

## Setting up the application starting point

```
// Require needed packages
var express = require("express");
var mongoose = require("mongoose");
var faker = require("faker");
var path = require("path");
var fakerModel = require("./models/user");
```

Connect to a new database called “fakedata” inside Mongodb database.

This is shown below:

```
// Connect to a local Mongodb database called fakedata
// If successful log out on the console "connected to db"
//    else "connection error"
mongoose
    .connect("mongodb://localhost:27017/fakedata", { useNewUrlParser: t
    .then(() => console.log("connected to db"))
    .catch((error) => console.log("connection error", error));
```

Use express to set up our Template engine (view engine) to “ejs” and set up our absolute directory path to the source directory.

This is shown below:

Set up our simple routing system to take us to the home webpage whenever the application is launched or new data generated. We'll also save all our data generated and stored in the variables inside our database:

```
// Set up our simple routing system to take us to the "home.ejs" file
// or the home webpage whenever the application is launched
app.get("/", (req, res) => {
    fakerModel.find((err, data) => {
        if (err) {
            console.log(err);
        } else if (data) {
            res.render("home", { data: data });
        } else {
            res.render("home", { data: {} });
        }
    });
});

// Return each data generated to the variables below and save all of th
// This will  be done ten times due to the for loop
// You can add the number of data to be generated by changing the value
app.post("/", (req, res) => {
    for (var i = 0; i < 10; i++) {
        var fakee = new fakerModel({
            firstname: faker.name.firstName(),
    }
    res.redirect("/");
});
```

Configure the ports to be used:

```
// Set up our ports in which we shall see the webpage and data generate
// We will use 5000 as our port number
var port = process.env.PORT || 5000;

app.listen(port, () => console.log("server running at port " + port));
```

As you can see, we have added a *for loop* which will be repeated ten times. This in turn adds ten new rows of data each time it is run. You can increase or decrease the number of data to be generated by changing the value as you please.

## Setting up models

### user.js

Sect

```
// Require the mongoose model
var mongoose = require("mongoose");

// Add the following columns in it and their datatypes
var fakerSchema = new mongoose.Schema({
    firstname: String,
    lastname: String,
    phonenumber: String,
    city: String,
    state: String,
    country: String,
});

// Require it as the output
module.exports = mongoose.model("fakerCollection", fakerSchema);
```

## Setting up views

### home.ejs

In this file, we will create a web structure to display the data. It will have a button that when pressed, will generate new data and automatically save while displaying it.

We'll also add a table that will have the same number of columns as the

```
    <head>
        <title>Fake Data Generator</title>
        <!-- Required meta tags -->
        <meta charset="utf-8" />
        <meta
            name="viewport"
            content="width=device-width, initial-scale=1, shrink-to-fit
        />

        <!-- Bootstrap CSS -->
        <link
            rel="stylesheet"
            href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/cs
            integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJ
            crossorigin="anonymous"
        />

        <!-- Custom CSS -->
        <style>
            @import url("https://fonts.googleapis.com/css2?family=Monts
            * {
                font-family: Montserrat;
            }

            .card {
                margin: 0 auto;
                /* Added */
                float: none;
                /* Added */
                margin-bottom: 10px;
            integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smX
            crossorigin="anonymous"
        ></script>
        <script
            src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.
            integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa
            crossorigin="anonymous"
        ></script>
        <script
            src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/
            integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/
            crossorigin="anonymous"
        ></script>

        <!-- Main Container -->
        <div class="container">
            <!-- Header -->
            <br />
            <h1 class="text-center">FAKE DATA GENERATOR</h1>
            <hr />
            <p class="text-secondary">This is a simple Fake Data genera
            <hr />
            <!-- Header end.//-->
        </div>
        <!--container end.//-->
    </body>
</html>
```

```
        value="click to generate the data"
    />
</form>
<br />
<!-- Generate button end.//-->

<a class="btn btn-primary stretched-link" href="#footer">Go to the bott
<hr />
```

As you can see, it will trigger the process when clicked. The additional link will enable us to use the in-page HTML links to navigate from the top to bottom of the page quickly at a button click as we shall see later on.

> Add a table and use the “ejs” template engine format when referring to the variables to be used. This shall be placed below the button in the “container” div.

See the code below:

```
<!-- Table -->
<%if(data.length>0){%>
<table
    class="table table-striped table-inverse table-responsive"
```

```
            <th>state</th>
            <th>country</th>
        </tr>
    </thead>
    <tbody>
        <%for(var i=0;i< data.length; i++){%>
        <tr>
            <td><%= i%></td>
            <td><%= data[i].firstname%></td>
            <td><%= data[i].lastname%></td>
            <td><%= data[i].phonenumber%></td>
            <td><%= data[i].city%></td>
            <td><%= data[i].state%></td>
            <td><%= data[i].country%></td>
        </tr>
        <%}%>
    </tbody>
</table>
<%}%>
<!-- Table end.//-->
```

Just to make it easy for one with large amounts of data, we'll add another "Generate" button at the end of the page. We will now add another link that has the id of "footer". We had set the initial "Go to the bottom" link to point at it through its id.

All these we shall put inside the footer element as shown below:

```
<!-- Generate button-->
<form action="/" method="post">
    <input
        type="submit"
        class="btn btn-primary btn-lg btn-block"
        value="click to generate the data"
    />
</form>
<br />
<!-- Generate button end.//-->
<footer id="footer">Bye...</footer>
<!-- Footer section end.//-->
```

## Run the project

Before running the code, make sure that MongoDB is running in the background.

In your integrated terminal, run the following command:

```
nodemon run dev
```

Fakedata home.ejs webpage

You can view the generated data in MongoDB installed on your computer. Just open the “MongoDB Compass” and connect to the default localhost, that is, at `localhost:27017`. You will see a new database called “fakedata”. Once you open it, you will see a collection by the name “fakercollection”.

All the data generated in the session will be found there.

You can find, clone, and use the code above in the repository found here.

Congratulations! You have generated fakedata for your need using Node.js.

## Conclusion

Fake data is very important for system testing. One can obtain as much as needed and store it locally for immediate or future use using Node.js modules. One can choose a variety as he/she needs depending on the underlying purpose.

## Further projects

You can also decide to do an intense AI-driven analysis on the data generated to know more about the patterns the data generated has.

Happy coding!

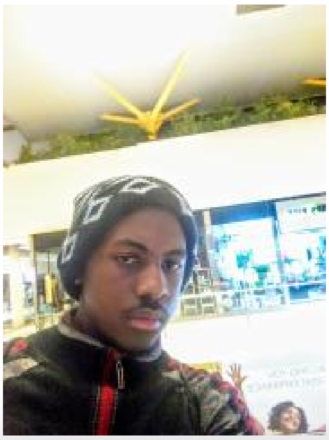
## References

The following were used as references:

Faker.js documentation.

Peer Review Contributions by: Peter Kayere

### About the author



Justus Mbuvi

Justus Mbuvi is an undergraduate student pursuing a Bachelor of Science in Computer Technology at Jomo Kenyatta University of Agriculture and Technology. He is interested in System development, Network Automation,

Education Program. Please report any errors or innaccuracies to enged@section.io.

## Want to learn more about the EngEd Program?

Discover Section's community-generated pool of resources from the next generation of engineers.

Learn more

+ QUICK LINKS // More Section offerings

Careers

Legals

**Resources**

Blog

Case Studies

Content Library

Solution Briefs

Partners

Changelog

**Support**

Docs

Community Slack

Help & Support

Platform Status

Pricing

Contact Us