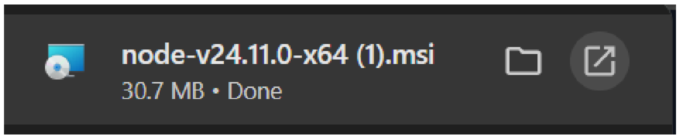
Intro to Back-End Workshop - Guide

Setting Up Your Project

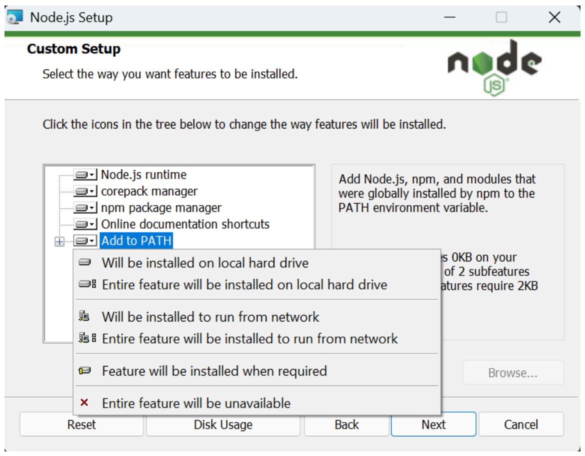
What you need first:

1. VS Code (or any IDE installed)
2. Node.js installed
   - https://nodejs.org/en/download→ Select your OS (E.g., Windows) then click Windows Installer

   1. Open the installer:



   2. Go through the set-up and make sure to click the little + button to add Node to your PATH:



Node.js Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

- Node.js runtime
- corepack manager
- npm package manager
- Online documentation shortcuts
- Add to PATH

Add Node.js, npm, and modules that were globally installed by npm to the PATH environment variable.

s 0KB on your
of 2 subfeatures
atures require 2KB

- Will be installed on local hard drive
- Entire feature will be installed on local hard drive
- Will be installed to run from network
- Entire feature will be installed to run from network
- Feature will be installed when required
- Entire feature will be unavailable

   3. Continue through the set-up and then allow Node to make changes to your computer.

4. Verify that it is installed correctly by typing

   i. `node --version` into your terminal. If it displays a number you are good to go.

   Note: You may receive this error:

   ```
   node : The term 'node' is not recognized as the name of a cmdl
   et, function, script file, or operable program. Check
   the spelling of the name, or if a path was included, verify that th
   e path is correct and try again.
   At line:1 char:1
   + node --version
   + ~~~~
       + CategoryInfo          : ObjectNotFound: (node:String) [], Com
   mandNotFoundException
       + FullyQualifiedErrorId : CommandNotFoundException
   ```

   To fix this, re-open your terminal as an administrator and try again.

   ii. Then type: `npm --version` into your terminal to see if it also displays a number.

Creating Your Project:

1. Make a new folder for your project or use the existing project folder and create a directory for the backend.

   Note: you can do this in VS Code directly or in your terminal using the commands `mkdir` and then `cd`

2. Start your node project:

   a. Inside your `backend` folder run: `npm init -y`

      This creates your package.json folder, which helps you keep track of tools. You may already have one in your main directory from the previous workshops.

   b. Install Express.js:

      a. In your terminal: `npm install express`

c. Create your server file:

    a. Inside your `backend` folder, make a new file called `server.js` and copy this code into it:

```
const express = require("express");
const app = express();
const PORT = 3000;

app.get("/", (req, res) => {
  res.send("Hello from your MERN backend!");
});

app.listen(PORT, () => {
  console.log(
    `Server is running! Open http://localhost:${PORT} in your brows
er`
  );
});
```

d. Run your server:

    a. Inside your terminal: `node server.js`

    b. To test: Ctrl + click on the link displayed or paste it into your browser. You should see 'Hello from MERN Backend!"

Creating Your Own RESTful API

You should already have your project (including the server) set-up (step 2 of 'Creating your project'). From there we...

1. Add RESTful Endpoints (Routes):

   For this example we will consider our API is for a list of animals, and for simplicity, we'll use a pretend list right inside our server so no database is required.

   i. Make a new directory inside VS code or your terminal: `mkdir routes`

ii. Navigate to inside your routes folder and make a new file called `animals.js`

Your project structure will now look similar to this:

```
backend/
├── server.js
├── package.json
├── routes/
│   └── animals.js
```

Note: It is possible for your backend folder to be a sub-folder of your larger project structure.

iii. Create your API and commands for your animal API inside `animals.js` . For example:

```
const express = require("express");
const router = express.Router();

let animals = [
  { id: 1, name: "Lion" },
  { id: 2, name: "Elephant" },
];

// 1. GET: See all animals
app.get("/animals", (req, res) => {
  res.json(animals);
});

// 2. POST: Add a new animal
app.post("/animals", (req, res) => {
  const newAnimal = { id: animals.length + 1, name: req.body.name };
  animals.push(newAnimal);
  res.json(newAnimal);
});

// 3. PUT: Change an animal’s name
```

```
app.put("/animals/:id", (req, res) => {
  const animal = animals.find((a) => a.id == req.params.id);
  if (animal) {
    animal.name = req.body.name;
    res.json(animal);
  } else {
    res.status(404).send("Animal not found");
  }
});

// 4. DELETE: Remove an animal
app.delete("/animals/:id", (req, res) => {
  animals = animals.filter((a) => a.id != req.params.id);
  res.send("Deleted!");
});

module.exports = router;
```

2. Connect `animals.js` to your `server.js` file:

   Go back to your server.js file from the setup, and update it to include the animal API. For example, your server.js would now look like:

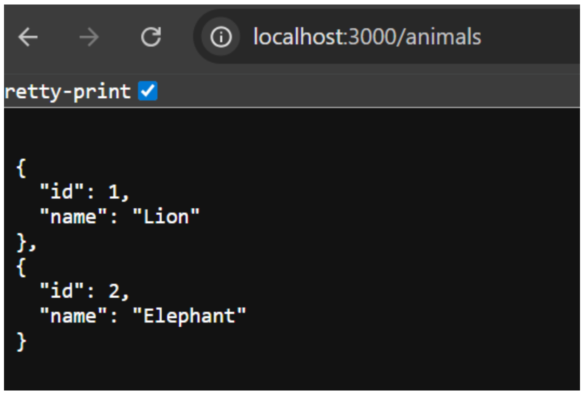
```
const express = require("express");
const app = express();
const animalRoutes = require("./routes/animals");

app.use(express.json());
app.use("/animals", animalRoutes);

app.listen(3000, () => {
  console.log("Server running at http://localhost:3000");
});
```

3. Test it out!

   a. To simply view your API:

i. Run `node server.js` in your terminal

ii. Paste `http://localhost:3000/animals` into your browser. You should see something along the lines of:



```
{
  "id": 1,
  "name": "Lion"
},
{
  "id": 2,
  "name": "Elephant"
}
```

b. Use Postman to test API commands:

a. Open Postman (program or web version)

b. Click on the big + tab to start a new request

c. Set the method to GET (default)

d. Enter `http://localhost:3000/animals` into the URL bar

e. Click send

Now you'll see the JSON list of animals under the "Body" tab.

Not working? Check these steps first:

1. Make sure your server is still running (don't close the terminal or vs code window)
2. Double-check your URL is correct
3. Look at your terminal for error messages
4. Make sure you have `app.use(express.json());` in your `server.js` file before you make any requests

Connecting to an Existing API

Existing API we'll use in this demo is the https://dog.ceo/dog-api/

(Optional) Try out the API first

1. Open your browser and paste this URL into the address bar:

   ```
   https://dog.ceo/api/breeds/image/random
   ```

2. Press enter → you'll see some JSON text with a link to a random dog image.
3. Paste the image link into a new browser tab to be able to see the image.

Creating a Node.js script to make API requests:

Important note: You can do this without needing to complete step 2 of "Creating Your Project"

1. Make a new folder inside your `backend` directory (for example `mkdir dog-api`)
2. Inside your terminal, run the following commands:
   a. `npm init -y`
   b. `npm install node-fetch`
3. Create a file inside your new folder called `dog.js` . This is where you'll put your requests. Add the following code:

   ```
   const fetch = require('node-fetch');

   async function getRandomDogImage() {
     const res = await fetch('https://dog.ceo/api/breeds/image/random');
     const data = await res.json();
     console.log('Here is your random dog image:', data.message);
   }

   getRandomDogImage();
   ```

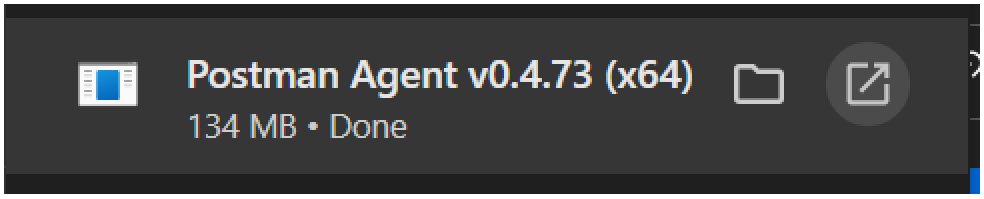
4. In the terminal, run: `node dog.js`

Note: The dog API is read-only so we can't create or change any data.

Postman

Setting Up:

1. Go to https://www.postman.com/downloads/ and download Postman (or you can use the web browser version)
2. Open the download:

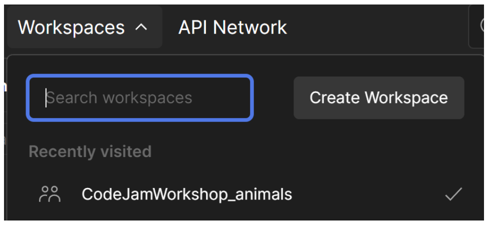


3. Sign in or sign up for free

Connecting your API:

1. Select 'create new workspace'
2. Fill in the details:
    a. Give it a name related to your project
    b. (Optional) Summary or description
    c. (Optional) Choose how you want the workspace to be shared
3. Click "Create"

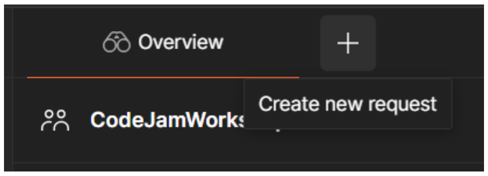
Note: You can easily select and switch between workspaces in Postman by clicking on "Workspaces" at the top left:



Creating a new request:

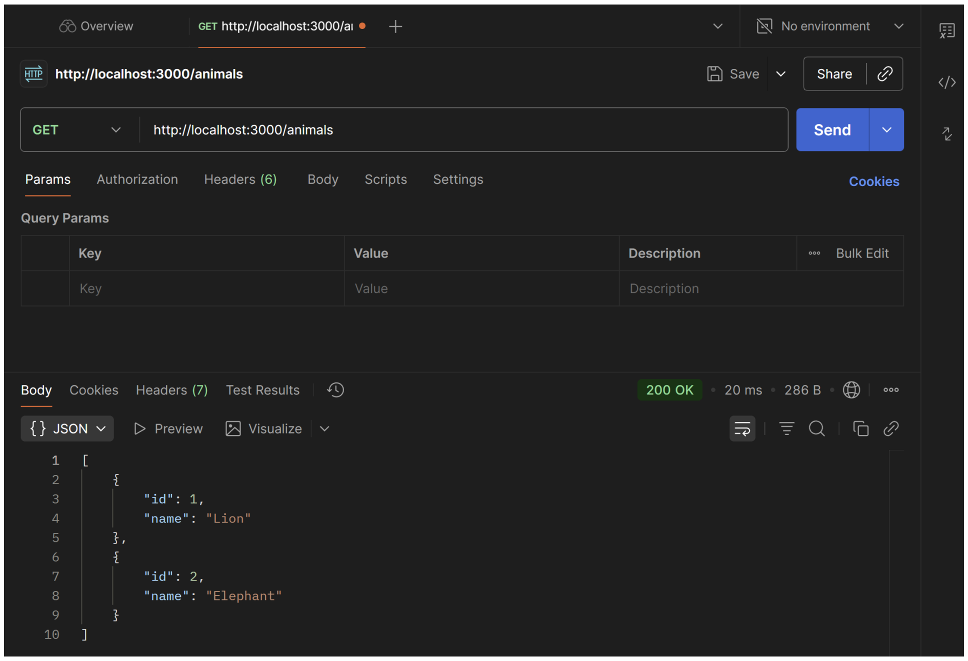
Sending GET Requests:

1. Click on the big + at the top:



2. In the left dropdown, select GET (default value)
3. Paste your local host link
4. Click send

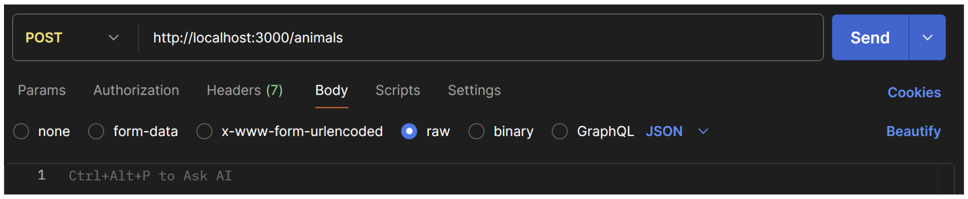
   You should be able to see your list of animals in the body:



```
[
    {
        "id": 1,
        "name": "Lion"
    },
    {
        "id": 2,
        "name": "Elephant"
    }
]
```

Making a POST Request:

1. Click on the big + at the top
2. Choose POST from the dropdown
3. Paste your API link
   a. For our example it is still `http://localhost:3000/animals`
4. Click on the “Body” tab under the URL
5. Select “raw” and choose JSON:

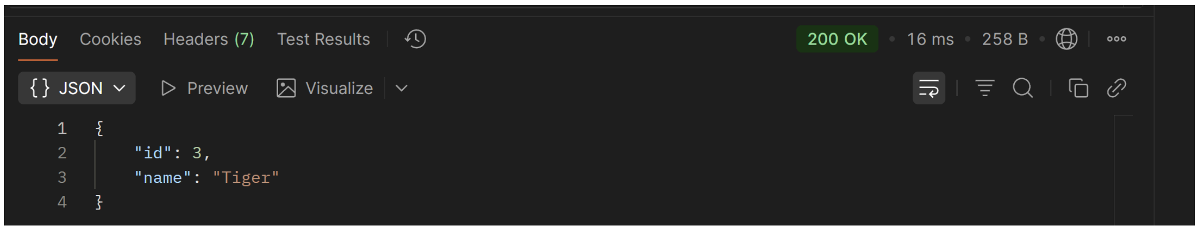


6. Type your data
   a. E.g.,

```
{
   "name": "Tiger"
}
```

7. Click Send

   Then you should be able to see the animal you just added in the same body as the GET results.



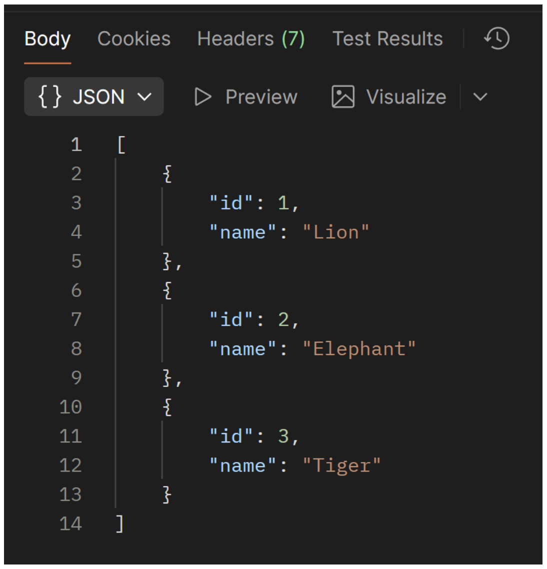
```
{
    "id": 3,
    "name": "Tiger"
}
```

Making a PUT Request:

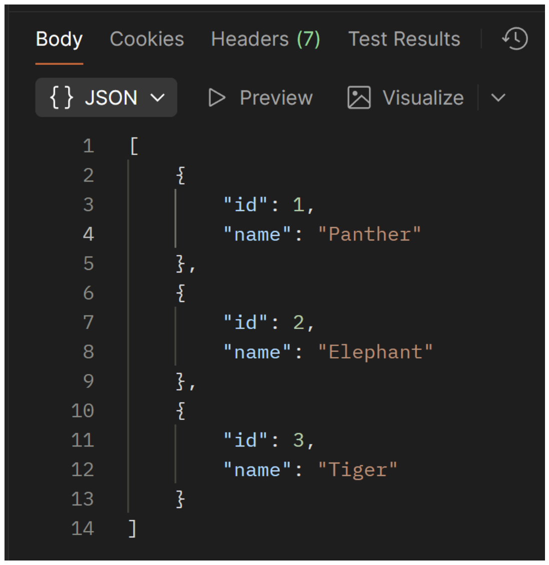
1. Create a new request by pressing the + button
2. Select PUT from the dropdown and add your API URL, but this time with the id of the data you want to change.
   a. E.g., To change the animal with id = 1, we would paste `http://localhost:3000/animals/1`
3. Then select Body, raw + JSON (same as step 5 for POST requests) and type your update.
4. Click send

   The update animal object will be displayed. For the animal example, we have:

Running a GET request before PUT: Running a GET request after PUT:



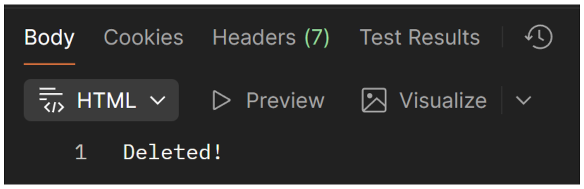
```
[
    {
        "id": 1,
        "name": "Lion"
    },
    {
        "id": 2,
        "name": "Elephant"
    },
    {
        "id": 3,
        "name": "Tiger"
    }
]
```



```
[
    {
        "id": 1,
        "name": "Panther"
    },
    {
        "id": 2,
        "name": "Elephant"
    },
    {
        "id": 3,
        "name": "Tiger"
    }
]
```

Making a DELETE Request:

1. New request tab, select DELETE
2. Paste the URL with the id of the data you want to delete
    a. E.g., To delete the panther: `http://localhost:3000/animals/1`
3. Click send, and the result will be a confirmation message.



```
Deleted!
```