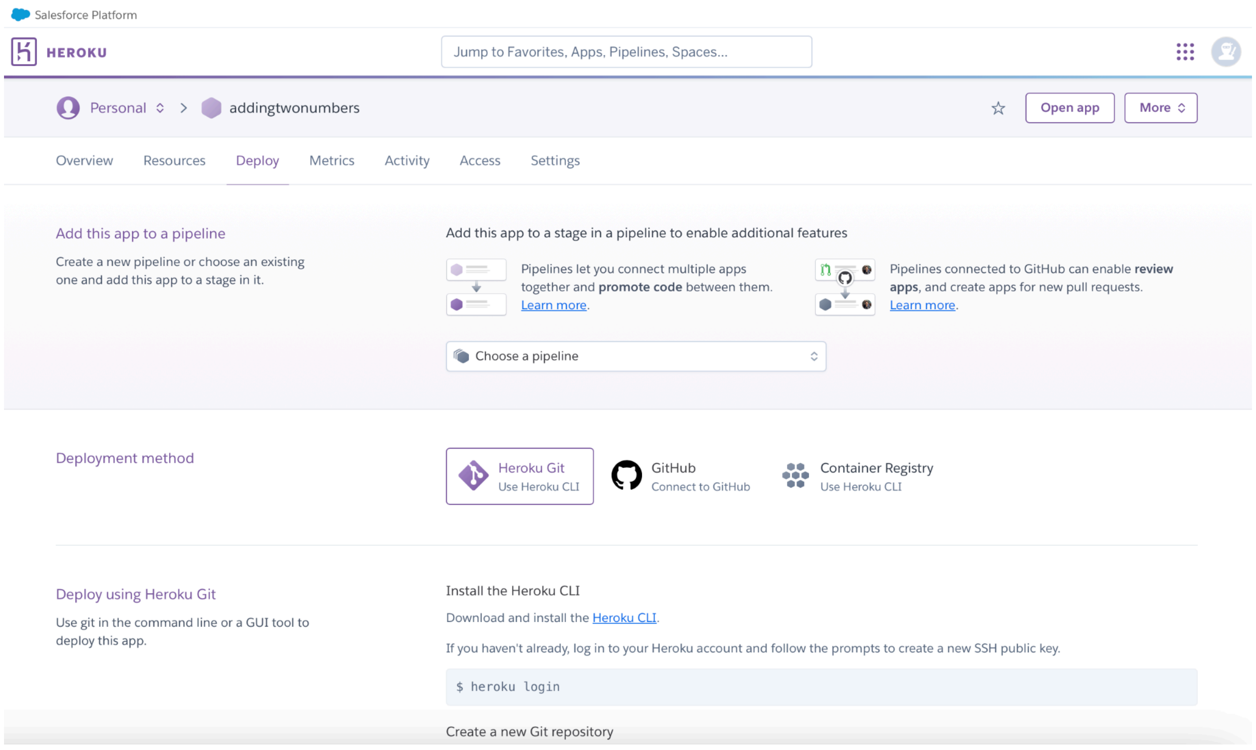
# Deployment of node.js application on heroku

## Step-1 (Setting Up Heroku for Node.js Deployment)

Signup for a heroku account at Heroku | Login . Follow the instructions on the site to set up multi factor authentication (MFA). After successfully creating your account and setting up MFA, you'll be taken to the Heroku dashboard.

In the heroku dashboard, click on the "Create New App" button. Select Node.js as the language for your application during the app creation process.

Note: To continue with the deployment process, heroku requires you to add a valid payment method. Follow the prompts to add a payment method. Once the payment method is added, you will be redirected to the app's dashboard, where you can proceed with the deployment.



Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Deployment method

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the Heroku CLI.

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

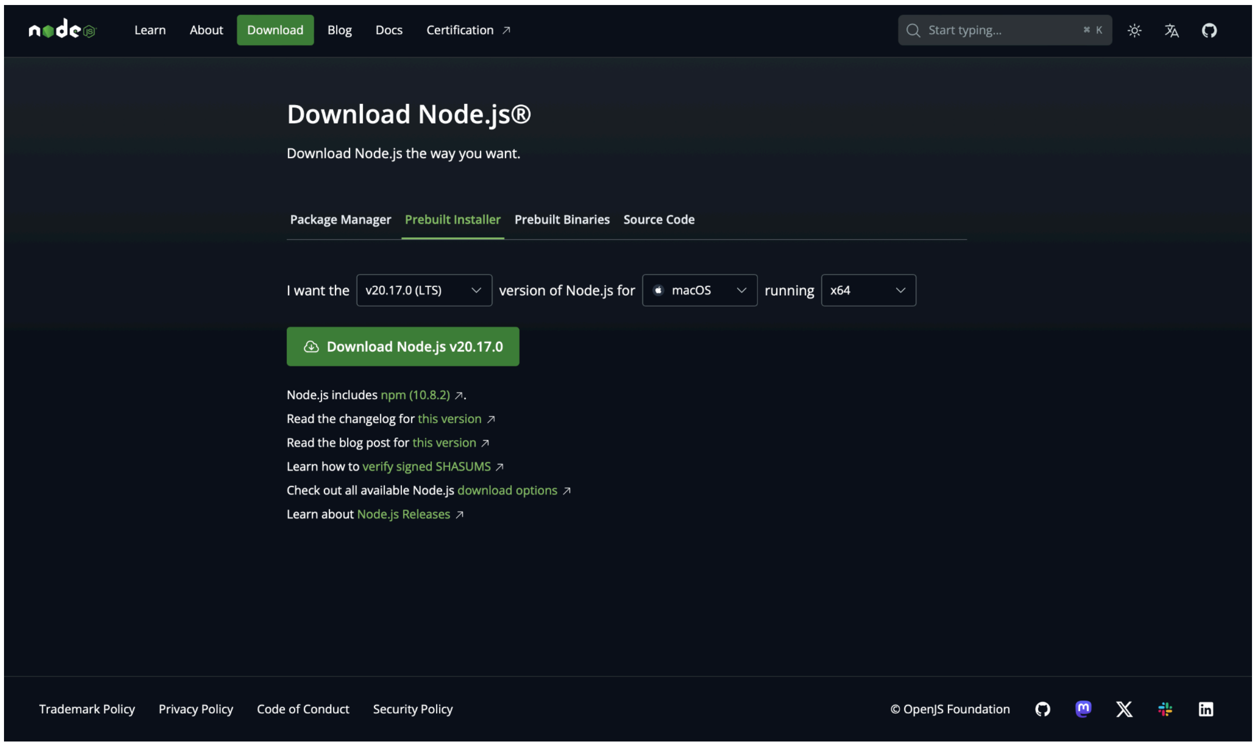
```
$ heroku login
```

Create a new Git repository

## Step-2 (Installing Node.js environment on your device)

Once your Heroku account is set up, the next step is to prepare your Node.js application for deployment.

To begin, download and install the Node.js. visit Node.js — Download Node.js® Ensure you download and install the LTS (Long-Term Support) version from the prebuilt installer. As shown in the figure.



Download Node.js®

Download Node.js the way you want.

I want the v20.17.0 (LTS) version of Node.js for macOS running x64

Node.js includes npm (10.8.2).
Read the changelog for this version
Read the blog post for this version
Learn how to verify signed SHASUMS
Check out all available Node.js download options
Learn about Node.js Releases

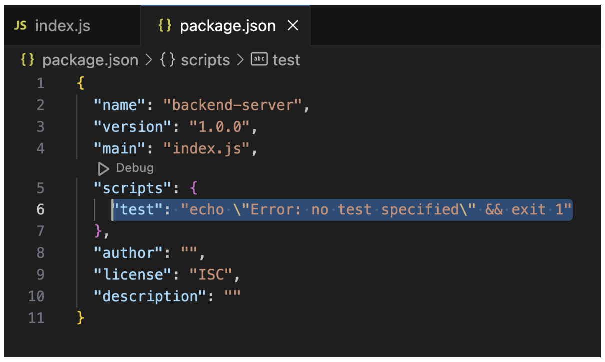
Verify installation: After installation, open your terminal and run the following command to check if Node.js has been installed correctly by typing in **node —version**

When you install node.js, npm (Node Package Manager) is included automatically.

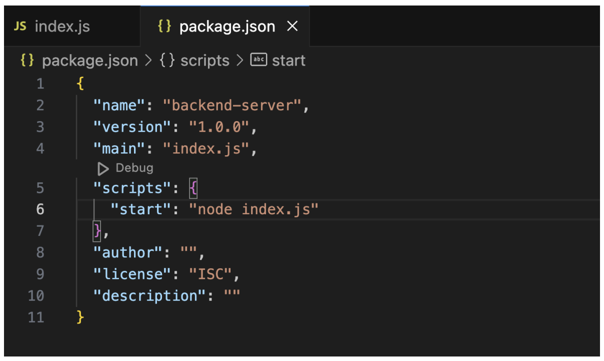
## Step-3 (setting up the node environment)

start by opening Visual Studio Code (VSCode). Use the **Open Folder** option to navigate to the directory where you want to create your project. Once you have the folder open, you can easily access the integrated terminal by going to the top menu, clicking on **Terminal**, and selecting **New Terminal**. In the terminal that appears, type **npm init** to initialize your project. This command will guide you through a series of prompts, asking for details about your project; feel free to fill in the information or simply press Enter to accept the default values. After you complete this setup, a new file called **package.json** will be created in your project directory. This file acts as the manifest for your Node.js application, containing important information like the project name, version, description, and any dependencies you might need.

It's important to modify this file to include the necessary configurations for your application. Update the **package.json** file as shown in the pictures below.



```
{
  "name": "backend-server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": ""
}
```

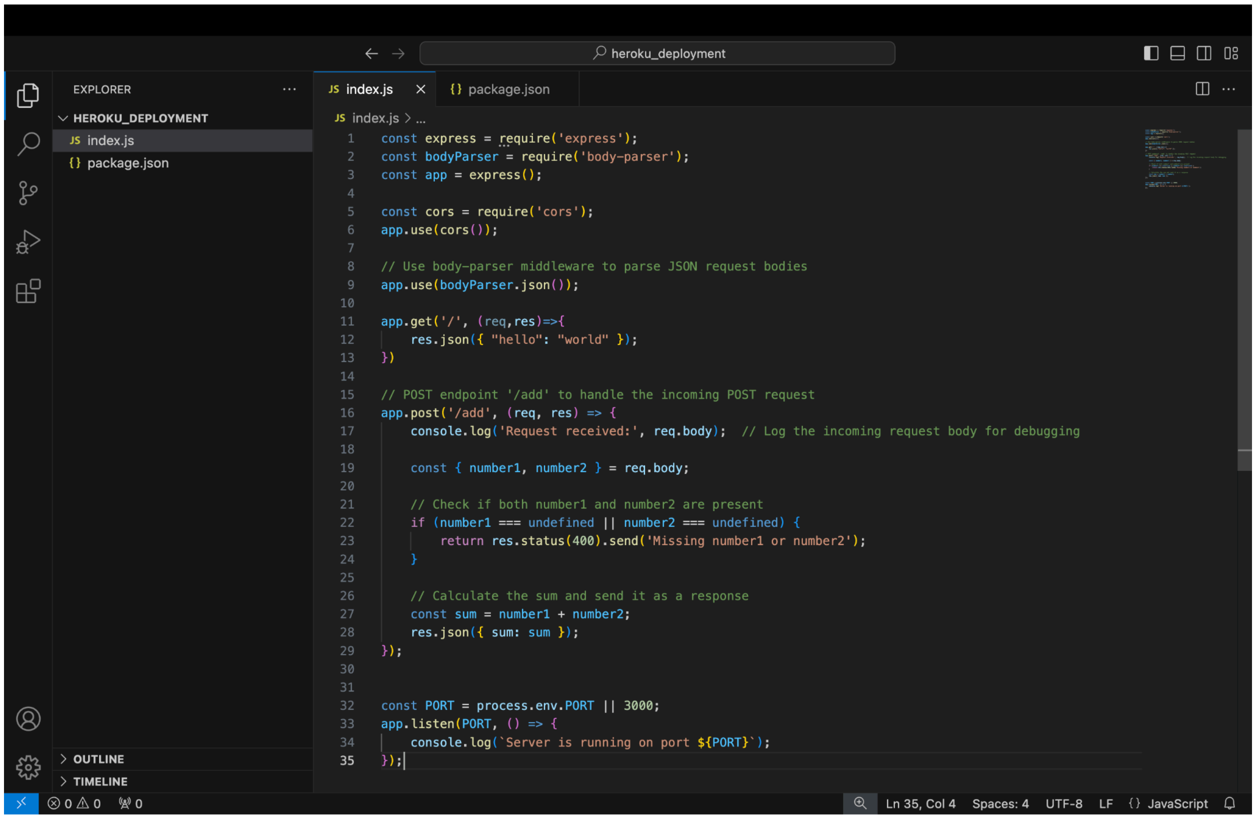


```
{
  "name": "backend-server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC",
  "description": ""
}
```

## Step-4 (server side code implementation)

Next, create a new file in the same directory and name it **index.js**. This file will contain the code for your server. After creating **index.js**, open your **package.json** file and make any necessary edits to ensure it aligns with your project requirements.

Once you have the **package.json** configured, you can proceed to write the server-side code in **index.js**. This code will set up your server and define the necessary routes and middleware to handle incoming requests. Make sure to include all required libraries and frameworks, such as Express, to get your server up and running smoothly.



```javascript
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

const cors = require('cors');
app.use(cors());

// Use body-parser middleware to parse JSON request bodies
app.use(bodyParser.json());

app.get('/', (req,res)=>{
    res.json({ "hello": "world" });
})

// POST endpoint '/add' to handle the incoming POST request
app.post('/add', (req, res) => {
    console.log('Request received:', req.body);  // Log the incoming request body for debugging

    const { number1, number2 } = req.body;

    // Check if both number1 and number2 are present
    if (number1 === undefined || number2 === undefined) {
        return res.status(400).send('Missing number1 or number2');
    }

    // Calculate the sum and send it as a response
    const sum = number1 + number2;
    res.json({ sum: sum });
});


const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

## Step-5 (downloading required packages)

**Express.js** as the server framework, built on top of Node.js. Express.js is designed to simplify the development of web applications and APIs by providing a set of features that enhance the capabilities of Node.js.

**Body-parser** is a middleware in Express.js that is used to parse incoming request bodies in a middleware before your handlers, making the data available under the req.body property. This is especially useful for handling data sent in HTTP requests, such as form submissions or JSON payloads.

**CORS (Cross-Origin Resource Sharing)** is a security feature implemented by web browsers that allows or restricts web applications running at one origin (domain) from making requests to resources hosted on a different origin. In the context of a Node.js application, particularly when using Express.js, the cors() middleware is used to enable CORS, allowing your API to be accessed by clients from different origins.

Run the following commands to install express.js, body-parser and cors

**npm install express**

**npm install body-parser**

**npm install cors**

After installing all the required packages. Test the server by running the **node index.js** command in the terminal. It should say “server is running on port 3000”

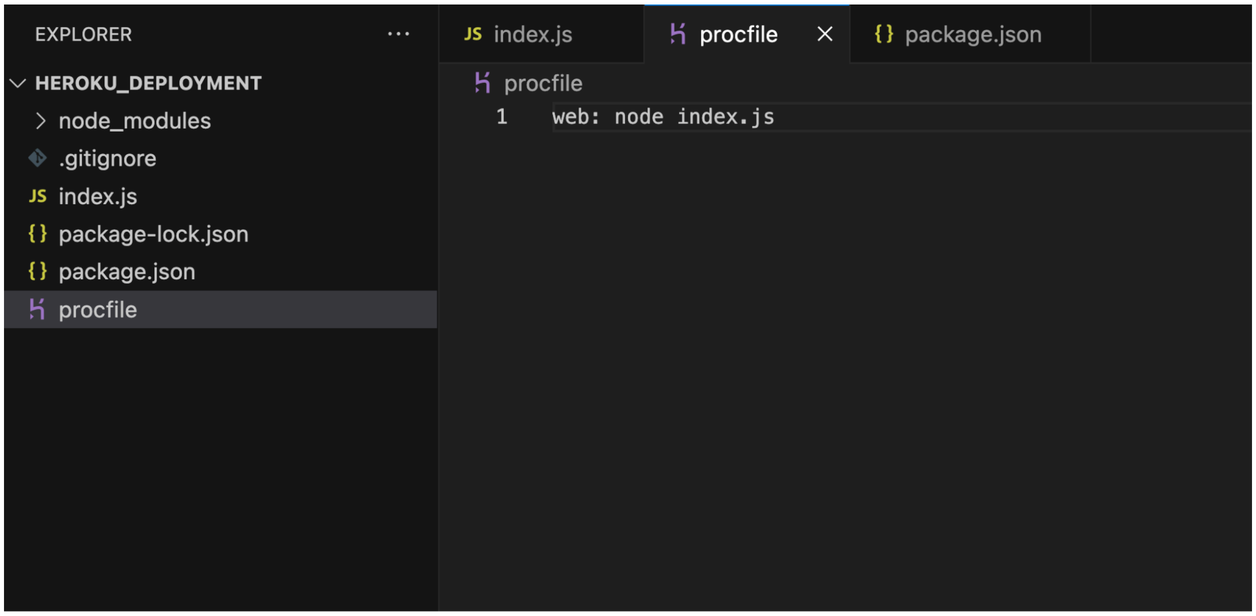
## Step-6 (setting up a Procfile)

A **Procfile** is used in Heroku deployment to specify the commands that start your application, ensuring Heroku knows how to run it. It defines process types, such as web or worker, allowing for clear management and scaling of different application components. Without a Procfile, Heroku may struggle to detect the necessary processes, leading to deployment issues.

Create a procfile and the content as shown in the figure.

The line **web: node index.js** in a Procfile specifies that the **web** process type should run the command **node index.js** to start the Node.js application.

Creating a .gitignore file helps keep your Git repository clean by excluding files and directories that do not need to be tracked, ensuring that only relevant project files are included in version control.



```
web: node index.js
```

## Step-7 (Pushing code into GitHub)

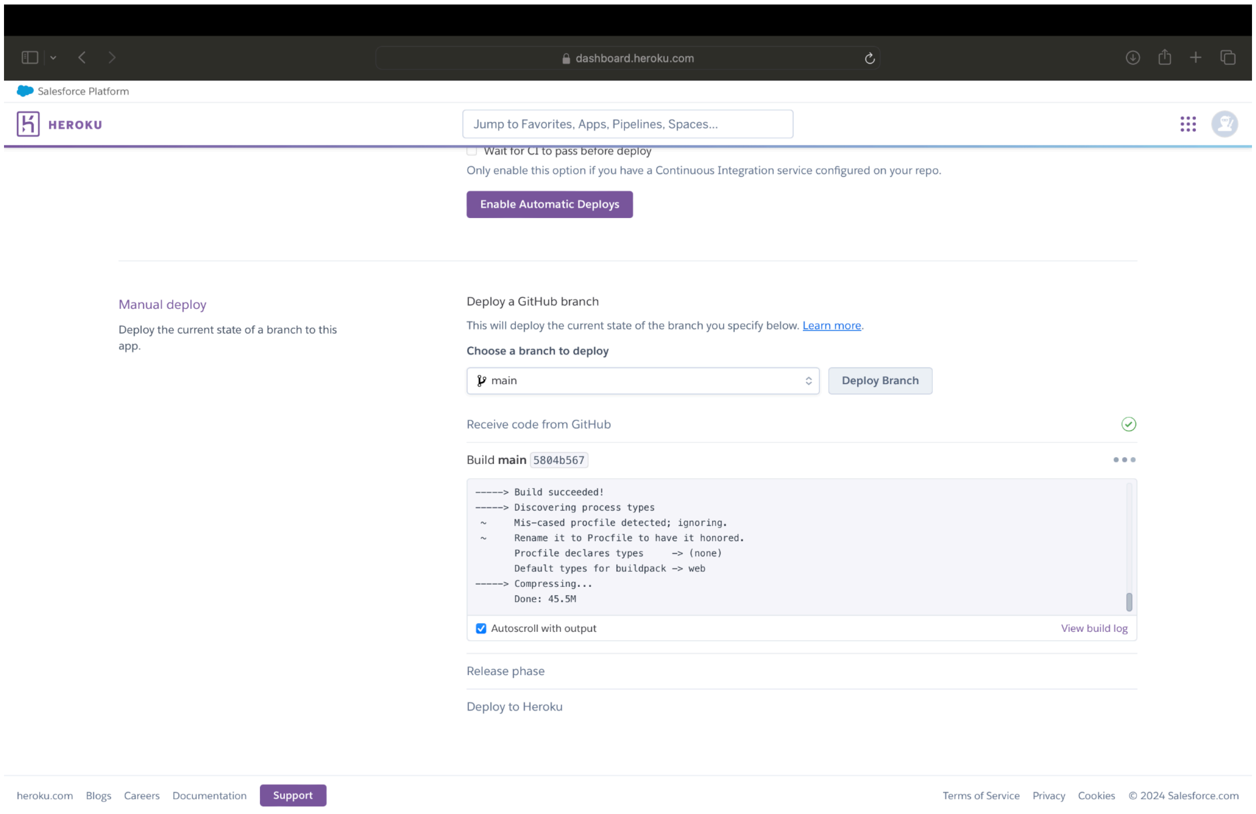
This guide provides essential steps for creating a GitHub repository, setting up Git locally, linking your local repository to GitHub, and pushing your code to the repository. By following these steps, you will effectively manage your project's version control and collaborate seamlessly with others.

For a detailed walkthrough, refer to the article "How to Push a Project to GitHub" by Sameer Katija, which covers everything you need to know about using Git and GitHub for your projects.

## Step-8 (Deployment on heroku)

To deploy your application, open the Heroku dashboard and navigate to the app you created earlier. From there, select **GitHub** as your deployment method. Click the **Connect to GitHub** button and authorize Heroku to access your GitHub account. Once authorized, you'll be able to select the name of the GitHub repository you created or wish to host, allowing you to connect it to your Heroku app.

After successfully connecting your GitHub repository, scroll to the bottom of the page and click the **Deploy Branch Button**. This action will initiate the deployment process for your application. Once the deployment is complete, your app will be live and accessible!



Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub ✓

Build **main** 5804b567

```
-----> Build succeeded!
-----> Discovering process types
 ~     Mis-cased procfile detected; ignoring.
 ~     Rename it to Procfile to have it honored.
       Procfile declares types     -> (none)
       Default types for buildpack -> web
-----> Compressing...
       Done: 45.5M
```

☑ Autoscroll with output — View build log

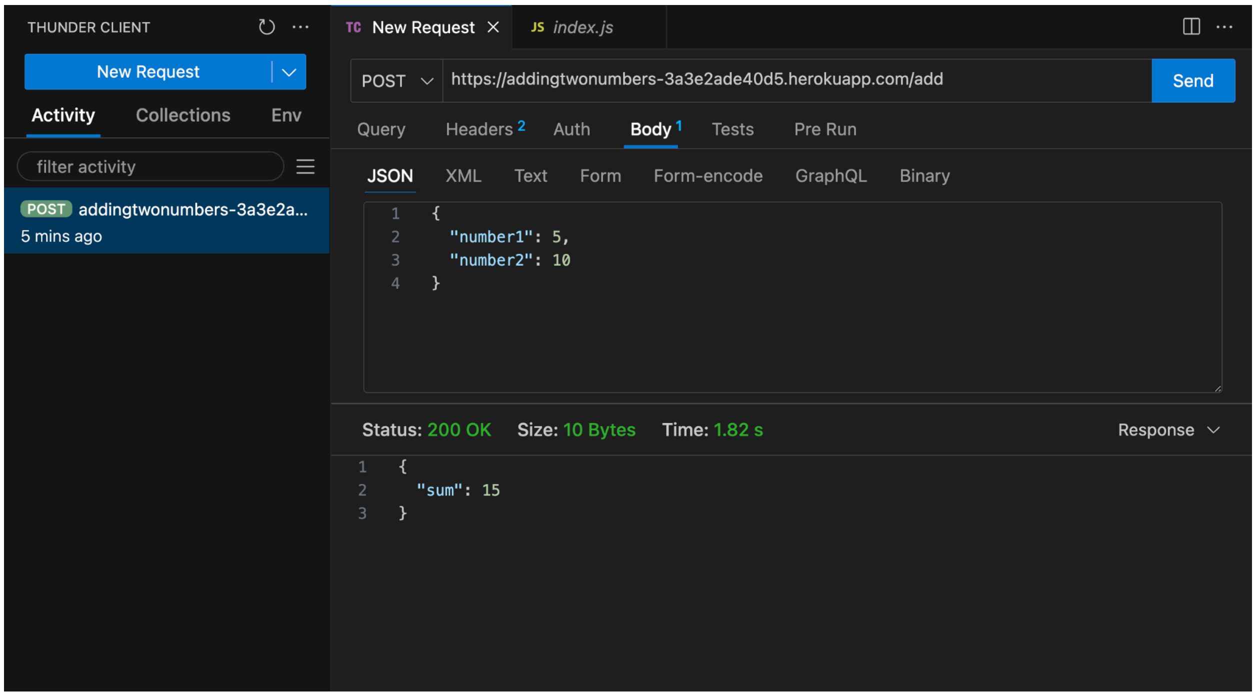
Release phase

Deploy to Heroku

## Step-9 (verification of the hosted server)

Once your application is deployed, you can view it by clicking the **View** button on the Heroku dashboard. However, you might see an error message like "Cannot GET /" in your browser, which indicates that we haven't defined a route for the root URL ("/").

To verify that your app is running correctly, you can use a tool like Postman to send a POST request to your server. Alternatively, you can utilize the **Thunder Client** extension in Visual Studio Code to perform the same verification. This extension allows you to easily create and send requests directly from your editor, helping you confirm that your application is functioning as intended.



```
POST https://addingtwonumbers-3a3e2ade40d5.herokuapp.com/add
```

```
{
  "number1": 5,
  "number2": 10
}
```

Status: 200 OK Size: 10 Bytes Time: 1.82 s

```
{
  "sum": 15
}
```