

Topic: Plant Analysis Tool using Gemini AI and Express.js Part 1

Speaker: Masynctech / Notebook: Node.js (JavaScript) Projects



We used NODE.JS (Javascript) and NVM.


[MAIN VIDEO RESOURCE:](#)

1. We created a new folder, PLANTANALYSIS TOOL.
2. We open a Gitbash Terminal here and use `CODE .` (code dot) to open our VS CODE editor.
3. We [Install NODE.JS](#), CODEIUM EXTENSION IN VSCODE EXTENSIONS and get Google API key from Google API dashboard.



Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

[Download Node.js \(LTS\)](#) 

Downloads Node.js **v20.17.0**¹ with long-term support.
Node.js can also be installed via package managers.

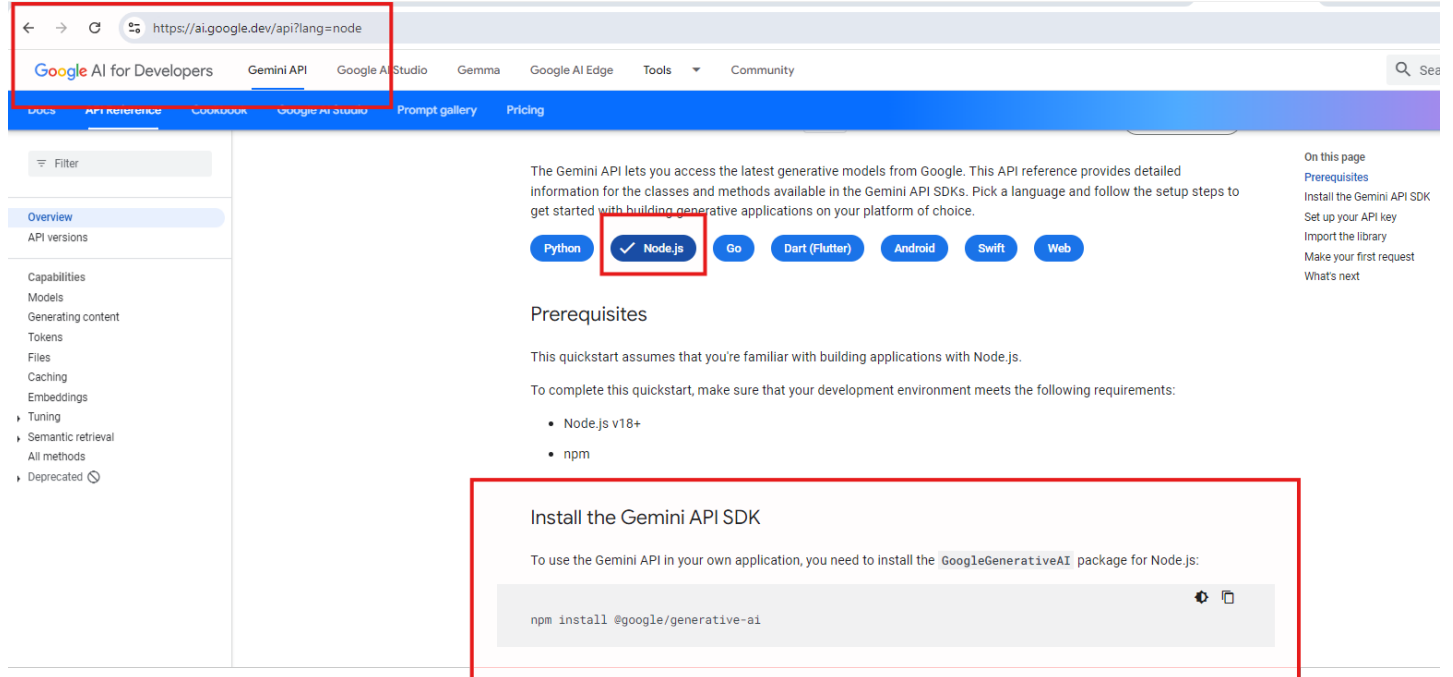
Want new features sooner? Get **Node.js v22.8.0**¹ instead.

The image shows the Visual Studio Code extension marketplace page for the Codeium extension. The extension is titled "Codeium: AI Coding Autocomplete and Chat for Python, Javascript, Typescript, Java, Go, and more". It has a rating of 5 stars (1352 reviews) and 1,532,738 installations. The description states it is a free AI code acceleration plugin. The page includes sections for "What is Codeium?", "With Codeium, you get:" (listing features like unlimited completions and IDE-integrated chat), and "Categories" (AI, Chat, Programming Languages, Machine Learning, Snippets, Education). A sidebar on the left shows other installed and available extensions like Codeium Enterprise, Python Silver Pack, and VSCode Essentials.

The image shows the "Get API key" page in Google AI Studio. The page provides instructions on how to use API keys securely and includes a cURL command to test the API. A "Create API key" button is highlighted with a red box. Below the instructions, a table lists the user's API keys. The table has columns for Project number, Project name, API key, Created, and Plan. One key is listed for the "Generative Language Client" project, with a partial API key "...f8IM" and a creation date of "Sep 14, 2024". The plan is "Free of charge".

Project number	Project name	API key	Created	Plan
...9052	Generative Language Client	...f8IM	Sep 14, 2024	Free of charge Set up Billing View usage data

We use GOOGLE API REFERENCE to install our GENERATIVE AI in NODE.JS:



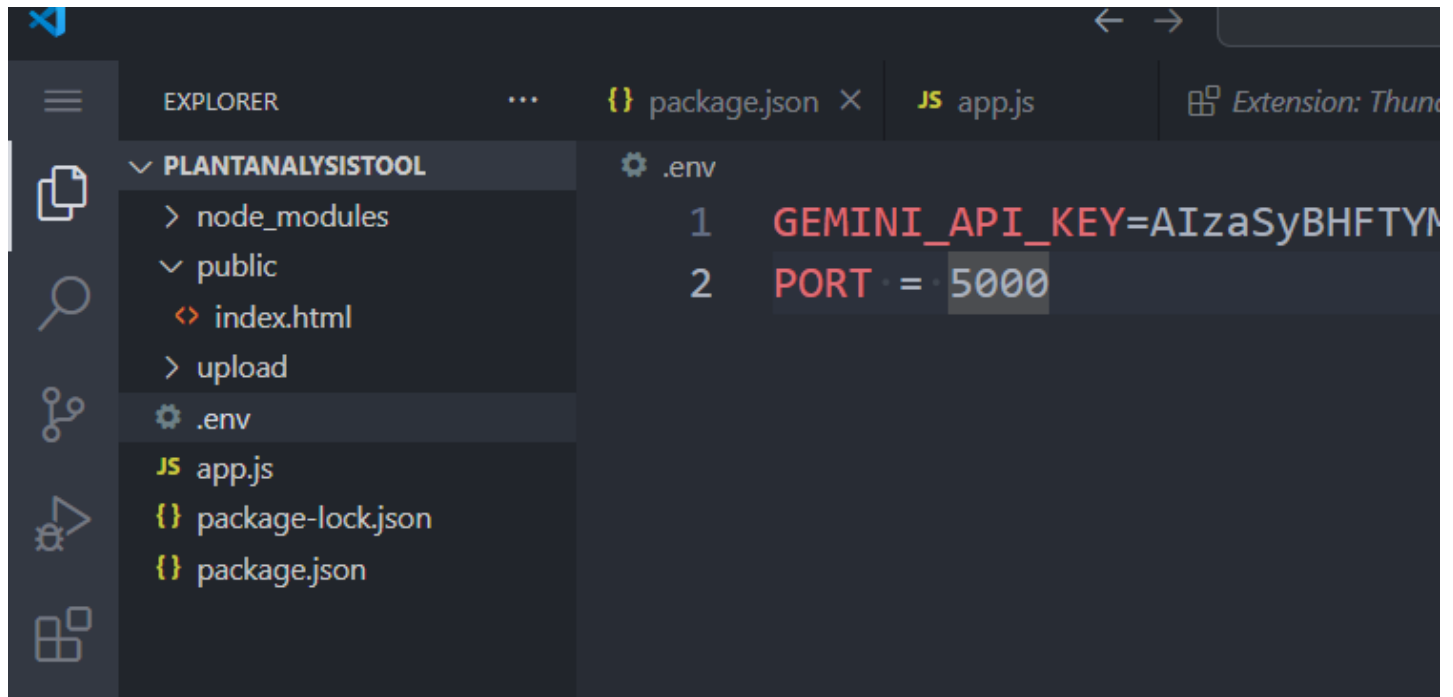
The screenshot shows the Google AI Reference website for Node.js. The URL in the browser is `https://ai.google.dev/api?lang=node`. The page title is "Google AI for Developers Gemini API". The main content area is titled "Install the Gemini API SDK" and includes a code block for installing the package:

```
npm install @google/generative-ai
```

The "Node.js" button in the language selection bar is highlighted with a red box. The "Install the Gemini API SDK" section is also highlighted with a red box.

4. We created new folders like UPLOAD, PUBLIC and created APPS.JS and .ENV files

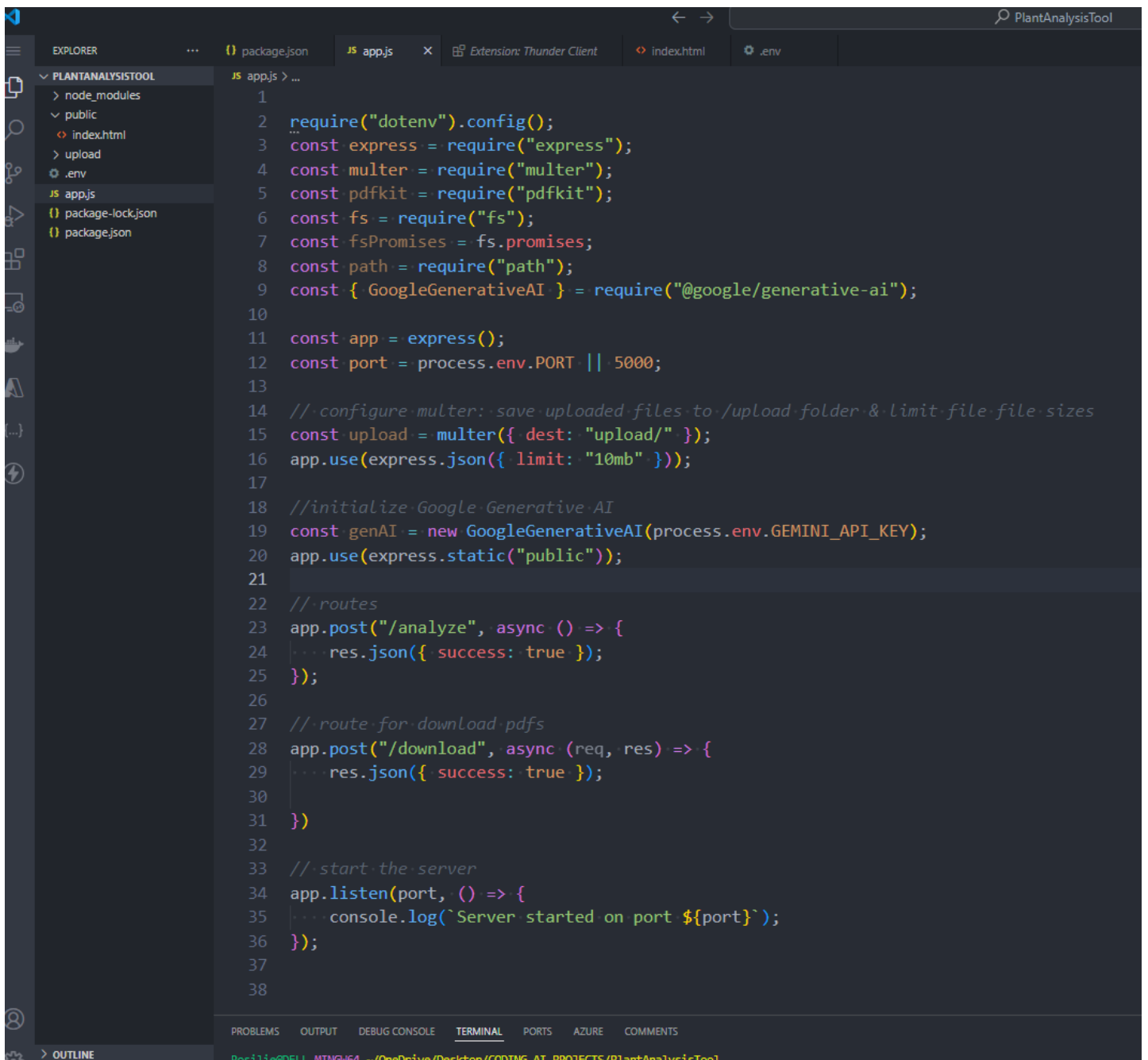
.ENV FILE:



The screenshot shows the Visual Studio Code interface. The Explorer view on the left shows the project structure for "PLANTANALYSISTOOL" with folders "node_modules", "public", and "upload", and files "index.html", ".env", "app.js", "package-lock.json", and "package.json". The ".env" file is open in the editor, showing the following content:

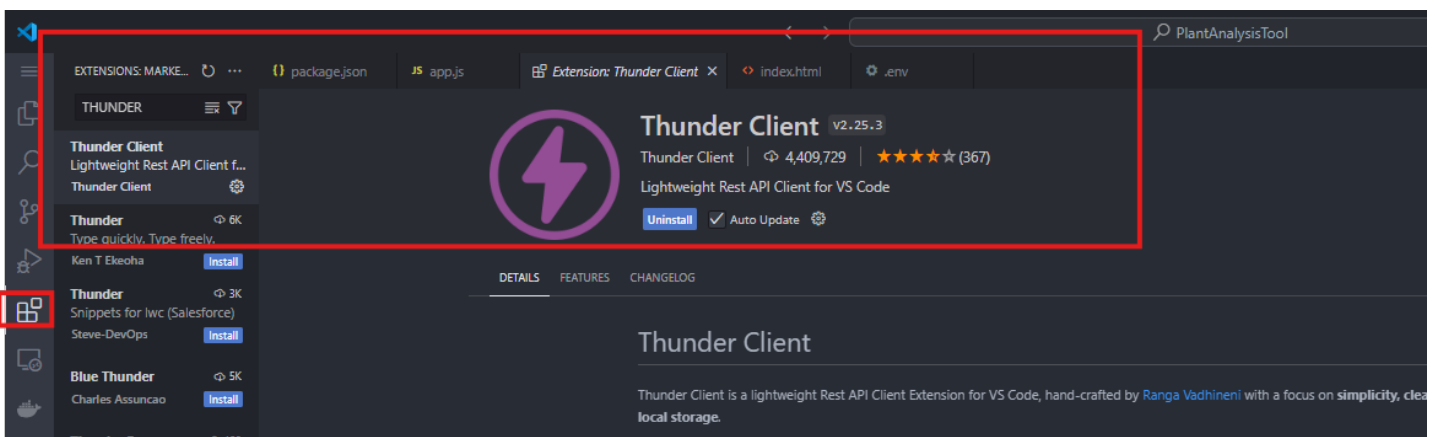
```
1 GEMINI_API_KEY=AIzaSyBHFTYM  
2 PORT = 5000
```

APPS.JS



```
1 require("dotenv").config();
2 const express = require("express");
3 const multer = require("multer");
4 const pdfkit = require("pdfkit");
5 const fs = require("fs");
6 const fsPromises = fs.promises;
7 const path = require("path");
8 const { GoogleGenerativeAI } = require("@google/generative-ai");
9
10
11 const app = express();
12 const port = process.env.PORT || 5000;
13
14 //configure multer: save uploaded files to /upload folder & limit file file sizes
15 const upload = multer({ dest: "upload/" });
16 app.use(express.json({ limit: "10mb" }));
17
18 //initialize Google Generative AI
19 const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
20 app.use(express.static("public"));
21
22 //routes
23 app.post("/analyze", async () => {
24   ... res.json({ success: true });
25 });
26
27 //route for download pdfs
28 app.post("/download", async (req, res) => {
29   ... res.json({ success: true });
30 });
31
32
33 //start the server
34 app.listen(port, () => {
35   ... console.log(`Server started on port ${port}`);
36 });
37
38
```

5. To test our ENDPOINT, we will use POSTMAN (you used INSOMNIA) or we can install the VS CODE EXTENSION, THUNDER CLIENT.

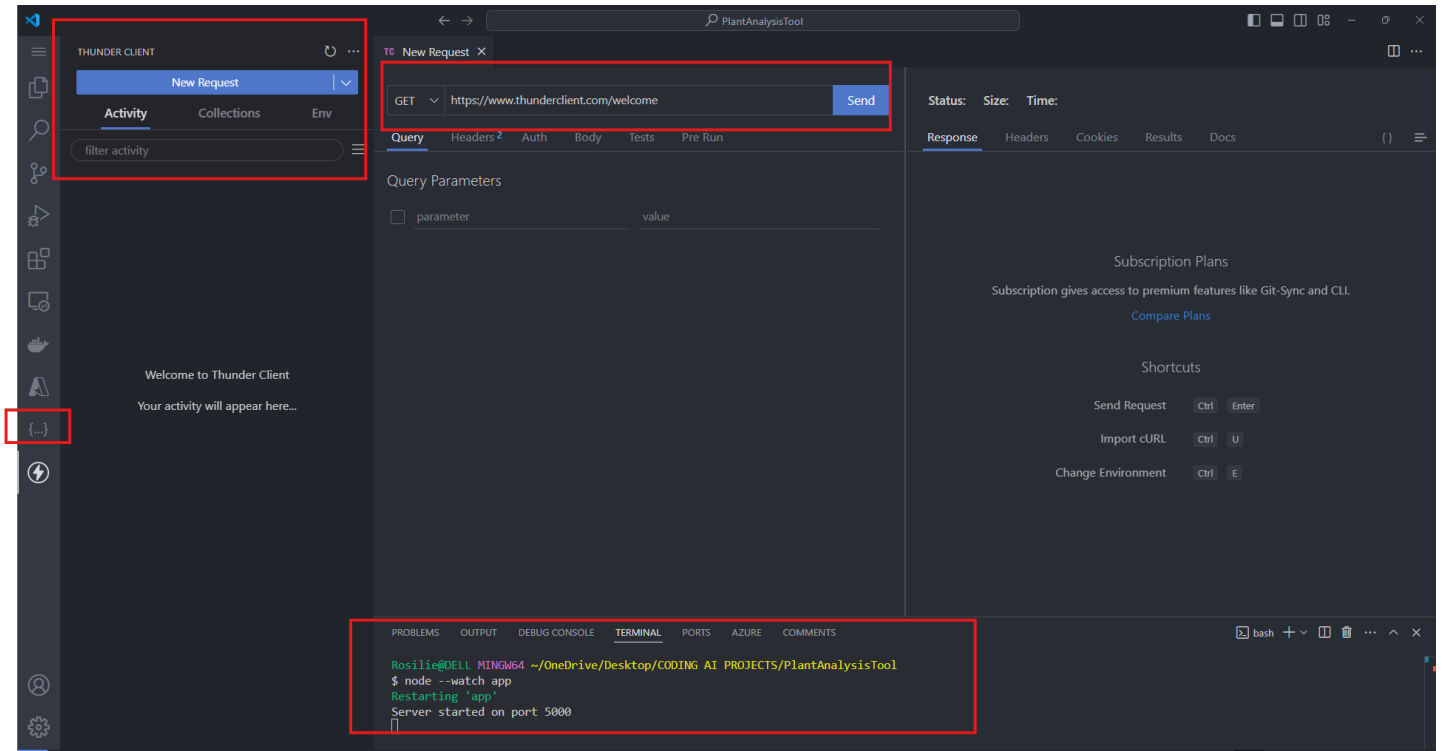


6. Run the app by issuing this code.

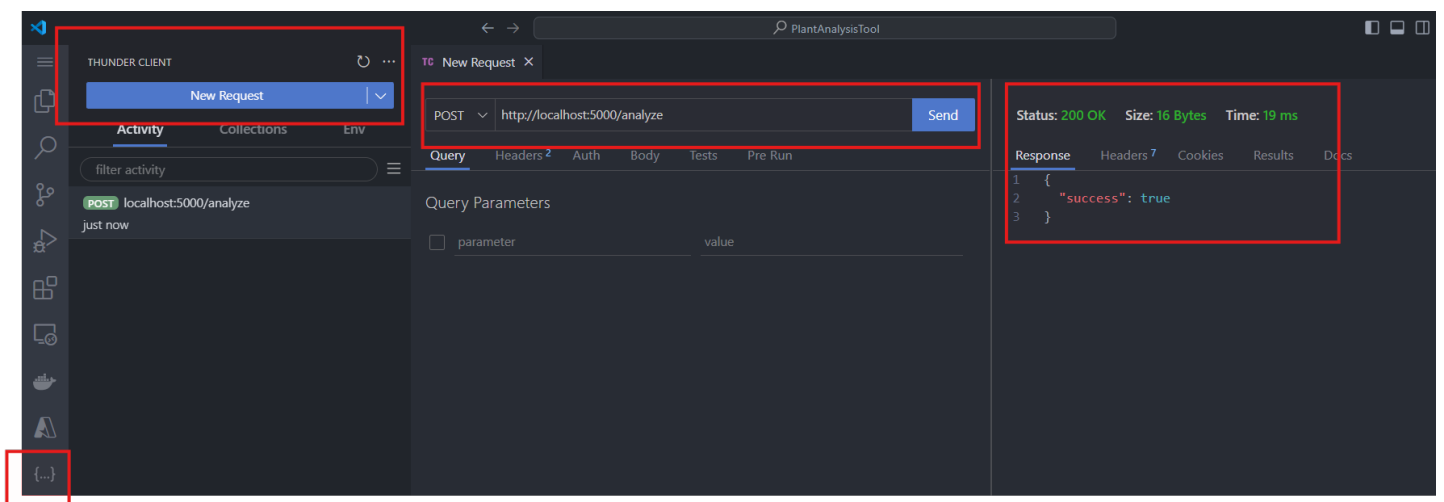
```
$ node --watch app (where app is our APPS.JS)
```

```
PROBLEMS OUTPUT TERMINAL
Rosilie@DELL MINGW64 ~/OneDrive/Desk
ysisTool
$ node --watch app
Restarting 'app'
Server started on port 5000
█
```

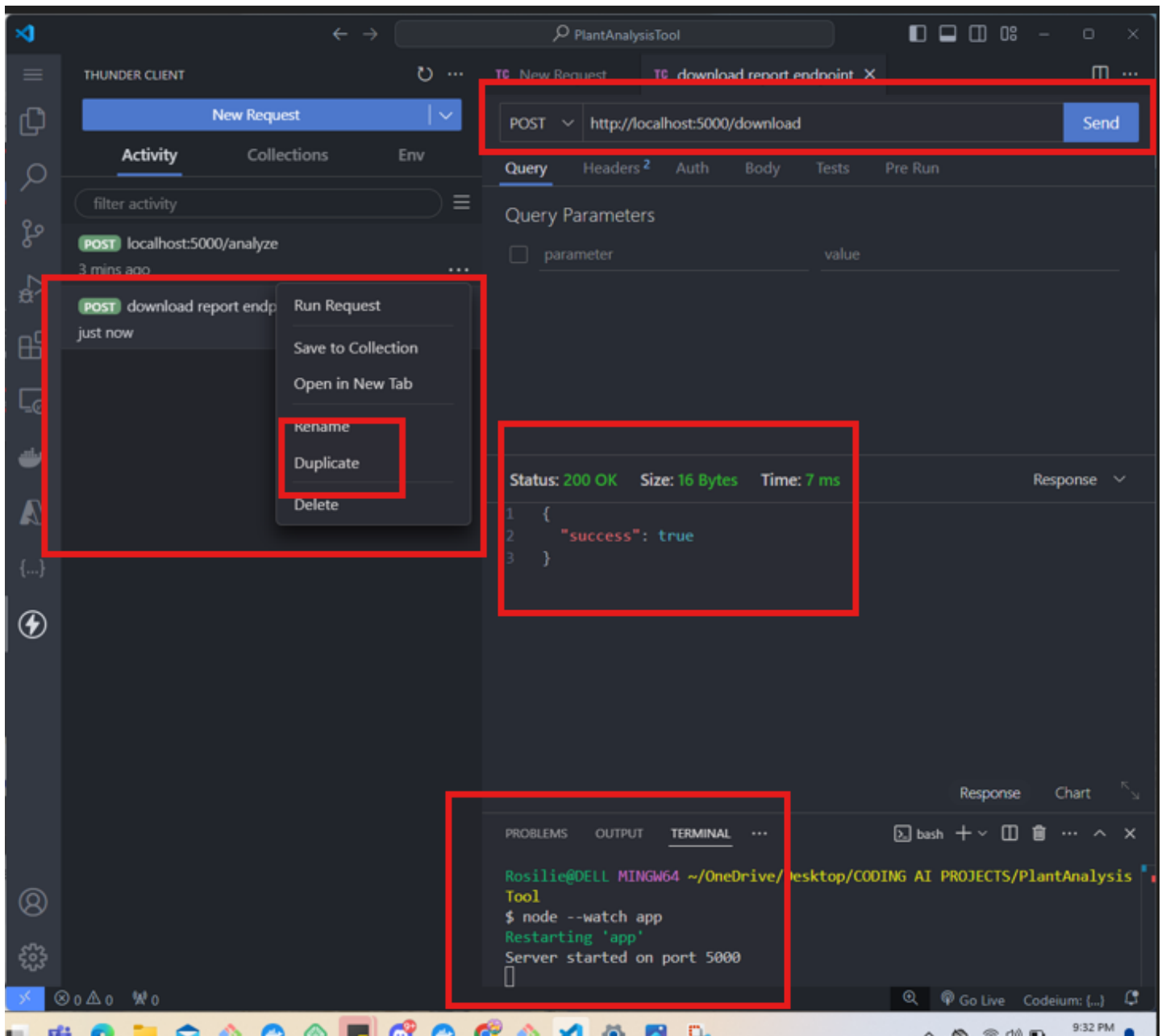
7. Close all your tabs in VS Code. Right click on the THREE DOTS where the EXTENSION button is, and select THUNDER POINT. Select NEW REQUEST.



8. To access our work, we issue our URL path: HTTP://localhost: 5000. This should show a SUCCESS MESSAGE



9. We test our other endpoint, HTTP://LOCALHOST:DOWNLOAD/ We duplicate our first request and name it. Then, we change our URL PATH.



10. Just like in Django where we test our paths using Django's views.py, the logic for Node.js is this:

```

// routes
// analyze route
app.post("/analyze", async (req, res) => {
  ... res.json({ success: true });
});

// route for download pdfs
app.post("/download", async (req, res) => {
  ... res.json({ success: true });
});
}

```

11. To test the upload function, we can use the THUNDER BODYFORM and add the variable we used 'IMAGE' and upload a file from our local device. We should be able to see the details of this image.

APPS.JS:

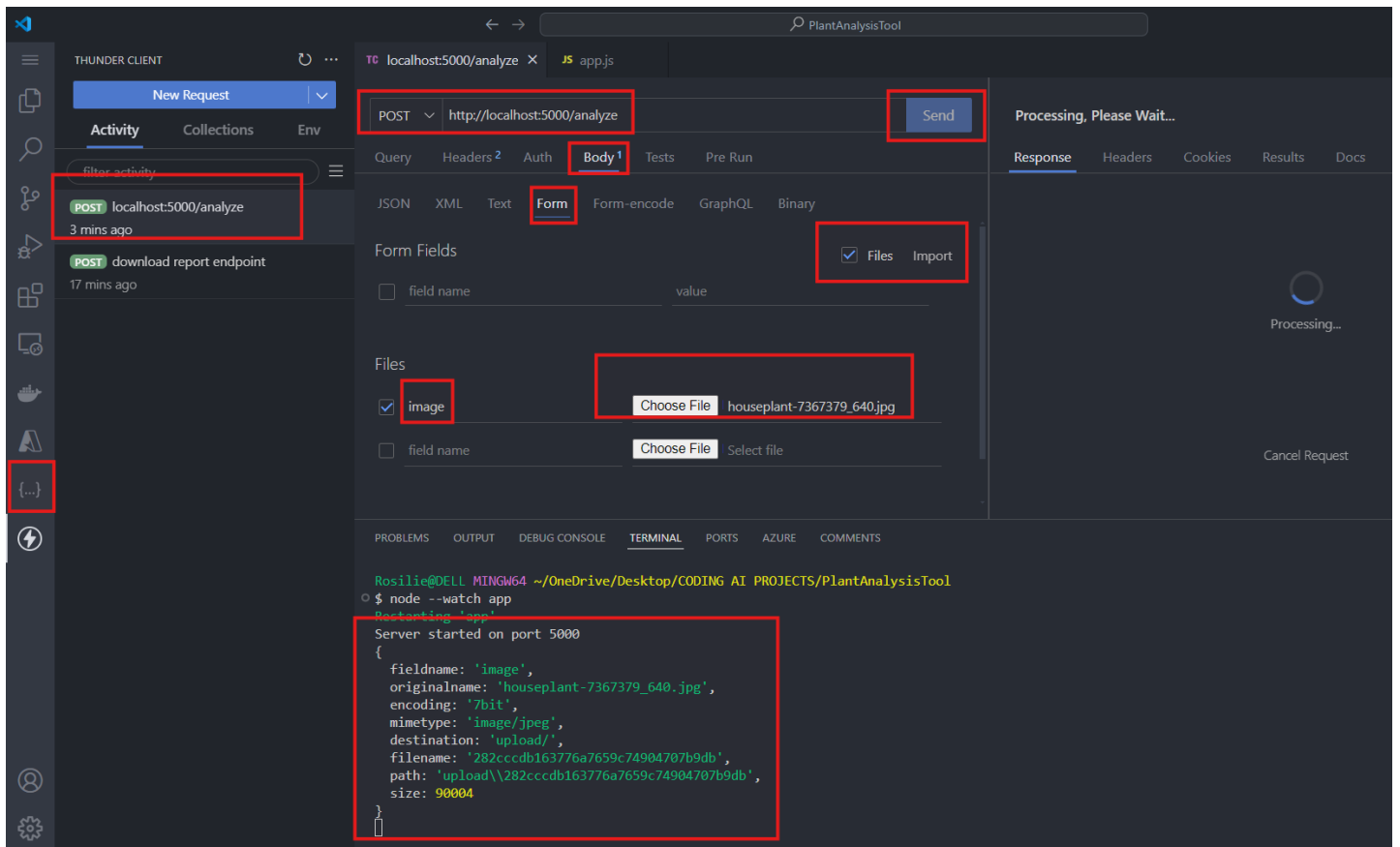
The screenshot shows a code editor with the following code in app.js:

```

13
14 // configure multer: save uploaded files to /upload folder & limit file sizes
15 const upload = multer({ dest: "upload/" });
16 app.use(express.json({ limit: "10mb" }));
17
18 // initialize Google Generative AI
19 const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
20 app.use(express.static("public"));
21
22 // routes
23 // analyze route and uses multer upload variable to save uploaded files as images
24 app.post("/analyze", upload.single("image"), async (req, res) => {
25   ... const file = req.file;
26   ... console.log(file);
27 });
28

```

Two red boxes highlight the multer configuration (lines 14-16) and the analyze route handler (lines 24-27).



12. To allow Gemini AI to use the details captured from step 11, we have to indicate the GEMINI VERSION:

Make your first request

Use the `generateContent` method to generate text.

```

// Make sure to include these imports:
// import { GoogleGenerativeAI } from "@google/generative-ai";
const genAI = new GoogleGenerativeAI(process.env.API_KEY);
const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });

const prompt = "Write a story about a magic backpack.";

const result = await model.generateContent(prompt);
console.log(result.response.text());

```

text_generation.js

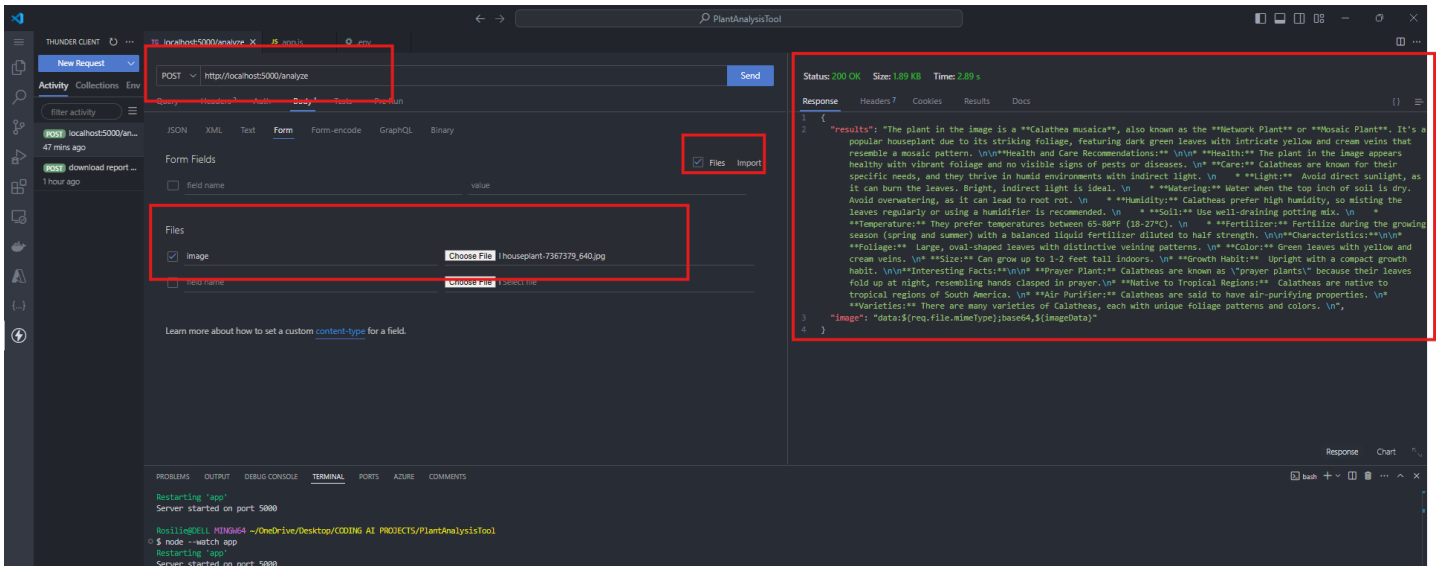
13. We updated our APPS.JS to include GEMINI API.

This is the PROMPT we used for Gemini "Analyze this plant image and provide detailed analysis of its species, health and care recommendations, its characteristics, care instructions and interesting facts. Please provide the response in plain text without using any markdown formatting "

Our function:

```
22 // routes
23 // analyze route and uses multer upload variable to save uploaded files as images
24 app.post("/analyze", upload.single("image"), async (req, res) => {
25   const file = req.file;
26   //console.log(file); use the image details for Gemini AI
27   if (!file) {
28     return res.status(400).json({ error: "Please upload an image" });
29   }
30   const imagePath = req.file.path;
31   const imageData = await fsPromises.readFile(imagePath, {
32     encoding: "base64",
33   });
34   // use the gemini AI API to analyze the image
35   const model = genAI.getGenerativeModel({
36     model: "gemini-1.5-flash",
37   });
38   const results = await model.generateContent([
39     "Analyze this plant image and provide detailed analysis of its species, health and care recommendations, its characteristics",
40     {
41       inlineData: {
42         mimeType: req.file.mimetype,
43         data: imageData,
44       },
45     },
46   ]);
47   const plantInfo = results.response.text();
48   // remove the uploaded image
49   await fsPromises.unlink(imagePath);
50   // send the response
51   res.json({ results: plantInfo, image: `data:${req.file.mimetype};base64,${imageData}` });
52 } catch (error) {
53   res.status(500).json({ error: error.message });
54 }
55 });
```

14. We run our endpoint using Thunder Client:



15.