Episode-03 | Creating our Express Server

I'm writing everything from scratch, so **just keep your focus and follow along🫡**.

First Step

I will create a folder named `devTinder.`

We'll keep a **separate folder for the frontend**.

Now, open this folder in **VS Code**.

How Do You Start Building the Project?

The first thing you need to do is **initialize the project.**

Open your terminal, and inside your folder, run the command:

```
npm init
```

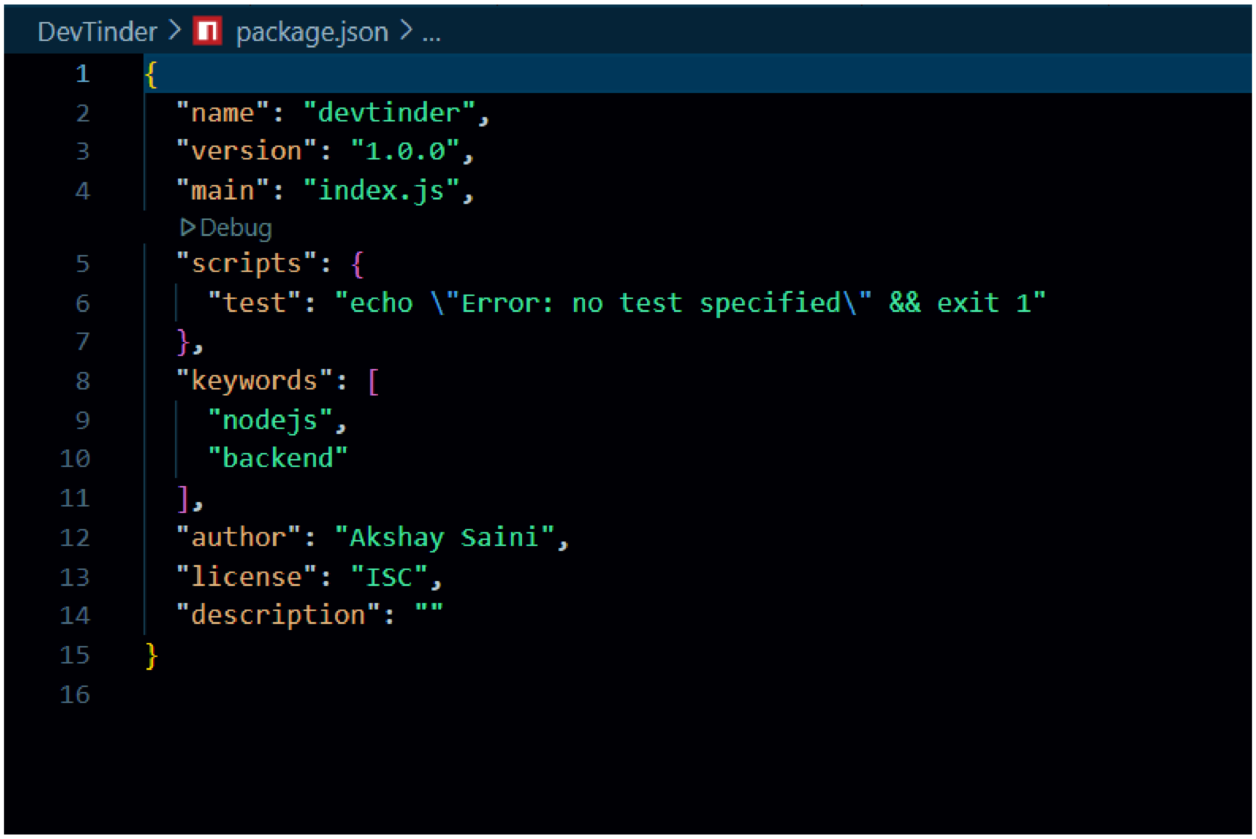
It will ask you a few questions:

1. **Package name:** Choose a name for your package.
2. **Description:** You can write anything you want.

3. **Entry point:** Don't touch it (leave as default).
4. **Test command:** We won't worry about this now.
5. **Keywords:** You can add keywords like `nodejs`, `backend`.
6. **Author:** Write your name (e.g., Akshay Saini).

Press **Enter**.

This will create a `package.json` file `{ a configuration file for your project }.`



DevTinder > package.json > ...

```json
{
  "name": "devtinder",
  "version": "1.0.0",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "backend"
  ],
  "author": "Akshay Saini",
  "license": "ISC",
  "description": ""
}
```

What Is the Use of `package.json`?

This file is like the **index of your project**, similar to an index in your notebook.

It contains all the **metadata and important information** about your project.

Now, Let's Write Some Code

To start coding, you need to create a **JavaScript file**.

1. First, create a folder called `src`.
2. Inside it, create a file named `app.js`.

This `app.js` file will be the **starting point of your application**, where we'll initialize and write our code.

Let's write a simple line:

```
console.log("Namaste Node.js");
```

To run this file, open the terminal and type:

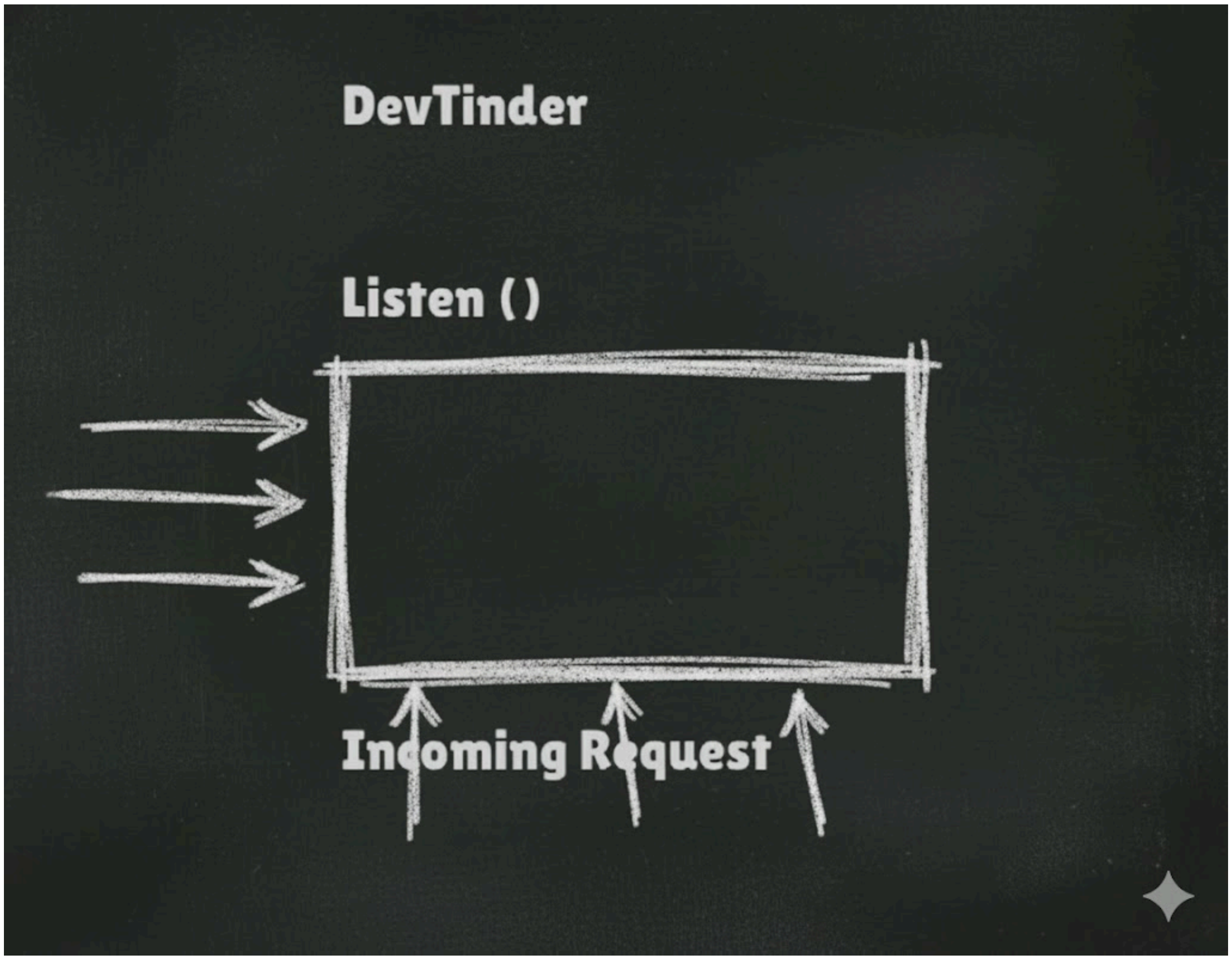
```
node src/app.js
```

You'll see the output `(Namaste Node.js)` printed in the terminal.

`This is something we already learned in` **`Season 1`** `.`

`for creating our devtinder we need server`

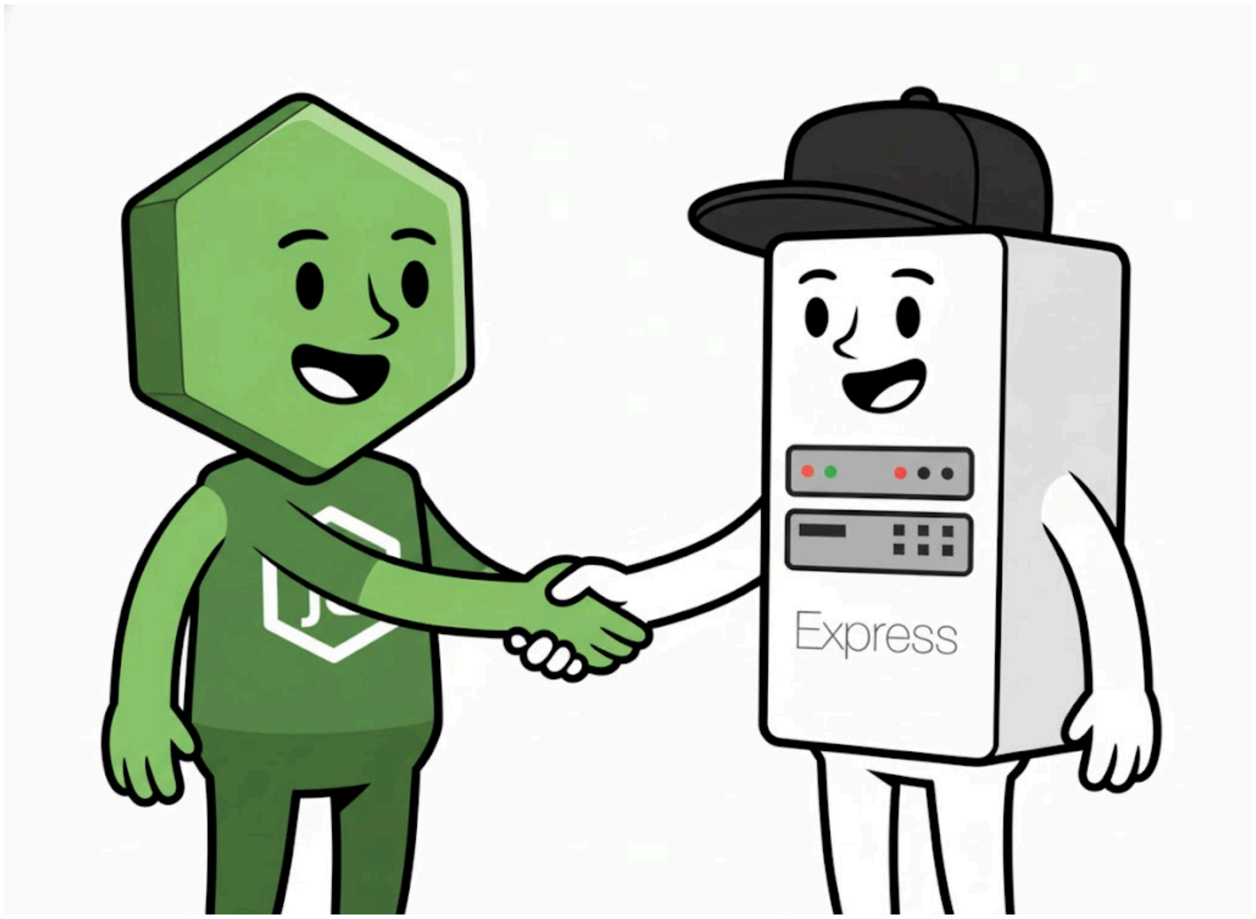
How Do You Create a Server?



To create a server, we’ll be using **Express**, which makes our work much easier compared to building a server in pure Node.js.

Using Express, we can set up a server in just a few lines of code.

What Is Express.js?



Express.js, often called **Express**, is a popular, minimalist, and flexible web application framework for Node.js.

It provides a robust set of features for building web applications and APIs **quickly and efficiently**.

Key Characteristics and Features of Express.js

- **Server-Side Development:** Simplifies building server-side applications with an easy-to-use API for routing, middleware, and handling HTTP requests and responses.
- **Routing:** Provides a powerful routing system, allowing you to define different routes (paths) and associate them with functions to handle requests.
- **Middleware:** Uses a middleware-based architecture, so you can integrate functions that process requests and responses at different stages (e.g., logging, authentication, input validation, error handling).

- **HTTP Utilities:** Offers methods for managing request and response objects, like sending strings or JSON, setting status codes, and handling file downloads.
- **Template Engines:** Works with template engines like EJS, Pug, or Handlebars for server-side rendering of HTML pages.
- **Building APIs:** Widely used to create RESTful APIs that handle CRUD (Create, Read, Update, Delete) operations for front-end apps or mobile apps.
- **Scalability and Flexibility:** Lightweight and modular design, suitable for scalable solutions including microservices.
- **Open-Source:** Free and open-source under the MIT License, with a large and active community.

In short, Express.js provides the structure and tools Node.js developers need to build the backend of web applications, manage server-side logic, and handle client interactions efficiently.

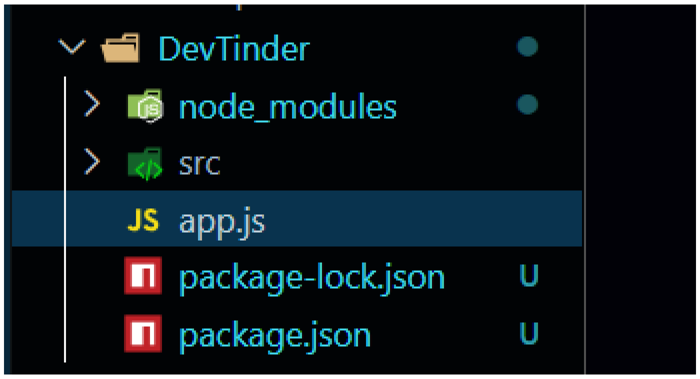
Now, Let's Create Our First Server

Before we create the server, we need to **install Express.js** in our project.

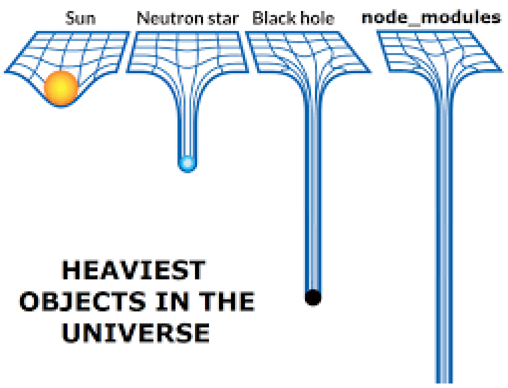
It's available on **npm**, so just run the command:

```
npm i express
```

Once it's installed, you'll see a `node_modules` folder and a `package-lock.json` file created in your project.



What Is `node_modules` and `package-lock.json` ?



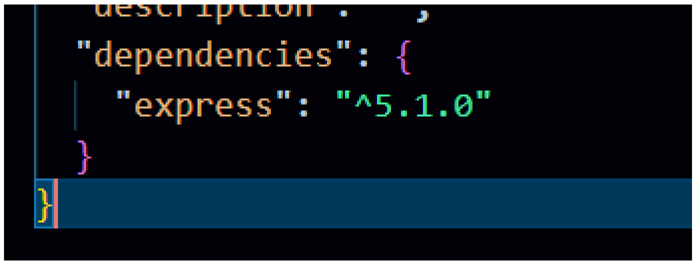
- The `node_modules` folder contains **all the code of the packages** you've installed → in this case, Express.js.
- The `package-lock.json` file keeps track of the **exact versions of the packages** installed, ensuring consistency across different environments.

If you open `package.json`, you'll see a section called **dependencies.**

What are dependencies?

Dependencies are **packages your project relies on to run**.

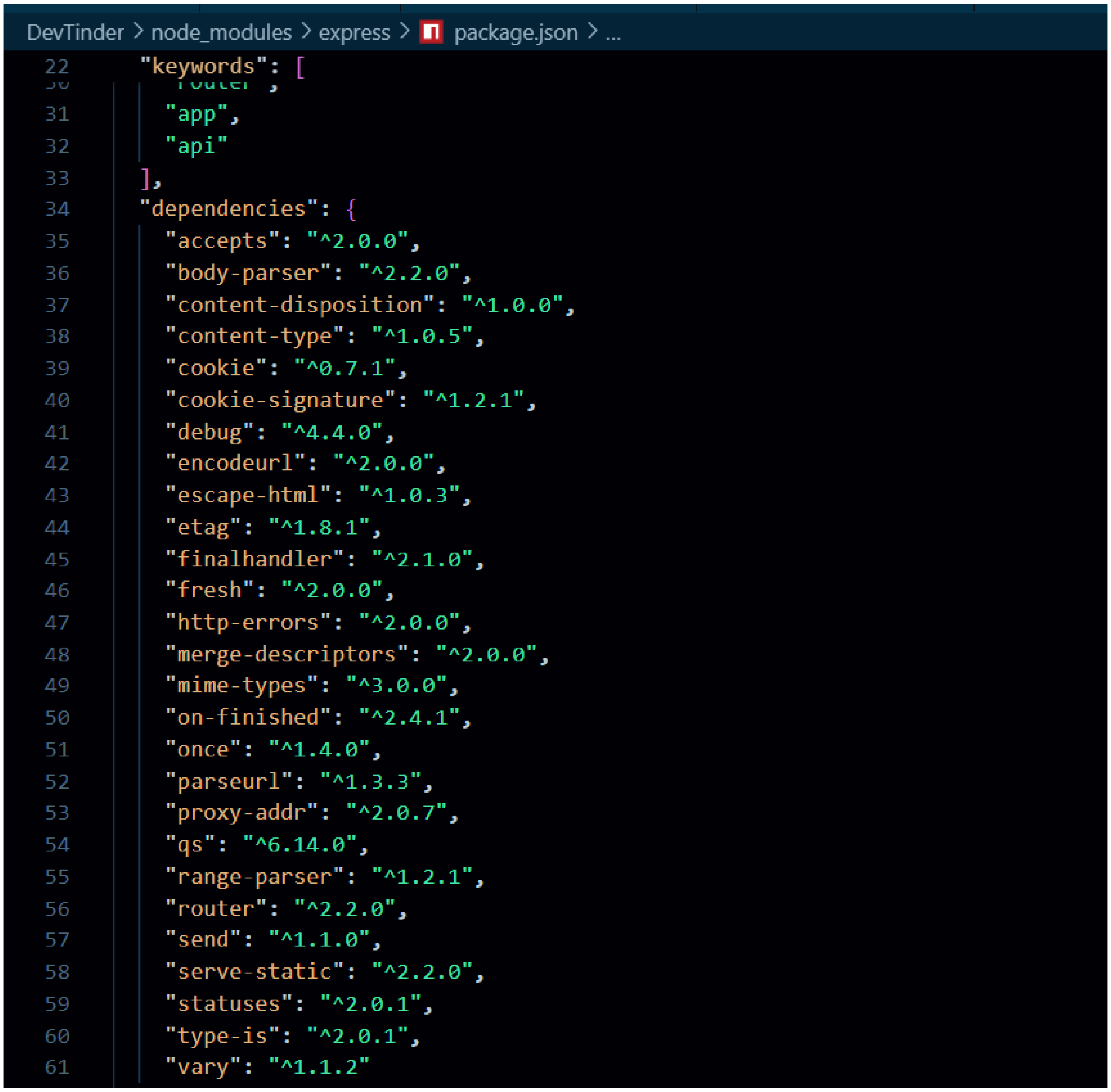
> Your project **won't work without them**, and all these dependencies are installed in the `node_modules` folder.



```
  "dependencies": {
    "express": "^5.1.0"
  }
}
```

If you go to the `node_modules/express` folder and open its `package.json`, you'll see the **dependencies that Express itself depends on**.

That's the beauty of **open-source software** `{ Express is open-source, so you can see and use all the code freely }.`



DevTinder > node_modules > express > package.json > ...

```json
    "keywords": [
      "router",
      "app",
      "api"
    ],
    "dependencies": {
      "accepts": "^2.0.0",
      "body-parser": "^2.2.0",
      "content-disposition": "^1.0.0",
      "content-type": "^1.0.5",
      "cookie": "^0.7.1",
      "cookie-signature": "^1.2.1",
      "debug": "^4.4.0",
      "encodeurl": "^2.0.0",
      "escape-html": "^1.0.3",
      "etag": "^1.8.1",
      "finalhandler": "^2.1.0",
      "fresh": "^2.0.0",
      "http-errors": "^2.0.0",
      "merge-descriptors": "^2.0.0",
      "mime-types": "^3.0.0",
      "on-finished": "^2.4.1",
      "once": "^1.4.0",
      "parseurl": "^1.3.3",
      "proxy-addr": "^2.0.7",
      "qs": "^6.14.0",
      "range-parser": "^1.2.1",
      "router": "^2.2.0",
      "send": "^1.1.0",
      "serve-static": "^2.2.0",
      "statuses": "^2.0.1",
      "type-is": "^2.0.1",
      "vary": "^1.1.2"
```

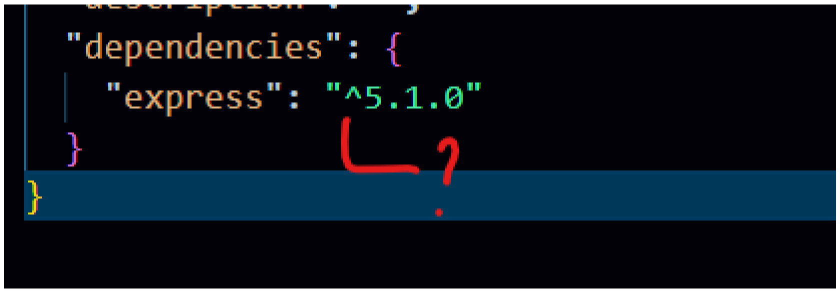
Now, What Is `package-lock.json`?

And why do we have **two different files**?

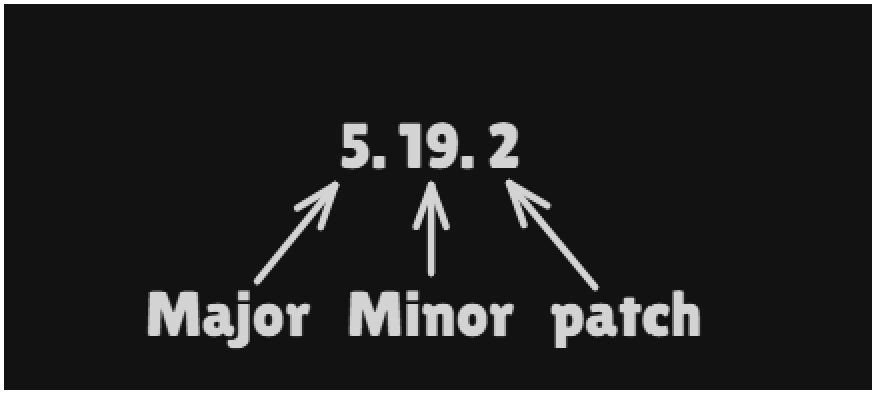
> This is a **famous interview question**:
>
> **What is the difference between `package.json` and `package-lock.json`?**

- `package.json`: Lists the **packages your project depends on**, along with their allowed versions.
- `package-lock.json`: Locks the **exact versions of all installed packages and their dependencies**, ensuring that everyone working on the project has the **same environment**.

What Is This `^` Sign?



```
  "dependencies": {
    "express": "^5.1.0"
  }
}
```



You'll often see a `^` before the version number of a package, like:

```
"express": "^5.19.2"
```

What does it mean?

- The `^` allows npm to install **any minor or patch updates** that are **compatible with the major version.**
- For example, `^5.19.2` can install `5.19.3` or `5.20.0`, but **not** `6.x.x`.

Yes, this sign is important for **maintaining compatibility**. You **can remove it**, but then npm will install **exactly that version** and no updates.

Now, Let's Finally Start Writing Our Server

First, we'll **import Express**.

Then, we'll **initialize the app** and make it **listen to a port**.

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

app.listen(3000);
```

That's it!

In just **three lines**, we've created our first server 🤯 { that's the power of Express }

But right now, it's not doing anything.

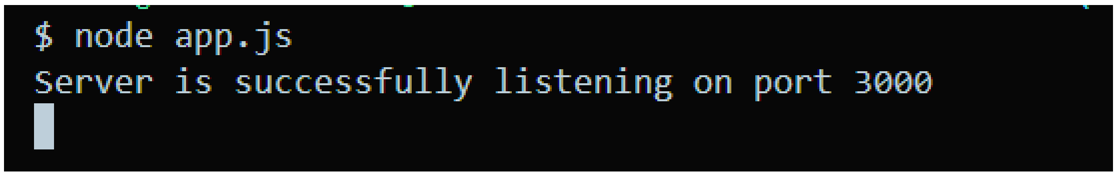
Let's **upgrade** our server a bit:

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

app.listen(3000, () => {
  console.log("Server is successfully listening on port 3000");
});
```

Now, if you run your code, you'll see this message printed in your terminal →

which means **our server has started successfully!**



```
$ node app.js
Server is successfully listening on port 3000
```

Now, How Do You Handle Requests?

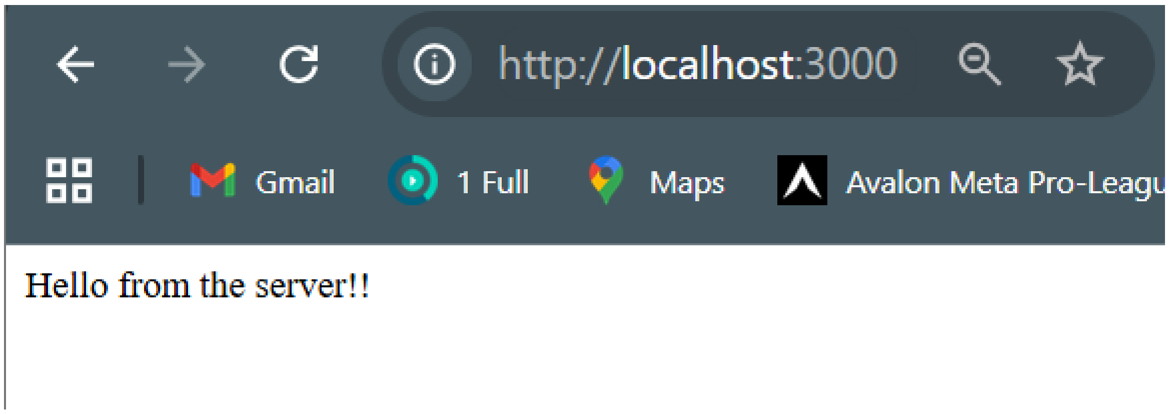
```
const express = require("express");
const app = express();

app.use((req, res) => {
  res.send("Hello from the server!!");
});

app.listen(3000, () => {
  console.log("Server is successfully listening on port 3000");
});
```

Now, **restart your server** and open your browser.

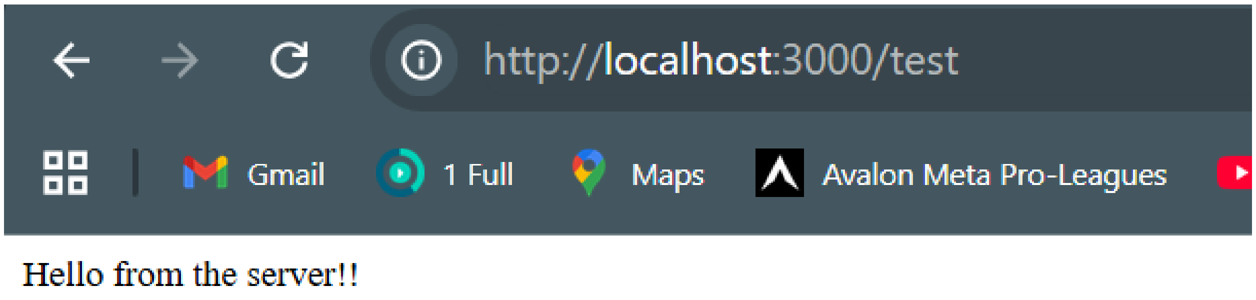
Go to http://localhost:3000, and you'll see **"Hello from the server!!"** displayed.



Hello from the server!!

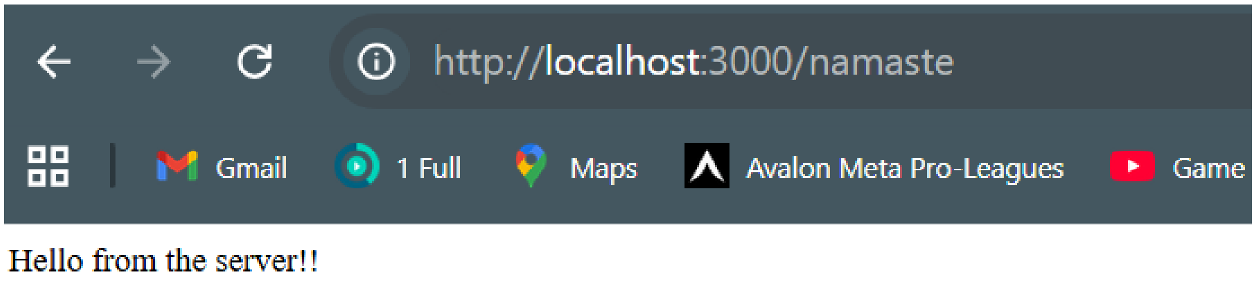
Handling Specific Routes

If you go to `http://localhost:3000/test`, it will still work, but what if we want to handle specific routes properly?



Hello from the server!!

or http://localhost:3000/namaste



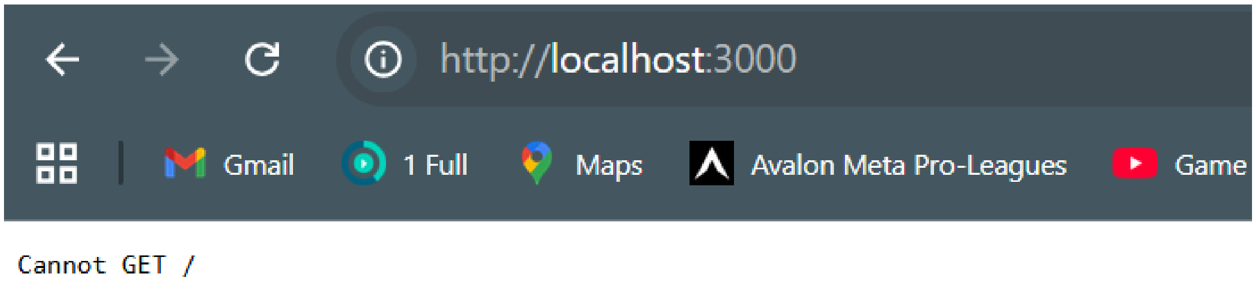
Hello from the server!!

How we can fix it?

We can define a route like this:

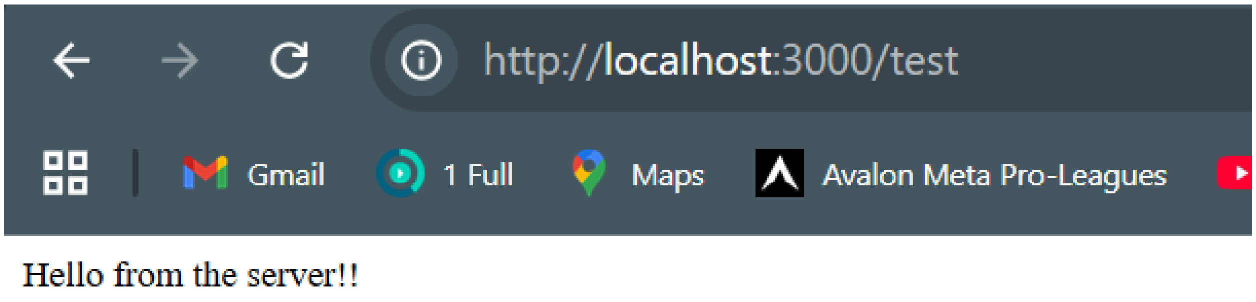
```
app.use("/test", (req, res) => {
  res.send("Hello from the server!!");
});
```

Now, if you go to `http://localhost:3000`, it will show **"Cannot GET /".**



```
Cannot GET /
```

But `http://localhost:3000/test` will work.



```
Hello from the server!!
```

We can add more paths easily:

```
const express = require("express");

const app = express();

app.use("/test", (req, res) => {
  res.send("Hello from the server!!");
});

app.use("/namaste", (req, res) => {
  res.send("namaste!!");
});
```

```
app.listen(3000, () => {
  console.log("Server is successfully listening on port 3000");
});
```

Avoiding Constant Restarts with Nodemon

Restarting the server every time can be annoying.

We can use `Nodemon` to automatically refresh the server whenever we make changes.

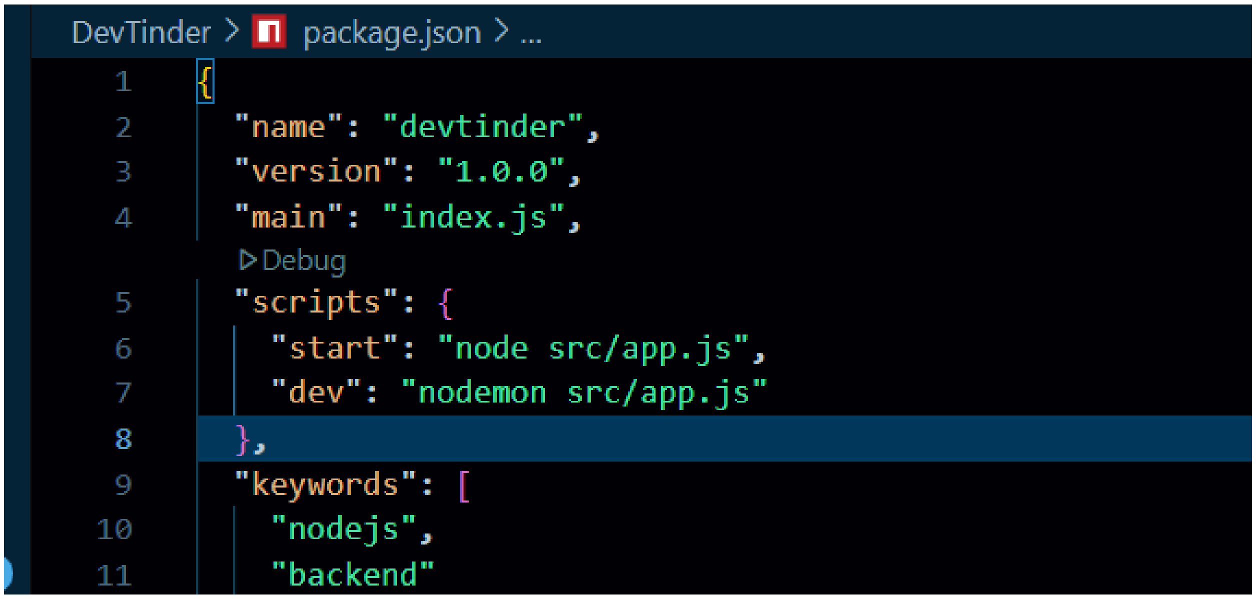
Install it globally:

```
npm i nodemon -g
```

- `g` → installs it **globally**.
- With Nodemon, any code changes will **automatically restart the server**.

Improving the Run Command

You can also update `package.json` scripts:



```
{
  "name": "devtinder",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node src/app.js",
    "dev": "nodemon src/app.js"
  },
  "keywords": [
    "nodejs",
    "backend"
```

- `npm run start` → runs the server normally
- `npm run dev` → runs the server with Nodemon, which **automatically refreshes**
- Using `npm run dev` is **more preferable** during development.

Homework

1. Create a **Git repository**.
2. Initialize the repository.
3. Understand the purpose of **`node_modules`**, `package.json`, and `package-lock.json`.
4. Install **Express**.
5. Create a **server**.
6. Make it **listen to port 7777**.
7. Create **request handlers** for `/test` and `/hello`.
8. Install **Nodemon** and update **scripts** in `package.json`.

We're so glad you were here! Until next time, Keep coding and stay curious!
V8
the next episode!
Javesóunt Engine