

MERN

Aaditya Vardhan Narain

16th September, 2025

<https://aadityanarain2003.github.io/about/>

Preface:

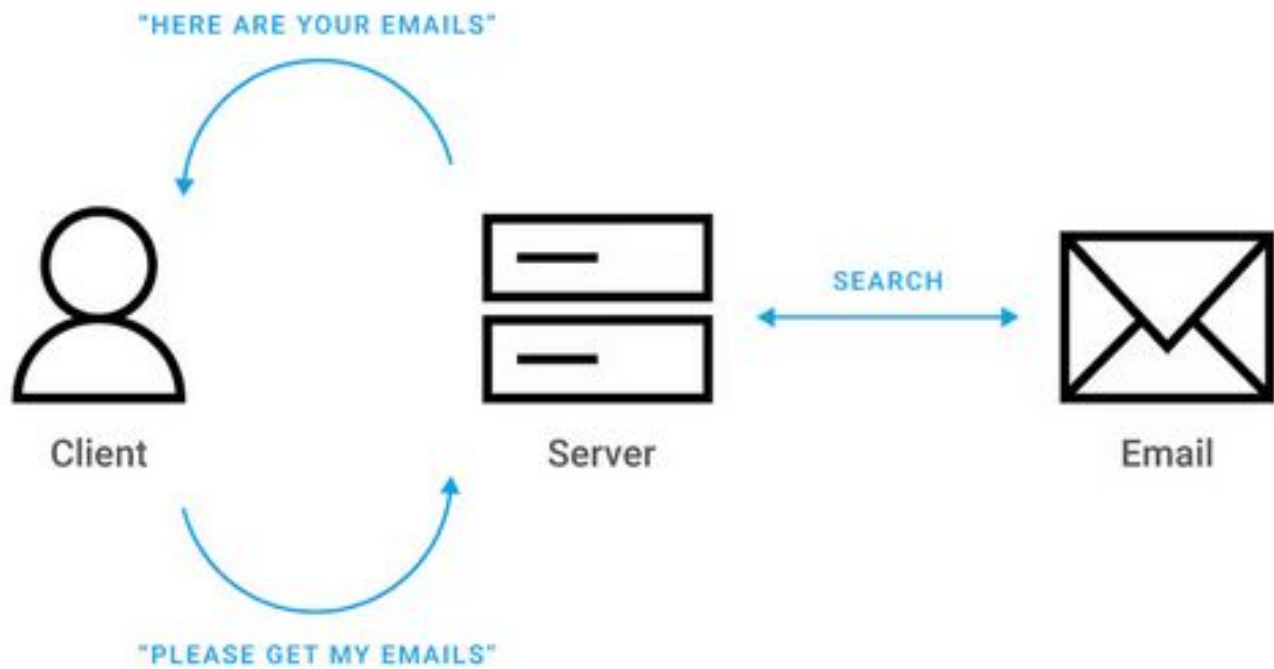
- Have node and npm installed - <https://nodejs.org/en/download>
- Be familiar with Javascript and ES6 syntax
- Be familiar with Restful and Postman
- Most likely be working with a linux based OS (I will be using linux the whole time)
- Be familiar with HTML, CSS and DOM

Some Resources:

- <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>
- <https://expressjs.com/>
- <https://medium.com/nerd-for-tech/client-server-architecture-explained-with-examples-diagrams-and-real-world-applications-407e9e04e2d1>

Content: (Node and Express)

1. Client - Server Schema
2. Why MERN
3. What are we gonna do in both these sessions
4. The actual content



Why MERN ???

M – MongoDB (Database)

E – Express.js (Backend framework)

R – React.js (Frontend library)

N – Node.js (Runtime environment)

one language, modern stack, fast development, easy JSON flow.

It is a **single page application**, with a **decoupled architecture**.

It is **reusable**.

It is **real time and dynamic**.

It is build on **Restful** or **GraphQL**.

Two Sessions (9th and 12th)

Session 1:

- Understand what is **Node and Express**.
- We will build a **single server** for a To-Do List App.
- **CRUD API** calls for **In-Memory** system for this To-Do List Server.
- Express **File Structure and Routers**.

Take Home 1:

- Add **support for MongoDB**. That is to make sure the Data in this To-Do List Server is Persistent

Session 2:

- Understand **what is React**.
- Understand how to **design Components**.
- Understand what **Hooks** are.
- Understand **what CORS is**.

Take Home 2:

- **Dockerize** the full MERN app into a container and use it locally as your personal To-Do App.

Setup your folder initialize the app

- Initialize a folder
- `git init`
- Create a folder called **Backend**
- `npm init -v`

You will notice that a **package.json** file got created in this folder with some initial values.

- Now do 'npm install express'

You will now notice that two items get created which are **node_module** folder and **package-lock.json**

- Now add the **.gitignore** file in the root directory (you need to only add what is specified by us)
- Create a new file in the backend folder called **server.js**

git checkout step-01

Add the following code to server.js

```
import express from "express";

const app= express();

app.listen(5001, () => {
  console.log("Server started at Port 5001");
});
```

Go to package.json and add this

```
"dev" : "node server.js"
```

Inside the script { }

You got an alias now, you can simply run **'npm run dev'**

On the terminal run **'node server.js'**

And voila you ran your first express server

But do you see an error printed on the first line

Go to package.json and add this

"type": "module"

Type Module tells that we are using the **modern ES syntax** and not old school JS

But what is this package.json...

In one sense it is the core file in a Node application. It is a blueprint for your application.

Defines your project (name, version, description)

Lists dependencies (like Express, React, etc.)

Defines scripts you can run (like `npm start`, `npm test`)

Contains metadata needed for Node.js and npm

```
{  
  "name": "backend",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit  
1",  
  },  
  "type": "module",  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^5.1.0"  
  }  
}
```

<https://docs.npmjs.com/cli/v11/configuring-npm/package-json>

Don't Mess with package-lock.json

If you noticed, after you ran `npm install express`, two new things popped up, one is `package-lock.json` and the other is `node_module`.

Always commit `package-lock.json` if you building an app.

`package-lock.json` is a **lockfile** that:

- Records the **exact versions** of all installed packages (and their dependencies).
- Ensures that **everyone working on the project gets the exact same setup**, even across different machines or at different times.
- Is **automatically generated** when you run `npm install`.

Let's Now create some simple APIs

```
app.get("/api/notes", (req, res) => {  
  res.send("You got some notes");  
});
```

Add the above code to your [server.js](#) file before `app.listen`.

And run 'npm run dev'

Go and hit this url on your browser

`http://localhost:5001/api/notes`

What do you notice that you got a response

Let's understand some format. This uses ES6 arrow function where `app.get` makes a callback to the function you wrote.

"api/notes" is the endpoint

"req" is whatever requests are coming in

"res" is whatever response is going out

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides>

Isn't it annoying to stop and start the server again and again...

Here comes '**Nodemon**' to save you

run:

npm install -g nodemon

Add this to scripts in package.json

```
"dev": "nodemon server.js"
```

It watches for changes in your files
(e.g., **.js**, **.ts**, **.json**, etc.)

When a file changes, it **restarts your app automatically** — no need to stop and start the server manually every time.

git checkout step-02

What are Routers

A **router** in Express is like a **mini Express app** — it can:

- Handle routes (like `/users`, `/posts`, etc.)
- Use middleware
- Be mounted in your main app

`export default` is used to **export a single value, function, object, or class** from a file (module), so that it can be **imported easily elsewhere**.

Instead of writing everything in `server.js`:

```
app.get('/users', ...)
```

```
app.post('/users', ...)
```

```
app.get('/products', ...)
```

```
app.post('/products', ...)
```

You split them:

- `routes/users.js` → handles `/users`
- `routes/products.js` → handles `/products`

Cleaner, modular, and easier to maintain

File Structure

backend

- package.json
- .env
- src
 - Other files in src (that is your source code)
 - [server.js](#)
- *.[config.js](#) files

It can be anything you like. There is no standard across. I use the one on the right hand side

git checkout step-03



What are Dev Dependencies

Dev dependencies are simply packages that are required during development and not in production.

These won't get added to the build image.

```
"devDependencies": {  
  "@babel/core": "^7.28.3",  
  "@babel/preset-env": "^7.28.3",  
  "babel-jest": "^30.0.5",  
  "supertest": "^7.1.4"  
}
```

What is .env file

A **.env** file is a **text file** that stores **environment variables**. key-value pairs used to configure your app without hardcoding secrets or environment-specific settings in your code.

npm install dotenv

```
import dotenv from "dotenv";  
dotenv.config();
```

Why use a **.env** file?

- Keep secrets (DB URLs, API keys) **out of source code**
- Easily **switch between environments** (development, testing, production)
- Avoid hardcoding config values inside JS files
- Safe to change without touching the code

git checkout step-04

Use this as your env file

```
PORT=5001
```

```
NODE_ENV=development
```

```
MONGO_URI=mongodb://127.0.0.1:27017/SSD
```

What is a middleware

Middleware is a **function** that has access to the request, response, and next function in the lifecycle of an HTTP request in an Express app.

Typical Use Cases:

- Logging
- Authentication

```
function middleware(req, res, next) {  
  // Do something with req or res  
  next(); // Pass control to the next middleware  
}
```

Used with

```
app.use(express.json());  
app.use("/api/notes", router);
```

git checkout step-05

Your Take Home Tasks

Compulsory Tasks:

1. **Port these API to your own MongoDB clusters. It can be either local or cloud storage. Use Mongoose for the same.**
2. **Try to Make it work with local itself.**

Extra if you want to:

1. **Can you play around and add other routers to it.**
2. **Can you create two or more servers with Mongo, Express and Node and make them talk to each other ???**

Due Date: 18th Sep, 2025 11:59 p.m.