MERN

Aaditya Vardhan Narain
16th September, 2025
https://aadityanarain2003.github.io/about/

Preface:

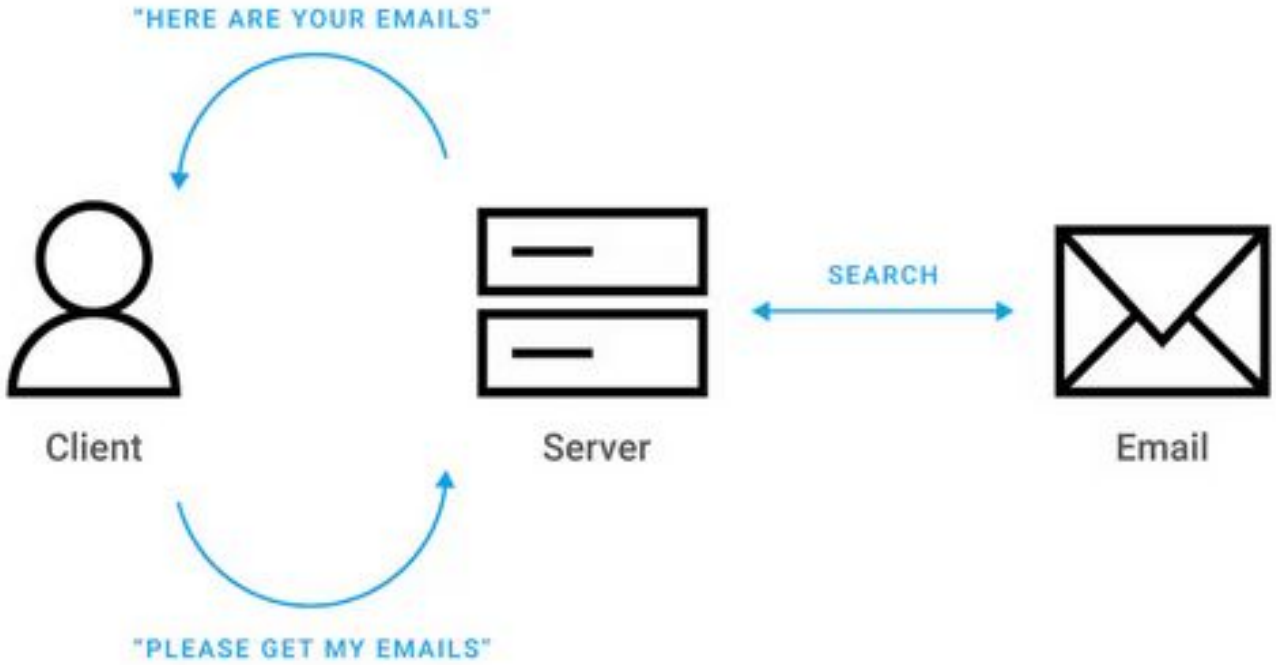
- Have node and npm installed - https://nodejs.org/en/download
- Be familiar with Javascript and ES6 syntax
- Be familiar with Restful and Postman
- Most likely be working with a linux based OS (I will be using linux the whole time)
- Be familiar with HTML, CSS and DOM

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


"HERE ARE YOUR EMAILS"
SEARCH
Client
Server
Email
"PLEASE GET MY EMAILS"

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.

MERN

Aaditya Vardhan Narain
19th September, 2025
https://aadityanarain2003.github.io/about/

Preface:

- Make sure you revise what was taught on 9th.
- Complete the compulsory task.

Some Resources:

- https://react.dev/
- https://tailwindcss.com/

git checkout step-06

How to setup react

Go to the root directory

- **mkdir frontend**
- **cd frontend**
- **npm create vite@latest .**

Then select React

Click Enter

Then select Javascript

Click Enter

Voila !!! Your basic react is setup

Let’s Understand what each of these part is

/frontend$ ls

eslint.config.js

index.html

package.json

public

README.md

src

vite.config.js

package.json is the same as this also uses node package manager

index.html is the main page of the React App (React is a SPA)

you do not need to worry about vite.config.js and eslint.config.js for now

public and src are two folders where your code will be mainly kept

git checkout step-07

What is JSX

In simple terms - “*JSX* is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file. “

Rules of JSX:

1. **Return a single root element**
2. **Close all the tags**
3. **camelCase ~~all~~ most of the things**

```
export default function Bio() {
  return (
    <div>
      <div className="intro">
        <h1>Welcome to my website!</h1>
      </div>
    </div>
  );
}
```

https://react.dev/learn/writing-markup-with-jsx

What is tailwind

To install Tailwind in the frontend directory

npm install tailwindcss @tailwindcss/vite

In vite.config.js make the following changes

vite.config.ts

```ts
import { defineConfig } from 'vite'
import tailwindcss from '@tailwindcss/vite'

export default defineConfig({
  plugins: [
    tailwindcss(),
  ],
})
```

@import "tailwindcss";

in index.css or any other css which you are using

And then you can use it across wherever you are applying vite

Why tailwind

- Fast Development
- No Naming Conflict
- Customizable

Example: (instead of writing the below)

```
.title {
  font-size: 2rem;
  color: blue;
  text-align: center;
}
```

You can write these:

```
<h1 className="text-2xl text-blue-500 text-center">Hello</h1>
```

Not a requirement to develop react app, but can come very handy. It is a development tool.

React has loads of dev tools…

git checkout step-08

Let’s Understand what index.html and the content in src is…

```
<div id="root"></div>
    <script type="module"
src="/src/main.jsx"></script>
```

The above is one of the most important part of the index.html.

The above directs main to get the root div of index.html and open it in StrictMode

StrictMode is nothing but a development tool that helps you to detect bugs.

App is where you write the actual code.

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).ren
der(
 <StrictMode>
    <App />
 </StrictMode>,
)
```

Remember SPA… This is what it is all about

Let’s Use Browser-Router

It allows your app to **handle routes using the browser's URL**, without refreshing the page.

npm install react-router-dom

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { BrowserRouter } from
'react-router-dom'

createRoot(document.getElementById('root')).ren
der(
 <StrictMode>
   <BrowserRouter>
     <App />
   </BrowserRouter>
 </StrictMode>,
)
```

Using Link, Routes and Route

`<Link>` = **Navigate**

`<Route>` = **Render something if URL matches**

Link is like the html tag <a> and Route is actually for that particular element to render.

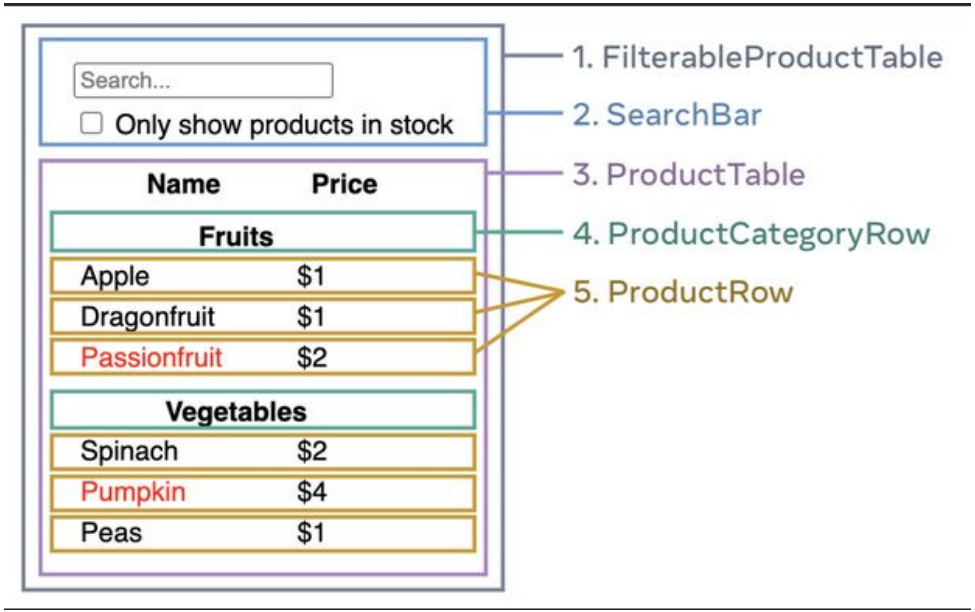
Do note that it's an SPA and it is the same page, just the components are changing.

```
import { Routes, Route, Link } from
'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
export default function App() {
 return (
   <div>
     <nav style={{ marginBottom: '20px' }}>
       <Link to="/">Home</Link> |
       <Link to="/about"> About</Link>
     </nav>

     <Routes>
       <Route path="/" element={<Home />} />
       <Route path="/about" element={<About />} />
     </Routes>
   </div>
 );
}
```

git checkout step-09

What is a component



All the coloured boxes you see are different component.

It’s a segregation of concern from a design point of view, where you do not need to worry about all things at once.

You can create various components .

The best parts its reusable like classes. So create once define parameters and use forever.

You can use props to pass values. (Explore this on your own)

git checkout step-10

What is a Hook

Hooks are **functions** that “hook into” React features like state, lifecycle, or context.

They let functional components **have state, side effects, refs**, etc.

You **cannot use hooks in class components**—they are only for functional components.

Examples of built-in hooks:

- `useState` → state
- `useEffect` → side effects (like lifecycle methods)
- `useRef` → reference to DOM elements
- `useContext` → context AP

`useState`

- **Purpose:** Store and manage **state** in a functional component.
- Returns a **state variable** and a **function to update it**.
- Every time the state changes, React **re-renders the component** automatically.

`useEffect`

- **Purpose:** Run **side effects** in functional components.
- Side effects = anything that interacts with the outside world or occurs **after render**, like:
  - Fetching data from an API
  - Subscribing/unsubscribing to events
  - Setting timers
- `useEffect` can mimic **componentDidMount, componentDidUpdate, componentWillUnmount**.

git checkout step-11

CORS

CORS is a policy enforced by web browsers to control which domains (origins) can access resources (e.g., APIs, files) from your server. It comes into play when:

- A frontend (e.g., JavaScript running in a browser) makes requests to a backend server.
- The frontend and backend are hosted on **different origins** (e.g., http://frontend.com vs. http://api.backend.com).
- from fastapi.middleware.cors import CORSMiddleware

```
// CORS configuration

const corsOptions = {

  origin: 'http://localhost:3000', // frontend origin

  credentials: true,              // allow cookies to be sent

  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'], // allowed HTTP methods

  allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With'], // allowed headers

};

app.use(cors(corsOptions));
```

Your Take Home Tasks

Compulsory Tasks:

1. **Remember the To-Do List. Create a React app that talks to the server you created earlier and fetches all notes, edits a note, deletes a note and creates a note. UI is upto you, can be minimal. (NOTE: You might have CORS issue)**

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

Extra if you want to:

1. **Can you Dockerize it and use Nginx as a server to serve both React build and Node server. (Will be useful in the next lab)**

Docker:
https://medium.com/@techsuneel99/docker-from-beginner-to-expert-a-comprehensive-tutorial-5efec10c82ab

Nginx:

https://www.youtube.com/watch?v=q8OleYuqntY