**Setup + Node.js**

1. Create frontend and backend folder.

2. Open terminal and initialize node: ***npm init -y***

3. Then install express: ***npm install*** [***express@4.18.2***](mailto:express@4.18.2)

4. Create an **index.js or app.js or server.js** file

5. Go to package.json and add **“type”:”module”** (If you want the express to be imported using

***import express from “express”****),* Else

Import it using: ***Const express = require(“express”)***

**-** If you want it to run using ***npm run dev,*** you need to add ***“dev”: “node server.js”*** in **scripts**

6. Start the server using – ***node server.js***

7. Everytime you make changes in the server.js, you will be required to restart the server to reflect them. To solve this problem, we will install nodemon. ***npm install nodemon -D.*** Also change the dev from node server.js to nodemon server.js

8. Create a routes folder in backend and a notesRoutes.js to manage the requests.

9. server.js **-->** notesRoutes.js **-->** notesController.js

10. A good practice is to create a ‘src’ folder and put all the files related to applications in it

11. When you move files into the src folder, also update the script in package.json file.

**MongoDB**

1. Install mongoose: ***npm install*** [***mongoose@7.0.3***](mailto:mongoose@7.0.3)

2. Create ‘config’ folder inside ‘src’

3. process.exit(1) , here 1 means exit with failure, 0 means success

4. Put the name of your database before the? in the link you copied from MongoDB atlas

5. This code is not yet safe because if you upload this file on github, people can see your connection string and thus use your database. This is where .env file comes into play. Create a .env file and put the connection string in MONGO\_URI = connection string (without quotes)

To use the above file, you need to install npm dotenv

6. You can use the variables from dotenv only when you import dotenv from “dotenv” and then call a function dotenv.config()

Similarly, put your PORT i.e. 5001 in this case in. env

And console.log the PORT dynamically.

7. Create a folder named ‘models’ and create a Note.js inside it (naming convention is first letter capital and singular)

8. Inside Note.js, create a schema and the create a model based off that schema

9. In our application till now we are listening first, starting the server first and then connecting to the database which is a bad practice. So change the code to connectDB.then(() => *listening code)*

So we first connect to the database and only then start listening.

**Express**

1. Without the middleware i.e without the code app.use(express.json()) the controller would not be able to get the response from the request. And the title and content would be undefined because the req.body will be undefined.

Middleware helps you parse the JSON body: req.body

**Rate Limiting**

1. Go to upstash and create a database and copy the .env variables and paste into the .env file.

2. Install the package npm install @upstash/ratelimit@2.0.5 @upstash/redis@1.34.9

3. Go to config folder and create a file upstash.js

4. Write code in upstash.js

5. Create a middleware folder in src and in it create rateLimiter.js

6. import the rateLimit from upstash.js into the rateLimiter, write its function and export the rateLimiter.

7. Use the middleware app.use(rateLimiter) in the server.js

8. In rateLimiter.js when you apply authentication, instead of “my-limit-key”, there you write user\_id so that every user is rate limited specifically

***Frontend***

1. ***npm create vite@latest .***

And then select React and JavaScript

Then ***npm install*** to install all the dependencies

2. ***npm install react-router –*** For different pages

3. Delete the folder assets, delete app.css, clear the contents of index.css and app.jsx.

4. In app.jsx, type rafce

5. In main.jsx, under <StrictMode>, add <BrowserRouter> and import it. We have wrapped our entire app with browser routing and can apply it on our website.

6. Create pages folder and create your pages in it.

7. Add Routes to the pages in App.jsx

8. Install react hot toast using: npm add react-hot-toast - react for notifications in the applications

9. You can either put it under the App.jsx or the Main.jsx.

10. Create a button in App.js and test the toaster.

11. Install the tailwind for vite by going to the website and copying the code from the framework guide.

Delete the content of tailwind.config.js file and copy the next code of the documentation and paste it there

Then update the index.css file using the next code on the documentation.

12. Install DaisyUI on top of Tailwind to help simplify and reduce the syntax.

Npm install [daisyui@4.12.24](mailto:daisyui@4.12.24)

Follow steps mentioned on the daisyui website on how to use it in tailwind (Add it in the tailwind.config file)

13. Go to the tailwind.config and add the theme by creating a daisyui attribute, now type data-theme=”theme\_name” in the div of App.jsx

14. Install the lucide-react for some symbols that we will use and the axios package so that we can replace our fetch APIs with the axios

15. Create a components folder and start adding components to it.

16. To prevent the CORS error and make the API allow call from frontend to backend, first npm I cors, then go to server.js and call the app.use(cors()) and import it. If you add origin inside the app.use(cors()), it only allows access from that url.

***Deployment***

1. Put the gitignore file from the frontend folder to the root folder that is the folder where you have frontend and backend so that it applies to both of them

2. Add .env to the gitignore

3. Open terminal in the main folder and type –

*git init*

*git add .*

*git commit -m “initial commit”*

Now go to github, create repository and copy and paste the second code from it and your code will be uploaded to github.

4. Run npm init -y in main folder

Now, go to package.json and change test to build and remove its text and write –

“npm install --prefix backend && npm install --prefix frontend && npm run build –prefix frontend”

Npm run build –prefix frontend will create an optimised version of your react application frontend

5. Though we can deploy frontend and backend on different domains, we will in this case deploy them in the same domain.

Also, if you have the client and backend under the same domain, we can get rid of CORS error automatically.

6. go to terminal and: ***npm run build***

7. Go to server.js and import path from “path”

App.use(express.static(path.join()))

Const \_\_dirname = path.resolve()

Console logging - ‘\_\_dirname’ will give you the source of backend

So now go in path.join method and make it path.join(\_\_dirname, "../frontend/dist")

The below code says that implement the code only if we are satisfying the if condition i.e we are using render.com

if(process.env.NODE\_ENV === "production"){

    app.use(express.static(path.join(\_\_dirname,"../frontend/dist")))

app.get("\*", (req,res)=>{

    res.sendFile(path.join(\_\_dirname,"../frontend","dist","index.html"))

})

}

Similarly,

Write an if statement for app.cors that:

if (process.env.NODE\_ENV !== 'production') {

    app.use(cors(

        {

            origin: "http://localhost:5173"

        }

    ));

}

8. Add start command in package.json below build:

“start”: “npm run start –prefix backend”

9. Add NODE\_ENV = production in .env file