MERN (installations and setup)

Folder Structure

- *Make 2 folders **client** and **server.***

Client contains :

How to Create React App with Tailwind?

1. Npm create vite@latest appname
2. Now choose react and then javascript
3. O + enter to run the app
4. Now close the app for a sec and install :
5. npm install -D tailwindcss postcss autoprefixer
6. npx tailwindcss init -p
7. now edit tailwind.config.js

```
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

8. Now add this in index.css

```
@tailwind base;

@tailwind components;

@tailwind utilities;
```

9. Delete app.css and now run “npm start” to check if it is working or not
10. Use https://nerdcave.com/tailwind-cheat-sheet if you want to manually do the tailwind or google the components and directly copy paste the code
11. Download extension in vs code “**Tailwind CSS IntelliSense**”
12. Now delete unnecessary files and remove the unnecessary codes from index.html and also App.jsx
13. Here your main.jsx contains your <app/> (i.e it wraps your entire app)
14. Now all the routes will be there in the app.jsx

- **How to create routes?**

```jsx
import { BrowserRouter as Router, Routes, Route } from
"react-router-dom";
import React from "react";
import LoginPage from "./pages/loginpage";

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path = "/" element = {<LoginPage/>}/>
      </Routes>
    </Router>
  );
};

export default App;
```

15. Now make folders such as pages, components, context, assets (all this will be there in the src folder)

    - Here pages folder contains the pages which are main routes of the website and all the components related to that page are there in the components folder.

16. Now you can start creating the layout of the pages

17. You must keep the names of the files in the camelcase.

18. You also can change script and write this instead.

"start": "vite"

And this way you can directly run app using npm start

- Now after all the apis are made in the server to fetch them in the client, we need to install and import axios library in client and cors middleware(it enables fetching of apis from backend to frontend) in the server.
- Right our app is running via two URLs (localhosts) separately i.e one for client and one for server.

So to merge those (to run the app only on 1 localhost) we need to perform some steps :

--> before you do this step make sure that your client is completely ready.

How to integrate backend with frontend?

Server Contains:

How to create a server in the node.js?

1. npm i express
2. npm i dotenv (for env variables)
3. npm init -y (for package.json file)
4. npm install --save-dev nodemon (live server for node applications)
5. now make .gitignore file and write :

node_modules

env

6. now add in scripts in package.json and write :

“start” : “nodemon index.js”

7. Now the structure of your index.js file will be :
    1. Imports
    2. Db connection
    3. Middlewares
    4. Routes
    5. Server starting
8. now create server in the index.js file and check if it is working properly

```
import "dotenv/config";
import express from "express";

const server = express();

server.get("/", (req, res) => {
   res.send("Hello, world! The server is running.");
```

```
 });

server.listen(process.env.PORT,()=>{
    console.log('server started at http://localhost:${process.env.PORT}/');
//process.env.PORT is used to access the env variables.
});
```

9. Add folders in the server such as controllers, routes, models

10. Now connect the database

npm i mongoose (in the server) (install)

Import {connect} from “mongoose”; (import)

```
main().catch((err) => {

    console.error("DB Connection failed: ", err);

  });

  async function main() {

    try {

      await connect(process.env.MONGO_URL);

      console.log("DB Connected !!");

    } catch (error) {

      console.error("Database connection error: ", error);

    }

}
```

11.Now add middlerwares

```
server.use(express.json());
```

11.Now create schema for the model in the models folder for that particular model and then create model and export it.

12. Now import the model and Now create controller file for that model and then make apis in that file.

13. Now import controller in this file and Now create route for that particular model and then export it

14. Now import it in index.js to use as middle wares.

For AUTHENTICATION :

(INSTALL)

npm i bcrypt

npm i jsonwebtoken

(IMPORT)

Import {hashSync, compareSync} from “bcrypt”;

import jwt from “jsonwebtoken”;

(and now you can use it)