SETTING UP THE PROJECT DIRECTORY

- Create a root directory name ‘Mern_Project’ or whatever you want
- Create a folder name ‘Backend’ in root directory
- Open that root directory in VS Code

GETTING STARTED WITH BACKEND

- Open terminal in ‘Backend’ folder & run ‘npm init -y’
- Now install your very first package express by running ‘npm i express’ in Backend’s terminal
- Now create a file in ‘Backend’ folder name ‘server.js’
- Require express first in ‘server.js’ file
- Create routes for ‘Home page’ & ‘Registration page’ & create server using express

Tired of stopping & starting server again & again?

- Install another package in ‘Backend’s’ terminal name ‘NODEMON’
    - **Nodemon** automatically restarts your Node.js server whenever you make changes to your code.
- Now we have to separate our code
- We will separate sever.js file’s code in 2 other files

EXPRESS ROUTERS, CREATING & ORGANIZING ROUTERS

- First of all, we will separate our ROUTING
- Create a folder name ‘routes’ in ‘Backend’
- Create a file name ‘auth-router.js’ in ‘routes’ folder
- Require express & initialize **Router** function there
- Take **home & register** code of routing to ‘auth-router.js’ file from ‘server.js’
- Don’t forget to export those routes because we will have to import those routes in ‘server.js’ file
- After importing we have to use **app.use()** middleware for using our routes from ‘auth-router’
    - **app.use()** is a method in Express used to apply middleware or mount routers at a specified path for handling incoming requests.

CONTROLLERS IN EXPRESS, MANAGE YOUR APPLICATION’S LOGIC

- Now we will move our logic code to controllers, and let ‘auth-router.js’ handle only the routing
- Create a folder name ‘Controllers’ in ‘Backend’ & create a fie name ‘auth-controller.js’
- Now take your logic code in ‘auth-controller.js’ file by creating specified methods for different pages there (i.e. **Home, Register, Login** etc.)
- **Again**, don’t forget to export those methods from ‘auth-controller.js’ because we will have to import those methods in ‘auth-router.js’ file

USER REGISTRATION IN EXPRESS WITH POSTMAN

- Introduction to **Postman** (Theory)
- Open Postman, write your one of routes(http://localhost:8000/api/auth/register) URL in Postman & send GET request for Registration route
- Send a **POST** request for Registration route
- Use JSON Middleware in Express
    - **app.use(express.json())** is a middleware that parses incoming JSON requests and makes the data available in **req.body**.

CONNECTING BACKEND WITH MONGODB (Node.js & Mongoose Connection)

- Create a folder name '**Config**' in 'Backend' & create a file name **'db.js'** inside
- Installing **Mongoose(npm i mongoose)**
    - **Mongoose** is an ODM (Object Data Modeling) library for MongoDB and Node.js that provides a schema-based solution to model application data.
- Writing **Mongoose** connection code
- **Againnnn**, after writing mongoose connection code in a function, don't forget to export that function, because we will have to import that function in **'server.js'** file
- After importing that 'connectDb' function in 'server.js' file, integrate it with your server listener

Creating the User Schema & Model for User Registration

- Introduction to **Schema & Model**
- Understanding the concept of **Schema**
- Now create a folder name ‘models’ in ‘Backend’ & create a file name **’user-model.js’** inside
- Now write the schema for your registration model using mongoose
- Then create a model using mongoose by naming it (means your collection name)
- **Once Againnnn,** don’t forget to export the model, because we will have to import it in our auth-controller file

Storing Registered User Data in MongoDB

- Retrieve User Data Using **req.body**
- Destructuring the **req.body** data
- Check if email already exists
- Store data if email doesn’t exist by using **create()** method

  But why not insertOne() instead of create() ?

  - **We use create() instead of insertOne() in Mongoose because create() includes schema validation and middleware support, which insertOne() does not.**
- Testing User data storage with **Postman**

Secure User Password using Bcrypt.js

- Install **'bcryptjs'** in Backend
  - **bcryptjs** is a JavaScript library used to securely hash and compare passwords using the bcrypt hashing algorithm.
- Implementing **'bcryptjs'**
- Understanding **'Hash'** method
  - **Hash** method is a function that converts data, like a password, into a fixed-length string of characters that hides the original input.
- Understanding **'Salt'** & **'Salt-Round'**
  - **Salt** is a random string added to a password before hashing to make each hash unique and enhance security against attacks.
  - **Salt-Round** is the number of times the hashing algorithm processes the password and salt, increasing the computation time to enhance security.
- Testing User data storage with **Postman**

Secure User Authentication withss JSON Web Token (JWT)

Slight Introduction to Authentication & Authorization

- Understanding JWT (JsonWebToken)
  - JWT (JSON Web Token) is a secure, compact token used to transmit data between parties as a JSON object, commonly for authentication and authorization. It's always stored in Local Storage or Cookies.
  - Header Specifies the type of token and the signing algorithm used.
    Example: "alg": "HS256", "typ": "JWT"
  - Payload Contains the claims or user data being transmitted (e.g., user ID, email, isAdmin).
    Example: "userId": "123", "email": abc@example.com
  - Signature a hashed value created using the header, payload, and secret key to verify token integrity and authenticity.
    Example: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Working with JSON Web Token (JWT)

- Install 'jsonwebtoken' package using 'npm i jsonwebtoken'
- Implementing jsonwebtoken
- Creating payload and signature for JWT in user authentication using userSchema in 'auth-model.js file'
- Calling & verifying JWT functionality in login in 'auth-controller.js' file

Contact Form Functionality

- Create files **'contact-model.js' , 'contact-controller.js' , 'contact-router.js'** in models , controllers , routes folders respectively
- Create contact schema in 'contact-model.js' file using mongoose
- Export & import contact model from 'contact-model.js' to 'contact-controller.js'
- Write contact form backend code in 'contact-controller.js' file
- Export contact form function from 'contact-controller.js'
- Import it in 'contact-router.js' file
- Create a post route for contact form and export it & import it in 'server.js' file

Introduction To ReactJs, Installation & Working With ReactJs

- **Introduction to ReactJs**
- Open your root folder 'mern_project' in which you created 'Backend' folder for node.js
- Open terminal in root folder & run '**npx create-react-app frontend**' (npm install -g npm)
- You'll see the folder name "**frontend**" in your root project directory
- Let's understand the project directory structure
- Running react page on browser using '**App.js**' file in '**src**' folder

Creating Pages for our Project

- Create a folder name ‘pages’ in ‘src’ folder
- Now create all files for your pages (i.e. **Home.js, Services.js About.js** etc.) in ‘pages folder’
    - **All the files you are creating in ‘src’ folder either for pages or components, make sure their first letter of name should be capital i.e. Home.js, Services.js, it is necessary**
- Install **ES7 React/Redux/GraphQL/React-Native snippets(write ES7 in search bar, you’ll find it in first place)** extension in vs-code for shortcuts to quickly write commonly used pieces of code in React
- Now write code for every page you created
- Create function in every file & don’t forget to export
- We will generate code using that extension, we just installed
- We will use ‘**rfc**’ or ‘**rfce**’ for creating functional component in every file
- After all these, now the most important work, we have to create routes for every page & register in ‘**App.js**’ file

Understanding Routing in Reactjs

- Install a package name **react-router-dom** using '**npm i react-router-dom**' in '**frontend**' folder's terminal
    - React Router DOM is a standard routing library for React that enables navigation between different components and pages in a single-page application (SPA) without reloading the browser.
- Import **'BrowserRouter , Routes , Route'** in 'App.js' file
    - **BrowserRouter -** Wraps the app and enables history-based routing
    - **Routes -** Container for all <Route> elements
    - **Route -** Defines a path and the component/page to render
- Now create a container tag of '**BrowserRouter**', inside it, create another container tag of '**Routes**', inside it, create a standalone tag of '**Route**' for every page's route
    - **Route** tag contains two attributes,
    - **Path -** Defines the URL pattern for the route. ('/home' , '/about')
    - **Element -** Specifies the React component to render when the path matches. ({ <Component_Or_Page_Name/> })
- Don't forget to import all the pages in '**App.js**' file for registering their routes (i.e. import Register from './pages/Register')