**Web Development using MERN Stack**

MERN stack refers to MongoDB, Express.js, React.js and Node.js tools altogether as a development stack to develop a website.

Basic Architecture of a website: A website is mostly a combination of the 3 most important components- frontend, backend, and database. The front end is the front page of the website that a user interacts with while the backend is the server-side end where the application sends API requests to make the components of the website interact with one another. And finally, the database is the data storage unit where all the dynamic data is stored.

**Front-end Development frameworks and libraries**

Among the latest front-end frameworks and libraries, React, Angular, Vue, jQuery and Bootstrap are the most popular ones used in the industry. In today’s market, React.js and Vue are on the rise, but these frameworks are independent frameworks which always developers to work with anyone of these independently.

Besides the functioning side of the frontend, there lies the structural and designing part. HTML and CSS are the strong tools for that purpose. With HTML the textual structure of the website is developed while with Cascading Style Sheet (CSS), that raw structure is designed.

**Back-end Development frameworks and libraries**

The back end or server side of a website can be developed using JavaScript frameworks like express.js and node.js. Other languages like Python, Java, C have their own backend frameworks like Django, Flask from Python, Asp.Net from C, Spring from java and such. In MERN stack, Express.js is used as the backend development library accompanied by Node.js as the runtime environment.

The backend of a website deals with the data rendering process which means that the data is passed form the backend to the frontend of the website. The front end sends API requests to the server which is then converted to an equivalent query for retrieving the required data from the database. This framework is also called the MVC model. The Model, View, and Controller model in which the model is responsible for all the data, the view component deals with the front end while the controller acts as the connector between model and view.

Difference between MERN and MEAN

|  |  |
| --- | --- |
| MEAN | MERN |
| MongoDB, Express, Angular, Node | MongoDB, Express, React and Node |
| It uses TypeScript Language | It uses JavaScript Language |
| It is a JS Framework | It is an open-source JS library |
| Tha data flow is bidirectional, any change in the backend or frontend gets automatically updated on the other side | Tha data flow is unidirectional, any change in the backend or frontend needs to be rendered separately on the other end |

Front-end Framework- REACT

React.js is an open-source reusable JavaScript based library which is used to develop the front-end of a web application. It was developed by John Walker, a software engineer of Facebook. Some common features of react includes:

* Reusable components
* Unidirectional flow of data
* Small Learning curve
* Can be used for mobile apps such as: React Native
* Easy debugging tools
* Uses JSX (JS + HTML)

JSX: const abc = <h2> Hello </h2>;

Virtual DOM: Document Object Model defines how documents are accessed and manipulated in a webpage. DOM treats an XML or HTML doc as a tree structure in which each node is an object representing a part of the document. In DOM, a tag is referred to as an element.

React keeps a copy of the real DOM in its memory called the vDOM. MAnipuolating vDOM is faster than the real DOM and whenever the state of an object changes, vDOM changes that object only in the real DOM instead of the updating all the objects. This makes the process very slow. As such, react encounters this by comparing the previous state of the vDOM with the updated state of the vDOM and gets to know about the objects that were updated.

Components in React: Components are building blocks or elements of a react app that represent a part of the UI. Such as, the login/registration part of the UI, app in the UI, any image in the UI is separate React components. Components are:

Reuasble: a single component can be reused multiple times in the application.

Nested components: A component can contain several other components.

Render: A component must define a render method that specifies how the component renders to the DOM

Passing properties: A component can receive props (properties passed by its parent to specify values).

Components are of 2 types-

Stateless Components: JS functions which return HTML and can be contained in a .js or .jsx file

Stateful Class: Regular classes that extend the component class and must contain a render method that returns HTML. They can be contained in a .js and .jsx file.

A jsx file doesnot allow to write multiple html tags like <h1>, <p> without enclosing them in a <div> tag first.

Key feature of a class component is the usage of the render method.

Class and Function name has to be in camel notation

Export and Imports in React.js: Any function, clss, variable can be exported form a file and imported into another file using export and import key words. Export function can export multiple files separately by using ‘export’ in front of the entities such as:

Export class <ClassName1> extends <React.Extension> {

}

Export class <ClassName2> extends <React.Extension> {

}

Or, individually at the end of the script:

Export default <ClassName>

But using the latter approach, only one entity can be exported from one page. As such, it is a wise decision to use the first approach.

Similarly, multiple entities from the same path can be imported using the following syntax:

Import {EntityName1, EntityName2} from <./path>

High Order Component: It is a function that takes one component and returns a new component by facilitating the reusing of component logic

Const NewComponent = higherOrderComponent (OriginalComponents)

React.Component is the base class for React components. React.PureComponent is a variation of the React.Component class and does a shallow comparison of props and states. A react component is called a pure component if it renders the same output for the same state and props

State Object: A state object of a component is the place where the property value of the component is stored. When the state object of a component changes, the component re-renders. setState() is used to schedule an update to the component’s state object.

Props or properties allow users to pass arguments or data to components. It helps make components more dynamic. Props are passed to components in a way similar to that of HTML-tag attributes. Props are read-only which are only sent to the children comp by the parent comp. Children components can not make any change to the props.

{this.props.children} can be used when components do not know about their children ahead of time.

Props are defined in parent components and passed in the child components.

In parent component,

<ClassComp name=”name1” place=”place1”/>

In child component,

<div>Hello child whose name is {this.props.name} who is from {this.props.place}</div>

In a function, multiple properties can be passed similarly as in a child component (class). But in a function child component, ‘props’ need to be used as the function parameter. Such as :

Function Function(props){

return(

<h1>something something called {props.name}</h1>

)}

State: A state is an object that holds the properties of components. A state can be modified based on the user action or network changes. The state object can store multiple properties and with the help of setState() method, the states of a component can be changed. Everytime the state of a component changes, React re-renders to the browser. The state object is initialized in the constructor. setState() method performs a shallow merge between the old and the new state of a component property.

Difference between Props and State:

|  |  |
| --- | --- |
| **Props** | **State** |
| Props are used to pass data and event handlers to its children components | State is used to store the data of the components that has been rendered to the view |
| Props are immutable and cannot be changed | State holds the data and can change over time |
| Props can be used in both functional and class components | State can only be used in class components |
| Props are set by parent component for the children components | State is generally updated by event handlers |

Npm install react-router-dom

npm install @mui/material @emotion/react @emotion/styled

In the portfolio project, the header is a component which will have two parts- parent component (App.js) and a child component (header.js). The header will also have a style sheet (header.css). Apart form the child component, other child components such as the individual pages of the website (contact, downloads, portfolio etc.) can also be created and linked with the main child component (header.js).

**Index.js**

In React, index.js is the routigng js file that is responsible for routing the pages throughout the browsers.

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<BrowserRouter>

<App/>

</BrowserRouter>

);

Import {Link, Routes, Route} from react-route-dom;

<nav>

<ul>

<div class=”closed”>

</div>

<li>

<Link to = ‘/’>Home</Link>

</li>

<li>

<Link to = ‘/downloads’> Downloads</Link>

</li>

</ul>

<Routes>

<Route path=’/’ element={<Home/>}>

</Routes>

</nav>

Here, routes, route, link are components while path and elements are props that’s storing values for the components. With the help of Button component, the nav bar can be customized very well. The button component can be imported from @mui/material/button package using ‘import Button from @mui/material/Button’ command.

Errors:

1. Peer dependency Error: While two or more peer dependencies have conflicting versions then this error rises. React-router-dom and @material-ui/icons conflicted, and the error generated which I solved by uninstalling @material-ui/icons and installing @mui/material: latest version. I also installed react-router as well as react-router-dom.

The MVC framework: The Model-View-Controller framework is a web development framework in which, there are three components- client (Frontend), Server (Backend) and the database. The database is the data store which falls under the model tier while the front end is considered the view. The server or backend is the controller that interacts with both the view and the model.

When a user sends for an object or data, the view sends an API request to the controller, and it retrieves the requested data from the model (database) and sends it back to the client (view). As such, a model and view component of the MVC architecture never interacts directly with each other.

Node.js: Node.js is an open-source cross-platform JavaScript runtime environment and library for building and running web apps across servers. Node.js is extremely fast. Node Package Manager has over 50000 bundles for at the developer’s disposal. It is perfect for data intensive, real-time web app as it never waits for an API to return data (asynchronous- non blocking).

Modules: These are JavaScript libraries that can be used in a node.js app to include a set of functions. These are imported within the require() methods.

Console: It is a module that provides a way to debug like the JS console provided by an internet browser. Ex: console.log(“Hello world”)

Cluster: Node.js is developed upon the concept of single threaded programming. Cluster is a module that allows multi-threading by creating child processes that share the same server port and run simultaneously.

Ex:

Var cluster = require(‘cluster’)

If (cluster.isWorker){

Console.log(“Child thread”)}

Else{

Console.log(Parent thread);

Cluster.fork();

Cluster.fork();

}

Global: Global objects are objects that are available in all node.js modules. Global objects can be functions, models, strings etc. Ex: \_dirname, \_filename, export, require().

Error handling: node.js applications experience 4 types of errors. They are: Standard JS errors, System errors, User-specific errors and assertion errors. Errors in node.js are handled through exceptions.

Try{

Var m=1;

Var n=m/0;}

Catch(err){

}

Streams: Streams are objects that let the user read data or write data continuously. 4 types of streams are readable, writable, duplex and transform.

Express.js framework example:

Var expres = require(‘express’);

Var app=express() ;

App.get(‘/’, function(req, res){

Res.send(“Hello world”)

})

Var port= process.env.PORT || 3000 ;

App.listen(port, function(req, res){

Console.log(‘App is listening to : ‘, port)

})

Here, req and res are API objects that contain the HTTP request and response

Node callback functions are function when they are passed as an argument of another function. The functions passed as arguments is the callback function and when the execution of the containing function is done, the callback function is called.

|  |  |
| --- | --- |
| **Things to do** | **Status** |
| Create email verification for new users while registration. | Done |
| Create recruitment forms (based on existing forms) | Done |
| Confirm password | To be completed by Qisheng Hu |
| Forget password/reset password | Done |
| Password strength check while registration (Parekh) | To be completed by Qisheng Hu |
| Add registration status bar in user registration |  |

**Process of user registration**: First users/students will fill in their recruitment forms (primary key: student ID which will also prove that the person filling the form is a student). If eligible, the admin will send the student an email with a code in the mail asking the user to open an user account in the website using the sign-up option (primary key: userID(try to make std\_id and userID same for simplicity)).

**Email verification process:** In the email verification process, the link will be cehced whether it is a valid and a non-expired link. After that, if the link passes the validation check, the user email verification will start which will compare the hashed uniqueString (generated at the moment when the user sign-up) with the non-hashed uniqueString that the app.get HTTP client req url passed ("/verify/:userID/:uniqueString"). The comparison will be done using bcrypt. The verification link consists of ${currentURL + "/user/verify" + \_id + "/" + uniqueString}

**Types of Users:** For the website, users can be of three types- students, admins, and staffs. While registration, users will have options to select their user type and the registration forms will be presented to them accordingly. The main difference in the form will be the student ID which will be present in the student’s registration form and only they will be able to use email/Student ID to login.

**Errors in node.js:**

*node:internal/errors:465*

*ErrorCaptureStackTrace(err);*

*^*

*Error [ERR\_HTTP\_HEADERS\_SENT]: Cannot set headers after they are sent to the client*

*at new NodeError (node:internal/errors:372:5)*

*at ServerResponse.setHeader (node:\_http\_outgoing:576:11)*

*at ServerResponse.header (E:\Deakin Uni\S779\Trimester 2 (2023)\SIT764\AppAttack\FE\Sehandu-FE-main\Test Webpage\node\_modules\express\lib\response.js:794:10)*

*at ServerResponse.send (E:\Deakin Uni\S779\Trimester 2 (2023)\SIT764\AppAttack\FE\Sehandu-FE-main\Test Webpage\node\_modules\express\lib\response.js:174:12)*

*at ServerResponse.json (E:\Deakin Uni\S779\Trimester 2 (2023)\SIT764\AppAttack\FE\Sehandu-FE-main\Test Webpage\node\_modules\express\lib\response.js:278:15)*

*at E:\Deakin Uni\S779\Trimester 2 (2023)\SIT764\AppAttack\FE\Sehandu-FE-main\Test Webpage\hp.js:245:13*

*at processTicksAndRejections (node:internal/process/task\_queues:96:5) {*

*code: 'ERR\_HTTP\_HEADERS\_SENT'*

*}*

Explanation: The error "Error: Can't set headers after they are sent." means that you're already in the Body or Finished state, but some function tried to set a header or statusCode. When you see this error, try to look for anything that tries to send a header after some of the body has already been written. For example, look for callbacks that are accidentally called twice, or any error that happens after the body is sent.

**Error in nodemailer:**

Err 1: Missing credentials for "PLAIN"

Soln: inside auth.nodemailer.createTransport use ‘user’ and ‘pass’ as key words instead of ‘email’ and ‘password’

Err 2: Invalid login: 535-5.7.8 Username and Password not accepted.

1. Step1: Open this link <https://myaccount.google.com/security>
2. Step2: Enable 2 factor authentication
3. At the end of the 2 factor authentication page, click ‘App Password’
4. From **Select App** options select **Other** and write your app name it could be any name like mycustomapp
5. It will generate you the password copy the password from the popup and use the following code.
6. Use that copied password in the Auth password section my password was this **rkancqhzgvmzsdaqyx**

Err 3: Error: 11660:error:1408F10B:SSL routines:ssl3\_get\_record:wrong version number:c:\ws\deps\openssl\openssl\ssl\record\ssl3\_record.c:332:]

Soln: Use smtp port as 465 (SSL)

**Error with ejs file load in node.js**

Err 1: {

*acknowledged: true,*

*modifiedCount: 1,*

*upsertedId: null,*

*upsertedCount: 0,*

*matchedCount: 1*

*}*

*No default engine was specified and no extension was provided.*

Explanation: This error indicates that no file or format has been identified or specified for the .ejs file to load. This error originated when I was rendering the ‘email-verfied.ejs’ file after the email verification link has been opened and a user has been verified. I solve the error in the following way:

Soln: Add the .ejs file in a separate ‘view’ folder and add the following lines in the server file.

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'ejs');

**Docker deployment**

Deploying all the developed units individually using Docker Deamon and checking the deployments from the browsers using ‘*localhost://<port>:urls’*. Since I have developed the login, registration, recruitment, enquiry, user verification and reset-password components of the website, I have deployed these units from the Docker both individually and collectively.