**CREATING BLOG WITH REACT**

**FRONTEND**

**Install React using: npx create-react-app app\_name**

**edit app.js file by:**

* removing default html element leaving the first div

**Create a folder for pages and page files.**

* create all your pages files with .js extension.
* edit the files by creating an ES6 function and assigning it to a const variable. Within the function, create return (), and within the return, create your html elements.
* at the end of the file outside the div, export the file using the variable name and the keyword **export default variable\_name.**

**Configure routes.**

* Install react router (npm install react-router-dom
* import the **BrowserRouter, Routes, and** Route from **react-router-dom.**
* Wrap entire UI into BrowserRouter and all the pages into Routes.
* Define the Route specifying the path and element.

**Configuring Router Links**

* Create a navbar file.
* Import **Link** from **react-router-dom.**
* In the file, create a function and assign it to a const.
* Within the function, return the html elements.
* Create the nav element.
* Within the element, create an unordered list. Within the ul, create the list tags for each page.
* Within the li element, create the link tag and within the opening link tag, use the **to=”/path”** to specify the path of the link
* Go to the App.js file and import the navbar component.
* Insert the navbar component before the inner div of the pages.

**Configuring URL Parameters with React Router**

* Create a file that contains article contents.
* Update the file with the article content using the **name, title, and content** parameter where the name is the **id**.
* Import the **useParams** react hook from the react-router-dom. The useParams hook lets you access the parameters of the current route.
* Import the name of the variable within the file that has the contents from that file.
* Use the JavaScript **find** to find the articles using the article name as id, and assign it to a const (e.g., article).
* Return the title of the article-content when the article’s name is typed, by using **article.title**.
* The article content is an array of paragraph. In order to get the actual paragraph, return the content by iterating through the paragraphs. Ex.: {article.content.map (paragraph => ( <p>{paragraph}</p>))}
* React cannot return more than one top-level element. To display all the elements, put them all in react fragment (empty tags).

**List Heading of Each Article with Short Content Under Each Heading**

* Open the articleList file.
* Import the articles from article-content.
* In place of the h1 tag, create a react fragment. Within the fragment, create a heading.
* Under the heading, display only a brief portion of each paragraph. To do this, map through each paragraph and create a function called **article** that will display both the paragraph title and the brief portion of the paragraph. To do this, the **article.title** will get the title and display while the **{article.content[0].subsstring(0,150)}**, placed within the p tag will do the magic.

**Link Each List to its actual article.**

* Import Link from react-router-dom within the articleList file.
* Replace the div tags (the div tag that encloses the article hist heading and content) Link tags.
* Define the path.
* Assign the name (article.name) as a key for the link items
* Develop a style to remove the link color from linked items.
* Add key to paragraph using the paragraph index

**Make the Article List Reusable**

* Create a folder called **Component.**
* In the component folder, create a file called **AllArticlesList.**
* Import articles from article-content by simply passing it as props.
* Import Link from react-router-dom.
* Cut the DOM elements from the ArticleListPage file and paste it into the AllArticlesList file.
* Remove the imported link from the ArticlesListPage file and past it in the AllArticlesList.
* Import the AllArticlesList into the ArticlesListPage.

**Preparing the 404 Page**

* Create the variable for the not-found page. It could be called **NotFoundPage.**
* Assign the function to it that will return the “404: Page Not Found!”.

**Develop a code that returns the NotFoundPage whenever a User goes to a Route that is not Defined.**

* Import NotFoundPage in App.js
* Define a route for the NotFoundPage using the **\*** character as path and **NotFoundPage** as element. It tells react to the NotFoundPage whenever any paths other than the defined paths are typed in the browser.

**Do the Above for the Articles but in another way.**

* Open the ArticlePage.
* Import the NotFoundPage.
* Use an if-statement to validate the existence of an article.
* The if-statement should return NotFoundPage if the article does not exist.

**BACKEND DEVELOPMENT OF THE APP**

**Create a New Directory and Setup Environment for the Backend**

* Creates a new directory. --------------- **mkdir backend**
* Open the directory in your ide.
* Initialize it as a new **npm** package ----- **npm init -y**
* Install Express packages -------- **npm install express.**
* Create a new folder within the backend directory that contains the source code for the backend. By convention, it is called the **src** directory.
* Within the src directory, create a file for writing server code. Let the file be known as **server.js**.
* Setup the project in a way that allows modern JavaScript syntax. To do that, open the package.json file and at the end of the first object, add: **“type”: “module”,**.

**Creating Express Server**

* Import the express package from express.
* Create the express App object.
* Create a **put** request that increases the user’s upvotes on an article.
* Run the server with node *server\_name.*
* Use postman to send the request and test the application.

**Automatically Updating with Nodemon**

* Install nodemon into the project. ------- **npm install nodemon --save-dev**
* It should only be used in development not prodection.
* Run the server using this: npx nodemon *server\_name.*
* Make it easier to run the command.
* To do this, open the package.json file, look for the script section, and add dev using the command excluding npx as value.

**Adding Comments**

* Go to the in-built database
* Under the upvotes, add comment with an empty array as it’s value.
* Create the logic for adding comments using the post request.
* In this logic, specify the URL parameter almost the same way you’ve specified it for the adding upvotes except that you’ll use comments in place of upvotes.
* The incoming request body should use json object to specify who’s commenting and the text of the comment.