1. Create new project “ Sebha”
2. Install boostrap : npm i bootstrap

import "bootstrap/dist/css/bootstrap.css";

1. Import in index.js
2. Create new component Sebha
3. Install simple react snippets
4. Import in index.js
5. Start normal component In sebha : Add imrc , cc , state , render , export
6. Add this state

state = {

repeatedText: "Zikr from database",

numberOfRepetition: 0,

colorHeader: "white",

num : 5,

profilePicture: " https://images.pexels.com/photos/3023502/pexels-photo-3023502.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500 ",

Azkar: [

{id: 1, text: " سبحان الله"},

{id: 2, text: " الحمد لله"},

{id: 3, text: " لا إله إلا الله"},

{id: 4, text: " الله أكبر"},

]

}

1. Add profile image by src link from state
2. Style image width height 200 , border radius 100%
3. Add header : Seb7a returned from a function header
4. Style header by : fontSize 40 , class badge badge-primary
5. Add button
6. Add counter badge displaying number of repetition, if zero it display : Start zikr
7. Style counter black Fontsize 35 class badge badge-warning m-2
8. Add button with text “start zikr” and warning class if count is zero

turned to text “continue” and success class if 1 or above

1. onClick : increase counter by number (step) taken from state
2. Create new component Azkar (parent) containing Azkar data

state = {

    Azkar: [

        { id: 1, text: "Zikr 1", source: "source 1", target: 100 },

        { id: 2, text: "Zikr 2", source: "source 2", target: 7 },

        { id: 3, text: "Zikr 3", source: "source 3", target: 30 },

        { id: 4, text: "Zikr 4", source: "source 4", target: 3 },

      ],

  };

1. Create new component Zikr (child) containg table : each row has zikr data
2. Map array of Azkar in parent inside zikr child
3. Pass props from azkar to zikr : id , key , text , source , target
4. NB : you can pass only zikr object and access all properties in child
5. Raise ondelete event in child and handle it in parent to remove zikr
6. Add start button and counter badge to start each zikr individyually
7. Add reset button to reset each zikr individually
8. If list has no items it displays : Create new item
9. Lift the state up of Azkar and sebha to show no of Azkar in sebha component (sync of two component without parent child relationship)
10. Create badge of total no of Azker in sebha component
11. Make **object destructuring** before return inside render (){……….} to avoid repetition of this.state , this.props this:

render() {

*let* {count} = this.state

*const* {text, source, target} = this.props

    return (

1. Add navbar to the project in every page
   1. Create navbar component

Link : <https://getbootstrap.com/docs/4.0/components/navbar>

* 1. Add it to the main rendered component you added to index.js

1. React routing library (npm i [react-router-dom@4.3.1](mailto:react-router-dom@4.3.1)
2. Import {BrowserRouter} from react-router-dom : inside index.js
3. Inside index.js wrap the main component inside <BrowserRouter>
4. Inside the main component(app) where you want to add routing : import {Route} from react- router-dom
5. Create navbar component and put it inside main component

Note that : in main component we put only the components we want to display in every page

1. Separate Sebha and azkar component in new component called my Sebha
2. Take all state and fuctions including render () from app to my sebha
3. Create link in navbar called : My ssebha
4. In Zikr component : add info link which carry count from state and id from props
5. Add route in app receive these data in link
6. Display these data in a new component called azkardetails
7. Redirect wrong links as the following:
   1. Import redirect from react router dom
   2. Add the following:

<*Route* path="/notFound" component={NotFound}></*Route*>

<*Route* path="/" exact component={Home}></*Route*>

<*Redirect* to="/notFound" />

**Steps of routing project (MOSH)**

1. React routing library (npm i [react-router-dom@4.3.1](mailto:react-router-dom@4.3.1)
2. Import {BrowserRouter} from react-router-dom : inside index.js
3. Inside index.js wrap the main component inside <BrowserRouter>
4. Inside the main component where you want to add routing : import {Route} from reacr- router-dom
5. Add four routes below navbar
6. Mechanism of Routing:
   1. Click Hyperlink
   2. Hyperlink navigate to a link
   3. React check the link , if it is identical to one links of routes it will take you there (react match the original with any correct start of routes)

Ex: original “ **/** ” , path in route “ **/newpath** ” : React will consider it **matches**

Ex: original “ **/products** ” , path in route **“ /products/weri** ” : React will consider it **matches**

Ex: original “  **/products** ” , path in route “ **/products0** ” : React will consider it **does not match**

Ex: original “ **/products/p** ” , path in route “ **/products** ” : React will consider it **does not match**

لو الأصل موجود في أي path ولو حتى جزء منه سيعتبره متوافق وسيسعرض الكومبوننت

<*Route* path="/Products" component={Products}></*Route*>

<*Route* path="/Posts" component={Posts}></*Route*>

<*Route* path="/Admin" component={Dashboard}></*Route*>

<*Route* path="/" component={Home }></*Route*>

1. You will notice repetition of Home as its path found in start of the first three paths as react check links from start if it matches any routs it will display its content
2. Avoid the error by adding exact to home

<*Route* path="/Products" component={Products}></*Route*>

<*Route* path="/Posts" component={Posts}></*Route*>

<*Route* path="/Admin" component={Dashboard}></*Route*>

<*Route* path="/"  **exact** component={Home }></*Route*>

1. The second and better solution is to use **SWITCH** : import it from react-router-dom then wrap **ALL** routes inside switch and ordering routes from the most specific to most generic

من الأخص للأعم

Best practice to give HOME PAGE path of **“/ ”** to get back to it if there is no valid link

<*Switch*>

        <*Route* path="/Products" component={Products}></*Route*>

        <*Route* path="/Posts" component={Posts}></*Route*>

        <*Route* path="/Admin" component={Dashboard}></*Route*>

        <*Route* path="/" component={Home }></*Route*>

</*Switch*>

1. To avoid reloading page every time you click a link convert all <a> to <Link> and href to “to” this will make navigation superfast without reloading , this occurs in navbar component where the old links found (don not forget to import link component)

Mechanism of **link to**

1. Add a normal a , href inside the DOM with additional function
2. This function prevents sending http request to the server , instead it only updates the link which will display the specific component accordingly
3. Any route has default props added to it by react
4. Check props which injected automatically in Route (open react extension in browser and try to check it)
5. To Add custom prop to <Products> component inside the Route Component in addition to normal automatic props: you need to pass props and add it , without that only SortBy will be passed

<*Route* path="/Products"

render={(*props*)*=>* <*Products* newProp="its value"{...props}/>}

></*Route*>

1. To pass parameter inside Route you can pass it in path value :

<*Route* path="/Products/**:id**"  component={ProductDetails}></*Route*>

<*Route* path="/Products"render={(*props*) *=>* <*Products* sortBy="newest" {...props} />}></*Route*>

<*Route* path="/Posts/:year?/:month?" component={Posts}></*Route*>

Notice that :

* we Add the more specific one first
* Receive id from the main component “ Product” where all products were mapped from its array

To this point we pass the parameter without receiving it inside the target component ProductDetails

1. In ProductDetails component where we passed the parameter we can retrieve it by

<div>

<h1>Product Details - {this.props.match.params.id} </h1>

<button onClick={this.handleSave}>Save</button>div>

1. In Route of Path : ="/Posts/:year?/:month?" these parameter were made optional by ? so that it will give us an error if the optional parameter is not included inside the link, without this question mark we have to pass all parameters
2. To pass these parameters to posts component you have to pass match from the props in function parameter as it is functional component not a class

*const* Posts = ({*match*}) *=>* {

  return (

    <div>

      <h1>Posts</h1>

      Year: {*match*.params.year} , Month:{*match*.params.month}

    </div>

  );

};

Or pass all props without argument destructuring

*const* Posts = (*props*) *=>* {

  return (

    <div>

      <h1>Posts</h1>

      Year: {*props*.match.params.year} , Month:{*props*.match.params.month}

    </div>

  );

};

Generally speaking we better to avoid optional prameters

1. QUERY STRING needs revision
2. To redirect wrong links we add a new ROUTE with path of not found and choose the component to which it navigates as <NotFound>
3. Add redirecr in the end of Routes with the same path of not found

<*Route* path="/notFound" component={NotFound}></*Route*>

<*Route* path="/" exact component={Home}></*Route*>

<*Redirect* to="/notFound" />

1. We can redirect old page ti new oage as the following :

<*Redirect* from="/messages" to="Posts" />

It will drive us to posts even if messages is not found

1. You can add programatic navigation throught navigation functions executes when you press a button (e.g. : submit a form)
2. We can use two functions : PUSH or REPLACE
3. Push will navigate to the new link with availability to go back to the previous page (having history)
4. handleSave = () *=>* {.props.history.push('/products')  };
5. Replace do the same without ability to go back (no history) : used often in login page

handleSave = () *=>* {.props.history.replace('/products')  };

1. Both PUSH and REPLACE are part of history prop
2. NESTED ROUTES : routes can not only used in app.js but also can be used any where as aregular component (revise nested component video)

Adding new zikr via form :

* + - 1. Add form template with required number of fields.
      2. Receive input of these fields:
         1. Make properties in state corresponding to each input

data: { id: "", text: "", source: "", target: "" }, // receiving new tasks

* + - * 1. Link the value with state.

1. value={this.state.data.text} //bind value of input to state
   * + - 1. Make handle change function which receive the value or (e) and update state to current target value
2. onChange={this.handleChange} //event handler
3. handleChange = (*e*) *=>* {
4. //receive event
5. *let* data = { ...this.state.data }; //store current tasks
6. data[*e*.currentTarget.name] = *e*.currentTarget.value; // bind each inout field with correct vale in state
7. this.setState({ data }); // update the state
8. };
   * + - 1. Store the inputs in one object
         2. Send object to parent component and push it to the array of Azkar
         3. Final components here: NewZikrForm <https://drive.google.com/file/d/1WKlhs0p-sXbRcOPbVlbHXm6JUz-IMWPQ/view?usp=sharing>
         4. MySEbha: <https://drive.google.com/file/d/1RkyeKnc3bo6KFgtcJVnZe0LsQWMtd2v7/view?usp=sharing>

**Login Form MOSH coarse:**

* + - 1. **Create LoginForm component**
      2. **Add route to App.js Leading to this component**
      3. **Add link to navbar**
      4. **Add bootstrap form inside div.container**
      5. **Convert class to className , for to htmlFor**
      6. **Add unique label & id for each input field (htmlFor , id)**
      7. **Handle submit by adding onSubmit {this.handleSubmit()} to from tag**

<form onSubmit={this.handleSubmit}>

* + - 1. **Make handleSubmit prevent reload and check it is working by log**

handleSubmit = (*e*) *=>* {*e*.preventDefault();

    console.log("Submitted");};

* + - 1. **Use autofocus in email field**
      2. **To receive input make the following steps:**
         1. **Create a state in the login form with account object containing all required inputs as empty strings**

state={account : {username:"" , password:""} }

* + - * 1. **Add**

**value in input field depending on state to make the input controlled (this will make the field does not accept any writing**

**onChange = this.handleChange**

**Add name=property**

<label htmlFor="username">User name</label>

              <input

                value={this.state.account.username}

                onChange={this.handleChange}

                name="username"

<label htmlFor="Password">Password</label>

              <input

                value={this.state.account.password}

                onChange={this.handleChange}

                name="password"

**Add handle change in state as the following**

handleChange = (*e*) *=>* {

*const* account = { ...this.state.account };

    account[*e*.currentTarget.name] = //this is to take value dynamically *e*.currentTarget.value;

    this.setState({ account });

  };

**This method of receiving input make change in input field is in sync with state**

**Visualize state by adding two badges taking their values from username and pass word**

**This is the complete code:**

[**https://docs.google.com/document/d/1Fl903G-5PTxSLvAU8F9H\_vbdHoGosb0eeWn8ujDx7SQ/edit?usp=sharing**](https://docs.google.com/document/d/1Fl903G-5PTxSLvAU8F9H_vbdHoGosb0eeWn8ujDx7SQ/edit?usp=sharing)

**We have noticed that we repeat fixed pattern in making input:**

<div className="form-group">

            <label htmlFor="username">username</label>

            <input

              value={this.state.account.username}

              onChange={this.handleChange}

              name="username"

              id="username"

**we can make it a functional component passing props to it and use these props for any input**

**Steps:**

1. **Inside component mak a coomon folder , in common folder make input component. (functional) called FormInput**
2. **Pass props as the following for example:**

import React, { Component } from 'react';

*const* FormInput = (*props*) *=>* {

    return (

        <div className="form-group">

            <label htmlFor={this.props.name}>{this.props.label}</label>

            <input

              value={this.props.name}

              name={this.props.name}

              id={this.props.name}

              onChange={this.props.onChange}

              type="text"

              class="form-control"

              aria-describedby="emailHelp"

              placeholder={`Enter ${this.props.name} `}

              autoFocus

            />

          </div>

     );

}

export default FormInput;

**where you want to add form input use it as the following :**

<*Input*

          name="password"

          value={account.password}

          label="Password"

          onChange={this.handleChange}

          />

**Argument destructuring:**

**We can remove repeated ” this.props. “ as the following:**

import React, { Component } from 'react';

*const* FormInput = ({*label*, *name*, *onChange*}) *=>* {

    return (

        <div className="form-group">

            <label htmlFor={*name*}>{*label*}</label>

            <input

              value={*name*}

              name={*name*}

              id={*name*}

              onChange={*onChange*}

              type="text"

              class="form-control"

              aria-describedby="emailHelp"

              placeholder={`Enter ${*name*} `}

              autoFocus

            />

          </div>

     );

}

export default FormInput;

**Form validation :**

1. **In state add : errors property as empty object :**

**Errors: {}**

1. **Add validate function return object:**

validate = () *=>* {

    return {username: "user name is required."}

// dummy object to return errors

  };

1. **Inside handle submit function Add :**

handleSubmit = (*e*) *=>* {

*e*.preventDefault();

*const* errors = this.validate();

    this.setState({errors});

    if (errors) return; // return only if errors not null

  };

1. **Implement validation as the following :**

validate = () *=>* {

*const* errors = {}; // create errors object

*const* { account } = this.state; // destructuring

    if (account.username.trim() === "")

    errors.username = "Username required"; // validate username + add peoperty to errors object

    if (account.password.trim() === "")

    errors.password = "Password required"; // validate username + add peoperty to errors object

    return *Object*.keys(errors).length === 0 ? null : errors; // return errors object if exsists

  }

  handleSubmit = (*e*) *=>* {

*e*.preventDefault(); // prevent reload

*const* errors = this.validate(); // receive errors object from validate ()

    console.log(errors); // monitor errors

    this.setState({errors}); // update errors object

    if (errors) return; // to stop compiling here if there is errors

    console.log("Submitted"); // Confirm there is no errors

  };

1. **Display error message inside single input component:**

**JOY**

1. **Install : npm i joi-browser@13.4**

**http app :**

1. **Npm i**
2. **Npm i** [**axios@0.18**](mailto:axios@0.18)
3. **Import axios from “axios” in app.js**
4. **Getting data from api:**
   1. **Add an array of object in status to receive data in.**

state = {

    posts: []

  };

* 1. **Add componentDidMount lifecycle hooks to receive data in.**
  2. **Inside it add axios.get(“ url of API ”)**

async componentDidMount() { // any function containing await have to be decorated with async

*const* response = await Axios.get("https://jsonplaceholder.typicode.com/posts"  ); // await means that response will come in the future

    console.log(response.data); // response has a property called data which has the required array of data

    this.setState({posts : response.data}) // here we update our posts to the new one

    // Another simple way by object destructuring

*const* {data:posts} = await Axios.get("https://jsonplaceholder.typicode.com/posts"  );

    console.log(posts);

    this.setState({posts})

  }

1. **Adding data to API:**
   1. **Add handleAdd async method.**

 handleAdd =  async () *=>* {

  };

* 1. **Create const of the new data you want to Add:**

*const* obj = {title: "new title" , body: "new body"}

* 1. **Send the new post to the server using post method with two arguments : url and new object**

Axios.post(apiEndpoint, obj)

* 1. **Receive response from the server : the response wil be ONLY the new post not the complete data**

*const* response = await Axios.post(apiEndpoint, obj)

* 1. **Response contains a data property so we can destructure it and rename to post**

*const* {data : post} = await Axios.post(apiEndpoint, obj)

* 1. **You can monitor the response by logging it**

    console.log(post);

* 1. **Create a new array of posts with the new one in the beginning and setstate to the new array**

*const* posts = [post, ...this.state.posts];

    this.setState({posts})

* 1. **Final Function:**

handleAdd =  async () *=>* {

*const* obj = {title: "new title" , body: "new body"};

*const* {data : post} = await Axios.post(apiEndpoint, obj)

*const* posts = [post, ...this.state.posts];

    this.setState({posts}) };

1. **Updating data :**
   1. **Create handleUpdate method with argument of the obj to be updated**

handleUpdate = *post* *=>* {

  };

* 1. **Update the required property:**

handleUpdate = *post* *=>* {

*post*.title = "Updated";  };

* 1. **Send the entire object to API specific link (link of target object) :**

    Axios.put(apiEndpoint+"/"+*post*.id, *post*);

* 1. **Or send only the updated property:**

    Axios.patch(apiEndpoint+"/"+*post*.id, {title: *post*.title});

* 1. **Receive the response (or data property from response) -which Is ONLY THE NEW UPDATED OBJECT- using await async (log to monitor the response)**

handleUpdate = async *post* *=>* {

*post*.title = "Updated";

*const* {data} = await Axios.put(apiEndpoint+"/"+*post*.id, *post*);

    console.log(data);  };

* 1. **You can skip receiving new object here and directly add the new object to API array**
  2. **Clone the posts array from state, get the index of updated post, update specific object usng index , then set state**

*const* posts = [...this.state.posts ]; //Clone the posts array from state

*const* index = posts.indexOf(*post*); //get the index of updated post

    posts[index] = { ...*post* }; //update specific object usng index

    this.setState({ posts });

* 1. **Final function :**

handleUpdate = async (*post*) *=>* {

*post*.title = "Updated";

*const* { data } = await Axios.put(apiEndpoint + "/" + *post*.id, *post*);

    console.log(data);

*const* posts = [...this.state.posts ]; //Clone the posts array from state

*const* index = posts.indexOf(*post*); //get the index of updated post

    posts[index] = { ...*post* }; //update specific object usng index

    this.setState({ posts });

  };

1. **Deleting as the following :**

handleDelete = (*post*) *=>* {

    Axios.delete(apiEndpoint + "/" + *post*.id);

*const* posts = this.state.posts.filter(*p* *=>* *p*.id !== *post*.id);

    this.setState({posts})

  };

1. **Optimistic delete and update :** 
   1. **The concept is : start with update the UI before calling the server and f any error occur revert the UI to the original state**