step by step project setup

- React Router
- Tailwind
- .env.local: 69-7 (Recap) Create a simple Login page with firebase integration
- eslint file error solve: "react/prop-types","off",
- project build command

```
>mkdir coffee-store-server
>npm init -y
>npm i express cors mongodb dotenv
>nodemon index.js
>In index.js file: require('dotenv').config()
>npm install sweetalert2
```

CRUD Method(Database Integrate)

step by step New database setup

- Mongodb Database connection for new database

```
Go to mongodb atlas site
> Database Access: create username password
> Database > connect >Drivers >copy
```

POST Method

- create POST Method api in backend

```
> app.post("/coffee", async (req, res) => {
  const newCoffee = req.body;
  console.log("new NewCoffee", newCoffee);
  });
```

- Send data from client side to server side using fetch

```
> fetch("http://localhost:5000/coffee", {
    method: "POST",
    headers: {
      "content-type": "application/json",
    },
    body: JSON.stringify(newCoffee),
    })
    .then((res) => res.json())
    .then((data) => {
      console.log(data);
    });
```

- create database & collection name

```
> const database = client.db("usersDB");
  const usersCollection = database.collection("users");

or
> const coffeeCollection=client.db('coffeeDB').collection('coffee');
```

- In POST method API >send server data to Database

```
> const result=await coffeeCollection.insertOne(newCoffee);
   res.send(result)
```

- Give a response to a user from client side

```
>npm install sweetalert2
>import Swal from 'sweetalert2'
>if(data.insertedId){
       Swal.fire({
         title: "Error!",
         text: "Do you want to continue",
         icon: "error",
         confirmButtonText: "Cool",
       });
     }
```

READ Method

- Data READ

```
step-1:get/read data from server site
> app.get('/coffee',async(req,res)=>{
   const cursor=coffeeCollection.find()
   const result=await cursor.toArray()
   res.send(result)
 })
step-2: load data in client side from server link
>path: "/",
 element: <App></App>,
 loader: () => fetch("http://localhost:5000/coffee"),
step-2.1: In the created file use useLoaderData
>const coffees=useLoaderData()
```

DELETE Method

- Data DELETE

```
          send user delete data to backend
 step-1: In users file create delete button and handler
 > <button
          onClick={()=>handleDelete(user._id)}
          >X</button>
 > const handleDelete=_id=>{
  console.log('delete',_id)
  fetch(`http://localhost:5000/users/${_id}`,{
    method:'DELETE',

  })
  .then(res=>res.json())
  .then(data=>{
      console.log(data);
      if(data.deletedCount>0){
        alert('Deleted successfully');
      }
  })
 }
 or, if use sweet alert:
 >const handleDelete=_id=>{
  console.log(_id)
  Swal.fire({
    title: "Are you sure?",
    text: "You won't be able to revert this!",
    icon: "warning",
    showCancelButton: true,
    confirmButtonColor: "#3085d6",
    cancelButtonColor: "#d33",
    confirmButtonText: "Yes, delete it!",
  }).then((result) => {
    if (result.isConfirmed) {
      // Swal.fire("Deleted!", "Your file has been deleted.", "success");
      fetch(`http://localhost:5000/coffee/${_id}`, {
        method: "DELETE",
      })
        .then((res) => res.json())
        .then((data) => {
          console.log(data);
          if (data.deletedCount > 0) {
             Swal.fire("Deleted!", "Your coffee has been deleted.", "success");
          }
        });
    }
  });
}


  Remove data from server side
 step-2: In server file
```

```
> app.delete('/users/:id', async(req,res)=>{
   const id=req.params.id;
   console.log('please delete from database',id);
   const query={_id: new ObjectId(id)}
   const result=await userCollection.deleteOne(query);
   res.send(result)
   })

step-3: Delete without Refresh
> const loadedUsers=useLoaderData();
  const [users,setUsers]=useState(loadedUsers);
> const remaining=users.filter(user=>user._id!==_id);
  setUsers(remaining);
```

Update Data

- Update data

```
step-1: create user data loader api in backend side
 > app.get("/coffee/:id", async (req, res) => {
    const id = req.params.id;
    const query = { _id: new ObjectId(id) };
    const result = await coffeeCollection.findOne(query);
    res.send(result);
    //after write this method check by id in browser
  });

 step-2: create a file and dynamic route
 > {
  path: "/updateCoffee/:id",
  element: <UpdateCoffee></UpdateCoffee>,
  loader: ({ params }) => fetch(`http://localhost:5000/coffee/${params.id}`),
},

 step-3: create a dynamic link button
  > <Link to={`/updateCoffee/${_id}`}>
    <button className="btn">Edit</button>
    </Link>
 step-4: display coffee in update route
  > const coffee=useLoaderData()

 step-4.1:create the UI like post method(if need copy then copy)

 step-5: set default value for update data entry like create value
  > defaultValue={loadedUser?.name} or, > defaultValue={name}

 step-6: Now change the handle  name, object name, method name,update condition & make dynar

  send from client-side> receive server-side> update database>client side>display user
  step-4: client side data send by PUT method like post method
  > fetch(`http://localhost:5000/users/${loadedUser._id}`,{
    method:'PUT',
    headers:{
      'content-type':'application/json'
    },
    body:JSON.stringify(updatedUser)
  })
    .then((res) => res.json())
    .then((data) => {
      console.log(data);
    });

  step-5: Receive backend data by PUT method
  > /* put method update data */
     app.put("/coffee/:id", async (req, res) => {
    const id = req.params.id;
    const updatedCoffee = req.body;
    console.log(updatedCoffee);
```

```
  });
step-6: Database receive :In server file
> const filter={_id: new ObjectId(id)}
   const options={upsert: true}
   const updatedUser={
    $set: {
      name:user.name,
      email:user.email
    }
   }
   const result=await userCollection.updateOne(filter,updatedUser,options);
   res.send(result);


step-5&6: Together
>app.put("/coffee/:id", async (req, res) => {
  const id = req.params.id;
  const updatedCoffee = req.body;
  // console.log(updatedCoffee);
  const filter = { _id: new ObjectId(id) };
  const options = { upsert: true };
  const coffee = {
    $set: {
      // name: user.name,
      // email: user.email,
      name: updatedCoffee.name,
      quantity: updatedCoffee.quantity,
      supplier: updatedCoffee.supplier,
      taste: updatedCoffee.taste,
      category: updatedCoffee.category,
      details: updatedCoffee.details,
      photo: updatedCoffee.photo,
    },
  };
  const result = await coffeeCollection.updateOne(
    filter,
    coffee,
    options
  );
  res.send(result);
 });
```

express.js(Backend)

Reuse:step by step set-up

```
   step-1: create file >mkdir file-name, run command on this file >npm init -y & Express install
   or npm i express or npm  i express cors mongodb dotenv
   step-2: Add in script > "start": "node index.js",


               Go to hello world doc
step-3.1:  create file named same as entry point from package.json file (index.js)
step-3.2: In index.js file import express > const express=require('express') (it used after come
step-3.3: create app using express > const app=express();
step-3.4: create port > const port =process.env.PORT ||5000;  (port)server entry point(a server
step-3.5: get data by root path from server
> app.get('/',(req,res)=>{res.send('Hello from my first ever server')}) it send response when ge
step-3.6: get data by custom path from server
> app.get('/data',(req,res)=>{res.send('Hello from my first ever server')})
step-3.7: check connection with app & port(check the console In which server port is running the
 > app.listen(port,()=>{
   console.log(`My first server is running on port:${port}`)
   })
 This code starts the server and listens on a specified port, printing a message to the console
step-3.8: check by nodemon for watch live updating
    check version >nodemon -v
    start server watch >nodemon index.js

                     middleware setup
              To  Allow access-control-allow-origin
              Need middleware from express>resource
step-4.1: In server folder >npm install cors
step-4.2: In server index.js file import
 > const cors=require('cors')
 > app.use(cors())
 > app.use(express.json()); {
   In a Node.js and Express application, the app.use(express.json()) middleware is used to parse
   incoming JSON data from client requests. When a client sends a request with JSON data in the
   request body, this middleware parses the JSON data and populates the req.body object with the
```

- Additional > if I want to run data from json file

```
step-1: import file in a variable
> const phones=require('./phones.json');
step-2: Get data by server path
> app.get('/phones',(req,res)=>{
res.send(phones);
})

                Now to get id data from json file
step-1: create server path for dynamic id  or specific id news
> app.get('/phones/:id',(req,res)=>{
  const id=parseInt(req.params.id);
  console.log('I need data for id:',id);
  const phone=phones.find(phone=>phone.id===id) ||{};
  res.send(phone);
})

or,
app.get('/news/:id',(req,res)=>{
const id=req.params.id;
console.log(id);
const selectedNews=news.find(n=>n._id===id);
res.send(selectedNews)
})

or,
pp.get('/categories/:id',(req,res)=>{
const id=parseInt(req.params.id);
console.log(id)
if(id===0){
res.send(news)
}
else{
const categoryNews=news.filter(n=>parseInt(n.category_id)===id)
res.send(categoryNews);
}
})
```

- connect api from created server if there had no route

```
step-1: fetch by state
>const [categories,setCategories]=useState([]);
 useEffect(()=>{
 fetch("https://the-news-dragon-server-bariulmunshi.vercel.app/categories")
.then(res=>res.json())
.then(data=>setCategories(data))
.catch(error=>console.error(error))
 },[])
step-2: check length
```

- connect sever api with client side if there had route

```
 create component for fetch data use
           for fetch all data
step-1: create route for component
> path: "/phones",
 element: <Phones />,
 loader:()=>fetch('https://the-news-dragon-server-bariulmunshi.vercel.app/phones')

step-2: load data in created component
> const phones=useLoaderData();
> <div>
 <h2>all phones here:{phones.length}</h2>
{
 phones.map(phone=><li key= {phone.id}>
   <Link to={`/phone/${phone.id}`}>{phone.name}</Link></li>)
}
</div>

           for fetch individual data
step-1: create dynamic route for component
> {
 path:'/phone/:id',
 element:<Phone></Phone>,
 loader:({params})=>fetch(`https://the-news-dragon-server-bariulmunshi.vercel.app/phones/$
}
step-2: load individual data by dynamic id
> const phone =useLoaderData();
> <div>
<h2>{phone.name}</h2>
<img src={phone.image} alt="" />
</div>
```

step-5: add file >.gitignore write > node_modules & check it by git init

step-8: create a post api on the server side

```
app.post('/users',(req,res)=>{
console.log('Post API hitting')
console.log(req.body);
const newUser=req.body;
newUser.id=users.length+1;
users.push(newUser);
res.send(newUser);
})
```

DirectUse: express.js set up(Backend)

```
step-1: > npm init -y
step-2: > npm  i express cors mongodb dotenv
step-3: in package.json file create >"start": "node index.js",
step-4: create file > index.js
step-5: In index.js file
>const express = require('express');
 const cors = require('cors');
 const app=express();
 const port=process.env.PORT || 5000;

 /* middleware */
 app.use(cors());
 app.use(express.json());

 /* check root path */
 app.get('/',(req,res)=>{
   res.send('Simple crud running')
 })

 /* check running server port */

 app.listen(port,()=>{
   console.log(`Simple crud is running:${port}`);
 })

step-6: import code from atlas dbms
step-7: set password carefully and see the server is pinged or not
step-8: Now go to for > set up client side
step-9: send server data to mongodb
>  const database = client.db("usersDB");
   const usersCollection = database.collection("users");
   const result = await usersCollection.insertOne(user);
   res.send(result);
```

Firebase(React Authentication)

setup firebase in project

1. create firebase project & create a web app
2. npm install firebase & save firebase config and export app
3. Build >Authentication >Get started >Enable SignIn method

4. Complete the sign up & Login form
5. Add onSubmit handler for collect form data

```
onSubmit={handleSignUp}

const handleSignUp=event=>{
  event.preventDefault()
  const form=event.target
  const email=form.email.value
  const password=form.password.value
  const confirmPassword=form.confirm.value
  console.log(email,password,confirmPassword)
  }
```

6. validation form data

```
  step-1: declare state
  const [error, setError] = useState("");

  step-2:Add condition for validation
  if(password!==confirmPassword){
  setError('Your password did not match')
  return
}
  else if (!/(?=.*[A-Z])/.test(password)) {
    setError("Please Add at least one uppercase");
    return;
  } else if (!/(?=.*[A-Z].*[A-Z])/.test(password)) {
    setError("Please add at least two numbers");
    return;
  } else if (password.length < 6) {
    setError("Please add at least 6 character in your password");
    return;
  }

  step-3: Display the catch error if exist
  <p className="text-error">{error}</p>
```

7. {context Api/redux(redux-toolkit)/database} for share form authentication information in every route

   here for context api:

   step-1: create context Provider file(AuthProvider)

   step-2: createContext with export & set context value with children props

```
export const AuthContext=createContext(null);

const AuthProvider = ({children}) => {
const user={displayName:'Bariul'}
 const authInfo={
     user
 }
return (
<AuthContext.Provider value={authInfo}>
 {children}
</AuthContext.Provider>
);};
```

8. set the AuthProvider path

```
 ReactDOM.createRoot(document.getElementById("root")).render(
 <React.StrictMode>
    <AuthProvider>
      <RouterProvider router={router} />
    </AuthProvider>
 </React.StrictMode>
 );
```

9. Now createContext as useContext For use Another component

1. const {user,createUser}=useContext(AuthContext)
1. const {user,signIn}=useContext(AuthContext)
2. check: console.log(user);
2. check: console.log(signIn,createUser);
3. call the function in eventHandler function: createUser(email,password)
3. call the function in eventHandler function: signIn(email,password)

10. Now set Auth in AuthProvider file from firebase authentication doc

```
import { getAuth } from "firebase/auth";
const auth = getAuth(app);
import app from './../../firebase/firebase.config';
```

11. set auth user value to useState

```
const [user,setUser]=useState(null)
```

12. for register

```
step-1: In AuthProvider file
const createUser=(email,password)=>{
  return createUserWithEmailAndPassword(auth,email,password); //its firebase function
}

 step-2:set createUser in context object

 step-3: In register/signUp file: call the function in eventHandler function:

  {before call follow step-9:}

   createUser(email,password)
  .then(result=>{
    const loggedUser=result.user;
    console.log(loggedUser);
  })
  .catch(error=>{
    console.log(error)
  })
```

13. For reset Error

```
  setError("") //call it before validation
```

14. for Login/signIn

```
step-1: In AuthProvider file
const signIn=(email,password)=>{
  return signInWithEmailAndPassword(auth,email,password); //its firebase function
}

 step-2:set signIn in context object

 step-3: In signIn/Login file: call the function in eventHandler function:

  {before call follow step-9:}

   signIn(email,password)
  .then(result=>{
    const loggedUser=result.user;
    console.log(loggedUser);
    form.reset()
  })
  .catch(error=>{
    console.log(error)
  })
```

15. For logOut: we will use logout in header file

```
step-1: In AuthProvider file
const logOut=()=>{
return signOut(auth)
}
step-2:set logOut in context object
step-3:In header file: call the function in eventHandler button:
{before call follow step-9:}

{user && <span>Welcome{user.email} <button onClick={handleLogOut}>Sing Out</button> </spa
```

Here Just set up firebase sign Up, Login & LogOut using Context API from one file & step-11 still null

16. For set value in step-11 user need call outside api by useEffect
    what's the purpose of onAuthStateChanged in Firebase authentication?
    Answer: It listens for changes in the user authentication state.
    What's the work of unsubscribe? Answer: catch the changes

```
  /* observer user auth state */
useEffect(()=>{
 const unsubscribe= onAuthStateChanged(auth,currentUser=>{
   setUser(currentUser)
  })
  /* stop observing while unmounting */
  return ()=>{
    return unsubscribe()
  }
},[])
```

17. Private Route & Navigate after Login

```
  step-1: create file
  const PrivateRoute = ({children}) => {
  const {user}=useContext(AuthContext);
  if(user){
    return children;
  }
  return <Navigate to='/login' replace={true}></Navigate>;
};
step-2: do private which route want to private
 <PrivateRoute>
        <CheckOut></CheckOut>
</PrivateRoute>
```

18. For ignore Re-loading issue

```
  step-1: In AuthProvider file create a useState
  const [loading,setLoading]=useState(true)

  step-2: In AuthProvider file set setLoading(true) in createUser , signIn & logOut
  > setLoading(true)

  step-3: If state gonna change then call setLoading in useEffect
  > setLoading(false)

  step-4: For use it now call it in context
  loading,

  step-5:use it PrivateRoute
  const {user,loading}=useContext(AuthContext)
  if(loading){
  return <progress className="progress w-56"></progress>;
}
```

19. After Login where I want to go

```
    useNavigate from react Router dom
step-1: In login file set useNavigate state
const navigate=useNavigate()

step-2: In Login file call it in below of signIn(email,password) & within   .then(result=>
> navigate('/')
```

20. After Login Redirect Navigate to the right route

```
step-1: In login & PrivateRoute both file set
  > const location=useLocation()
  check location: console.log(location)

step-2:In PrivateRoute set state={{from: location}} replace in Navigate
return<Navigate to="/login" state={{from: location}} replace></Navigate>

step-3: In login file
const from=location.state?.from?.pathname || '/'

step-4: In login file call it in signIn/login function
 navigate(from,{replace:true})
```

21. Host your react app to firebase and Show password

```
just one time need install in pc
step-1: npm install -g firebase-tools
step-2: firebase login

for each project one time
 * HOSTING
  * --------------------
  * One time per PC
  * 1. npm install -g firebase-tools
  * 2. firebase login
  *
  * For each project one time
  * 1. firebase init
  * 2. proceed
  * 3. hosting: firebase (up and down arrow) use space bar to select
  * 4. existing project
  * 5. select the project careful
  * 6. which project as public directory: dist
  * 7. single page application: yes
  * 8. continuous deployment: no
  *
  * For every time deploy
  * 1. npm run build
  * 2. firebase deploy
```

22. show password

```
step-1: set state
  const [show,setShow]=useState(false)

step-2: set onClick button
<p onClick={()=>setShow(!show)}><small>
      {
        show? <span>Hide Password</span>:<span>Show password</span>
      }
      </small></p>

step-3: set input type with ternary condition
  type={show? "text": "password"}
```

23. Accept Terms and conditions

```
step-1: create a component >set Route >In route set a link
> <p>Go back to <Link to="/register">Register</Link></p>

step-2: In register file create a checkbox with onClick handler
> <Form.Group className="mb-3" controlId="formBasicCheckbox">
      <Form.Check
      onClick={handleAccepted}
      type="checkbox"
      name='a'
      label={<>Accept<Link to="/terms">Terms & condition</Link></>} />
    </Form.Group>

step-3: Declare a state in register file
> const [accepted,setAccepted]=useState(false);

step-4: add function & call for checked
>const handleAccepted=event=>{
//console.log(event.target.checked)
setAccepted(event.target.checked);
}

step-5: set button disable if not accept term & condition
> <Button variant="primary" disabled={!accepted} type="submit">
      Register
    </Button>
```

24. firebase setup

```
step-1: firebase init
step-2: npm run build
step-3: firebase deploy
```

VS code set up

- Word wrap
- cursor expand
- Prettier - Code formatter
- formatter: format on save ,prettier formatter
- vs code font family
- Mouse Wheel zoom
- mini map
- Material icon theme
- Path Intelligence
- Markdown Preview Enhanced

- Image preview
- Markdown Preview Enhanced
- Live server
- code runner
- Code Spell Check
- Tailwind CSS intelligence
- Learn with sumit
- Terminal set up
- React Extension Pack
- ES7+ React/Redux/React-Native snippets
- React Native Tools