**Firebase + User Authentication**

**Firebase: backend application development software**

Access backend features without developing backend

For authentication and database

**Step: 1: Create a firebase project**

Go to ‘https://**firebase.google.com**/’

Sign-in into google account

Click on ‘**Go to console’** button at the top

Click create project

Give project name and create a project

Click continue and it shows firebase Dashboard

**Step: 2: Configure firebase**

Click project overview tab - > click the icon web </>

Type nickname for app (mostly internal)

Click register app (no need for firebase hosting)

Copy the firebaseConfig object code starting from {}

Got to our terminal (our project) – yarn add firebase

**Api key** is there in the object (may be it should not be shown in open)

**Step: 4: Add google authentication details into our app**

**App.js**

**Import {auth} from ‘./firebase/firebase.utils’;**

**this.state{ currentUser : null }**

**unsubscribeFromAuth = null;**

**componentDidMount(){**

this.unsubscribeFromAuth = auth.onAuthStateChanged(user =>{

this.setState({currentUser:user})// auth obj is returned

}) // console.log(user); many user details will be displayed

When ever somebody signin and signsout the we should be aware of the change

}

**componentWillUnmount(){**

**this.unsubscribeFromAuth();**

**}**

onAuthStateChanged – is a method on auth lib from firebase . this connection is always open .

since it is open subscription , it should be unsubscribed .

*When we call atuh.onAuthStateChanged, that function actually returns us a NEW function that will close the subscription. What's happening here is that when we pass our callback to auth.onAuthStateChanged, we are instantiating a new listener to auth state changes with our function that our auth code will manage. Whenever we want to stop listening though, we need to call that function that we got back in order for us to close that listener! This is what's called the observer pattern and is central to the use of observables (which we'll go into detail about in the observer pattern lesson later on in the course).*

*So if you think back to our firebase config, that lets our firebase know that our application is ours and it needs the specific database that we set up, all the while it will set up an instance in firebase to let firebase know this application is being run from this client. Any subscription channels that we open, will be locked to this instance as long as the instance is alive. These channels include any auth.onAuthStateChanged subscriptions we instantiate inside of this instance of the application that we have live. This means that if you had 100 users on 100 different computers, each one of them is running their own instance of firebase that firebase is keeping track of. It's definitely abstracted away from us and not something we need to worry about. When you unsubscribe from the auth using the unsubscribeFromAuth function, you are not signing out the user, you are just saying that you don't want to listen for any more auth state change events that fire with the function that you provided to auth.onAuthStatechanged . So the auth state will still change, and firebase will still fire off auth objects whenever it happens to any listeners that are listening to these changes, but it won't fire to the one we just unsubscribed with using our unsubscribeFromAuth method*

**Step: 3: setup firebase in our project**

Create new directory in src ( in our project) – name it firebase

Create a new file inside firebase folder – firebase.utils.js

inside firebase.utils.js – import firebase from ‘firebase/app’

// pulling in the firebase utility library(already added- yarn add firebase)

// for the database import ‘firebase/firestore’

// for the auth import ‘firebase/auth’

//these are automatically attached to the firebase keyword

create **const config** ={

copy and paster the object (api object of firebase) ( that can be copied from ***projectoverview->project->settings icon -> sdk code snippet)***}

**firebase.iniitalizeApp(config);**

**export const auth = firebase.auth(); //anything related with authentication**

**export const firestore = firebase.firestore();**

**Google authentication utility**

**const provider = new firebase.auth.GoogleAuthProvider();**

**// gives access to new googleAuthProvider class from authentication library**

**provider.setCustomParameters({prompt:’select\_account’});**

**// always trigger the google popup whenever we use this google auth provider for authentication and sign in**

**export const signInWithGoogle = () => auth.signInWithPopup(provider)**

**// many different pop ups are available like twitter etc, so to use singin pop up pass the provider as parameter for that**

**export default firebase;**





**\* {**

**box-sizing: border-box;**

// universal selector

**}**

**Sign out**

**App.js**

<Header **currentUser={this.state.currentUser**}/>

**Header.component.js**

import { auth } from '../../firebase/firebase.utils'; // import auth to access signOut function from firebase

const Header = ({**currentUser})** =>(

{ ***currentUser ?***

***<div className="option" onClick={()=>auth.signOut()}> sign out </div>***

***: // functional call back / anonymous function -> firebase function***

***<Link className='option' to="/signin">signin</Link>*** }

**Google signin button modification**

**Custom-button.component.jsx**

const CustomButton =({children,**isGoogleSignIn,**...otherButtonProps}) => (

<button className**={`${isGoogleSignIn?'google-sign-in':''}** custom-button`} {...otherButtonProps} >

{children}

</button> )

**Sign-in.component.jsx**

<CustomButton onClick={signInWithGoogle} **isGoogleSignIn**>Sign In with Google</CustomButton> **// isGoogleSignIn will be true if no value is passed**

**Email and password sign-in:**

**Firebase firestore**

Go to your project🡪 **develop** 🡪 select **database** 🡪**create database** 🡪 **start in test mode(**less secured) 🡪click next

You can change the rules setting to have restrictions on your data

It is nosql database – it is a single **gigantic json object**

Collection – group of objects ; it has documents

It can have collection as properties

**DB -> collection of items/objects 🡪 that has documents; and documents can have collections as its properties**

Example : collection -> users -> docid ->doc: msk +cart(collection) - > doc id -> doc:

To retrieve data :

Import firebase from ‘fireabase/app’;

Import ‘firebase/firestore’;

**Const firestore = firebase.firestore();**

**firestore.collection(‘users’).doc(‘docid’).collection(‘cartitems’).doc(‘docid’); (or)**

**firestore.doc(‘/users/docid/cartitems/docid’)** // **to retrieve docs**

**firestore.collection(‘/users/docid/cartitems’)** // **to retrieve collection**

**Storing user data into firebase database**

We have to Store google authenticated user in to our db . so fart the authenticated user is got from auth library**(google auth object**) .**Google sign in returns authentication object** that has lot of properties in it (ex uid - important)

***uid -> represents the dynamically generated id string that google made for us when we authenticated the user using google signin .***

firebase -> develop -> atuthentication -> when ever user signs in to our application using the auth library we have , regardless of its email sign in or google sign in we will get this user which has uid . we have to take this *user auth object figure out the properties that we want and put it in our db* .

**App.js**

import {auth,createUserProfileDocument} from './firebase/firebase.utils';

componentDidMount(){

this.unsubscribeFromAuth = auth.onAuthStateChanged(**async** user =>{ **//async because of api request**

**createUserProfileDocument(user); // going to interact with firestore**

console.log(user);

} ) }

**firebase.utils.js**

**export const createUserProfileDocument = async(userAuth,additionalData) =>{**

//asynchronous , because we are making an api request . userAuth – obj that we get from auth lib while loggin in.

**if(!userAuth) return;** // when user signs out we get back null as data . Checking whether we are getting back actual object . if it doesnot have a valid obj we are going to return from the function , it should not do anything.if its not null

//if it exits ,query inside firestore to see if it already exists

on querying firestore -> the firestore library gives us back object either 1. QueryReference or 2.QuerySnapshot

query - > asking firestore for collection or doc

firestore will give us the objects, even if there is no data , from the object we determine whether or not any data there

**queryReference obj** 🡪 object that represents the ***current place*** in the database that we are querying .it does not have the actual data of the collection or document. It instead has properties that tell us details about it or the method to get the Snapshot object which gives the data we are looking for

**querySnapshot obj** 🡪***(data)***we get the snapshotObject from the referenceObject using the .get() methods. i.e . **documentRef.get() or collectionRef.get()** .use it for us to determine whether there is any data

**documentRef** returns a **documentSnapshot** object

**collectionRef** returns a **querySnapshot** object

(queryReference)DocumentReference 🡪parent is a collection reference that this document is in

**We use documentRef objects** **to perform our CRUD methods(creae,retrieve,update and delete**). The **documentRef methods are .set() , .get() ,.update() and .delete() .**

.get is pulling out snapshot object ; //string interpolation / back ticks

**const userRef = firestore.doc(`users/${userAuth.uid}`);** ; we get back the user reference at that location

**const snapShot = await userRef.get();** // async function , so use await . get the snapshot of userreference

snap shot has exists : property , it tells us wherther there is any data or not ‘true’ or ‘false’ .

id : id of the document

if(!snapShot.exists){ // if it doesnot exist we have to create the data // to create use the documentRef obj

const {displayName, email} = userAuth;

const createdAt = new Date(); // creating a new js date object

try{

await userRef.set({ // async request to database to actually store data . **.set** is the create method

displayName, // pass an object to the .set method

email,

createdAt,

...additionalData

})

}catch(error){

console.log('error creating user',error.message);

} }

return userRef; // return userRef for other uses . so that’s why returning the user reference

} **// now the firebase stores our data .auth obj continuously passes the data but still multiple copies are not made because our code checks whether snapshot at this location exists or not**

**Storing user data in our App** // to use it in our state

**App.js**

componentDidMount(){

this.unsubscribeFromAuth = auth.onAuthStateChanged(**async** userAuth=>{

if(userAuth) { **// if userAuth exists**

const userRef = await createUserProfileDocument(userAuth);

userRef.onSnapshot(snapshot => { //check whether snapshot or database has updated . when this.setState({ // the code runs it sends the snapshot/user data

currentUser:{ // update current user

id:snapshot.id,

**…snapshot.data();** **//get the actual properties on object by using .data() method (returns JSON obj of document)**

}},() =>{

// Console.log(this.state) // optional to check data//function as Second parameter to setState

}) ;

}

else{ this.setState({currentUser:userAuth}); // when user logs out set the current use to null

});}

handleChange = event =>{

const {name,value} =event.target;

this.setState({[name]:value});

}

render(){

const {displayName,email,password,confirmPassword} = this.state;

return(

<div className="sign-up">

**<h2 className="title">I do not have an account</h2>**

**<span>Sign up with your email and password</span>**

<form className="sign-up-form" onSubmit={this.handleSubmit}>

<FormInput

type='text'

name='displayName'

value={displayName}

onChange={this.handleChange}

label='DisplayName'

required

/>

//Similarly for email ,password and confirm password

<CustomButton type="submit">Sign-Up</CustomButton>

</form>

</div>

); } }

export default SignUP;

**Sign-up Component .jsx**

import FormInput from '../forminput/form-input.component';

import CustomButton from '../custombutton/custom-button.component';

import './sign-up.style.scss';

**import {auth,createUserProfileDocument} from '../../firebase/firebase.utils';**

class SignUp extends React.Component{

constructor(){

super();

this.state={

displayName:'',

email:'',

password:'',

confirmPassword:'',

} }

handleSubmit = **async event =>**{

event.preventDefault(); // prevent default form submit

const {displayName,email,password,confirmPassword} = this.state;

**if(password !== confirmPassword){**

**alert('passwords dont match');**

**return ;**// we don’t want to anything else unless they fix passwords

**}**

**try{**

**const {user} = await auth.createUserWithEmailAndPassword(email,password); //this Auth method creates a new user account . it gives us back user auth obj . that’s why destructure to store**

**createUserProfileDocument(user,{displayName});**

this.setState({

displayName:'', // reset the state to clear out the form

email:'',

password:'',

confirmPassword:'',

})

}catch(error){

console.log('error creating user profile',error)

} }

**Signup with email and password**

***Firebase*** - > develop -> authentication 🡪 signin method 🡪

Edit 🡪 **enable email / password**

To work

**Enable - > google and email/password to work**

Sign-in-and-sign-up.component.jsx

import SignUp from '../../components/sign-up/sign-up.component';

const SignInAndSignUp = ()=> (

<div className="sign-in-and-sign-up">

<SignIn />

**<SignUp />**

</div> );

**Sign-In with email and password**

**Sign-in.component.jsx**

import { auth ,signInWithGoogle } from '../../firebase/firebase.utils';

handleSubmit = async event =>{

event.preventDefault();

const {email,password} = this.state;**// to sign in with email and password using auth lib method**

**try{ const {user} =await auth.signInWithEmailAndPassword(email,password);**

createUserProfileDocument(user);

this.setState({ email:'', password:'', })

}catch(error){

console.log('error creating user profile',error)

} }