UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

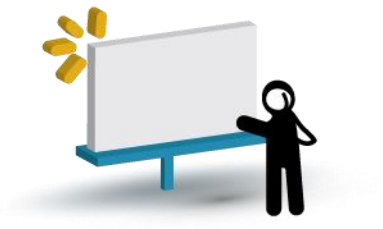
Desarrollo en React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo III

Eventos, Firebase

Unidad 2: Firebase parte 2



Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación:

En esta unidad aprenderemos como insertar, leer, actualizar y eliminar datos en nuestra base de datos de firebase.

Objetivos:

Que los participantes*:

- Aprendan a insertar, leer, eliminar y actualizar datos en firebase
- Analicen cambios a realizar en la estructura de la aplicación

Bloques temáticos*:

- Organicemos nuestro código
- Autenticación con Firebase
- Leer, Crear, Actualizar y Eliminar Documentos

Consignas para el aprendizaje colaborativo

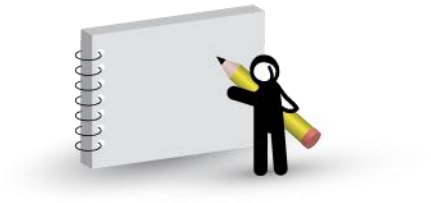
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



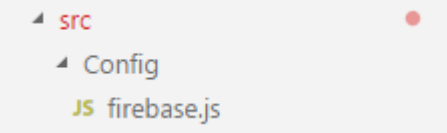
Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Organicemos nuestro código

Crear una carpeta **Config** dentro de src y dentro de esa carpeta crear un archivo **firebase.js**



En el archivo **firebase.js** incluimos el código que teníamos en app.js, quedando de la siguiente manera:

```
import  firebase from 'firebase'

// Initialize Firebase
var firebaseConfig = {
  apiKey: "AIzaSyDTBifG9hGKRqTfnY9TWx0iJv_9jHpop_0",
  authDomain: "react-74ab8.firebaseapp.com",
  databaseURL: "https://react-74ab8.firebaseio.com",
  projectId: "react-74ab8",
  storageBucket: "react-74ab8.appspot.com",
  messagingSenderId: "306351405488"
};
firebase.initializeApp(firebaseConfig);
firebase.auth = firebase.auth();
firebase.db=firebase.firestore();
export default firebase;
```

firebaseConfig es el código provisto por firebase para la conexión.

En este caso creamos un modulo propio el cual luego exportamos, el mismo contiene la siguiente configuración:

- Auth: Resultado de **firebase.auth(),** hace referencia a la utilización del modulo de autenticación de firebase
- Db: Resultado de **firebase.firestore(),** hacer referencia a la utilización de la base de datos cloud firestore.

Al generar nuestro propio modulo podemos importar el mismo en el componente deseado y comenzamos a utilizar ambos módulos.

Autenticación con Firebase

Armemos un formulario de registro

Vamos a crear un componente **Registro**, en el colocaremos un formulario cuyos datos estén bindeados con el componente.

Primero debemos importar el modulo de firebase previamente creado de la siguiente manera:

```
import firebase from "../Config/firebase"
```

Luego generamos el formulario en el JSX del método:

```
return(
    <div>
        <h1>Registro</h1>
        <form onSubmit={handleSubmit}>
            <div>
                <label>Nombre</label>
                <input type="text" name="nombre" value={form.nombre} onChange={handleChange}></input>
            </div>
            <div>
                <label>Apellido</label>
                <input type="text" name="apellido" value={form.apellido} onChange={handleChange}></input>
            </div>
            <div>
                <label>Email</label>
                <input type="email" name="email" value={form.email} onChange={handleChange}></input>
            </div>
            <div>
                <label>Contraseña</label>
                <input type="password" name="password" value={form.password} onChange={handleChange}></input>
            </div>
            <button type="submit">Registrarse</button>
        </form>

    </div>

)
```

Vemos que cada elemento hace referencia en su value a un hook form.nombre, form.apellido, form.email, form.password

Dicho hook esta definido de la siguiente manera dentro de la función:

```
const [form, setForm] = useState({nombre:'',apellido:'',email:'',password:''});
```

Por lo cual el value de cada elemento del formulario quedara asociado a dicho hook inicializándose en vacío.

Para poder capturar los valores colocados por el usuario en el formulario debemos capturar el change en cada elemento, es por ello que vemos la asociación al evento onchange y al hacerlo se ejecuta la función handleChange:

```
const handleChange = (e)=>{

  const target = e.target;
  const value = target.value
  const name = target.name;


  setForm({
     ...form,
     [name] : value});

}
```

Por ultimo al form le asociamos el método onsubmit con la función handleSubmit (ver en el código jsx el onsubmit declarado en el elemento form):

```
const handleSubmit = (e)=>{
    e.preventDefault();
    let email=form.email;
    let password=form.password;
    firebase.auth.createUserWithEmailAndPassword(email, password)
    .then((data)=>{
        console.log("Usuario creado",data.user.uid)

    })
    .catch((error)=>{
        console.log("Error",error)
        setSpinner(false);
    })

}
```

Dentro del handleSubmit hacemos el llamado a **firebase.auth.createUserWithEmailAndPassword**

Este método recibe email y password y genera un nuevo usuario en el modulo de autenticación de firebase.

El método retorna un promise el cual podemos tratar con el .then (como se visualiza en la imagen) o bien con un async / await

Al ver la respuesta del console.log que esta dentro del .then vemos que el método realiza validaciones:

Email ya dado de alta

```
Error
▼ t {code: "auth/email-already-in-use", message: "The email address is already in use by another account.", a: null}
    a: null
    code: "auth/email-already-in-use"
    message: "The email address is already in use by another account."
  ▶ __proto__: Error
```

Contraseña invalida

```
Error ▼ t {code: "auth/weak-password", message: "Password should be at least 6 characters", a: null}
          a: null
          code: "auth/weak-password"
          message: "Password should be at least 6 characters"
        ▶ __proto__: Error
```

En caso de que el registro sea exitoso nos retorna la información del usuario, si accedemos a **data.user.uid** obtendremos el id del usuario creado:

```
Usuario creado 1xSMCaNloWMhgT9ktDWDVEhdufz2
```

El modulo de autenticación nos permite generar un usuario con email y password pero no nos permite asociar mas datos al mismo. Entonces podemos combinar dicho modulo con la base de datos y almacenar el resto de los datos en la misma:

```
let email=form.email;
let password=form.password;
firebase.auth.createUserWithEmailAndPassword(email, password)
.then((data)=>{
    console.log("Usuario creado",data.user.uid)
    firebase.db.collection("usuarios").add({
        nombre: form.nombre,
        apellido: form.apellido,
        email: form.email,
        userId: data.user.uid
      })
      .then((data)=>{
          setSpinner(false);
        console.log(data)
        //history.push("/login");
      })
      .catch((err)=>{
        console.log(err)
        setSpinner(false);
      })
})
.catch((error)=>{
    console.log("Error",error)
    setSpinner(false);
})
```

Vemos que realizamos el llamado a **firebase.db.collection(“usuarios”).add({..})**

Este método crea un nuevo documento dentro de la colección usuarios. Entonces collection recibe como parámetro la collecion en la cual queremos trabajar y add genera un nuevo documento según la estructura que le enviemos como parámetro.

Armemos un formulario de login

Para armar el formulario de login debemos seguir los mismos pasos que en el registro:

- Import de firebase
- Armar el JSX del mismo
- Creación del hook form
- Asociar a cada elemento del formulario en su value el hook correspondiente y capturar las modificaciones mediante el onchange

Lo que difiere respecto del formulario de registro es el método handleSubmit, el cual realizaremos lo siguiente:

```
const handleSubmit = (e)=>{
    e.preventDefault();
    let email=form.email;
    let password=form.password;
    firebase.auth.signInWithEmailAndPassword(email, password)
    .then((data)=>{
        console.log("Usuario logueado",data)
        history.push("/")
    })
    .catch((error)=>{
        console.log("Error",error)

    })

}
```

El llamado que realizaremos es a **firebase.auth.signInWithEmailAndPassword**

Pasaremos el email y password, en caso de salir por el then entonces el login del usuario fue correcto (corresponde el email y contraseña ingresado)

En caso de salir por el catch entonces el email y password introducidos no se corresponden, por lo cual informaremos dicho error al usuario.

Vemos también la utilización de **history.push("/"),** en caso de que el ingreso sea correcto entonces redirigimos al usuario a la pagina de inicio.

Para que este funcione debemos importarlo previamente:

```
import { useHistory } from "react-router-dom";
```

Dentro de la declaración de la función:

```
function Login(){

    const history = useHistory();
```

Leer, Crear, Actualizar y Eliminar Documentos

Leer documentos de una colección

Mediante el método **firebase.db.collection("nombre_coleccion").get**() obtendremos todos los documentos de una colección

Este método retorna un promise el cual debemos tratar con un .then o un async/await

Ejemplo, desde la home realizamos una consulta a firebase para traer los productos y mostrarlos en pantalla:

Importamos firebase en el componente

```
import firebase from "../Config/firebase"
```

Luego en el componente consultas con el método mencionado y mostramos los productos en el render del componente:

```
class Home extends Component{
    constructor(){
        super();
        this.state={
            productos:[],
            loading:true
        }
    }
    componentDidMount(){

        firebase.db.collection("productos")
        .get()
        .then(querySnapshot=>{
            this.setState({
                productos:querySnapshot.docs,
                loading:false
            })
        })
    }
    render(){
        if(this.state.loading){
            return(
                <Container>
                    <div style={{position:"fixed",top:"50%",left:"50%"}}>
                            <Spinner  animation="grow" />
                            <Spinner  animation="grow" />
                            <Spinner  animation="grow" />
                    </div>
                </Container>
            )
        }else{
            return(
                <Container>
                <div>
                    {this.state.productos.map((producto,i)=><Productos key={producto.id} productos={producto}/>)}
                </div>
                </Container>
            )
        }

    }
```

Vemos que en el promise se recibe un parámetro querySnapshot (se puede nombrar como se desee). El array de documentos lo obtendremos de **querySnapShot.docs**

Leer un documento dado su id

Para leer un documento dado su id debemos utilizar el método **firebase.db.doc("nombre_coleccion"+id).get()**

Este método también retorna un promise. Ejemplo:

```
import React,{useState,useEffect} from "react";
import {getProducto} from "../Services/ProductosServices"
import firebase from "../Config/firebase"
function ProductoDetalle(props){

    const [producto,setProducto] = useState({})

    useEffect(
        ()=>{

            firebase.db.doc("productos/"+props.match.params.id)
            .get()
            .then(doc=>{
                console.log("doc",doc)
                setProducto(doc)
            })
        },[]
    )

    return(
        <div>
            <div>{producto.data().name}</div>
            <div>{producto.data().price}</div>
        </div>

    )

}

export default ProductoDetalle;
```

Vemos que en el hook de efectos equivalente a un componentDidMount consultamos el documento dado el id que nos llega como parámetro a nuestra url de productoDetalle.

En este caso el then recibe directamente el documento buscado, sin necesidad de acceder a una propiedad docs como en el querySnapshots

Insertar un documento

Para insertar un documento debemos seleccionar la colección y llamar al método add:

```
const handleSubmit =  (e)=>{
    console.log(datos)
    firebase.db.collection('productos').add(datos)
    .then(doc=>{console.log(doc)})
    e.preventDefault();
}
```

Vemos que llamamos a **firebase.db.doc("nombre_coleccion"+id).add({…})**. Entre llaves debemos colocar el objeto con las propiedades a modificar.

Ver ejemplo de registro

Editar un documento

Para editar un documento debemos combinar la lectura del mismo con el método set:

```
const handleSubmit =  (e)=>{
    console.log(datos)
    const id = props.match.params.id;
    firebase.db.doc("productos/"+id)
    .set({
        name:datos.name,
        price:datos.price
    },{merge:true})
    .then(doc=>{
        console.log(doc)
    })
    e.preventDefault();
}
```

Vemos que llamamos a **firebase.db.doc("nombre_coleccion"+id).set({…})**. Entre llaves debemos colocar el objeto con las propiedades a modificar.

Eliminar un documento

Para eliminar un documento debemos combinar la lectura del mismo con el método delete:

```
const handleDelete = (e)=>{
    const id = props.match.params.id;
    console.log("Eliminar",id)
    firebase.db.doc("productos/"+id)
    .delete()
    .then(doc=>{
        console.log(doc)
    })
}
```

Vemos que llamamos a **firebase.db.doc("nombre_coleccion"+id).delete()** Entre llaves debemos colocar el objeto con las propiedades a modificar.

Bibliografía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.

https://reactjs.org/docs/forms.html

https://firebase.google.com/docs/database/web/read-and-write?hl=es-419

https://www.sitepoint.com/work-with-forms-in-react/

Lo que vimos:

En esta unidad aprendimos cómo insertar, leer, actualizar y eliminar datos en firebase

Lo que viene:

En la próxima unidad aprenderemos cómo realizar nuestra aplicación con todos los temas vistos de firebase.