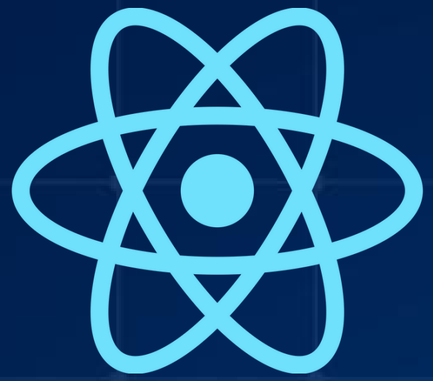
Secretaría de Desarrollo Económico Sustentable
Grandeza de México

Universidad Nacional Autónoma de México
Escuela Nacional de Estudios Superiores

IECAGto
Instituto Estatal de Capacitación



DESARROLLO WEB CON REACT

Temario

Fundamentos de desarrollo web

Javascript

Aplicaciones con React

Hooks y navegabilidad

Consumo de Web APIs

Herramientas en la nube

Sitios estáticos y SSR

Aplicaciones en tiempo real

Herramientas en la nube

Patrón State Reducer

El patrón State Reducer permite mantener un control más estricto sobre el manejo de estados de un **hook** o un **componente**, permitiendo solamente la ejecución de cambios predecibles que nos ayudan a no perder el control de nuestra información y evitar renderizados innecesarios.

```
const reducer = (state, action) => {
  switch (action.type) {
    case "ADD_TO_CART":
      return {
        ...state,
        cart: [...state.cart, action.payload],
      };
    case "REMOVE_FROM_CART":
      return {
        ...state,
        cart: state.cart.filter((product) => product.id !== action.payload.id),
      };
    case "TOTAL_PRICE":
      return {
        ...state,
        totalPrice: state.cart.reduce(
          (prev, product) => prev + product.price,
          0
        ),
      };
    default:
      return state;
  }
};
```

Redux

Redux es una biblioteca de código abierto que permite realizar **manejo de estado** en una aplicación. Su funcionamiento consiste en permitir a los **componentes** de React leer datos de un elemento llamado **Redux store**, y **despachar** acciones que permitan modificar la información de dicho store. Siendo que su funcionamiento es unidireccional.

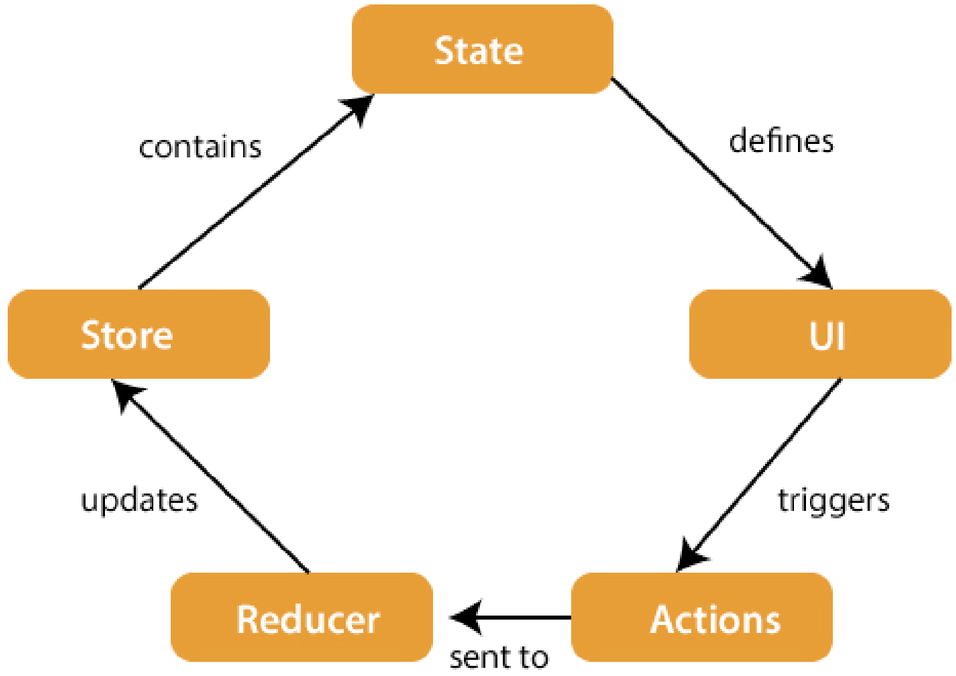
Arquitectura de Redux

Los componentes de la arquitectura de Redux son:

STORE: Es el lugar en que el estado es guardado. Administra un estado y cuenta con una función dispatch.

ACTION: Las acciones son lanzadas desde la interfaz gráfica (UI) y contienen información (payload) que puede ser interpretada por los reducers, cuyo objetivo es modificar información del estado.

REDUCER: Es el encargado de leer la carga de información de las acciones y ejecutar la actualización del estado de forma correspondiente.



Instalación de Redux

La instalación de Redux se puede realizar ejecutando el comando:

```
npm install @reduxjs/toolkit react-redux
```

Provider

React Redux incluye un componente llamado <Provider />, cuyo objetivo es poner a disponibilidad de toda la aplicación el **Redux Store**. El funcionamiento de este Provider es muy similar al **Context Provider.**

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import { Provider } from 'react-redux'
import store from './store'

import App from './App'

// As of React 18
const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(
  <Provider store={store}>
    <App />
  </Provider>
)
```

Hooks

React Redux pone a disposición el uso de dos Hooks que permiten a los componentes interactuar con el **Store**.

useSelector: Lee el valor del store y se suscribe para recibir actualizaciones y renderizarlas en el componente.

useDispatch: Provee la función despachadora que permite enviar acciones para modificar el **store.**

```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import {
  decrement,
  increment,
  incrementByAmount,
  incrementAsync,
  selectCount,
} from './counterSlice'
import styles from './Counter.module.css'

export function Counter() {
  const count = useSelector(selectCount)
  const dispatch = useDispatch()

  return (
    <div>
      <div className={styles.row}>
        <button
          className={styles.button}
          aria-label="Increment value"
          onClick={() => dispatch(increment())}
        >
          +
        </button>
        <span className={styles.value}>{count}</span>
        <button
          className={styles.button}
          aria-label="Decrement value"
          onClick={() => dispatch(decrement())}
        >
          -
        </button>
      </div>
      {/* omit additional rendering output here */}
    </div>
  )
}
```

Redux Toolkit

Toolkit es un conjunto de herramientas que permite hacer más eficiente el desarrollo de aplicaciones con React Redux.

Permite el manejo de casos comunes como la configuración del store, la creación de reducers, y lógica de actualización inmutable.

Su enfoque se basa en la segmentación de piezas (slices) para la distribución de estados complejos.

Creación del Redux Store

El toolkit de Redux permite configurar y crear el store de acuerdo a nuestras necesidades utilizando la función **configureStore**. Una vez creado el store, este se hace disponible mediante el componente **<Provider/>**

```
import { configureStore } from '@reduxjs/toolkit'

export const store = configureStore({
  reducer: {},
})
```

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { store } from './app/store'
import { Provider } from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Piezas de estado (state slices)

Los slices son creados utilizando la función **createSlice** disponible en la API del React Toolkit, un slice está compuesto por un nombre, un estado inicial, y las funciones reducer que permitirán la actualización del estado, el objetivo de estos elementos es partir el estado en elementos más pequeños que puedan ser manipulados con mayor facilidad.

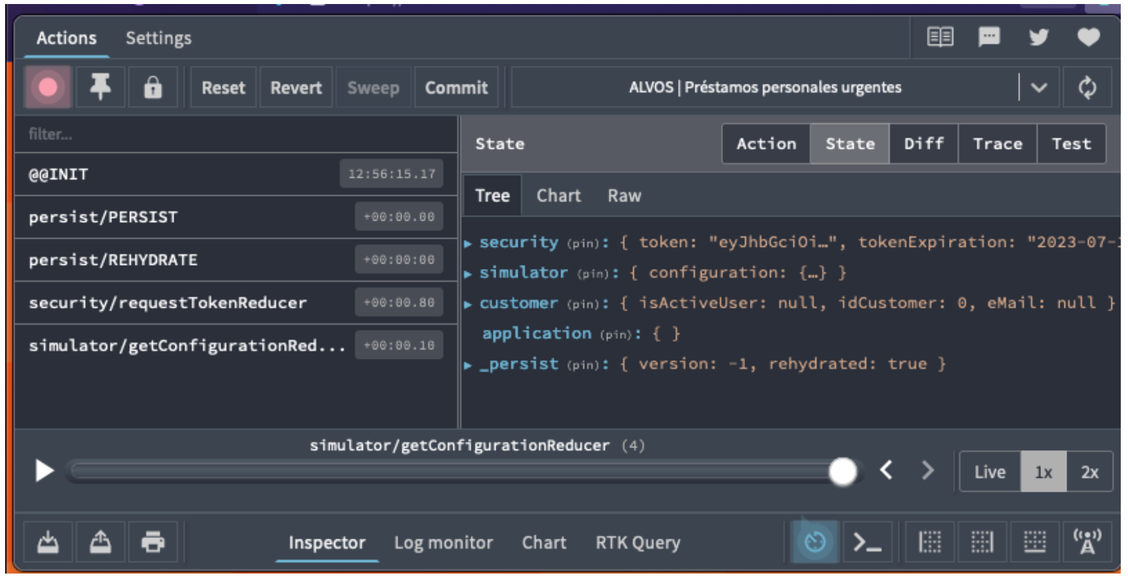
```
import { configureStore } from '@reduxjs/toolkit'
import counterReducer from '../features/counter/counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
})
```

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

Redux DevTools

Redux DevTools es una extensión para navegador que permite debuggear el estado de las aplicaciones que utilizan Redux.

Esta extensión es útil para identificar el estado actual, las acciones disponibles y las modificaciones realizadas al estado a través del tiempo.



Otros recursos

Redux Persist: Biblioteca que permite la persistencia de un estado en un storage local.

Redux Thunks: Permite la ejecución de funciones asyncronas para las acciones despachadas por redux.

Cómputo en la nube

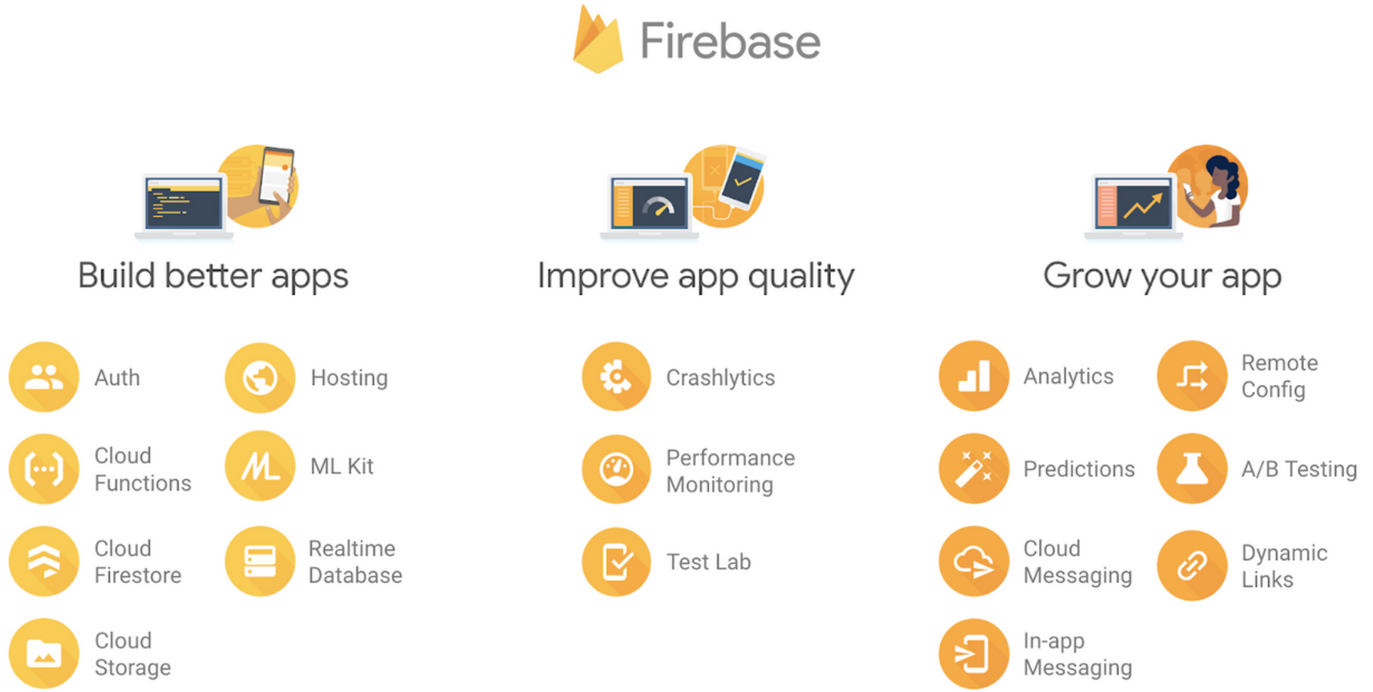
El **cómputo en la nube** se refiere a la entrega de servicios de computación, como servidores, almacenamiento, bases de datos, redes y software, a través de Internet. En lugar de tener que comprar y mantener **infraestructura** física costosa, las empresas y los desarrolladores pueden aprovechar los recursos en la nube para escalar y ejecutar sus aplicaciones de manera más eficiente.

Cómputo en la nube

Existen diferentes tipos de servicios en la nube, como la **infraestructura como servicio (IaaS)**, la plataforma como **servicio (PaaS)** y el **software como servicio (SaaS)**. Estos servicios permiten a las empresas centrarse en desarrollar sus aplicaciones sin preocuparse por la administración y mantenimiento de la infraestructura subyacente.

Firebase

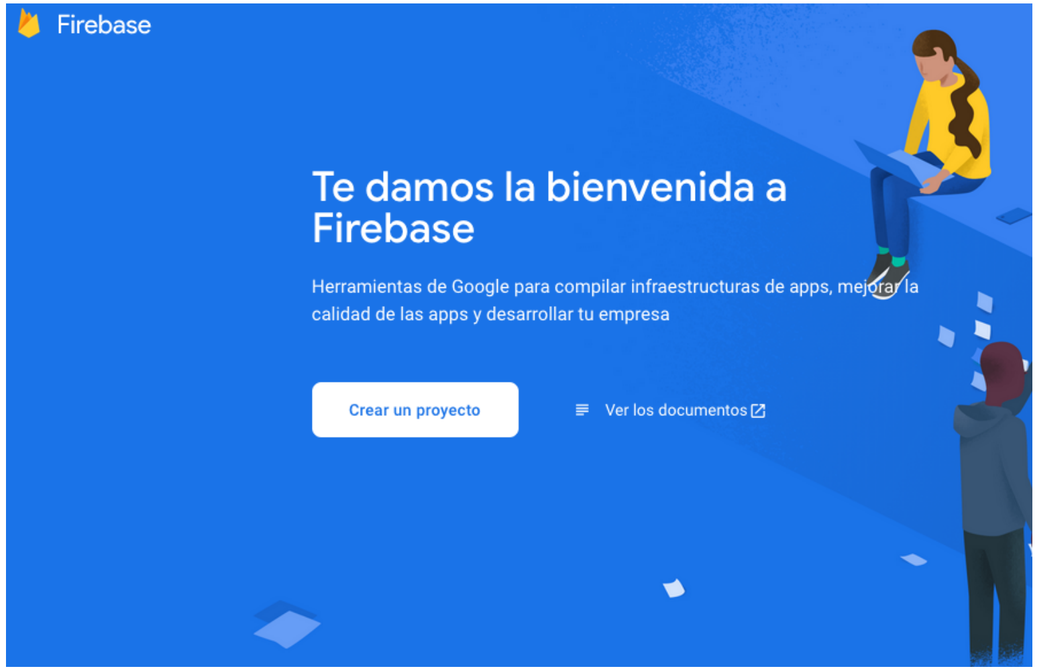
Firebase es una **plataforma de desarrollo de aplicaciones móviles y web** desarrollada por Google. Ofrece una amplia gama de servicios en la nube que permiten a los desarrolladores construir, mejorar y escalar sus aplicaciones de manera eficiente.



Integración de Firebase con React

Los servicios de Firebase pueden ser integrados de forma directa con aplicaciones de React. El primer paso consiste en crear una aplicación dentro de la consola de Firebase:

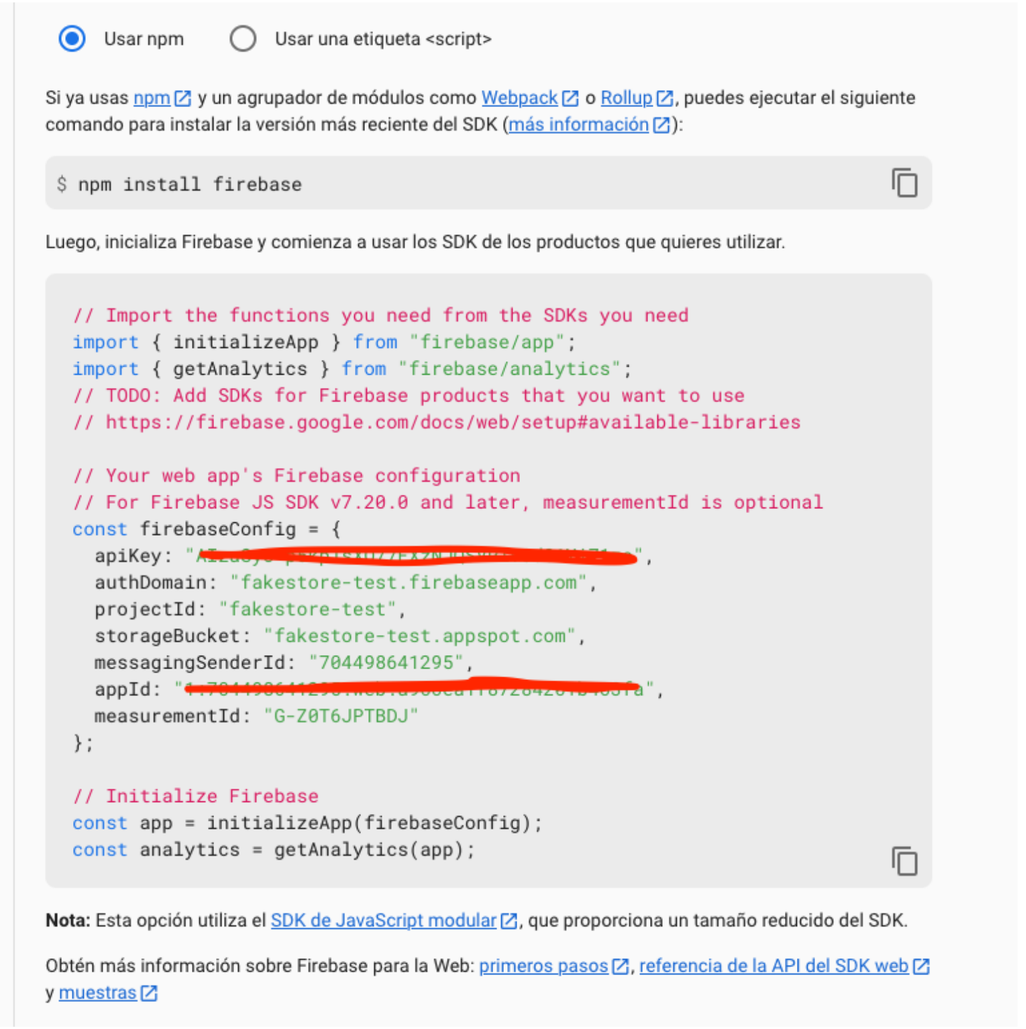
https://console.firebase.google.com



Integración de Firebase con React

Posterior a la creación de la aplicación, es necesario registrar esta como una aplicación web, de esta manera se brindarán las instrucciones adecuadas para la integración en el proyecto de React.

https://console.firebase.google.com



Usar npm Usar una etiqueta <script>

Si ya usas npm y un agrupador de módulos como Webpack o Rollup, puedes ejecutar el siguiente comando para instalar la versión más reciente del SDK (más información):

```
$ npm install firebase
```

Luego, inicializa Firebase y comienza a usar los SDK de los productos que quieres utilizar.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "[redacted]",
  authDomain: "fakestore-test.firebaseapp.com",
  projectId: "fakestore-test",
  storageBucket: "fakestore-test.appspot.com",
  messagingSenderId: "704498641295",
  appId: "[redacted]",
  measurementId: "G-Z0T6JPTBDJ"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

Nota: Esta opción utiliza el SDK de JavaScript modular, que proporciona un tamaño reducido del SDK.

Obtén más información sobre Firebase para la Web: primeros pasos, referencia de la API del SDK web y muestras

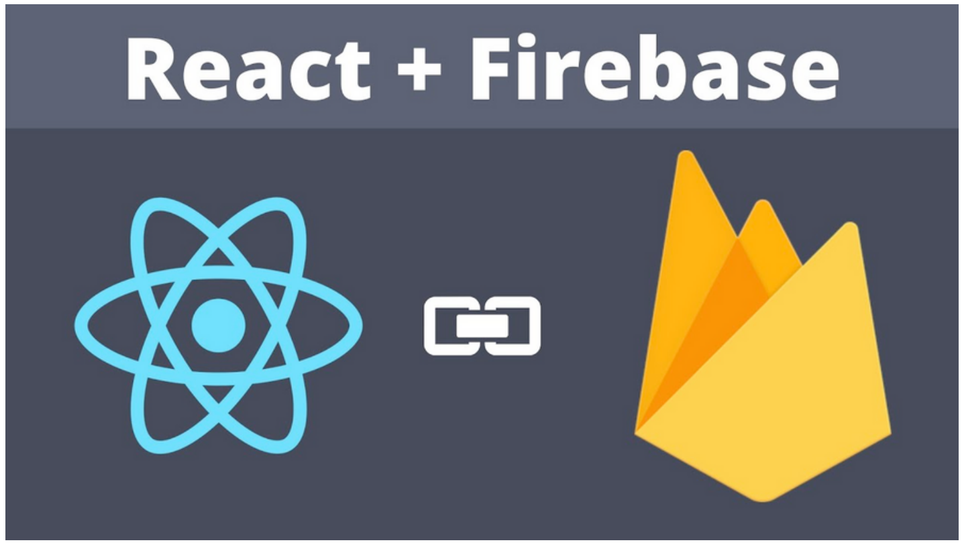
Integración de Firebase con React

Una vez creado y registrado el proyecto, es necesario instalar las dependencias de Firebase dentro de la aplicación de React. Esto puede realizarse con el siguiente comando:

```
npm install firebase
```

Posterior se copia el código generado por Firebase en el archivo index.js, y se procede a la instalación del CLI.

```
npm install -g firebase-tools
```



Integración de Firebase con React

El siguiente paso es loguearse utilizando el CLI a la cuenta de Firebase.

```
firebase login
```

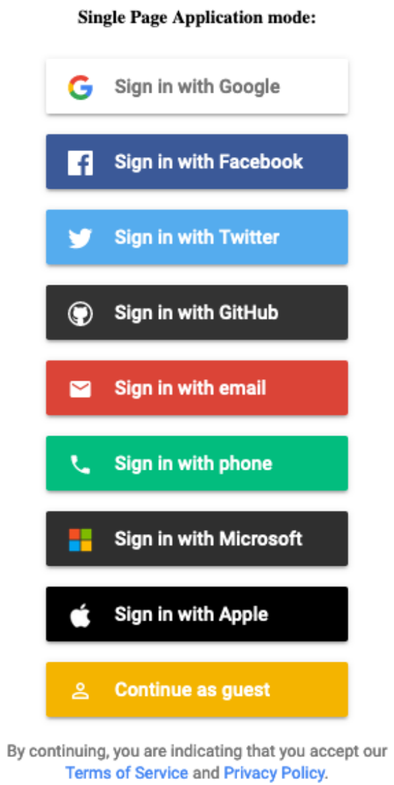
Iniciar el proyecto:

```
firebase init
```

Por último, se despliega la aplicación:

```
firebase deploy
```

Firebase Authentication



Firebase Authentication proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya elaboradas para autenticar a los usuarios en una app. Admite la autenticación mediante contraseñas, números de teléfono, proveedores de identidad federada populares, como Google, Facebook y Twitter, y otros.

Firebase Authentication

Integración:

```
const firebaseConfig = {
  apiKey: "AIzaSyC-Kksl1sXO77FXzNJQSVPlVOd81Yi71zc",
  authDomain: "fakestore-test.firebaseapp.com",
  projectId: "fakestore-test",
  storageBucket: "fakestore-test.appspot.com",
  messagingSenderId: "708194641295",
  appId: "1:704498641295:web:a900ea1f87284201b40839",
  measurementId: "G-Z0T6JPTYJT",
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Firebase Authentication and get a reference to the service
export const auth = getAuth(app);
```

```
import { auth } from "../../index";
import { signInWithEmailAndPassword } from "firebase/auth";

const handleLogin = async (e) => {
  e.preventDefault();
  try {
    const res = await signInWithEmailAndPassword(auth, email, password);

    // Aquí puedes redirigir al usuario a la página de inicio o realizar
    // otras acciones después de iniciar sesión correctamente
  } catch (err) {
    setError(err.message);
  }
};
```

Firebase Authentication

```jsx
import { onAuthStateChanged } from "firebase/auth";
import { useEffect, useState } from "react";

const Auth = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (user) => {
      setUser(user);
      setLoading(false);
    });

    return () => unsubscribe();
  }, []);

  if (loading) {
    // Aquí puedes mostrar un spinner de carga mientras se verifica el estado de autenticación
    return <div>Cargando...</div>;
  }

  if (!user) {
    // Aquí puedes redirigir al usuario a la página de inicio de sesión si no está autenticado
    // Por ejemplo: <Redirect to="/login" />;
    return <div>No autenticado. Redirigiendo...</div>;
  }

  return children;
};

export default Auth;

```

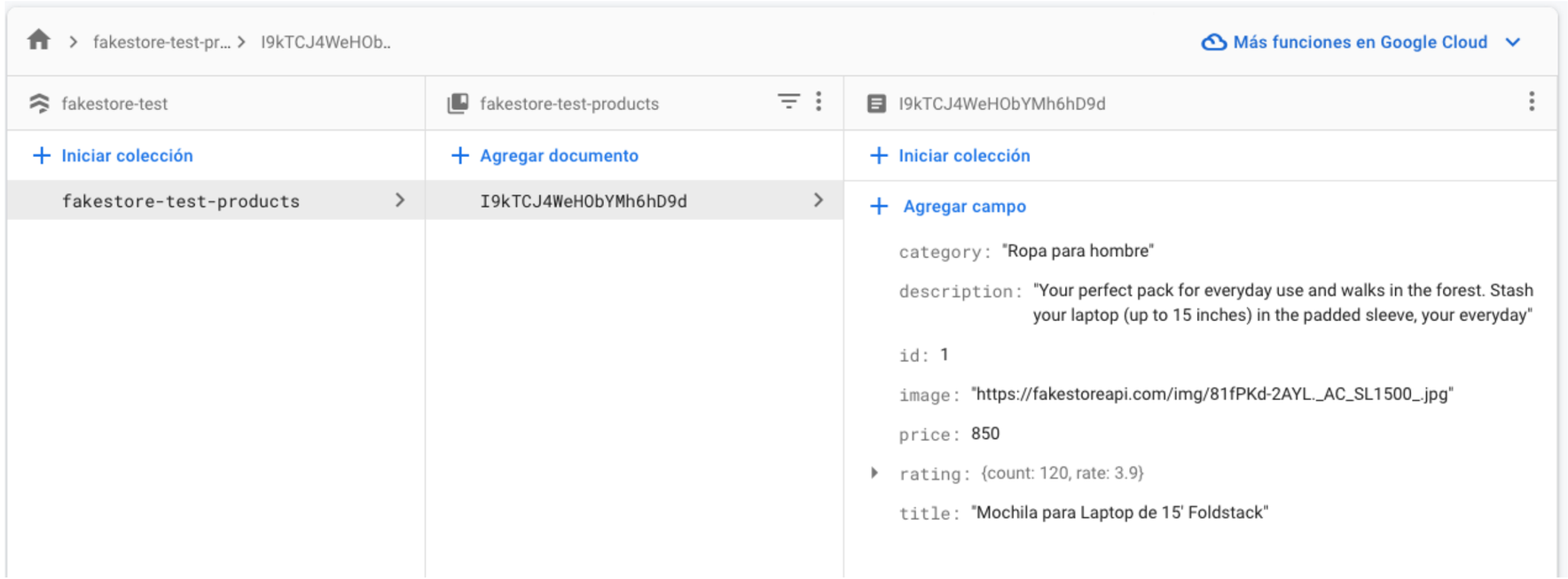
Un caso de uso muy común en la autenticación es la práctica de proteger vistas para que solo puedan ser visualizadas por un usuario autenticado. Esto se logra generando un **Componente de Orden Superior (HOC, High Order Component)**.

Cloud Firestore

Cloud Firestore es una base de datos NoSQL basada en documentos que permite el almacenamiento, sincronización y consulta de datos para aplicaciones web y móviles a una escala global.

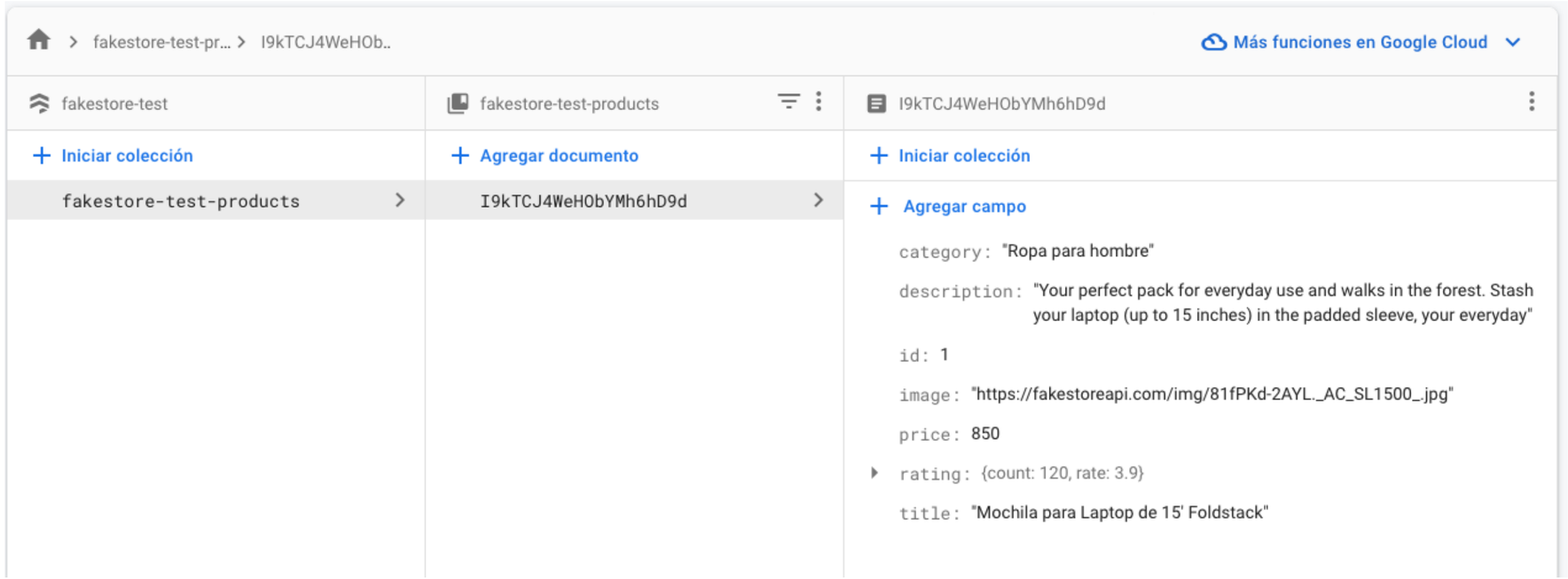
Cloud Firestore

Firebase ofrece los mecanismos necesarios para la consulta y creación de documentos desde su consola, sin necesidad de instalar programas adicionales.



Cloud Firestore

Firebase ofrece los mecanismos necesarios para la consulta y creación de documentos desde su consola, sin necesidad de instalar programas adicionales.



Cloud Firestore

Uso para lectura de datos

```
useEffect(() => {
  const getData = async () => {
    const data = await query(collection(firestore, "test_data"));

    onSnapshot(data, (querySnapshot) => {
      const databaseInfo = [];
      const dataIds = []

      querySnapshot.forEach((doc) => {
        databaseInfo.push(doc.data().testData);
        dataIds.push(doc.id)
      });

      setIds(dataIds)
      setInfo(databaseInfo)
    });
  }

  getData()
}, [])
```

Uso para escritura

```
const submithandler = (e) => {
  e.preventDefault()
  const ref = collection(firestore, "test_data")

  let data = {
      uuid: uuidv4(),
      testData: detail
  }

  try {
      addDoc(ref, data)
  } catch(err) {
      console.log(err)
  }

  setDetail("")
}

```

Ejercicio final (1)

Crear una aplicación de React para un **blog personal**. El blog debe contar con las siguientes características:

1. Las entradas del blog deben contar con, al menos, los siguientes datos:
    a. Título
    b. Contenido (preferentemente HTML)
    c. Fecha de creación
    d. Autor
    e. Imagen de portada
2. La página principal de la aplicación debe mostrar un listado de las entradas registradas.
3. Debe existir una página de detalle de blog para leer el contenido de la entrada.

Ejercicio final (2)

4. Utilizando Firebase o una herramienta similar, proporcionar las siguientes funcionalidades:
   - El sitio debe ser accesible públicamente desde una URL.
   - Debe existir una sección para listar, crear y eliminar las entradas de blog.
   - Dicha sección debe estar protegida por un mecanismo de autenticación.
   - La información de las entradas deberá persistir en una base de datos remota.

5. Funcionalidades opcionales:
   - Añadir una sección de comentarios a cada entrada que requiera que los usuarios inicien sesión para comentar.
   - Integrar Context API o Redux para el manejo de la sesión y la consulta de las entradas.

Secretaría de Desarrollo Económico Sustentable

Grandeza de México

Escuela Nacional de Estudios Superiores

IECAGto

Instituto Estatal de Capacitación

¡GRACIAS!