

Cuestionario Técnico Abierto: Mythos Web

Autenticación y Sesión

1. ¿Cómo se implementa el inicio de sesión en la aplicación web? Explicá el flujo desde el formulario hasta la redirección.
2. ¿Qué función de Firebase se usa para autenticar al usuario y cómo se maneja el error en caso de credenciales incorrectas?
3. ¿Cómo verifica la aplicación si el usuario ya está autenticado al cargar el componente Login? ¿Qué efecto tiene eso en la navegación?
4. ¿Qué significa `setPersistence(auth, browserLocalPersistence)` y por qué es importante para la experiencia del usuario?

Firebase y Base de Datos

5. ¿Cómo hace la web para traer los datos desde la base de datos? Explicá el flujo completo desde `onAuthStateChanged` hasta `onSnapshot`.
6. ¿Qué tipo de estructura se guarda en Firestore para las aventuras? ¿Cómo se transforma esa estructura para mostrarla en la interfaz?
7. ¿Qué pasa si el campo `sessionJson` está mal formado o no es un string válido? ¿Cómo lo maneja el código?
8. ¿Qué función se usa para eliminar una aventura y cómo se asegura que el usuario confirme la acción?

React y Componentes

9. ¿Qué rol cumple el componente `GameSessionItem` y cómo se comunica con `Aventuras.jsx`?
10. ¿Cómo se organiza la navegación entre páginas en `App.jsx` y qué librería se usa para manejar las rutas?
11. ¿Qué ventajas tiene usar `useEffect` para escuchar cambios en la autenticación y en la base de datos?
12. ¿Cómo se ordenan las aventuras por fecha y por qué es importante hacerlo antes de renderizar?

Manejo de Errores y UX

13. ¿Cómo se manejan los errores de login y de lectura de aventuras? ¿Qué feedback recibe el usuario?

14. ¿Qué pasa si el usuario no tiene aventuras guardadas? ¿Cómo se comunica eso en la interfaz?
15. ¿Qué medidas toma el código para evitar que se ejecuten acciones no deseadas al hacer clic en el botón de eliminar?

Infraestructura y Configuración

16. ¿Qué contiene el archivo firebase/config.js y por qué es fundamental para el funcionamiento del proyecto?
17. ¿Qué implicancias tiene usar Firestore como base de datos NoSQL en este tipo de aplicación?
18. ¿Cómo se asegura el código de que cada usuario solo vea sus propias aventuras? ¿Qué filtro se aplica en la consulta?

1. ¿Cómo se implementa el inicio de sesión en la aplicación web?

Respuesta: El inicio de sesión se implementa en el componente `Login.jsx`. El usuario ingresa su email y contraseña en un formulario. Al enviar el formulario, se ejecuta la función `handleLogin`, que llama a `signInWithEmailAndPassword(auth, email, password)` de `Firebase Authentication`.

Si las credenciales son válidas, `Firebase` devuelve un objeto `user`, y se redirige al usuario a la ruta `/aventuras` usando `navigate("/aventuras")`.

Ejemplo del código:

```
const userCredential = await signInWithEmailAndPassword(auth, email, password);
const user = userCredential.user;
navigate("/aventuras");
```

2. ¿Qué función de Firebase se usa para autenticar al usuario y cómo se maneja el error?

Respuesta: Se usa `signInWithEmailAndPassword` del módulo `firebase/auth`. Esta función recibe el objeto `auth`, el email y la contraseña. Si hay un error (por ejemplo, credenciales incorrectas), se captura en el bloque `catch` y se muestra un mensaje de error al usuario.

Ejemplo del código:

```
try {
  const userCredential = await signInWithEmailAndPassword(auth, email, password);
} catch (err) {
  setError("Email o contraseña incorrectos");
}
```

3. ¿Cómo verifica la aplicación si el usuario ya está autenticado al cargar el componente Login?

Respuesta: Dentro de un `useEffect`, se consulta `auth.currentUser`. Si ya hay un usuario autenticado, se redirige automáticamente a `/aventuras`. Esto evita que un usuario logueado vea el formulario de login.

Ejemplo del código:

```
useEffect(() => {  
  if (auth.currentUser) {  
    navigate("/aventuras");  
  }  
}, [navigate]);
```

4. ¿Qué significa `setPersistence(auth, browserLocalPersistence)`?

Respuesta: Esta línea configura Firebase para que mantenga la sesión del usuario en el `localStorage` del navegador. Esto significa que aunque el usuario cierre la pestaña o recargue la página, seguirá autenticado. Mejora la experiencia del usuario al evitar que tenga que loguearse cada vez.

Ejemplo del código:

```
setPersistence(auth, browserLocalPersistence)
```

5. ¿Cómo hace la web para traer los datos desde la base de datos?

Respuesta: En `Aventuras.jsx`, se usa `onAuthStateChanged` para detectar el usuario logueado. Luego se arma una consulta con `query` y `where("userId", "==", user.uid)` sobre la colección `game_sessions`.

Se usa `onSnapshot` para escuchar los cambios en tiempo real. Cada documento se transforma en un objeto de aventura, extrayendo `titulo`, `descripcion` y `updatedAt`. Finalmente, se ordenan por fecha y se renderizan.

Ejemplo del código:

```
const q = query(collection(db, "game_sessions"), where("userId",
"==", user.uid));
onSnapshot(q, (snapshot) => {
  const items = snapshot.docs.map((docSnap) => {
    const d = docSnap.data();
    const parsed = JSON.parse(d.sessionJson);
    return {
      titulo: parsed.metadata?.gameName,
      descripcion: parsed.metadata?.summary,
      updatedAt: new Date(parsed.metadata?.lastUpdated),
    };
  });
});
```

6. ¿Qué tipo de estructura se guarda en Firestore para las aventuras? ¿Cómo se transforma esa estructura para mostrarla en la interfaz?

Respuesta: Cada documento en la colección `game_sessions` contiene un campo `sessionJson`, que es un string en formato JSON. Este JSON incluye metadatos como `gameName`, `summary` y `lastUpdated`.

En el frontend, este string se parsea con `JSON.parse()` y se extraen esos campos para construir objetos que se muestran en la interfaz como tarjetas (`GameSessionItem`).

Ejemplo del código:

```
const raw = d.sessionJson;
const parsed = typeof raw === "string" ? JSON.parse(raw) : raw;
const titulo = parsed.metadata?.gameName;
const descripcion = parsed.metadata?.summary;
```

7. ¿Qué pasa si el campo `sessionJson` está mal formado o no es un string válido? ¿Cómo lo maneja el código?

Respuesta: El código intenta parsear `sessionJson` con `JSON.parse()`. Si el string está mal formado, se lanza una excepción que se captura en un bloque `try/catch`. En ese caso, se asigna un objeto vacío (`{}`) para evitar que la aplicación se rompa.

Ejemplo del código:

```
try {
  parsed = typeof raw === "string" ? JSON.parse(raw) : raw;
} catch (e) {
  console.error("Error parseando sessionJson:", e);
  parsed = {};
}
```

8. ¿Qué función se usa para eliminar una aventura y cómo se asegura que el usuario confirme la acción?

Respuesta: Se usa `deleteDoc(doc(db, "game_sessions", id))` para eliminar el documento correspondiente en Firestore. Antes de ejecutar la eliminación, se

muestra un `confirm()` al usuario para validar que realmente quiere borrar la aventura.

Ejemplo del código:

```
if (!confirm("¿Eliminar esta partida?")) return;  
await deleteDoc(doc(db, "game_sessions", id));
```

9. ¿Qué rol cumple el componente `GameSessionItem` y cómo se comunica con `Aventuras.jsx`?

Respuesta: `GameSessionItem` es un componente visual que representa una aventura individual. Recibe props como `aventura`, `onClick` y `onDelete`.

Desde `Aventuras.jsx`, se renderiza una lista de estos componentes, pasando los datos y funciones correspondientes para manejar eventos como abrir o eliminar.

Ejemplo del uso:

```
<GameSessionItem  
  key={a.id}  
  aventura={a}  
  onClick={handleAbrir}  
  onDelete={handleEliminar}  
>  
</>
```

10. ¿Cómo se organiza la navegación entre páginas en `App.jsx` y qué librería se usa para manejar las rutas?

Respuesta: Se usa `react-router-dom` para definir las rutas. En `App.jsx`, se configura un `Router` con dos rutas principales: `/` para el login y `/aventuras` para la vista de aventuras.

Cada ruta está asociada a un componente (`Login` o `Aventuras`) que se renderiza según la URL.

Ejemplo del código:

```
<Router>  
  <Routes>  
    <Route path="/" element={<Login />} />  
  </Routes>  
</Router>
```

```
    <Route path="/aventuras" element={<Aventuras />} />  
  </Routes>  
</Router>
```

Perfecto Bruno, acá van las respuestas explicadas para las preguntas 11 a 18. Estas te van a ayudar a dominar tanto la lógica del sistema como los detalles técnicos del código:

11. ¿Qué ventajas tiene usar useEffect para escuchar cambios en la autenticación y en la base de datos?

Respuesta: useEffect permite ejecutar lógica cuando el componente se monta o cuando cambian ciertas dependencias.

En este proyecto se usa para:

- Detectar si el usuario está autenticado (onAuthStateChanged).
- Escuchar cambios en la colección game_sessions en tiempo real (onSnapshot).

Esto garantiza que la interfaz se actualice automáticamente cuando el usuario cambia o cuando se modifican sus aventuras en la base de datos.

12. ¿Cómo se ordenan las aventuras por fecha y por qué es importante hacerlo antes de renderizar?

Respuesta: Después de transformar los documentos de Firestore en objetos de aventura, se ordenan con `items.sort((a, b) => b.updatedAt - a.updatedAt)`.

Esto asegura que las aventuras más recientes aparezcan primero en la lista, lo cual mejora la experiencia del usuario al mostrarle lo más relevante arriba.

13. ¿Cómo se manejan los errores de login y de lectura de aventuras? ¿Qué feedback recibe el usuario?

Respuesta:

- En el login, si las credenciales son incorrectas, se muestra un mensaje con `setError("Email o contraseña incorrectos")`.

- En la lectura de aventuras, si ocurre un error en `onSnapshot`, se muestra un mensaje rojo con `setError("Error leyendo partidas desde Firestore")`.

Esto le da al usuario una respuesta clara y evita que la app falle silenciosamente.

14. ¿Qué pasa si el usuario no tiene aventuras guardadas? ¿Cómo se comunica eso en la interfaz?

Respuesta: Si la lista aventuras está vacía, se muestra un mensaje amigable:

`<p>No tenés aventuras todavía 😞</p>`

Esto evita que el usuario vea una pantalla vacía y le da contexto sobre el estado actual de su cuenta.

15. ¿Qué medidas toma el código para evitar que se ejecuten acciones no deseadas al hacer clic en el botón de eliminar?

Respuesta:

- Se usa `e.stopPropagation()` para evitar que el clic en el botón de eliminar dispare el evento de abrir la aventura.
- Se muestra un `confirm()` para que el usuario valide la acción antes de eliminar.

Esto protege contra errores de clic y asegura que el usuario tenga control sobre la eliminación.

16. ¿Qué contiene el archivo `firebase/config.js` y por qué es fundamental para el funcionamiento del proyecto?

Respuesta: Este archivo inicializa Firebase con las credenciales del proyecto (`firebaseConfig`).

Exporta:

- `auth`: para autenticación.
- `db`: para acceder a Firestore.
- Configura la persistencia de sesión con `setPersistence`.

Sin este archivo, la app no podría conectarse a Firebase ni autenticar usuarios.

17. ¿Qué implicancias tiene usar Firestore como base de datos NoSQL en este tipo de aplicación?

Respuesta: Firestore permite guardar documentos con estructura flexible, ideal para aventuras que pueden tener distintos campos.

Además:

- Soporta sincronización en tiempo real (onSnapshot).
- Escala automáticamente.
- No requiere backend propio, lo que simplifica el desarrollo.

Es perfecto para apps móviles/web que necesitan rapidez y flexibilidad.

18. ¿Cómo se asegura el código de que cada usuario solo vea sus propias aventuras? ¿Qué filtro se aplica en la consulta?

Respuesta: Se usa `where("userId", "==", user.uid)` en la consulta a Firestore.

Esto filtra los documentos de la colección `game_sessions` para que solo se traigan los que tienen el `userId` del usuario autenticado.

Ejemplo del código:

```
const q = query(collection(db, "game_sessions"), where("userId",  
"==", user.uid));
```