Memoria del Proyecto: Juego de Piedra, Papel o Tijera utilizando REST y GraphQL

1. Introducción

Este proyecto consiste en implementar un juego de piedra, papel o tijera que permite a dos jugadores competir a través de una red local o internet. La comunicación entre los jugadores se realiza mediante APIs desarrolladas en Node.js, utilizando dos enfoques distintos:

- Un modelo basado en REST utilizando Express.
- Una implementación basada en GraphQL.

El objetivo principal del proyecto es comprender y comparar los enfoques de REST y GraphQL en el desarrollo de servicios web, implementando las mismas funcionalidades y lógica para ambos casos.

2. Tecnologías utilizadas

Node.js

Palabra clave: Entorno de ejecución para JavaScript. **Usos:**

- Proveer la base para ejecutar el código en el servidor.
- Gestionar las solicitudes y respuestas del cliente.

Express

Palabra clave: Framework para construir aplicaciones web. **Usos:**

- Facilitar la definición de rutas.
- Manejar solicitudes HTTP para las operaciones del juego.

GraphQL

Palabra clave: Lenguaje de consulta para APIs. **Usos:**

- Diseñar un esquema que permita mutaciones y consultas estructuradas.
- Implementar un único punto de entrada para las solicitudes del cliente.

Postman

Palabra clave: Herramienta para probar y consumir APIs. **Usos:**

- Verificar la funcionalidad de las rutas REST.
- Enviar mutaciones y consultas para probar GraphQL.

3. Funcionalidades del juego

- Cada jugador puede realizar su jugada enviando su elección ("piedra", "papel" o "tijera") y su nombre.
- Las jugadas se almacenan temporalmente hasta que ambos jugadores hayan enviado sus elecciones.
- Se determina un ganador según las reglas clásicas del juego:
    - Piedra vence a Tijera.
    - Tijera vence a Papel.
    - Papel vence a Piedra.
    - Jugadas iguales resultan en empate.
- Una vez determinado el resultado, las jugadas se reinician para una nueva partida.

4. Descripción del código REST

Secciones principales

Configuración inicial

- **Palabra clave:** Middleware de Express.
- **Usos:**
    - Procesar datos en formato JSON.
    - Inicializar variables para almacenar jugadas.

Ruta principal

- **Palabra clave:** app.get('/').
- **Usos:**
    - Proveer un mensaje de bienvenida al acceder a la ruta base.

Ruta para realizar jugadas

- **Palabra clave:** app.post('/jugar').
- **Usos:**
    - Validar las solicitudes de los jugadores.
    - Registrar las jugadas y evaluar si ya se puede determinar un ganador.

Lógica para determinar el ganador

- **Palabra clave:** Función determinarGanador().
- **Usos:**
    - Comparar las elecciones de los jugadores.
    - Determinar el resultado del juego.

Inicio del servidor

- **Palabra clave:** app.listen().
- **Usos:**
    - Iniciar el servidor en un puerto específico.

5. Descripción del código GraphQL

Esquema y características

Middleware graphqlHTTP

- **Palabra clave:** Middleware GraphQL.
- **Usos:**
    - Manejar las solicitudes GraphQL en un único punto de entrada.

Definición del esquema

- **Palabra clave:** Tipos y mutaciones de GraphQL.
- **Usos:**
    - Representar las jugadas y los resultados.
    - Registrar jugadas y determinar el ganador.

Diferencias clave respecto a REST

- **Palabra clave:** Flexibilidad.
- **Usos:**
    - Permitir consultas y mutaciones en un único punto.
    - Proveer respuestas estructuradas y personalizables.

6. Pruebas

Se utilizaron Postman y GraphiQL para probar ambas implementaciones. Las pruebas realizadas incluyeron:

- Enviar una jugada válida y recibir una respuesta indicando que se espera la jugada del otro jugador.
- Enviar dos jugadas válidas y verificar que el resultado es correcto.
- Probar casos inválidos, como:
    - Jugadas no permitidas ("lagarto").
    - Campos faltantes en la solicitud.

7. Conclusiones

REST

- **Simplicidad:** Sencillo de implementar y probar.
- **Enfoque estructurado:** Cada acción está vinculada a una ruta específica.

GraphQL

- **Flexibilidad:** Más adecuado para consultas y mutaciones complejas.
- **Configuración:** Requiere mayor configuración inicial.

Ambos enfoques tienen ventajas y desventajas dependiendo del caso de uso. GraphQL es más útil cuando se requiere un mayor control sobre los datos devueltos o se manejan estructuras complejas. REST es adecuado para servicios simples y rápidos de implementar.

8. Posibles mejoras

- Implementar autenticación para los jugadores.
- Almacenar las partidas en una base de datos.