GAMEDLE

Enrique Castillo Colombás, Jaime Medem Recio, Carlos Agustí Canga, Gabriela Fernandez García-Poggio

PAT GAMEDLE

Contenido

Introducción

Con nuestro proyecto no queríamos limitarnos a copiar Wordle; por lo que montamos un sistema que cada día cambiara la palabra, guardara quién juega y cómo lo hace, y controlara que solo los admins pudieran editarla.

Toda la está en los servicios: ahí revisamos el rol del usuario, evitamos puntuaciones repetidas y gestionamos el login/logout metiendo la cookie de sesión. Luego los controladores REST solo exponen lo justo: un endpoint para conseguir la palabra del día, otro para cambiarla si eres ADMIN, y uno más para guardar los resultados de cada partida.

Para utilizar la aplicación recomiendo correr el servidor y acceder a http://localhost:8080/login.html para empezar. Tendrá la opción de iniciar sesión utilizando las claves que hay en data.sql o creándose su propio usuario.

Funcionalidades principales

En GAMEDLE empiezas creando tu cuenta y definiendo si eres jugador o admin. Al registrarte, metes tu correo, contraseña y nombre, y en el backend guardamos todo. Cuando haces login, se da una cookie de sesión que luego usamos para saber quién es quién en cada petición.

Una vez dentro, lo primero que ves es la “palabra del día”. El cliente pide a /api/wordle/today y el servidor devuelve justo la entrada de hoy de la tabla WORDS. Vas intentando adivinarla y al terminar la partida se hace un POST a /api/score/wordle con tu resultado: número de intentos, fecha y si acertaste o no. Eso se guarda en GAME_SCORE, así luego puedes ver tu historial y comparar con otros usarios.

Si eres admin y piensas que la palabra propuesta no vale, aparece un endpoint especial (PUT /api/wordle/change) que solo acepta la petición si tu cookie de sesión pertenece a un rol ADMIN. Con eso puedes cambiar la palabra del día directamente en la base de datos, sin tocar nada manualmente. Todo ese flujo (registro, login, obtener palabra, enviar puntuación y editarla si toca) funciona usando controladores sencillos y servicios que solo evalúan roles y guardan lo necesario.

Endpoints del Backend

En GAMEDLE la comunicación entre cliente y servidor se basa en rutas muy claras que reflejan cada una de las acciones del juego y la gestión de usuarios. Cuando un nuevo jugador quiere unirse, envía sus datos a POST /api/users; el servidor valida que no haya otro correo igual y devuelve al instante su perfil con un 201 Created. Para conectarse, el frontend hace un POST /api/users/me/session con el correo y la contraseña, y si todo encaja el servidor genera un token, lo guarda en la tabla TOKEN y lo envía en una cookie de solo lectura llamada “session”. Esa cookie es la llave que utilizamos después para identificar al usuario: cualquier ruta protegida comprueba que exista y que corresponda a un usuario válido.

Para recuperar datos del propio perfil el cliente pide GET /api/users/me, y si la cookie es correcta recibe un JSON con email, nombre y rol. Actualizar el perfil es tan sencillo como un PUT /api/users/me con los campos nuevos, y borrar la cuenta se hace con DELETE /api/users/me, que elimina tanto la fila de APP_USER como el token en cascada. El logout, en cambio, es otro DELETE /api/users/me/session, que no toca el usuario en la base de datos pero fuerza la expiración de la cookie para que deje de funcionar.

La parte del juego arranca con GET /api/wordle/today, que devuelve directamente la palabra asignada para el día. Si quieres revisar una fecha concreta, hay una ruta GET /api/wordle/{date} que acepta la fecha en formato ISO y devuelve la palabra de aquel día. En el caso de que la selección automática no convenza, los administradores pueden hacer un PUT /api/wordle/change enviando la nueva palabra en el cuerpo de la petición; el servidor comprueba la cookie, valida que el rol sea ADMIN y, si todo está bien, actualiza el registro de hoy en la tabla WORDS.

Cuando el jugador termina una partida, sea victoria o derrota, el frontend envía un POST /api/score/wordle con un objeto ScoreRequest que incluye el identificador del usuario (se extrae de la sesión), la fecha, el número de intentos y si acertó o no. El backend revisa que no exista ya una puntuación para ese día y guarda el resultado

en la tabla GAME_SCORE. De este modo, cada petición REST refleja una acción concreta: desde registrarse hasta cambiar la palabra o almacenar la puntuación, todo el flujo está cubierto con rutas sencillas que utilizan cookies de sesión para la autenticación y roles para la autorización.

Servicios complejos

El servicio que más tela tiene en GAMEDLE es el de usuarios y sesiones: todo parte de UserService. Ahí está la lógica de registro, login y logout. Cuando alguien se registra, el servicio comprueba que no haya otro email igual, guarda la contraseña y crea el AppUser. En el login, UserService busca al usuario por email, compara la contraseña y genera un Token nuevo solo si no existe uno activo (así se evita quedarse con muchos tokens innecesarios por usuario).

Para el logout, basta con decirle al servicio que borre el token que tenemos guardado y devolver una cookie vacía al cliente. Todo esto se hace con llamadas al TokenRepository, que sabe buscar o eliminar el token en cascada gracias a la relación 1-1 con AppUser. De esta forma, los controladores solo piden “dale al servicio esto” o “quita esto otro” y no se meten en internals de la base de datos.

El WordService también tiene algo de complejidad. Cuando se arranca la app o algún cliente pide la palabra de hoy, el servicio mira en la tabla WORDS si ya existe la fecha y, si no, la inicializa con una lista prefijada. La palabra se cambia pasándole el nuevo valor al método changeWordle(), que antes de actualizar comprueba que el token que llega de la cookie pertenece a un admin. Aquí usamos WordsRepository.findByDate() y luego un save(), y el servicio se encarga de lanzar un error si algo falla. Así mantenemos todas las reglas (no duplicar entradas, permisos de admin) en un solo sitio y el resto del código puede llamarlo sin preocuparse.

TEST

Los tests que se han realizado verifican la lógica interna de los servicios (por ejemplo, que la palabra del día se seleccione o cambie como toca), pruebas rápidas con una base en memoria para simular la base de datos y ver que guardar o borrar registros funciona, y comprobaciones de los modelos para asegurarnos de que los datos cumplen las restricciones que definimos. Además, desde el cliente enviamos peticiones a la API y revisamos que las rutas respondan con los códigos esperados y gestionen bien la cookie de sesión.