Requisitos de la Prueba - Informe de Implementación Técnica

Tipos de usuario

- En la aplicación se han definido dos roles principales: Cuidadores y Maestros. Ambos roles se almacenan en la base de datos mediante un campo 'role' tipado (enum) gestionado por Prisma.

- Cuidadores: este rol puede crear, listar y actualizar criaturas asociadas a su propio usuario. En la interfaz no se muestra el botón de eliminar y, adicionalmente, el backend valida el userId para evitar operaciones no permitidas.

- Maestros de criaturas: este rol hereda las mismas capacidades que el Cuidador (crear, leer y actualizar) y, además, tiene acceso a la funcionalidad de eliminar criaturas. El endpoint DELETE comprueba tanto el userId como la existencia de la criatura antes de borrar.

Funcionalidades de la aplicación - Autenticación de usuario

- La autenticación se ha implementado utilizando NextAuth (versión 5, con el App Router de Next.js). Se emplea un proveedor de credenciales (Credentials Provider) para gestionar el inicio de sesión mediante email y contraseña almacenados en la base de datos.

- Durante el proceso de registro, el usuario selecciona explícitamente si quiere ser Cuidador o Maestro. Este valor se guarda en la tabla User y posteriormente se adjunta al token y a la sesión de NextAuth.

- Tras el login, la aplicación consulta la sesión activa y, en función del rol del usuario, redirige automáticamente a la ruta correspondiente: '/[locale]/cuidador/santuario' para cuidadores y '/[locale]/maestro/santuario' para maestros.

Funcionalidades de la aplicación - Gestión de criaturas mágicas

- La gestión de criaturas se ha implementado con un CRUD completo expuesto a través de API Routes en el App Router: GET y POST en '/api/creatures' y PUT/DELETE en '/api/creatures/[id]'.

- Cada criatura se modela en Prisma con los campos obligatorios: nombre (String), tipo (enum TipoCriatura con valores DRAGON, FENIX, GOLEM, VAMPIRO, UNICORNIO) y nivelPoder (String). Además se incluye un campo entrenada (Boolean) y claves de auditoría (createdAt, updatedAt).

- Todas las operaciones de lectura y escritura se filtran por userId, que se obtiene a partir de la sesión autenticada (auth()). De esta forma, cada usuario solo puede ver y manipular sus propias criaturas, incluso aunque intente acceder manualmente a IDs de otros usuarios.

Interfaz de usuario

- La capa de presentación se ha construido con el App Router de Next.js y estilos modulares utilizando ficheros SCSS (CSS Modules), cumpliendo el requisito de usar SASS para el diseño.

- Se ha creado una página de inicio de sesión con diseño personalizado, integrando el formulario con NextAuth para disparar el flujo de autenticación mediante credenciales.

- La página de registro permite introducir el email, la contraseña y seleccionar el rol (Cuidador/Maestro). Esta pantalla está conectada a un endpoint específico de registro que persiste los datos con Prisma.

- Tras iniciar sesión, el usuario es redirigido a una vista principal (Santuario) donde se muestra una tabla con la lista de criaturas mágicas asociadas a su cuenta. Esta tabla se alimenta mediante llamadas fetch al endpoint '/api/creatures'.

- Tanto cuidadores como maestros disponen de un formulario en la interfaz para crear y editar criaturas. El formulario controla el estado local con React (useState) y, según si existe un id en edición, realiza un POST (crear) o un PUT (editar).

- En el caso de los Maestros, la tabla incluye un botón adicional para eliminar criaturas. Esta acción dispara una petición DELETE al backend y, si la operación tiene éxito, se sincroniza el estado del frontend eliminando la fila correspondiente.

- Para los Maestros se ha implementado además una sección especial en su perfil donde se muestra información adicional relacionada con su actividad, incluyendo el total de criaturas creadas (dato que puede obtenerse filtrando por userId en la tabla de criaturas).

Protección de rutas

- La protección de rutas se ha resuelto mediante un middleware de Next.js que utiliza la función auth() de NextAuth. Este middleware inspecciona la ruta solicitada e identifica si pertenece a las secciones '/[locale]/cuidador/*' o '/[locale]/maestro/*'.

- Si la ruta es protegida y no existe una sesión válida, el middleware redirige automáticamente al usuario a la página de login correspondiente al locale detectado (por ejemplo, '/es/login'). De esta manera, se evita el acceso a zonas privadas sin autenticación.

- Además de la protección a nivel de middleware, la aplicación diferencia el comportamiento según el tipo de usuario (rol) tanto en la interfaz (mostrando u ocultando botones como eliminar) como en el backend (validando el rol en las operaciones sensibles).

Requisitos técnicos

- El proyecto está desarrollado con Next.js y TypeScript, utilizando el App Router para organizar las páginas y las rutas de la API.

- Para la autenticación se ha integrado NextAuth v5, utilizando un proveedor de credenciales y callbacks para propagar la información del usuario y su rol al token y a la sesión.

- La capa de estilos se ha implementado con SCSS (SASS), utilizando módulos (ficheros .module.scss) para aislar los estilos por componente y mantener una arquitectura de estilos clara y escalable.

- La capa de acceso a datos se realiza mediante Prisma como ORM, conectando con una base de datos SQLite durante el desarrollo.

Aspectos valorados positivamente

- La aplicación soporta multilenguaje gracias a la integración de next-intl. El parámetro [locale] en las rutas permite que la interfaz se adapte al idioma configurado en la URL.

- La arquitectura de estilos se basa en módulos SCSS organizados por páginas (login, registro, santuario, perfil), lo que facilita el mantenimiento y la reutilización de patrones de diseño.

- El proyecto sigue una estructura clara separando la lógica por roles (cuidador/maestro) y por ámbitos (páginas, lógica de autenticación, acceso a datos, etc.), lo que mejora la legibilidad y la mantenibilidad.

- Los endpoints de la API se han definido de forma explícita y coherente ('/api/creatures', '/api/creatures/[id]', '/api/auth/register', etc.), y cada uno valida correctamente la sesión y el userId.

- En el frontend se gestionan los estados con React (useState, useEffect), sincronizando la información mostrada en tablas y formularios con las

respuestas de la API (por ejemplo, actualizando la lista tras crear, editar o eliminar una criatura).

Diseño, colores y tipografías

- Para el diseño se han utilizado los assets proporcionados, así como la estructura visual propuesta (pantallas de login, registro, santuario y perfil).

- Se han respetado los colores definidos en el enunciado: Primary (#9C5CE1), Secondary (#C77E01), Dark (#222222) y Light (#777777), aplicados a botones, textos destacados y fondos.

- En cuanto a la tipografía, se han utilizado las fuentes Sedan SC y Sedan, combinándolas para títulos (tipografía en mayúsculas y con tracking) y textos de párrafo, logrando una interfaz coherente con la guía visual propuesta.