Isabella Ocampo Soto - A00382369
Alejandro Córdoba Erazo - A00395678
Valentina Gonzalez Tapiero -A00394152

Nombre y Descripción del Proyecto

TickTopia - Boletos seguros y rápidos para todos.

El proyecto consiste en una aplicación web desarrollada con Next.js, que consume una API previamente implementada en NestJS. La aplicación permite a los usuarios autenticarse, gestionar eventos (crear, editar, listar, eliminar) y subir imágenes usando Cloudinary. Se implementan roles diferenciados (admin y usuario) con control de acceso. Además, cuenta con pruebas automatizadas (unitarias y E2E con Playwright) y está desplegada en la nube.

1. **Funcionalidades Implementadas**

- Autenticación con JWT.
- Registro, login y logout de usuarios.
- Diferenciación de roles (admin/usuario).
- CRUD de eventos con validaciones.
- Subida de imágenes a Cloudinary.
- Interfaz responsiva y validación de formularios.
- Rutas protegidas según autenticación y rol.
- Simulación de pagos con Stripe.
- Selección dinámica de ciudades y departamentos con API pública de Colombia.
- Despliegue en nube y ejecución de pruebas en CI.

2. **Autenticación**

   Se utiliza JWT para manejar sesiones seguras. Al iniciar sesión, el token se almacena como cookie httpOnly, impidiendo accesos desde el frontend. Las rutas protegidas se validan tanto desde el servidor (middleware) como en el cliente (hooks useAuth). El token se refresca automáticamente si el backend lo permite (a través de cookies persistentes).

3. **Autorización**

   Se definieron cuatro roles:

   - Admin: puede crear, editar y eliminar eventos, de igual forma crear tickets y presentaciones
   - Usuario: Puede ver los eventos, comprar tiquetes, visualizar sus propios tiquetes
   - Event-manager: puede crear, editar, eliminar eventos.
   - Ticket-Checker: Puede redimir tiquetes el dia de la presentación

   El frontend verifica el rol del usuario mediante un estado global (authStore con Redux y ajusta la UI en consecuencia (botones, rutas, menús). En el backend se valida también el rol antes de permitir ciertas acciones (por ejemplo, eliminar un evento).

4. **Gestión del Estado**
   Se implementó la gestión del estado con Redux. Se mantiene centralizado:
   - El estado de autenticación (user, isAuthenticated).
   - El rol del usuario (role).
   - Los eventos cargados desde la API.

   Esta gestión permite compartir datos entre componentes sin prop drilling, y facilita proteger rutas y condicionar componentes UI.

5. **Interfaz de Usuario**
   La interfaz fue construida con React y Tailwind CSS, cumpliendo criterios de usabilidad:
   - Navegación intuitiva con Navbar persistente.
   - Páginas de login, dashboard, eventos, crear y editar evento.
   - Validaciones en formularios con feedback visual.
   - Componente de carga para imágenes (previsualización).
   - Paginación y manejo de errores desde la API
   - Campos dinámicos para ciudades/departamentos usando API pública de Colombia.
   - Botón de pago con Stripe en flujo de compra de boletas (modo de prueba).
6. **Pruebas Automatizadas**
   Se implementaron dos tipos de pruebas:

   Pruebas Unitarias (Jest)
   - Validación de funciones auxiliares.
   - Verificación de hooks y componentes.

   Pruebas E2E (Playwright)
   Las pruebas E2E se realizaron con Playwright, abarcando escenarios clave como:
   - **Inicio de sesión exitoso y fallido**
     Se simula un flujo completo de login, con validaciones ante credenciales incorrectas y éxito redirigiendo al usuario al dashboard.
   - **Creación de evento por administrador**
     Un usuario con rol de administrador puede ingresar al formulario de creación de eventos, completar los datos y enviar el formulario correctamente.
   - **Restricción de acceso por rol**
     Un usuario sin permisos (rol distinto a admin) no puede acceder a rutas protegidas como la de crear o editar eventos. Se verifica el redireccionamiento o mensaje de acceso denegado.
   - **Visualización de eventos públicos**
     Se testea la carga y visualización de la lista de eventos accesibles para cualquier usuario autenticado.
   - **Subida de imágenes**

El test valida que se pueda subir correctamente una imagen como parte de la creación de un evento, incluyendo el manejo de vista previa.

7. **Despliegue**
   La aplicación fue desplegada en AWS. El backend NestJS está disponible en AWS (o plataforma equivalente). Las variables de entorno están correctamente configuradas en producción. Se utilizaron GitHub Actions para ejecutar pruebas antes del despliegue.

Link al repositorio

https://github.com/elkofix/tickopia-shop