Documentación Técnica del Sistema de Tickets

Contenido

Resumen del Proyecto

Este sistema de tickets permite a distintos tipos de usuarios (usuario, funcionario, admin) registrar, gestionar, y resolver solicitudes. Incluye visualización de métricas, un chatbot integrado y un sistema de calendario.

Cuenta con una arquitectura desacoplada frontend-backend, permite la autenticación mediante JWT, consumo de API REST, visualización de gráficos con métricas, y una base escalable para futuras extensiones.

Tecnologías Utilizadas

Frontend

- **React 18**: librería base para la interfaz de usuario.
- **React Router DOM**: ruteo SPA entre vistas.
- **React Context API**: gestión de autenticación global.
- **Axios**: consumo de API con interceptores.
- **Chart.js**: visualización de métricas.
- **React Toastify**: notificaciones.
- **React Query**: manejo del estado remoto y cacheo de datos.
- **CSS modular**: estilos separados por componente.

Backend

- **Node.js + Express**: servidor web y lógica de rutas.
- **MongoDB + Mongoose**: base de datos no relacional y ODM.
- **JWT**: autenticación mediante tokens.
- **Multer**: subida de archivos (si aplica).
- **Dotenv**: gestión de variables de entorno.

Estructura del Proyecto

Frontend

```
Frontend/
├── index.js                                  # Punto de entrada React
├── App.js                                    # Configuración de rutas y layout
├── components/                               # Componentes funcionales
│   ├── Login.js, Dashboard.js, CrearTicket.js, ...
├── context/                                  # Manejo de sesión y autenticación
│   └── AuthContext.js
├── services/                                 # Configuración y llamadas Axios
│   └── api.js
```

Backend

```
Backend/
├── app.js                    # Configuración principal de Express y middlewares
├── .env                      # Variables de entorno
├── routes/                   # Definición de rutas por módulo
├── controllers/              # Lógica de negocio
├── models/                   # Esquemas Mongoose
├── middlewares/              # Autenticación, roles, errores, uploads
```

Frontend: Detalles Técnicos

Autenticación

- Se implementa con AuthContext, que se inyecta en el árbol mediante un AuthProvider.
- Los datos del usuario (token, rol, email) se almacenan en localStorage.
- La función login(data) guarda el token y datos, mientras que logout() los elimina.
- En api.js, se configura Axios con un interceptor que adjunta automáticamente el Authorization: Bearer <token>:

```
api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
```

Ruteo

- Declarado en App.js, utilizando React Router v6.
- Rutas con carga perezosa (React.lazy) para rendimiento.
- Estructura condicional basada en roles:

```
<Route path="/dashboard" element={<MainLayout><Dashboard /></MainLayout>} />
<Route path="/adetalleticket/:id" element={<MainLayout><ADetalleTicket /></MainLayout>} />
```

Componentes clave

- Dashboard.js: muestra estadísticas dinámicas según el rol (admin, funcionario, usuario) usando Chart.js.
- CrearTicket.js: formulario para crear tickets y enviarlos al backend.
- ADetalleTicket.js / UDetalleTicket.js: vista de detalle condicional por rol.
- ChatBot.js: permite enviar mensajes a la API de chatbot y renderizar respuestas.
- ErrorBoundary.js: componente de clase para capturar errores de renderizado.

Conexión con backend

- Todas las peticiones deben usar la instancia de Axios exportada por api.js, que ya incluye el baseURL y el interceptor de token.
- Ejemplo de uso:

```
export const getTicketById = async (id) => {
  const response = await api.get(`/tickets/${id}`);
  return response.data;
};
```

Backend: Detalles Técnicos

Arquitectura

- Modelo **MVC clásico**:
    - routes/: define los endpoints.
    - controllers/: contiene funciones que responden a esos endpoints.
    - models/: define los esquemas de datos.
- Modularidad clara: rutas separadas para auth, tickets, users, peticiones, chatbot, events.

Autenticación y Roles

- JWT generado en authController.js.
- Protecciones:
    - authMiddleware: verifica y decodifica el token.
    - roleMiddleware: valida si el usuario tiene permisos para la ruta.
- Ejemplo:

```
router.get("/admin-only", authMiddleware, roleMiddleware("admin"), controller);
```

Modelos de Datos

- User.js

```
nombre: String,
email: { type: String, unique: true },
password: String,
rol: { type: String, enum: ["admin", "funcionario", "usuario"] }
```

- Ticket.js

```
titulo: String,
descripcion: String,
estado: { type: String, enum: ["abierto", "cerrado"] },
categoria: String,
asignadoA: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
creadoPor: { type: mongoose.Schema.Types.ObjectId, ref: "User" }
```

Endpoints Principales

- POST /api/auth/login
- GET /api/tickets – admin obtiene todos
- GET /api/tickets/mis-tickets – usuario obtiene los suyos
- PUT /api/tickets/:id/estado – actualizar estado
- GET /api/users – admins
- POST /api/events – calendario
- POST /api/chatbot/prompt – IA

Manejo de Errores

Backend

- Middleware global errorHandler.js

```
module.exports = (err, req, res, next) => {
  res.status(err.status || 500).json({ error: err.message });
};
```

Frontend

- Uso de console.error() o toast.error() para feedback visual.
- ErrorBoundary captura errores de renderizado que rompen el componente React.

Guía para Desarrolladores

Frontend

```
cd Frontend
npm install
npm start
```

Backend

```
cd Backend
npm install
node app.js
```

.env recomendado

```
PORT=5000
MONGO_URI=mongodb+srv://usuario:clave@host/db
JWT_SECRET=supersecreto123
REACT_APP_API_URL=http://localhost:5000/api
```

Cómo extender el sistema

1. Crear un nuevo modelo Mongoose (models/NuevoModelo.js)
2. Crear controlador (controllers/nuevoController.js)
3. Crear rutas (routes/nuevaRuta.js)
4. En frontend:
    - Crear componente en components/
    - Agregar ruta en App.js
    - Conectar con API usando api.js

Sugerencias de Mejora

- Reemplazar localStorage por cookies seguras HttpOnly.
- Extraer lógica repetida a hooks (useTickets, useUserData, etc).
- Modularizar componentes grandes como Dashboard.js.
- Agregar documentación Swagger.
- Escribir tests unitarios para endpoints y componentes clave.
- Implementar paginación y búsqueda en listados grandes.
- Refinar experiencia de usuario (carga, errores, formularios).