# **Connect BQ**

## **Una Plataforma Inteligente de Movilidad Urbana para Barranquilla**

**Autores:** Sebastian Linero Willman Giraldo Samuel Arenas Santiago Comas Felipe Palmar

**Proyecto Integrador RIWI**

**2025**

# **1. Resumen Ejecutivo**

## **1.1. Propósito del Proyecto**

El propósito de Connect BQ es modernizar la experiencia de movilidad urbana en Barranquilla mediante una plataforma web centralizada. El proyecto busca ofrecer a los ciudadanos una herramienta digital para planificar rutas de transporte público de manera eficiente, recibir alertas en tiempo real sobre rutas por parte de otros usuarios

## **1.2. Planteamiento del Problema**

Actualmente, los ciudadanos de Barranquilla enfrentan desafíos significativos al utilizar el sistema de transporte público. La falta de una fuente de información sobre rutas, paradas imprevistas generan incertidumbre y tiempos de espera prolongados. Además, la ausencia de un canal de comunicación generales por el cual los usuarios reporten alertas a rutas específicas de imprevistos o accidentes

## **1.3. Solución Propuesta: Connect BQ**

Connect BQ es una aplicación web responsiva que aborda estos problemas integrando tres funcionalidades clave: un planificador de rutas multimodal, un sistema de alertas ciudadanas en tiempo real y un sistema de autenticación para funciones personalizadas. La plataforma utiliza las rutas almacenadas para ofrecérselas a sus usarios, permitiendo a los usuarios visualizar paradas, transbordos y tiempos estimados de viaje en un mapa interactivo al igual que ver las alertas en las rutas seleccionadas.

## **1.4. Tecnologías Clave Utilizadas**

El proyecto se desarrolló utilizando un stack tecnológico moderno y escalable. El frontend se construyó con **HTML5, Tailwind CSS y JavaScript (ES6+)**, utilizando la librería **Leaflet.js** para la visualización de mapas. El backend se implementó con **Express** del entorno Node.js para crear una API RESTful eficiente, todo desplegado como funciones serverless. La persistencia de datos se gestionó con una base de datos **Mongo DB** alojada en la nube con **Mongo** **DB** **Atlas.** El despliegue y la integración continua se realizaron a través de **Vercel** y **GitHub**.

## **1.5. Resumen de Resultados**

El resultado final es una plataforma web funcional y que cumple con todos los objetivos propuestos. Se logró desarrollar una biblioteca de rutas con puntos clave, un sistema de alertas ciudadanas funcional y una interfaz de usuario intuitiva y atractiva. La arquitectura serverless garantiza la escalabilidad y un bajo costo de mantenimiento, sentando las bases para futuras mejoras y la integración de nuevas funcionalidades.

# **2. Objetivos**

## **2.1. Objetivo General**

Desarrollar e implementar una plataforma web de movilidad urbana inteligente, llamada Connect BQ, para optimizar la planificación de viajes en transporte público y mejorar la conexión con la información a través de alertas ciudadana en Barranquilla a través de herramientas interactivas y en tiempo real.

## **2.2. Objetivos Específicos**

### **2.2.1. Desarrollar un planificador de rutas multimodal.**

Crear una base de datos recopilando las rutas más relevantes para ser mostradas a los usuarios

### **2.2.2. Implementar un sistema de alertas ciudadanas en tiempo real.**

Habilitar una funcionalidad que permita a los usuarios registrados reportar y visualizar incidentes (ej. congestión, accidentes, bloqueos) directamente en el mapa referente a rutas, fomentando una red de información comunitaria.

### **2.2.3. Habilitar un sistema de autenticación para funcionalidades específicas.**

Implementar un módulo de registro e inicio de sesión seguro para que los usuarios puedan acceder a características personalizadas, como el historial de búsqueda y la capacidad de generar alertas.

### **2.2.4. Diseñar una interfaz de usuario intuitiva y responsiva con modo oscuro.**

Crear un diseño visualmente atractivo, fácil de usar en cualquier dispositivo (móvil, tableta, escritorio) y que incluya una opción de modo oscuro para mejorar la experiencia de usuario en condiciones de poca luz.

### **2.2.5. Desplegar la aplicación en una plataforma moderna y escalable (Vercel).**

Configurar y automatizar el proceso de despliegue continuo (CI/CD) para garantizar que la aplicación esté siempre disponible, sea performante y pueda crecer en función de la demanda de los usuarios.

# **3. Alcance y Limitaciones**

## **3.1. Funcionalidades Dentro del Alcance**

* **Planificación de Rutas:** Rutas a puntos centralizados con opciones de transporte público existente.
* **Visualización en Mapa:** Representación gráfica de rutas, paradas y alertas en un mapa interactivo.
* **Registro e Inicio de Sesión de Usuarios:** Creación de cuentas y autenticación para acceder a funciones protegidas.
* **Sistema de Alertas:** Creación y visualización de alertas ciudadanas por parte de usuarios autenticados.
* **Historial de Búsqueda:** Almacenamiento de las últimas rutas buscadas por el usuario.
* **Diseño Responsivo:** Adaptabilidad completa de la interfaz a dispositivos móviles y de escritorio.
* **Modo Oscuro:** Alternancia entre temas claro y oscuro.

## **3.2. Funcionalidades Fuera del Alcance**

* **Datos de GPS en Tiempo Real:** La versión inicial no integrará datos en vivo de la ubicación de los autobuses.
* **Algoritmo Dijkstra:** Debido al tiempo limitado del proyecto, no se contempló la adecuación de un algoritmo que calcule según nodos la ruta mas rápida.
* **Notificaciones Push:** No se implementarán notificaciones automáticas a los dispositivos de los usuarios.
* **Sistema de Pagos o Recargas:** La plataforma no incluirá funcionalidades de transacciones monetarias.
* **Reservas de Asientos o Billetes:** No se podrá comprar ni reservar pasajes a través de la aplicación.
* **Aplicación Móvil Nativa:** El proyecto se centra exclusivamente en una aplicación web (no se desarrollará para iOS o Android de forma nativa).

## **3.3. Limitaciones del Proyecto**

* **Búsqueda de los Datos:**  Al no tener acceso a las rutas por parte de la AMB , se hizo de manera manual con una cantidad mínima de rutas.
* **Conocimiento Sobre Algoritmo Dijkstra:** Al no contemplar la dificultad de la implementación con las herramientas base se decidió no acoger el algoritmo.
* **Recursos de Equipo:** El desarrollo fue realizado por un equipo 5 Developers Junior en 4 semanas.
* **Dependencia de Servicios Externos:** El rendimiento de la aplicación depende de servicios de terceros como Vercel (hosting) y el proveedor de la base de datos en la nube en este caso Mongo con Atlas.

# **4. Metodología de Desarrollo: SCRUM**

## **4.1. Justificación de la Elección de SCRUM**

Se seleccionó una metodología híbrida entre SCRUM y Kanban debido a su naturaleza ágil e iterativa, ideal para un proyecto con requisitos que pueden evolucionar. SCRUM nos permitió adaptarnos rápidamente a los cambios, fomentar la colaboración constante dentro del equipo y entregar valor de manera incremental en ciclos cortos llamados *Sprints*. Este enfoque facilitó la identificación temprana de problemas y la priorización efectiva de tareas, asegurando un progreso constante y visible.

## **4.2. Composición y Roles del Equipo**

### **4.2.1. Equipo de Desarrollo (2 Frontend, 2 Backend)**

* **Desarrolladores Frontend:** Responsables de implementar la interfaz de usuario, la lógica del lado del cliente, la interactividad del mapa y la conexión con la API del backend.
* **Desarrolladores Backend:** Encargados de diseñar y desarrollar la API RESTful, la lógica de negocio (recopilación de rutas, gestión de alertas), la autenticación y la comunicación con la base de datos.

### **4.2.2. Gestión del Doble Rol: Product Owner y Scrum Master**

En este proyecto, una sola persona asumió los roles de Product Owner (PO) y Scrum Master (SM) simultáneamente por la magnitud del proyecto elegido igualmente apoyándose en un Developer(con el rol SM) para garantizar la parcialidad y seguimiento de la metodología.

#### **4.2.2.1. Definición y responsabilidades de cada rol.**

* **Product Owner (PO):** Responsable de definir la visión del producto, gestionar el *Product Backlog*, priorizar las funcionalidades y asegurar que el equipo de desarrollo entregue el máximo valor a los usuarios.
* **Scrum Master (SM):** Responsable de facilitar los eventos de SCRUM, eliminar impedimentos que bloqueen al equipo, y asegurar que el equipo siga las prácticas y principios de la metodología.

#### **4.2.2.2. Estrategias para mitigar conflictos de interés y carga de trabajo.**

* **Separación de "Sombreros":** Se estableció una clara distinción de cuándo se actuaba como PO (definiendo el "qué" y el "porqué") y cuándo como SM (enfocándose en el "cómo" y optimizando el proceso).
* **Transparencia Radical:** Todas las decisiones de priorización (rol de PO) y las acciones para eliminar impedimentos (rol de SM) se comunicaron abiertamente al equipo para evitar percepciones de sesgo.
* **Empoderamiento del Equipo:** Se fomentó que el equipo de desarrollo tomara un papel más activo en la auto-gestión y en la resolución de problemas técnicos, reduciendo la dependencia directa del SM.

#### **4.2.2.3. Impacto en la dinámica del equipo y lecciones aprendidas.**

Esta dualidad de roles presentó un desafío, especialmente en la gestión del tiempo. Sin embargo, también agilizó la toma de decisiones al contar con un único punto de contacto para la visión del producto y la gestión del proceso. La lección principal fue la importancia de la disciplina y la comunicación para mantener el equilibrio entre la dirección del producto y la salud del proceso ágil.

## **4.3. Eventos SCRUM**

* **Sprint Planning:** Reuniones al inicio de cada Sprint (de 1 semanas de duración) para seleccionar los ítems del *Product Backlog* que se trabajarían.
* **Daily Scrums:** Reuniones diarias de 15 minutos para sincronizar al equipo, discutir el progreso y identificar impedimentos.
* **Sprint Review:** Sesión al final del Sprint para demostrar el incremento de producto funcional a los stakeholders y obtener feedback.
* **Sprint Retrospective:** Reunión interna del equipo para reflexionar sobre el Sprint, identificar qué funcionó bien y qué se puede mejorar en el siguiente ciclo.

## **4.4. Herramientas de Gestión**

Se utilizó **GitHub Projects** como herramienta principal para la gestión del *Product Backlog* y el seguimiento del progreso durante los Sprints. Las *issues* de GitHub se usaron para representar las historias de usuario y tareas, organizadas en un tablero Kanban (To Do, In Progress, Done).

# **5. Arquitectura del Sistema**

## **5.1. Arquitectura General (Diagrama de Componentes)**

La arquitectura de Connect BQ se basa en un modelo desacoplado que separa claramente el frontend del backend. El cliente (navegador web) interactúa con una aplicación de página estática (MPA) que, a su vez, consume una API RESTful. La base de datos opera como un servicio independiente en la nube.

Diagrama

El contenido generado por IA puede ser incorrecto.

## **5.2. Arquitectura Frontend**

### **5.2.1. Sitio Estático Servido por**

El frontend es una aplicación web estática compuesta por archivos HTML, Tailwind CSS y JavaScript. Vercel se encarga de servir estos archivos a través de su red de distribución de contenido (CDN) global, lo que garantiza tiempos de carga rápidos y una alta disponibilidad.

### **5.2.2. Interacción con la API (Flujo de Datos)**

El JavaScript del cliente realiza llamadas asíncronas (usando fetch) a los endpoints de la API del backend para obtener datos (ej. Mostrar una ruta, cargar alertas) o para enviar información (ej. registrar un usuario, crear una alerta). Los datos se intercambian en formato JSON.

## **5.3. Arquitectura Backend**

### **5.3.1. Funciones Serverless en Vercel**

El backend no se ejecuta en un servidor tradicional siempre activo. En su lugar, cada endpoint de la API está implementado como una función serverless (Lambda) independiente. Vercel gestiona automáticamente la ejecución de estas funciones, escalándolas según la demanda, lo que optimiza costos y reduce la sobrecarga de mantenimiento.

### **5.3.2. Lógica de Negocio y Conexión a la Base de Datos**

Las funciones serverless, escritas en JavaScript con Express, contienen toda la lógica de negocio. Esto incluye la validación de datos de entrada, el procesamiento de peticiones de rutas, la autenticación de usuarios y la gestión de la conexión con la base de datos Mongo DB para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

## **5.4. Arquitectura de Datos**

### **5.4.1. Base de Datos MongoDB Alojada en la Nube (Mongo DB Atlas)**

Se optó por una base de datos no relacional NoSQL gestionada en una plataforma en la nube como Mongo DB Atlas. Este enfoque ofrece ventajas como escalabilidad automática, copias de seguridad gestionadas y alta disponibilidad, liberando al equipo de las tareas de administración de la base de datos.

A pesar de la dependencia de servicios de terceros, se mitigaron los riesgos asociados a la infraestructura de varias maneras. **Vercel** y **MongoDB Atlas** son plataformas líderes que ofrecen **alta disponibilidad**, **escalabilidad automática** y **copias de seguridad gestionadas**, lo que minimiza el riesgo de caídas del servicio y pérdida de datos.

# **6. Stack Tecnológico**

## **6.1. Frontend**

* **Lenguajes:** HTML5, JavaScript (ES6+).
* **Estilos:** Tailwind CSS para un diseño basado en utilidades y responsivo.
* **Mapas:** Leaflet.js, una librería de código abierto para mapas interactivos.

## **6.2. Backend**

* **Lenguaje:** JavaScipt
* Libreria: Express

## **6.3. Base de Datos**

* **Sistema Gestor:** Mongo DB

## **6.4. Despliegue y DevOps**

* **Plataforma:** Vercel para hosting de frontend y ejecución de funciones serverless.
* **Control de Versiones:** Git.
* **Repositorio:** GitHub para el almacenamiento del código fuente y la integración continua.

# **7. Diseño y Modelado de la Base de Datos**

## 7.1. Modelo de Datos (Diagrama de Clases/Documentos)

El modelo de datos se diseñó para representar las entidades clave de la aplicación: **usuarios**, **rutas**, **paradas**, **alertas** e **historial de búsqueda**. A diferencia de un modelo Entidad-Relación tradicional, este se enfoca en la estructura de los **documentos** dentro de las **colecciones** de MongoDB, lo que permite un esquema más flexible y relaciones incrustadas.

En este diagrama, cada caja representa una **colección**, y las líneas indican las relaciones entre ellas. Por ejemplo, en el documento de Routes podemos ver que el arreglo de stops está **incrustado** en el documento principal, lo que es una práctica común en MongoDB para optimizar las consultas.

## 7.2. Diccionario de Datos

Este apartado detalla el esquema de las colecciones que conforman la base de datos de la plataforma.

## 7.2.1. Colección users

Almacena la información de los usuarios registrados. A diferencia de las bases de datos relacionales, cada usuario se almacena como un **documento** en una colección.

| Campo | Tipo de Dato | Descripción | Clave |
| --- | --- | --- | --- |
| \_id | ObjectId | Identificador único del documento, generado automáticamente por MongoDB. | PK |
| username | String | Nombre de usuario, debe ser único para cada documento. |  |
| email | String | Correo electrónico, también debe ser único. |  |
| password | String | Contraseña del usario. |  |
| created\_at | ISODate | Fecha y hora en que se creó el documento. |  |

## 7.2.2. Colección routes

Contiene la información sobre las rutas de transporte. La flexibilidad de MongoDB permite incrustar los documentos de paradas (stops) directamente dentro de cada documento de ruta.

| Campo | Tipo de Dato | Descripción | Clave |
| --- | --- | --- | --- |
| \_id | ObjectId | Identificador único de la ruta. | PK |
| name | String | Nombre o código de la ruta (ej. "A7-3"). |  |
| path\_geojson | Object | Objeto GeoJSON que representa la geometría de la ruta. |  |
| Tiempo estimado | float | Tiempo estimado en llegar al punto de destino. |  |
| Precio estimado | Float |  |  |
| stops | Array of Objects | Un arreglo de documentos incrustados que representan las paradas de la ruta. |  |

Estructura de stops (dentro de routes):

| Campo | Tipo de Dato | Descripción | Clave |

| :--- | :--- | :--- | :--- |

| name | String | Nombre de la parada. | |

| location | Object | Coordenadas de la parada en formato GeoJSON (Point). | |

## 7.2.2.1 Tipo De Dato alerts

Registra las alertas ciudadanas creadas por los usuarios. Los documentos de alerta referencian al usuario que las creó mediante su \_id.

| Campo | Tipo de Dato | Descripción | Clave |
| --- | --- | --- | --- |
| \_id | ObjectId | Identificador único de la alerta. | PK |
| user\_id | ObjectId | Referencia al \_id del usuario que creó la alerta. | FK (users.\_id) |
| alert\_type | String | Tipo de alerta (ej. "Tráfico", "Seguridad"). |  |
| description | String | Descripción detallada de la alerta. |  |
| location | Object | Coordenadas del incidente en formato GeoJSON (Point). |  |
| created\_at | ISODate | Fecha y hora de creación de la alerta. |  |

## 7.2.1.2 Tipo de Dato search\_history

Guarda el historial de búsquedas de rutas para cada usuario. Cada documento de búsqueda hace referencia al usuario que la realizó.

| Campo | Tipo de Dato | Descripción | Clave |
| --- | --- | --- | --- |
| \_id | ObjectId | Identificador único del registro de búsqueda. | PK |
| user\_id | ObjectId | Referencia al \_id del usuario que realizó la búsqueda. | FK (users.\_id) |
| searched\_at | ISODate | Fecha y hora de la búsqueda. |  |

## **8. Implementación de Módulos y Funcionalidades**

## **8.1. Módulo de Autenticación de Usuarios**

### **8.1.1. Frontend: Formularios de Registro e Inicio de Sesión.**

Se crearon componentes de interfaz de usuario utilizando HTML y Tailwind CSS para los formularios de registro y login. La lógica en JavaScript se encarga de capturar los datos del usuario, validarlos en el cliente y enviarlos al backend a través de una petición POST.

## **8.2. Módulo de Planificación de Rutas**

### **8.2.1. Frontend: Interfaz de Mapa y Visualización de Rutas.**

La interfaz principal muestra un mapa interactivo de Barranquilla usando Leaflet.js. Incluye campos de entrada para el origen y el destino. Al recibir la respuesta del backend, el JavaScript del cliente dibuja la ruta sugerida, las paradas y cualquier transbordo necesario sobre el mapa.

### **8.2.2. Backend: Endpoint y Algoritmo de Cálculo.**

* GET /Routes/: Este endpoint obtiene las rutas transporte alamacenadas y muestra las paradas y alertas relacionadas.

## **8.3. Módulo de Alertas Ciudadanas**

### **8.3.1. Frontend: Formulario de Reporte y Visualización.**

Los usuarios autenticados tienen acceso a un formulario para reportar una nueva alerta, seleccionando el tipo, añadiendo una descripción y marcando la ubicación en el mapa. Las alertas existentes, obtenidas del backend, se muestran como marcadores en el mapa.

### **8.3.2. Backend: Endpoints y Validación de Usuario.**

* GET /api/alerts: Devuelve una lista de todas las alertas activas para ser mostradas en el mapa.
* POST /api/alerts: Recibe los datos de una nueva alerta. Antes de procesarla, verifica que la solicitud contenga un token de autenticación válido. Si el usuario está autenticado, guarda la nueva alerta en la base de datos.

## **8.4. Funcionalidades Adicionales**

## **8.4.2. Historial de Búsqueda**

Para los usuarios autenticados, cada búsqueda de ruta exitosa se guarda en la tabla search\_history. Esta información se muestra en un apartado del perfil del usuario, permitiéndole repetir búsquedas frecuentes con un solo clic.

### **8.4.3. Implementación de Modo Oscuro con Tailwind CSS**

Aprovechando las capacidades de Tailwind CSS, se implementó un sistema de temas. Utilizando la estrategia de clase (darkMode: 'class'), se aplicaron estilos condicionales (dark:\*) que se activan cuando la etiqueta <html> tiene la clase dark, controlada por un botón en la interfaz.

# **9. Despliegue en Producción**

## **9.1. Configuración del Proyecto en Vercel**

El proyecto se conectó al repositorio de GitHub desde el dashboard de Vercel. Se configuró el framework correspondiente (ej. "Other" para un sitio estático) y se definieron los comandos de construcción y el directorio de salida. Vercel detecta automáticamente el código del frontend y las funciones serverless del backend.

## **9.2. Gestión de Variables de Entorno**

Las credenciales sensibles, como la cadena de conexión a la base de datos y las claves secretas para la generación de tokens, no se almacenaron en el código fuente. En su lugar, se configuraron como variables de entorno directamente en la configuración del proyecto en Vercel, asegurando que solo sean accesibles en el entorno de producción.

## **9.3. Proceso de Integración y Despliegue Continuo (CI/CD)**

Vercel automatiza el proceso de CI/CD. Cada vez que se realiza un git push a la rama principal (main), Vercel activa automáticamente un nuevo despliegue: clona el repositorio, instala las dependencias, ejecuta el proceso de construcción y, si tiene éxito, actualiza la aplicación en producción sin tiempo de inactividad.

# **10. Pruebas y Aseguramiento de Calidad (QA)**

## **10.1. Pruebas de Endpoints con Postman**

Se utilizó Postman para probar cada uno de los endpoints de la API de forma aislada. Se verificó el correcto funcionamiento de los métodos (GET, POST), los códigos de estado de respuesta (200, 201, 400, 403), y la estructura y contenido de las respuestas en JSON para casos exitosos y de error.

## **10.2. Pruebas Manuales de Flujos de Usuario End-to-End**

Se realizaron pruebas manuales completas simulando la experiencia de un usuario final. Esto incluyó los siguientes flujos:

* Registro de un nuevo usuario.
* Inicio y cierre de sesión.
* Búsqueda de una ruta y muestra correctamente.
* Creación de una nueva alerta.
* Activación del modo oscuro.

## **10.3. Pruebas de Responsividad y Compatibilidad entre Navegadores**

Se verificó que la interfaz de usuario se visualizara y funcionara correctamente en diferentes tamaños de pantalla, desde dispositivos móviles pequeños hasta monitores de escritorio grandes. Además, se comprobó la compatibilidad en las últimas versiones de los principales navegadores web (Chrome, Brave,Firefox).

## **10.4. Pruebas Unitarias y de Integración con Jest**

Para garantizar la fiabilidad del **Backend**, se implementó una suite de **pruebas automatizadas** usando el framework **Jest**. Estas pruebas se centraron en validar las **reglas de negocio** de los controladores y la correcta manipulación de datos en los modelos. El objetivo fue asegurar que las funciones individuales se comportaran como se esperaba y que las interacciones entre los diferentes módulos (ej. controladores, modelos) funcionaran sin errores. La suite de pruebas incluyó los siguientes casos:

* **database.test.js**: Pruebas para la conexión y la configuración de la base de datos, verificando que la aplicación pueda establecer y mantener una conexión estable.
* **route-controller.test.js**: Pruebas de integración para los controladores de rutas, validando que manejen correctamente las solicitudes HTTP y que la lógica de negocio se ejecute de forma precisa antes de enviar la respuesta al cliente.
* **route-model.test.js**: Pruebas unitarias para los modelos de datos de las rutas, asegurando que las funciones de validación y manipulación de datos funcionen correctamente.
* **user-controller.test.js**: Pruebas para los controladores de usuarios, verificando que los procesos de autenticación, registro y gestión de perfiles funcionen como se diseñó.
* **user.test.js**: Pruebas unitarias para el modelo de usuario, centradas en la lógica de negocio como la validación del formato del correo electrónico, la fortaleza de la contraseña y la correcta manipulación de la información del usuario.

El uso de **Jest** nos permitió detectar errores de forma temprana, facilitar el mantenimiento del código y asegurar que cada nueva funcionalidad o cambio en el sistema no introdujera errores en componentes ya existentes. para un proceso de **desarrollo ágil** y un **aseguramiento de calidad (QA)** robusto

# **11. Conclusiones**

## **11.1. Logro de Objetivos**

El proyecto Connect BQ cumplió exitosamente con todos los objetivos generales y específicos planteados. Se entregó una plataforma web completamente funcional que integra un planificador de rutas, un sistema de alertas y autenticación de usuarios, todo ello desplegado en una infraestructura moderna y escalable, y con una interfaz de usuario intuitiva y atractiva.

## **11.2. Desafíos Encontrados y Soluciones Aplicadas**

* **Desafío:** Acceder a la API con las rutas oficiales de la AMB para incluirlas en el proyecto y asegurar que concordaran con la realidad.
* **Solución:** Se ayudaron con la conversión de los datos KMZ a latitud y longitud manualmente por aplicaciones como Qruta.
* **Desafío:** Gestionar el estado de la aplicación en el frontend de manera coherente, especialmente la autenticación del usuario.
* **Solución:** Se adoptó un enfoque simple de gestión de estado utilizando el localStorage del navegador para persistir el token de autenticación y un contexto global en JavaScript para compartir el estado del usuario entre componentes.

## **11.3. Lecciones Aprendidas**

La principal lección aprendida fue la importancia de un diseño de API bien definido desde el inicio del proyecto. Tener un "contrato" claro entre el frontend y el backend facilitó el trabajo en paralelo y redujo los problemas de integración. Además, la elección de una arquitectura serverless demostró ser altamente beneficiosa en términos de costos y facilidad de despliegue.

# **12. Trabajo Futuro y Posibles Mejoras**

## **12.1. Implementación de Algoritmo de Rutas**

Desarrollar con más conocimiento el algoritmo de rutas el cual permita poner el punto de partida y de destino y por medio de la base de datos calcular las rutas más optimas, rápidas o alternativa.

## **12.2. Integración con Datos en Tiempo Real (GPS de Autobuses)**

Integrar la opción que por medio de los usuarios en tiempo real proporcionen la ubicación GPS de los autobuses. Esto permitiría mostrar la posición actual de los vehículos en el mapa y ofrecer estimaciones de llegada mucho más precisas a otros usuarios.

## **12.3. Desarrollo de una Progressive Web App (PWA)**

Mejorar la aplicación web para convertirla en una PWA, lo que permitiría a los usuarios "instalarla" en la pantalla de inicio de sus dispositivos móviles, acceder a ciertas funcionalidades sin conexión y, eventualmente, recibir notificaciones push de alertas o estimaciones de llegada de sus buses.

## **12.4. Alianza Estratégica con la AMB para Datos Oficiales**

Para llevar la precisión de la plataforma al siguiente nivel, una mejora futura clave es establecer una alianza estratégica con la **Autoridad Metropolitana de Barranquilla** (AMB). Esto permitiría a **Connect BQ** acceder a una API con las rutas oficiales de toda el área metropolitana de la ciudad. Esta colaboración aseguraría la regularización y **actualización** constante de las rutas, paradas y horarios, eliminando la necesidad de la recopilación manual de datos y garantizando que la información de la plataforma sea siempre precisa y fiable para los usuarios

# **13. Apéndices**

## **13.1. Contrato de la API**

# 13.1. Contrato de la API

A continuación, se detalla la documentación completa de cada endpoint de la API, incluyendo sus especificaciones técnicas y ejemplos de uso.

## Gestión de Usuarios

### POST /users

Método: POST  
Descripción: Crea un nuevo usuario en el sistema.  
Autenticación: No requerida.  
Cuerpo de la Petición:

{  
 "identityNumber": "string",  
 "username": "string",  
 "password": "string",  
 "email": "string",  
 "phone": "string",  
 "age": "number"  
}

Respuesta Exitosa (201 Created):

{  
 "message": "Usuario creado exitosamente",  
 "user": {  
 "id": "ObjectId",  
 "identityNumber": "string",  
 "username": "string",  
 "email": "string",  
 "phone": "string",  
 "age": "number",  
 "routesHistory": [],  
 "favoritesRoutes": [],  
 "createdAt": "date",  
 "updatedAt": "date"  
 }  
}

Respuesta de Error (400 Bad Request):

{  
 "message": "Los datos de entrada no son válidos."  
}

### GET /users

Método: GET  
Descripción: Obtiene la lista de todos los usuarios registrados en el sistema.  
Autenticación: No requerida.  
Respuesta Exitosa (200 OK):

[  
 {  
 "id": "ObjectId",  
 "identityNumber": "string",  
 "username": "string",  
 "email": "string",  
 "phone": "string",  
 "age": "number",  
 "routesHistory": [],  
 "favoritesRoutes": [],  
 "createdAt": "date",  
 "updatedAt": "date"  
 }  
]

### GET /users/:id

Método: GET  
Descripción: Obtiene la información de un usuario específico por su ID.  
Parámetros: id (string, en la URL).  
Autenticación: Requerida (a través de localStorage).  
Respuesta Exitosa (200 OK):

{  
 "id": "ObjectId",  
 "identityNumber": "string",  
 "username": "string",  
 "email": "string",  
 "phone": "string",  
 "age": "number",  
 "routesHistory": [],  
 "favoritesRoutes": [],  
 "createdAt": "date",  
 "updatedAt": "date"  
}

Respuesta de Error (404 Not Found):

{  
 "message": "Usuario no encontrado."  
}

## Gestión de Rutas

### GET /routes

Método: GET  
Descripción: Obtiene todas las rutas disponibles.  
Autenticación: No requerida.  
Respuesta Exitosa (200 OK):

[  
 {  
 "id": "ObjectId",  
 "name": "string",  
 "initialPoint": {  
 "name": "string",  
 "coordinates": {  
 "latitude": "number",  
 "longitude": "number"  
 }  
 },  
 "endPoint": {  
 "name": "string",  
 "coordinates": {  
 "latitude": "number",  
 "longitude": "number"  
 }  
 },  
 "distance": "decimal",  
 "estimatedTime": "decimal",  
 "estimatedCost": "decimal",  
 "path": [],  
 "alerts": []  
 }  
]

### GET /routes/:id

Método: GET  
Descripción: Obtiene los detalles de una ruta específica por su ID.  
Parámetros: id (string, en la URL).  
Autenticación:No requerida.  
Respuesta Exitosa (200 OK):

{  
 "id": "ObjectId",  
 "name": "string",  
 "initialPoint": {  
 "name": "string",  
 "coordinates": {  
 "latitude": "number",  
 "longitude": "number"  
 }  
 },  
 "endPoint": {  
 "name": "string",  
 "coordinates": {  
 "latitude": "number",  
 "longitude": "number"  
 }  
 },  
 "distance": "decimal",  
 "estimatedTime": "decimal",  
 "estimatedCost": "decimal",  
 "path": [],  
 "alerts": []  
}

Respuesta de Error (404 Not Found):

{  
 "message": "Ruta no encontrada."  
}

## Gestión de Alertas

### POST /alerts

Método: POST  
Descripción: Crea una nueva alerta en el sistema.  
Autenticación: Requerida (el usuario debe estar logeado y tener la sesión activa en localStorage).  
Cuerpo de la Petición:

{  
 "alertType": "string",  
 "description": "string",  
 "location": {  
 "latitude": "number",  
 "longitude": "number"  
 }  
}

Respuesta Exitosa (201 Created):

{  
 "message": "Alerta creada exitosamente",  
 "alert": {  
 "id": "ObjectId",  
 "userId": "string",  
 "alertType": "string",  
 "description": "string",  
 "location": {  
 "latitude": "number",  
 "longitude": "number"  
 },  
 "createdAt": "date"  
 }  
}

Respuesta de Error (401 Unauthorized):

{  
 "message": "No autorizado. Usuario no logeado o sesión no válida."  
}

Respuesta de Error (400 Bad Request):

{  
 "message": "Los datos de la alerta no son válidos."  
}

### GET /alerts

Método: GET  
Descripción: Obtiene todas las alertas activas en el sistema.  
Autenticación: No requerida.  
Respuesta Exitosa (200 OK):

[  
 {  
 "id": "ObjectId",  
 "userId": "string",  
 "alertType": "string",  
 "description": "string",  
 "location": {  
 "latitude": "number",  
 "longitude": "number"  
 },  
 "createdAt": "date"  
 }  
]

Respuesta de Error (404 Not Found):

{  
 "message": "No se encontraron alertas."  
}

## **13.2. Manual de Usuario Básico**

**Ejemplo:** **Cómo ver una ruta:**

**Ejemplo:** **Como poner una alerta:**

1.incia sesión

2.ve al dashboard

3.