# CONNECT BQ

# An Intelligent Urban Mobility Platform for Barranquilla

# Authors:

## Sebastián Linero De Castro.

## Willman Giraldo Zambrano.

## Samuel Arenas Jiménez.

## Santiago Comas Duran.

## Felipe Palmar Ramírez.

# RIWI Integration Project

# 2025

# Executive Summary

## Project Purpose

The purpose of Connect BQ is to modernize the urban mobility experience in Barranquilla through a centralized web platform. The project aims to offer citizens a digital tool to efficiently plan public transport routes and receive real-time alerts about routes from other users.

## 1.2. Problem Statement

Currently, the citizens of Barranquilla face significant challenges when using the public transport system. The lack of an information source on routes and unforeseen stops generates uncertainty and long waiting times. Additionally, there is an absence of a general communication channel for users to report alerts on specific routes regarding unforeseen events or accidents.

## 1.3. Proposed Solution: Connect BQ

Connect BQ is a responsive web application that addresses these problems by integrating three key functionalities: a multimodal route planner, a real-time citizen alert system, and an authentication system for personalized features. The platform uses stored routes to offer them to its users, allowing them to view stops, transfers, and estimated travel times on an interactive map, as well as see alerts on selected routes.

## 1.4. Key Technologies Used

The project was developed using a modern and scalable technology stack8. The frontend was built with HTML5, Tailwind CSS, and JavaScript (ES6+), using the Leaflet.js library for map visualization. The backend was implemented with Express from the Node.js environment to create an efficient RESTful API, all deployed as serverless functions. Data persistence was managed with a MongoDB database hosted in the cloud with MongoDB Atlas. Deployment and continuous integration were carried out through Vercel and GitHub.

## 1.5. Summary of Results

The final result is a functional web platform that meets all the proposed objectives. A library of routes with key points, a functional citizen alert system, and an intuitive and attractive user interface were successfully developed. The serverless architecture ensures scalability and low maintenance costs, laying the foundation for future improvements and the integration of new functionalities.

# 2. Objectives

## 2.1. General Objective

To develop and implement an intelligent urban mobility web platform, called Connect BQ, to optimize travel planning on public transport and improve the connection with information through citizen alerts in Barranquilla, using interactive and real-time tools.

## 2.2. Specific Objectives

### 2.2.1. Develop a multimodal route planner.

* + Create a database by collecting the most relevant routes to be displayed to users.

### 2.2.2. Implement a real-time citizen alert system.

* + Enable a functionality that allows registered users to report and view incidents (e.g., congestion, accidents, blockages) directly on the map related to routes, fostering a community information network.

### 2.2.3. Enable an authentication system for specific functionalities.

* + Implement a secure registration and login module so that users can access personalized features, such as search history and the ability to generate alerts.

### 2.2.4. Design an intuitive and responsive user interface with a dark mode.

* + Create a visually attractive design that is easy to use on any device (mobile, tablet, desktop) and includes a dark mode option to improve the user experience in low-light conditions.

### 2.2.5. Deploy the application on a modern and scalable platform (Vercel).

* + Configure and automate the continuous deployment (CI/CD) process to ensure the application is always available, performant, and can grow based on user demand.

# 3. Scope and Limitations

## 3.1. In-Scope Functionalities

* **Route Planning:** Routes to centralized points with existing public transport options.
* **Map Visualization:** Graphical representation of routes, stops, and alerts on an interactive map.
* **User Registration and Login:** Account creation and authentication to access protected features.
* **Alert System:** Creation and viewing of citizen alerts by authenticated users.
* **Search History:** Storage of the user's last search routes.
* **Responsive Design:** Full adaptability of the interface to mobile and desktop devices.
* **Dark Mode:** Toggling between light and dark themes.

## 3.2. Out-of-Scope Functionalities

* **Real-Time GPS Data:** The initial version will not integrate live data of bus locations.
* **Dijkstra's Algorithm:** Due to the limited project time, the adaptation of an algorithm that calculates the fastest route based on nodes was not contemplated35.
* **Push Notifications:** Automatic notifications to users' devices will not be implemented36.
* **Payment or Top-up System:** The platform will not include monetary transaction functionalities37.
* **Seat or Ticket Reservations:** It will not be possible to buy or reserve tickets through application.
* **Native Mobile Application:** The project focuses exclusively on a web application (it will not be developed natively for iOS or Android).

## 3.3. Project Limitations.

* **Data Retrieval:** As there was no access to the routes from the AMB, it was done manually with a minimum number of routes.
* **Knowledge about Dijkstra's Algorithm:** As the difficulty of implementation with the base tools was not contemplated, it was decided not to adopt the algorithm.
* **Team Resources:** The development was carried out by a team of 5 Junior Developers in 4 weeks.
* **Dependence on External Services:** The application's performance depends on third-party services like Vercel (hosting) and the cloud database provider, in this case, Mongo with Atlas.

# 4. Development Methodology: SCRUM

## 4.1. Justification for Choosing SCRUM

A hybrid methodology between SCRUM and Kanban was selected due to its agile and iterative nature, ideal for a project with requirements that may evolve. SCRUM allowed us to adapt quickly to changes, foster constant collaboration within the team, and deliver value incrementally in short cycles called Sprints. This approach facilitated the early identification of problems and the effective prioritization of tasks, ensuring constant and visible progress.

## 4.2. Team Composition and Roles

### 4.2.1. Development Team (2 Frontend, 2 Backend)

* **Frontend Developers:** Responsible for implementing the user interface, client-side logic, map interactivity, and connection with the backend API.
* **Backend Developers:** In charge of designing and developing the RESTful API, business logic (route collection, alert management), authentication, and communication with the database.

### 4.2.2. Management of the Dual Role: Product Owner and Scrum Master.

In this project, a single person assumed the roles of Product Owner (PO) and Scrum Master (SM) simultaneously due to the magnitude of the chosen project, also relying on a Developer (with the SM role) to ensure impartiality and adherence to the methodology.

### 4.2.2.1. Definition and responsibilities of each role.

* + **Product Owner (PO):** Responsible for defining the product vision, managing the Product Backlog, prioritizing functionalities, and ensuring that the development team delivers maximum value to the users.
  + **Scrum Master (SM):** Responsible for facilitating SCRUM events, removing impediments that block the team, and ensuring that the team follows the practices and principles of the methodology.

### 4.2.2.2. Strategies to mitigate conflicts of interest and workload.

* + **Separation of "Hats":** A clear distinction was established for when acting as PO (defining the "what" and "why") and when as SM (focusing on the "how" and optimizing the process).
  + **Radical Transparency:** All prioritization decisions (PO role) and actions to remove impediments (SM role) were communicated openly to the team to avoid perceptions of bias.
  + **Team Empowerment:** The development team was encouraged to take a more active role in self-management and in solving technical problems, reducing direct dependence on the SM.

### 4.2.2.3. Impact on team dynamics and lessons learned.

This duality of roles presented a challenge, especially in time management60. However, it also streamlined decision-making by having a single point of contact for product vision and process management. The main lesson was the importance of discipline and communication to maintain a balance between product direction and the health of the agile process.

## 4.3. SCRUM Events

* **Sprint Planning:** Meetings at the beginning of each Sprint (1-week duration) to select the items from the Product Backlog that would be worked on63.
* **Daily Scrums:** Daily 15-minute meetings to synchronize the team, discuss progress, and identify impediments.
* **Sprint Review:** A session at the end of the Sprint to demonstrate the functional product increment to stakeholders and get feedback.
* **Sprint Retrospective:** An internal team meeting to reflect on the Sprint, identify what worked well, and what can be improved in the next cycle.

## 4.4. Management Tools

GitHub Projects was used as the main tool for managing the Product Backlog and tracking progress during the Sprints. GitHub issues were used to represent user stories and tasks, organized on a Kanban board (To Do, In Progress, Done).

# 5. System Architecture

## 5.1. General Architecture (Component Diagram)

The architecture of Connect BQ is based on a decoupled model that clearly separates the frontend from the backend. The client (web browser) interacts with a static page application (MPA) which, in turn, consumes a RESTful API. The database operates as an independent service in the cloud.

Diagrama

El contenido generado por IA puede ser incorrecto.

## 5.2. Frontend Architecture

### 5.2.1. Static Site Served by Vercel

The frontend is a static web application composed of HTML, Tailwind CSS, and JavaScript files. Vercel is in charge of serving these files through its global content delivery network (CDN), which guarantees fast loading times and high availability.

### 5.2.2. Interaction with the API (Data Flow)

The client-side JavaScript makes asynchronous calls (using

fetch) to the backend API endpoints to obtain data (e.g., display a route, load alerts) or to send information (e.g., register a user, create an alert). Data is exchanged in JSON format.

## 5.3. Backend Architecture

### 5.3.1. Serverless Functions in Vercel.

The backend does not run on a traditional always-on server. Instead, each API endpoint is implemented as an independent serverless function (Lambda). Vercel automatically manages the execution of these functions, scaling them according to demand, which optimizes costs and reduces maintenance overhead.

### 5.3.2. Business Logic and Database Connection

The serverless functions, written in JavaScript with Express, contain all the business logic. This includes input data validation, processing of route requests, user authentication, and management of the connection with the MongoDB database to perform CRUD (Create, Read, Update, Delete) operations.

## 5.4. Data Architecture

### 5.4.1. MongoDB Database Hosted in the Cloud (MongoDB Atlas)

A non-relational NoSQL database managed on a cloud platform such as MongoDB Atlas was chosen. This approach offers advantages such as automatic scalability, managed backups, and high availability, freeing the team from database administration tasks.

Despite the dependence on third-party services, the risks associated with the infrastructure were mitigated in several ways. Vercel and MongoDB Atlas are leading platforms that offer high availability, automatic scalability, and managed backups, which minimizes the risk of service outages and data loss.

# 6. Technology Stack

## 6.1. Frontend

* **Languages:** HTML5, JavaScript (ES6+).
* **Styles:** Tailwind CSS for a utility-based and responsive design.
* **Maps:** Leaflet.js, an open-source library for interactive maps.

## 6.2. Backend

* **Language:** JavaScript.
* **Library:** Express.

## 6.3. Database

* **Management System:** MongoDB.

## 6.4. Deployment and DevOps

* **Platform:** Vercel for frontend hosting and execution of serverless functions.
* **Version Control:** Git.
* **Repository:** GitHub for source code storage and continuous integration.

# 7. Database Design and Modeling

## 7.1. Data Model (Class/Document Diagram)

The data model was designed to represent the key entities of the application: users, routes, stops, alerts, and search history. Unlike a traditional Entity-Relationship model, this one focuses on the structure of the documents within the MongoDB collections, which allows for a more flexible schema and embedded relationships.

Diagrama

El contenido generado por IA puede ser incorrecto.

In this diagram, each box represents a collection, and the lines indicate the relationships between them. For example, in the Routes document, we can see that the. Stops array is embedded in the main document, which is a common practice in MongoDB to optimize queries.

## 7.2. Data Dictionary

This section details the schema of the collections that make up the platform's database.

### 7.2.1. users Collection

Stores the information of registered users. Unlike relational databases, each user is stored as a document in a collection105.

| Field | Data Type | Description | Key |
| --- | --- | --- | --- |
| **\_id** | ObjectId | Unique identifier of the document, generated automatically by MongoDB. | PK |
| **username** | String | Username, must be unique for each document. |  |
| **email** | String | Email address, must also be unique. |  |
| **password** | String | User's password. |  |
| **created\_at** | ISODate | Date and time the document was created. |  |
|  |  |  |  |

### 7.2.2. Routes Collection.

Contains information about transport routes. The flexibility of MongoDB allows embedding stop documents (stops) directly within each route document.

| Field | Data Type | Description | Key |
| --- | --- | --- | --- |
| **\_id** | ObjectId | Unique identifier of the route. | PK |
| **name** | String | Name or code of the route (e.g., "A7-3"). |  |
| **path\_geojson** | Object | GeoJSON object representing the route's geometry. |  |
| **Estimated time** | float | Estimated time to reach the destination. |  |
| **Estimated price** | Float |  |  |
| **stops** | Array of Objects | An array of embedded documents representing the route's stops. |  |
|  |  |  |  |

* **Structure of stops (within routes):**

| Field | Data Type | Description | Key |
| --- | --- | --- | --- |
| **name** | String | Name of the stop. |  |
| **location** | Object | Coordinates of the stop in GeoJSON format (Point). |  |

### 7.2.3. alerts Collection

Records citizen alerts created by users. The alert documents reference the user who created them through their \_id.

| Field | Data Type | Description | Key |
| --- | --- | --- | --- |
| **\_id** | ObjectId | Unique identifier of the alert. | PK |
| **user\_id** | ObjectId | Reference to the \_id of the user who created the alert. | FK (users.\_id) |
| **alert\_type** | String | Type of alert (e.g., "Traffic", "Security"). |  |
| **description** | String | Detailed description of the alert. |  |
| **location** | Object | Coordinates of the incident in GeoJSON format (Point). |  |
| **created\_at** | ISODate | Date and time of the alert's creation. |  |
|  |  |  |  |

### 7.2.4. search\_history Collection

Saves the route search history for each user. Each search document references the user who performed it.

| Field | Data Type | Description | Key |
| --- | --- | --- | --- |
| **\_id** | ObjectId | Unique identifier of the search record. | PK |
| **user\_id** | ObjectId | Reference to the \_id of the user who performed the search. | FK (users.\_id) |
| **searched\_at** | ISODate | Date and time of the search. |  |
|  |  |  |  |

# 8. Implementation of Modules and Functionalities

## 8.1. User Authentication Module

### 8.1.1. Frontend: Registration and Login Forms.

User interface components were created using HTML and Tailwind CSS for the registration and login forms. The JavaScript logic is responsible for capturing user data, validating it on the client-side, and sending it to the backend via a POST request.

## 8.2. Route Planning Module

### 8.2.1. Frontend: Map Interface and Route Visualization.

The main interface displays an interactive map of Barranquilla using Leaflet.js. It includes input fields for origin and destination. Upon receiving the response from the backend, the client-side JavaScript draws the suggested route, stops, and any necessary transfers on the map.

### 8.2.2. Backend: Endpoint and Calculation Algorithm.

**GET /routes/**: This endpoint obtains the stored transport routes and displays the related stops and alerts.

## 8.3. Citizen Alerts Module

### 8.3.1. Frontend: Report and Visualization Form.

Authenticated users have access to a form to report a new alert, selecting the type, adding a description, and marking the location on the map. Existing alerts, obtained from the backend, are displayed as markers on the map.

### 8.3.2. Backend: Endpoints and User Validation.

**GET /alerts**: Returns a list of all active alerts to be displayed on the map.

**POST /alerts**: Receives the data for a new alert. Before processing it, it verifies that the request contains a valid authentication token. If the user is authenticated, it saves the new alert in the database.

## 8.4. Additional Functionalities

### 8.4.1. Search History

For authenticated users, each successful route search is saved in the

search\_history collection. This information is displayed in a section of the user's profile, allowing them to repeat frequent searches with a single click.

### 8.4.2. Dark Mode Implementation with Tailwind CSS

Taking advantage of Tailwind CSS's capabilities, a theme system was implemented. Using the class strategy (

darkMode: 'class'), conditional styles (dark:\*) were applied that are activated when the <html> tag has the dark class, controlled by a button in the interface.

# 9. Production Deployment

## 9.1. Project Configuration in Vercel

The project was connected to the GitHub repository from the Vercel dashboard. The corresponding framework was configured (e.g., "Other" for a static site) and the build commands and output directory were defined. Vercel automatically detects the frontend code and the backend serverless functions.

## 9.2. Environment Variable Management

Sensitive credentials, such as the database connection string and secret keys for token generation, were not stored in the source code. Instead, they were configured as environment variables directly in the project settings in Vercel, ensuring that they are only accessible in the production environment.

## 9.3. Continuous Integration and Continuous Deployment (CI/CD) Process

Vercel automates the CI/CD process. Every time a git push is made to the main branch, Vercel automatically triggers a new deployment: it clones the repository, installs the dependencies, runs the build process, and, if successful, updates the application in production with no downtime.

# 10. Testing and Quality Assurance (QA)

## 10.1. Endpoint Testing with Postman

Postman was used to test each of the API's endpoints in isolation. The correct functioning of the methods (GET, POST), the response status codes (200, 201, 400, 403), and the structure and content of the JSON responses for successful and error cases were verified.

## 10.2. Manual End-to-End User Flow Testing

Complete manual tests were performed simulating the experience of an end user. This included the following flows:

* Registration of a new user.
* Login and logout.
* Searching for a route and displaying it correctly.
* Creating a new alert.
* Activating dark mode.

## 10.3. Responsiveness and Cross-Browser Compatibility Testing

It was verified that the user interface was displayed and functioned correctly on different screen sizes, from small mobile devices to large desktop monitors. In addition, compatibility was checked in the latest versions of the main web browsers (Chrome, Brave, Firefox).

## 10.4. Unit and Integration Testing with Jest

To ensure the reliability of the Backend, an automated test suite was implemented using the Jest framework. These tests focused on validating the business rules of the controllers and the correct data manipulation in the models. The objective was to ensure that individual functions behaved as expected and that the interactions between the different modules (e.g., controllers, models) worked without errors. The test suite included the following cases:

* **database.test.js:** Tests for the connection and configuration of the database, verifying that the application can establish and maintain a stable connection.
* **route-controller.test.js:** Integration tests for the route controllers, validating that they correctly handle HTTP requests and that the business logic is executed accurately before sending the response to the client.
* **route-model.test.js:** Unit tests for the route data models, ensuring that the data validation and manipulation functions work correctly.
* **user-controller.test.js:** Tests for the user controllers, verifying that the authentication, registration, and profile management processes work as designed.
* **user.test.js:** Unit tests for the user model, focused on business logic such as email format validation, password strength, and correct manipulation of user information.
* The use of Jest allowed us to detect errors early, facilitate code maintenance, and ensure that each new functionality or change in the system did not introduce errors in existing components, leading to an agile development process and robust quality assurance (QA).

# 11. Conclusions

## 11.1. Achievement of Objectives

The Connect BQ project successfully met all the general and specific objectives set. A fully functional web platform was delivered that integrates a route planner, an alert system, and user authentication, all deployed on a modern and scalable infrastructure, and with an intuitive and attractive user interface.

## 11.2. Challenges Encountered and Solutions Applied

* **Challenge:** Accessing the API with the official AMB routes to include them in the project and ensure they corresponded to reality.
* **Solution:** They were helped with the conversion of KMZ data to latitude and longitude manually by applications such as Qruta.
* **Challenge:** Managing the state of the application on the frontend in a coherent manner, especially user authentication.
* **Solution:** A simple state management approach was adopted using the browser's localStorage to persist the authentication token and a global JavaScript context to share the user's state between components.

## 11.3. Lessons Learned

The main lesson learned was the importance of a well-defined API design from the beginning of the project. Having a clear "contract" between the frontend and the backend facilitated parallel work and reduced integration problems174. In addition, the choice of a serverless architecture proved to be highly beneficial in terms of costs and ease of deployment.

# 12. Future Work and Possible Improvements

## 12.1. Implementation of a Route Algorithm

To develop with more knowledge the route algorithm which allows setting the starting and destination points and through the database calculate the most optimal, fastest, or alternative routes.

## 12.2. Integration with Real-Time Data (Bus GPS)

Integrate the option for users to provide the GPS location of buses in real time. This would allow showing the current position of the vehicles on the map and offering much more accurate arrival estimates to other users.

## 12.3. Development of a Progressive Web App (PWA)

Improve the web application to turn it into a PWA, which would allow users to "install" it on the home screen of their mobile devices, access certain functionalities offline, and eventually receive push notifications for alerts or arrival estimates of their buses.

## 12.4. Strategic Alliance with the AMB for Official Data

To take the platform's accuracy to the next level, a key future improvement is to establish a strategic alliance with the Metropolitan Authority of Barranquilla (AMB)180. This would allow Connect BQ to access an API with the official routes of the entire metropolitan area of the city181. This collaboration would ensure the regularization and constant updating of routes, stops, and schedules, eliminating the need for manual data collection and guaranteeing that the platform's information is always accurate and reliable for users.

# 13. Appendices

## 13.1. API Contract

Below is the complete documentation for each API endpoint, including its technical specifications and usage examples.

**User Management**

POST /users

Method: POST

Description: Creates a new user in the system.

Authentication: Not required.

Request Body:

JSON

{

"identityNumber": "string",

"username": "string",

"password": "string",

"email": "string",

"phone": "string",

"age": "number"

}

Successful Response (201 Created):

JSON

{

"message": "User created successfully",

"user": {

"id": "ObjectId",

"identityNumber": "string",

"username": "string",

"email": "string",

"phone": "string",

"age": "number",

"routesHistory": [],

"favoritesRoutes": [],

"createdAt": "date",

"updatedAt": "date"

}

}

Error Response (400 Bad Request):

JSON

{

"message": "The input data is not valid."

}

**GET /users**

Method: GET

Description: Gets the list of all registered users in the system.

Authentication: Not required.

Successful Response (200 OK):

JSON

[

{

"id": "ObjectId",

"identityNumber": "string",

"username": "string",

"email": "string",

"phone": "string",

"age": "number",

"routesHistory": [],

"favoritesRoutes": [],

"createdAt": "date",

"updatedAt": "date"

}

]

**GET /users/:id**

Method: GET

Description: Gets the information of a specific user by their ID.

Parameters: id (string, in the URL).

Authentication: Required (through localStorage).

Successful Response (200 OK):

JSON

{

"id": "ObjectId",

"identityNumber": "string",

"username": "string",

"email": "string",

"phone": "string",

"age": "number",

"routesHistory": [],

"favoritesRoutes": [],

"createdAt": "date",

"updatedAt": "date"

}

Error Response (404 Not Found):

JSON

{

"message": "User not found."

}

Route Management

GET /routes

Method: GET

Description: Gets all available routes.

Authentication: Not required.

Successful Response (200 OK):

JSON

[

{

"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

Method: GET

Description: Gets the details of a specific route by its ID.

Parameters: id (string, in the URL).

Authentication: Not required.

Successful Response (200 OK):

JSON

{

"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": []

}

Error Response (404 Not Found):

JSON

{

"message": "Route not found."

}

Alert Management

POST /alerts

Method: POST

Description: Creates a new alert in the system.

Authentication: Required (the user must be logged in and have an active session in localStorage).

Request Body:

JSON

{

"alertType": "string",

"description": "string",

"location": {

"latitude": "number",

"longitude": "number"

}

}

Successful Response (201 Created):

JSON

{

"message": "Alert created successfully",

"alert": {

"id": "ObjectId",

"userId": "string",

"alertType": "string",

"description": "string",

"location": {

"latitude": "number",

"longitude": "number"

},

"createdAt": "date"

}

}

Error Response (401 Unauthorized):

JSON

{

"message": "Unauthorized. User not logged in or invalid session."

}

Error Response (400 Bad Request):

JSON

{

"message": "The alert data is not valid."

}

GET /alerts

Method: GET

Description: Gets all active alerts in the system.

Authentication: Not required.

Successful Response (200 OK):

JSON

[

{

"id": "ObjectId",

"userId": "string",

"alertType": "string",

"description": "string",

"location": {

"latitude": "number",

"longitude": "number"

},

"createdAt": "date"

}

]

Error Response (404 Not Found):

JSON

{

"message": "No alerts found."

}

## 13.2. Basic User Manual.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

## Diagrama El contenido generado por IA puede ser incorrecto.13.4 Diagram Navegation Users

# 14. Acknowledgements and Final Considerations

We appreciate you taking the time to review our project, Connect BQ. We hope this document has provided a clear and comprehensive overview of our work. Thank you for your consideration.