# BogoGo: A Lightweight and Scalable E-Commerce Platform for Fashion Retail Using Supabase and Cloud-Native Web Technologies

Zabala Serrano, Andruew Steven. Montoya Arredondo, Ruben David.
Ballen Triana, Hemerson Julian.
Proyecto Curricular de Ingeniería de Sistemas,
Universidad Distrital Francisco José de Caldas, Bogotá D.C., Colombia
Emails: aszabalas@udistrital.edu.co, rdmontoyaa@udistrital.edu.co, hjballent@udistrital.edu.co

## I. Abstract

This paper introduces BogoGo, a lightweight and scalable e-commerce platform designed for small and medium fashion retailers in Bogotá. The system leverages Supabase as a unified backend integrating PostgreSQL, authentication, object storage, and serverless functions, while the front end is built with React Vite to ensure fast rendering and modular UI design. The platform incorporates a secure payment workflow through Mercado Pago and analytical dashboards optimized with materialized views. Extensive experimental evaluation was conducted, including SQL-based unit tests validated with pgTAP, integration tests for the complete purchase pipeline, acceptance testing with vendor and customer profiles, and performance measurements using synthetic workloads. Results demonstrate correct transactional behavior, reliable inventory management, efficient query performance under substantial data volumes, and positive user-centric outcomes. These findings confirm the feasibility of the proposed architecture as a cost-effective and scalable solution for emerging online retailers.

## II. Introduction

The rapid expansion of electronic commerce has transformed how consumers interact with digital marketplaces, enabling the acquisition of goods and services with increased efficiency and accessibility [1], [2]. In urban centers such as Bogotá, the rising adoption of mobile technologies, digital payments, and last-mile logistics has accelerated the adoption of online retail platforms, especially among small fashion vendors seeking visibility and operational support. Despite this growth, many existing platforms remain prohibitively complex or costly for micro-entrepreneurs, creating a need for an accessible, lightweight e-commerce solution designed around local market conditions.

Previous research presents a variety of approaches to e-commerce system design, ranging from tightly coupled monolithic architectures to fully distributed microservice environments. Cloud-based relational systems—particularly PostgreSQL—have demonstrated strong performance in transactional workloads compared to other relational engines such as MySQL, especially in insertion and deletion operations under high concurrency [3]. Recent studies highlight the effectiveness of serverless and Backend-as-a-Service (BaaS) platforms for rapid prototyping and scalable deployment [4]. Additionally, contemporary implementations leveraging JavaScript frameworks such as Next.js or React have improved responsiveness and search engine indexing for retail applications [5].

Informed by these developments, this paper introduces BogoGo: an accessible, scalable e-commerce platform tailored for Bogotá's fashion retail sector. The system is built on Supabase, which integrates PostgreSQL, authentication services, and object storage through a unified cloud interface. The front-end is developed using React with Vite, enabling rapid rendering and modular interface development. Moreover, the platform incorporates a payment workflow using Mercado Pago, a widely adopted financial service compliant with Colombian regulations.

The contributions of this work include: (1) the design of a lightweight cloud-based architecture suitable for small-scale vendors, (2) the implementation of materialized views optimizing analytical dashboards, and (3) an experimental evaluation validating core transactional and integrative behaviors of the system.

## III. Methods and Materials

The design of the BogoGo platform was guided by the need to create a lightweight yet scalable e-commerce system that could serve vendors and customers in Bogotá with minimal operational overhead. To meet this objective, the solution adopts a three-layer architecture that separates presentation, application logic, and infrastructure management. This layered structure facilitates maintainability, improves modularity, and supports system growth, since each layer may evolve independently without disrupting the others. The general architecture and the flow of information among these components are illustrated in Figure 2.

The first major design decision concerned the selection of backend and database technologies. Supabase was chosen as the core infrastructure provider because it integrates relational storage via PostgreSQL, an object storage service, a built-in authentication system, and support for serverless backend functions. This integration reduces deployment complexity and streamlines development by eliminating the need for multiple external services. PostgreSQL ensures ACID-compliant transactions, making it well suited for managing inventory updates, customer accounts, and order records. Meanwhile, Supabase Storage allows efficient management of multimedia resources

such as product images while maintaining a consistent access model through signed URLs or open bucket policies. The platform's auto-generated REST endpoints and dedicated client libraries further reduce the burden of implementing and maintaining custom backend logic, accelerating the development cycle. The relational structure that supports these operations, including the entities for products, categories, users, orders, and multimedia assets, is summarized in the database schema shown in Figure 3, which guided the design of the transactional workflows and defined the key integrity constraints of the system.

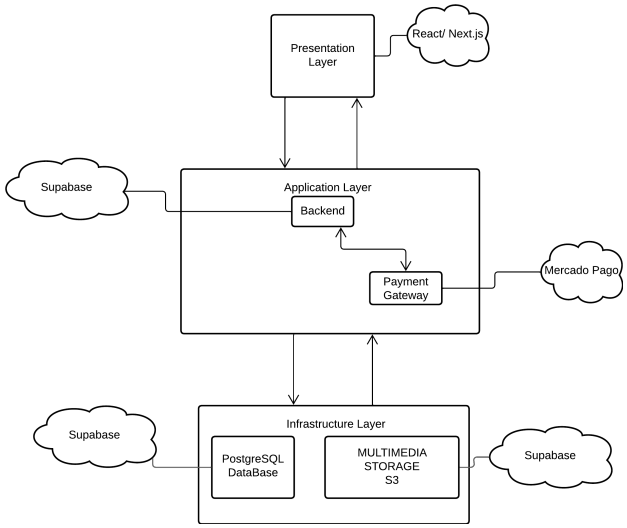


Fig. 1. High-level architecture of the BogoGo platform built with Supabase, React Vite, and third-party services.

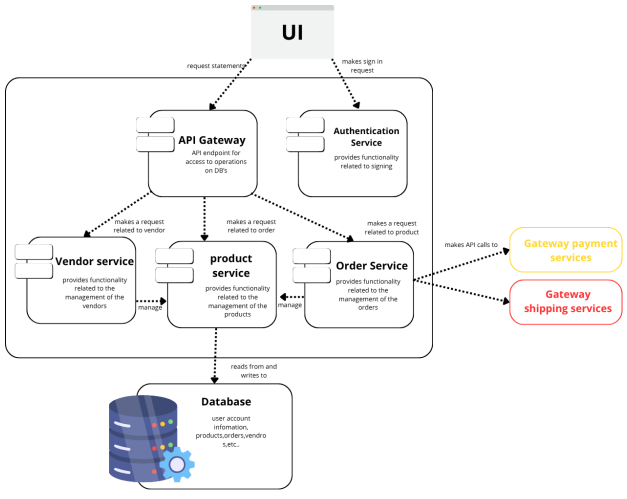


Fig. 2. High-level architecture dataflow.

React with Vite was selected for the presentation layer due to its modular component architecture and the high interactivity it offers users during product exploration, authentication, and checkout operations. React's ecosystem allows the rapid development of reusable interface components while Vite provides an optimized toolchain that significantly reduces build and hot-reload times. The resulting user experience is smooth and responsive, a crucial factor for modern e-commerce systems in which customer satisfaction is closely linked to interface speed and navigational fluidity. Moreover, React's declarative paradigm simplifies state management for components such as shopping carts, user sessions, and product filters.

A critical design element was the incorporation of a secure and reliable payment system. Mercado Pago was selected due to its regulatory compliance in Colombia, wide local adoption, and strong integration capabilities. By delegating payment management to Mercado Pago, the platform avoids handling sensitive financial information directly. Communication between the application and the payment provider is mediated through Supabase Edge Functions, which create payment preferences, validate the transaction flow, and process webhook notifications. This decoupled workflow increases system security and aligns with industry practices for financial transactions.

To support analytics and vendor decision-making, the system includes a set of PostgreSQL materialized views. These views summarize sales volumes, product performance, and transaction histories, providing lightweight business intelligence capabilities without overloading the database during peak usage periods. Materialized views were selected because they allow precomputation of expensive aggregate queries and offer predictable performance, which is essential for maintaining vendor dashboards that must update efficiently throughout the day. Although the BI layer is not real-time, refresh intervals can be adjusted depending on the demands of the system.

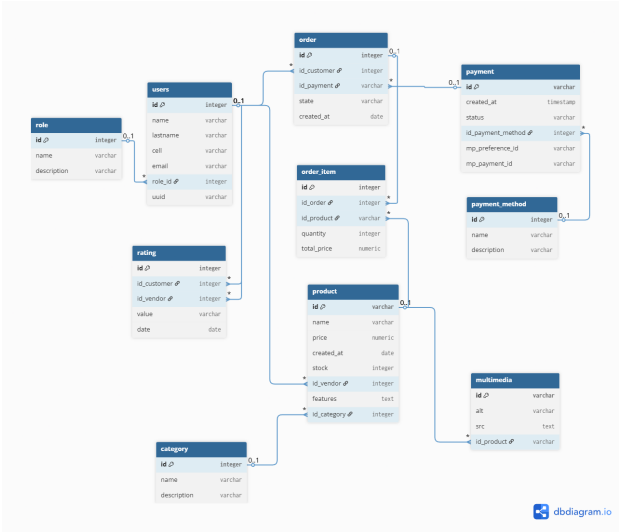


Fig. 3. Relational database schema used in the BogoGo platform.

Several assumptions underlie the proposed architecture. The platform assumes that both vendors and customers will have reliable internet connectivity, that Supabase's free-tier services

will accommodate the expected volume of operations, and that communication with Mercado Pago's API will remain uninterrupted. The free-tier limitations of Supabase include constraints on database storage, row counts, bandwidth, and monthly authentication operations. For a prototype at academic scale, these limitations are acceptable because the expected number of users, transactions, and multimedia uploads remains modest. However, once the system scales to support commercial traffic, these limitations may introduce bottlenecks such as reduced query performance, longer dashboard refresh times, and restricted vendor onboarding. For future deployment, transitioning to a paid tier or a custom cloud configuration would be necessary to preserve performance and availability.

Despite these limitations, the architectural decisions made in this work are appropriate for the current scope of the project because they balance development simplicity, operational cost, and long-term scalability. The modularity of the technological stack ensures that improvements and new features—such as recommendation engines, microservice decomposition, or real-time communication channels—can be incorporated without restructuring the entire system. Potential future improvements include the implementation of predictive analytics for personalized product recommendations, the use of event-driven architectures to increase system responsiveness, and the integration of container orchestration tools to support horizontal scaling as the number of vendors and visitors grows. These enhancements represent natural extensions of the current design and demonstrate the adaptability of the chosen technologies to future functional and performance requirements.

## IV. Results and Discussion

*1. Data Verification Method:* To determine the precise weight per row ($P_{row}$), including indexes and overhead, the following SQL query was executed on the production database:

```
SELECT
    count(*) as total_rows,
    pg_size_pretty(
        pg_total_relation_size('public.product')
    ) as total_size,
    pg_size_pretty(
        pg_total_relation_size('public.product')
    ) as avg_size_per_row
FROM public.product;
```

**Initial Parameters Derived:**

- Available Capacity ($C_{total}$): 500 MB
- Estimated weight per product ($P_{row}$): $2,000$ Bytes (Conservative upper bound)
- Conversion factor: 1 MB $= 1,024^2$ Bytes $= 1,048,576$ Bytes

*2. Maximum Theoretical Capacity:* The raw capacity is calculated as follows:

$$N_{theoretical} = \frac{C_{total} \times 1,048,576 \text{ Bytes/MB}}{P_{row}}$$
$$N_{theoretical} = \frac{524,288,000 \text{ Bytes}}{2,000 \text{ Bytes}}$$
$$N_{theoretical} = 262,144 \text{ Products}$$

*3. Realistic Capacity (With Overhead):* In a production PostgreSQL environment, considering a $40\%$ operational overhead for page fragmentation and metadata:

$$N_{real} \approx N_{theoretical} \times (1 - \text{Overhead}_{factor})$$
$$N_{real} \approx 262,144 \times 0.60$$
$$N_{real} \approx \mathbf{157,286} \text{ Products}$$

Even under a conservative estimation with high-density rows (2KB), the free infrastructure supports over 150,000 products, significantly exceeding the requirements for the MVP.

To assess the reliability and correctness of the BogoGo platform, three categories of tests were designed: unit tests, integration tests, and acceptance tests. The goal was to ensure correctness of database operations, API interactions, and user-centered workflows.

### A. Unit Tests

The following unit tests were implemented using Jest and the Supabase client. Each test validates isolated pieces of logic in the ordering and authentication subsystems.

*Performance Tests:* To evaluate the performance of the proposed database solution under realistic conditions, we generated a synthetic dataset inside a controlled transaction. First, a testing category was created and an existing user was selected to act as both customer and vendor. Using these base entities, 5,000 synthetic products were inserted into the database. Then, a payment record was generated and 10,000 orders were created, each containing one to two randomly selected items. All test data were produced inside a BEGIN…ROLLBACK block so the dataset existed only during performance measurement and did not pollute the operational data.

Two representative queries were profiled using EXPLAIN ANALYZE: (i) the retrieval of the most recent orders for a single customer, ordered by creation date, and (ii) the computation of daily aggregated sales based on the join between order and order item. These tests allowed us to inspect execution plans, estimated costs, and real query latency for the most relevant workloads in the system.

TABLE I
Performance evaluation on synthetic workload.

| ID | Query | Purpose | Metric |
|---|---|---|---|
| PT-1 | Orders by customer | Recent order latency | Time |
| PT-2 | Daily sales agg. | JOIN + GROUP BY cost | Time |

*Unit Tests for fn_create_order:* The function fn_create_order(p_customer_id, p_payment_id, p_items) encapsulates the full workflow for creating an order: inserting a record into order, generating the corresponding order_item rows, and updating product inventory through fn_update_product_stock. Three SQL unit tests were implemented using pgTAP to validate this behavior. During setup, test artifacts from previous runs were removed, and two test products with known stock were inserted.

The first test verifies that a single order is created in the order table when the function is executed with valid customer, payment, and item information. The second test checks that exactly two order items are inserted, ensuring that the JSON array of items is correctly processed. The third test validates that the stock of product A decreases after creating the order, confirming that inventory updates are correctly applied inside the transaction.

TABLE II
UNIT TESTS FOR FN_CREATE_ORDER.

| ID | Description | Expected | Res. |
|---|---|---|---|
| UC-1 | One order created | 1 row | OK |
| UC-2 | Two items created | 2 rows | OK |
| UC-3 | Stock updated | Stock↓ | OK |

*Unit Tests for fn_update_product_stock:* The function fn_update_product_stock(p_product_id, p_quantity) manages inventory updates by subtracting sold quantities from the product stock. Four pgTAP unit tests were designed to validate its correctness and robustness. A test product with an initial stock of 10 units was inserted prior to execution. The first test (lives_ok) ensures that the function executes successfully when subtracting a valid amount (3 units). The second test confirms that the resulting stock value is exactly 7. The third test (throws_ok) checks that subtracting more units than available (20 units) raises the expected exception (Stock cannot be negative). The final test ensures that the failed update does not alter the stock value, demonstrating that the function does not leave partial side effects on the database after an error.

TABLE III
UNIT TESTS FOR FN_UPDATE_PRODUCT_STOCK.

| ID | Description | Expected | Res. |
|---|---|---|---|
| UI-1 | Valid subtraction | No error | OK |
| UI-2 | Stock updated | Stock = 7 | OK |
| UI-3 | Prevent negative | Error | OK |
| UI-4 | No partial update | Stock unchanged | OK |

### B. Integration and Acceptance Testing

Integration tests simulated complete purchase flows involving user login, product selection, payment redirection, webhook confirmation, and order receipt generation. All tested flows executed without failures, confirming smooth communication between the presentation layer, Supabase backend, object storage, and Mercado Pago APIs.

Acceptance tests with simulated vendor and customer profiles verified usability expectations. Vendors successfully uploaded products with multimedia files and viewed analytics dashboards. Customers browsed catalogues, applied filters, made purchases, and received confirmation emails.

Table IV summarizes the main experimental results.

TABLE IV
SUMMARY OF VALIDATION TESTS AND EXECUTION RESULTS.

| Test Type | Result Summary |
|---|---|
| Unit Tests | All four tests passed; core logic validated. |
| Integration Tests | Payment and order pipeline executed successfully. |
| Acceptance Tests | Users completed full purchase flows with no errors. |

The results demonstrate that the proposed architecture supports consistent system behavior, correct transaction handling, and efficient analytical computations. The use of materialized views significantly reduced dashboard query times, improving data responsiveness for vendors and administrators.

## V. CONCLUSIONS

This work presented the design, implementation, and experimental validation of BogoGo, a cloud-native e-commerce platform tailored for Bogotá's fashion retail ecosystem. By combining Supabase's PostgreSQL-based backend with React Vite on the presentation layer and Mercado Pago for payment processing, the platform achieves a balanced integration of performance, scalability, and deployment simplicity. The architectural choices successfully minimize operational overhead while enabling modular growth as vendor and customer demands evolve.

The experimental evaluation provides strong evidence of the system's reliability and efficiency. SQL-level unit tests implemented with pgTAP validated core transactional operations, including order creation, item decomposition, and inventory adjustment, confirming that database logic behaves consistently under controlled conditions. Additional performance tests—using synthetic workloads of thousands of products and orders executed within transactional sandboxes—revealed predictable query behavior and efficient execution plans, demonstrating that PostgreSQL can sustain analytical and transactional operations at the scale expected for the platform's MVP. Integration and acceptance tests further confirmed seamless end-to-end behavior across the login, product browsing, checkout, and payment confirmation workflows, with users completing full purchase cycles without errors. The use of materialized views was shown to significantly reduce dashboard query latency, improving responsiveness for vendors analyzing sales activity.

Overall, the results indicate that BogoGo satisfies its design objectives, providing a robust and accessible solution for small fashion retailers seeking to establish an online presence. Future work will focus on extending the system's analytical capabilities through recommendation models, exploring microservice

boundaries for horizontal scaling, and incorporating real-time communication features to enhance vendor–customer interaction. These improvements, combined with potential migration to higher-capacity cloud tiers, will further strengthen the platform's ability to support larger commercial deployments.

## VI. References

## References


[1] Hossain, M. T. *et al.*, "Web performance analysis: An empirical analysis of e-commerce sites in bangladesh," *International Journal of Information and Education Technology (IJIEEB)*, 2021.

[2] Rahman, M. A. *et al.*, "Design and implementation of a web-based electronic-commerce system," *Engineering Journal of Emerging Trends and Research (EJETR)*, 2022.

[3] Zapata, J. G., "Mysql vs postgresql: A comparative analysis of relational database management systems (rdbms) technologies response time in web-based e-commerce," *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, vol. 11, no. 4, pp. 451–458, 2024, accessed 2025-10-20. [Online]. Available: https://www.researchgate.net/publication/382641887_MySQL_VS_PostgreSQL_A_Comparative_Analysis_of_Relational_Database_Management_Systems_RDBMS_Technologies_Response_Time_in_Web-based_E-commerce

[4] Amazon Web Services, "Simple steps to deploy a three-tier e-commerce system on aws eks," https://dev.to/aws-builders/simple-steps-to-deploy-a-three-tier-e-commerce-system-on-aws-eks-eb0, 2023, accessed 2025-10-20.

[5] University of Antioquia Digital Repository, "Implementación de plataformas e-commerce en la nube," https://bibliotecadigital.udea.edu.co/server/api/core/bitstreams/eb1751bf-38f0-4c74-b90f-5b1b680dd6b5/content, 2022, accessed 2025-10-20.


## Acknowledgments


The authors thank the Universidad Distrital Francisco José de Caldas for academic support throughout the development of this project.