Design and Architecture of an Event Registration Platform

Carlos Andres Abella, Daniel Felipe Paez Mosquera, Leidy Marcela Morales
Department of Computer Engineering
Universidad Distrital Francisco José de Caldas
Bogotá D.C., Colombia
Emails: {caabella, dfpaezm, lmmorales}@udistrital.edu.co

***Abstract*—Digital transformation has transformed the way events are organized and experienced. Traditional registration processes often lead to inefficiencies, errors, and limited accessibility. This article presents the design and architecture of an event registration platform designed to automate event management, ticket purchasing, and user interaction. The proposed solution adopts a three-tier software architecture, integrating relational databases for transactional reliability and a NoSQL component for scalability and real-time search. The results demonstrate that the hybrid data strategy, coupled with containerized implementation, provides a solid and extensible foundation for a secure, efficient, and user-centric event management system. Index Terms: Event management, software architecture, hybrid databases, microservices, Docker.**

***Index Terms*—Event management, software architecture, hybrid databases, microservices, Docker.**

I. Introduction

The growing demand for professional, cultural, and entertainment events has highlighted the limitations of traditional management methods, which often rely on fragmented, manual processes for registration, payment, and communication. These approaches create inefficiencies for organizers and inconveniences for attendees, highlighting the need for digital platforms capable of integrating all stages of the event lifecycle into a unified system.

The Event Registration Platform addresses these challenges by offering a secure, scalable, and intuitive environment for creating, publishing, and purchasing event tickets. The system follows modern software engineering principles, ensuring modularity, maintainability, and performance through a layered architecture.

Previous work on distributed and microservices-based systems highlights the advantages of modularity and independent deployment capabilities. Similarly, relational databases offer strong data integrity guarantees [3], [4], while NoSQL technologies improve the flexibility and performance of semi-structured data [5], [6]. Combined, these paradigms enable hybrid persistence architectures tailored to diverse data requirements.

In parallel, usability studies highlight the importance of intuitive design and accessibility in web systems [7]. Therefore, the platform integrates user-centered design practices to ensure an efficient and enjoyable experience. This article details the methodological, architectural, and experimental aspects of the proposed solution and analyzes its expected impact and scalability.

II. Methods and Materials

The methodology adopted for this project follows an iterative and incremental approach inspired by the principles of Agile Programming and Scrum [8], [9]. The design process was structured in several stages: requirements elicitation, architecture definition, system modeling, and interface design.

A. Requirements and Business Modeling

A *Business Model Canvas* was used to define the platform's value propositions, customer segments, and revenue structure. Three main user roles were identified: *Event Organizer*, *Ticket Buyer*, and *Platform Administrator*. User stories were written for each role and organized into a *User Story Map* to prioritize features and define the Minimum Viable Product (MVP).

B. Architectural Design

The system is based on a classic three-tier model:

- **Presentation Layer:** Implements the user interface through a web application for event navigation, registration, and payments.
- **Business Logic Layer:** Manages event publishing, ticket validation, and transaction processing.
- **Data Access Layer:** Manages communication with data storage components.

A hybrid persistence approach combines relational databases for transactional operations and NoSQL for flexible, semi-structured data and real-time analytics. The architecture provides for a decomposition into microservices to improve scalability and isolation of modules such as authentication, payments, and notifications.

C. Deployment and Containerization

The system's deployment model uses Docker containers to encapsulate services, ensuring environmental consistency and simplifying CI/CD integration. Each major component (API Gateway, database, web client) is packaged as a container image, facilitating scalability in cloud environments such as AWS or Azure.

D. System Modeling

UML class diagrams describe the main entities—*User, Event, Ticket, Payment*—and their relationships. BPMN models represent the main business process: *Ticket Purchase*, which details user actions and decision points from selection to confirmation. These models ensure consistency between business logic and software implementation.

E. User Interface Design

High-fidelity mockups were created to illustrate the user experience. The interfaces were designed following usability heuristics [7], prioritizing clarity, feedback, and responsiveness across different devices.

III. Results & Discussion

The proposed system artifacts constitute a complete framework for implementation and validation.

A. Business Model and Requirements

The Business Model Canvas defined the strategic alignment of technological and business objectives. User stories captured detailed interactions for each role, e.g., event creation, ticket purchase, and real-time sales tracking. The resulting User Story Map facilitated feature prioritization for iterative development.

B. Software and Deployment Architecture

Fig. 1 presents the logical software architecture. The platform's modular design ensures loose coupling between layers and supports independent service scaling. The deployment strategy (Fig. 2) illustrates Docker-based containerization, where each microservice operates independently, communicating via REST APIs. This configuration enhances scalability and fault tolerance.

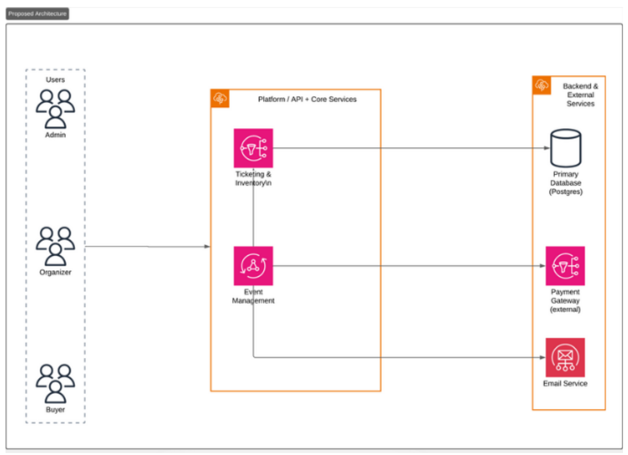
Proposed Architecture
Users
Admin
Organizer
Buyer
Platform / API + Core Services
Ticketing & Inventoryln
Event Management
Backend & External Services
Primary Database (Postgres)
Payment Gateway (external)
Email Service

Fig. 1. Logical Software Architecture of the Event Registration Platform.

C. System and Process Modeling

The UML Class Diagram defines structural relations among entities, while the BPMN diagram maps the purchase workflow. Together, these models bridge business and technical design, ensuring functional traceability. Table I summarizes the principal testing and design outcomes.

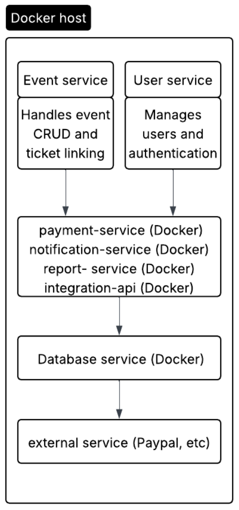
Docker host
Event service
User service
Handles event CRUD and ticket linking
Manages users and authentication
payment-service (Docker)
notification-service (Docker)
report- service (Docker)
integration-api (Docker)
Database service (Docker)
external service (Paypal, etc)

Fig. 2. Deployment Architecture leveraging Docker containers.

TABLE I
Summary of Platform Design Results

Aspect	**Key Result**
Architecture	Three-layer + microservice-ready
Database	Hybrid (Relational + NoSQL)
Deployment	Docker containerization
UI Design	Responsive, role-based mockups
Modeling Tools	UML, BPMN, User Story Map

D. Discussion

The combination of relational and NoSQL storage proved conceptually effective for handling diverse data types while maintaining transactional reliability. Containerization guarantees consistent deployments and easier scaling. Compared with traditional monolithic systems, this architecture offers superior maintainability and resilience.

Nonetheless, empirical validation remains pending; performance and scalability assessments will be conducted upon implementation. The proposed framework, however, provides a solid basis for future functional and non-functional testing.

IV. Conclusions

This study presents a comprehensive design for an *Event Registration Platform* that integrates modern software engineering principles with user-centered design. The architecture's modularity, hybrid data persistence, and containerized deployment demonstrate a balance between flexibility and robustness.

Key outcomes include:

- A scalable and maintainable software architecture suitable for real-world deployment.
- A hybrid relational–NoSQL persistence strategy optimizing data integrity and performance.
- UML and BPMN models ensuring design traceability.
- A visual user interface consistent with usability standards.

Future work will focus on implementation, empirical performance evaluation, and integration of advanced analytics (e.g., recommendation systems or predictive demand forecasting). The results confirm that combining layered and service-oriented architectures enables efficient, secure, and adaptive event management platforms.

Acknowledgments

The authors thank **Professor Carlos Andrés Sierra** for his invaluable guidance and the Department of Computer Engineering at Universidad Distrital Francisco José de Caldas for institutional support.

References

[1] M. Fowler, "Microservices," *martinfowler.com*, 2014.
[2] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.
[3] C. J. Date, *An Introduction to Database Systems*, 8th ed., Addison-Wesley, 2003.
[4] PostgreSQL Global Development Group, "PostgreSQL Official Website." [Online]. Available: https://www.postgresql.org/
[5] E. Brewer, "Towards Robust Distributed Systems," in *Proc. 19th ACM Symp. Principles of Distributed Computing*, 2000.
[6] MongoDB Inc., "MongoDB Documentation." [Online]. Available: https://docs.mongodb.com/
[7] J. Nielsen and D. Norman, *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, 1999.
[8] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed., Addison-Wesley, 2004.
[9] M. Cohn, *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.