Universidad Distrital Francisco José de Caldas

Dept. of Computer Engineering

Systems Engineering

Technical Report: Design and Architecture of an Event Registration Platform Software Engineering Seminar

Carlos Andres Abella
Daniel Felipe Paez Mosquera

Leidy Marcela Morales

Supervisor: Carlos Andrés Sierra

Bogotá D.C.

October 23, 2025

Abstract

This technical report details the design and architectural considerations for an **Event Registration Platform**, a software system aimed at streamlining event management and ticket purchasing. The project leverages a comprehensive methodology, starting from business model definition and user story mapping, progressing to detailed UI mockups, and concluding with robust software and database architecture designs. Key methodologies include a three-layer architectural model, a PostgreSQL relational database for transactional data, and a MongoDB-style NoSQL solution for advanced search and real-time analytics. The findings present a scalable, secure, and user-centric platform, demonstrating a hybrid data system architecture that balances data integrity with performance and flexibility. This design ensures efficient event creation and management for organizers, alongside a seamless and secure ticket purchasing experience for attendees.

1 Introduction

The proliferation of digital tools has transformed various industries, with event management being a prime example where efficiency and user experience are paramount. Traditional methods of organizing and registering for events often suffer from logistical complexities, manual processing errors, and limited accessibility, leading to inefficiencies for organizers and frustrations for attendees. This report addresses these challenges by presenting a comprehensive design and architectural blueprint for an **Event Registration Platform**. This system aims to provide a robust, scalable, and intuitive solution for event creators to manage their activities, from publication to attendee tracking, while offering users a seamless and secure pathway to discover and purchase tickets. The platform is designed to cater to a diverse range of events, enhancing operational effectiveness and fostering broader participation. The subsequent sections will detail the methodology, architectural choices, technical implementation, and anticipated outcomes of this digital transformation initiative.

2 Literature Review

The landscape of event management and digital registration platforms has evolved significantly, drawing upon advancements in software engineering, database technologies, and user experience design. This section provides an overview of foundational concepts and existing solutions pertinent to the development of the Event Registration Platform.

Modern event registration systems are often built on **microservices architectures** to achieve scalability, resilience, and independent deployability, as highlighted by works on distributed systems design (e.g., Fowler (2014), Newman (2015)). This approach contrasts with traditional monolithic structures, offering greater flexibility in handling varying loads across different functional domains—such as user authentication, event creation, and payment processing.

Key to these platforms is the effective management of diverse data types. Relational databases, primarily PostgreSQL, are widely adopted for their ACID compliance and robust support for transactional data, ensuring data integrity for critical operations like ticket sales and user profiles (e.g., Date (2003), PostgreSQL Official (n.d.)). Concurrently, the rise of NoSQL databases, exemplified by MongoDB, addresses the need for flexible schema, horizontal scalability, and high-performance read operations for features like real-time search, personalized recommendations, and logging (e.g., Brewer (2000), MongoDB Official (n.d.)). The combination of these, forming a a **hybrid data architecture**, is a well-established pattern for balancing different data requirements.

User Interface (UI) and User Experience (UX) design principles, as advocated by Nielsen & Norman (1999), are also critical. The literature emphasizes intuitive navigation, clear calls to action, and responsive design to ensure accessibility across various devices, which directly informs the platform's UI mockups. Furthermore, security protocols for online transactions and data protection, adhering to standards like GDPR and PCI DSS, are fundamental requirements discussed extensively in contemporary software security literature (e.g., Schneier (2000)).

Finally, software development methodologies, particularmente **Agile approaches** y sus derivados, proporcionan marcos para el desarrollo iterativo y la colaboración de las partes interesadas, asegurando que el sistema evolucione en respuesta a la retroalimentación y a los requisitos cambiantes. El uso de Historias de Usuario y Mapas de Historias de Usuario, como propuso Cohn (2004), es fundamental para capturar los requisitos funcionales desde una perspectiva centrada en el usuario y priorizar los esfuerzos de desarrollo de manera efectiva. Además, los principios de Programación Extrema (XP), a menudo discutidos en el contexto de Agile, proporcionan orientación sobre prácticas técnicas para el desarrollo de software de alta calidad (ej., Beck (2004)).

3 Background

The development of the Event Registration Platform necessitates an understanding of both the core business domain—event management—and the foundational technical paradigms driving modern software systems. This section provides an overview of these essential contexts.

3.1 Event Management Domain

Event management encompasses the process of planning, organizing, promoting, and executing events, ranging from small workshops to large conferences or concerts. Key activities include event creation, venue and schedule definition, ticket pricing and sales, attendee registration, communication, and post-event analytics. Digital platforms aim to automate and streamline these tasks, reducing manual effort and potential errors while enhancing reach and user experience. Understanding the workflow and specific needs of event organizers and attendees is crucial for designing a platform that genuinely adds value.

3.2 Core Technologies and Architectural Concepts

The proposed platform's architecture is built upon established principles of modern software engineering. It employs a **three-layer architecture** (Presentation, Business Logic, Data Access) to ensure a clear separation of concerns, facilitating maintainability and scalability. For data persistence, a **hybrid data system** is utilized:

- **Relational Databases (PostgreSQL):** Chosen for their robustness, ACID compliance, and mature support for complex transactional operations, vital for secure ticket purchasing, user management, and event definitions.
- **NoSQL Databases (MongoDB-style):** Integrated to handle high volumes of semi-structured data, support real-time search capabilities, manage logs, and provide flexibility for evolving data schemas, essential for features like dynamic event details or user activity tracking.

Furthermore, the system design considers principles of **service-oriented architectures**, which advocate for breaking down complex applications into smaller, independent, and loosely coupled services. This approach, often realized through microservices, enhances agility, fault isolation, and the ability to scale individual components independently based on demand. The deployment strategy will leverage **containerization technologies** (e.g., Docker), which encapsulate applications and their dependencies, ensuring consistency across different environments and simplifying deployment and scaling operations.

This foundational understanding of event dynamics and technological choices forms the bedrock upon which the platform's detailed design and implementation are constructed.

4 Objectives

The primary objective of this technical report is to document the comprehensive design and architectural considerations for an **Event Registration Platform**. This platform aims to modernize event management processes by providing a robust, scalable, and intuitive digital solution for event organizers and attendees.

More specifically, the objectives of this report are to:

- **Define the foundational business model** and functional requirements of the platform, outlining its value proposition, key activities, and user roles (Organizer, Buyer, Administrator).
- **Present a detailed software architecture**, including a three-layer model and a hybrid data persistence strategy leveraging both relational (PostgreSQL) and NoSQL (MongoDB-style) databases, designed for scalability and data integrity.
- **Illustrate the core system functionalities** through user stories, class diagrams, and BPMN process models, focusing on critical workflows such as event creation, ticket purchasing, and user management.
- **Detail the deployment architecture** using containerization technologies (e.g., Docker) to ensure consistency, portability, and efficient scaling across different environments.
- **Provide a tangible representation of the user interface** through mockups, showcasing the intended user experience for key interactions within the platform.
- **Establish a clear blueprint** for the future development phases, serving as a guide for implementation, testing, and eventual deployment of the Event Registration Platform.

5 Scope

The scope of this technical report is precisely delineated to focus on the **design and architectural blueprint** of the Event Registration Platform. This includes the conceptualization, functional definition, and technical specification necessary for its foundational development.

Specifically, this report encompasses:

- **Requirements Analysis and Business Modeling:** Detailing the problem domain, system objectives, key user roles (Organizer, Buyer, Administrator), and the underlying business model as presented in the Business Model Canvas and User Stories.
- **Software Architecture Design:** Establishing the logical and physical architecture, including the selection of architectural styles (e.g., three-layer, service-oriented principles), data persistence strategies (hybrid relational and NoSQL), and deployment considerations (Docker-based containerization).
- **System Modeling:** Providing detailed models suchs as Class Diagrams, User Story Maps for feature prioritization, and BPMN diagrams for critical business process flows (e.g., ticket purchasing).
- **User Interface (UI) Mockups:** Presenting wireframes and mockups for key screens and interactions to illustrate the intended user experience and functional layout.

Conversely, this report **explicitly excludes** the following aspects:

- **Full-scale Implementation:** The report does not include the actual code development, testing, or deployment of the Event Registration Platform. It serves as a pre-implementation design document.
- **Live Production Deployment and Maintenance:** While deployment architecture is discussed, the ongoing management, monitoring, and operational maintenance of a live production environment are outside the current scope.
- **Detailed Non-Functional Requirements Testing:** Performance, security, and scalability testing results are not included, as these require a fully implemented system. The report focuses on *designing for* these qualities.
- **Financial Analysis Beyond Business Model Canvas:** A deep dive into financial projections, return on investment (ROI), or detailed budget analysis for the project's implementation is not covered.
- **Integration with Third-Party Systems (beyond conceptual):** While payment gateways or external notification services might be mentioned conceptually, detailed integration specifics are beyond this design phase.

This defined scope ensures that the report provides a focused and comprehensive architectural foundation, setting clear boundaries for the current stage of the project.

6 Assumptions

During the design and architectural planning of the Event Registration Platform, several key assumptions have been made. These assumptions underpin the proposed solutions and define the context within which the platform is intended to operate. Should any of these assumptions prove incorrect during subsequent development or deployment phases, the design may require re-evaluation and adjustment.

The primary assumptions are:

- **Availability of Cloud Infrastructure:** It is assumed that a suitable cloud computing environment (e.g., AWS, Azure, Google Cloud Platform) will be available for hosting the platform's microservices and databases. This includes access to virtual machines, managed database services, and container orchestration capabilities.
- **Developer Skill Set:** The development team responsible for implementing the platform is assumed to possess proficiency in modern web development frameworks, database management (SQL and NoSQL), containerization (Docker), and cloud deployment practices.
- **Standard Internet Connectivity:** Users (both organizers and attendees) are assumed to have reliable and standard internet connectivity to access and interact with the platform effectively.
- **Security Standards Compliance:** It is assumed that the external payment gateway (if integrated) will handle PCI DSS compliance, and that the platform will be developed following general security best practices to protect user data (e.g., encryption in transit and at rest, secure authentication mechanisms).
- **Third-Party Service Reliability:** Any external services potentially integrated (e.g., email notification services, SMS gateways, payment processors) are assumed to be reliable, stable, and to provide well-documented APIs.
- **System Scalability Requirements:** While the design aims for scalability, the specific thresholds for concurrent users and events are assumed to be within typical limits for a medium-to-large-scale event platform, not requiring hyper-scale infrastructure at initial launch. Exact scaling needs will be refined with further non-functional requirements analysis.
- **Legal and Regulatory Environment:** It is assumed that the platform will operate within a clear legal and regulatory framework regarding data privacy (e.g., GDPR, local regulations) and online transactions, and that these requirements will be addressed during implementation.
- **Technology Stack Stability:** The core technologies selected (e.g., PostgreSQL, MongoDB, Docker, chosen programming languages/frameworks) are assumed to remain stable and supported throughout the development and initial operational lifecycle.

7 Limitations

This technical report, while comprehensive in its design and architectural blueprint for the Event Registration Platform, is subject to several limitations inherent to its current stage of development. Recognizing these limitations is essential for proper interpretation of the presented design and for guiding future development efforts.

The primary limitations include:

- **Absence of Empirical Validation:** The designs and architectural choices presented herein have not yet been subjected to empirical validation through actual implementation, testing, and performance benchmarking. Therefore, conclusions regarding system performance, scalability, and robustness are theoretical and based on established industry best practices and literature.
- **Focus on Design, Not Implementation:** This report is a design document. It details *how* the system should be built from an architectural and functional perspective but does not include the specifics of source code, detailed integration logic, or resolved implementation-level challenges that would arise in a full development cycle.
- **Generalized Non-Functional Requirements:** While non-functional requirements (e.g., performance, security, scalability) are considered in the design, their detailed specification and testing are beyond the scope of this report. The current design aims to *support* these qualities, but their concrete achievement requires a fully built and tested system.
- **Conceptual Integration with Third-Party Services:** Integrations with external payment gateways, notification services, or other APIs are discussed conceptually. The complexities, specific protocols, and potential challenges of these real-world integrations are not fully detailed or resolved at this design stage.
- **Limited User Feedback on Mockups:** The UI mockups presented are based on initial user story analysis and design principles. Extensive user testing and iterative feedback cycles on the actual user interface are yet to be conducted, meaning the proposed UI/UX is subject to refinement.
- **Future Technology Evolution:** The chosen technology stack is current at the time of this report. Rapid advancements in software development mean that certain technologies or approaches may evolve, become deprecated, or new, more efficient alternatives may emerge before full implementation, potentially requiring design adjustments.
- **Static Representation of Dynamic Processes:** Business process models (BPMN) and user stories provide a static representation of dynamic interactions. Edge cases, error handling for complex real-time scenarios, and highly variable user behaviors are considered at a high level but would require more granular analysis during implementation.

These limitations highlight the fact that this report provides a foundational design, requiring subsequent stages of development, testing, and iterative refinement to fully realize the Event Registration Platform.

8 Methodology

The design and architectural blueprint of the Event Registration Platform were developed following a structured and iterative methodology, drawing upon principles of Agile software development. This approach ensured a comprehensive understanding of requirements, flexible design choices, and a strong focus on delivering a user-centric solution.

8.1 Agile Development Principles

The project adopted an iterative and incremental approach, characteristic of Agile methodologies, specifically influenced by Scrum and Extreme Programming (XP) practices. This allowed for continuous feedback, adaptation to evolving requirements, and a focus on delivering value in manageable increments. The emphasis was on clear communication and collaboration among stakeholders throughout the design phase.

8.2 Requirements Elicitation and Business Modeling

The initial phase focused on a deep understanding of the problem domain and stakeholder needs.

- **Business Model Canvas (BMC):** The **Business Model Canvas** was utilized to define the fundamental aspects of the platform's business model, including value propositions, customer segments, key activities, resources, and revenue streams. This provided a holistic view of the platform's strategic context.
- **User Stories and User Story Mapping:** Functional requirements were captured as **User Stories**, focusing on user roles (Organizer, Buyer, Administrator) and their desired functionalities. These stories were then organized into a **User Story Map** to visualize the user's journey, prioritize features, and establish a release roadmap for the platform.

8.3 Software Architecture Design

The architectural design phase established the structural foundation of the platform.

- **Three-Layer Architecture:** A classical **three-layer architectural pattern** was adopted to ensure separation of concerns, improve maintainability, and facilitate scalability. These layers include:
  - **Presentation Layer:** Responsible for handling user interaction and displaying information.
  - **Business Logic Layer:** Contains the core business rules and orchestrates data flow.
  - **Data Access Layer:** Manages interaction with data storage mechanisms.
- **Hybrid Data Persistence Strategy:** A **hybrid data system** fue diseñada, leveraging the strengths of different database paradigms:
  - **PostgreSQL (Relational Database):** Chosen for transactional data requiring strong consistency, such as user accounts, event details, and ticket sales.
  - **MongoDB-style (NoSQL Database):** Selected for its flexibility and scalability in handling semi-structured data, supporting real-time search, logging, and dynamic event content.
- **Service-Oriented Principles and Containerization:** The architecture integrates principles of **service-oriented design**, anticipating a microservices-like decomposition in future implementation. **Docker containerization** was planned for encapsulating services, ensuring portability, consistencia, y eficiente despliegue a través de varios entornos.

8.4 System Modeling

Detailed visual models were created to represent the platform's structure and behavior.

- **UML Class Diagrams: UML Class Diagrams** were developed to define the static structure of the system, illustrating key entities (e.g., User, Event, Ticket, Payment), their attributes, relationships, and operations.

- **BPMN Diagrams: Business Process Model and Notation (BPMN)** diagrams were used to model critical business workflows, such as the event creation process by an organizer and the ticket purchasing flow for an attendee, visualizing the sequence of activities and decision points.

8.5 User Interface (UI) Design

The user-facing aspects of the platform were conceptualized through visual design artifacts.

- **Mockups and Wireframes:** Low-fidelity **wireframes** and higher-fidelity **mockups** were created for key user interfaces (e.g., event listing, event detail, registration form, organizer dashboard). These designs focused on usability, accessibility, and an intuitive user experience, serving as a visual guide for frontend development.

This methodical approach allowed for a structured progression from high-level business understanding to detailed technical design, laying a solid foundation for the Event Registration Platform.

9 Results

This section presents the primary findings derived from the design and architectural planning of the Event Registration Platform. The results are comprised of the key design artifacts that collectively form the functional and technical blueprint for the system. Each artifact is a direct outcome of the methodology outlined previamente y se presenta objetivamente para documentar la solución propuesta.

9.1 Business Model Definition

The strategic foundation of the platform was defined using the Business Model Canvas. This model, visually articulates the project's value propositions, customer segments, key activities, revenue streams, and cost structure, ensuring alignment between business goals and technical design. For a detailed visual representation, please refer to Figure 1 in Appendix B.

9.2 Functional Requirements and User Flows

The functional requirements were captured through a comprehensive set of User Stories tailored to the three primary roles: Event Organizer, Ticket Buyer, and Platform Admin. These stories were subsequently organized into a User Story Map to visualize the end-to-end user journey, prioritize features, and define the scope for the initial product release (MVP). The detailed user stories are provided in Appendix A, and the User Story Map is presented in Figure 2 of Appendix B.

9.3 Software and Deployment Architecture

The system's high-level architecture is defined by a three-layer model that separates presentation, business logic, and data persistence, promoting modularity and scalability. The proposed deployment strategy is based on Docker containerization, ensuring environment consistency and portability. The logical software architecture is detailed in Figure 3 in Appendix B, while the physical deployment model is shown in Figure 4 of Appendix B.

9.4 System and Process Modeling

The detailed object-oriented structure of the platform is formally documented in the UML Class Diagram. This diagram illustrates the system's core classes, their attributes, methods, and the relationships between them. Complementing this, the primary business workflow, the "Ticket Purchase Process," was modeled using BPMN to represent the sequence of user actions, decisions, and business rules involved. The UML Class Diagram is presented in Figure 5 of Appendix B, and the BPMN Diagram in Figure 6 of Appendix B.

9.5 User Interface Design

The visual and interactive design of the platform is presented through a series of high-fidelity mockups. These mockups provide a tangible representation of the user experience for key flows, including event discovery, ticket purchasing, and event management, directly translating functional requirements into a visual design. A comprehensive gallery of these mockups can be found in Appendix C (Figures 7 through 12).

10 Discussion

The design and architectural blueprint presented in this report lay a solid foundation for the Event Registration Platform, addressing the initial objectives set forth. The comprehensive approach, from defining the business model to detailing UI mockups, demonstrates a coherent strategy for developing a robust, scalable, and user-centric system.

10.1 Alignment with Objectives and Requirements

The developed artifacts, including the Business Model Canvas (refer to Figure 1 in Appendix B) and the User Story Map (refer to Figure 2 in Appendix B), directly address the objective of defining the foundational business model and functional requirements. These tools have been instrumental in capturing the core value propositions for both event organizers and attendees, ensuring that the platform's features are aligned with real-world needs. The detailed user stories, particularly those related to event creation and ticket purchasing, serve as a clear guide for implementation, breaking down complex functionalities into manageable, user-focused tasks.

10.2 Architectural Decisions and Justification

The chosen software architecture, particularmente el modelo de tres capas (refer to Figure 3 in Appendix B) y la estrategia de persistencia de datos híbrida, refleja un esfuerzo consciente para equilibrar la integridad de los datos con el rendimiento y la flexibilidad. El uso de PostgreSQL para datos transaccionales (ej., ventas de boletos, autenticación de usuarios) proporciona sólidas garantías ACID, cruciales para las operaciones financieras. Por el contrario, la base de datos de estilo MongoDB propuesta para datos semiestructurados y capacidades de búsqueda avanzada ofrece la flexibilidad y escalabilidad necesarias para el contenido dinámico de eventos y operaciones de lectura de alto volumen, alineándose con las demandas de las plataformas modernas de capacidad de respuesta en tiempo real. La arquitectura de despliegue, centrada en Docker (refer to Figure 4 in Appendix B), mejora aún más la escalabilidad de la plataforma y asegura entornos consistentes en desarrollo, pruebas y producción, un beneficio clave para la integración y entrega continuas.

10.3 System Modeling and Workflow Efficiency

The UML Class Diagram (refer to Figure 5 in Appendix B) provides a clear and unambiguous representation of the system's static structure, defining the relationships between key entities such as 'User', 'Event', 'Ticket', and 'Payment'. This level of detail is critical for guiding the database design and object-oriented programming. Furthermore, the BPMN Diagram for the Ticket Purchase Process (refer to Figure 6 in Appendix B) vividly illustrates the user journey, identifying critical steps, decision points, and potential areas for optimization. This process modeling helps ensure that the platform's core functionalities are not only technically sound but also intuitive and efficient for the end-user.

10.4 User Experience and Interface Design

The series of UI mockups (refer to Appendix C, Figures 7 to 12) provides a tangible vision of the user experience. From the intuitive homepage (Figure 7) for event discovery to the streamlined checkout process (Figure 11), these designs prioritize clarity, ease of use, and visual appeal. They serve as a critical bridge between functional requirements and the final user-facing product, allowing for early feedback on the proposed interface before significant development effort is invested. The consistency in design across different screens contributes to a cohesive user experience, which is essential for user adoption and satisfaction.

10.5 Addressing Limitations and Future Work

While this report establishes a robust design blueprint, it is important to acknowledge the inherent limitations, particularly the absence of empirical validation. The theoretical advantages of the chosen architecture regarding scalability and performance will need to be verified through rigorous testing during the implementation phase. Future work will involve translating this design into executable code, conducting extensive functional and non-functional testing, and iterating on the UI/UX based on real user feedback. The modular nature of the design should facilitate these iterative enhancements and

allow for future expansions, such as integrating advanced analytics or more sophisticated personalization features.

11 Conclusion

This technical report successfully outlines a comprehensive design and architectural blueprint for an Event Registration Platform, effectively addressing the initial objectives of the project. Through a methodical application of agile principles, business modeling with the Business Model Canvas, and detailed system modeling using User Story Maps, UML Class Diagrams, and BPMN diagrams, a robust foundation has been established.

The proposed architecture, leveraging a three-layer model with a hybrid data persistence strategy (PostgreSQL for transactional data and MongoDB-style for flexible content), is designed to ensure scalability, data integrity, and responsiveness. Furthermore, the deployment strategy utilizing Docker containers provides a clear path for consistent and efficient deployment. The accompanying UI mockups translate functional requirements into a user-centric and intuitive visual experience.

Key Implications:

- The detailed design ensures a clear understanding of the system's structure and behavior, minimizing ambiguities for the development phase.
- The hybrid database approach optimizes for both transactional reliability and flexible content management, catering to diverse operational needs.
- The focus on user experience through comprehensive UI mockups promises a platform that is not only functional but also engaging and easy to use.
- The modular architecture facilitates future enhancements and integrations, positioning the platform for long-term adaptability.

Recommendations for Future Research and Practical Applications:

- **Implementation and Empirical Validation:** The most immediate next step is the full-scale implementation of the platform based on this blueprint. This phase will allow for empirical testing of performance, scalability, and security to validate the architectural assumptions.
- **Advanced Features Integration:** Exploring the integration of AI-powered recommendation engines for events, advanced analytics dashboards for organizers, and blockchain for secure ticket verification could significantly enhance the platform's value proposition.
- **User Experience Refinement:** Conducting extensive usability testing with target users will be crucial to refine the UI/UX mockups into a highly optimized and intuitive interface.
- **Security Audit and Compliance:** A thorough security audit, coupled with adherence to specific data privacy regulations (e.g., GDPR, CCPA) beyond conceptual mentions, should be a priority during implementation.

In summary, this report serves as a definitive guide for the next stages of development, providing a clear vision for an Event Registration Platform capable of meeting modern demands for efficiency, scalability, and user satisfaction in the event management domain.

References

References

Fowler, M. (2014). Microservices. *martinfowler.com*. Retrieved from `https://martinfowler.com/articles/microservices.html`

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.

PostgreSQL Global Development Group. (n.d.). *PostgreSQL Official Website*. Retrieved from `https://www.postgresql.org/`

Brewer, E. A. (2000, July). Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '00)*, pp. 7-10.

MongoDB, Inc. (n.d.). *MongoDB Documentation*. Retrieved from `https://docs.mongodb.com/`

Nielsen, J., & Norman, D. (1999). *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing.

Schneier, B. (2000). *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addison-Wesley.

A Detailed User Story List

This appendix provides the full list of user stories as defined in the project's initial requirements phase.

Title: Create and Publish a New Event	**Priority:** High	**Estimate:** Large

User Story:
As an **Event Organizer**, I want to create and publish a new event with detailed information, seating types, and prices, so that I can make it available for ticket buyers on the platform.

Acceptance Criteria:
Given I am logged in as an "Event Organizer",
When I navigate to the "Create Event" page, fill in all mandatory fields, define at least two ticket tiers, and click the "Publish" button,
Then the event is assigned a unique URL, becomes publicly visible, and its tickets are available for purchase.

Title: View Real-Time Sales Dashboard	**Priority:** High	**Estimate:** Medium

User Story:
As an **Event Organizer**, I want to view a real-time sales dashboard for my event, so that I can track ticket sales and revenue.

Acceptance Criteria:
Given I am logged in as an "Event Organizer" and am viewing one of my active events,
When I click on the "Dashboard" or "Reports" tab,
Then the system displays the total number of tickets sold per tier and the total gross revenue, updated in near real-time.

Title: Purchase Event Ticket Securely	**Priority:** High	**Estimate:** Large

User Story:
As a **Ticket Buyer**, I want to select tickets for an event and complete the purchase securely, so that I can receive a valid digital ticket to attend.

Acceptance Criteria:
Given I am viewing the page for an event with available tickets,
When I select my tickets, proceed to checkout, and confirm the mock payment,
Then I am redirected to a confirmation page and receive an email with the digital tickets as a PDF with a unique QR code.

Title: Browse and Discover Events	**Priority:** High	**Estimate:** Medium

User Story:
As a **Ticket Buyer**, I want to search for events by name, date, or category, so that I can easily find events I'm interested in.

Acceptance Criteria:
Given I am on the homepage,
When I use the search bar or apply a category filter,
Then the search results page displays only the events that match my criteria.

Title: Suspend User Account	**Priority:** Medium	**Estimate:** Medium

User Story:
As a **Platform Admin**, I want to suspend an Event Organizer's account, so that I can prevent fraudulent or harmful activity on the platform.

Acceptance Criteria:
Given I am logged in as a "Platform Admin" and am viewing a specific organizer's profile,
When I click the "Suspend Account" button and confirm my action,
Then the organizer's account is marked as suspended, they can no longer log in, and all their published events are unpublished.

B Architectural and Modeling Diagrams

This appendix includes detailed diagrams related to the platform's software architecture, deployment strategy, and system modeling. These visuals provide a deeper insight into the structural and behavioral aspects of the Event Registration Platform.

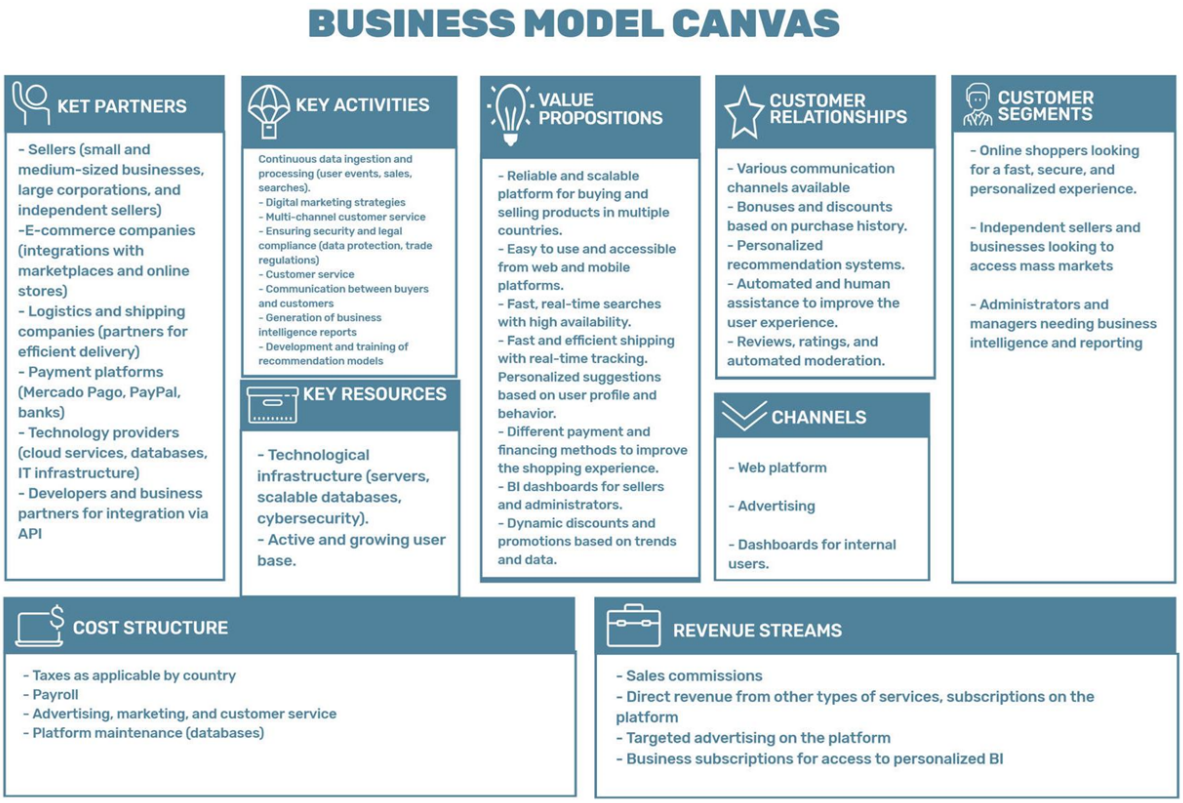


Figure 1: Business Model Canvas for the Event Registration Platform.

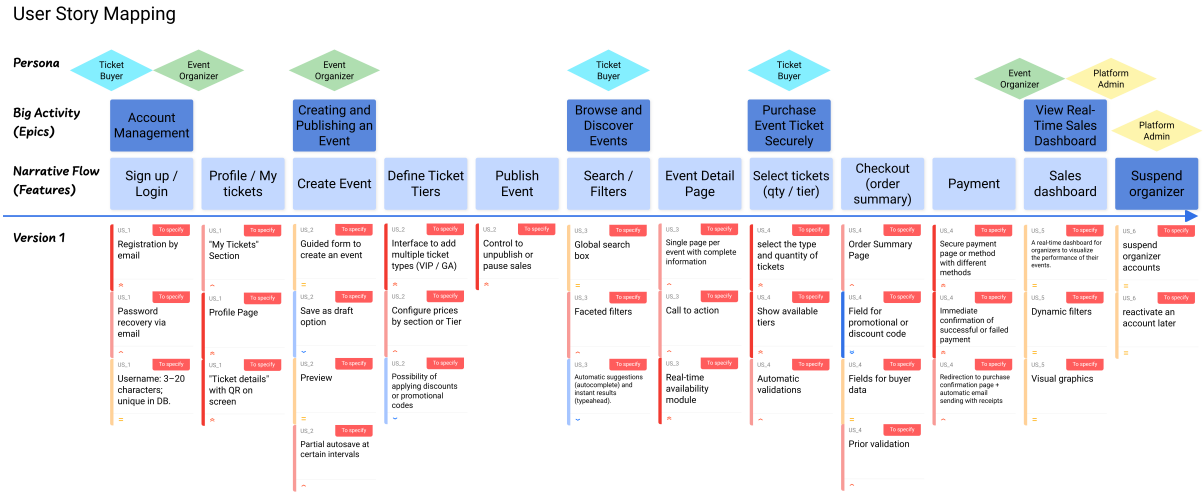


Figure 2: User Story Map illustrating user journeys and feature prioritization.

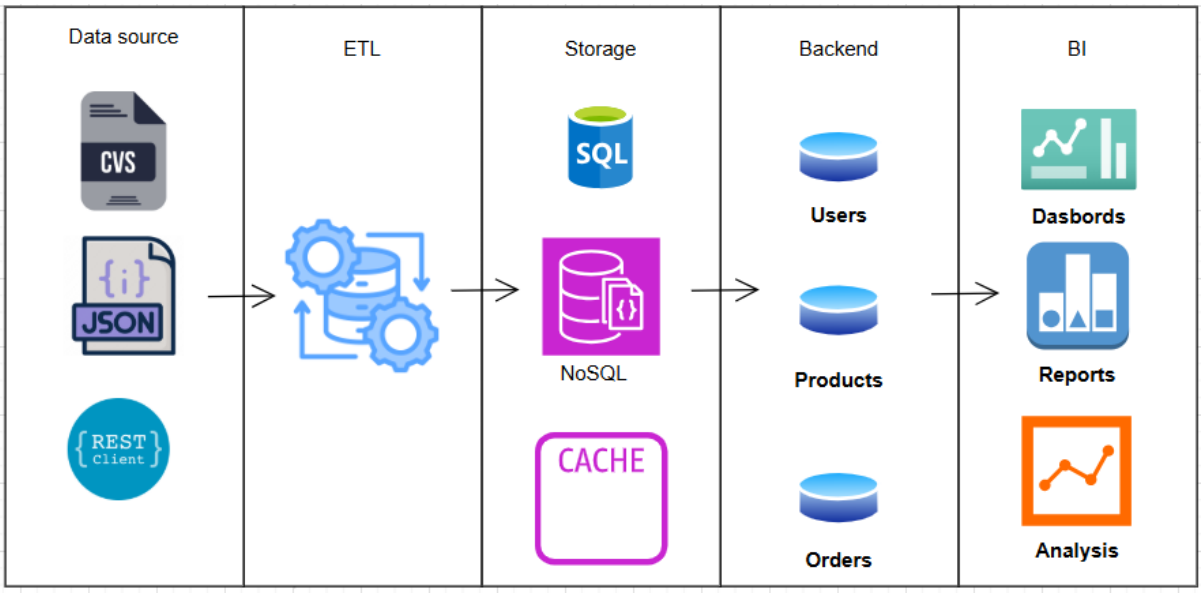
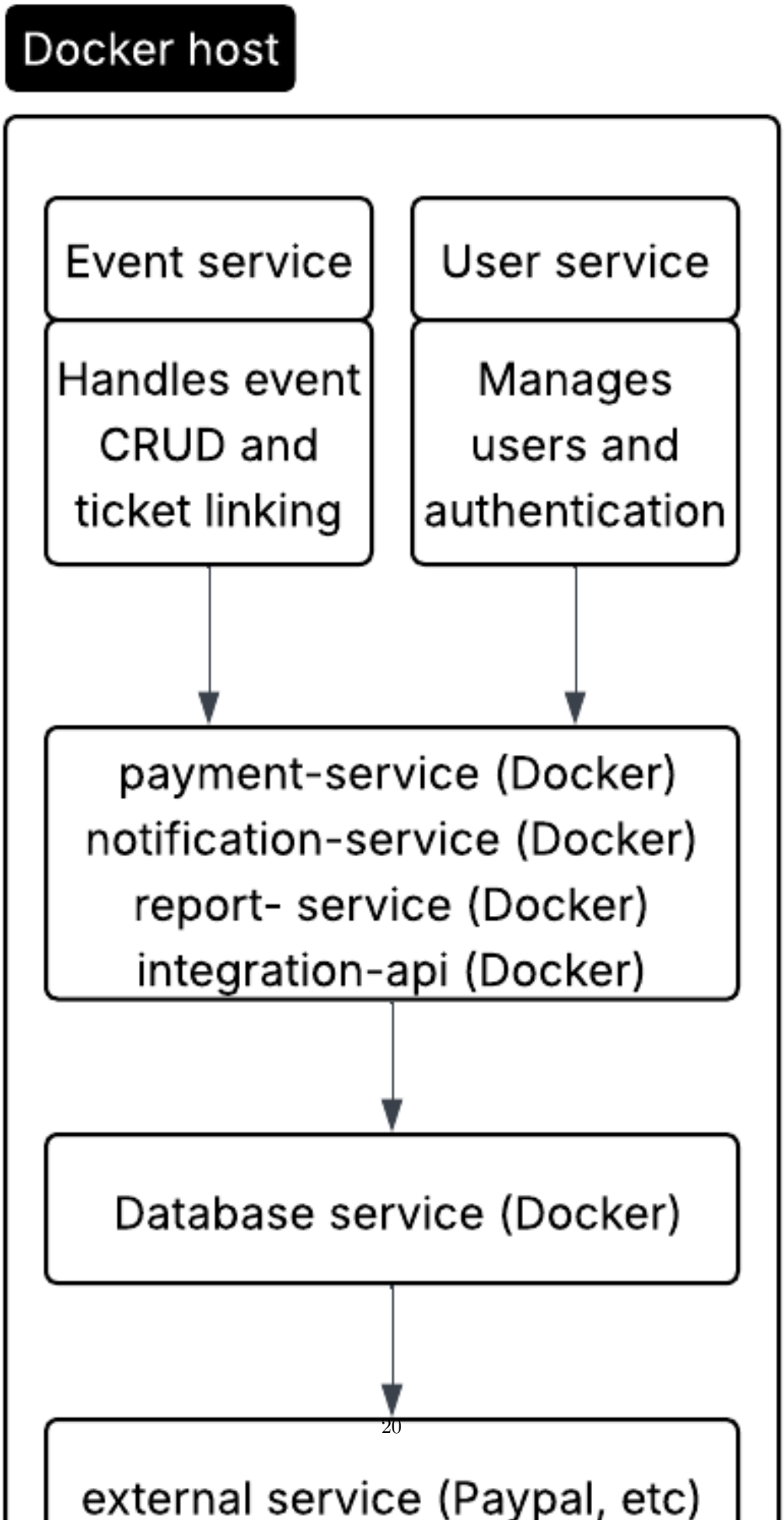


Figure 3: Logical Software Architecture of the Event Registration Platform (Three-Layer Model).


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

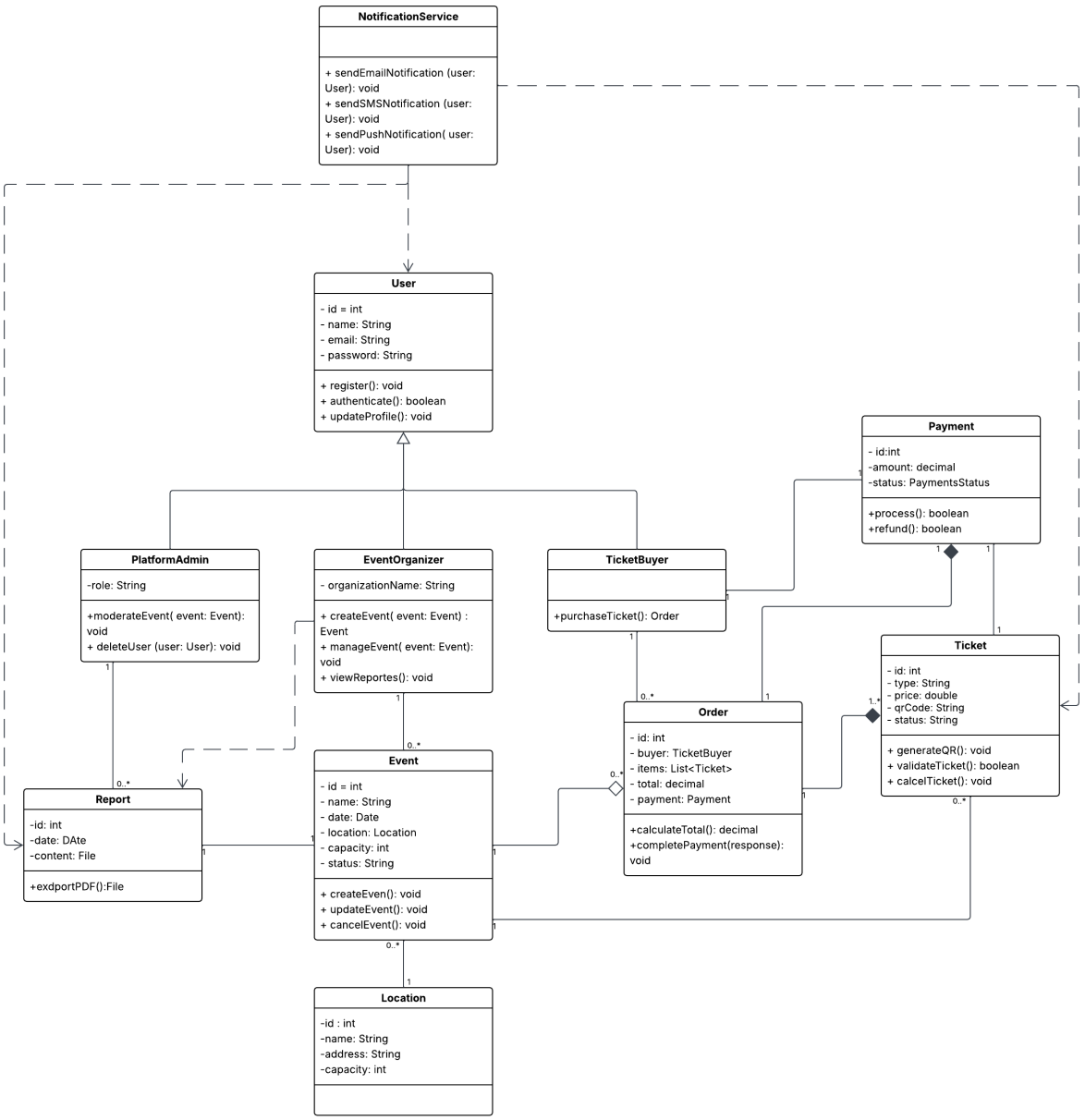


Figure 5: UML Class Diagram for the Event Registration Platform.

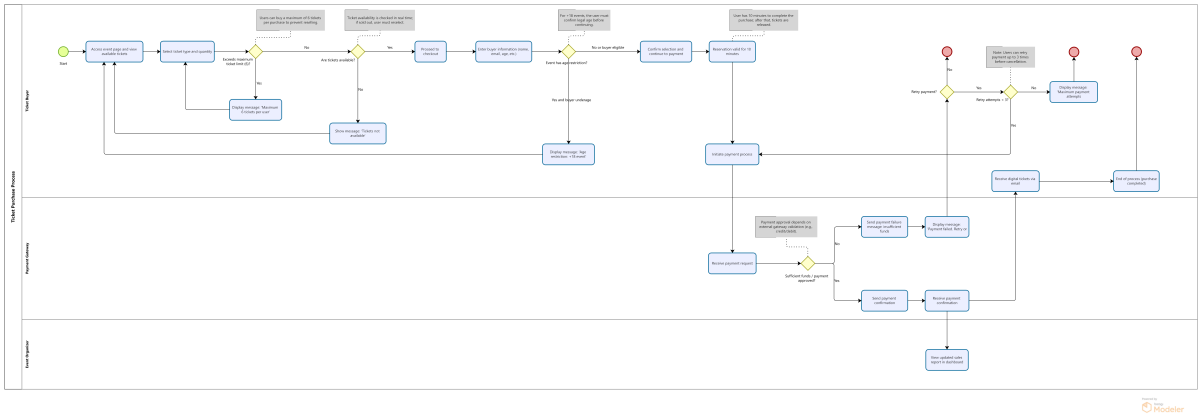


Figure 6: BPMN Diagram: Ticket Purchase Process.

C User Interface Mockups

This appendix contains a gallery of high-fidelity mockups representing key user interfaces and interactions within the Event Registration Platform. These designs serve as a visual guide for the intended user experience.

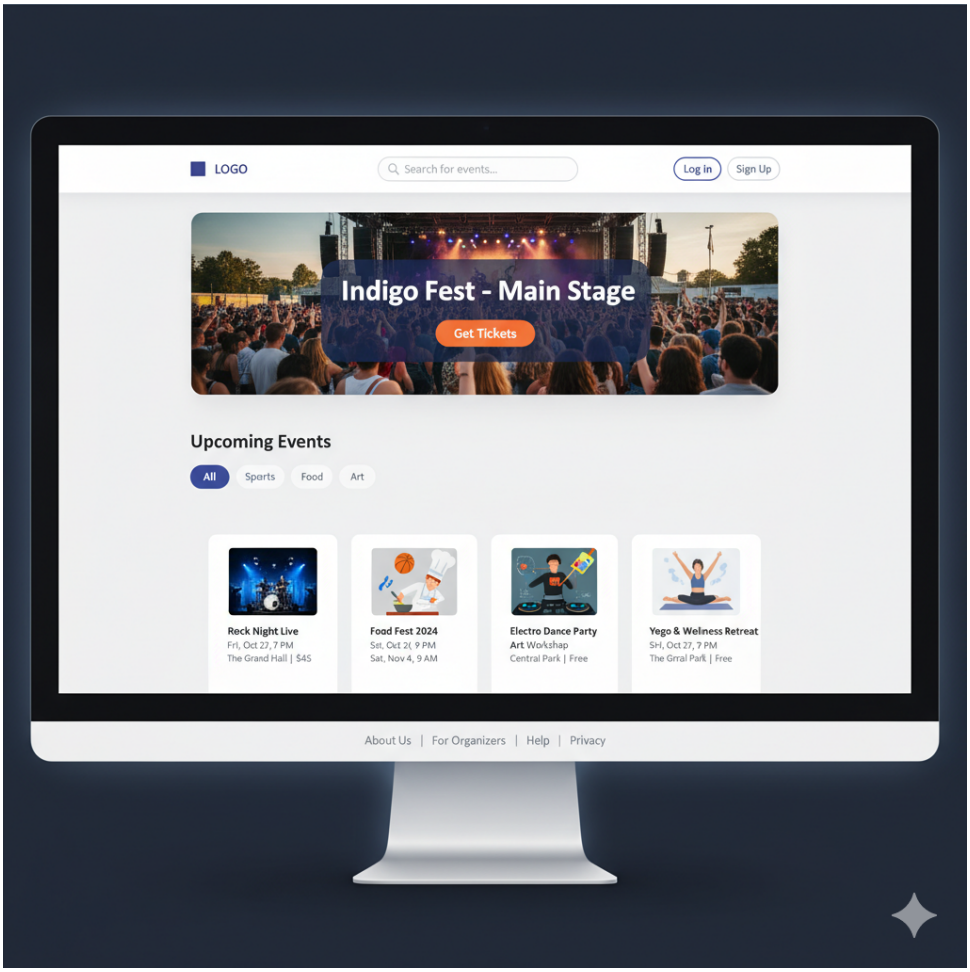


Figure 7: Mockup: Homepage / Event Listing.

Figure 8: Mockup: Event Details Page.

Figure 9: Mockup: Create New Event Form (Organizer).

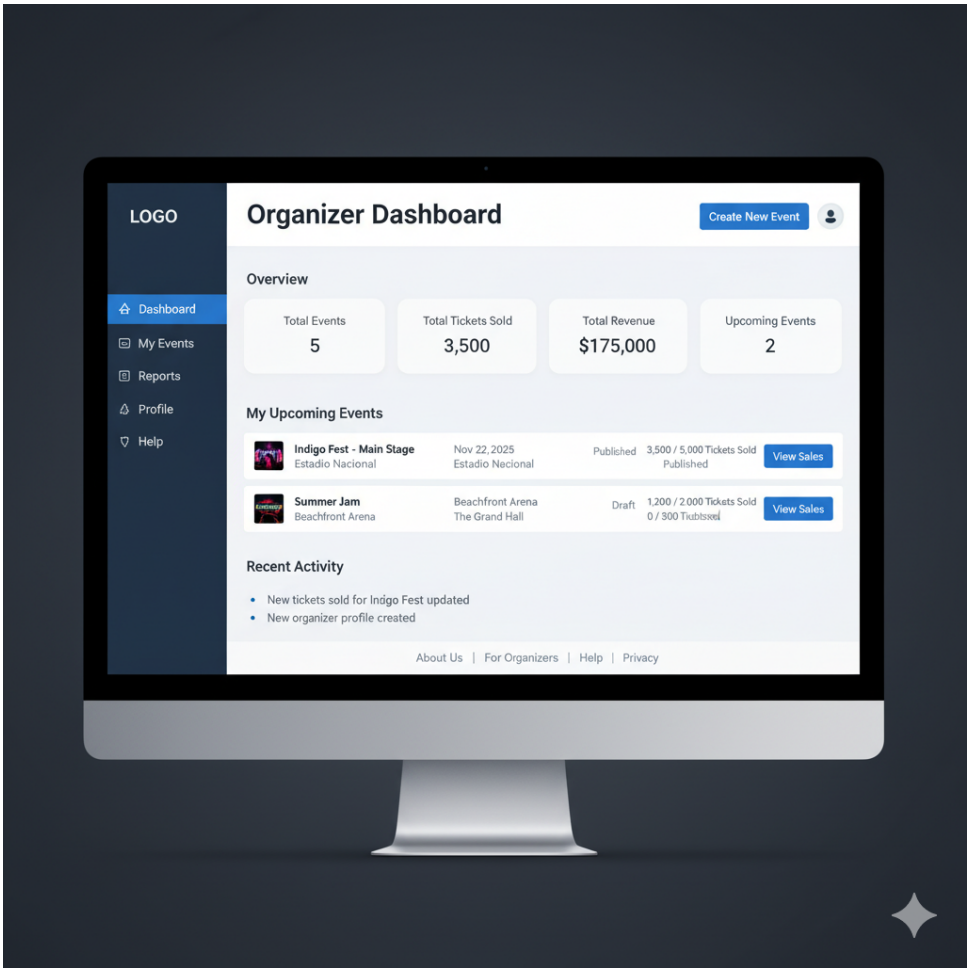


Figure 10: Mockup: Organizer Dashboard.

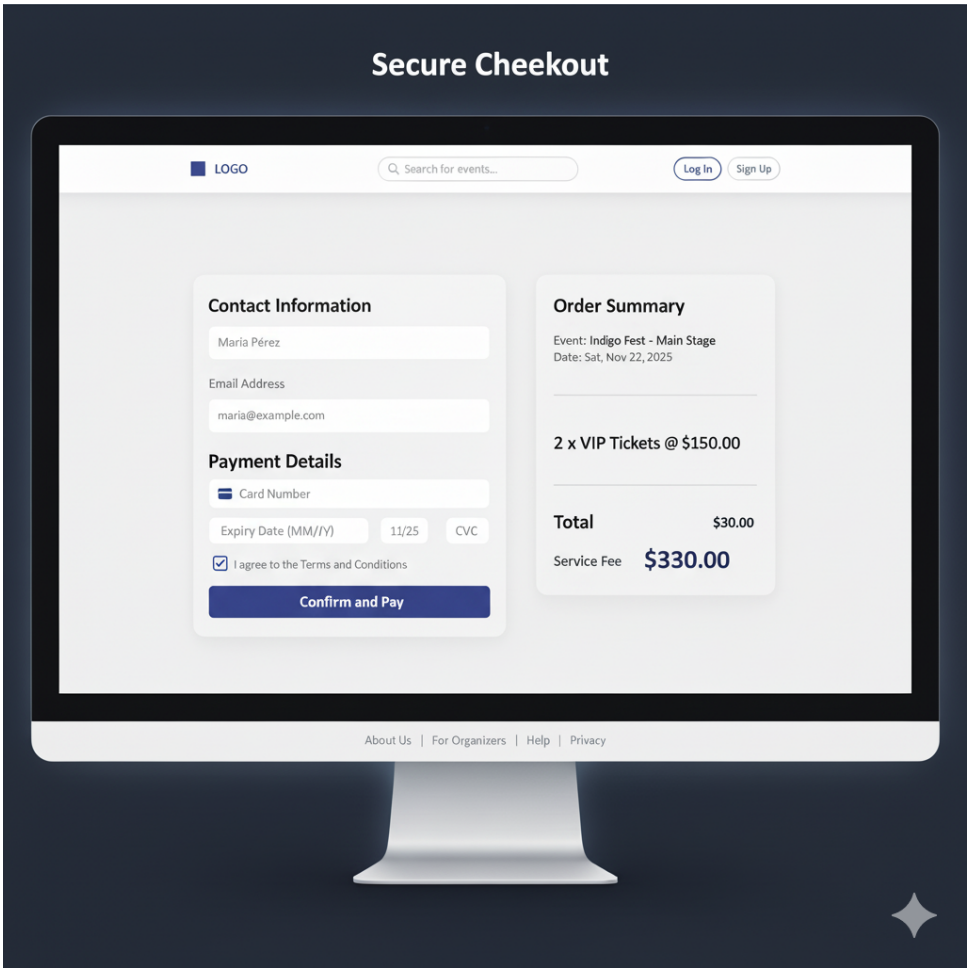


Figure 11: Mockup: Secure Checkout Process.

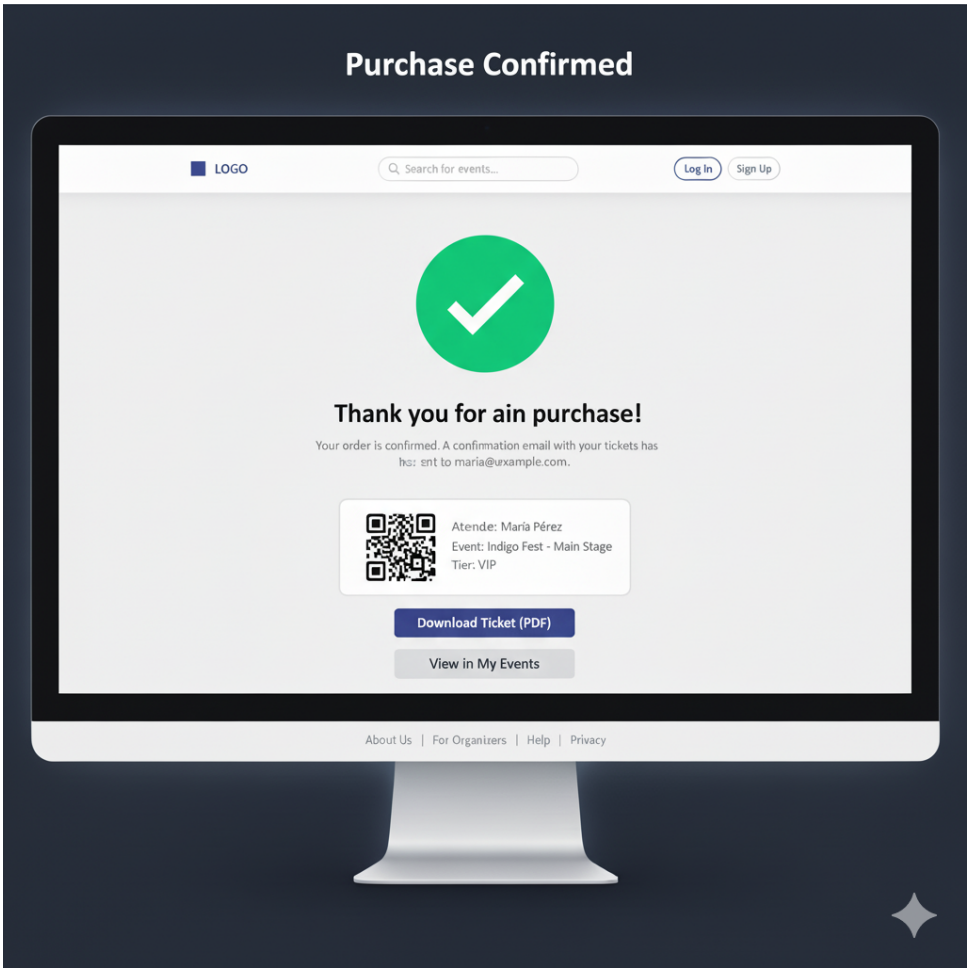


Figure 12: Mockup: Purchase Confirmed Page.

Acknowledgements

This project would not have been possible without the invaluable support and guidance of several individuals and organizations.

First and foremost, we extend our sincere gratitude to our supervisor, **Carlos Andrés Sierra**, for his expert advice, continuous encouragement, and insightful feedback throughout the entire design and architectural process. His experience and dedication were instrumental in shaping the direction and quality of this report.

We would also like to thank the **Department of Computer Engineering at Universidad Distrital Francisco José de Caldas** for providing the academic framework and resources necessary for this research.

Finally, we are deeply grateful to our families and friends for their unwavering patience, understanding, and support during the demanding phases of this project.

Glossary

ACID *Atomicity, Consistency, Isolation, Durability.* Un conjunto de propiedades que garantizan que las transacciones de bases de datos se procesen de forma fiable.

Agile Un enfoque iterativo e incremental para la gestión de proyectos y el desarrollo de software, que enfatiza la colaboración y la respuesta al cambio.

API *Application Programming Interface.* Un conjunto de definiciones y protocolos que se utiliza para diseñar e integrar software de aplicación, permitiendo que diferentes sistemas se comuniquen entre sí.

Architecture (Software) La estructura fundamental de un sistema de software, que incluye sus componentes, sus relaciones y los principios que guían su diseño y evolución.

Backend La parte de una aplicación de software que se encarga de la lógica del negocio, la gestión de la base de datos y la interacción con el servidor, no directamente visible para el usuario final.

BMC *Business Model Canvas.* Una herramienta de gestión estratégica que permite describir, diseñar, desafiar e inventar modelos de negocio.

BPMN *Business Process Model and Notation.* Una notación gráfica estandarizada para especificar procesos de negocio en un flujo de trabajo, facilitando su comprensión y comunicación.

Business Logic Layer La capa de una arquitectura de software que contiene las reglas de negocio, validaciones y procesos que definen cómo se transforman los datos y cómo la aplicación responde a las interacciones del usuario.

Cloud Computing La entrega de servicios informáticos (servidores, almacenamiento, bases de datos, redes, software, análisis, inteligencia) a través de Internet ("la nube").

Containerization Un método de virtualización a nivel de sistema operativo para desplegar y ejecutar aplicaciones y sus dependencias de forma aislada, garantizando consistencia en diferentes entornos.

Data Access Layer (DAL) La capa de una arquitectura de software responsable de manejar la interacción con los mecanismos de almacenamiento de datos (ej., bases de datos), aislando la lógica de negocio de los detalles de persistencia.

Database Una colección organizada de información estructurada, o datos, típicamente almacenada electrónicamente en un sistema informático.

Deployment Architecture La descripción de cómo los componentes de software de un sistema se asignan y organizan en la infraestructura física o virtual donde se ejecutan (servidores, contenedores, red).

Docker Una plataforma de código abierto que facilita la creación, el despliegue y la ejecución de aplicaciones utilizando contenedores, permitiendo la portabilidad y la consistencia del entorno.

Frontend La parte de una aplicación que interactúa directamente con el usuario, incluyendo la interfaz gráfica de usuario (GUI) y la lógica del lado del cliente.

GDPR *General Data Protection Regulation.* Un reglamento de la Unión Europea sobre protección de datos y privacidad para todos los individuos dentro de la UE y el Espacio Económico Europeo.

Hybrid Data System Una arquitectura de base de datos que combina diferentes tipos de bases de datos (ej., relacionales y NoSQL) para aprovechar las fortalezas de cada una según las necesidades de los datos.

Microservices Architecture Un estilo arquitectónico que estructura una aplicación como una colección de servicios pequeños, autónomos y acoplados de forma flexible, cada uno ejecutándose en su propio proceso.

MongoDB Una base de datos NoSQL orientada a documentos, utilizada para manejar grandes volúmenes de datos no estructurados o semiestructurados, y por su escalabilidad horizontal y flexibilidad de esquema.

Mockup Una representación estática y de alta fidelidad de cómo se verá una interfaz de usuario, centrándose en el diseño visual y la estética.

MVP *Minimum Viable Product*. La versión de un nuevo producto que tiene las características suficientes para satisfacer a los primeros clientes y proporcionar retroalimentación para el desarrollo futuro del producto.

NoSQL *Not Only SQL*. Una categoría de bases de datos que proporciona un mecanismo para almacenar y recuperar datos de formas diferentes a las de las bases de datos relacionales, a menudo para escalar horizontalmente y manejar datos no estructurados.

PCI DSS *Payment Card Industry Data Security Standard*. Un estándar de seguridad de la información para organizaciones que manejan información de tarjetas de crédito de marcas importantes, como Visa y MasterCard.

PostgreSQL Un sistema de gestión de bases de datos relacionales de objetos (ORDBMS) de código abierto, conocido por su fiabilidad, robustez de características y rendimiento.

Presentation Layer La capa de una arquitectura de software responsable de la interacción con el usuario, incluyendo la interfaz de usuario (UI) y la forma en que la información se presenta al usuario.

Relational Database Una base de datos que organiza los datos en tablas, que tienen un esquema predefinido y están interconectadas por relaciones, permitiendo la integridad de los datos a través de transacciones ACID.

Scrum Un marco de trabajo ágil para desarrollar y mantener productos complejos, con un enfoque en la entrega iterativa e incremental.

Service-Oriented Architecture (SOA) Un estilo arquitectónico de software que promueve la construcción de aplicaciones utilizando servicios acoplados de forma flexible.

SQL *Structured Query Language*. Un lenguaje de dominio específico utilizado en programación y diseñado para gestionar datos en un sistema de gestión de bases de datos relacionales (RDBMS).

Three-Layer Architecture Un patrón de arquitectura de software que organiza la aplicación en tres capas lógicas: Presentación, Lógica de Negocio y Acceso a Datos, para mejorar la modularidad y la mantenibilidad.

UML *Unified Modeling Language*. Un lenguaje de modelado estandarizado para especificar, visualizar, construir y documentar artefactos de sistemas de software, como diagramas de clases o de actividad.

User Story Una descripción informal y de propósito general de una característica de software, escrita desde la perspectiva del usuario final o de un rol específico, y con enfoque en el valor que aporta.

User Story Map Una herramienta de desarrollo ágil que ayuda a organizar las historias de usuario de una manera visual para representar el recorrido del usuario a través de un sistema y priorizar las características.

UI *User Interface*. La interfaz con la que los usuarios interactúan con un sistema de software o hardware, incluyendo elementos visuales y de control.

UX *User Experience*. La experiencia general de una persona al utilizar un producto, sistema o servicio, incluyendo sus emociones, percepciones y respuestas.

Wireframe Una representación esquemática de bajo nivel de una interfaz de usuario, centrándose en la estructura, el diseño y el contenido, sin detalles visuales ni de interactividad.

XP *Extreme Programming*. Una metodología ágil de desarrollo de software que busca mejorar la calidad del software y la capacidad de respuesta a los requisitos cambiantes del cliente.

Contents

List of Figures