SportGear Online: A Full-Stack E-Commerce Platform Developed Through Modern Software Engineering Practices

Juan Esteban Carrillo García
Dept. of Systems Engineering
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
Email: jecarrillog@udistrital.edu.co

Alejandro Sebastián González Torres
Dept. of Systems Engineering
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
Email: asgonzalezt@udistrital.edu.co

Miguel Ángel Babativa Niño
Dept. of Systems Engineering
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
Email: mababativan@udistrital.edu.co

Abstract—The development of end-to-end web applications requires integrating business modeling, software architecture, testing, and deployment into a cohesive engineering process. This paper presents SportGear Online, a complete academic e-commerce platform implemented using a distributed architecture with two backend services (Java/Spring Boot and Python/FastAPI), a React frontend, containerized deployment, and an automated CI/CD pipeline. The project follows modern engineering practices including user-story-driven analysis, UML-based architectural modeling, Test-Driven Development (TDD), Behavior-Driven Development (BDD), and Docker-based orchestration. The resulting system demonstrates how engineering methods can be combined to deliver a functional, modular, and reproducible solution aligned with professional development standards.

Index Terms—E-commerce, Software Engineering Education, DevOps, Full-Stack Development, Distributed Architectures

I. INTRODUCTION

Modern software engineering emphasizes reproducibility, modularity, and automation across the full development lifecycle. These principles are increasingly important for web applications, where reliability, scalability, and continuous deployment are now standard expectations. The sports retail sector, driven by rapid digital transformation, presents an ideal scenario for illustrating these engineering practices through the implementation of an e-commerce platform.

This paper describes the full-stack development of **SportGear Online**, a capstone project built as part of the Software Engineering Seminar at Universidad Distrital Francisco José de Caldas. The platform allows users to register, authenticate, browse sports products, manage a shopping cart, and complete orders. The project is intentionally designed to integrate heterogeneous technologies, ensuring students experience real-world architectural decisions.

Previous academic work often isolates specific techniques—such as TDD, microservices, or DevOps pipelines—without demonstrating how they unify into a single system. This work contributes a holistic case study where Scrum, user stories, architectural modeling, testing, and CI/CD automation converge into a functional product.

Following the instructor's recommendations, this paper focuses on documenting the methodology, system design, implementation, testing, and deployment evidence.

II. METHODS AND MATERIALS

A. Business Modeling and Requirements

The project began with a structured requirements process using industry-standard tools. User stories were defined to cover authentication, product browsing, cart management, and order checkout. These were organized into a User Story Map to visualize customer flows and delivery priorities. From these stories, system actors and responsibilities were identified, enabling a domain-driven analysis phase.

B. System Overview

Before formal modeling, a high-level domain overview was constructed to clarify the system's conceptual modules: User Management, Product Catalog, Shopping Cart and Checkout, Order Processing, Payment Processing, and Inventory Management. Figure 1 illustrates this conceptual decomposition and the relationships between domains.
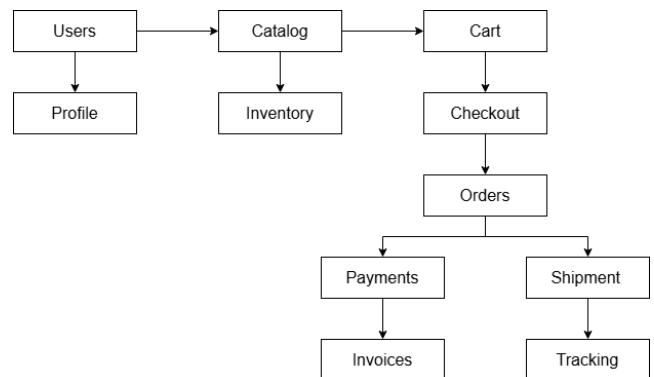

Fig. 1. System Overview Diagram

C. Architectural Modeling

A UML-based design was produced from the domain overview. The central artifact is the Class Diagram, which defines entities, their responsibilities, and interactions. To ensure clarity, the diagram was separated into domains rather than combining all classes in a multiples views according to his function structure.
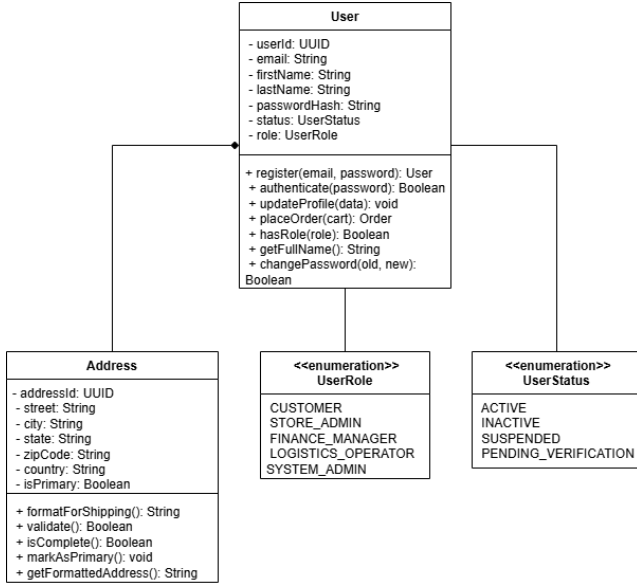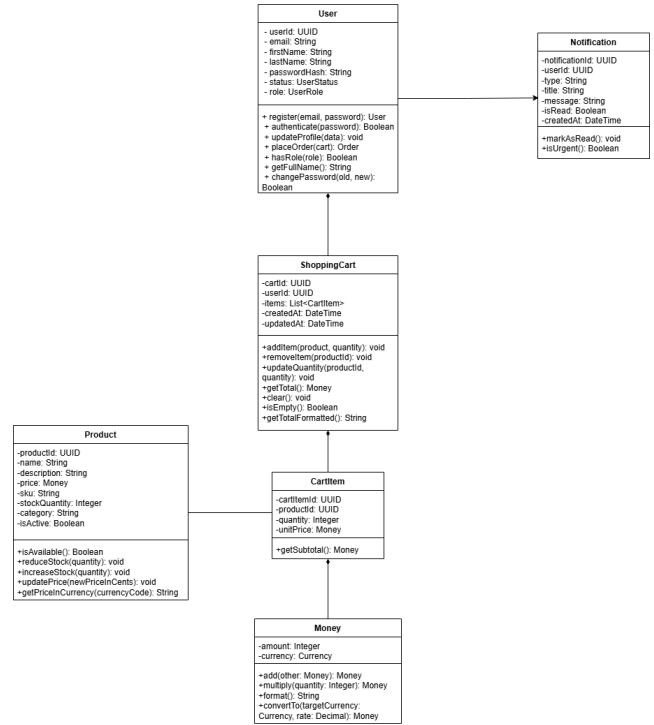

Fig. 2. Users Domain Diagram

Fig. 3. Products Domain Diagram
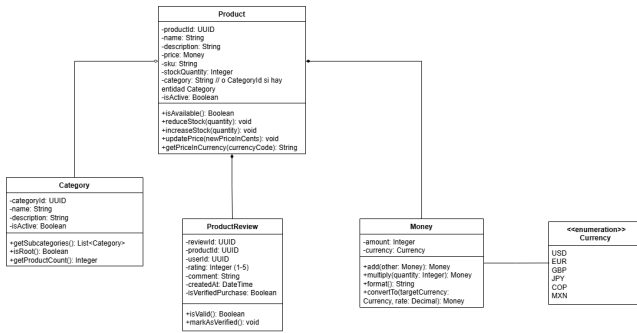

Fig. 4. ShoppingCart Domanin Diagram

Fig. 5. Orders Domain Diagram

D. System Architecture

The implemented architecture consists of three main components:

- **Java/Spring Boot Backend**: Handles authentication, session management, and security using JWT.
- **Python/FastAPI Backend**: Implements core business logic: products, cart, orders, and payments.
- **React Frontend**: Provides a dynamic web interface consuming REST APIs from both backends.

All services are containerized using Docker and orchestrated using Docker Compose. Figure 7 shows the corrected architecture diagram required by the instructor.
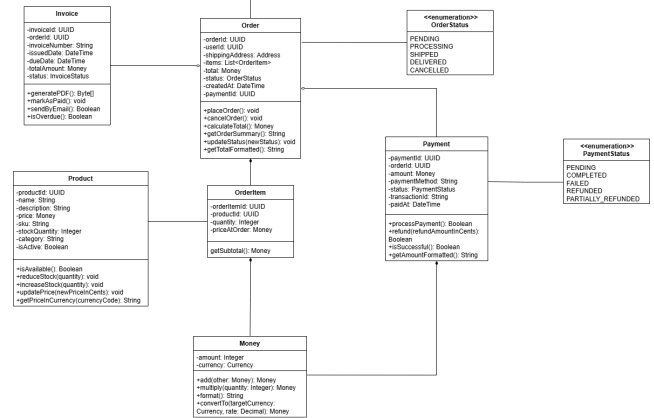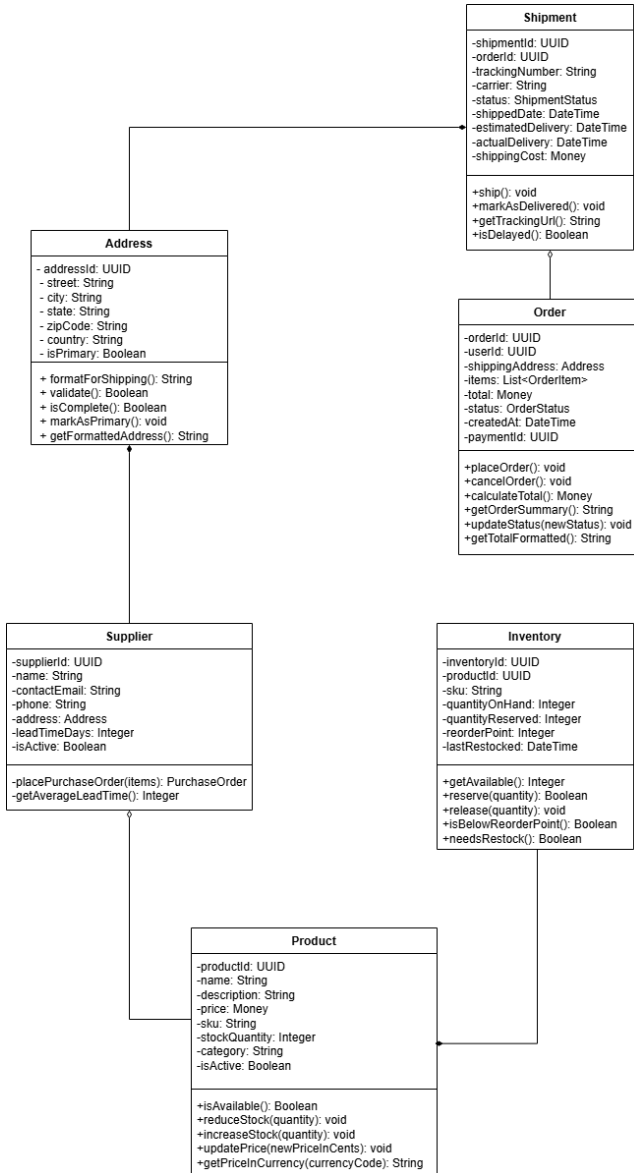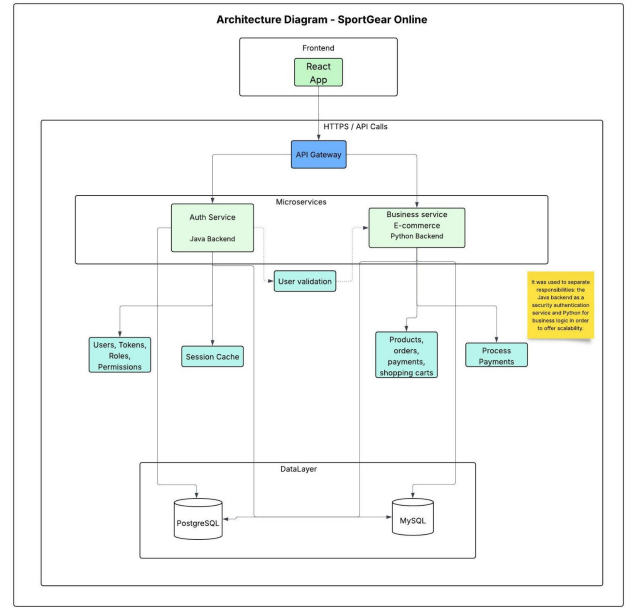
Fig. 6. Inventory Domain Diagram

Fig. 7. System Architecture

The project incorporates a complete CI/CD pipeline implemented through GitHub Actions. This workflow is automatically triggered on each push or pull request to the main branch. The pipeline initiates with two parallel test jobs, `test-java` and `test-python`, which execute the JUnit and pytest suites to validate authentication, business logic, and API functionality. Only when these tests succeed does the pipeline advance to the containerization stage, where Docker images are built for all services, guaranteeing consistent deployable artifacts.

A final integration stage deploys the multi-service environment using Docker Compose and verifies the interaction between components, including REST API communication, database connectivity, and security workflows such as JWT validation. Although no graphical pipeline diagram is included, this automated workflow demonstrates a complete, reproducible, and professionally aligned DevOps process that ensures software reliability across the entire development lifecycle.

III. RESULTS AND DISCUSSION

A. Functional Implementation

The system implements all core features planned in the user stories. The authentication backend exposes secure endpoints (`/api/auth/register`, `/api/auth/login`), while the FastAPI service supports product listing, cart updates, and checkout operations. The React frontend integrates these services into a functional user interface, validated through manual tests.

B. Testing Results

Testing provided quantitative evidence of system reliability. The Java backend achieved complete test coverage for authentication logic, while the Python backend achieved over 70%

E. Development Methodology

The system was implemented following Test-Driven Development (TDD). The Java backend was tested with JUnit 5, while the Python backend used pytest and behavior-driven tests with pytest-bdd. Cucumber-style acceptance tests validated user story compliance. API endpoints were verified using Postman.

F. Deployment and Automation

The deployment strategy of SportGear Online relies on full containerization and an automated continuous integration workflow. All system components—both backends, the frontend, and their respective databases—are packaged as Docker containers and orchestrated using Docker Compose, ensuring reproducible environments for development and testing.

code coverage across unit and behavior tests. All endpoints behaved as expected during Postman validation. JMeter stress tests verified performance under parallel requests.

C. CI/CD and Deployment Validation

The GitHub Actions pipeline executed successfully, confirming compatibility between services and reproducible builds. Docker Compose correctly orchestrated the multi-container system, validating the system's deployability and adherence to DevOps principles.

D. Discussion

The project demonstrates the viability of integrating heterogeneous technologies into a coherent engineering workflow. The separation of concerns between authentication and business logic simplifies scaling and maintenance. The use of TDD and BDD strengthened code correctness, and CI/CD automation ensured reproducibility. Limitations include the absence of cloud deployment and incomplete performance benchmarking, which remain opportunities for future work.

IV. Conclusions

This paper presented the design and implementation of SportGear Online, a full-stack e-commerce platform developed using modern software engineering practices. The project successfully integrated distributed architectures, REST API communication, automated testing, and CI/CD pipelines. The methodology followed—combining business modeling, UML design, TDD/BDD, and containerized deployment—proved effective for delivering a functional and maintainable system. Future improvements include cloud deployment, enhanced test coverage, UI refinement, and advanced data-driven features.

Acknowledgments

References

[1] G. Dašić, "Digital Transformation in the Sports Industry," in *Proc. First Int. Sci. Conf. SPORTICOPEDIA-SMB2023*, Belgrade, 2023.
[2] K. Beck, *Test-Driven Development: By Example*. Addison-Wesley, 2003.
[3] Docker Documentation, "Docker Compose Overview," 2024. [Online].
[4] GitHub, "GitHub Actions Documentation," 2025. [Online].