DriveMaster: Subscription-based Management System for Colombian Driving Academies

Cristian Arturo Parra Gonzales - 20201578102
David Gerardo Díaz Gómez - 20201020087
Daniel Mateo Montoya González - 20202020098
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia

***Abstract*—Driving academies in Colombia face difficulties in complying with national regulations and efficiently managing processes such as student registration, progress tracking, and certification. To address this issue, we developed *DriveMaster*, a Subscription-as-a-Service (SaaS) platform that integrates academic management, instructor and vehicle control, and automated certificate generation through a microservices architecture built with Spring Boot, FastAPI, and Angular. The results demonstrate that the system centralizes administrative operations, improves regulatory compliance, and optimizes certification times for driving schools nationwide.**

I. Introduction

In Colombia, driving academies are required to meet strict governmental standards established by entities such as RUNT and the Ministry of Transport, which define mandatory theoretical and practical hours for each driving license category (A1–C3). Despite this, most academies still depend on manual or fragmented systems that make it difficult to ensure compliance, track student progress, and manage instructors and vehicles effectively. This situation often leads to administrative inefficiencies, human errors, and delays in certification. To overcome these limitations, *DriveMaster* was created as a web-based Subscription-as-a-Service (SaaS) platform that automates the entire operational and regulatory workflow of driving schools. Its architecture follows a microservices model composed of a Spring Boot authentication service, a FastAPI backend for business logic, and an Angular frontend for user interaction, supported by PostgreSQL and Redis for data and session management. This paper presents the system's conceptual design, implementation approach, and validation through unit, acceptance, and performance testing, demonstrating its capacity to improve regulatory adherence and streamline operations across multiple driving academies in Colombia.

II. Methods and Materials

The proposed solution, **DriveMaster**, was conceived as a platform focused on the **academic and administrative management of driving schools in Colombia**, aiming to automate essential processes and ensure compliance with national regulatory requirements. Its design seeks to optimize core tasks such as student registration, class scheduling, and progress tracking, reducing staff workload and improving the traceability of the training process.

The functional design of the system was based on a set of **user stories** written in *Gherkin format (Given–When–Then)*, which describe the expected behaviors of each role. These stories were prioritized to cover the most critical functionalities within the available development period (three weeks), selecting only the essential workflows that constitute a **Minimum Viable Product (MVP)**.

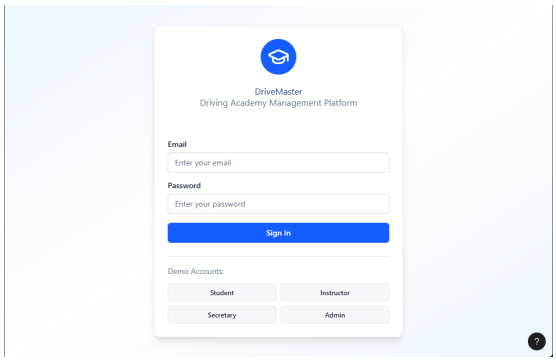
Four simplified roles were defined for the MVP:

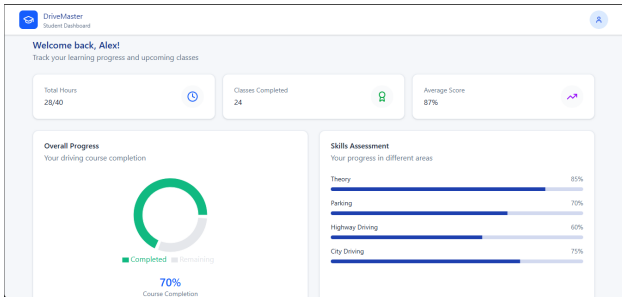
- **Secretary:** registers new students, assigns the license category (e.g., C1 or A2), monitors student progress, and generates the final certificate once the required hours are completed.
- **Instructor:** accesses their schedule, marks sessions as completed, or records absences; these actions automatically update the student's progress.
- **Student:** views theoretical and practical progress through progress bars and downloads the certificate upon completing all requirements.
- **Administrator:** manages user accounts (create or deactivate basic accounts), adjusts critical regulatory parameters (e.g., required hours per license type), and generates summarized activity reports for internal supervision. Administrative capabilities were limited to the minimum set necessary to maintain control and enable system testing during the short development period.

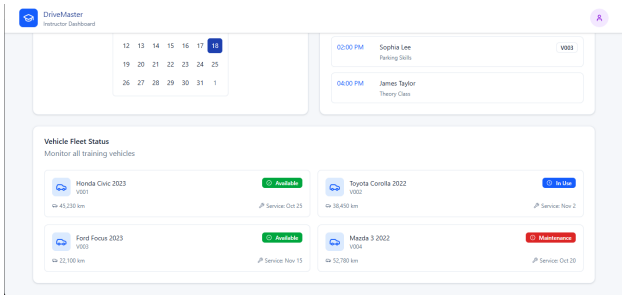
These user stories and their prioritization served as the foundation for defining the **class model** and the main **development modules**. Each functionality was designed to be verifiable through unit testing (*JUnit/pytest*) and automated acceptance testing (*Cucumber*), maintaining clear traceability between requirements and implementation.

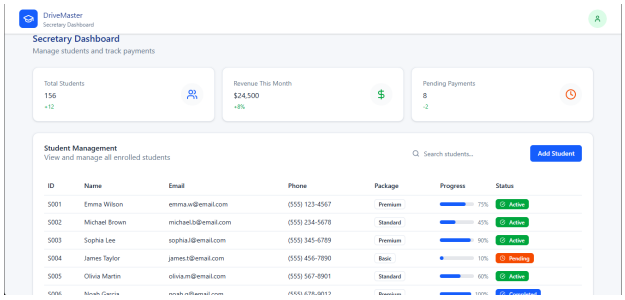
In parallel, **mockups** were produced for the login flow and the main role-based dashboards. These visual prototypes were used to validate usability and narrow the visual scope of the MVP. The login mockup defines the authentication and password recovery flow, while the role dashboards present minimalistic and task-focused interfaces (registration and certificates for the Secretary, scheduling for the Instructor, progress visualization for the Student, and basic controls for the Administrator).

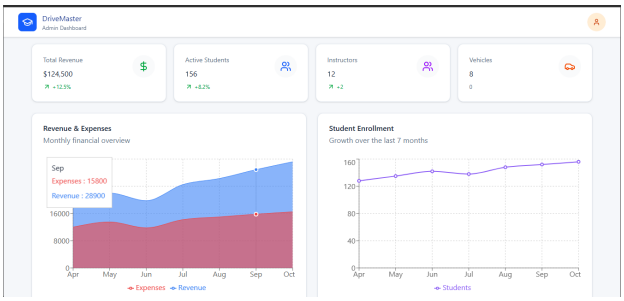
The proposed technical architecture includes a **Java-based authentication service**, a **Python backend** implementing the business logic and regulatory validation, and an **Angular frontend** that materializes the mockups. This modular approach facilitates parallel development, supports incremental deliveries, and ensures that the MVP can be realistically completed within the three-week timeframe.











The **DriveMaster** class diagram represents the logical structure of the system and defines its main entities, their responsibilities, and the relationships among them. This model was built from the user stories and CRC cards developed during the design phase, ensuring consistency between functional requirements and the object-oriented implementation.

The main classes that compose the system are described as follows:

- **Student:** Represents the students enrolled in the academy. It stores basic information, license type, theoretical and practical hours, and academic status. Its methods allow updating class progress and requesting the certificate once all regulatory requirements are met.
- **CourseRegulation:** Defines the rules governing each license type (A1, B1, C1, etc.), specifying the required theoretical and practical hours. It also validates whether a student meets certification criteria and can be updated by the system administrator.
- **Schedule:** Manages the scheduling of classes, linking students, instructors, and vehicles. It controls the state of each session (scheduled, completed, or absent) and automatically updates the student's progress.
- **Instructor:** Represents the personnel responsible for conducting the classes. It records availability, license type, and assigned sessions. Instructors can mark completed classes, register absences, and report vehicle incidents.
- **Vehicle:** Manages the vehicles available for driving practice. Each vehicle has attributes such as plate number, type, and status, and can be marked as "Under Maintenance" if an issue is reported.
- **Admin:** Centralizes system configuration operations, including user management, modification of regulatory parameters (e.g., required hours per license type), and the generation of global activity reports.

Main Relationships Between Classes:

- A **Student** can have multiple records in **Schedule** (1:N).
- An **Instructor** can be assigned to multiple classes (**Schedule**).
- A **Vehicle** can participate in several sessions (**Schedule**).
- A **CourseRegulation** is associated with multiple **Student** entities.
- The **Admin** can modify instances of **CourseRegulation**.

Conceptual Description: The design follows the principles of **object-oriented programming**, where each class encapsulates a specific responsibility and collaborates with others

through well-defined relationships. This model ensures modularity, code reusability, and strong traceability with the user stories.

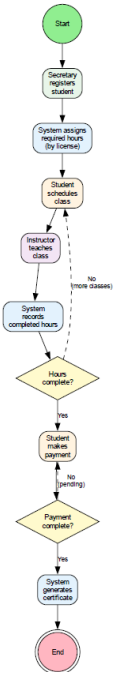
For instance, the story "When the student completes 100% of the required hours and pays all fees, the system allows certificate generation" is reflected in the relationship between the *Student* and *CourseRegulation* classes. Likewise, the instructor's action of marking a class as completed is represented by the interaction between *Instructor* and *Schedule*.

Thus, the class diagram not only defines the system structure but also documents the direct connection between functional analysis and technical implementation, serving as a bridge between conceptual design and development.

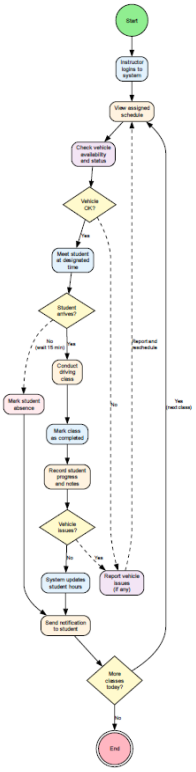
A. Business Process Modeling

To complement the class model and ensure alignment between functional requirements and technical implementation, several **Business Process Models (BPMN)** were developed. These diagrams describe the operational workflows of the main system actors —Student, Instructor, and Administrator— allowing a clear visualization of how each interacts with the system components.

Student Process Model: The **Student Process Model** represents the complete academic workflow from registration to certification. The process begins when the Secretary registers a new student, after which the system automatically assigns the required theoretical and practical hours based on the selected license category. The student can then schedule practical sessions, attend classes with an instructor, and accumulate progress hours. Once all required hours are completed and payment is confirmed, the system automatically generates the final certificate following national driving standards.



Instructor Process Model: The **Instructor Process Model** details the workflow of teaching personnel. The process begins with instructor authentication, followed by checking the daily schedule and verifying vehicle availability. During class sessions, the instructor can mark attendance, set a session as completed, or report vehicle incidents. In case a student is absent, the instructor can register the absence, which triggers an automatic notification to the student. This process ensures synchronization between instructor actions and the backend business logic.



Subscription Process Model: The **Subscription Process Model** defines the administrative workflow for academy registration and access management. The process starts when a driving academy requests access to the platform. After the Administrator reviews the request and confirms payment, the system activates the corresponding license and sends secure credentials to the institution. This process enforces permission control and lays the foundation for future multi-institution scalability.

Together, these business process models establish a strong link between functional requirements and technical implementation. They also validate the structure defined in the **class diagram** (Figure **??**), as each process directly corresponds to the interactions among the main entities: *Student*, *Instructor*, *CourseRegulation*, and *Admin*. This dual representation —BPMN and UML— reinforces traceability, maintainability, and scalability of the **DriveMaster** system, allowing the platform to evolve without compromising design coherence.

REFERENCES

[1] Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," OMG Specification, January 2011. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/

[2] Object Management Group, "Unified Modeling Language (UML) Specification, Version 2.5.1," December 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/

[3] Google LLC, "Angular Documentation," 2024. [Online]. Available: https://angular.dev

[4] Python Software Foundation, "Python Documentation," 2024. [Online]. Available: https://docs.python.org/3/

[5] IEEE Computer Society, "IEEE Standard for Software and Systems Engineering — Software Life Cycle Processes," IEEE Std 12207-2017, 2017.

[6] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide," 2nd ed., Addison-Wesley, 2005.