République Tunisienne

Ministère de l'Enseignement Supérieur

et de la Recherche Scientifique

École Supérieur Privée d'ingénierie et de technologie

TEK-UP

Rapport de Projet de Fin d'Études

Présenté en vue de l'obtention du

Diplôme National d'Ingénieur en Informatique

Spécialité : Genie Logiciel

Réalisé par

BOUSSAIDI Bayrem

Système de Location en Ligne avec Interaction Multi-Utilisateurs et Gestion Dynamique des Réservations

Encadrant professionnel :	**M. ZELLIT Mootaz**	Ingénieur Informatique
Encadrant académique :	**Mme. JALEL Sawsen**	Poste

Année Universitaire 2024 - 2025

J'autorise l'étudiant à faire le dépôt de son rapport de stage en vue d'une soutenance.

Encadrant professionnel, **M. XXXXXXX**

Signature et cachet

J'autorise l'étudiant à faire le dépôt de son rapport de stage en vue d'une soutenance.

Encadrant académique, **M. YYYYYY**

Signature

Thanks

I dedicate this work :

To my dear mother Zahida,
Thank you for your unconditional love and unwavering support that have guided me throughout this journey.
Your love has been the light that showed me the way, and I am forever grateful to you.

To my dear father Ali,
Your strength, hard work, and encouragement have always inspired me to give my best.
You have been a model of perseverance and determination.

To my brother Ahmed and my sister Tasnim,
Your presence and support have been invaluable to me.
Thank you for all the moments we have shared and for your precious care.

This work is dedicated to all of you, with all my gratitude and affection.

– Bayrem Boussaidi –

Table des matières

Table des figures

Liste des tableaux

List of Abbreviations

API = Application Programming Interface

CI/CD = Continuous Integration / Continuous Deployment

CRUD = Create, Read, Update, Delete

DB = Database

DoD = Definition of Done

DT = Development Team

ER = Entity-Relationship

GLSI = Software Engineering and Information Systems

HTTP = Hypertext Transfer Protocol

JWT = JSON Web Token

K8s = Kubernetes

MoSCoW = Must have, Should have, Could have, Won't have

MVC = Model-View-Controller

OAuth = Open Authorization

PB = Product Backlog

PDF = Portable Document Format

PO = Product Owner

QR = Quick Response (Code)

REST = Representational State Transfer

SB = Sprint Backlog

SM = Scrum Master

SMTP = Simple Mail Transfer Protocol

SOA = Service-Oriented Architecture

SP = Story Points

UI = User Interface

US = User Story

UX = User Experience

YAML = YAML Ain't Markup Language

General Introduction

The automotive rental industry has undergone significant transformation in recent years, driven by the increasing demand for digital solutions that streamline operations and enhance user experience. In this context, we developed **MyLoc**, a comprehensive car rental platform designed to connect administrators, rental agencies, and customers in a seamless digital ecosystem.

Our platform provides a complete solution for car rental management, enabling agencies to list their vehicles, manage their inventory, and process rental requests efficiently. Administrators oversee the entire platform, managing agencies, customers, and content while maintaining direct communication with agencies through an integrated real-time chat system.

The rental workflow is streamlined : when a customer submits a rental request for a specific vehicle, the responsible agency receives the request and can either approve or decline it. Upon approval, the customer receives an email containing a PDF rental contract with all booking details and a unique QR code for verification. In case of rejection, a notification email is sent to inform the customer of the decision. Additionally, in-app notifications keep customers informed of their request status in real-time.

The platform also features a blog section where administrators can publish news and special offers, with customers able to comment and engage with the content. A **Follower subscription system** allows visitors to enter their email in the website footer to receive automatic email notifications whenever new cars are added to the platform.

To enhance customer support, we integrated an AI-powered chatbot using **Flask API** (Python) connected to ChatGPT, providing instant answers to visitor and customer inquiries in multiple languages.

Our application is built with modern technologies : **Angular** for the frontend, **Spring Boot** (REST architecture) for the backend, and **MySQL** for data persistence. Security is managed through **Keycloak** for authentication, with AuthGuard protecting admin and agency routes based on user roles.

For deployment, the entire infrastructure is containerized using **Docker** and **Docker Compose**, then orchestrated on **Kubernetes** (Minikube) with dedicated deployment files for each service. A complete **CI/CD pipeline** using **GitLab CI** automates the build, test, and deployment processes.

Development was conducted using **Visual Studio Code** as the IDE, **XAMPP** for local database management, and **Postman** for API testing. The project followed the **Scrum** methodology with organized sprints and backlogs to ensure iterative and reliable delivery.

This report details the design, development, and deployment approaches adopted for the MyLoc platform. It is organized as follows :

— **Chapter 1** : Presents the project framework and general context, analyzes existing car rental solutions, and introduces the Scrum methodology adopted for project management.

— **Chapter 2** : Covers requirements analysis and specification, including actor identification, functional and non-functional requirements, use case diagrams, class diagrams, and system architecture.

— **Chapter 3** : Details the first release implementation focusing on user authentication (Keycloak integration), car management, and agency management features.

— **Chapter 4** : Presents the second release with rental request processing, email notifications, PDF contract generation with QR codes, and the real-time chat system.

— **Chapter 5** : Covers the third release including the AI chatbot integration, blog management, customer reviews, and notification system.

— **Chapter 6** : Describes the DevOps implementation with Docker containerization, Kubernetes deployment, and GitLab CI/CD pipeline setup.

Chapitre 1

Project Context

Plan

1.1 Introduction

This chapter presents the general context of the project. It introduces the host organization, **Myloc Agency**, analyzes existing methods, and describes the proposed solution. Additionally, it outlines the project architecture, design approaches, management methodology, and development environment.

1.2 Project Problematic

Before the development of this application, car rental agencies faced numerous challenges that hindered their efficiency, customer satisfaction, and business growth. This section identifies the key problems that motivated the creation of the MyLoc platform.

1.2.1 Challenges Faced by Agencies Before This Project

1.2.1.1 Manual and Time-Consuming Processes

Traditional car rental agencies relied heavily on manual processes :

— **Paper-based booking systems** : Agencies used physical logbooks or basic spreadsheets to track reservations, leading to errors, double bookings, and lost records.

— **Phone-only communication** : Customers had to call agencies during business hours to check availability, make reservations, or ask questions—limiting accessibility and creating bottlenecks.

— **Manual contract preparation** : Rental contracts were typed or handwritten for each customer, consuming significant time and prone to errors.

— **No real-time availability** : Agencies couldn't provide instant availability information, often requiring callbacks after checking physical records.

1.2.1.2 Poor Communication Channels

— **Disconnected stakeholders** : There was no unified platform for administrators, agencies, and customers to communicate efficiently.

— **Delayed responses** : Customer inquiries via email or phone often took hours or days to receive responses.

— **No instant messaging** : Agencies couldn't engage in real-time conversations with customers or administrators.

— **Language barriers** : International customers faced difficulties communicating with local

agencies.

1.2.1.3 Inefficient Request Management

— **No centralized request tracking** : Rental requests were scattered across emails, phone logs, and paper notes.
— **Slow approval process** : Agencies manually reviewed each request, often taking 24-48 hours to confirm or decline.
— **No automated notifications** : Customers had to repeatedly contact agencies to check their request status.
— **Lost opportunities** : Delayed responses led to customers booking with competitors.

1.2.1.4 Lack of Digital Presence

— **Limited visibility** : Small and medium agencies had minimal online presence, losing customers to larger competitors with websites.
— **No car showcase** : Agencies couldn't display their fleet with photos, specifications, and pricing online.
— **No filtering capabilities** : Customers couldn't search for specific car types, models, or price ranges.
— **No customer reviews** : Potential customers had no way to read feedback from previous renters.

1.2.1.5 Administrative Overhead

— **No centralized management** : Administrators managing multiple agencies had no unified dashboard.
— **Difficult reporting** : Generating reports on bookings, revenue, or customer activity required manual data compilation.
— **No subscriber management** : Agencies couldn't maintain lists of interested customers for marketing purposes.

1.2.2 Impact of These Problems

These challenges resulted in :

— **Revenue loss** : Slow processes and poor communication led to lost bookings.
— **Customer frustration** : Long wait times and lack of transparency drove customers away.
— **Operational inefficiency** : Staff spent excessive time on administrative tasks instead of

customer service.

— **Competitive disadvantage** : Agencies without digital tools fell behind tech-enabled competitors.

— **Scalability issues** : Manual processes couldn't handle growth in customer volume.

1.3 Project Objectives

The MyLoc platform was developed to address the problematic issues identified above. This section outlines the specific objectives that guided the development of the application.

1.3.1 Primary Objectives

1.3.1.1 Digitize the Car Rental Process

— Create a comprehensive web platform replacing manual booking systems.

— Enable 24/7 online access for customers to browse and book vehicles.

— Provide real-time car availability checking to prevent double bookings.

— Automate the entire rental workflow from request to contract generation.

1.3.1.2 Facilitate Communication Between Stakeholders

— Implement real-time chat between administrators and agencies for operational coordination.

— Enable instant email notifications for booking confirmations, rejections, and updates.

— Provide in-app notifications to keep customers informed of their request status.

— Integrate an AI-powered chatbot for instant customer support in multiple languages.

— Allow direct email contact between customers and administrators.

1.3.1.3 Streamline Request Management

— Create a centralized dashboard for agencies to view and manage all rental requests.

— Enable one-click approval or rejection of requests with automatic customer notification.

— Reduce response time from days to minutes through instant processing.

— Automatically generate PDF contracts with QR codes upon approval.

1.3.1.4 Enhance Customer Experience

— Provide intuitive car browsing with advanced filtering (type, model, price, availability).

— Display detailed car information with multiple photos.

— Allow customers to track their booking history and request status.

— Enable customers to leave reviews and comments on cars and blog posts.

— Send email updates to followers when new cars are added.

1.3.1.5 Empower Agency Management

— Provide dedicated dashboards for each agency to manage their fleet.

— Enable full CRUD operations on cars (add, edit, delete, upload photos).

— Give agencies control over their booking calendar and availability.

— Allow agencies to communicate directly with administrators.

1.3.1.6 Centralize Administration

— Create a comprehensive admin dashboard for platform oversight.

— Enable management of agencies, customers, and blog content from one interface.

— Provide tools for monitoring platform activity and user engagement.

1.3.2 Technical Objectives

1.3.2.1 Security and Authentication

— Implement robust authentication using Keycloak with role-based access control.

— Protect admin and agency routes with AuthGuard.

— Ensure secure data transmission and storage.

1.3.2.2 Modern Architecture

— Build a scalable microservices architecture with clear separation of concerns.

— Use Angular for a responsive, dynamic frontend.

— Implement Spring Boot REST APIs for robust backend services.

— Integrate FastAPI for the AI chatbot service.

1.3.2.3 DevOps and Deployment

— Containerize all services using Docker.

— Orchestrate deployment with Docker Compose and Kubernetes (Minikube).

— Automate CI/CD pipelines using GitLab CI.

— Ensure consistent environments across development and production.

1.4 Solutions Provided by MyLoc Platform

This section details how the MyLoc platform addresses each of the identified problems, transforming the car rental experience for all stakeholders.

1.4.1 Transforming Communication

1.4.1.1 Real-Time Admin-Agency Chat

The platform includes an integrated instant messaging system that enables :

— Direct communication between administrators and agencies.

— Real-time message delivery without page refresh.

— Message history for reference and accountability.

— Coordination on operational matters, promotions, and issues.

1.4.1.2 Automated Email Notifications

The system automatically sends emails for :

— **Request Acceptance** : Customer receives a confirmation email with PDF contract and QR code.

— **Request Rejection** : Customer receives a polite notification explaining the decline.

— **New Car Alerts** : Followers and customers receive updates when new vehicles are added.

— **Contact Confirmations** : Acknowledgment emails for customer inquiries.

1.4.1.3 AI-Powered Chatbot

The integrated chatbot, built with Flask API and connected to ChatGPT, provides :

— **24/7 Availability** : Instant responses at any time, even outside business hours.

— **Multilingual Support** : Assistance in multiple languages for international customers.

— **Instant Answers** : Quick responses to common questions about cars, pricing, and booking.

— **Reduced Staff Workload** : Automated handling of routine inquiries.

1.4.1.4 In-App Notifications

Customers receive real-time notifications within the application for :

— Rental request status changes (pending, approved, declined).

— New messages or updates from agencies.

— Important platform announcements.

1.4.2 Streamlining the Booking Process

1.4.2.1 Smart Availability Checking

Before submitting a rental request, the system :

— Checks the car's availability for the selected date range.

— Prevents double bookings automatically.

— Shows real-time availability status on car listings.

— Suggests alternative dates if the car is unavailable.

1.4.2.2 One-Click Request Processing

Agencies can process requests efficiently :

— View all pending requests in a centralized dashboard.

— Accept or decline with a single click.

— Automatic email sent to customer upon decision.

— PDF contract generated instantly for approved requests.

1.4.2.3 Automated Contract Generation

Upon approval, the system automatically :

— Generates a professional PDF rental contract.

— Includes all booking details (dates, car info, customer info, pricing).

— Creates a unique QR code for verification.

— Sends the contract via email to the customer.

1.4.3 Enhancing Car Discovery

1.4.3.1 Advanced Filtering System

Customers can filter cars by :

— **Vehicle Type** : SUV, sedan, compact, luxury, etc.

— **Brand/Model** : Search for specific manufacturers.

— **Price Range** : Set minimum and maximum daily rates.

— **Availability** : Show only cars available for selected dates.

— **Features** : Air conditioning, GPS, automatic transmission, etc.

1.4.3.2 Rich Car Profiles

Each car listing includes :

— Multiple high-quality photos.

— Detailed specifications (engine, seats, fuel type, transmission).

— Daily rental price and any additional fees.

— Customer reviews and ratings.

— Real-time availability calendar.

1.4.4 Saving Time for All Users

Task	Before MyLoc
Check car availability	Phone call, wait for callback
Submit rental request	Visit agency or phone
Get request response	24-48 hours
Receive rental contract	Visit agency to sign
Ask a question	Phone during business hours
Browse available cars	Visit multiple agencies
Agency-Admin communication	Email chains, phone calls

Tableau 1.1 : Time Savings Comparison : Before vs. After MyLoc

1.4.5 Empowering Agencies

1.4.5.1 Dedicated Agency Dashboard

Each agency has access to :

— **Fleet Management** : Add, edit, delete cars with full details and photos.

— **Request Center** : View and process all rental requests.

— **Communication Hub** : Chat with administrators, receive notifications.

— **Profile Management** : Update agency information and settings.

1.4.5.2 Increased Visibility

Agencies benefit from :

— Online presence without building their own website.

— Exposure to all platform visitors and customers.

— Customer reviews that build trust and reputation.

— Email marketing through the follower system.

1.4.6 Follower Subscription System

The platform includes a unique follower feature :

— Visitors enter their email in the website footer to subscribe.

— Followers receive automatic email notifications when new cars are added.

— No account creation required—low barrier to engagement.

— Keeps potential customers informed and engaged with the platform.

— Helps agencies reach interested customers with new offerings.

1.5 Presentation of the Host Organization : Myloc Agency

This section provides an overview of the host organization, its mission, activities, and products, highlighting its innovative approach and global impact.

1.5.1 Host Organization

Myloc Agency leverages expertise in car rental services to offer customized solutions that provide reliable and affordable transportation options for clients. Founded by Amine Ben Chaabane, it consists of a passionate and dedicated team of skilled professionals who work closely with clients to deliver transportation solutions tailored to their specific needs.

1.5.2 Activities and Products

As part of this project, I developed a full-featured car rental web application that supports three main user roles : Admin, Agencies, and Customers, with additional functionalities for visitors and followers. The application includes a secure, role-based system powered by Keycloak for authentication and route protection. It was built using Spring Boot (backend), Angular (frontend), FastAPI (chatbot backend), and a separate payment microservice, all containerized with Docker, orchestrated with Docker Compose, deployed on Kubernetes (K8s), and integrated with GitLab CI/CD pipelines for continuous deployment.

1.6 Project Frame

This section delves into the framework of the project, starting with an analysis of existing systems to identify their strengths and weaknesses. By understanding gaps and opportunities in current solutions, we propose an innovative approach that addresses these shortcomings and enhances the overall user experience.

1.6.1 Study of Existing Systems

This section explores existing car rental platforms to identify their functionalities, strengths, and limitations. The aim is to evaluate how current systems handle administrative, agency, and customer interactions, and to highlight areas where improvements can be made.

— **Traditional Car Rental Websites** : Many companies offer web platforms where users can search and book vehicles. These platforms often lack real-time availability checks, multi-role management, or automated document generation. Interaction between customers and agencies is usually limited or entirely missing.

- **Aggregator Platforms (e.g., Kayak, Rentalcars)** : These platforms centralize listings from multiple rental agencies, allowing users to compare prices and availability. While convenient for users, they do not provide backend control for agencies or real-time request management. Personalized communication is also limited.
- **Custom Enterprise Systems** : Some large agencies have internal systems for managing cars, customers, and payments. These solutions are often closed-source, expensive, and not scalable for small or medium-sized agencies. Many lack modularity, chat features, or automated workflows like PDF contract generation and QR code integration.
- **Limitations Identified** :
  - Lack of real-time communication between agencies and customers.
  - No integrated notification or email system for status updates.
  - Limited role-based access (admin, agency, customer).
  - Absence of automation in rental confirmation workflows (e.g., contract generation, online payment).
  - Poor support for multi-agency environments within a single system.

1.6.2 Gaps and Opportunities

This section identifies the key limitations in existing car rental systems and highlights opportunities for technical and functional improvement.

- **Lack of Multi-Role Interaction** : Most platforms do not support smooth interaction between different types of users. A robust role-based system with protected routes and custom dashboards is necessary.
- **Insufficient Real-Time Communication** : Many systems lack instant messaging features. Adding a built-in chat system can improve customer service and speed up booking confirmation.
- **Limited Automation** : Automated PDF contract generation, QR code creation, and email notifications are rarely offered. These features streamline workflows and enhance user satisfaction.
- **Weak Integration of Notifications and Updates** : Timely updates for bookings, new cars, and promotions are missing. An integrated notification system (email + in-app) is essential.
- **Lack of Dynamic Availability Checking** : Users can book cars without real-time validation, creating double bookings. Implementing date-based availability checks resolves this.
- **Limited Customization for Agencies** : Smaller agencies often lack control over fleet management or platform display. Individual dashboards with CRUD capabilities are an

opportunity.

— **Underuse of Modern Deployment Practices** : Few platforms use CI/CD pipelines, containerization, or Kubernetes, making scaling and maintenance difficult. Leveraging Docker, Docker Compose, K8s, and GitLab CI/CD ensures stability and faster updates.

Gap	Exists in current platforms
Multi-role management with custom permissions	LIMITED
Real-time messaging between users	RARE
Automatic notifications and email updates	LIMITED
Dynamic car availability checking	NO
PDF contracts with QR code generation	NO
Online payment integration with validation flow	LIMITED
Scalable deployment (CI/CD, Docker, K8s)	NO

Tableau 1.2 : Identified Gaps in Existing Car Rental Platforms

1.6.3 Proposed Solution

The proposed solution addresses the gaps identified in current rental platforms :

— **Role-Based System with Secured Access** : Supports multiple roles (admin, agency, customer, visitor) with dashboards and protected routes via Keycloak.

— **Real-Time Communication** : Built-in chat for agencies and customers.

— **Automated Notification and Email System** : Sends automatic updates for bookings, new cars, and account activities.

— **Contract Generation with QR Code** : PDF rental agreements generated and sent via email after successful payment.

— **Smart Availability Checker** : Validates car availability before confirming bookings.

— **Modern Infrastructure and Deployment** : Deployed using Docker Compose, Kubernetes, and GitLab CI/CD for scalability and maintainability.

— **Integrated Chatbot Assistant** : FastAPI-based chatbot using OpenAI for real-time user assistance.

1.7 Project Architectures and Design Approaches

1.7.1 System Architecture : Microservices with Three-Tier Logical Structure

The system follows a modern microservices architecture with clear separation of concerns across multiple tiers. The architecture ensures scalability, maintainability, and security through containerized deployment and CI/CD practices.

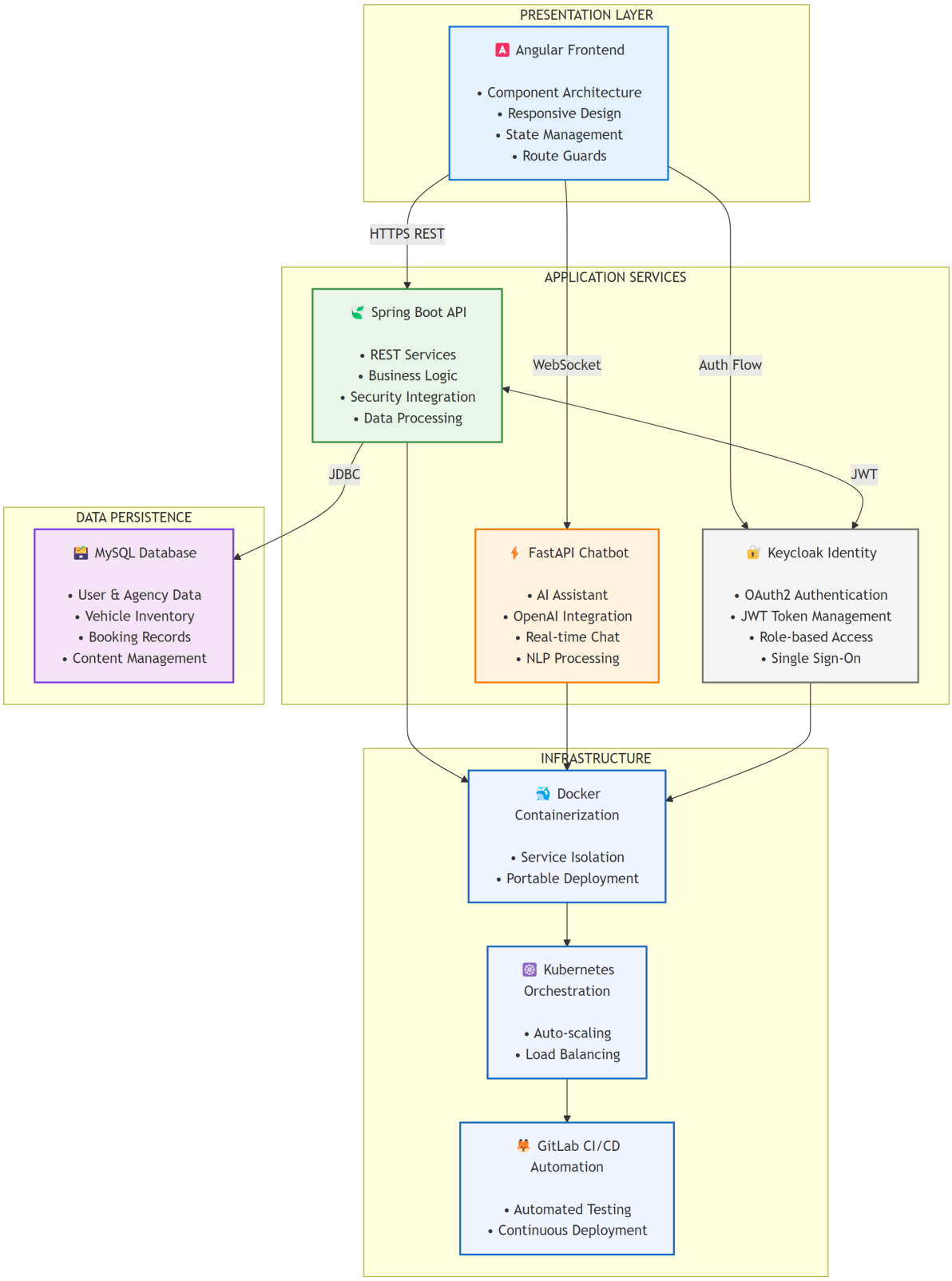

Figure 1.1 : System Architecture Overview - Myloc Agency Car Rental Platform

1.7.1.1 Architecture Layers

— **Presentation Layer (Frontend)** : Angular 16+ application providing responsive UI for all user roles. Features include dynamic routing, lazy loading, and role-based component

rendering. Protected by AuthGuard for secure route access.

— **Business/Application Layer (Backend Services)** :
  — **Main API (Spring Boot)** : Core backend handling cars, agencies, customers, bookings, comments, and notifications via RESTful endpoints.
  — **Payment Service (Spring Boot)** : Separate microservice for payment processing, contract generation, and QR code creation.
  — **Chatbot Service (Flask/FastAPI)** : AI-powered assistant integrated with ChatGPT API for customer support.
— **Security Layer (Keycloak)** : OAuth2/OpenID Connect authentication server managing user identities, roles, and access tokens.
— **Data Layer** : MySQL relational database ensuring data integrity, referential constraints, and complex queries for bookings and availability checks.

1.7.1.2 Service Communication Flow

1. User interacts with Angular frontend.
2. Frontend sends HTTP requests to Spring Boot API with JWT token.
3. Backend validates token with Keycloak.
4. Business logic executes, database queries processed.
5. For payments : Main API communicates with Payment microservice.
6. For AI assistance : Frontend calls Flask chatbot API.
7. Responses returned to frontend, UI updated.

1.7.1.3 Containerization and Deployment

— **Docker** : Each service (Angular, Spring Boot, Flask, MySQL, Keycloak) containerized separately.
— **Docker Compose** : Orchestrates all containers locally with defined networks and volumes.
— **Kubernetes (Minikube)** : Production-like orchestration with pods, services, deployments, and ingress controllers.
— **GitLab CI/CD** : Automated pipeline for testing, building Docker images, and deploying to Kubernetes.

1.7.2 Design Pattern : RESTful Service-Oriented Architecture (SOA)

— Stateless interactions, resource-oriented endpoints, loose coupling, reusability, and clear separation of concerns.

1.7.3 Design Pattern : MVC for Payment Service

— Model : Data entities and payment logic.

— View : JSON responses and webhook callbacks.

— Controller : API request handling, transaction validation, and workflow triggering (PDF/QR).

1.7.4 FastAPI Chatbot Microservice

— Lightweight RESTful endpoints for AI-powered chatbot interactions.

— Modular logic for maintenance and scaling.

— Containerized via Docker, deployed with Kubernetes.

1.8 Project Management and Design Methodology

1.8.1 Traditional vs Agile Approaches

Before selecting a methodology for this project, it is essential to understand the fundamental differences between traditional (classical) and agile approaches to project management.

Traditional methodologies (such as Waterfall, V-Model) follow a linear, sequential approach where each phase must be completed before the next begins. Requirements are defined upfront, and changes are difficult to incorporate once development starts. This approach works well for projects with stable, well-defined requirements but struggles with evolving needs.

Agile methodologies embrace change and iterative development. Work is divided into short cycles (sprints or iterations), allowing for continuous feedback, adaptation, and improvement. Agile prioritizes working software, customer collaboration, and responding to change over rigid planning.

Given the dynamic nature of our car rental platform—with multiple user roles, evolving features, and the need for frequent stakeholder feedback—an agile approach was clearly more suitable.

1.8.2 Methodology Selection : Why Scrum ?

The choice of the Scrum methodology for our project is based on several fundamental considerations. Before detailing the advantages of Scrum, it is relevant to compare this methodology with other recognized agile approaches, such as **Kanban** and **Extreme Programming (XP)**.

1.8.2.1 Comparative Analysis of Agile Methods

Criteria	Scrum
Structure	Fixed sprints (2-4 weeks), defined roles (PO, SM, Daily Scrum)
Flexibility	Priorities revised at sprint end
Collaboration	Strong via ceremonies and retrospectives
Tracking	Burndown charts, Daily Scrum
Ideal For	Frequent feature deliveries
Adaptability	At sprint boundaries

Tableau 1.3 : Comparative Analysis of Agile Methods : Scrum, Kanban, and XP

1.8.2.2 Justification for Choosing Scrum

After careful analysis, **Scrum** was selected as the project management methodology for the following reasons :

1. **Clear Structure with Flexibility** : Scrum provides a well-defined framework with sprints, roles, and ceremonies, while still allowing adaptation at each iteration. This balance was essential for managing a complex multi-module platform.
2. **Regular Deliveries** : Our project required frequent delivery of functional increments to stakeholders. Scrum's sprint-based approach ensured that new features (authentication, car management, booking system, chat, chatbot) were delivered and validated regularly.
3. **Stakeholder Collaboration** : The platform involves multiple actors (Admin, Agency, Customer, Visitor, Follower). Scrum ceremonies—especially Sprint Reviews and Retrospectives—facilitat continuous feedback and alignment with user needs.
4. **Risk Management** : By breaking the project into 8 sprints, we could identify and address risks early. Each sprint delivered tested, working functionality, reducing the risk of major failures at the end.
5. **Team Coordination** : Daily Stand-ups ensured that blockers were identified and resolved quickly, maintaining project momentum.
6. **Transparency** : Scrum artifacts (Product Backlog, Sprint Backlog, Burndown Charts) provided clear visibility into project progress for all stakeholders.

Why not Kanban ? While Kanban offers excellent flexibility, it lacks the structured iterations needed for our project, which required planned releases and defined milestones.

Why not XP ? Extreme Programming emphasizes technical practices like pair programming and test-driven development, which are valuable but require a highly disciplined team. Our project needed a broader project management framework, not just development practices.

1.8.3 Scrum Framework Overview

Scrum is an agile framework that enables teams to work together to deliver high-value products iteratively and incrementally. It provides a structured yet flexible approach to project management, emphasizing collaboration, accountability, and continuous improvement. Figure 2.5 illustrates the complete Scrum process flow.

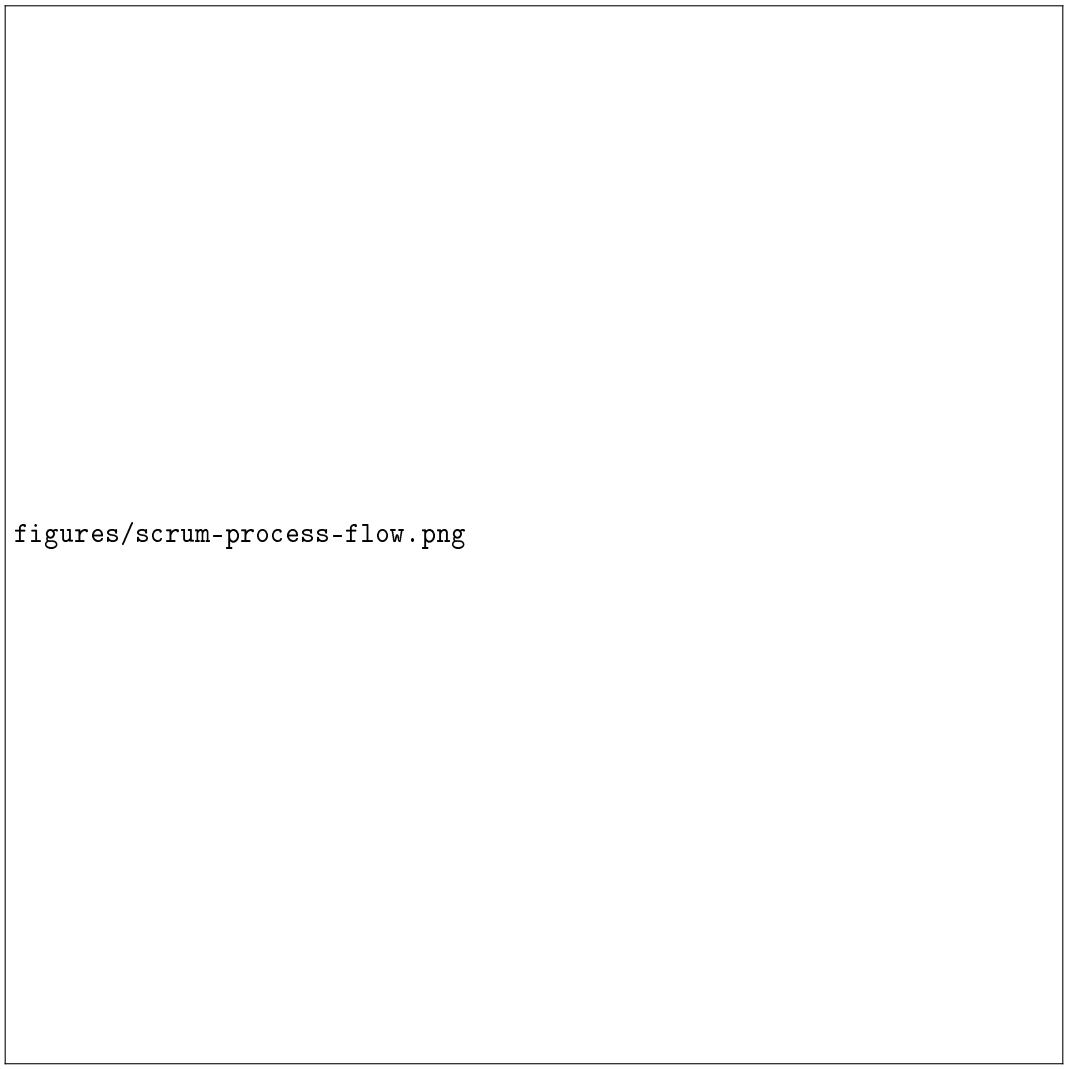

Figure 1.2 : Scrum Methodology Framework

The Scrum framework consists of three main components : **Roles**, **Events** (Ceremonies), and **Artifacts**. Each component plays a crucial role in ensuring the successful delivery of the project.

1.8.3.1 Scrum Team Roles

The Scrum Team is composed of three distinct roles, each with specific responsibilities that contribute to the project's success. The team is self-organizing and cross-functional, meaning members have all the skills necessary to deliver the product increment.

The Product Owner (PO) The Product Owner is the guardian of the project and the representative of the client and their needs. The PO serves as the single point of contact between the development team and stakeholders, ensuring that the team always works on the most valuable features.

Key Responsibilities :

— **Vision Management** : Defines and communicates the product vision to ensure all team members understand the project goals.

— **Backlog Management** : Creates, maintains, and prioritizes the Product Backlog, ensuring items are clearly defined with acceptance criteria.

— **Stakeholder Communication** : Acts as the liaison between stakeholders and the development team, gathering requirements and feedback.

— **Value Maximization** : Makes decisions that maximize the value delivered by the team in each sprint.

— **Acceptance** : Accepts or rejects completed work based on the Definition of Done.

In our MyLoc project, the Product Owner was responsible for :

— Gathering requirements from different user perspectives (Admin, Agency, Customer).

— Prioritizing features like authentication, car management, booking, and chatbot.

— Validating completed increments during Sprint Reviews.

The Scrum Master (SM) The Scrum Master is a facilitator and servant-leader who ensures the team follows Scrum practices correctly. The SM is focused on the framework itself and removes any obstacles that prevent the team from achieving their sprint goals.

Key Responsibilities :

— **Process Facilitation** : Ensures all Scrum events take place and are productive, time-boxed, and positive.

— **Impediment Removal** : Identifies and removes obstacles that block the team's progress.

— **Team Protection** : Shields the team from external interruptions and distractions during the sprint.

— **Coaching** : Helps team members understand Scrum theory, practices, and rules.

— **Continuous Improvement** : Facilitates retrospectives and drives process improvements.

— **Collaboration** : Works with the Product Owner to ensure the backlog is well-maintained and understood.

In our project, the Scrum Master :

— Facilitated Daily Stand-ups and ensured they stayed within 15 minutes.

— Resolved blockers related to environment setup, API integration, and deployment.

— Ensured smooth communication between frontend and backend developers.

The Development Team The Development Team is responsible for delivering potentially shippable product increments at the end of each sprint. The team is multidisciplinary, self-organized, and collectively accountable for the work delivered.

Key Characteristics :

— **Cross-Functional** : The team possesses all skills needed to create the product increment (frontend, backend, database, DevOps, testing).

— **Self-Organizing** : Team members decide how to accomplish their work without being directed by others.

— **Collaborative** : Members work together, share knowledge, and help each other overcome challenges.

— **Accountable** : The entire team is responsible for delivering the sprint commitment, not individual members.

— **Optimal Size** : Typically 3-9 members to ensure effective communication and productivity.

Responsibilities :

— Transform backlog items into working software increments.

— Estimate the effort required for backlog items.

— Participate actively in all Scrum ceremonies.

— Maintain code quality through testing and code reviews.

— Deploy and document completed features.

In our MyLoc project, the Development Team handled :

— Frontend development (Angular) : UI components, routing, guards, services.

— Backend development (Spring Boot) : REST APIs, business logic, database operations.

— AI Service (Flask) : Chatbot integration with ChatGPT API.

— DevOps : Docker containerization, Kubernetes deployment, GitLab CI/CD pipelines.

1.8.3.2 Scrum Events (Ceremonies)

Scrum defines five key events (ceremonies) that create regularity and minimize the need for meetings not defined in Scrum. Each event is time-boxed, meaning it has a maximum duration that ensures efficient use of time.

Sprint The Sprint is the heart of Scrum—a time-boxed iteration during which a "Done", usable, and potentially releasable product increment is created. Sprints have consistent durations throughout

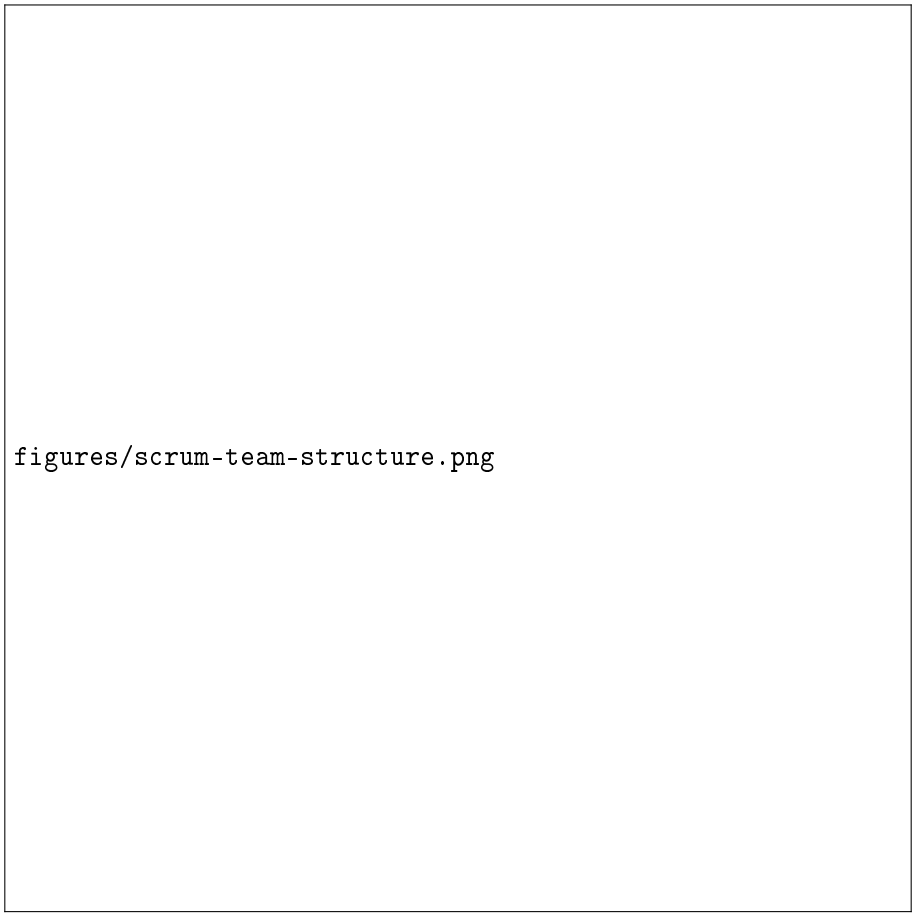

Figure 1.3 : Scrum Team Structure and Interactions

development and a new sprint starts immediately after the previous one concludes.

Key Characteristics :

— **Duration** : Fixed length of 1-4 weeks (our project used 10-15 day sprints).

— **Consistency** : Sprint length remains constant throughout the project for predictability.

— **Protection** : No changes are made during the sprint that would endanger the Sprint Goal.

— **Scope Flexibility** : Scope may be clarified and re-negotiated between the PO and Development Team.

— **Cancellation** : Only the Product Owner can cancel a sprint if the Sprint Goal becomes obsolete.

During a sprint, the team :

— Develops the selected features from the Sprint Backlog.

— Attends Daily Scrum meetings to synchronize work.

— Refines backlog items for future sprints.

— Prepares for the Sprint Review and Retrospective.

Our project was divided into **8 sprints**, each focusing on specific modules :

— Sprint 0 : Project setup, environment configuration, Keycloak integration.

— Sprints 1-2 : User authentication, role management, admin dashboard.

— Sprints 3-4 : Agency management, car CRUD operations, image upload.

— Sprints 5-6 : Customer features, booking system, availability checking.

— Sprint 7 : Chat system, chatbot integration, notifications.

— Sprint 8 : Payment integration, PDF contracts, QR codes, final testing.

Sprint Planning Meeting Sprint Planning is a collaborative meeting that initiates the sprint. The entire Scrum Team participates to define what can be delivered in the upcoming sprint and how the work will be achieved. This meeting should not exceed 4 hours for a 2-week sprint.

Meeting Structure :

1. **Part 1 - What ?** : The Product Owner presents the highest priority items from the Product Backlog. The team discusses and selects items they can commit to completing.
2. **Part 2 - How ?** : The Development Team plans the work needed to deliver the selected items, breaking them into tasks.

Inputs :

— Prioritized Product Backlog

— Team velocity (historical data on work completed per sprint)

— Team capacity (availability considering holidays, other commitments)

— Definition of Done

Outputs :

— Sprint Goal : A clear objective for the sprint

— Sprint Backlog : Selected items + plan for delivering them

— Team commitment to deliver the increment

Daily Scrum (Stand-up) The Daily Scrum is a 15-minute time-boxed event held every day at the same time and place. It is designed to synchronize activities, identify impediments, and plan the next 24 hours of work. Only the Development Team members participate actively, though others may attend as observers.

The Three Questions : Each team member answers three questions :

1. **What did I do yesterday ?** - Progress report on completed tasks.
2. **What will I do today ?** - Planned tasks for the current day.
3. **Are there any impediments ?** - Blockers preventing progress.

Key Rules :

— Same time, same place every day (consistency).

— Standing up to keep it short (hence "stand-up").

— No problem-solving during the meeting—issues are taken offline.

— Focus on progress toward the Sprint Goal, not status reporting.

Benefits :

— Improves communication and team cohesion.

— Quickly surfaces blockers for resolution.

— Eliminates the need for other meetings.

— Promotes quick decision-making and accountability.

In our project, Daily Scrums helped identify :

— API endpoint mismatches between frontend and backend.

— Docker configuration issues.

— Keycloak token validation problems.

— Database schema updates needed for new features.

Sprint Review The Sprint Review is held at the end of the sprint to inspect the increment and adapt the Product Backlog if needed. The team presents what was accomplished during the sprint, and stakeholders provide feedback. This meeting should not exceed 2 hours for a 2-week sprint.

Participants :

— Scrum Team (PO, SM, Development Team)

— Key stakeholders (invited by the Product Owner)

Meeting Activities :

1. The Product Owner explains what backlog items have been "Done" and what has not been "Done".
2. The Development Team demonstrates the completed work and answers questions.
3. The Product Owner discusses the current state of the Product Backlog and projects likely completion dates.
4. The entire group collaborates on what to do next, providing input for the next Sprint Planning.
5. Review of timeline, budget, and marketplace for the next anticipated product release.

Outputs :

— Updated Product Backlog based on feedback.

— Acceptance or rejection of completed items.

— Input for the next sprint's planning.

Sprint Retrospective The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be implemented during the next sprint. It occurs after the Sprint Review and before the next Sprint Planning. The meeting should not exceed 1.5 hours for a 2-week sprint.

Focus Areas :

— **People** : How well did the team collaborate ?

— **Relationships** : How was the communication within the team and with stakeholders ?

— **Processes** : Were the Scrum practices followed effectively ?

— **Tools** : Did the tools support or hinder the work ?

Common Format (Start-Stop-Continue) :

— **Start** : What should we start doing ?

— **Stop** : What should we stop doing ?

— **Continue** : What should we continue doing ?

Alternative Format (Liked-Learned-Lacked-Longed For) :

— **Liked** : What did we enjoy this sprint ?

— **Learned** : What new knowledge did we gain ?

— **Lacked** : What was missing or insufficient ?

— **Longed For** : What do we wish we had ?

Outputs :

— Identified improvements for the next sprint.

— Action items with owners and deadlines.

— Updated team working agreements if needed.

In our project, retrospectives led to improvements such as :

— Better Git branching strategy (feature branches).

— More thorough API documentation.

— Earlier integration testing between frontend and backend.

— Improved Docker compose configurations for faster local development.

Backlog Refinement (Grooming) Backlog Refinement is an ongoing activity where the Product Owner and Development Team collaborate to add detail, estimates, and order to Product Backlog items. This is not an official Scrum event but is essential for maintaining a healthy backlog.

Activities :

— Adding details and acceptance criteria to user stories.

— Estimating effort using story points or hours.

— Splitting large items into smaller, more manageable ones.

— Re-prioritizing items based on new information.

— Removing obsolete items.

Best Practice : The Development Team should spend no more than 10% of their capacity on refinement activities.

1.8.3.3 Scrum Artifacts

Scrum artifacts represent work or value and are designed to maximize transparency and provide opportunities for inspection and adaptation. The three primary artifacts are the Product Backlog, Sprint Backlog, and Increment.

Product Backlog The Product Backlog is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

Characteristics :

— **Dynamic** : Constantly evolves as new requirements emerge.

— **Ordered** : Items are prioritized based on value, risk, dependencies, and need.

— **Detailed Appropriately** : Higher priority items are more detailed ; lower priority items are less refined.

— **Estimated** : Items have size estimates provided by the Development Team.

— **Living Document** : Never complete—it grows and changes with the product.

Backlog Item Structure (User Story Format) :

As a [type of user], I want [goal/desire] so that [benefit/value].

Prioritization Methods :

— **MoSCoW** : Must have, Should have, Could have, Won't have.

— **Value vs. Effort** : Prioritize high-value, low-effort items first.

— **WSJF (Weighted Shortest Job First)** : Cost of delay divided by duration.

In our project, the Product Backlog contained 40 user stories organized by epic :

— Authentication & Authorization (US-01 to US-05)

— Admin Management (US-06 to US-12)

— Agency Management (US-13 to US-19)

— Customer Features (US-20 to US-30)

— Communication & Notifications (US-31 to US-35)

— Payment & Contracts (US-36 to US-40)

Sprint Backlog The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. It is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver it.

Components :

— Selected Product Backlog items (user stories).

— Tasks breakdown for each item.

— Estimated effort for each task.

— Task assignments (self-selected by team members).

— Sprint Goal.

Characteristics :

— Owned exclusively by the Development Team.

— Updated daily as work progresses.

— Only the Development Team can add or modify items during the sprint.

— Highly visible to all stakeholders.

Increment The Increment is the sum of all the Product Backlog items completed during a sprint and the value of the increments of all previous sprints. At the end of a sprint, the new Increment must be "Done," meaning it must be in usable condition and meet the Scrum Team's Definition of Done.

Definition of Done (DoD) : Our project's Definition of Done included :

— Code is written and follows coding standards.

— Code is reviewed by at least one other team member.

— Unit tests are written and passing.

— Integration tests are passing.

— Feature is documented (API endpoints, user guide).

— Feature is deployed to staging environment.

— Feature is demonstrated and accepted by Product Owner.

Burndown Chart The Burndown Chart is a visual representation of work left to do versus time. It shows the progress of the team toward completing the Sprint Backlog and helps predict whether the team will complete all planned work by the end of the sprint.

Reading the Burndown Chart :

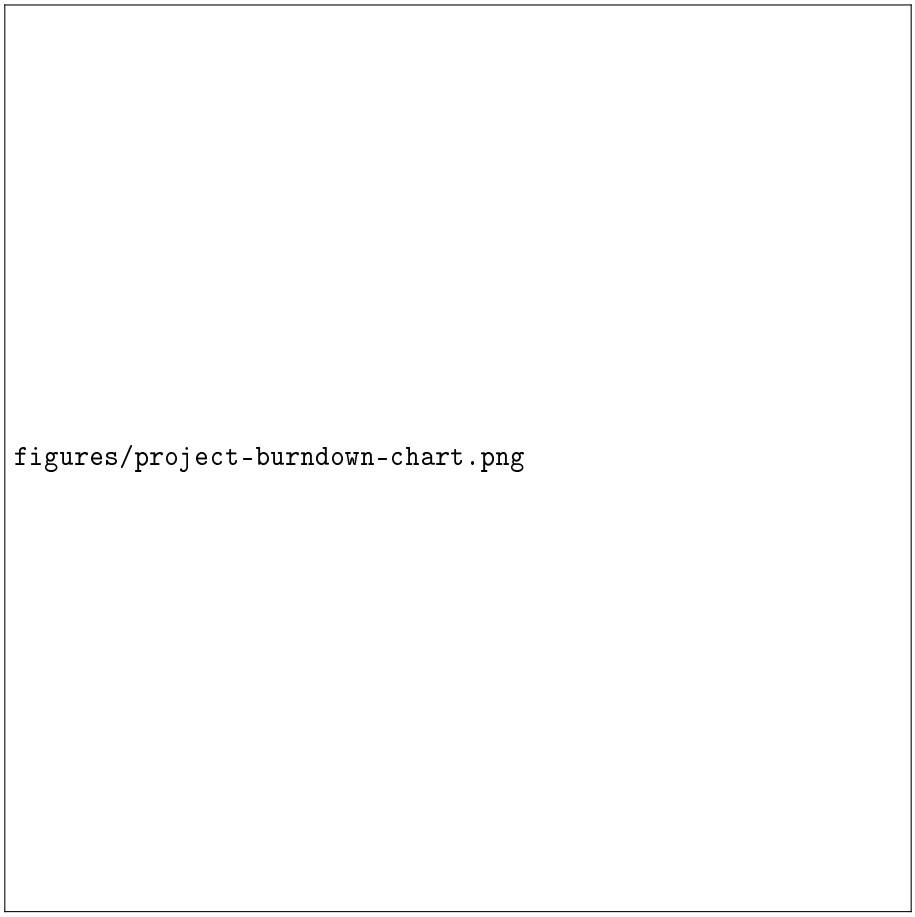

Figure 1.4 : Example Sprint Burndown Chart

— **Ideal Line** : The theoretical progression if work is completed evenly throughout the sprint.

— **Actual Line** : The real progress based on completed work.

— **Above Ideal** : Team is behind schedule.

— **Below Ideal** : Team is ahead of schedule.

— **Plateau** : No progress—possible blockers or impediments.

Velocity Chart Velocity is the amount of work a team completes during a sprint, measured in story points or hours. Tracking velocity helps the team forecast how much work they can complete in future sprints.

Uses of Velocity :

— Capacity planning for Sprint Planning.

— Release forecasting (when will feature X be ready ?).

— Identifying trends (improving or declining productivity).

— Detecting process problems (sudden velocity drops).

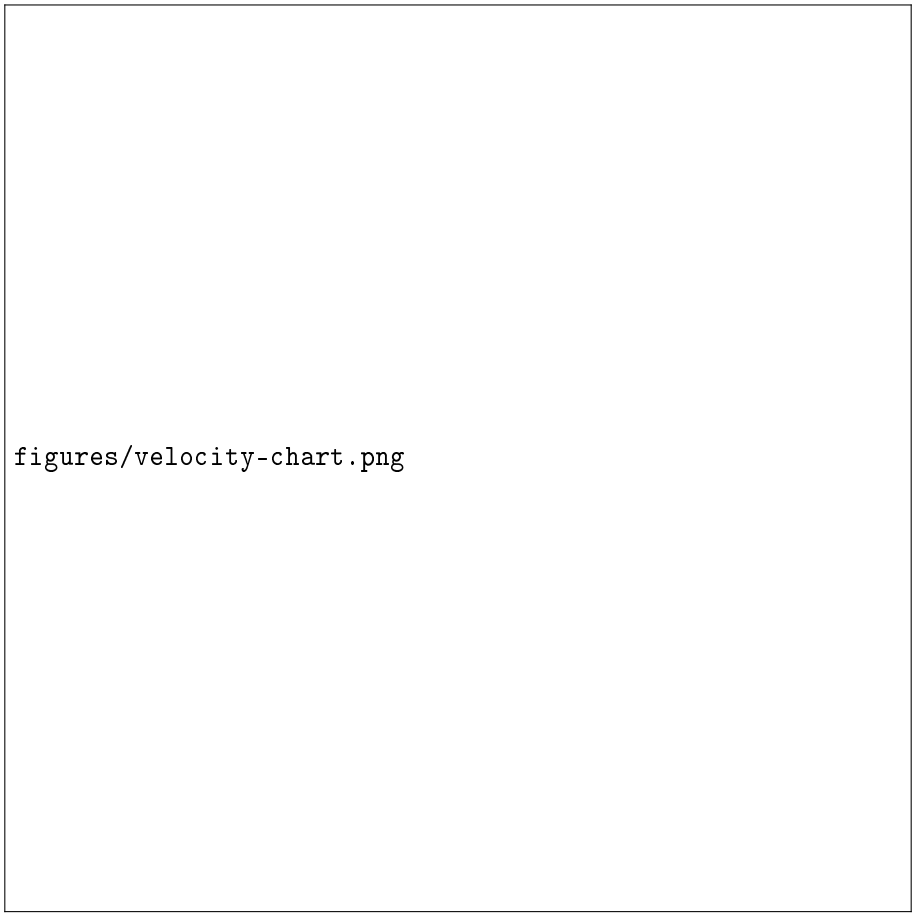

Figure 1.5 : Team Velocity Chart Across Sprints

1.8.4 Agile Methodology Implementation

Development was organized in 8 sprints of 10-15 days each, encouraging feedback, team collaboration, and continuous improvement. The project followed these Scrum practices :

— **Sprint Planning** : Each sprint began with planning sessions to select user stories and define acceptance criteria.

— **Daily Stand-ups** : Brief daily meetings to synchronize team activities and identify blockers.

— **Sprint Reviews** : Demonstrations of completed features to validate functionality with stakeholders.

— **Retrospectives** : Team reflections to improve processes and address challenges.

— **Continuous Integration** : Automated builds and tests via GitLab CI/CD to ensure code quality.

Component	Specification
Processor	Intel Core i7
RAM	16 GB DDR4
Storage	512 GB SSD
Operating System	Windows 10 + WSL2
Display	15.6-inch Full HD

Tableau 1.4 : Development Machine Specifications

1.8.5 Development Environment

1.8.5.1 Hardware Environment

1.8.5.2 Software Environment

— Programming Languages : Java, Python, TypeScript, HTML/CSS/JS.

— Frameworks : Spring Boot, FastAPI, Angular, Keycloak.

— Containerization : Docker, Docker Compose.

— Orchestration : Kubernetes (K8s).

— CI/CD : GitLab CI/CD.

— Database : MySQL.

— Testing : Postman, XAMPP.

— IDE : Visual Studio Code.

— Version Control : Git, GitLab.

— Project Management : Scrum, Trello/GitLab Issues.

1.8.5.3 Deployment Tools

— Docker Compose for local service orchestration.

— Kubernetes (K8s) for scalable microservice deployment.

1.9 Conclusion

This chapter established the foundation of the MyLoc project by thoroughly examining the problematic situation faced by car rental agencies before digitalization, defining clear objectives for the platform, and demonstrating how the proposed solution addresses each identified challenge. The analysis of existing systems revealed significant gaps in multi-role management, real-time communication, and automation—all of which our platform solves through modern technologies and best practices.

The Scrum methodology was selected after careful comparison with other agile approaches, providing the structured yet flexible framework needed for delivering a complex, multi-stakeholder platform. The development environment, combining Angular, Spring Boot, FastAPI, Keycloak, Docker,

Kubernetes, and GitLab CI/CD, ensures a scalable, secure, and maintainable solution.

The next chapter will detail the requirements analysis, including a comprehensive study of actors, functional and non-functional requirements, and use case specifications for each user role.

Chapitre 2

Needs Analysis and Specification

Plan

2.1 Introduction

This chapter formally defines and analyzes the functional and non-functional needs of the car rental platform. By identifying limitations in existing systems, this section translates user expectations into specific system requirements, focusing on secure, real-time, and scalable services for car rentals, including role-based interactions, authentication, booking workflows, communication, and deployment. Following the context and problem analysis outlined in Chapter 1, this chapter ensures that the platform meets real user needs efficiently and securely.

The requirements analysis follows a systematic approach :

1. **Actor Identification** : Defining who interacts with the system and their responsibilities.
2. **Functional Requirements** : Specifying what the system must do for each actor.
3. **Non-Functional Requirements** : Defining quality attributes like performance, security, and scalability.
4. **Use Case Modeling** : Visualizing interactions between actors and the system.
5. **Class Modeling** : Defining the data structures and relationships.

2.2 Actors Identification

An actor represents an external entity that interacts with the system. Actors can be humans (users with different roles) or systems (automated services). Understanding actors is crucial for defining system boundaries and requirements.

The MyLoc platform supports the following primary actors :

2.2.1 Human Actors

2.2.1.1 Administrator

The Administrator is the highest-privilege user responsible for the overall management and oversight of the platform.

Role Description :

— Manages the entire platform ecosystem including agencies, customers, and content.

— Ensures platform security and enforces access control policies.

— Monitors system health and performance metrics.

— Coordinates with agencies through the built-in chat system.

— Manages blog content to engage users and promote services.

Key Permissions :

— Full CRUD operations on agencies, customers, and blog posts.

— Access to all dashboards and administrative panels.

— System configuration and settings management.

— Direct communication with all agencies.

2.2.1.2 Agency

Agencies are the car rental businesses that list their vehicles on the platform and manage customer bookings.

Role Description :

— Represents car rental companies that offer vehicles for rent.

— Manages their own fleet of cars with full control over listings.

— Processes customer rental requests (approve/reject).

— Communicates with administrators for support and coordination.

— Tracks booking history and manages availability calendars.

Key Permissions :

— Full CRUD operations on their own cars only.

— View and process rental requests for their vehicles.

— Chat with administrators.

— Manage agency profile and settings.

2.2.1.3 Customer

Customers are registered users who browse cars and make rental requests.

Role Description :

— Primary consumers of the car rental service.

— Can browse the full catalog and make informed decisions.

— Submit rental requests with date ranges and preferences.

— Track their booking history and request status.

— Engage with the platform through comments, reviews, and the chatbot.

Key Permissions :

— Browse all public content (cars, blogs).

— Submit and manage their rental requests.

— View their booking history and notifications.

— Post comments and reviews.

— Contact administrators via email.

— Use the AI chatbot for assistance.

2.2.1.4 Visitor

Visitors are unauthenticated users exploring the platform before registration.

Role Description :

— Potential customers browsing the platform without an account.

— Can explore available cars and read blog content.

— Can interact with the chatbot for information.

— May subscribe as a Follower to receive updates.

— Can initiate registration to become a Customer.

Key Permissions :

— View public car listings (limited details).

— Read blog posts.

— Use the chatbot.

— Subscribe as a Follower.

— Register for a full account.

2.2.1.5 Follower

Followers are visitors who subscribe to receive email updates without creating a full account.

Role Description :

— Interested users who want to stay informed about new offerings.

— Subscribe by entering their email in the website footer.

— Receive automatic notifications when new cars are added.

— Represents a marketing channel for the platform.

— Low-commitment engagement method to attract potential customers.

Key Permissions :

— Subscribe to new car alerts.

— Receive email notifications.

— Unsubscribe at any time.

2.2.2 System Actors

2.2.2.1 Chatbot (AI Assistant)

The Chatbot is an automated AI-powered assistant that provides instant support to users.

System Description :

— Built with Flask/FastAPI backend connected to ChatGPT API.

— Provides 24/7 automated assistance to visitors and customers.

— Answers common questions about cars, bookings, and platform usage.

— Supports multiple languages for international users.

— Reduces workload on human support staff.

Capabilities :

— Natural language processing for user queries.

— Context-aware responses based on conversation history.

— Integration with platform data for accurate information.

— Fallback to human support for complex issues.

2.2.3 Actor Hierarchy Diagram

Figure **??** shows the relationships between different actors and their inheritance of permissions.

Figure 2.1 : Actor Hierarchy and Permission Inheritance

2.3 Needs Identification

2.3.1 Functional Needs

Functional requirements define what the system must do—the features and capabilities it provides to each actor. These requirements are derived from user stories and stakeholder interviews.

2.3.1.1 Customer Functional Requirements

Customers are the primary end-users of the platform. Their requirements focus on ease of use, transparency, and efficient booking processes.

- **Browsing and Discovery :**
  - Browse available cars and blogs without logging in.
  - Search and filter cars based on availability, type, price, and specifications.
  - View detailed car information including photos, features, and pricing.
  - Read reviews and ratings from other customers.
- **Account Management :**
  - Register with email verification for security.
  - Log in securely using Keycloak authentication.
  - Update profile information and preferences.
  - Reset password through secure email link.
- **Booking and Rental :**
  - Check real-time car availability for selected dates.
  - Submit rental requests with date range selection.
  - Receive immediate validation of date availability.
  - Track rental request status (pending, approved, rejected).
  - Cancel pending requests before processing.
  - View complete booking history.
- **Communication and Support :**
  - Receive email notifications on rental request status changes.
  - Receive PDF contract with QR code upon approval.
  - Contact admin directly via email within the app.
  - Interact with the AI chatbot for instant support.
  - Receive in-app notifications for important updates.
- **Engagement :**
  - Comment and provide reviews on cars and blog posts.
  - Receive updates via email when new cars are added.
  - Share content on social media platforms.

2.3.1.2 Agency Functional Requirements

Agencies require tools to efficiently manage their fleet and customer interactions.

- **Account and Profile :**
  - Register as an agency with business details.
  - Log in securely with role-based authentication.
  - Manage agency profile, logo, and contact information.
- **Fleet Management :**
  - Add new cars with complete details (name, model, type, price, features).
  - Upload multiple photos for each car.
  - Edit car information and availability.
  - Delete cars from the platform.
  - Manage car availability calendars.
- **Booking Management :**
  - View all rental requests for their cars.
  - Accept or decline requests with one click.
  - Automatic email sent to customers upon decision.
  - Track booking history and statistics.
  - **Communication :**
    - Chat with administrators in real-time.
    - Receive notifications of new rental requests.
    - View message history for reference.

2.3.1.3 Administrator Functional Requirements

Administrators need comprehensive control over all platform aspects.

- **Agency Management :**
  - Add new agencies to the platform.
  - Edit agency information and status.
  - Delete or suspend agency accounts.
  - View agency performance metrics.
- **Customer Management :**
  - View all registered customers.
  - Edit customer information if needed.
  - Delete or suspend customer accounts.
  - Handle customer complaints and issues.
- **Content Management :**
  - Create blog posts with rich content.

- Edit and update existing blogs.
- Delete inappropriate content.
- Moderate comments and reviews.

— **Communication :**

- Chat with agencies for coordination.
- Send platform-wide announcements.
- Respond to customer contact emails.

— **System Administration :**

- Monitor system health and performance.
- Manage security settings via Keycloak.
- Configure platform settings and preferences.
- Access logs and audit trails.

2.3.1.4 Follower Functional Requirements

— Subscribe to updates by entering email in the website footer.

— Receive automatic email notifications when new cars are added.

— Stay informed about new offerings without a full account.

— Unsubscribe from notifications at any time via email link.

2.3.1.5 System-wide Functionalities

— Secure authentication and role-based authorization via Keycloak.

— Instant messaging system enabling admin-agency communication.

— Email notification system for updates, rental request status, and confirmations.

— PDF generation with QR codes for confirmed rental contracts.

— Integration of chatbot powered by Flask API for user assistance.

— Deployment automation using GitLab CI/CD pipelines, Docker Compose, and Kubernetes.

— Real-time availability checking for cars during rental request submission.

2.3.2 Non-Functional Needs

— **Performance :**

- The system must respond to user actions (e.g., browsing cars, submitting rental requests) within 2 seconds under normal load.
- Rental availability checks must provide real-time feedback during request submission.
- Email notifications and PDF generation should be triggered within 2 minutes after

any change in request status.

- **Security :**
  - Authentication and authorization must be enforced using Keycloak, with role-based access control (Admin, Agency, Customer).
  - All sensitive data (e.g., passwords, personal information) must be securely stored and transmitted using encryption protocols (HTTPS, password hashing).
  - All API endpoints must be protected from unauthorized access using AuthGuard and secure access control mechanisms.
  - The application must defend against common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- **Usability :**
  - The user interface must be intuitive, user-friendly, and responsive across desktop, tablet, and mobile devices.
  - Customers must be able to easily browse, search for cars, submit rental requests, and track their status.
  - Agencies and administrators should have dedicated dashboards for managing cars, requests, blogs, and communication.
- **Reliability and Availability :**
  - The system should ensure 99.9% uptime, with robust error handling and recovery mechanisms.
  - All rental and user data must be reliably saved and backed up regularly.
- **Maintainability and Extensibility :**
  - The codebase must follow clean architecture principles and separation of concerns to facilitate future updates.
  - Continuous integration and deployment (CI/CD) pipelines using GitLab must support automated testing and deployment.
  - Comprehensive documentation should be maintained for the API, deployment process, and system usage.
- **Scalability :**
  - The system must support increased user traffic and rental activity without performance degradation.
  - Kubernetes-based deployment must allow for horizontal scaling of services as demand grows.
- **Interoperability :**

— The system must integrate seamlessly with external services, including the Flask-based chatbot API and payment gateways.

— Email services must be compatible with SMTP or third-party providers to ensure reliable communication.

2.4 System Design

2.4.1 Global Use Case Diagram

The global use case diagram represents the overall interactions between primary actors (Customer, Agency, Administrator, Visitor, Follower) and the car rental platform. It highlights the key functionalities accessible to each actor, offering a high-level overview of the system's capabilities.

2.4.1.1 Customer Use Case Diagram

The customer use case diagram illustrates the primary interactions between customers and the car rental platform. It shows how registered customers can manage their profiles, search and book vehicles, interact with the AI chatbot, and engage with the platform's content system.

2.4.1.2 Agency Use Case Diagram

The agency use case diagram demonstrates how car rental agencies interact with the platform to manage their fleet, handle customer bookings, and maintain their business operations through the comprehensive agency dashboard.

2.4.1.3 Administrator Use Case Diagram

The administrator use case diagram showcases the comprehensive system management capabilities available to platform administrators, including user management, content moderation, financial oversight, and system maintenance operations.

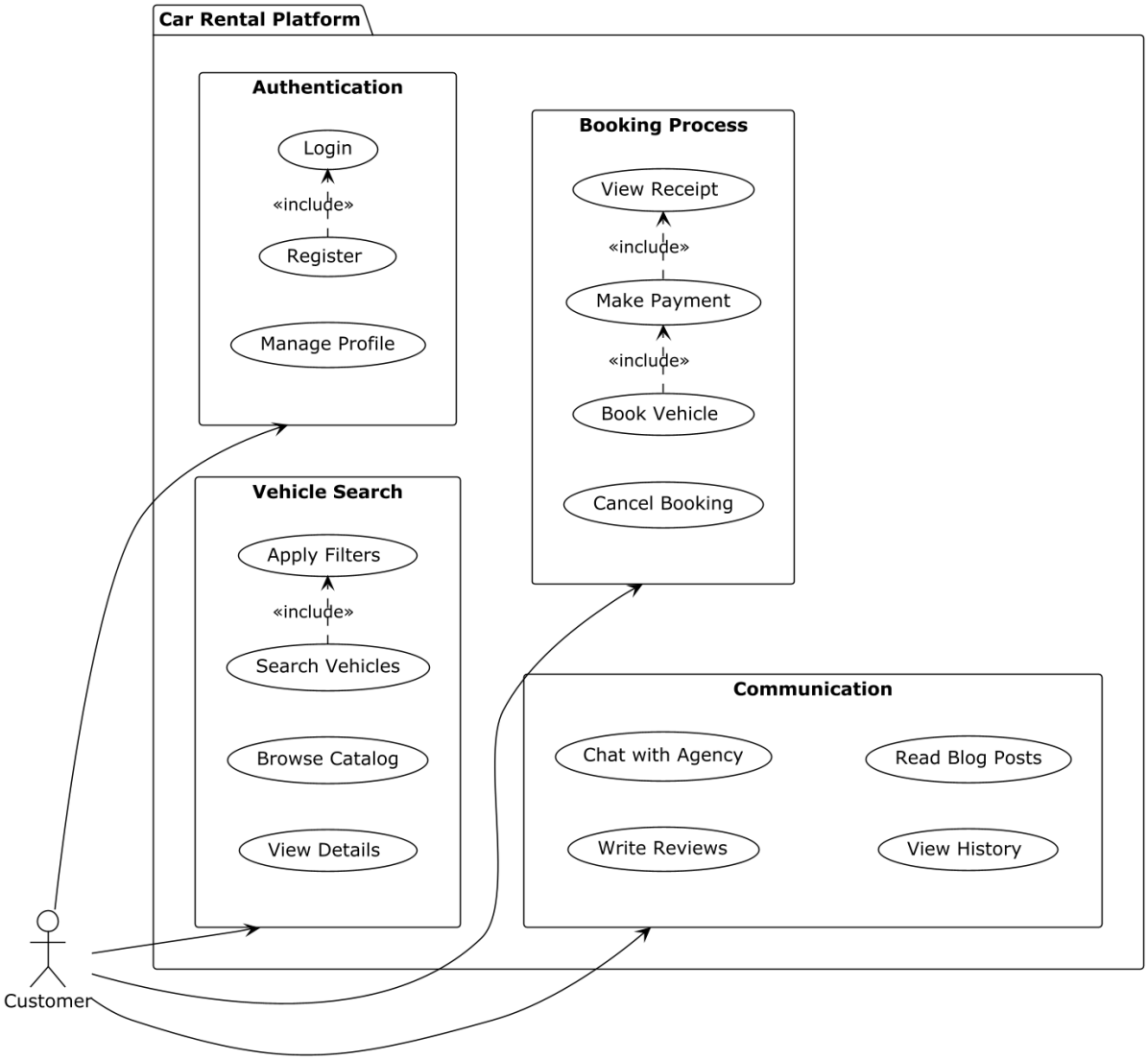

Figure 2.2 : Use Case Diagram for Customer - Myloc Agency Platform

2.4.2 Global Class Diagram

2.5 Project Management Methodology

2.5.1 Traditional vs Agile Methods

Before selecting a methodology for this project, it is essential to understand the fundamental differences between traditional (classical) and agile approaches to project management. The following table presents a comprehensive comparison :

2.5.2 Methodology Selection

The choice of the Scrum methodology for our project is based on several fundamental considerations. Before detailing the advantages of Scrum, it is relevant to compare this

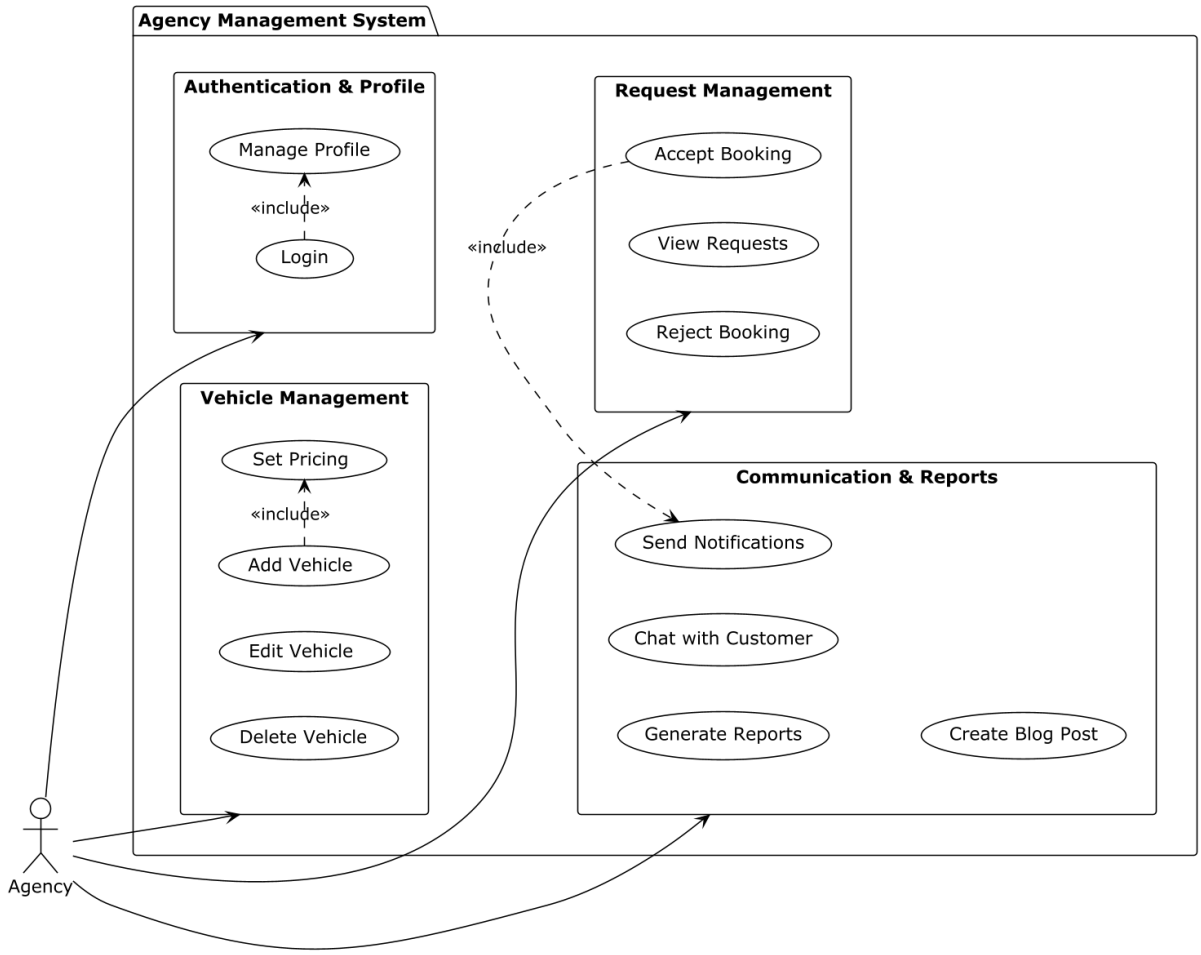

Figure 2.3 : Use Case Diagram for Agency - Fleet and Booking Management

Criteria	Traditional Methods
Process	Linear, sequential (V-Model, Waterfall)
Requirements	Detailed specs fixed at project start
Flexibility	Low, hard to integrate changes
Delivery	Single delivery at project end
Client Interaction	Limited after requirements phase
Risk Management	Upfront identification, few adjustments
Planning	Complete upfront, rigid
Ideal Project	Simple, well-defined, stable
Examples	V-Model, Waterfall, PERT

Tableau 2.1 : Comparison between Traditional and Agile Methods

methodology with other recognized agile approaches, such as **Kanban** and **Extreme Programming (XP)**.

The following comparative table presents the three agile methods and justifies the choice of Scrum :

Justification for Choosing Scrum :

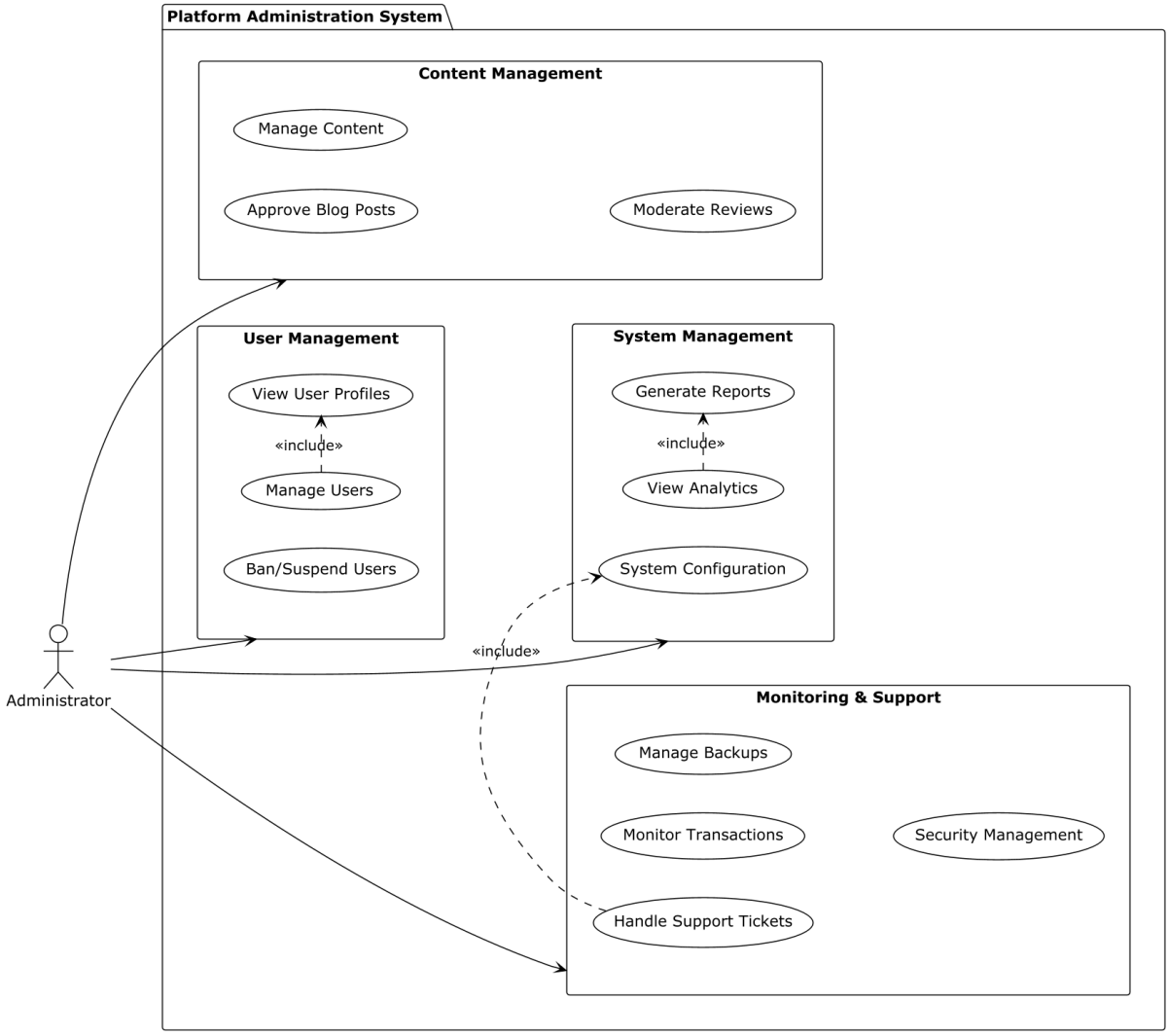

Figure 2.4 : Use Case Diagram for Administrator - System Management

	Criteria	**Scrum**
	Structure	Fixed sprints (10-15 days), defined roles & ceremonies
	Flexibility	Adaptation at sprint end
	Collaboration	Strong via ceremonies & retrospectives
	Tracking	Burndown charts, Daily Scrum
	Ideal For	Frequent deliveries
	Adaptability	High at sprint boundaries

Tableau 2.2 : Comparison between Agile Methods : Scrum, Kanban, and XP

— **Scrum** is chosen for its clear structure, which combines flexibility and rigor, while promoting close collaboration and continuous adaptation at each sprint.

— This methodology is particularly suited to our project, which requires regular deliveries and strong interaction with stakeholders.

— **Kanban** is flexible but lacks structure for projects requiring well-defined iterations.

— **XP** is very effective for code quality but demands rigorous discipline and is more oriented

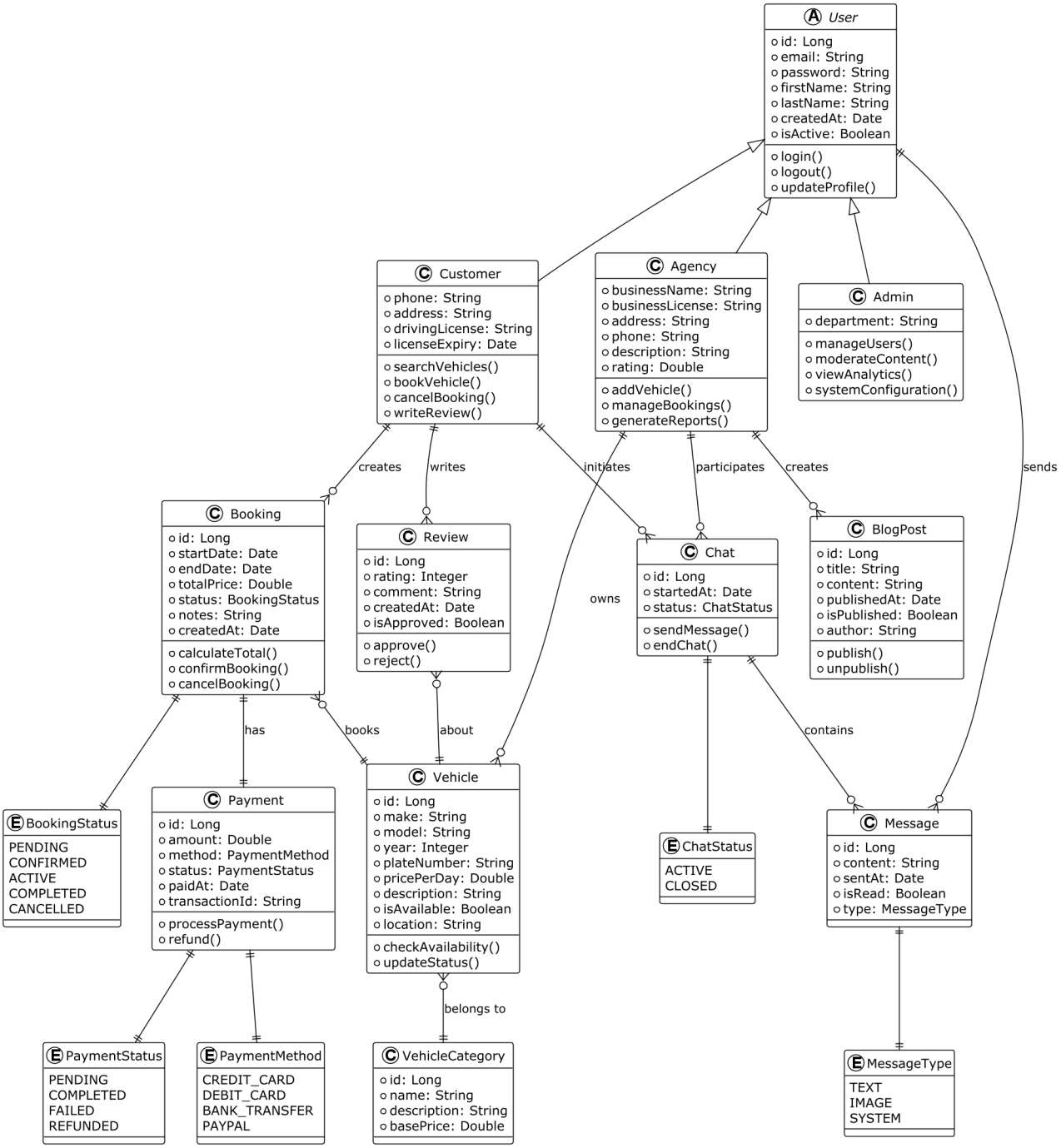

Figure 2.5 : Global Class Diagram Representing Car Rental Platform Architecture

towards pure software development.

2.5.3 Scrum Methodology Overview

Scrum is an agile project management method designed to improve team productivity, even remotely, while allowing continuous product optimization through regular user feedback. Inspired by rugby, where teams gather in a scrum, Scrum encourages a dynamic and participative approach to project management.

This ensures a fair balance between initial investment and the final delivered product,

offering flexibility to redirect the project along the way. Scrum is widely adopted by development teams because it promotes :

— Collaboration with the client

— Acceptance of change

— Interaction between team members

— Delivery of operational software

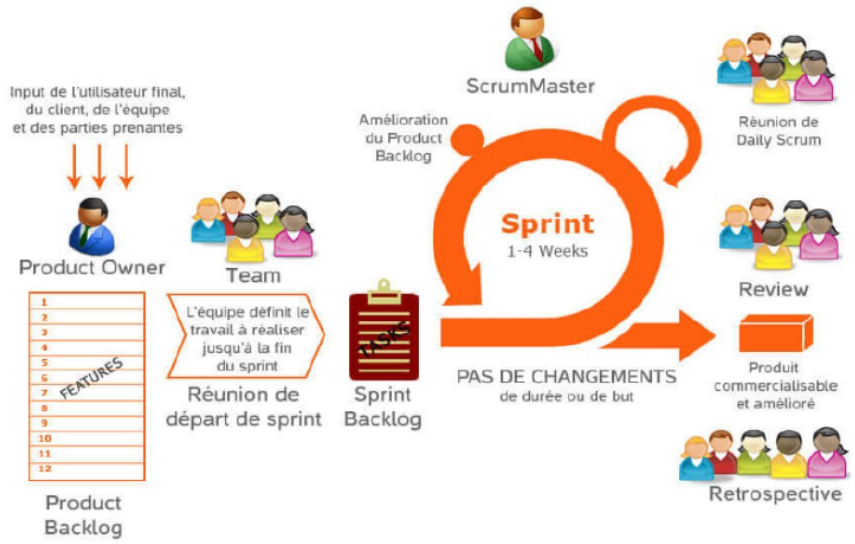

Figure 2.6 : Scrum Methodology Overview

2.5.3.1 Scrum Key Concepts

Sprint

Scrum breaks down the project into different iterations called sprints. Each sprint lasts no more than four weeks, during which the team develops the features specified in the product backlog.

Daily Scrum (Daily Stand-up)

This is a 15-minute daily meeting that tracks project progress. During this meeting, team members present :

— Tasks completed yesterday

— Tasks planned for today

— Identified obstacles preventing them from reaching their goal

Sprint Planning Meeting

A sprint planning meeting lasting no more than 4 hours, during which the team selects the

priority features to develop for the upcoming sprint.

Sprint Review

The sprint review is a meeting at the end of the sprint where the team presents the developed features to the client and collects corresponding feedback.

Sprint Retrospective

A meeting where the team reflects on the sprint to identify what went well, what could be improved, and action items for the next sprint.

2.5.3.2 Scrum Team Composition

The Scrum team consists of the Scrum Master, the Product Owner, and the Development Team. The team's objective is to have business value to display at the end of each sprint and to make the most profitable increments at each sprint.

Role
Product Owner (PO)
Scrum Master (SM)
Development Team

Tableau 2.3 : Scrum Team Roles and Responsibilities

The Scrum team for this project is presented in the following table :

Product Owner	Scrum Master	Development Team
Amine Ben Chaabane	Mme. Sawsen Jalel	Bayrem Boussaidi

Tableau 2.4 : Project Scrum Team

2.6 Scrum Implementation and Artifacts

This section presents the detailed Scrum artifacts used throughout the project, including the product backlog, sprint planning, burndown charts, and velocity tracking—similar to tools like Jira.

2.7 Introduction to Scrum Artifacts

Scrum artifacts provide key information that the Scrum Team and stakeholders need to understand the work being done, the value being created, and the progress being made. These artifacts are designed to maximize transparency and provide opportunities for inspection and adaptation.

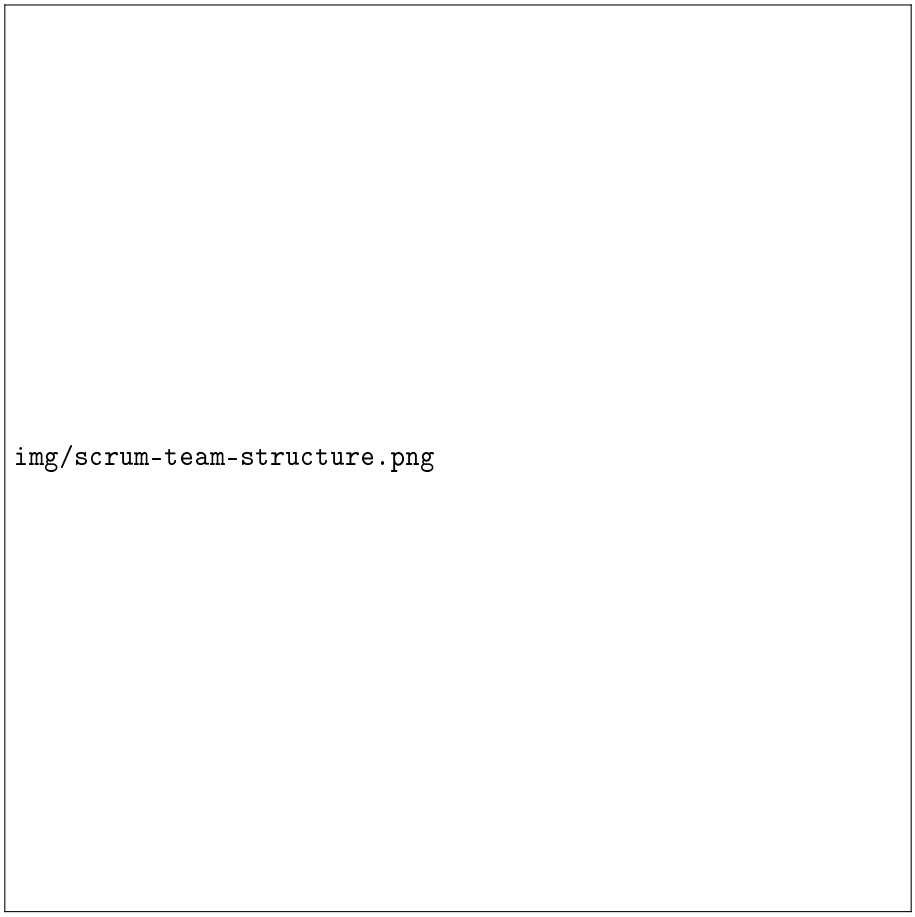

Figure 2.7 : Scrum Team Structure - Myloc Agency Project

In the context of the MyLoc car rental platform, Scrum artifacts played a crucial role in organizing development work, tracking progress, and ensuring all stakeholders had visibility into the project status.

2.7.1 Purpose of Scrum Artifacts

Product Backlog : Serves as the single source of truth for all work to be done on the product. It contains all features, requirements, enhancements, and bug fixes needed for the MyLoc platform.

Sprint Backlog : Provides a real-time picture of the work planned for the current sprint, helping the Development Team stay focused on the Sprint Goal.

Increment : Represents the cumulative value delivered at the end of each sprint—a potentially releasable version of the product.

Burndown Charts : Visual tools that helped our team track progress, identify potential delays early, and make data-driven decisions.

Velocity Charts : Helped forecast future capacity and establish realistic sprint commitments based on historical performance.

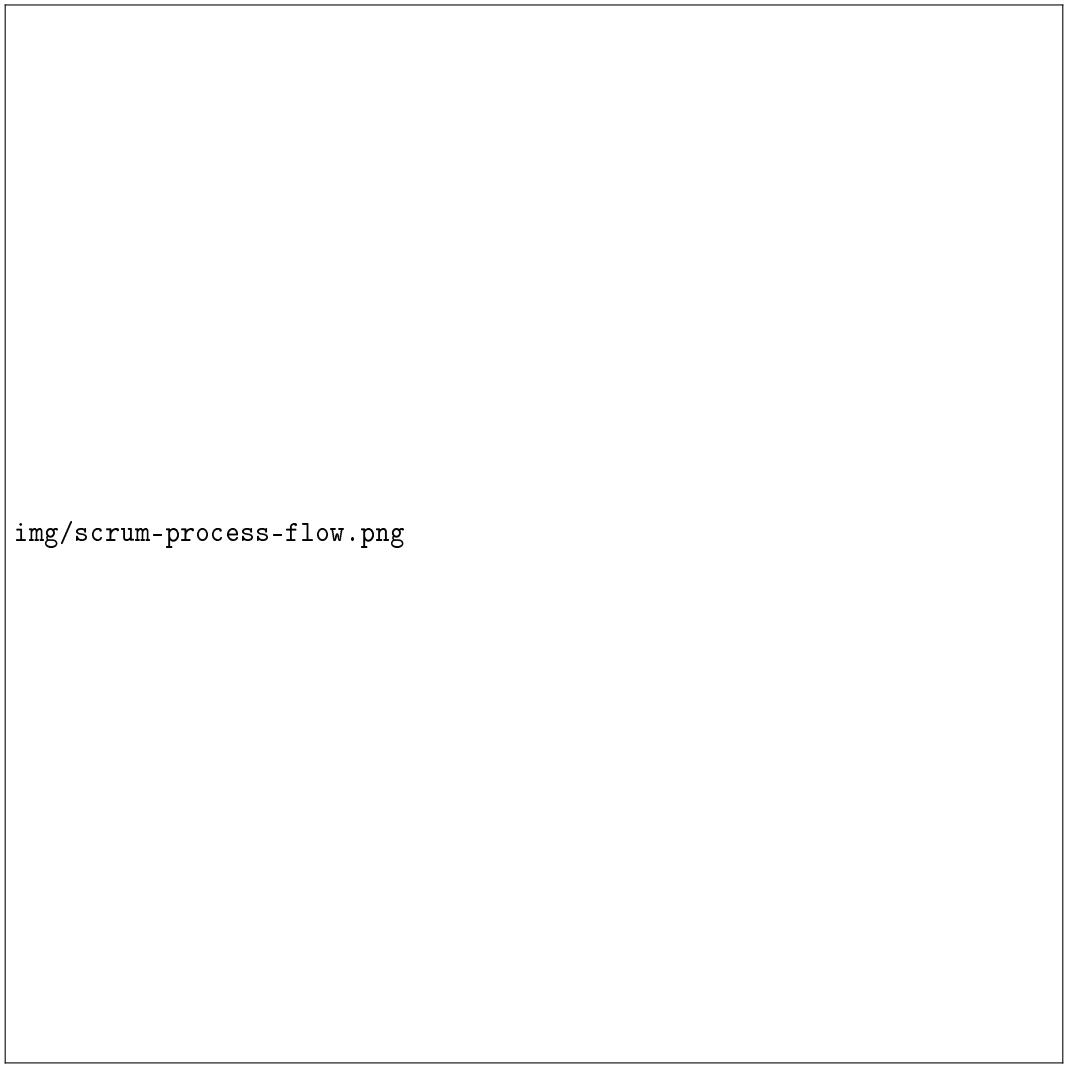

Figure 2.8 : Scrum Process Flow

2.7.2 How We Used Artifacts

Throughout the MyLoc project, these artifacts were :

— **Maintained Continuously** : The Product Backlog was refined weekly, with new items added and priorities adjusted based on stakeholder feedback.

— **Reviewed During Ceremonies** : Sprint Backlogs were created during Sprint Planning and reviewed daily during Stand-ups.

— **Shared Transparently** : All artifacts were accessible to stakeholders, promoting trust and collaboration.

— **Used for Decision Making** : Burndown and velocity data informed sprint planning and release forecasting.

2.8 Product Backlog

The Product Backlog is the master list of all features, requirements, enhancements, and fixes that constitute the changes to be made to the product. Each item is described as a User Story with acceptance criteria, priority, and story point estimation.

2.8.1 User Stories and Prioritization

User stories follow the format : *"As a [role], I want [feature] so that [benefit]"*. Priority levels are defined using MoSCoW method (Must have, Should have, Could have, Won't have).

2.8.1.1 Authentication, Car Management & Booking (Sprints 1-3)

ID	User Story Description	Priority	Sprint
US-01	Visitor registers an account	Must	1
US-02	User logs in securely	Must	1
US-03	Admin manages user roles	Must	1
US-04	User resets password	Should	1
US-05	User updates profile	Should	2
US-06	Agency adds new cars	Must	2
US-07	Agency edits car details	Must	2
US-08	Agency deletes cars	Must	2
US-09	Agency uploads car photos	Must	2
US-10	Customer browses cars	Must	2
US-11	Customer filters cars by criteria	Should	2
US-12	Customer views car details	Must	2
US-13	Customer checks car availability	Must	3
US-14	Customer submits rental request	Must	3
US-15	Agency views rental requests	Must	3
US-16	Agency approves/rejects requests	Must	3
US-17	Customer receives email notifications	Must	3

Tableau 2.5 : Product Backlog - Part 1 : Core Features (Sprints 1-3)

2.8.1.2 Communication, Blog, Contracts & DevOps (Sprints 4-8)

2.8.2 Backlog Summary

2.9 Sprint Planning

The project was executed over 8 sprints, each lasting 10-15 days. The team velocity stabilized at approximately 26 story points per sprint after the initial sprints.

ID	User Story Description	Priority	Sprint
US-18	Customer views booking history	Should	4
US-19	Customer cancels pending requests	Should	4
US-20	Admin chats with agencies	Must	4
US-21	Agency receives real-time messages	Must	4
US-22	User receives in-app notifications	Should	4
US-23	Customer contacts admin via email	Should	4
US-24	Admin creates blog posts	Should	5
US-25	Admin edits/deletes blogs	Should	5
US-26	Visitor reads blogs	Should	5
US-27	Customer comments on blogs	Could	5
US-28	Visitor becomes Follower (email subscription for new car alerts)	Could	5
US-29	Customer receives PDF contract	Must	6
US-30	Contract includes QR code	Should	6
US-31	Customer pays online	Must	6
US-32	Agency tracks payments	Should	6
US-33	Visitor uses chatbot	Should	7
US-34	Customer uses multilingual chatbot	Could	7
US-35	Admin manages agencies	Must	7
US-36	Admin views statistics	Should	7
US-37	Admin manages customers	Should	7
US-38	Developer creates Docker containers	Must	8
US-39	Developer sets up CI/CD pipelines	Must	8
US-40	Developer deploys on Kubernetes	Should	8

Tableau 2.6 : Product Backlog - Part 2 : Advanced Features (Sprints 4-8)

Priority Level	Number of Stories	Total Story Points
Must Have	22	132
Should Have	14	63
Could Have	4	14
Total	**40**	**209**

Tableau 2.7 : Product Backlog Summary by Priority

2.9.1 Sprint Overview

2.9.2 Sprint 1 : Authentication & Platform Setup

Sprint Goal : Set up development environment and implement secure multi-role authentication using Keycloak.

Duration : 12 days

Sprint 1 Retrospective :

— *What went well :* Keycloak integration was smoother than expected.

— *What could improve :* Initial environment setup took longer due to Docker configuration

Sprint	Goal	Duration	Planned SP	Completed SP
1	Authentication & Setup	12 days	24	24
2	Car Management Module	14 days	28	28
3	Booking System Core	15 days	31	31
4	Communication Features	12 days	24	24
5	Blog & Subscription	10 days	17	17
6	Contracts & Payments	15 days	31	31
7	Chatbot & Admin Panel	12 days	21	21
8	DevOps & Deployment	15 days	34	34
Total (105 days)			**210**	**210**

Tableau 2.8 : Sprint Overview - Planned vs Completed Story Points

Task	Description	Assignee	Hours	Status
T1.1	Initialize Angular 16 project	Dev	4h	Done
T1.2	Set up Spring Boot 3.x backend	Dev	4h	Done
T1.3	Configure MySQL database	Dev	6h	Done
T1.4	Deploy Keycloak container	Dev	4h	Done
T1.5	Implement registration API	Dev	8h	Done
T1.6	Implement JWT login	Dev	10h	Done
T1.7	Create AuthGuard	Dev	8h	Done
T1.8	Password reset feature	Dev	6h	Done
T1.9	Login/register UI	Dev	8h	Done
T1.10	Unit testing	Dev	6h	Done

Tableau 2.9 : Sprint 1 Backlog - Authentication & Setup

on Windows.

— *Action items :* Document Docker setup steps for future reference.

2.9.3 Sprint 2 : Car Management Module

Sprint Goal : Implement complete CRUD operations for car listings with image upload functionality.

Duration : 14 days

2.9.4 Sprint 3 : Booking System Core

Sprint Goal : Implement the complete rental request workflow with availability checking and email notifications.

Duration : 15 days

Task	Description	Assignee	Hours	Status
T2.1	Design Voiture entity	Dev	4h	Done
T2.2	Create car CRUD endpoints	Dev	8h	Done
T2.3	Image upload service	Dev	6h	Done
T2.4	Car listing component	Dev	6h	Done
T2.5	Car detail page	Dev	4h	Done
T2.6	Search & filter functionality	Dev	8h	Done
T2.7	Agency dashboard	Dev	10h	Done
T2.8	Profile update feature	Dev	6h	Done
T2.9	Form validation	Dev	4h	Done
T2.10	Integration testing	Dev	6h	Done

Tableau 2.10 : Sprint 2 Backlog - Car Management

Task	Description	Assignee	Hours	Status
T3.1	Design Booking entity	Dev	4h	Done
T3.2	Availability checking algorithm	Dev	10h	Done
T3.3	Booking request API	Dev	8h	Done
T3.4	Booking form with date picker	Dev	6h	Done
T3.5	Agency request management	Dev	8h	Done
T3.6	Approve/reject workflow	Dev	6h	Done
T3.7	Configure SMTP email	Dev	4h	Done
T3.8	Email notification templates	Dev	6h	Done
T3.9	Booking history page	Dev	6h	Done
T3.10	End-to-end testing	Dev	8h	Done

Tableau 2.11 : Sprint 3 Backlog - Booking System

2.9.5 Sprint 4 : Communication Features

Sprint Goal : Implement real-time chat system and notification infrastructure.

Duration : 12 days

Task	Description	Assignee	Hours	Status
T4.1	Design ChatMessage entity	Dev	3h	Done
T4.2	WebSocket configuration	Dev	8h	Done
T4.3	Chat service & endpoints	Dev	6h	Done
T4.4	Chat UI component	Dev	8h	Done
T4.5	Notification entity & service	Dev	6h	Done
T4.6	Notification dropdown	Dev	4h	Done
T4.7	Contact admin feature	Dev	4h	Done
T4.8	Booking cancel functionality	Dev	4h	Done
T4.9	Polish chat interface	Dev	4h	Done
T4.10	Real-time testing	Dev	6h	Done

Tableau 2.12 : Sprint 4 Backlog - Communication Features

2.9.6 Sprint 5 : Blog & Subscription System

Sprint Goal : Implement blog management and email subscription for visitors.

Duration : 10 days

Task	Description	Assignee	Hours	Status
T5.1	Design Blog entity	Dev	3h	Done
T5.2	Blog CRUD API endpoints	Dev	6h	Done
T5.3	Admin blog management	Dev	6h	Done
T5.4	Public blog listing	Dev	4h	Done
T5.5	Blog detail with comments	Dev	6h	Done
T5.6	Comment functionality	Dev	4h	Done
T5.7	Follower entity	Dev	2h	Done
T5.8	Subscription API	Dev	4h	Done
T5.9	Subscription form	Dev	3h	Done
T5.10	Feature testing	Dev	4h	Done

Tableau 2.13 : Sprint 5 Backlog - Blog & Subscription

2.9.7 Sprint 6 : Contracts & Payments

Sprint Goal : Implement PDF contract generation with QR codes and online payment integration.

Duration : 15 days

Task	Description	Assignee	Hours	Status
T6.1	Research PDF libraries (iText)	Dev	4h	Done
T6.2	Contract PDF template	Dev	6h	Done
T6.3	PDF generation service	Dev	10h	Done
T6.4	QR code integration (ZXing)	Dev	6h	Done
T6.5	Payment microservice	Dev	6h	Done
T6.6	Payment API endpoints	Dev	10h	Done
T6.7	Payment email template	Dev	4h	Done
T6.8	Payment webhook	Dev	6h	Done
T6.9	Auto-send PDF on payment	Dev	4h	Done
T6.10	Payment-to-contract testing	Dev	8h	Done

Tableau 2.14 : Sprint 6 Backlog - Contracts & Payments

2.9.8 Sprint 7 : Chatbot & Admin Panel

Sprint Goal : Integrate AI chatbot and complete admin dashboard functionalities.

Duration : 12 days

Task	Description	Assignee	Hours	Status
T7.1	FastAPI project setup	Dev	4h	Done
T7.2	OpenAI ChatGPT integration	Dev	6h	Done
T7.3	Chatbot REST endpoints	Dev	4h	Done
T7.4	Chatbot UI widget	Dev	6h	Done
T7.5	Multilingual support (FR/EN)	Dev	4h	Done
T7.6	Agency management (Admin)	Dev	6h	Done
T7.7	Customer management (Admin)	Dev	4h	Done
T7.8	Statistics dashboard	Dev	6h	Done
T7.9	Data visualization charts	Dev	4h	Done
T7.10	Integration testing	Dev	4h	Done

Tableau 2.15 : Sprint 7 Backlog - Chatbot & Admin Panel

2.9.9 Sprint 8 : DevOps & Deployment

Sprint Goal : Containerize all services and establish CI/CD pipeline with Kubernetes deployment.

Duration : 15 days

Task	Description	Assignee	Hours	Status
T8.1	Dockerfile for Angular	Dev	4h	Done
T8.2	Dockerfile for Spring Boot	Dev	4h	Done
T8.3	Dockerfile for FastAPI	Dev	3h	Done
T8.4	docker-compose.yml	Dev	6h	Done
T8.5	GitLab CI/CD pipeline	Dev	10h	Done
T8.6	GitLab Container Registry	Dev	4h	Done
T8.7	K8s deployment YAMLs	Dev	10h	Done
T8.8	K8s services & ingress	Dev	6h	Done
T8.9	Minikube deployment	Dev	8h	Done
T8.10	Documentation	Dev	4h	Done

Tableau 2.16 : Sprint 8 Backlog - DevOps & Deployment

2.10 Sprint Burndown Analysis

The burndown chart tracks the remaining work (in story points) over the course of each sprint. Figure 2.8 illustrates the project-level burndown across all sprints.

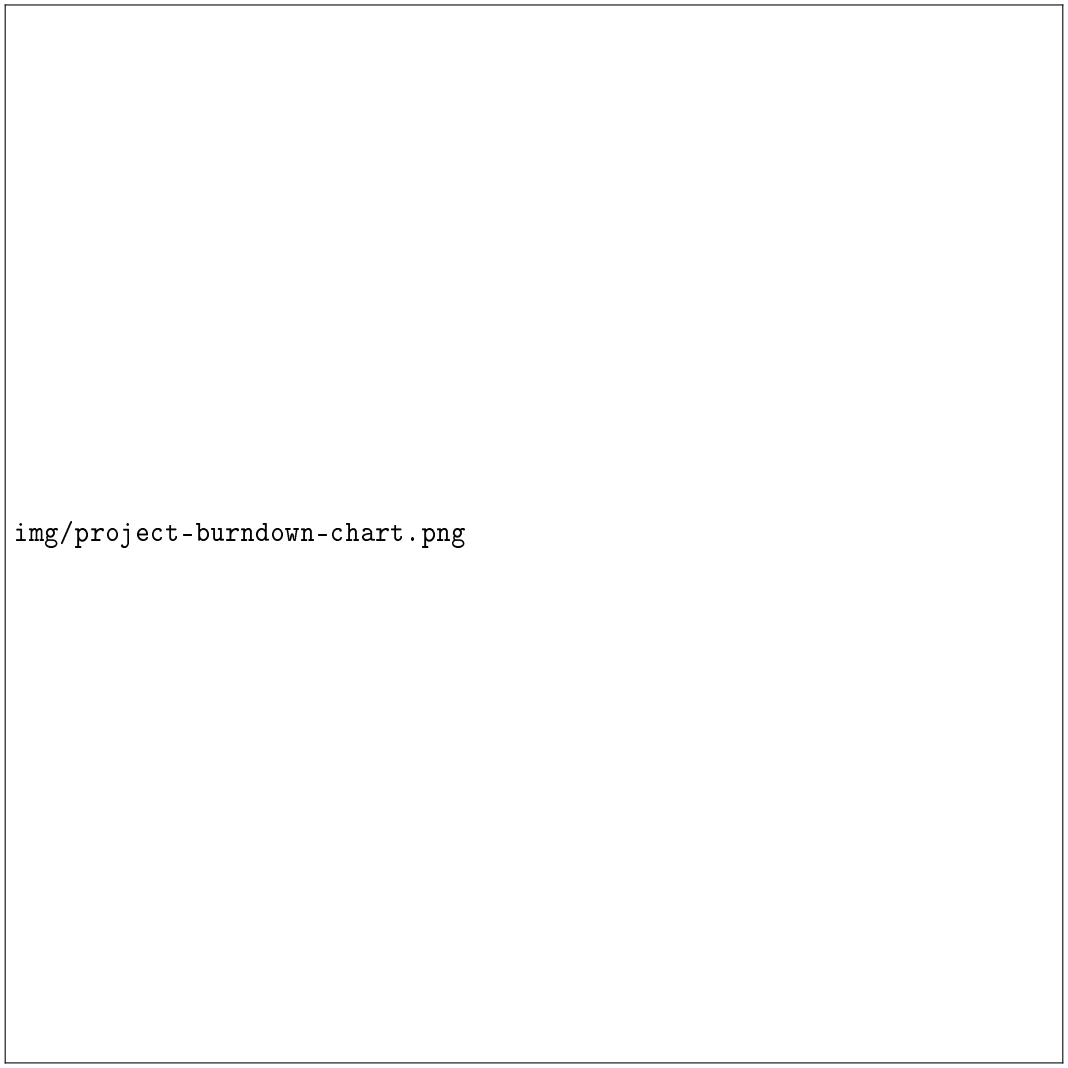

Figure 2.9 : Project Burndown Chart - Story Points Remaining Over Sprints

Sprint	Start SP	Completed SP	Remaining
Project Start	210	–	210
Sprint 1 (12 days)	210	24	186
Sprint 2 (14 days)	186	28	158
Sprint 3 (15 days)	158	31	127
Sprint 4 (12 days)	127	24	103
Sprint 5 (10 days)	103	17	86
Sprint 6 (15 days)	86	31	55
Sprint 7 (12 days)	55	21	34
Sprint 8 (15 days)	34	34	0
Total (105 days)	**210**	**210**	**0**

Tableau 2.17 : Project Burndown - Story Points per Sprint

2.10.1 Project Burndown Data

2.10.2 Sprint 3 Detailed Burndown (Example)

The following table shows the daily burndown for Sprint 3 (Booking System), demonstrating how work was completed throughout the 15-day period.

Day	Ideal	Actual	Done
Day 1	28.9	31	0
Day 2	26.8	29	2
Day 3	24.8	26	3
Day 4	22.7	24	2
Day 5	20.7	21	3
Day 6	18.6	18	3
Day 7	16.5	15	3
Day 8	14.5	12	3
Day 9	12.4	9	3
Day 10	10.3	7	2
Day 11	8.3	5	2
Day 12	6.2	3	2
Day 13	4.1	2	1
Day 14	2.1	1	1
Day 15	0	0	1

Tableau 2.18 : Sprint 3 Daily Burndown Chart Data

2.11 Team Velocity

Team velocity measures the amount of work completed per sprint, expressed in story points. This metric helps in future sprint planning and capacity estimation. Figure 2.9 visualizes the velocity trend across all sprints.

Velocity Analysis :

— The team maintained an average velocity of **26.25 story points per sprint**.

— Sprint 5 had lower velocity due to the complexity of blog features being lower than estimated.

— Sprint 8 had higher velocity due to familiarity with DevOps tasks and parallel work streams.

— Velocity remained consistent, indicating stable team performance and accurate estimation.

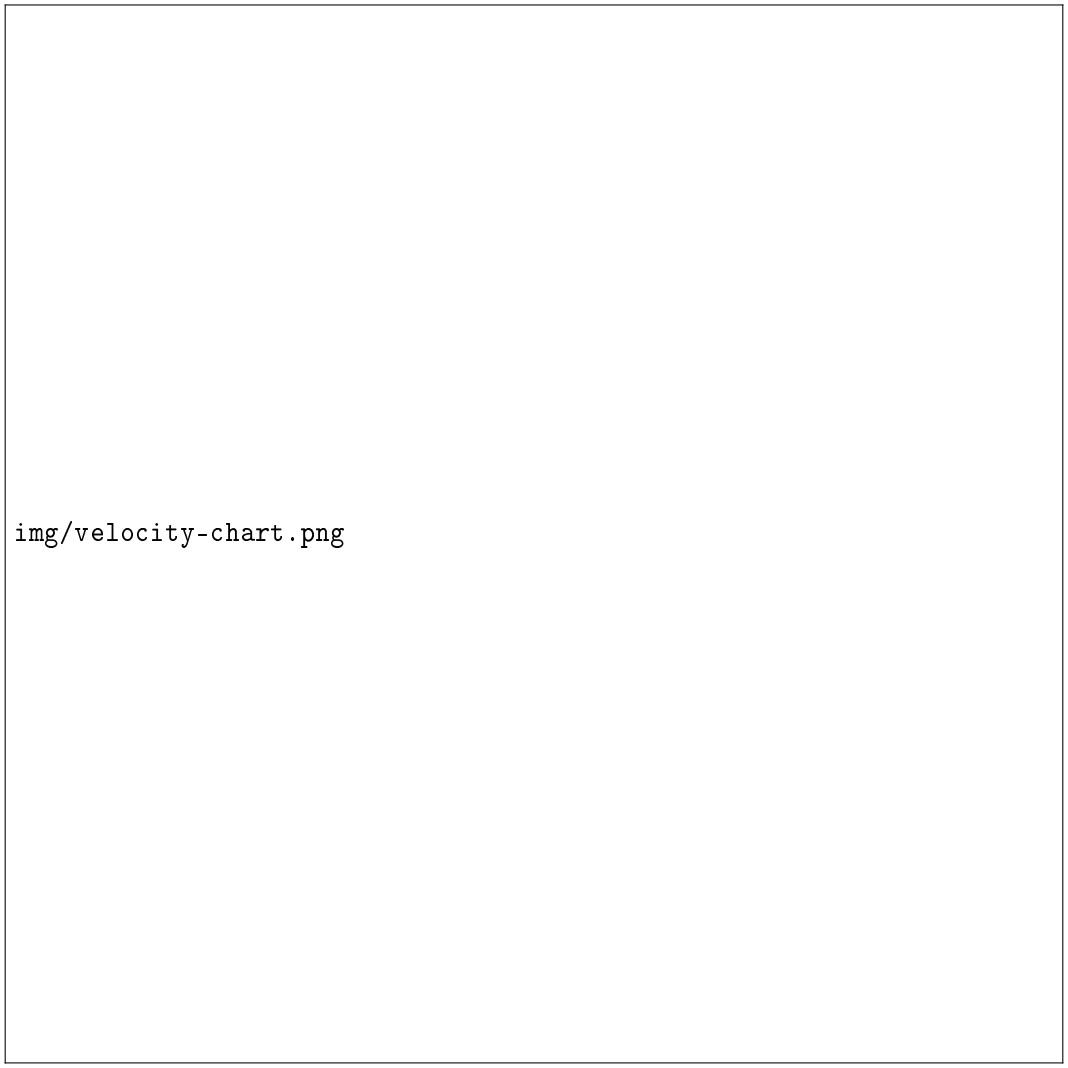

Figure 2.10 : Team Velocity Chart - Story Points Completed per Sprint

Sprint	Duration	Velocity (SP)	Cumulative SP
Sprint 1	12 days	24	24
Sprint 2	14 days	28	52
Sprint 3	15 days	31	83
Sprint 4	12 days	24	107
Sprint 5	10 days	17	124
Sprint 6	15 days	31	155
Sprint 7	12 days	21	176
Sprint 8	15 days	34	210
Average Velocity		**26.25 SP**	**210 SP Total**

Tableau 2.19 : Team Velocity per Sprint

2.12 Definition of Done (DoD)

A user story is considered "Done" when all the following criteria are met :

— Code is written and follows project coding standards.

— Unit tests are written and pass successfully.

— Code is reviewed by at least one team member.

— Feature is integrated with the main branch.

— Feature is tested in the staging environment.

— Documentation is updated (API docs, README if needed).

— No critical or high-severity bugs remain.

— Product Owner accepts the feature during Sprint Review.

2.13 Scrum Ceremonies

The following Scrum ceremonies were conducted throughout the project :

Ceremony	Frequency
Sprint Planning	Start of sprint
Daily Stand-up	Daily (15 min)
Sprint Review	End of sprint
Retrospective	End of sprint
Backlog Refinement	Weekly

Tableau 2.20 : Scrum Ceremonies Overview

2.13.1 Chapter Conclusion

In this chapter, we explored the functional and non-functional requirements for the car rental platform. We identified the different user roles (Admin, Agency, Customer, Visitor), their needs, and outlined the system features accordingly.

We also established the development methodology using Scrum, assigned team roles (Product Owner, Scrum Master, Development Team), and laid out the complete sprint plan with a detailed product backlog containing 40 user stories totaling 210 story points. The project was organized into 8 sprints with an average team velocity of 26.25 story points per sprint.

This structured specification ensures the platform is developed efficiently, securely, and in alignment with user needs through iterative development and continuous feedback.

2.14 Conclusion

The analysis and specification phase has established a solid foundation for the project. With clearly defined requirements, comprehensive use case diagrams, and a detailed Scrum

planning with product backlog, the development team is well-prepared to begin implementation.

In the following chapters, we will detail the realization of each sprint, starting with Sprint 1 focused on platform setup and secure authentication using Keycloak.

Chapitre 3

System Development and Implementation

3.1 Introduction

This chapter details the development environment, system architecture, database design, and implementation strategies used for the Car Rental Platform. It also explains how functional requirements are translated into working modules and how non-functional requirements such as security, scalability, and real-time communication are implemented.

3.2 Development Environment

— **Frontend :** Angular 16, Node.js 20

— **Backend :** Spring Boot 3.x, Java 17 (REST API)

— **Database :** MySQL 8

— **Authentication :** Keycloak (Docker container)

— **Chatbot :** Flask API (Python) integrated with ChatGPT

— **Containerization :** Docker Compose

— **Orchestration :** Kubernetes (Minikube)

— **IDE :** Visual Studio Code

— **API Testing :** Postman

— **Local DB :** XAMPP

3.3 Database Design

The system uses MySQL as its relational database. Key tables include :

3.3.1 Agence Table

```
CREATE TABLE agence (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    agency_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(64) NOT NULL,
    photo LONGTEXT,
    phone_number VARCHAR(20) NOT NULL,
    city VARCHAR(100)
);
```

3.3.2 Blog Table

```
CREATE TABLE blog (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    img_url VARCHAR(255),
    author VARCHAR(255),
    date VARCHAR(255),
    time VARCHAR(255),
    description VARCHAR(1024),
    quote VARCHAR(255)
);
```

3.3.3 Booking Table

```
CREATE TABLE booking (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255),
    user_email VARCHAR(255),
    phone VARCHAR(255),
    description VARCHAR(255),
    start_date DATE,
    end_date DATE,
    price DOUBLE,
    voiture_id BIGINT,
    nbr_jrs INT,
    booking_status VARCHAR(255),
    pickup_location VARCHAR(255),
    dropoff_location VARCHAR(255),
    car_name VARCHAR(255),
    agence TEXT
);
```

3.3.4 Other Tables

Additional tables include `customers`, `roles`, `notifications`, `followers`, and `chat_messages` to manage users, roles, communications, and subscription updates.

3.4 Backend Implementation

— **Spring Boot REST API :** Handles CRUD operations for cars, agencies, bookings, and blogs.

— **Security :**

  — Keycloak handles login and registration for admins and agencies.

  — AuthGuard in Angular protects routes based on roles.

  — Only authorized users can access admin or agency dashboards.

— **Booking Workflow :**

  — Customer requests a car rental ; the system checks car availability via `getUnavailableDates()`.

  — Request is sent to the responsible agency.

  — Agency accepts/declines the request ; email notifications are sent accordingly.

  — Accepted requests generate PDF contracts with a QR code sent to the customer.

— **Real-time Communication :** Admins and agencies communicate via instant messaging.

— **Chatbot :** Integrated via external Flask API, accessible to visitors and customers for queries.

3.5 Frontend Implementation

— Angular components handle car listings, rental requests, blogs, notifications, and chat.

— Reactive forms validate inputs for registration, login, booking, and comments.

— Guards ensure route protection according to user roles.

— Email and PDF confirmations are triggered using backend API services.

3.6 Deployment Strategy

— **CI/CD :** GitLab pipelines automate build and deployment.

— **Docker :** Separate containers for Angular frontend, Spring Boot backend, MySQL, and Flask chatbot.

— **Kubernetes :** Minikube manages deployments with YAML files such as `mysql-deployment.yml`, `frontend-deployment.yml`, `backend-deployment.yml`, `ai-deployment.yml`.

3.7 Development Methodology

— Scrum methodology with iterative sprints and backlog management.

— Team roles : Scrum Master, Developers, Product Owner.

— Regular sprint reviews and retrospectives ensure continuous improvement.

3.8 Conclusion

This chapter demonstrates the technical setup, database design, backend and frontend implementation, deployment strategy, and development methodology for the Car Rental Platform. It ensures secure, real-time, and scalable service delivery to admins, agencies, customers, and visitors.

Chapitre 4

Implementation and Results

4.1 Introduction

This chapter presents the practical implementation of the Car Rental Platform. It details how the system architecture, design decisions, and core functionalities have been translated into a working product. The chapter is organized by module, describing role-specific components, backend logic, frontend interaction, security integration, third-party service connections, and test scenarios. Screenshots and results are included to illustrate the platform's functionalities.

4.2 System Modules Implementation

The application is organized into modular components corresponding to specific functionalities, user roles, and workflows. Each module is independently testable, scalable, and maintained in alignment with the microservice architecture.

4.2.1 Authentication and Authorization Module (Keycloak)

— Role-based access control is implemented using Keycloak.
— Users are authenticated via OAuth2 and issued JWT tokens.
— Angular AuthGuard protects Admin, Agency, and Customer dashboards.
— Spring Boot backend routes are secured with role-based annotations (e.g., `@PreAuthorize`, `@RolesAllowed`).

4.2.2 Admin Module

— Add, update, and delete agencies.
— Manage customers and blogs (CRUD operations).
— View and manage all bookings.
— Communicate with agencies via real-time chat.

4.2.3 Agency Module

— Add, edit, and delete cars.
— Upload photos for existing cars.
— Manage availability and booking requests.
— Accept or decline rental requests, triggering email and notification.
— Communicate with admin via real-time chat interface.

4.2.4 Customer Module

— Browse available cars and blogs.

— Register and log in with Keycloak (or browse as a visitor).

— Submit rental requests for available cars.

— Receive notifications and emails upon status changes.

— View rental history and delete unconfirmed or declined requests.

— Comment on car listings and blogs.

— Receive PDF contract with QR code after payment confirmation.

4.2.5 Chatbot Module (FastAPI + ChatGPT)

— FastAPI backend receives questions from the frontend.

— OpenAI's ChatGPT API generates responses.

— Replies are displayed to customers and visitors.

— Supports multilingual queries (French and English).

4.2.6 Online Payment Module

— Upon booking acceptance, a payment link is sent to the user's email.

— Payment is processed and validated through the Spring Boot backend.

— A PDF with QR code is generated and sent after payment confirmation.

— Booking status updated to "confirmed".

4.2.7 Notification and Email System

— Customers receive emails for new cars, booking decisions, or updates.

— Followers (visitors who submit emails) receive notifications for new car listings.

— Notifications are stored in the database and displayed in-app.

4.3 Database Implementation

The platform uses a MySQL relational database. Entities are mapped via JPA annotations and structured around key objects :

— **Agence Table :** Stores agency profiles and login credentials.

— **Booking Table :** Stores rental requests, dates, customer info, price, and car details.

— **Blog Table :** Contains articles, authors, multimedia content.

— **Voiture Table :** Stores car details (name, type, price, photos).

— **User Table :** Stores customer information and login references.

— **Notification Table :** Stores in-app messages and alert statuses.

4.4 Screenshots of the Application

Screenshots illustrate the platform's functionalities for different user roles :

— Admin dashboard with agency management.

— Agency dashboard with car CRUD operations.

— Customer booking page and rental history view.

— Real-time chat interface between admin and agencies.

— Generated PDF contract with QR code.

— Chatbot interface integrated on the homepage.

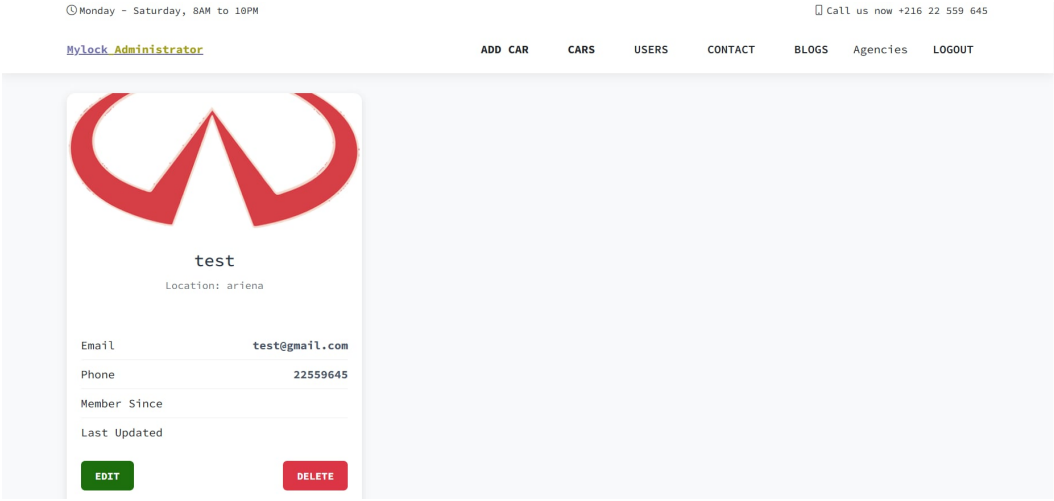

Figure 4.1 : Admin Dashboard for Managing Agencies

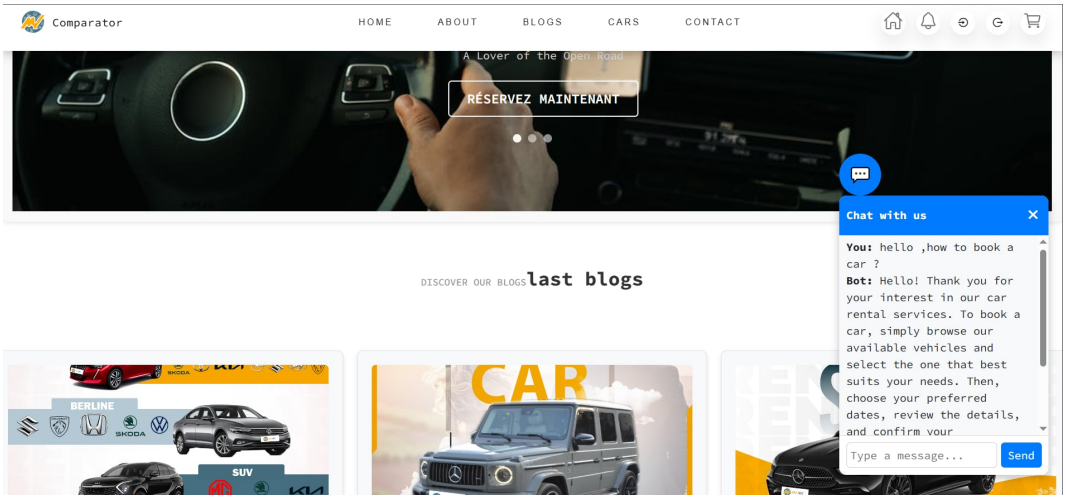

Figure 4.2 : Chatbot Integration on Homepage

4.5 Testing and Validation

The system has been validated through manual and automated testing.

4.5.1 Unit Testing

— Spring Boot unit tests for services and controllers.

— REST API endpoints tested with Postman.

— Angular unit tests using Jasmine and Karma.

4.5.2 Integration Testing

— End-to-end testing of booking flow from customer request to agency validation.

— Payment processing, PDF generation, and email dispatch tested with mock gateways.

— Real-time chat functionality tested between admin and agencies.

4.5.3 CI/CD Pipeline Validation

— GitLab CI pipelines test build processes, run unit tests, and deploy to Kubernetes.

— Docker containers tested locally and in staging environments.

— Rollback and hotfix procedures validated.

4.6 Conclusion

The implementation phase successfully transformed the design into a robust, functional web platform. All planned modules—from authentication and booking workflows to real-time messaging, automated email notifications, chatbot support, and online payments—have been developed, tested, and deployed. The system is stable, scalable, and provides a seamless digital experience for all user roles.

Chapitre 5

Conclusion and Future Work

5.1 Summary of the Project

This project developed a comprehensive Car Rental Platform that integrates multiple user roles, real-time communication, online payment processing, automated notifications, and AI-powered assistance. The platform enables :

— Admins to manage agencies, customers, blogs, and oversee bookings.

— Agencies to manage cars, accept or decline rental requests, and communicate with admins.

— Customers to browse cars, make rental requests, view history, comment, and receive automated emails and PDF confirmations.

— Visitors and customers to interact with a chatbot powered by ChatGPT for assistance.

The platform is developed using Angular for the frontend, Spring Boot for the backend, MySQL for the database, Keycloak for authentication, and FastAPI for the chatbot service. The system is containerized with Docker and deployed on Kubernetes using a CI/CD pipeline.

5.2 Key Achievements

— Successful implementation of role-based access control and secure authentication using Keycloak.

— Development of a real-time chat system for admin-agency communication.

— Automated booking workflow with email notifications and PDF contract generation.

— Integration of a multilingual AI chatbot for user assistance.

— Deployment of a scalable microservices architecture using Docker and Kubernetes, including :

  — Separate containers for frontend, backend, database, and AI services.

  — YAML deployment files for Minikube (`frontend-deployment.yml`, `backend-deployment.yml`, `mysql-deployment.yml`, `ai-deployment.yml`).

  — CI/CD pipeline using GitLab for automated builds, testing, and deployment.

— Implementation of unit, integration, and end-to-end testing to ensure system reliability.

5.3 Limitations

— Online payment module relies on mock gateways in the current implementation ; full integration with external payment providers is not yet complete.

— Chatbot responses may depend on external API latency and internet connectivity.

— Mobile responsiveness and offline capabilities can be further enhanced.

— Some modules could be optimized for performance under high concurrent user loads.

5.4 Future Work

— Integrate real payment gateways for live transactions and invoicing.

— Enhance the chatbot with advanced NLP capabilities and additional languages.

— Implement push notifications for mobile devices.

— Add analytics dashboards for admins and agencies to track usage, bookings, and revenue.

— Introduce recommendation systems to suggest cars to customers based on preferences and history.

— Optimize microservices performance and scalability for large-scale deployment.

— Improve deployment automation and monitoring using Kubernetes tools such as Helm, Prometheus, and Grafana.

5.5 Final Remarks

The Car Rental Platform provides a robust, secure, and user-friendly system that automates key rental workflows and integrates modern technologies such as AI and microservices. The deployment process demonstrates the ability to manage containerized applications, orchestrate them with Kubernetes, and use CI/CD pipelines for seamless updates. This platform serves as a solid foundation for future enhancements and real-world deployment in the car rental industry.

5.5.1 User Testimonials

To complement quantitative testing results, we collected feedback from real users who interacted with the platform. Selected testimonials include :

— **Bassem Ajengui (CEO & Founder) :** "Super application, très intuitive et facile à utiliser. Je recommande vivement !"

— **Assyl Kria (Designer) :** "Franchement, rien à dire ! Les voitures sont top et le service hyper pro."

— **Bayrem Boussaidi (Store Owner) :** "J'ai adoré la facilité de réservation et la qualité des véhicules proposés."

— **Aymen Arfaoui (Freelancer) :** "Une des meilleures agences de location de voitures, je reviendrai sans hésiter."

— **Mouhib Touati (Entrepreneur) :** "Je suis très satisfait du service, les prix sont abordables et les voitures impeccables."

These testimonials demonstrate positive reception from multiple user roles, confirming the platform's usability, clarity, and reliability.

Conclusion générale

Ce projet avait pour objectif de concevoir, développer et déployer une plateforme web complète de location de voitures, adaptée aux besoins de différents utilisateurs : administrateurs, agences, clients et visiteurs. En suivant une méthodologie Agile Scrum et en s'appuyant sur des outils DevOps modernes, nous avons réussi à mener à bien toutes les phases, depuis l'analyse des besoins jusqu'au déploiement sur un cluster Kubernetes.

La plateforme intègre avec succès plusieurs fonctionnalités clés :

— Authentification sécurisée multi-rôles et contrôle d'accès via Keycloak.
— Gestion des véhicules par les administrateurs et agences, incluant la gestion des photos.
— Processus de réservation fluide avec vérification en temps réel de la disponibilité et validation par l'agence.
— Notifications automatiques par email et génération de contrats PDF avec QR code intégré.
— Système de chat en temps réel facilitant la communication entre agences et administrateurs.
— Chatbot intelligent offrant un support 24h/24 via une intégration FastAPI et OpenAI.
— Pipelines CI/CD robustes utilisant GitLab, la containerisation Docker et un déploiement scalable via Kubernetes.

Ce projet a permis de combiner avec succès architecture RESTful, modularité microservices et outils d'intelligence artificielle pour offrir une solution complète, scalable et maintenable. Il ouvre la voie à de nombreuses améliorations futures tant techniques que fonctionnelles.

Ainsi, ce travail illustre parfaitement comment les technologies web modernes et les principes du cloud peuvent transformer un processus traditionnel de location en une expérience numérique avancée, apportant valeur ajoutée aux utilisateurs et aux gestionnaires du service.

Bibliographie

[B1] Pierre Pezziardi, Référentiel des Pratiques Agiles, édition ebook.2013

Résumé

FR

Mots clés : KW1, KW2

Abstract

EN

Keywords : KW1, KW2