Université Ibn Zohr – Agadir

FACULTÉ DES SCIENCES

Master d'excellence : Analytique des Données et Intelligence Artificielle

RAPPORT DE PROJET

Data Collector ADIA

Distributed AI-Powered Browser Automation System

Sujet : Conception et implémentation d'un système distribué Data Collector

Réalisé par :

Lamyae Chouinna	Mounir Aithammadi	Charaf Bathahi
Abderahim Ait Smain	Amin Aboukir	Abderhaman Laarigbat

Année Universitaire : 2025 – 2026

Contents

List of Figures

List of Tables

Abstract

The Data Collector ADIA project is a distributed microservices-based platform designed to automate web browsing tasks using artificial intelligence. The system relies on gRPC for high-performance inter-service communication, Playwright for browser automation, MongoDB for data persistence, and Streamlit for user interaction.

Each service operates independently and can be deployed on a separate machine, allowing scalability, fault isolation, and maintainability. This architecture makes the system suitable for large-scale automation, data collection, and research-oriented applications.

1. Introduction

The rapid evolution of web technologies has increased the demand for intelligent systems capable of interacting automatically with online platforms. Manual browsing and data extraction are inefficient and unsuitable for repetitive or large-scale tasks. As a result, browser automation combined with artificial intelligence has become a strategic solution for modern organizations.

However, many existing automation systems are monolithic, which makes them difficult to scale, maintain, and evolve. These limitations motivated the development of Data Collector ADIA, a distributed platform based on microservices that enables scalable and intelligent browser automation.

The main objective of this project is to design and implement a robust, modular, and production-ready system that separates concerns across independent services while ensuring efficient communication and real-time monitoring.

2. System Overview

Data Collector ADIA is designed as a collection of autonomous services that collaborate to execute browser automation tasks. Each service has a clearly defined responsibility and communicates with others exclusively through gRPC interfaces defined using Protocol Buffers.

The platform includes a frontend service for user interaction, a backend orchestration service that coordinates execution logic, a browser service that manages browser instances, and a database service that ensures reliable data persistence. MongoDB is used as the storage engine due to its flexibility in handling semi-structured execution data.

This modular approach improves system clarity, maintainability, and scalability, while enabling independent development and deployment of each service.

Table 2.1: *Vue d'ensemble des services du système*

Service	Rôle principal	Port
Frontend (Streamlit)	Interface utilisateur : création de tâches, suivi en temps réel, consultation de l'historique.	8501
Backend Service	Orchestration : coordination des services, exécution de l'agent IA, gestion du cycle de vie des tâches.	50050
Browser Service	Gestion Playwright : lancement/arrêt des navigateurs, isolation, gestion des ressources.	50051
Database Service	Persistance : stockage des tâches, logs et résultats via MongoDB (accès centralisé).	50052
MongoDB	Base de données : collections `tasks` et `outputs`, stockage durable.	27017

3. System Architecture

The architecture of Data Collector ADIA follows the microservices paradigm, where each component runs as an independent process. This design improves scalability, fault isolation, and deployment flexibility.

All services communicate using gRPC, which ensures fast binary communication, strict type safety, and language independence. The backend service acts as the central coordinator, while other services focus on specialized responsibilities such as browser management or data persistence.

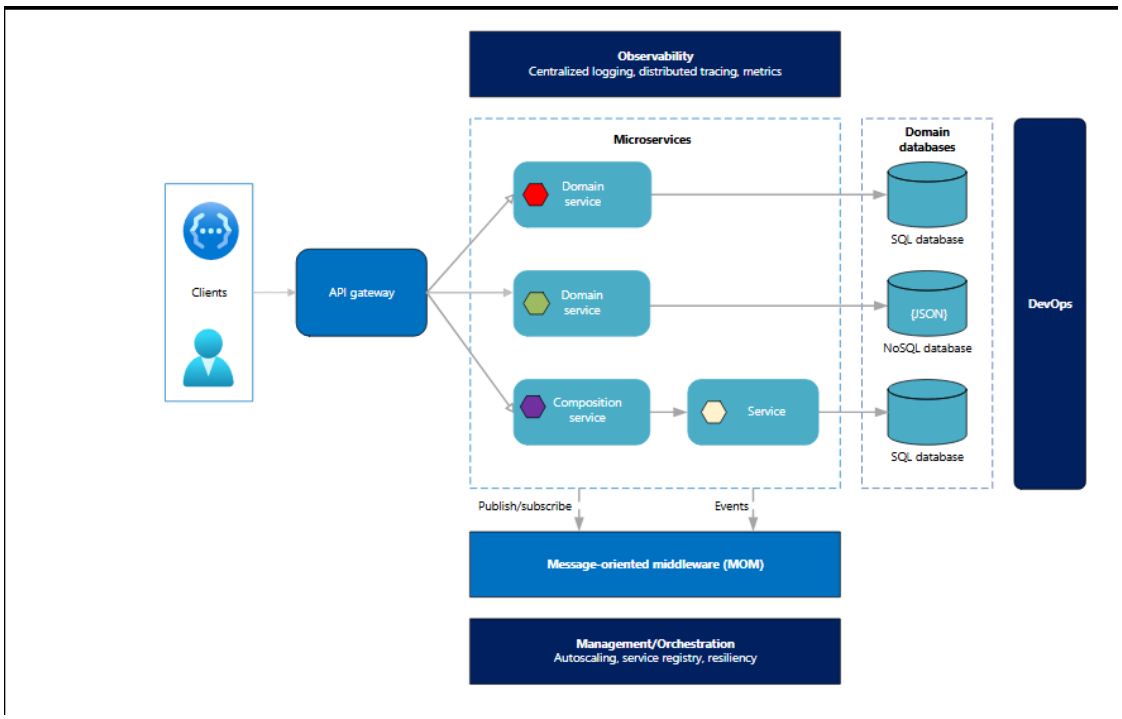

Figure 3.1: *Global microservices architecture of the Data Collector ADIA system*

3.1 Communication Protocol

gRPC was selected as the communication protocol due to its high performance and reliability. It uses binary serialization, which significantly reduces latency compared to traditional REST-based communication. Protocol Buffers enforce strict data contracts, reducing runtime errors and improving maintainability.

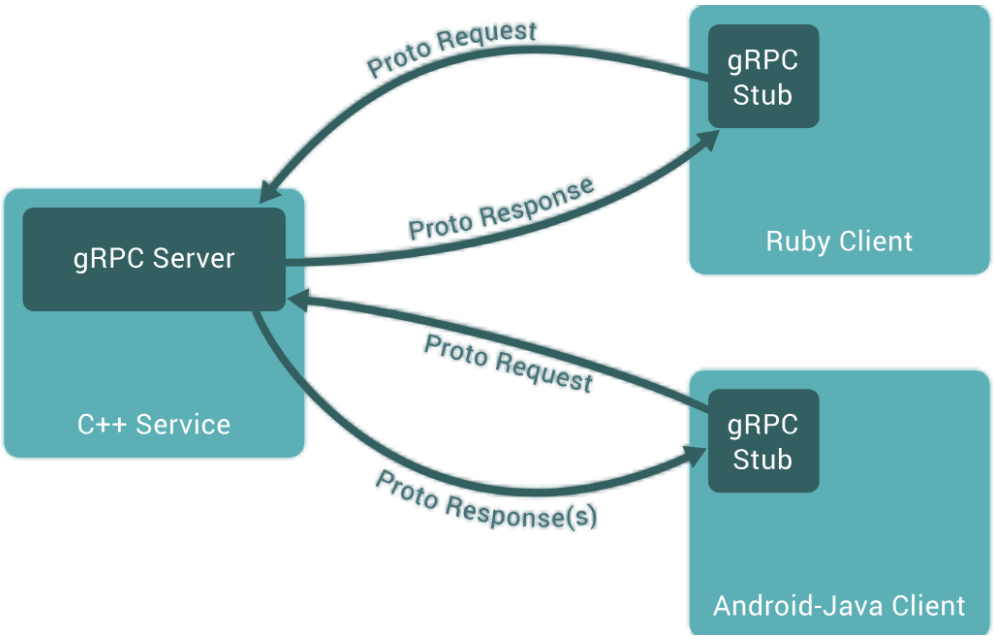

Figure 3.2: *Inter-service communication using gRPC and Protocol Buffers*

4. Services Description

4.1 Frontend Service

The frontend service is implemented using Streamlit and provides the primary interface between the user and the system. It allows users to create automation tasks by specifying prompts, execution parameters, and configuration options. In addition, the frontend offers real-time monitoring of task execution by displaying progress indicators, execution logs, and final outputs.

This service is designed to be lightweight, user-friendly, and easy to deploy. By decoupling the user interface from the orchestration logic, the system ensures that frontend evolution does not impact backend stability. This design also enables the potential introduction of alternative clients in the future (web applications, mobile apps, or command-line interfaces).

4.2 Backend Service

The backend service represents the core intelligence of the platform. It is responsible for orchestrating tasks, managing the execution lifecycle, and coordinating communication between independent services. When a task request arrives from the frontend, the backend validates the input parameters, initializes the task state, and triggers the automation workflow.

The backend integrates AI-driven logic (via the browser_use agent execution) to determine browser actions and handle dynamic environments. Unlike static scripts, AI-driven automation can adapt to changing web pages, unexpected user flows, and complex navigation structures. The backend also ensures that task execution remains consistent by handling retries, step validation, and task status updates.

4.3 Browser Service

The browser service encapsulates all browser-related functionality and provides an abstraction layer over Playwright. Its core mission is to manage browser instances and ensure controlled access to automation capabilities. It handles the lifecycle of browser processes,

including initialization, configuration, execution, and cleanup.

Separating browser management into a dedicated service improves overall system stability. It isolates heavy resources (browser processes) from the orchestration logic and prevents a browser crash from directly affecting the backend. Additionally, it allows independent scaling: if many automation tasks run concurrently, the browser service can be replicated or deployed on more powerful machines.

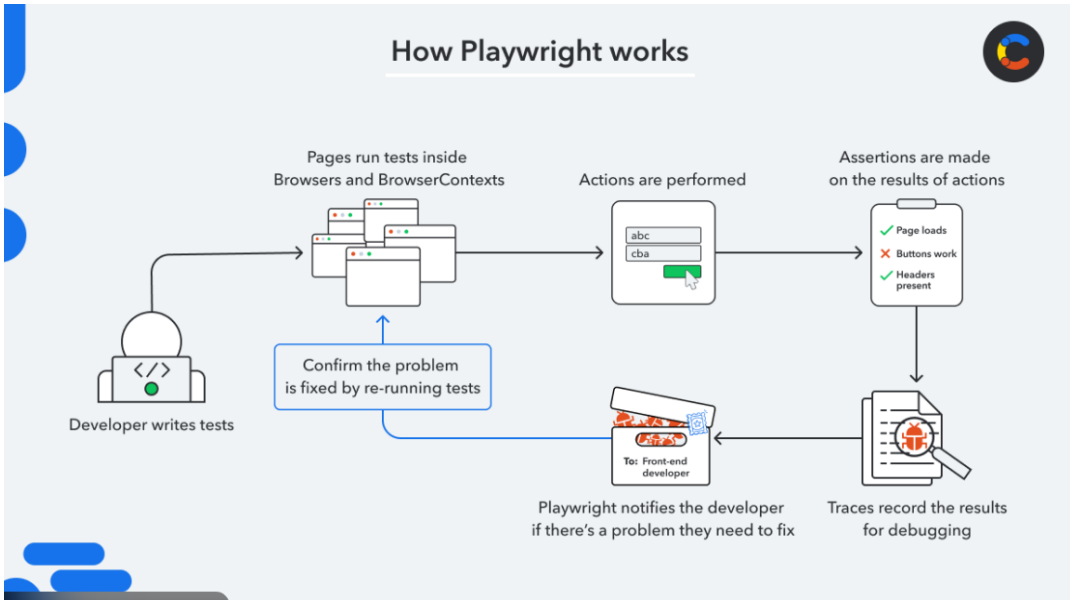

Figure 4.1: *Browser automation architecture using Playwright*

4.4 Database Service

The database service abstracts all persistence operations and acts as the unique entry point for MongoDB access. Instead of allowing direct database communication from multiple services, all read/write operations pass through the database service using gRPC. This approach improves security, consistency, and maintainability.

This service stores task metadata (status, parameters, timestamps), execution history (step-by-step logs), and outputs (results, extracted data, reports). By centralizing persistence logic, the system ensures traceability and supports analytics use cases, such as measuring performance, debugging failed tasks, or reviewing historical executions.

4.5 MongoDB

MongoDB was selected as the storage engine due to its document-oriented model and schema flexibility. Automation outputs are often semi-structured and may vary between tasks. MongoDB allows the system to store structured task metadata while keeping execution logs and outputs in flexible formats.

MongoDB also provides high write throughput, which is important because automation tasks generate continuous logs during execution. This makes it suitable for workloads where the system must store large volumes of execution data over time.

5. System Execution, Deployment and Perspectives

This chapter presents a comprehensive view of how the Data Collector ADIA system operates in practice. It combines the execution workflow, deployment strategy, system strengths, limitations, and future perspectives into a unified and coherent analysis.

5.1 Execution Workflow

The execution workflow of Data Collector ADIA begins with user interaction through the frontend interface. The user defines an automation task by specifying a prompt and execution parameters. Once submitted, the task request is transmitted to the backend service using gRPC.

The backend service validates the request and initializes the task lifecycle. It acts as the central coordinator, determining the sequence of operations required to execute the task. To perform browser automation, the backend requests a browser instance from the browser service. This instance is created and managed in isolation to ensure stability and prevent interference with other tasks.

During execution, the backend continuously controls browser actions through AI-driven logic. Each step of execution generates logs, intermediate states, and results. These data elements are sent to the database service, which persists them in MongoDB for traceability and historical analysis.

Meanwhile, the frontend receives real-time updates from the backend, allowing the user to monitor progress, view logs, and access results. This workflow ensures transparency, reliability, and a clear separation of responsibilities across services.

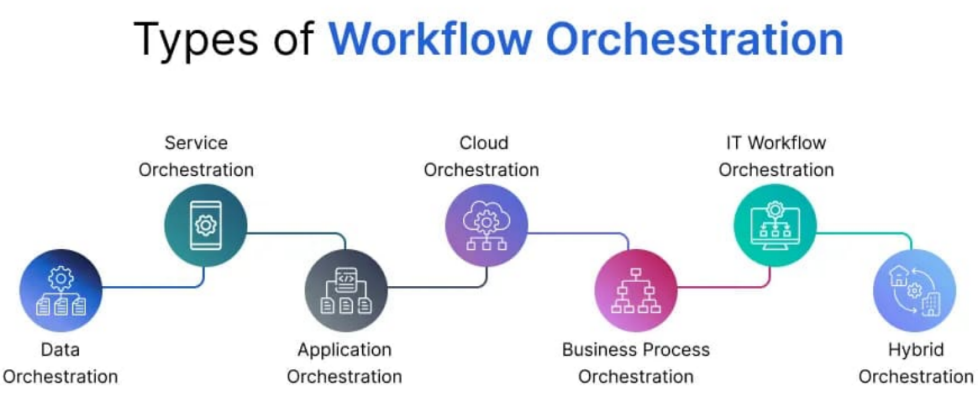

Figure 5.1: *End-to-end task execution workflow*

5.2 Deployment Strategy

The deployment strategy of Data Collector ADIA is designed to support both development and production environments. During development and testing, all services can be executed on a single machine using separate processes and ports. This approach simplifies debugging, integration testing, and rapid iteration.

In a production environment, each service is deployed independently on a dedicated machine or container. The frontend, backend, browser service, database service, and MongoDB can each be scaled according to workload requirements. This deployment model improves system availability, fault isolation, and scalability.

The use of gRPC enables efficient communication even in distributed environments, making the system suitable for cloud or on-premise deployment scenarios.

5.3 System Advantages

One of the main strengths of Data Collector ADIA lies in its microservices architecture. By separating responsibilities across independent services, the system achieves high modularity and maintainability. Each service can be updated, scaled, or replaced without impacting the entire platform.

The use of gRPC provides high-performance communication and strict data contracts, reducing latency and runtime errors. Additionally, the separation of browser management into a dedicated service improves stability and resource control. The integration of artificial intelligence in the backend enables flexible and intelligent task execution, allowing the system to adapt to complex automation scenarios.

5.4 Limitations

Despite its strengths, the current version of Data Collector ADIA presents some limitations. The system does not yet include authentication or authorization mechanisms, which may limit its use in sensitive environments. Deployment and scaling operations are currently performed manually, requiring operational expertise.

Furthermore, advanced fault tolerance mechanisms such as automatic retries, circuit breakers, service discovery, and centralized monitoring are not yet fully implemented. These limitations represent opportunities for future enhancements rather than fundamental design flaws.

5.5 Future Perspectives

Several improvements can be considered to enhance the system. The integration of authentication and authorization mechanisms would secure access to services and data. Containerization and orchestration technologies such as Docker and Kubernetes could automate deployment and scaling.

Monitoring and observability tools could be added to track system health, performance metrics, and error rates. Finally, additional AI capabilities could be integrated to further improve decision-making and automation intelligence. Together, these enhancements would transform Data Collector ADIA into a fully production-grade automation platform.

6. Conclusion

The Data Collector ADIA project demonstrates the effectiveness of applying modern distributed system principles to intelligent browser automation. By adopting a microservices architecture and leveraging gRPC for communication, the system achieves scalability, flexibility, and robustness while remaining modular and maintainable.

A key contribution of this project is the clear separation of concerns across services. The frontend focuses on usability and transparency by offering task creation and real-time monitoring. The backend serves as the orchestration and intelligence layer, coordinating execution and integrating AI-driven decision logic. The browser service isolates heavy browser workloads and provides controlled Playwright automation capabilities, which improves stability and simplifies scaling. The database service centralizes persistence operations, ensuring traceability and enabling historical analysis through structured task storage and execution logs.

From a software engineering perspective, the chosen architecture reflects industry-grade practices. The use of Protocol Buffers ensures strict service contracts, reduces integration errors, and facilitates future evolution. Moreover, the distributed deployment model allows the platform to adapt to real-world operational constraints, such as running browser instances on dedicated machines or scaling database capacity independently.

Although the current system can be improved by adding authentication, automated orchestration, and advanced fault tolerance mechanisms, the existing foundation is technically solid and extensible. The suggested perspectives (security, Kubernetes orchestration, observability, and richer AI capabilities) provide a clear roadmap toward a production-ready solution.

In conclusion, Data Collector ADIA is not only a successful academic project but also a practical and scalable platform that can be extended for industrial automation, large-scale data collection, and research-oriented experimentation. It represents a strong baseline for future work and demonstrates the value of combining microservices, gRPC communication, and AI-driven automation in a coherent engineering solution.

Bibliography

[1] gRPC Documentation. https://grpc.io

[2] Protocol Buffers. https://protobuf.dev

[3] Playwright Documentation. https://playwright.dev

[4] MongoDB Documentation. https://www.mongodb.com

[5] Streamlit Documentation. https://streamlit.io