Republic of Tunisia
The Ministry of Higher Education
and Scientific Research

The University of Carthage

The Higher Institute of Information and
Communication Technologies

END-OF-STUDY PROJECT REPORT

Submitted in Partial Fulfillment of the Requirements for the
BACHELOR DEGREE IN COMPUTER SCIENCE

Field of Study : Software Engineering and Information Systems

Legalyz

By
MAHDI ABDELRAHMEN SFEYHI
&
INSAF BEN TALOUBA

Conducted within Addval Solutions

Publicly defended on May 31, 2025 in front of the jury members:

Chairman:	Name SURNAME, University Relations Leader, IBM
Reporter:	Name SURNAME, Teacher, ISTIC
Examiner:	Name SURNAME, Teacher, ISTIC
Professional Supervisor:	Aymen BELKADHI, CEO, Addval Solutions
Academic Supervisor:	Saif eddine AZZABI, Teacher, ISTIC

Academic Year: 2024-2025

Republic of Tunisia
The Ministry of Higher Education
and Scientific Research

The University of Carthage

The Higher Institute of Information and
Communication Technologies

END-OF-STUDY PROJECT REPORT

Submitted in Partial Fulfillment of the Requirements for the
BACHELOR DEGREE IN COMPUTER SCIENCE

Field of Study : Software Engineering and Information Systems

Legalyz

By
MAHDI ABDELRAHMEN SFEYHI
&
INSAF BEN TALOUBA

Conducted within Addval Solutions

Authorization of graduation project report submission:

Professional Supervisor:	Academic Supervisor:
Issued on :	Issued on :
Signature:	Signature:

Résumé

Ce rapport propose le développement d'un chatbot destiné à accompagner les entrepreneurs tunisiens dans le choix d'un nom pour leur startup, en se basant sur le Registre National des Entreprises (RNE). L'outil permet aux utilisateurs de vérifier la disponibilité du nom souhaité et de s'assurer qu'il respecte les réglementations en vigueur. Grâce à l'intelligence artificielle, le système offre une expérience fluide et efficace, simplifiant ainsi une étape clé de la dénomination d'une startup. La solution repose sur une architecture agentique, utilisant des modèles d'IA spécialisés qui interagissent entre eux pour fournir une aide à la décision en temps réel, avec précision.

Mots clés : Inscription des startups, Chatbot, Agents IA, LLMs.

Abstract

This report proposes the development of a chatbot to help Tunisian entrepreneurs in selecting a name for their startup from The National Register of Companies (RNE). The chatbot helps users in verifying the availability of their desired startup name, and verifying whether it is in line with naming regulations. By utilizing AI technology, the system provides a smooth and efficient experience, simplifying a vital process in the registration of a startup. The solution uses an agentic architecture, with customized AI models that communicate with one another to deliver accurate, real-time decision support.

Keywords: Startup Registration, Chatbot, AI Agents, LLMs.

Dedication

To our parents, who gave us the gift of dreams and the ability to realize them,

To our friends, who taught us the true meaning of friendship,

To our professors, who taught me the value of critical thinking.

Acknowledgement

We would like to express our sincere gratitude to several individuals for supporting us throughout our End-of-Study Internship Project.

First and foremost, we wish to express our sincere thanks to our professional supervisor, Doctor Mr. **Aymen KADHI** at Addval Solutions, who encouraged us, guided us, and warmly welcomed us into his startup.

We would also like to extend our appreciation to our co-supervisor, Engineer Mr. **Khalil BESSAAD**, who provided us with valuable technical guidance and support. This project accomplishment would not have been possible without his input.

We are deeply grateful to our academic mentor, Professor **Mohammed Seifeddine AZZABI**, for his insightful feedback, continuous follow-up, and academic supervision throughout this internship.

Contents

Dedication **ii**

Acknowledgement **iii**

General Introduction **1**

1 General Context **2**

General Context **2**

1.1 Introduction 3
1.2 Hosting Company 3
1.3 Project Overview 3
1.4 Study of the Existing 4
1.4.1 Accessing the Simulator 5
1.4.2 Searching for a Proposed Name 5
1.4.3 Valid Name: No Conflicts Detected 5
1.4.4 Name Conflict: Similar or Identical Names Found . 6
1.5 Critique of the Existing System 6
1.5.1 The Distinctive Element Requirement 7
1.5.2 Technical Knowledge Barrier 7
1.5.3 Disjointed Verification Process 8
1.5.4 Technical Vulnerabilities 8
1.6 Proposed Solution 8
1.7 Methodology: LEAN Approach 9
1.8 Key Concepts 9
1.8.1 Natural Language Processing (NLP) 10
1.8.2 Large Language Models (LLMs) 11
1.8.3 AI Agents 12
1.8.4 LLM Improvement 17
1.9 Conclusion 18

2 State of the Art 20
2.1 Comparative Analysis 21
2.1.1 Workflow Engine 21
2.1.2 Python vs. TypeScript in Agentic AI Development 22
2.1.3 LLM 22
2.1.4 ORM 23
2.1.5 Database 23
2.2 Methodology 24
2.2.1 System Overview 24
2.2.2 Technologies Used 24
2.2.3 Architecture 25
2.3 Conclusion 26
2.4 POETRY 26

3 Requirements Specification 30
3.1 Introduction 31
3.2 Functional Requirements Identification 31
3.3 Non-Functional Requirements Identification 31
3.4 Actors 32
3.5 Use Case Diagram 33
3.6 Activity Diagram 35
3.7 Conclusion 35

4 Realization 37
4.1 Introduction 38
4.2 Iteration 1: Initial Graph Version 38
4.2.1 Objectives 38
4.2.2 Development Process 38
4.2.3 Results 41
4.3 Version 2 42
4.3.1 Objectives 42
4.3.2 Development and Technologies 42
4.3.3 Results and Audit 42
4.4 Version 3 42
4.4.1 Objectives 42
4.4.2 Development and Technologies 42
4.4.3 Results and Audit 43
4.5 Conclusion 43

General Conclusion 44

Bibliography 46

Annexe 1, Les candidats classés par ordre alphabétique 47

List of Figures

1.1 Example of a valid name search with no results 6
1.2 Search results showing conflicting or similar names 7
1.3 The 5 Principles of Lean Methodology[4] 9
1.4 langgraph-vs-autogen-vs-crewai[12] 15

2.1 langgraph-vs-autogen-vs-crewai[12] 21
2.2 drizzle vs prisma [5] 23
2.3 final graph 26

3.1 Legalyz Chatbot Use Case Diagram 34
3.2 Activity Diagram 36

4.1 initial graph 39
4.2 final graph 42
4.3 HITL 43
4.4 user declines 43

List of Tables

1.1 Comparison between AutoGen, CrewAI, and LangGraph[1] 14
1.2 Comparison of Python and TypeScript in Agentic AI Development 16

2.1 Comparison of Python and TypeScript in Agentic AI Development 27
2.2 Feature Comparison of Models[14] 28
2.3 Benchmark Results Comparison[14] 28
2.4 Comparison between Firebase and Supabase [2] 29

General Introduction

Ce document vise à aider les étudiants à rédiger des rapports en Latex.

Chapter 1

General Context

1.1 Introduction

This chapter presents the general context of the project. It begins with an overview of the hosting company, Addval Solutions, followed by a detailed description of the project and its objectives. We then examine the current existing system , highlighting its limitations. Finally, we introduce the proposed solution.

1.2 Hosting Company

The project is hosted by **Addval Solutions**, an AI technology startup that offers innovative tech solutions that drive business growth. Their service entails AI-powered automation, custom IT solutions, and cloud services to optimize operations and create lasting value for clients.

Created in 2019, Addval Solutions specializes in developing scalable, secure, and tailor-made software solutions to meet diverse business needs. With the power of artificial intelligence and intelligent automation, they enhance efficiency and automate processes. They specialize in a wide range of domains including fintech, cloud, automation, machine learning and AI, cybersecurity, web development, and SaaS.

By creating strategic alliances with premium industry brands and by maintaining an ongoing focus on the latest advancements in technology, Addval Solutions guarantees that their clients receive excellent services customized to meet their individual needs [1].

1.3 Project Overview

The Tunisian National Register of Companies (RNE), established in 2019, is a central and fundamental system for the administration of business. The RNE is the official database that registers the legal and operational information of all small and big businesses in the country. It aims to enhance transparency, ensure that businesses follow legal requirements, and maintain an up-to-date and accessible company register [2].

The RNE offers a set of services that includes the issue of authentic documents proving the company's legal existence and registration data as well as publication of the legal announcements. It facilitates reservation of company names, issues certificates of pledges or leasing arrangements, and allows access to beneficial ownership details.

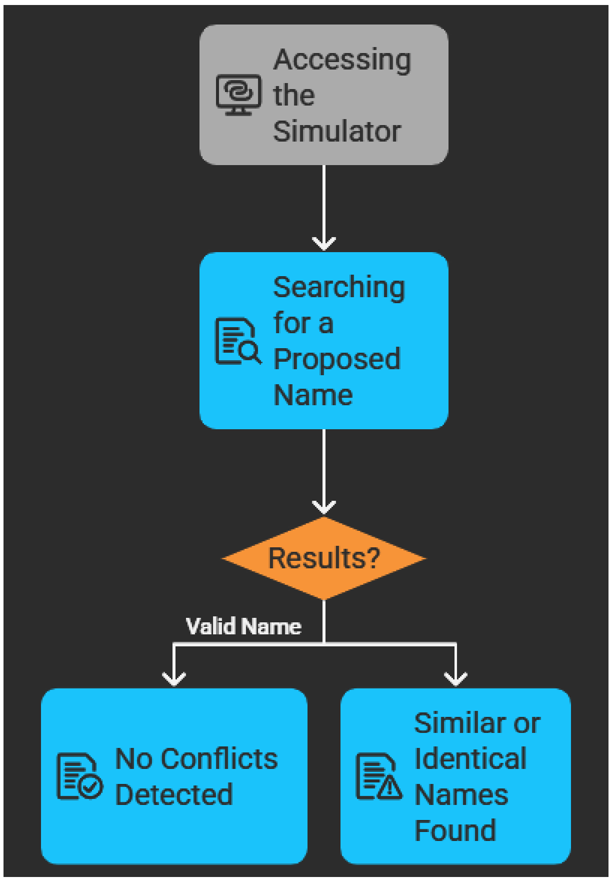
By being the sole repository of company data, the RNE makes it possible for public authorities and the general public to have access to reliable information on companies, their legal existence, and their activities. It promotes a secure business climate, simplicity of financial transactions, and allows government agencies to effectively monitor and regulate the business community.

One of the most crucial procedures for any entrepreneur who desires to have a legal presence in Tunisia is the registration of the company in the RNE. This process involves several phases, starting with the verification of the availability of the company name, which is essential before proceeding. Entrepreneurs must submit three name proposals in both Arabic and Latin scripts to increase the chances of approval. Once the name availability is confirmed, the next step is to provide details about the company's legal status and identify the applicant. In certain cases, supporting documents are required, such as when establishing a foreign branch or using a parent company name. Finally, after all information is verified, payment follows as the last step in the process.

To further improve the experience of entrepreneurs and make the registration process easier, this project proposes an AI chatbot that is custom-built to guide help through one of the most important part of the RNE registration process: choosing the name of the startup. The chatbot enables users to easily check the availability of their preferred company name and suggest new names if necessary. The chatbot considerably saves time and effort in business registration, making it more convenient and simpler for first-time entrepreneurs.

1.4 Study of the Existing

To understand the current registration flow, we analyzed the name availability simulator provided by the Tunisian National Register of Companies (RNE). This simulator is a critical part of the registration process, offering insight into name uniqueness and potential conflicts. The following subsections walk through each step or possible outcome of the simulator's usage which is illustrated in the Figure 1.4.



1.4.1 Accessing the Simulator

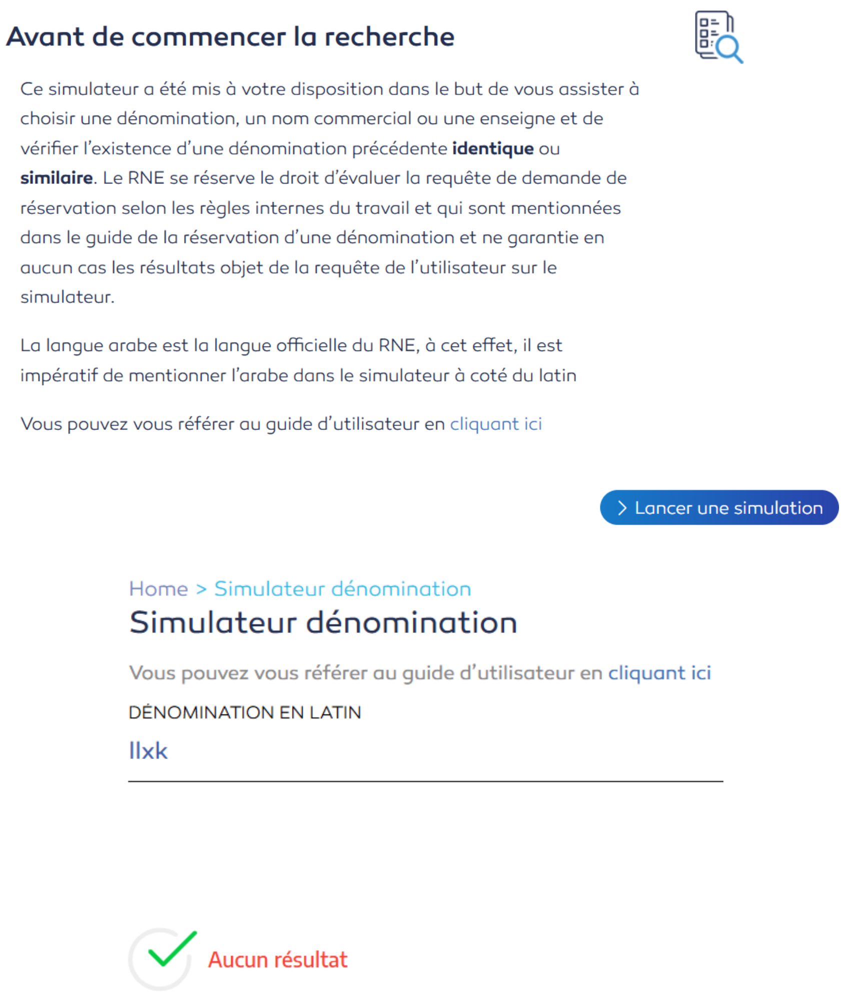
The simulator is accessible through the official RNE platform, but only after the user creates an account and logs in. As illustrated in Figure 1.4.1, users can then navigate to the name search interface. The interface supports bilingual input (Arabic and Latin), accommodating the linguistic diversity of proposed business names.

1.4.2 Searching for a Proposed Name

Once logged in, users input the desired company name into the search bar. The simulator acts like a search engine, scanning the RNE database for identical or similar entries. It supports both Latin and Arabic scripts, enhancing its effectiveness for bilingual evaluations.

1.4.3 Valid Name: No Conflicts Detected

If the entered name does not match any existing entries, the simulator displays a “No results” message, suggesting the name is potentially available. Figure 1.1 shows an example where the name “Ilxk” returned no matches.


Avant de commencer la recherche

Ce simulateur a été mis à votre disposition dans le but de vous assister à choisir une dénomination, un nom commercial ou une enseigne et de vérifier l'existence d'une dénomination précédente **identique** ou **similaire**. Le RNE se réserve le droit d'évaluer la requête de demande de réservation selon les règles internes du travail et qui sont mentionnées dans le guide de la réservation d'une dénomination et ne garantie en aucun cas les résultats objet de la requête de l'utilisateur sur le simulateur.

La langue arabe est la langue officielle du RNE, à cet effet, il est impératif de mentionner l'arabe dans le simulateur à coté du latin

Vous pouvez vous référer au guide d'utilisateur en cliquant ici

Home > Simulateur dénomination

Simulateur dénomination

Vous pouvez vous référer au guide d'utilisateur en cliquant ici

DÉNOMINATION EN LATIN

llxk

Aucun résultat

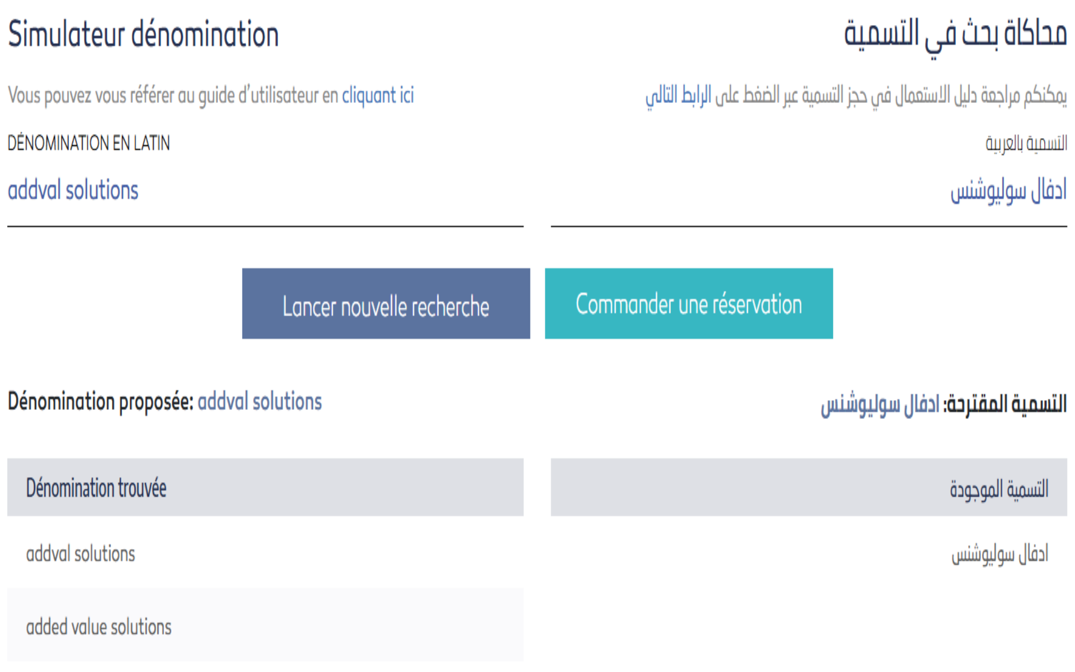
Figure 1.1: Example of a valid name search with no results

1.4.4 Name Conflict: Similar or Identical Names Found

If there are identical or similar names already registered, the system lists them in the results. This helps users gauge name availability and consider alternative formulations. Figure 1.2 shows the outcome for a search using ”addval solutions” where existing entries created potential conflicts.

1.5 Critique of the Existing System

This section highlights the main limitations and challenges users face with the current RNE name verification system.


Simulateur dénomination

محاكاة بحث في التسمية

Vous pouvez vous référer au guide d'utilisateur en cliquant ici

يمكنكم مراجعة دليل الاستعمال في حجز التسمية عبر الضغط على الرابط التالي

DÉNOMINATION EN LATIN: addval solutions

التسمية بالعربية: ادفال سوليوشنس

Dénomination proposée: addval solutions

التسمية المقترحة: ادفال سوليوشنس

Dénomination trouvée	التسمية الموجودة
addval solutions	ادفال سوليوشنس
added value solutions	

Figure 1.2: Search results showing conflicting or similar names

1.5.1 The Distinctive Element Requirement

One of the basic limitations arises from the way the system deals with *distinctive elements* - the distinguishing elements that make a business name different from generic terms. RNE regulations specify:

- **Definition**: A distinctive element is a novel word or combination that cannot be confused with generic product/service terms [3] (e.g., "Textile Manufacturing" is generic, but "BlueDune Textile Manufacturing" contains the distinctive "BlueDune")

The present system does not:

- Clearly explain this requirement during registration
- Give real-time name uniqueness feedback
- Automatically split generic and distinguishing features

1.5.2 Technical Knowledge Barrier

The system assumes that users have special legal and administrative expertise, such as:

- Knowledge of all forms of business legal entities (SARL, SA, etc.)

- Familiarity with technical jargon such as “distinctive elements”
- Knowledge of prohibited elements (symbols, state symbols, etc.)

1.5.3 Disjointed Verification Process

The workflow suffers from deep fragmentation:

- No real-time validation upon submitting name.
- Users must manually verify names in the simulator before registering.
- The majority of users prefer to type names in Latin script but struggle to type Arabic translations, yet the system provides no assistance for proper transliteration or bilingual consistency verification.

1.5.4 Technical Vulnerabilities

The system demonstrates several technical shortcomings:

- Typo-sensitivity with no autocorrection or suggestions
- Insufficient rule checking (it doesn’t enforce all the naming rules by default)
- No input checking for necessary distinctive elements

These limitations cumulatively create a system that is hard for non-professional users to complete successfully, commonly involving several submission attempts and external legal guidance.

1.6 Proposed Solution

To address the limitations of the current system, we present an agentic Generative AI chatbot engine that guides applicants through the verification and denomination process by clarifying complex requirements. This solution utilizes autonomous agents in task decomposition, parallel processing, and optimization processing. Features include step-by-step directions with smart guidance, bilingual processing with dynamic translation, real-time name verification, and an agent-based design that ensures cooperative and efficient execution.

1.7 Methodology: LEAN Approach

Lean methodology is a set of practices and principles aimed at the optimization of business processes. The two overarching pillars of lean thinking are continuous improvement and respect for people. In practice, lean methodology aims to create as much value for the customer as possible, while simultaneously fostering an efficient workplace that values respect and teamwork. Lean manufacturing provided the original framework for the application of lean practices. Today, lean methodology is applied across many fields, from healthcare to software development [11]. Figure 1.3 shows the five key principles that form the foundation of the Lean methodology.

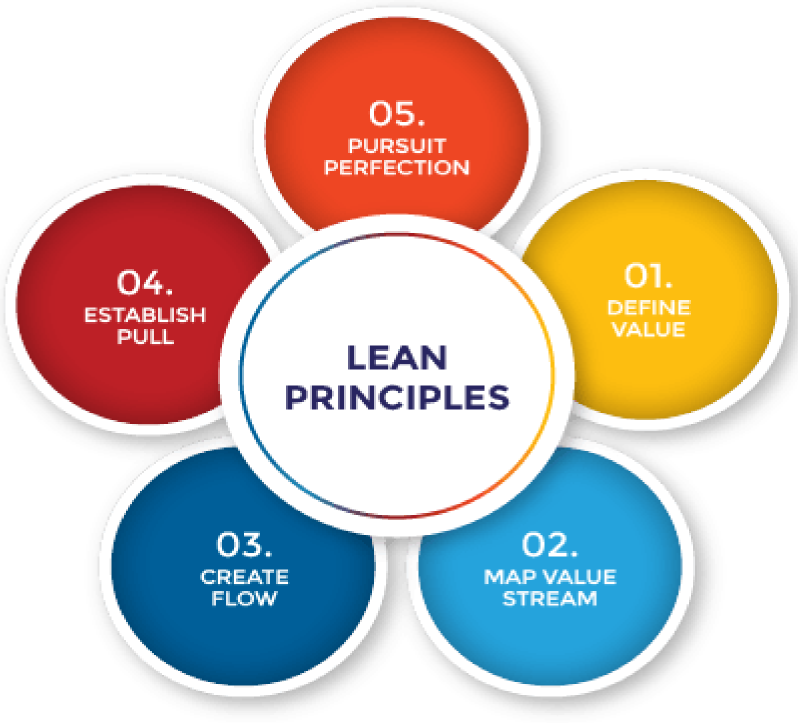


Figure 1.3: The 5 Principles of Lean Methodology[4]

1.8 Key Concepts

This section introduces the foundational ideas necessary to understand natural language processing, large language models (LLMs), agentic artificial intelligence and architectural patterns suited for agent-based systems.

1.8.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that deals with making machines comprehend, generate, and communicate in human language. NLP employs insights from linguistics, computer science, and machine learning to narrow the gap between human language and computer systems [8].

At its heart, Natural Language Processing (NLP) comprises a number of basic tasks. These are tokenization, dividing text into words or subwords; part-of-speech tagging, the tagging of each word in a sentence with a word class—like noun, verb, or adjective; and syntactic parsing, the breaking down of a sentence's structure to determine how the words depend upon each other, usually depicted as a parse tree [10]. NLP also involves sentiment analysis and machine translation. These operations attempt to extract structured sense from unstructured text so that machines can reason about language.

Traditionally, early Natural Language Processing (NLP) systems were grounded in symbolic, rule-based theories with a strong dependence on highly engineered grammars and lexicons. Although they worked well within limited domains, these systems were afflicted with poor robustness and scalability. Statistical NLP then provided probabilistic modeling, along with learning from data to boost performance [10].

The new age of Natural Language Processing (NLP) has been revolutionized by the advancements in the field of deep learning. Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) have made sequence modeling of text possible [6]. Furthermore, word embeddings have made it possible to model semantic word relationships in dense vector spaces [13].

A significant breakthrough was achieved with the introduction of transformer-based architectures. Transformers abandoned the constraints of sequential processing by completely depending on attention mechanisms [15]. This architectural paradigm forms the foundation for models such as BERT, which introduced bidirectional contextual representations [3], and GPT, which showed the effectiveness of large-scale pretraining for generalizing to a wide range of language tasks.

Today, Natural Language Processing (NLP) is the underlying foundation for systems more than static language comprehension; it enables conversational agents, intelligent assistants, and autonomous agents to reason, act, and engage in conversation in natural language. Thus, a profound comprehension of NLP is of utmost significance in the development of agentic systems that use language as a medium and interface for cognitive processes.

1.8.2 Large Language Models (LLMs)

Large Language Models (LLMs) are deep learning models that process and generate human language very effectively. LLMs generally follow the Transformer architecture, which has also become the basis for most current state-of-the-art NLP systems. LLMs can conduct a variety of natural language tasks, including text generation, translation, summarization, and question answering. Their capacity to express complex patterns of language a has had a major influence on many fields of NLP.

The main strength of Large Language Models (LLMs) lies in their scalability. LLMs are designed to handle enormous amounts of data, often drawn from a diverse range of material such as books, websites, research articles, and social media. By training on such large datasets, LLMs learn intricate relations between words, phrases, and sentences, thus enabling them to generate language that is contextually relevant and coherent. This capability is particularly significant in applications such as machine translation and text summarization, where it is essential to understand the full context in order to produce accurate output.

One of the crucial advances in the development of large language models (LLMs) was the introduction of unsupervised pretraining. In this phase, the models are trained on huge text unlabeled datasets. The training often involves next word prediction in a sequence through autoregressive training the model generates text one word at a time, predicting the next word based on the previous ones. It does this by conditioning each word prediction on the entire sequence that came before it, enables the model to discover general patterns and structures within language. After this phase, LLMs are fine-tuned on task-specific data, which enables them to be adapted to particular applications, such as answering questions or determining the sentiment of a sentence.

One of the most well-known examples of LLMs is GPT, which employs autoregressive training. GPT has been shown to have spectacular performance on an extremely wide range of NLP tasks, from producing coherent text, sentence completion, to even writing essays. Its massive size, in terms of billions of parameters, causes it to generalize incredibly well to a wide range of tasks, even in the absence of task-specific fine-tuning.

Despite their impressive capabilities, LLMs still face challenges. One of the major issues is the interpretability of these models. Due to their large size and complexity, it is difficult to fully understand how LLMs arrive at specific outputs, making it challenging to explain their decisions. Furthermore, these models can exhibit biases learned from the data they are trained on, which can result in unintended and harmful outputs. Ethical concerns, such as the potential for misuse in generating misleading or harmful content, also remain a significant issue for the deployment of LLMs in real-world applications.

Nevertheless, LLMs have revolutionized many areas of artificial intelligence and continue to drive advancements in conversational AI, automated content creation, and more. They serve as the backbone for modern intelligent assistants, such as chatbots and agents, which can understand and respond to user queries in a human-like manner. As research progresses, LLMs are likely to become even more powerful and versatile, opening up new possibilities for natural language understanding and generation.

1.8.3 AI Agents

Definition

Agentic AI refers to artificial intelligence systems that possess the capability to autonomously plan, make decisions, and execute actions based on complex reasoning processes. These systems typically integrate large language models (LLMs), reasoning frameworks, and action interfaces to interact with various tools or environments. Unlike traditional rule-based systems, Agentic AI dynamically adapts its behavior in real time, enabling it to handle nuanced, open-ended tasks through self-directed workflows and goal-driven planning.

Types of Agents

Agentic AI can be broadly categorized into several types based on their autonomy, purpose, and complexity:

- **Reactive Agents:** These agents operate based on immediate inputs and produce outputs without long-term planning. They are effective in fast, simple decision-making scenarios but lack adaptability to novel situations.

- **Deliberative Agents:** These agents construct and maintain internal models of the world, allowing them to plan actions by simulating outcomes. They are more sophisticated and capable of handling complex reasoning tasks.

- **Hybrid Agents:** Combining the strengths of both reactive and deliberative models, hybrid agents adaptively switch between fast reaction and strategic planning based on context.
- **Collaborative Agents:** These agents are designed to interact with other agents or humans, sharing goals and coordinating actions within multi-agent environments.

Agentic Architectures

Agentic architectures typically include three core components:

1. **Perception Module:** This component interprets inputs from the environment, converting unstructured data (e.g., text, speech, visuals) into structured representations that the agent can reason over.
2. **Cognitive Layer:** Powered by LLMs and reasoning engines, this layer handles decision-making, planning, and memory. It allows agents to break down high-level goals into actionable steps and adjust behavior based on feedback and learning.
3. **Action Interface:** This module connects the agent to tools, APIs, or physical systems. It enables the execution of tasks such as querying databases, calling external services, or initiating real-world actions.

Modern agentic systems often use orchestration frameworks to manage complex workflows, such as LangGraph or other graph-based models, allowing dynamic routing between reasoning steps, memory updates, and tool invocations.

Frameworks

Using a framework allows us to easily use standard patterns such as tools, memory, and components and connect them by calling certain parameters[7]. This 3-set Venn diagram (Figure 2.1) presents the top three trending agentic AI frameworks in 2024, along with their key differentiators.

We selected **LangGraph** as our project's workflow engine. LangGraph is tailored for graph-based state management and workflow control, and it is particularly well-suited for building complex, stateful applications such as our AI chatbot. LangGraph's architecture enables fine-grained control

Table 1.1: Comparison between AutoGen, CrewAI, and LangGraph[1]

Category	**LangGraph**	**CrewAI**	**AutoGen**
Developer Experience	High flexibility, steep learning curve. Limited pre-built components	Easy to use, fast setup. Clear object structure (Agent, Crew, Task). Good documentation.	Moderate ease of use. No built-in DAG support.
Agent Capabilities	Highly customizable. Supports complex hierarchies. DAG workflow support. Strong task execution and tool usage.	Stable task planning and execution. Shared memory collaboration. Automatic agent coordination.	Strong multi-agent collaboration Manual orchestration required Extensible workflows Advanced tooling support
Enterprise Features	Requires custom development.	Limited features in free version. Advanced security (paid).	Built-in security features. Enterprise-grade logging.
Strengths	Flexible customization. Complex behavior support. LangChain ecosystem.	Team collaboration focus. Non-technical configuration. Quick prototyping.	High extensibility. Complex workflow support. Strong memory handling.
Weaknesses	Hard to learn. Component development needed.	Limited extensibility. Debugging difficulties. Complex system limitations.	Requires familiarity with the ecosystem, Initial setup complexity.
Best Use Cases	Complex agent systems with custom interaction and tool usage patterns. Detailed control scenarios.	Rapid simple app development. Non-technical user scenarios.	Advanced multi-agent systems. Enterprisegrade applications with robust security and logging.

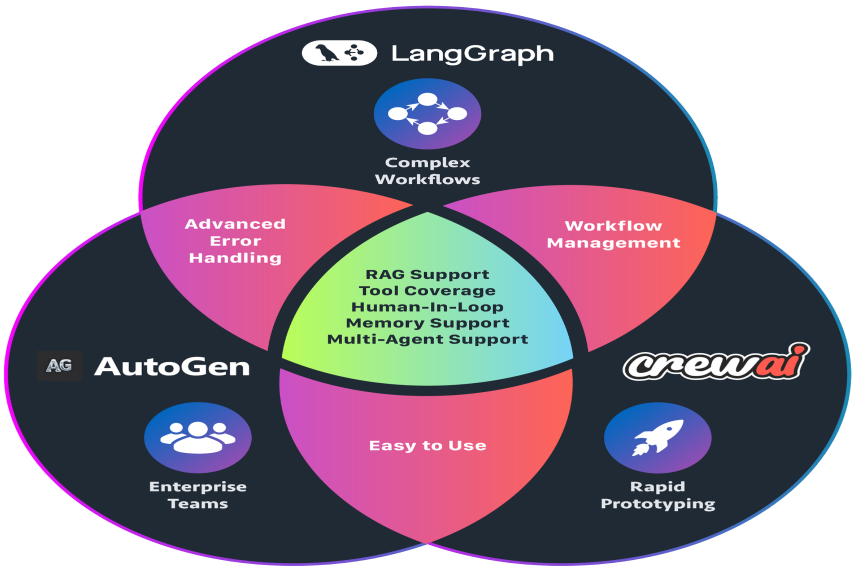


Figure 1.4: langgraph-vs-autogen-vs-crewai[12]

over agent interactions and facilitates dynamic, multi-turn dialogue that captures the real-world intricacies of the RNE registration process. One of the key strengths of LangGraph is that it effectively models both agent interactions and workflow paths, significantly contributing to the design, debugging, and optimization of our chatbot's logic. LangGraph facilitates the development of an intelligent and interactive assistant that offers users a cohesive and guided name verification process.

LanGraph[12]:

- Focuses on graph-based state management and workflow control.
- Best for complex, stateful applications requiring precise control over agent interactions.
- Strongest visualization of agent relationships and workflow paths.

Programming Languages

While Python has traditionally been the dominant language for AI and ML development due to its extensive libraries and frameworks, Typescript (a superset of JavaScript) has emerged as a strong contender for building scalable, maintainable, and efficient AI/ML applications.[7]

Table 1.2: Comparison of Python and TypeScript in Agentic AI Development

Criteria	**Python**	**TypeScript**
Popularity in AI/ML	Dominant language in ML/AI	Less popular for core ML tasks, but growing in AI orchestration and agentic frontend/backend tooling
Real-Time UX Integration	Requires extra effort to interface with UIs (e.g., via Flask/FastAPI + JS frontend).	Seamlessly integrated with frontend tools like React, Next.js, enabling real-time user interaction
Tooling Ecosystem	Excellent for data science; Jupyter, Hugging Face Transformers, Scikit-learn .	Excellent for web dev; VS Code, type-safe APIs, modular bundlers, real-time SDKs (e.g., Supabase, Firebase) [?, ?].
Performance in Deployment	Slower for real-time interactions unless optimized (e.g., asyncio, FastAPI) [?].	Better suited for event-driven architectures and WebSocket-based flows in full-stack apps [?].
Community Support	Mature AI-focused community, extensive documentation.	Rapidly growing community in AI-powered web tools (e.g., Vercel AI SDK, Kaiban.js) [?, ?].
Learning Curve	Simpler syntax, lower barrier for beginners in AI.	Requires understanding of modern JavaScript.
Use Case Fit (Agentic AI)	Ideal for backend-heavy AI agents with intensive NLP, data processing.	Ideal for conversational UIs, full-stack assistants, and integrating agents into web apps [?, ?].
Execution Speed (e.g., Vite benchmark)	Python tends to be slower, with latency between 10–100ms in API calls depending on setup [?].	TypeScript (with Vite/Node.js) often achieves cold start under 1s, hot reload under 100ms [?].
Ecosystem	Rich AI/ML libraries (TensorFlow, PyTorch, LangChain) [?, ?]	Growing AI ecosystem (TensorFlow.js, LangChain.js) [?, ?]
Performance	Slower (optimized via C extensions)	Faster in browser/Node.js (V8) [?]

For our project, we chose TypeScript due to its strong alignment with full-stack application development, especially when working with frontend frameworks like React. Compared to Python, which excels in AI and backend-focused tasks,TypeScript offers better tooling, and early error detection, which enhance code quality and development speed for UI-centric and user-facing applications. For teams aiming to deliver robust web interfaces and full-stack solutions efficiently, TypeScript is the more practical and productive option.

our project focuses more on integrated, user-facing applications rather than backend-heavy agent construction. Therefore, TypeScript is the ideal choice for our needs, enabling rapid development and tight integration between the interface and underlying logic.

1.8.4 LLM Improvement

Large Language Models (LLMs) can be enhanced post-training through various techniques to better adapt them to specific tasks, domains, or user preferences. Two of the most common methods for improving LLMs are fine-tuning and Reinforcement Learning from Human Feedback (RLHF).

Fine-tuning

Fine-tuning enhances the capabilities of pre-trained LLMs by training them on a custom dataset tailored to specific tasks or domains. This approach builds upon few-shot learning, which uses limited examples within the prompt, by enabling the model to learn from many more examples than can fit into a single prompt. As a result, fine-tuning delivers higher-quality and more consistent results across a variety of use cases.

Benefits of fine-tuning include:

Improved output quality compared to prompt-only methods.

Training on larger example sets, beyond the token limit of a single prompt.

Token savings, as fewer examples are needed at inference time.

Lower latency, due to shorter, simpler prompts.

Fine-tuning is especially useful when you need a model to closely follow specific formats, handle domain-specific language, or perform well on custom tasks with consistent accuracy.

Reinforcement Learning and RLHF

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. The goal is to learn a policy that maximizes cumulative reward over time. In the context of LLMs, RL can be applied to shape the model's output behavior by treating its responses as actions and optimizing them based on some reward signal.

Reinforcement Learning from Human Feedback (RLHF) builds upon standard RL by using human feedback as the reward signal. After pre-training and optionally fine-tuning the model, human evaluators are asked to rank or score different model outputs. This feedback is then used to train a reward model, which guides the reinforcement learning process. RLHF is particularly useful for:

- Aligning model behavior with human values, ethics, and intent.

- Improving subjective qualities such as helpfulness, coherence, or tone.
- Reducing harmful or undesirable outputs in open-ended generation.

RLHF involves a three-stage process:

Supervised Fine-Tuning (SFT): The model is initially fine-tuned on a high-quality dataset to perform the target task.

Reward Model (RM) Training: A preference dataset is created by generating multiple outputs for a single prompt and asking humans to rank or select between them. This feedback is used to train a Reward Model that scores the quality of responses.

Reinforcement Learning (RL): The model's behavior is updated through reinforcement learning using the feedback from the RM, gradually aligning it with human expectations—a process known as alignment.

While RLHF has shown success, newer methods have emerged that aim to simplify or improve upon this process:

Direct Preference Optimization (DPO) DPO eliminates the need for a separate Reward Model and the RL step by directly fine-tuning the model using a specially designed loss function. This loss function leverages the idea that language models can implicitly act as reward models. The approach simplifies the pipeline and reduces computational costs without compromising performance.

Kahneman-Tversky Optimization(KTO) Inspired by Prospect Theory—which suggests people are more sensitive to losses than gains—KTO emphasizes penalizing negative (disliked) responses more heavily during training. Unlike RLHF, KTO doesn't require paired Chosen/Rejected outputs; a single response labeled as good or bad is sufficient. This innovation simplifies data collection using basic feedback mechanisms like thumbUp or thumbDown, making it highly practical for real-world deployment.

These alternative methods—DPO, IPO, and KTO—represent important advances in aligning LLMs with human preferences, each offering different trade-offs in complexity, performance, and data requirements. Their effectiveness often depends on the specific dataset and model used, and frameworks like Hugging Face's trl have started supporting all three techniques for broader experimentation.

1.9 Conclusion

In summary, this chapter outlined the background and motivation behind the project. We explored the current challenges faced by entrepreneurs during the company registration process in Tunisia and identified the limitations of the existing RNE system. The chapter concluded by presenting the proposed AI-based chatbot solution and the foundational technologies and concepts that support its implementation.

Chapter 2

State of the Art

2.1 Comparative Analysis

In the fast-changing agentic artificial intelligence environment, a variety of tools, frameworks, and programming paradigms has proliferated. This comparative survey reviews important players in the ecosystem—from workflow engines and programming languages to base language models—pointing out their strong points, applications, and conduciveness to diverse development environments.

2.1.1 Workflow Engine

Using a framework allows us to easily use standard patterns such as tools, memory, and components and connect them by calling certain parameters[7]. This 3-set Venn diagram (Figure 2.1) presents the top three trending agentic AI frameworks in 2024, along with their key differentiators.

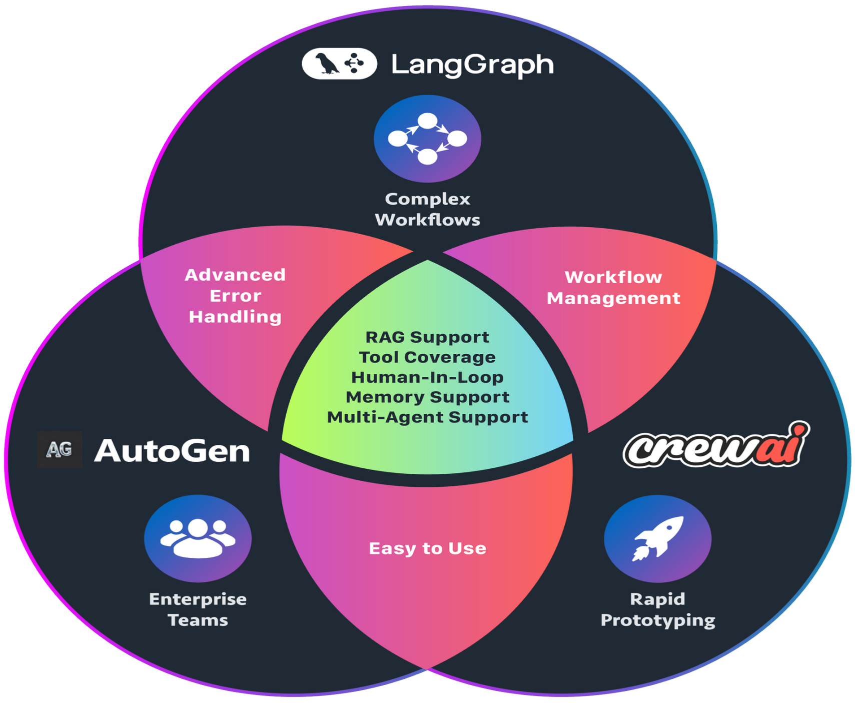


Figure 2.1: langgraph-vs-autogen-vs-crewai[12]

- **LanGraph[12]:**

  Focuses on graph-based state management and workflow control

  Best for complex, stateful applications requiring precise control over agent interactions

  Strongest visualization of agent relationships and workflow paths

- **AutoGen[12]:**

  Most mature and production-ready

  Excels at code execution and sandbox environments

  Strong error handling and reliability features

  Best documentation and community support

- **CrewAI[12]:**

  Prioritizes developer experience and ease of use

  Built-in project structure and YAML configuration

  Strong testing and CLI tools

  Best for rapid prototyping and getting started quickly

2.1.2 Python vs. TypeScript in Agentic AI Development

While Python has traditionally been the dominant language for AI and ML development due to its extensive libraries and frameworks, Typescript (a superset of JavaScript) has emerged as a strong contender for building scalable, maintainable, and efficient AI/ML applications.[7]

The choice between TypeScript and Python depends largely on the specific requirements of the project. TypeScript is well-suited for rapid full-stack application development, particularly when using frameworks like React, as it allows developers to maintain a streamlined stack without introducing additional technologies. In contrast, Python is more appropriate for building complex, modular AI agents due to its extensive support for AI and machine learning libraries and frameworks. Therefore, Python is generally preferred for backend-heavy, agent-centric applications, while TypeScript is ideal for full-stack development with a strong emphasis on frontend integration.[7].

2.1.3 LLM

The evolution of AI language models has set new standards, especially in the coding and programming landscape. Leading the charge are DeepSeek-V3, GPT-4o, and Llama 3.3 70B, each offering unique advantages [14]. The evolution of AI language models has set new standards, especially in the coding and programming landscape. Leading the charge are DeepSeek-V3, GPT-4o, and Llama 3.3 70B, each offering unique advantages [14]. The feature comparison in Table 2.2 and the benchmark results in Table 2.3 provide a detailed view of their strengths and trade-offs

2.1.4 ORM

Drizzle outdoes Prisma by a large margin in terms of pure performance.Figure **??** shows a performance benchmark between Drizzle and Prisma. [5].

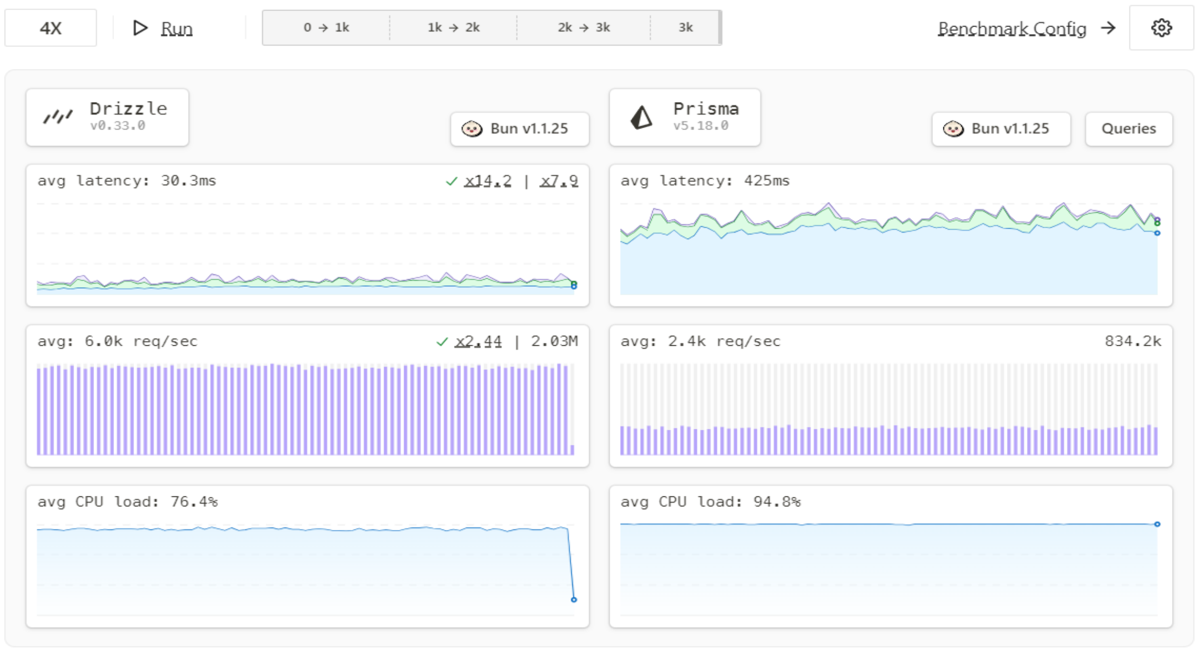


Figure 2.2: drizzle vs prisma [5]

2.1.5 Database

Both Firebase and Supabase provide robust solutions for software development services. They share common features such as real-time capabilities, built-in authentication and storage, automatic API generation, and easy deployment. However, when deciding between Firebase and Supabase, it is crucial to assess the specific project requirements [2]. Table 2.4 compares Firebase and Supabase across several critical dimensions including database support, hosting, and API capabilities.

2.2 Methodology

This section outlines the design principles, system architecture, and technologies we utilized to realize our multi-agent system. we aim to create a scalable, modular system where agents cooperate efficiently to achieve complex tasks.

2.2.1 System Overview

2.2.2 Technologies Used

LangGraph

LangGraph was chosen as the foundation of our system because it allows for flexible and transparent orchestration of language agents in a graph-based workflow. Unlike traditional agent frameworks, LangGraph enables persistent state management, human-in-the-loop operations, and real-time streaming, which are crucial for building dynamic, explainable systems.
Persistence:

LangGraph has a persistence layer, which offers a number of benefits:

Memory:LangGraph persists arbitrary aspects of your application's state, supporting memory of conversations and other updates within and across user interactions,

Human-in-the-loop: Because state is checkpointed, execution can be interrupted and resumed, allowing for decisions, validation, and corrections via human input.

Streaming: LangGraph also provides support for streaming workflow / agent state to the user (or developer) over the course of execution. LangGraph supports streaming of both events (such as feedback from a tool call) and tokens from LLM calls embedded in an application.

Debugging and Deployment: LangGraph provides an easy onramp for testing, debugging, and deploying applications via LangGraph Platform. This includes Studio, an IDE that enables visualization, interaction, and debugging of workflows or agents. This also includes numerous options for deployment.

- **TypeScript (LangGraph)**: Used to implement and orchestrate the LangGraph logic.
- **Python**: Used for fine-tuning the base language model to improve task-specific accuracy.

- **GPT-4o**: The primary language model used for natural language understanding and generation.
- **Supabase**: Serves as the backend-as-a-service.
- **Drizzle ORM**: Provides a type-safe, lightweight way to interact with the Supabase PostgreSQL database from TypeScript.
- **GitHub**: Used for version control, team collaboration, and CI/CD workflows.
- **LangGraph Studio**: Enables interactive testing, visualization, and debugging of the LangGraph workflows.

2.2.3 Architecture

The architecture of our system is based on a modular, multi-agent design built using LangGraph. At its core, each agent is a self-contained unit powered by a language model that handles a specific task within the overall workflow. Instead of relying on a monolithic agent with access to numerous tools which can lead to inefficient decision making and poor scalability we decompose the system into smaller, specialized agents connected through a directed graph structure.

The primary benefits of using multi-agent systems are[9]:

- **Modularity**:Separate agents make it easier to develop, test, and maintain agentic systems.
- **Specialization**: You can create expert agents focused on specific domains, which helps with the overall system performance.
- **Control**: You can explicitly control how agents communicate (as opposed to relying on function calling).

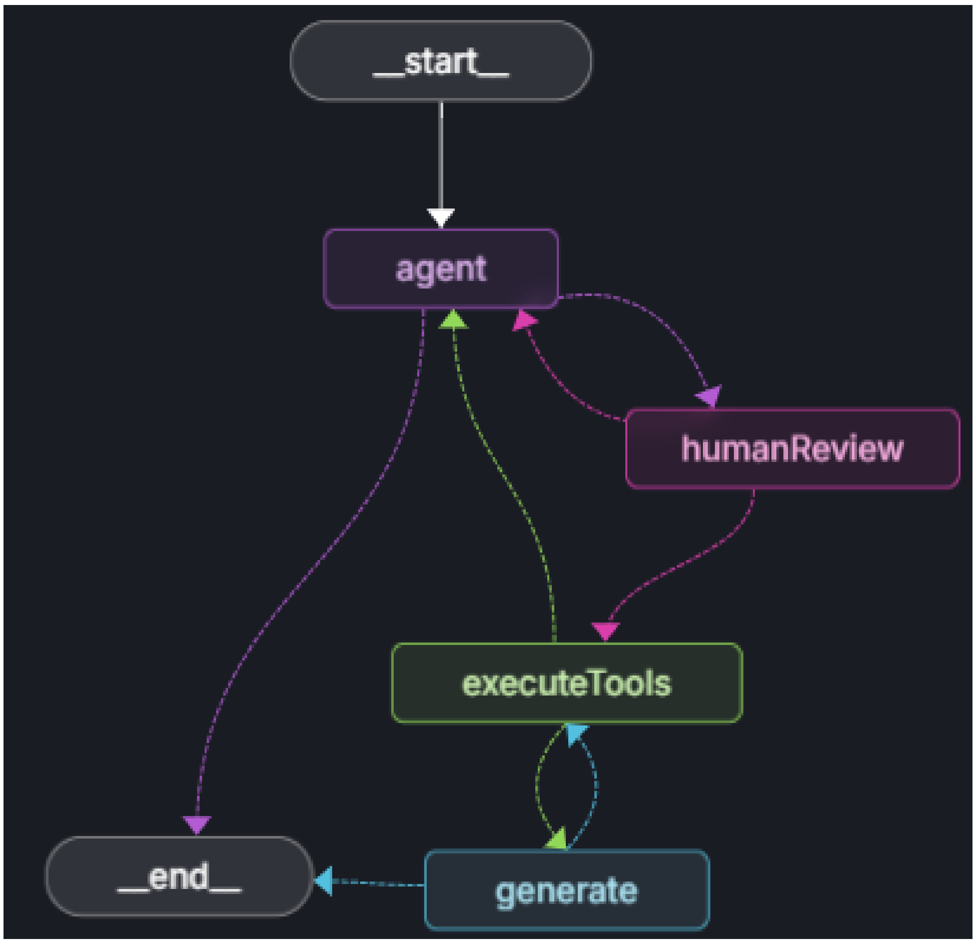


Figure 2.3: final graph

2.3 Conclusion

2.4 POETRY

Table 2.1: Comparison of Python and TypeScript in Agentic AI Development

Criteria	**Python**	**TypeScript**
Popularity in AI/ML	Dominant language in ML/AI	Less popular for core ML tasks, but growing in AI orchestration and agentic frontend/backend tooling
Typing & Safety	Dynamically typed — faster prototyping, but more prone to runtime errors [?].	Statically typed — better tooling, compile-time safety, more maintainable in large codebases [?].
Real-Time UX Integration	Requires extra effort to interface with UIs (e.g., via Flask/FastAPI + JS frontend).	Seamlessly integrated with frontend tools like React, Next.js, enabling real-time user interaction [?].
Tooling Ecosystem	Excellent for data science; Jupyter, Hugging Face Transformers, Scikit-learn [?].	Excellent for web dev; VS Code, type-safe APIs, modular bundlers, real-time SDKs (e.g., Supabase, Firebase) [?, ?].
Performance in Deployment	Slower for real-time interactions unless optimized (e.g., asyncio, FastAPI) [?].	Better suited for event-driven architectures and WebSocket-based flows in full-stack apps [?].
Community Support	Mature AI-focused community, extensive documentation.	Rapidly growing community in AI-powered web tools (e.g., Vercel AI SDK, Kaiban.js) [?, ?].
Learning Curve	Simpler syntax, lower barrier for beginners in AI.	Requires understanding of modern JavaScript and types, but easier for web developers to adopt.
Use Case Fit (Agentic AI)	Ideal for backend-heavy AI agents with intensive NLP, data processing.	Ideal for conversational UIs, full-stack assistants, and integrating agents into web apps [?, ?].
Execution Speed (e.g., Vite benchmark)	Python tends to be slower, with latency between 10–100ms in API calls depending on setup [?].	TypeScript (with Vite/Node.js) often achieves cold start under 1s, hot reload under 100ms [?].
Ecosystem	Rich AI/ML libraries (TensorFlow, PyTorch, LangChain) [?, ?]	Growing AI ecosystem (TensorFlow.js, LangChain.js) [?, ?]
Typing System	Dynamic typing (flexible but runtime errors) [?]	Static typing (better for large-scale code) [?]
Performance	Slower (optimized via C extensions)	Faster in browser/Node.js (V8) [?]

Table 2.2: Feature Comparison of Models[14]

Feature	**DeepSeek-V3**	**GPT-4o**	**LLaMA 3.3 70B**
Open Source	Yes	No	Yes
Release Date	27, December, 2024	6, August, 2024	6, December, 2024
Input Context Window	128K tokens	128K tokens	128K tokens
Maximum Output Tokens	8K tokens	16.4K tokens	2048
API Providers	HuggingFace, DeepSeek	OpenAI, Azure	Fireworks, Together, DeepInfra, Hyperbolic
Supported Input Type	Text	Text, Image, Audio	Text

Table 2.3: Benchmark Results Comparison[14]

Benchmark	**Description**	**DeepSeek-V3**	**GPT-4o**	**LLaMA 3.3 70B**
MMLU	Massive Multitask Language Understanding - Test knowledge across 57 subjects including maths, history, law and more	88.5%	88.7%	88.5%
MMLU-Pro	A more robust MMLU benchmark with more complex reasoning focused questions and reduced prompt sensitivity	75.9%	74.68%	75.9%
MMMU	Massive Multitask Multimodal Understanding: Text understanding across text, audio, images and videos	Not available	69.1%	Not available
HellaSwag	A challenging sentence completion benchmark	88.9%	Not available	Not available
HumanEval	Evaluates code generation and problem solving capabilities	82.6%	90.2%	88.4%
MATH	Tests mathematical problem solving abilities across various difficulty levels	61.6%	75.9%	77%

	Firebase	**Supabase**
Database Support	Offers NoSQL databases	Supports PostgreSQL databases
Deployment	Hosted on Google Cloud	Hosted on Amazon Web Services (AWS)
API Generation	Automatically generates APIs using REST or GraphQL	Automatically generates REST APIs
Hosting Options	Offers on-premises hosting options through Cloud Functions for Firebase	Currently limited to cloud hosting and does not offer on-premises hosting
Community	Established platform with a larger and active community	Growing community with available documentation, tutorials, and articles, but smaller compared to Firebase
Open-Source	Closed-source platform owned by Google, limiting customization and modification	Open-source platform, allowing users to view, modify, and contribute to the source code, offering transparency and flexibility

Table 2.4: Comparison between Firebase and Supabase [2]

Chapter 3

Requirements Specification

3.1 Introduction

This chapter defines the core requirements of Legalyz Chatbot System, detailing its main features, involved actors, and expected behavior through use case and activity diagrams.

3.2 Functional Requirements Identification

The system must provide the following core functionalities:

- **Name Input Handling:** The system must accept a proposed denomination in either Arabic or Latin script.
- **Name Validation:** The system validates that the proposed name complies with the official rules defined by the Registre National des Entreprises (RNE).
- **RNE Search:** The system must perform a structured search within the RNE database to detect potential conflicts with existing names.
- **Web Search Integration:** In addition to the RNE, the system must optionally perform web searches to identify informal or unregistered use of similar names.
- **Similarity Analysis:** The system analyzes search results to identify identical or highly similar names that could pose conflicts or confusion.
- **Name Suggestions:** Based on detected conflicts or rules violations, the system must suggest alternative denominations that comply with the RNE guidelines.
- **Session Context Management:** The system maintains session context to support multi-step refinement and interaction during the naming process.

3.3 Non-Functional Requirements Identification

The system must adhere to the following non-functional requirements to ensure a reliable, secure, and user-friendly experience:

- **Performance**: The system should return results within a few seconds under normal load conditions to ensure a smooth user experience.

- **Scalability**: The system architecture must be scalable to accommodate increasing numbers of users, legal cases, and document uploads without performance degradation.
- **Security**: The system must protect sensitive data by ensuring users' information is stored safely and only accessible by authorized users.
- **Availability**: The system should be available 99.9% of the time, including during maintenance windows, to support continuous user access.
- **Maintainability**: The system must follow modular design principles to facilitate updates, bug fixes, and feature enhancements with minimal downtime.
- **Portability**: The application should be platform-independent and compatible with major web browsers and mobile devices (responsive design).

3.4 Actors

In this section, we define the key actors involved in the system and describe their roles in more detail. Each actor plays a specific part in ensuring that the system functions as intended.

- **User**:
  - **Role**: The primary external actor who interacts with the system who is usually an entrepreneur.
  - **Description**: The user can be anyone who accesses the system, such as a customer, client, or other stakeholders. They submit requests (e.g., queries or data like company names) and receive responses from the system. The user expects efficient, accurate, and timely interaction with the system.
  - **Example**: A user may ask the system, ”What are similar company names to ABC Corp?” The system processes this request and provides relevant results.
- **LLM (Large Language Model)**:
  - **Role**: A model that processes user input and generates appropriate responses.

  - **Description**: The LLM interprets natural language input from users and generates contextually relevant responses, handling tasks such as answering questions and invoking the specific ressources for that.
  - **Example**: A user asks, ”What are similar companies to ABC Corp?” The LLM processes the input and returns a list of similar companies.

- **RNE Engine**:
  - **Role**: A backend engine that handles the search for similar company names to the user's input.
  - **Description**: The RNE Engine is responsible for searching a predefined database or external resources to find companies whose names are similar to the one entered by the user. It helps the system provide more accurate and relevant results by using algorithms to compare name similarities.
  - **Example**: The RNE Engine processes a user input like ”ABC Corp” and returns a list of companies with similar names, such as ”ABC Technologies” or ”ABC Enterprises.”

- **Web Engine**:
  - **Role**: A backend engine that retrieves data from the web to enhance the chatbot's responses.
  - **Description**: The Web Engine is responsible for querying external web sources (e.g., public databases, websites, or APIs) to retrieve information that enriches the LLM's response. This helps provide up-to-date information and detailed answers.
  - **Example**: If the LLM needs to provide data about a company, the Web Engine can fetch the latest details from a business database or an official website to give accurate and current information.

3.5 Use Case Diagram

The use case diagram in Figure 3.1 illustrates the interactions between the User and the *Legalyz Chatbot System*, outlining its main functionalities for company name validation and legal analysis. The chatbot integrates

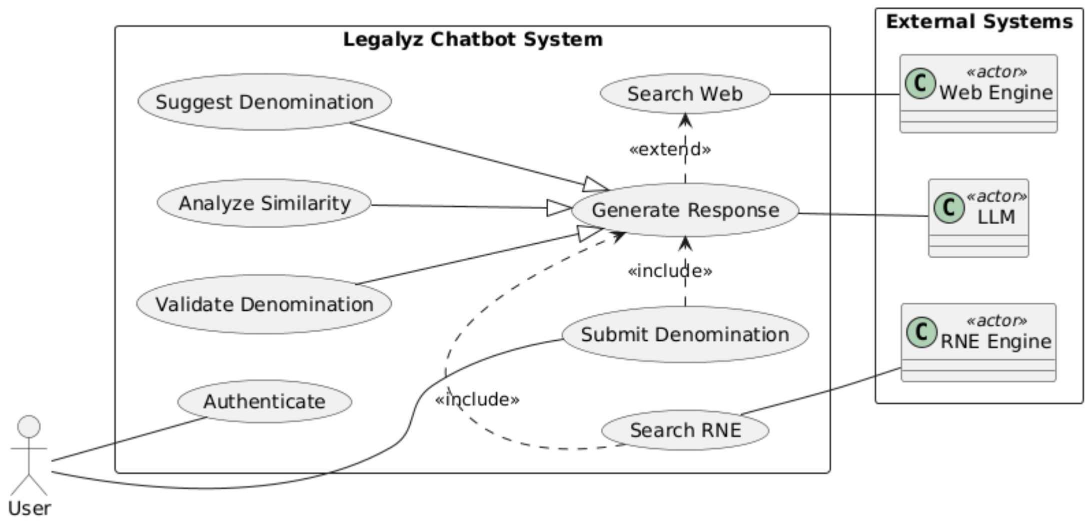


Figure 3.1: Legalyz Chatbot Use Case Diagram

artificial intelligence with external data sources to offer intelligent naming support.

- **Authenticate**: The user must authenticate before using the system's features, ensuring security and traceability.
- **Submit Denomination**: Allows the user to formally submit a selected and validated company name for registration.
- **Generate Response**: This is the core function of the chatbot. It synthesizes information from internal modules and external data sources to deliver tailored naming guidance. As part of the response generation process, the system dynamically produces:
  - **Denomination Validation**: A check of the name's legal availability and compliance with the regulations provided by the RNE.
  - **Similarity Analysis**: An evaluation of the resemblance between the proposed name and existing registered names.
  - **Suggested Denominations**: If the proposed name is already in use or invalid, the system automatically suggests alternative names that are legally acceptable and semantically similar.
- **Search RNE**: Interrogates the National Business Registry (RNE) to verify name availability and detect potential conflicts.
- **Search Web**: Conducts an extensive web search to detect potential trademark conflicts or naming issues that may not appear in official

registries, including names that are still pending registration but already associated with social media or online presence.

3.6 Activity Diagram

The activity diagram presented in Figure 3.2 illustrates the typical workflow of the Agentic AI Chatbot System during the company name validation process. It captures the most common user interactions, including name submission, validation, confirmation, and handling of rejections. This diagram serves to visualize the dynamic behavior of the system, detailing the sequence of activities and decision points encountered during standard operations. It's important to note that while this diagram focuses on prevalent scenarios, it may not depict all possible edge cases or exceptional conditions. The primary objective is to provide a clear understanding of the system's typical behavior and the flow of activities in response to user inputs.

3.7 Conclusion

With clearly defined requirements and system behavior, this chapter ensures alignment between user needs and technical implementation. It sets the stage for translating specifications into a functional AI-powered chatbot.

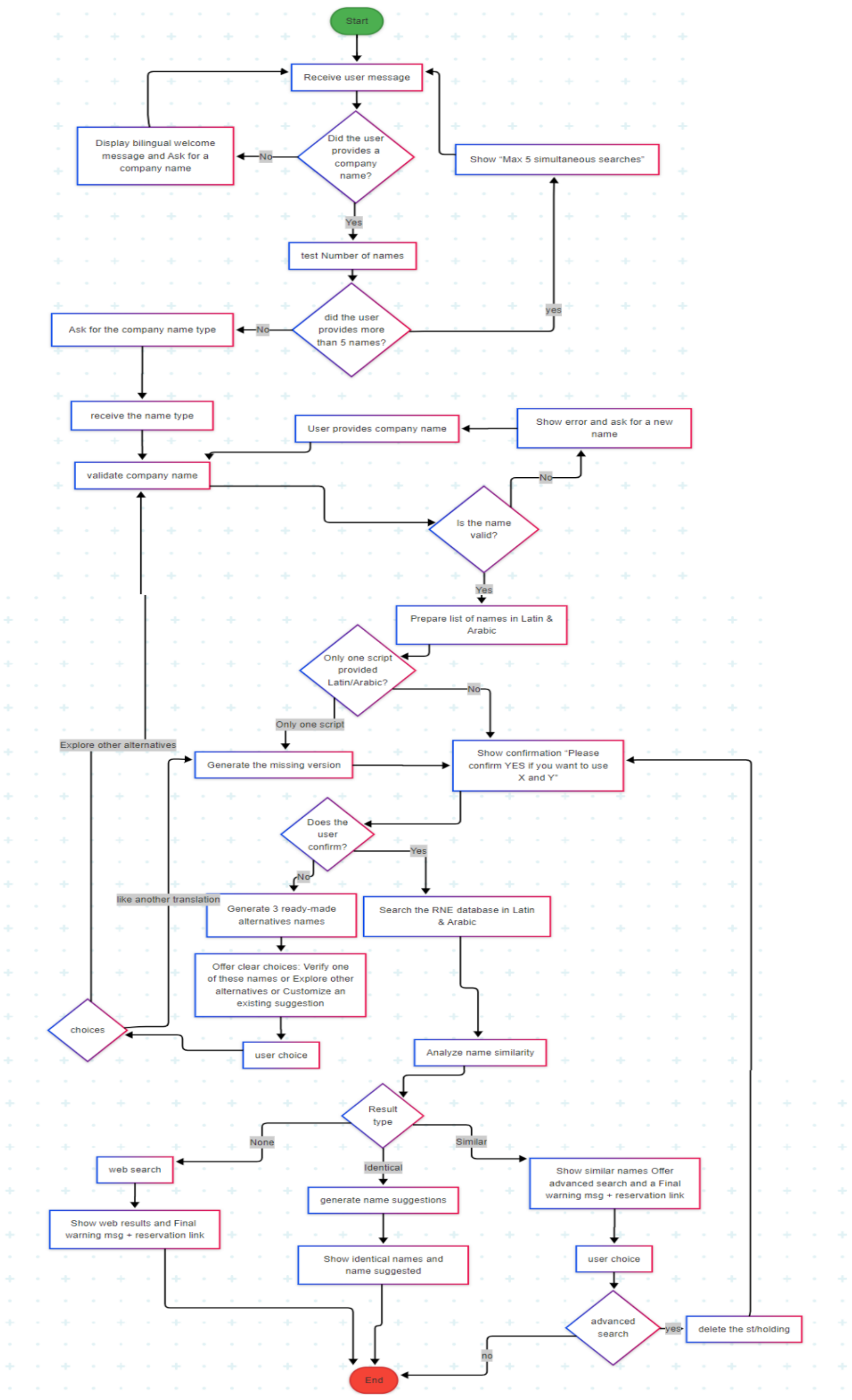

Start
Receive user message
Display bilingual welcome message and Ask for a company name
No
Did the user provides a company name?
Show "Max 5 simultaneous searches"
Yes
test Number of names
Ask for the company name type
No
did the user provides more than 5 names?
yes
receive the name type
User provides company name
Show error and ask for a new name
validate company name
No
Is the name valid?
Yes
Prepare list of names in Latin & Arabic
Only one script provided Latin/Arabic?
No
Only one script
Explore other alternatives
Generate the missing version
Show confirmation "Please confirm YES if you want to use X and Y"
Does the user confirm?
Yes
No
like another translation
Generate 3 ready-made alternatives names
Search the RNE database in Latin & Arabic
Offer clear choices: Verify one of these names or Explore other alternatives or Customize an existing suggestion
choices
user choice
Analyze name similarity
Result type
None
Similar
Identical
web search
generate name suggestions
Show similar names Offer advanced search and a Final warning msg + reservation link
Show web results and Final warning msg + reservation link
Show identical names and name suggested
user choice
advanced search
yes
delete the st/holding
no
End

Figure 3.2: Activity Diagram

Chapter 4

Realization

4.1 Introduction

In this chapter, we share the development process of our project using the Lean mindset with a focus on the iterative cycle of three main versions. Each cycle embodies a plan, build, and learn phase through ongoing experimentation and feedback. This enabled ongoing improvement, hypothesis testing, and adaptive adjustments.

4.2 Iteration 1: Initial Graph Version

This iteration served as the foundation for building a functional and testable prototype of the chatbot's reasoning capabilities.

4.2.1 Objectives

The objective of this first iteration was to design and validate **the core logic** of our LangGraph-based chatbot capable of replicating the basic functionalities of the RNE (Registre National des Entreprises) simulator. This version focused on bilingual company name processing and RNE verification. Key components included input validation, intent detection, tool integration, and user interaction flow management.

4.2.2 Development Process

The initial version consists of five main nodes: ‘agent‘, ‘searchRNE‘, ‘generate‘ and ‘searchWeb‘. The flow begins with the ‘agent‘ node and ends either directly or after one or more intermediate actions based on user input and system decisions. The figure 4.1 illustrates the initial graph structure:

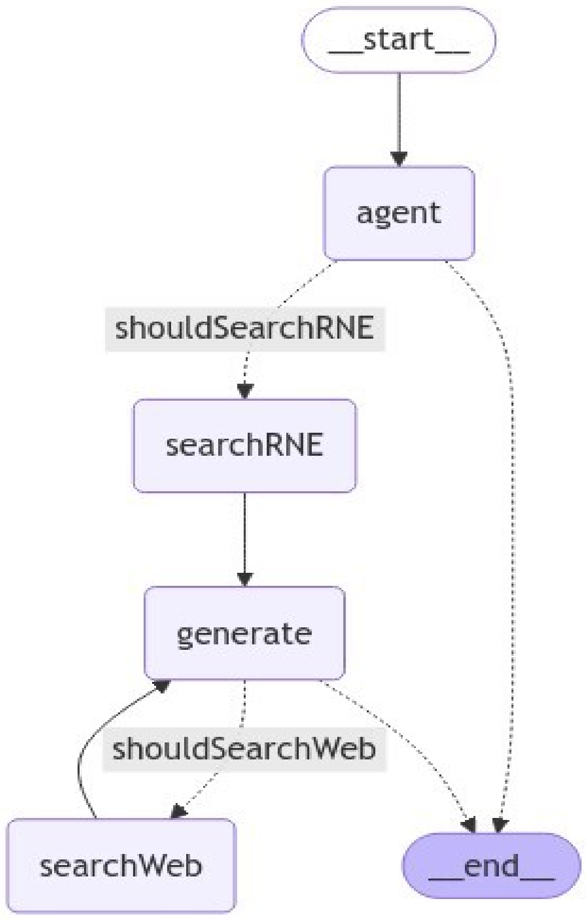


Figure 4.1: initial graph

Agent Node: Input Validation and Control Logic

The **agent** node plays a pivotal role in guiding the user through the input process using the following layered logic:

- **Greeting Detection**: Recognizes user salutations and returns a bilingual welcome message, prompting the user to input a company name.
- **Input Validation Pipeline**: Ensures both Latin and Arabic versions of the company name are present. If one is missing, it is automatically generated. Invalid or ambiguous names are rejected, and users are prompted for corrections.
- **Confirmation and Path Selection**: Prevents mono-language searches by enforcing compliance with RNE protocol. User confirmation is always required before proceeding to either RNE search.

searchRNE Tool Integration

The **searchRNE** node integrates directly with the RNE public API, using a sign-in mechanism and authenticated requests. The tool fetches registered business names based on the input denomination and language.

Internally, this tool is implemented in three steps:

1. Authenticate using a POST request to retrieve a token.
2. Construct a query URL dynamically using the company name and chosen language.
3. Fetch search results with the token and parse them to extract relevant matches.

This enables the chatbot to interact in real time with the RNE database and return structured business name data to downstream nodes.

Generate Node: Result Categorization and Guidance

The **generate** node plays a crucial role in interpreting RNE results and guiding the user accordingly. The categorization logic is as follows:

- **Identical Matches**: If an exact or very similar name exists in the RNE, the user is shown the conflicting entries and prompted to select an alternative name.
- **Similar Matches**: The user is warned about name similarity, which might pose registration issues. They are advised to modify the name or choose another.
- **Available Names**: If no conflicts are detected, the system confirms name availability and offers the option to proceed with a broader web search.

The chatbot always warns the user with a disclaimer like: ”Je ne peux pas vous garantir à 100 ...” followed by a reservation link, complying with RNE guidelines.

Search Web Node (Tavily)

If the user agrees to extend the search scope (‘shouldSearchWeb‘), the graph transitions to the **searchWeb** node. This node uses the Tavily API to explore external sources for company name availability beyond RNE. This serves as a secondary assurance layer for name uniqueness and helps identify potential naming conflicts across the web.

4.2.3 Results

This initial version successfully delivered a functional prototype that establishes the searchRNE integration and core chatbot logic. It demonstrates proper interaction flow, bilingual name handling, and basic decision control using LangGraph.

However, testing (langgraph studio) and client feedback revealed several limitations:

- Heavy Dependence on User Confirmation: The system relies heavily on explicit user confirmations before executing RNE or web searches. This increases interaction length and reduces automation. More autonomous decision-making is needed to reduce query cost and improve user experience.
- Certain logic blocks: like name validation and name suggestion generation, produced inconsistent or unreliable outputs.
- Inaccurate Similarity Sorting: The current approach to identifying and sorting similar names does not always yield relevant or useful results, requiring a refinement of the matching algorithm.
- The Tavily API, while useful for early experimentation, presented challenges related to cost efficiency and response relevance, prompting consideration of alternative solutions in future iterations.

These observations set the stage for the second iteration, which will focus on enhancing reasoning autonomy, refining validation pipelines, and improving result clarity.

4.3 Version 2

4.3.1 Objectives

Improve limitations identified in Iteration 1.

4.3.2 Development and Technologies

Technologies: serper. Enhancements and Development Work:
Adjustments in flow and intent handling.
Introduction of feedback loops and conditional logic.
Improved tool selection and edge configuration.

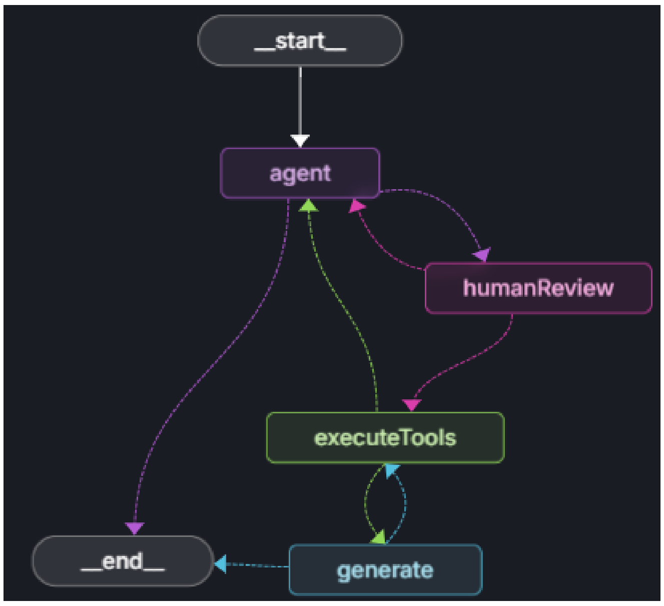


Figure 4.2: final graph

4.3.3 Results and Audit

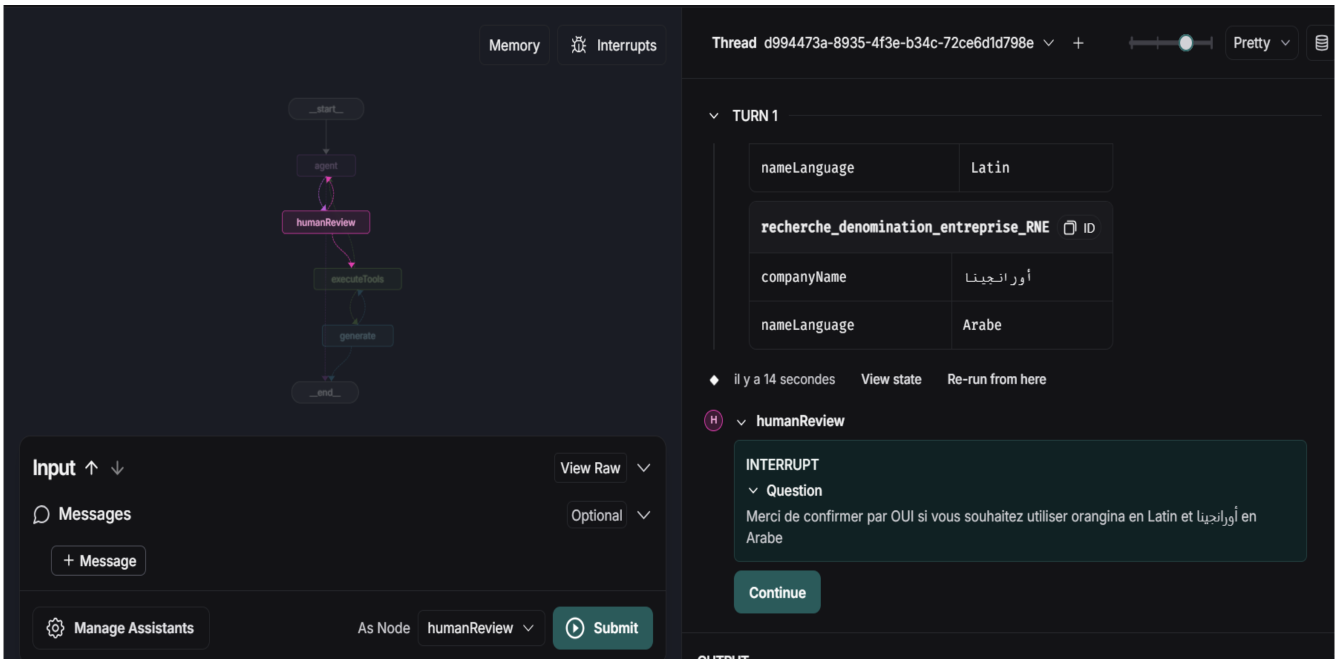
4.4 Version 3

4.4.1 Objectives

4.4.2 Development and Technologies

Human Review Node (HITL)

To modularize human oversight at critical decision points, we introduced a **human-in-the-loop** node dedicated to confirming company names before querying the RNE database. When a tool call to `"searchRNE"` is detected, this node extracts both Latin and Arabic versions of the company name



nameLanguage	Latin

recherche_denomination_entreprise_RNE

companyName	أورانجينا
nameLanguage	Arabe

il y a 14 secondes

humanReview

INTERRUPT

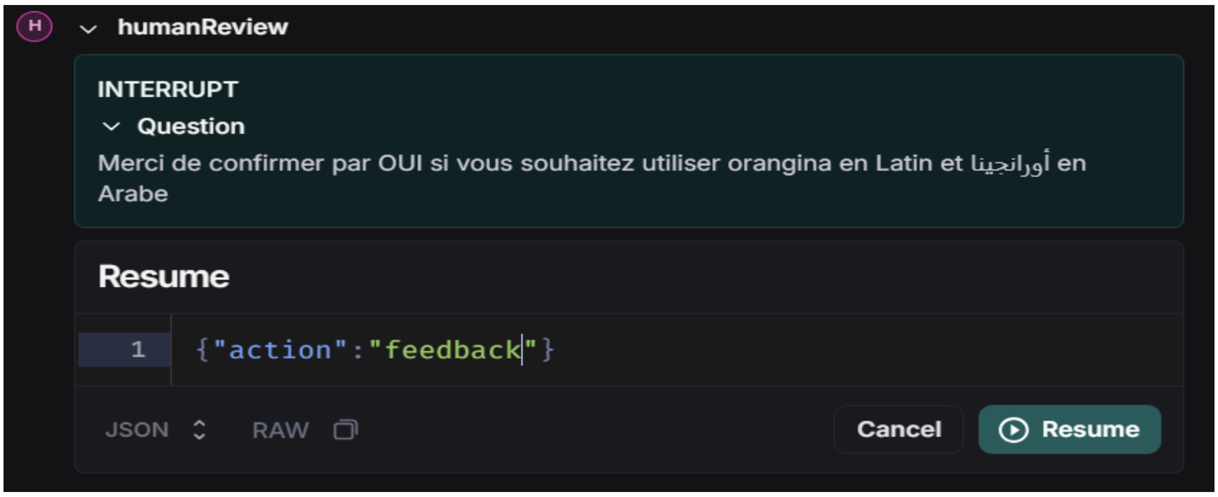
Question

Merci de confirmer par OUI si vous souhaitez utiliser orangina en Latin et أورانجينا en Arabe

Figure 4.3: HITL

and generates a bilingual confirmation prompt (e.g., *"Merci de confirmer par OUI si vous souhaitez utiliser X en Latin et Y en Arabe"*).

If the user declines, the system redirects to the `agent` node, appending an explanatory assistant message and canceling the tool execution. If the user confirms, the graph proceeds to the `executeTools` node.



INTERRUPT

Question

Merci de confirmer par OUI si vous souhaitez utiliser orangina en Latin et أورانجينا en Arabe

```
{"action":"feedback"}
```

Figure 4.4: user declines

4.4.3 Results and Audit

4.5 Conclusion

General Conclusion

Nous espérons que ce document vous a aidé à rédiger votre rapport convenablement.

Ce document vise à aider les étudiants à rédiger des rapports en Latex.

Bibliography

[1] aaronyu. comparison-of-langgraph-crewai-and-autogen, 2025.

[2] Sunil Chavda. Firebase vs. supabase, 2023.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Doanh Do. The five principles of lean, 2017.

[5] drizzle.team. drizzle orm benchmarks, 2025.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[7] Wahyu ikbal maulana. Building ai agent workflows with python typescript, 2025.

[8] Daniel Jurafsky and James H Martin. *Speech and Language Processing.* Pearson, 2021.

[9] LangGraph. Multi-agent systems, 2025.

[10] Christopher D Manning and Hinrich Schütze. Foundations of statistical natural language processing. *MIT Press*, 1999.

[11] masterclass. What is lean methodology?, 2022.

[12] Estevan McCalley. Top 3 trending agentic ai frameworks: Langgraph vs autogen vs crew ai, 2024.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[14] Vipin Vashisth. Deepseek-v3 vs gpt-4o vs llama 3.3 70b – let the best ai model win, 2025.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Annexe 1