PROJECT OVERVIEW

for

Bureaucracy AI

Version 1.0

Prepared by Cappellini Alessio (1693930)
Cataldi Bruno (2026604)
Sirico Giuseppe Gabriele (1810153)

October 29, 2023

Contents

1 Introduction

1.1 Purpose

The objective of this project is to lay the foundations for a new concept of smart assistant in the legal (both civil and criminal) and tax context using Artificial Intelligence. It aims to support professionals and others, in everyday life, dealing with European and national (particularly Italian) bureaucracy.

1.2 Project Scope

In the era in which we live, the bureaucratic complexity that underlies all relationships both in the world of work and outside is well known. For instance, Italy is renowned for the disproportionate quantity of laws in force: from an analysis deriving from "Il sole 24 ore" it is estimated that the total number of Italian laws is 111,000. Too many for anyone to know.

It is really difficult just to think that such a large number of laws can get along with each other without leaving room for quibbles or illogical contradictions. Unfortunately, Italy is one of the worst example but in general Europe and other member states are not immune to this problem.

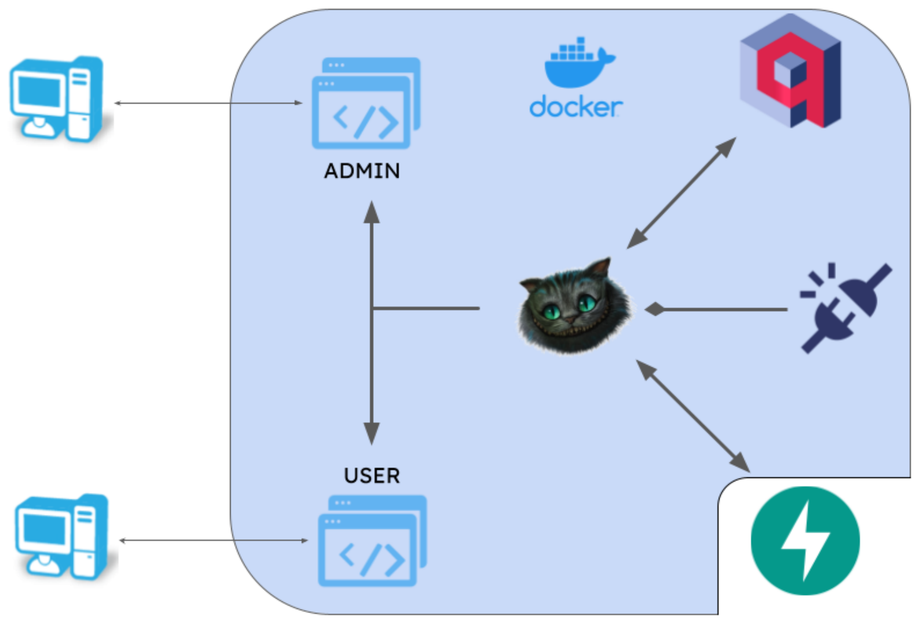
The idea is to lighten the complexity of bureaucracy by making available to people a tool that allows people to draw on an agnostic knowledge of the mechanisms of the bureaucratic machine. This instrument would overcome any linguistic and cultural barrier, especially in countries where there is significant migratory activity, helping all people to understand the laws in force in a particular country or confederation of such.

Thanks to interviews, several applications of our law helper have emerged concerning different scenarios. For instance, the application helps all the generation to encourage new generations to plan on their own without relying to much on their elders. In the same vein, it helps people of all ages to access a domain of knowledge otherwise hard reach and understand. We believe that given the user-friendliness nature of a chatbot, this is the best method to effectively and hastily place near the world of law.

2 Overview

2.1 Architecture

This section will illustrate the anatomy of the ecosystem in which Bureaucracy AI will work.



Broadly speaking, we can divide the ecosystem into two macro components:

1. 2 **Cheshire Cat Environment**
2. 2 **LLAMA2 instance**

The first will be a Docker containers cluster orchestrated with Compose having the following container images:

- **Vector DB**
- **Cheshire Cat Framework**

On the other hand, for the frontend, we want to implement a web application using Fast API framework. All the technologies mentioned so far will be explained in the next section which will give a quick overview of what they are in practice and how they will be used. Obviously, we will not dwell on technologies such as Docker or Kubernetes (and their respective advantages) as they have been thoroughly covered during the lessons.

2.2 Technologies

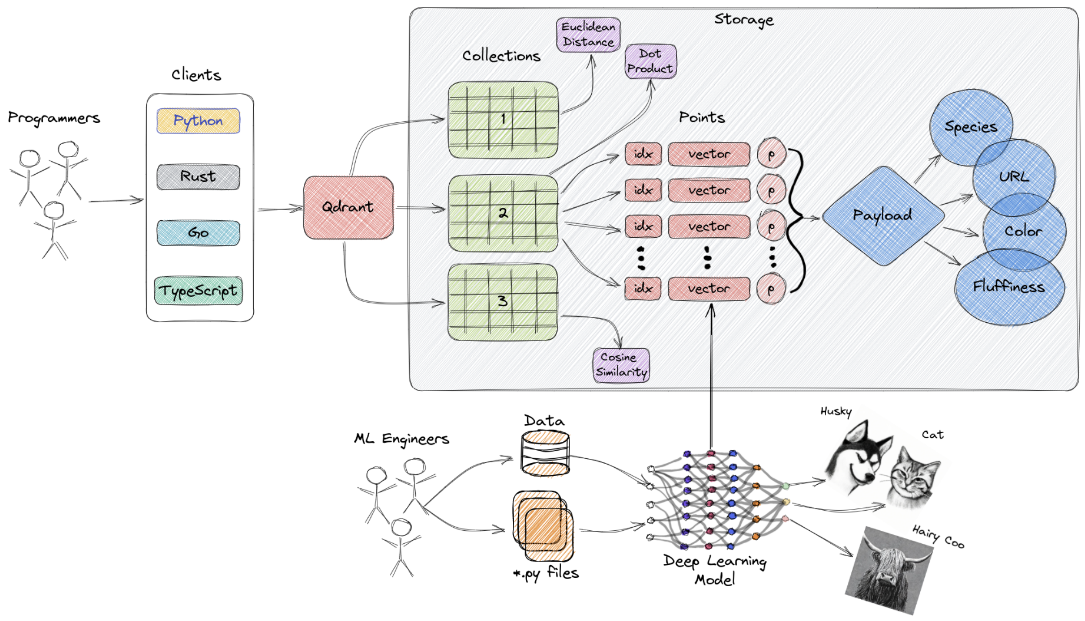
2.2.1 Vector database

Vector databases are a type of database designed to store and query high-dimensional vectors efficiently. In traditional OLTP and OLAP databases (as seen in the image above), data is organized in rows and columns (and these are called Tables), and queries are performed based on the values in those columns. However, in certain applications including image recognition, natural language processing, and recommendation systems, data is often represented as vectors in a high-dimensional space, and these vectors, plus an id and a payload, are the elements we store in something called a Collection a vector database like Qdrant.

A vector in this context is a mathematical representation of an object or data point, where each element of the vector corresponds to a specific feature or attribute of the object. For example, in an image recognition system, a vector could represent an image, with each element of the vector representing a pixel value or a descriptor/characteristic of that pixel. In a music recommendation system, each vector would represent a song, and each element of the vector would represent a characteristic song such as tempo, genre, lyrics, and so on.

Vector databases are optimized for storing and querying these high-dimensional vectors efficiently, and they often using specialized data structures and indexing techniques such as Hierarchical Navigable Small World (HNSW) – which is used to implement Approximate Nearest Neighbors – and Product Quantization, among others. These databases enable fast similarity and semantic search while allowing users to find vectors that are the closest to a given query vector based on some distance metric.

Qdrant



Qdrant “is a vector similarity search engine that provides a production-ready service with a convenient API to store, search, and manage points (i.e. vectors) with an additional payload.

Vector databases are a type of database designed to store and query high-dimensional vectors efficiently. In traditional OLTP and OLAP databases data is organized in rows and columns, and queries are performed based on the values in those columns. However, in certain applications including image recognition, natural language processing, and recommendation systems, data is often represented as vectors in a high-dimensional space, and these vectors, plus an id and a payload, are the elements we store in something called a Collection a vector database.

A vector in this context is a mathematical representation of an object or data point, where each element of the vector corresponds to a specific feature or attribute of the object. For example, in a music recommendation system, each vector would represent a song, and each element of the vector would represent a characteristic song such as tempo, genre, lyrics, and so on.

Vector databases are optimized for storing and querying these high-dimensional vectors efficiently, and they often using specialized data structures and indexing techniques such as Hierarchical Navigable Small World (HNSW) – which is used to implement Approximate Nearest Neighbors – and Product Quantization, among others. These databases enable fast similarity and semantic search while allowing users to find vectors that are the closest to a given query vector based on some distance metric.

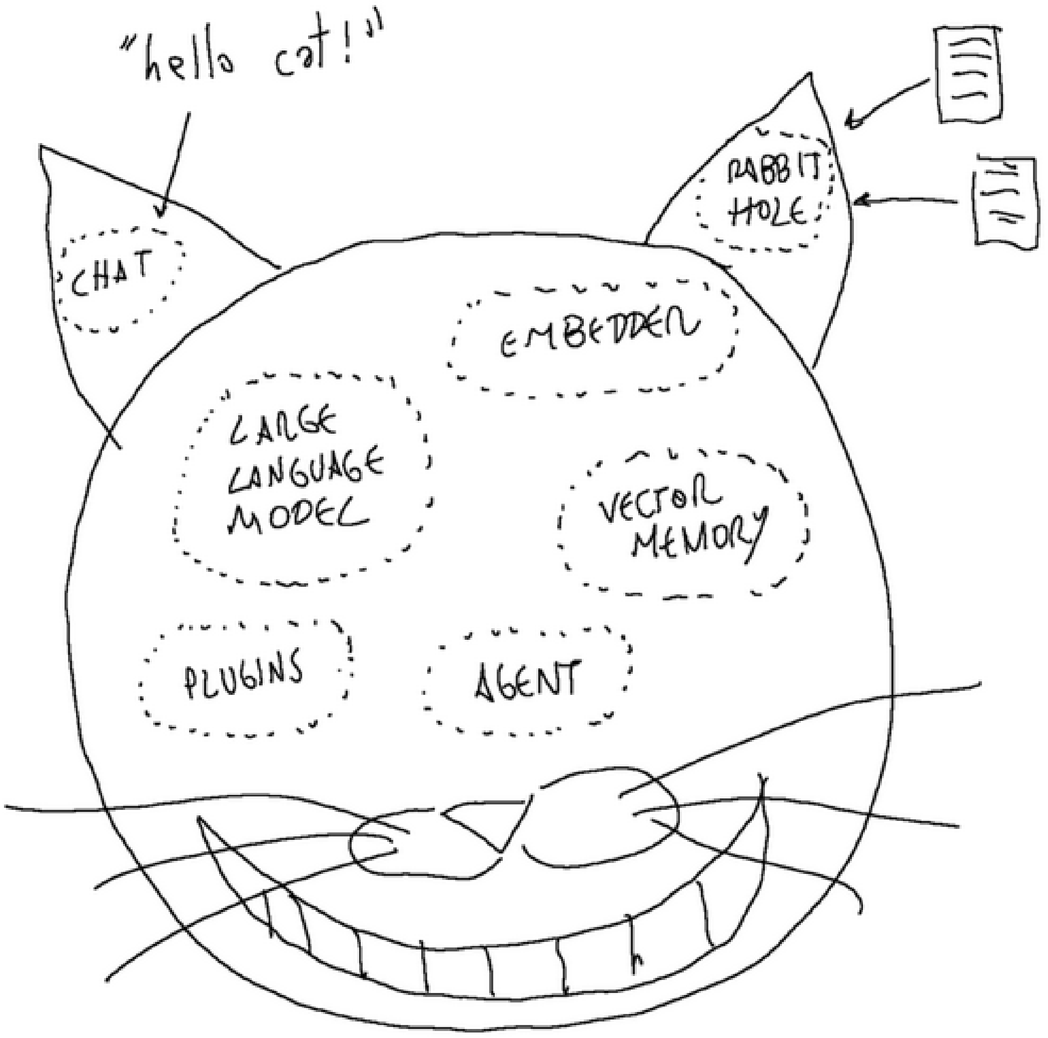
2.2.2 Cheshire Cat

Cheshire Cat is an Italian framework to build custom AIs on top of any language model to assist you in a wide range of tasks. It embeds a long-term memory system to save the user's input locally and answer knowing the context of previous conversations. It also has the ability to digest documents.

The main high level features offered by this framework are the following:

- **Language model agnostic**
- **Extensible via plugins**
- **Can use external tools (APIs, custom python code, other models)**
- **Long term memory**

How it works



The cat is made of many pluggable components to ensure a high degree of customizability. Those are The Chat, the Rabbit Hole, the LLM, the embedder, the vector memory and the Agent. The rabbit hole is responsible for digesting documents, that will then be sent to the vector memory that will remember them along things that the user said in the past. This data is worked on by the embedder and the LLM, which are in turn orchestrated by the Agent to ensure the execution of complex task.

On top of this architecture, also customizable plugins are implementable to fit anyone's needs.

2.2.3 Fast API

Fast API is a high-level Python micro web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. The main high level features offered by this framework are the following:

- **Fast**
- **Secure**
- **Scalable**

Thanks to its versatility we choose to build a small backend in order to create a custom instance of LLaMa.

2.2.4 LLaMA

LLaMA is a family of large language models (LLMs), released by Meta AI. As the name suggest, an LLM is a particularly big artificial neural network that can perform a variety of natural language processing (NLP) tasks. The goal of LLaMA is to provide the user with the capability of:

- Expressing a generic data structure independently of how it's stored.
- Providing generic facilities to map the user-defined data structure into a performant data layout
- Efficient, high throughput copying between different data layouts of the same data structure

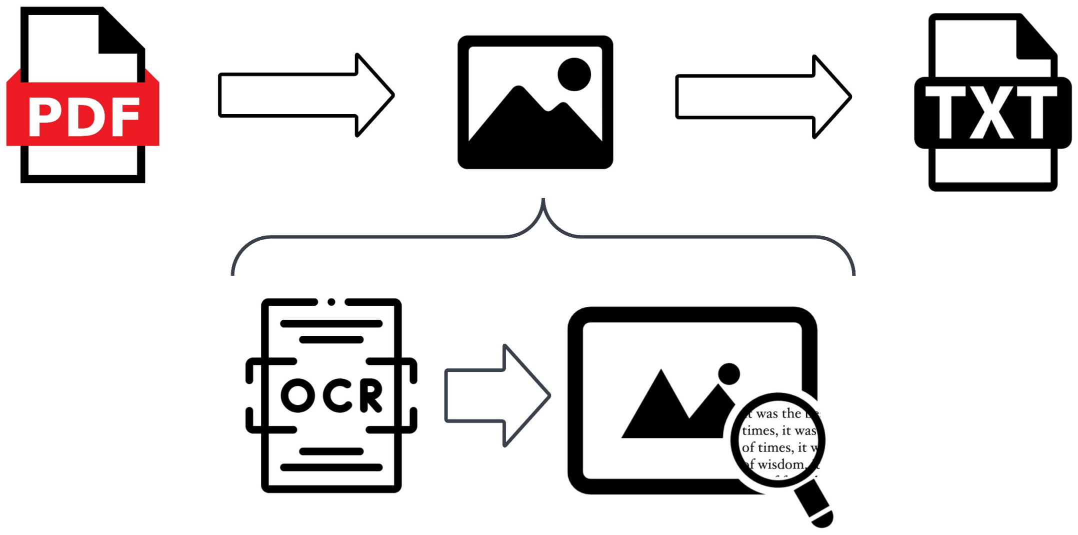
2.2.5 Plugins

These are the additional feature that we add to the cat in order to better fit our needs.

The main custom plugin we created is POCR. Essentially it consists in reading the content of a PDF file in order to extract the text contained in it.

The technique adopted to accomplish this task is called Optical Character Recognition (OCR): it is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text. It is a field of research in pattern recognition, artificial intelligence, and computer vision. This approach works by first pre-processing the image to improve the quality of the text and make it easier to recognize. This may involve tasks such as binarizing the image, noise reduction, and deskewing. Once the image has been pre-processed, the OCR software uses a variety of techniques to identify the individual characters in the image. Some common techniques include:

- Template matching: it compares the image to a database of known character templates. The character with the closest match is chosen.
- Feature extraction: it extracts features from the image, such as the shape of the character, the number of corners, and the distribution of pixels. The features are then used to identify the character.
- Neural networks: they are a type of machine learning algorithm that can be trained to recognize characters. Once trained, the neural network can be used to identify characters in new images.



Once all of the characters in the image have been identified, the OCR software assembles them into words and sentences. The output of the OCR process is a machine-encoded text file that can be edited, searched, and stored.

3 Non-Functional Requirements

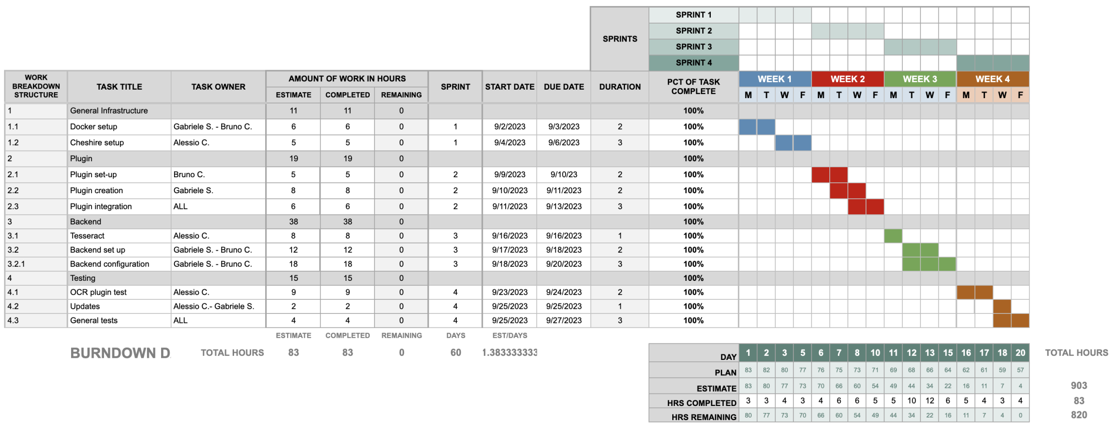
The following are the main non-functional requirements we want to provide:

1. **Availability:** the service provided by the system should be fully working whenever the user requests it.
2. **Efficiency:** Efficiency is the measurable ability to avoid wasting materials, energy, efforts, money, and time while performing a task. In our case the user interact with an easy and familiar system by using the chat in a fast and simple way.
3. **Reliability:** the system should be available during all the user interaction, avoiding errors and restoring the user session in case of disconnection.
4. **Effectiveness:** the user is able to easily achieve his goal by logging into the system, interrogate the chatbot and obtain all the information he needs.
5. **Usability:** the system is simple to understand, fast and easy to learn given the familiarity and immediacy of the chat interface.
6. **Scalability:** the system should be able to handle the requests of many different users at the same time.
7. **Extensibility:** the system could be extended with the addition of new functionalities with the implementation of plugins.
8. **Responsiveness:** it refers to the specific ability of a system or functional unit to complete assigned tasks within a given time. In our case the chatbot in the system should able to understand and carry out users' requests in a timely fashion.
9. **Portability:** the application should be usable and runnable in different platform and operating systems.

4 FP EVALUATION AND COCOMO II

4.1 Function Point

4.1.1 GANTT Chart



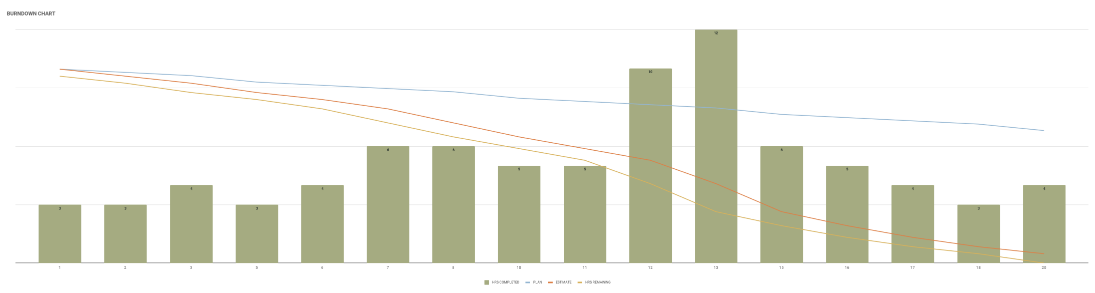
SPRINTS		M	T	W	F	M	T	W	F	M	T	W	F	M	T	W	F
	SPRINT 1	▨	▨	▨	▨												
	SPRINT 2					▨	▨	▨	▨								
	SPRINT 3									▨	▨	▨	▨				
	SPRINT 4													▨	▨	▨	▨

WORK BREAKDOWN STRUCTURE	TASK TITLE	TASK OWNER	AMOUNT OF WORK IN HOURS			SPRINT	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	WEEK 1				WEEK 2				WEEK 3				WEEK 4			
			ESTIMATE	COMPLETED	REMAINING						M	T	W	F	M	T	W	F	M	T	W	F	M	T	W	F
1	General Infrastructure		11	11	0					100%																
1.1	Docker setup	Gabriele S. - Bruno C.	6	6	0	1	9/2/2023	9/3/2023	2	100%	🟦	🟦														
1.2	Cheshire setup	Alessio C.	5	5	0	1	9/4/2023	9/6/2023	3	100%			🟦	🟦												
2	Plugin		19	19	0					100%																
2.1	Plugin set-up	Bruno C.	5	5	0	2	9/9/2023	9/10/23	2	100%					🟥	🟥										
2.2	Plugin creation	Gabriele S.	8	8	0	2	9/10/2023	9/11/2023	2	100%						🟥	🟥									
2.3	Plugin integration	ALL	6	6	0	2	9/11/2023	9/13/2023	3	100%							🟥	🟥								
3	Backend		38	38	0					100%																
3.1	Tesseract	Alessio C.	8	8	0	3	9/16/2023	9/16/2023	1	100%									🟩							
3.2	Backend set up	Gabriele S. - Bruno C.	12	12	0	3	9/17/2023	9/18/2023	2	100%										🟩	🟩					
3.2.1	Backend configuration	Gabriele S. - Bruno C.	18	18	0	3	9/18/2023	9/20/2023	3	100%										🟩	🟩	🟩				
4	Testing		15	15	0					100%																
4.1	OCR plugin test	Alessio C.	9	9	0	4	9/23/2023	9/24/2023	2	100%													🟧	🟧		
4.2	Updates	Alessio C.- Gabriele S.	2	2	0	4	9/25/2023	9/25/2023	1	100%															🟧	
4.3	General tests	ALL	4	4	0	4	9/25/2023	9/27/2023	3	100%															🟧	🟧

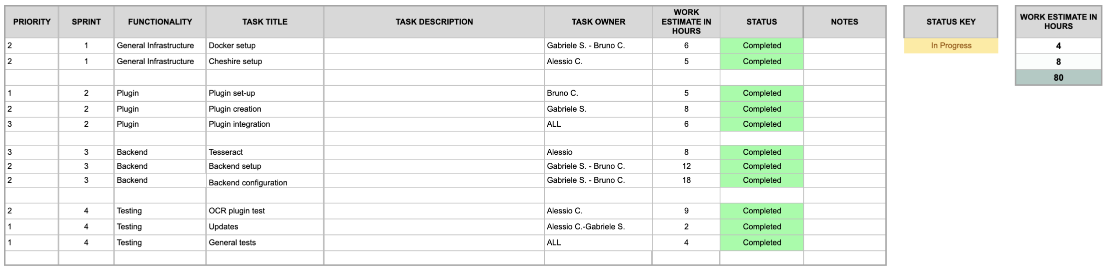
		ESTIMATE	COMPLETED	REMAINING	DAYS	EST/DAYS
BURNDOWN D	TOTAL HOURS	83	83	0	60	1.38333333

DAY	1	2	3	5	6	7	8	10	11	12	13	15	16	17	18	20	TOTAL HOURS
PLAN	83	82	80	77	76	75	73	71	69	68	66	64	62	61	59	57	
ESTIMATE	83	80	77	73	70	66	60	54	49	44	34	22	16	11	7	4	903
HRS COMPLETED	3	3	4	3	4	6	6	5	5	10	12	6	5	4	3	4	83
HRS REMAINING	80	77	73	70	66	60	54	49	44	34	22	16	11	7	4	0	820

4.1.2 Burndown Chart



4.1.3 Release backlog

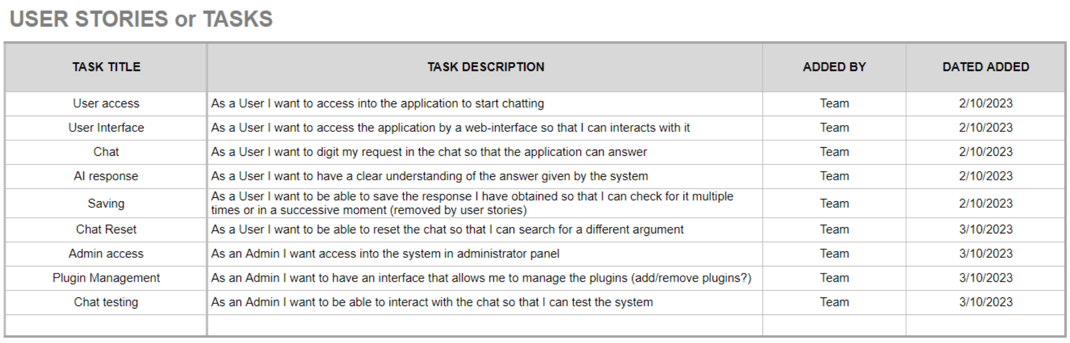


PRIORITY	SPRINT	FUNCTIONALITY	TASK TITLE	TASK DESCRIPTION	TASK OWNER	WORK ESTIMATE IN HOURS	STATUS	NOTES
2	1	General Infrastructure	Docker setup		Gabriele S. - Bruno C.	6	Completed	
2	1	General Infrastructure	Cheshire setup		Alessio C.	5	Completed	
1	2	Plugin	Plugin set-up		Bruno C.	5	Completed	
2	2	Plugin	Plugin creation		Gabriele S.	8	Completed	
3	2	Plugin	Plugin integration		ALL	6	Completed	
3	3	Backend	Tesseract		Alessio	8	Completed	
2	3	Backend	Backend setup		Gabriele S. - Bruno C.	12	Completed	
2	3	Backend	Backend configuration		Gabriele S. - Bruno C.	18	Completed	
2	4	Testing	OCR plugin test		Alessio C.	9	Completed	
1	4	Testing	Updates		Alessio C.-Gabriele S.	2	Completed	
1	4	Testing	General tests		ALL	4	Completed	

STATUS KEY
In Progress

WORK ESTIMATE IN HOURS
4
8
80

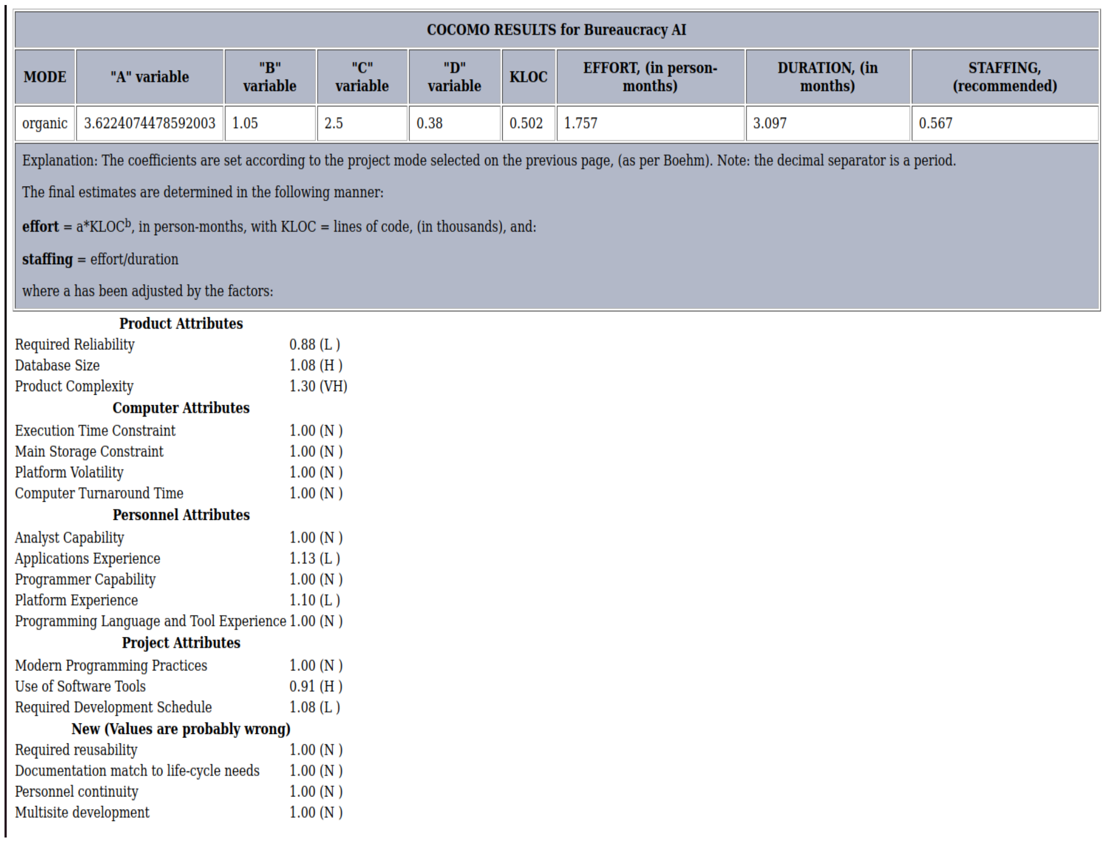
4.1.4 User Stories



USER STORIES or TASKS

TASK TITLE	TASK DESCRIPTION	ADDED BY	DATED ADDED
User access	As a User I want to access into the application to start chatting	Team	2/10/2023
User Interface	As a User I want to access the application by a web-interface so that I can interacts with it	Team	2/10/2023
Chat	As a User I want to digit my request in the chat so that the application can answer	Team	2/10/2023
AI response	As a User I want to have a clear understanding of the answer given by the system	Team	2/10/2023
Saving	As a User I want to be able to save the response I have obtained so that I can check for it multiple times or in a successive moment (removed by user stories)	Team	2/10/2023
Chat Reset	As a User I want to be able to reset the chat so that I can search for a different argument	Team	3/10/2023
Admin access	As an Admin I want access into the system in administrator panel	Team	3/10/2023
Plugin Management	As an Admin I want to have an interface that allows me to manage the plugins (add/remove plugins?)	Team	3/10/2023
Chat testing	As an Admin I want to be able to interact with the chat so that I can test the system	Team	3/10/2023

4.1.5 Cocomo II



COCOMO RESULTS for Bureaucracy AI

MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT, (in person-months)	DURATION, (in months)	STAFFING, (recommended)
organic	3.6224074478592003	1.05	2.5	0.38	0.502	1.757	3.097	0.567

Explanation: The coefficients are set according to the project mode selected on the previous page, (as per Boehm). Note: the decimal separator is a period.

The final estimates are determined in the following manner:

effort = a*KLOC^{b}, in person-months, with KLOC = lines of code, (in thousands), and:

staffing = effort/duration

where a has been adjusted by the factors:

Product Attributes	
Required Reliability	0.88 (L)
Database Size	1.08 (H)
Product Complexity	1.30 (VH)
Computer Attributes	
Execution Time Constraint	1.00 (N)
Main Storage Constraint	1.00 (N)
Platform Volatility	1.00 (N)
Computer Turnaround Time	1.00 (N)
Personnel Attributes	
Analyst Capability	1.00 (N)
Applications Experience	1.13 (L)
Programmer Capability	1.00 (N)
Platform Experience	1.10 (L)
Programming Language and Tool Experience	1.00 (N)
Project Attributes	
Modern Programming Practices	1.00 (N)
Use of Software Tools	0.91 (H)
Required Development Schedule	1.08 (L)
New (Values are probably wrong)	
Required reusability	1.00 (N)
Documentation match to life-cycle needs	1.00 (N)
Personnel continuity	1.00 (N)
Multisite development	1.00 (N)