DAMA-IT

GROUP MEMBERS

Lorenzo Gizzi 1907374
Lorenzo Russo 2091186

git hub repository link: `https://github.com/loregi01/DAMA-IT`

September 9, 2024

Contents

1 Introduction

1.1 The idea of the application

The idea of our project was to create a desktop application that allows users to play Dama online. Dama is an abstract game for two players, in which one player controls the black pieces and the other controls the white ones. The pieces can only move one square forward diagonally, but they can "eat" the opposing pieces by jumping over them and landing on the next square. Having more possibilities to grab, the following priorities must be respected in order: it is mandatory to eat where there are more pieces; with the same number of pieces to take, between the pawn and the "Dama" you are forced to hit the "Dama"; furthermore, if you can choose between eating a "Dama" or a pawn, it is mandatory to eat with the king. The "Dama" chooses the outlet where the most "Dame" are eaten; under equal conditions, you eat where the opposing "Dama" meets first. When a pawn reaches the last row of the board, it is promoted to a "Dama" and can move both forwards and backwards. The project develops following the rules of Italian "Dama". The pawn always moves diagonally on the dark squares, only forwards and one square at a time. The "Dame" can take the opponent's pieces as long as they are on a diagonal square next to their own, both forwards and backwards, and that they have the next diagonal square free; in the event that an identical capture situation occurs from the new arrival position, a "multiple capture" must be made, i.e. the greatest possible number of opposing pieces. The game is won if the opponent retires or when all the opponent's pieces are captured or blocked. In addition, each user has the possibility to follow other users in order to include them in their "friends list". In this way, a player can both play and chat with the people he follows.

2 Overview

2.1 Overview of the main components

First, is shown an image depicting the system architecture:

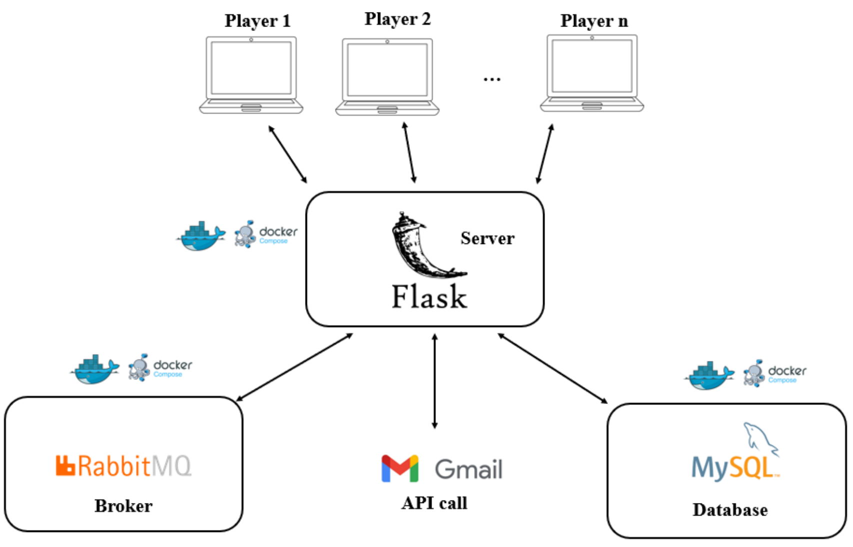


Figure 1: System architecture

This is a distributed system based on Docker containers and is composed by the following components:

- **Server Flask**: This is a server written in Python that takes care of managing all the logic of the system. It is the main component that acts as an intermediary between users and all the other components of the system and allows direct communication between players (exchange of moves during a game, chat).
- **MySql Database**: It is the component that takes care of saving all the fundamental data for the application, it is a relational database based on SQL. It is used for example to save user's data, the results of the played games and the messages exchanged in chat.
- **RabbitMQ**: It is a messaging broker that allows you to insert messages into queues. In the application it has been used to allow users to chat in real-time and the messages are sent end-to-end on the base of the user who is the correct receiver.

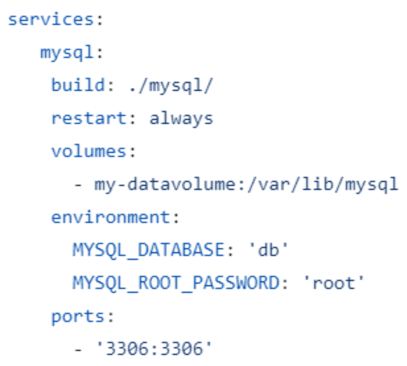
Additionally, the server communicates with **Google GMail** via API calls to send a confirmation email to each user who signs up correctly for the application.

2.2 Docker Deployment

In order to realize the structure defined in the previous section, we wrote a docker-compose file that allows to launch all the necessary microservices. The file structure is defined as follows for each service:

- **service name**: Specifies the name of the service that needs to be configured.
- **build**: Allows to build a Docker image from a specified Dockerfile.
- **volumes**: Maps volumes from the local filesystem to the container, for persistence or to share files between the container and the host.
- **environment**: Defines environment variables for the container.
- **ports**: It is used to map the Docker container ports to the host ports

Here is an excerpt from the docker-compose.yml file:



```
services:
   mysql:
    build: ./mysql/
    restart: always
    volumes:
      - my-datavolume:/var/lib/mysql
    environment:
      MYSQL_DATABASE: 'db'
      MYSQL_ROOT_PASSWORD: 'root'
    ports:
      - '3306:3306'
```

Figure 2: docker-compose.yml

2.3 Technology

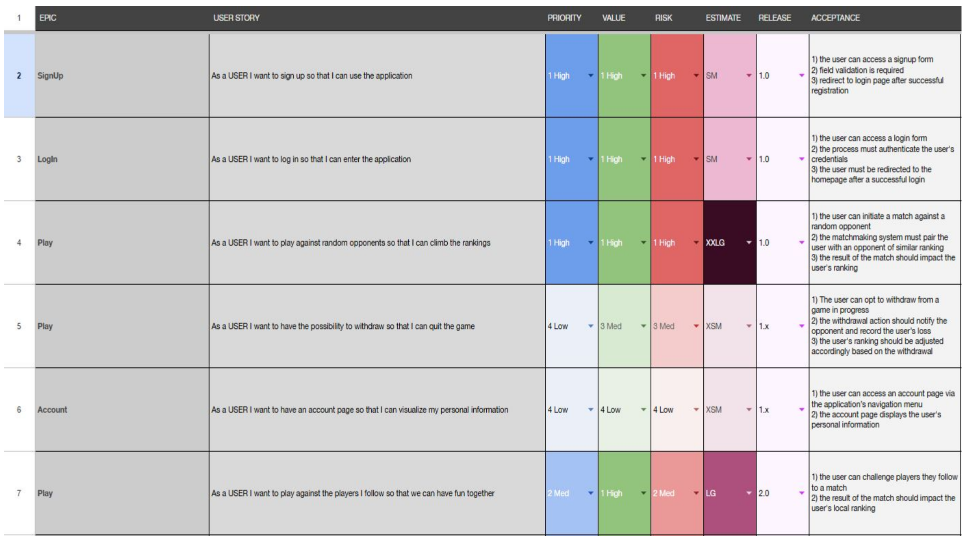
The main technology used in the development of the application is **Qt**. Qt is a powerful cross-platform framework for developing graphical user interfaces (GUIs), embedded applications, and softwares with rich user interfaces that allow programmers to write code that works on multiple platforms without significant modifications.
The main features of the technology are:

- **Cross-platform Support:** by using Qt, developers can create a single code base for applications that runs on Windows, macOS, Linux, Android and iOS environments, reducing the development and maintenance time.
- **Intuitive GUIs:** Qt provides a powerful set of tools for creating graphical user interfaces (GUIs). With advanced widgets and a modular design, helps for the creation of complex and interactive interfaces by using automatic layouts and supports the programmer for different graphic styles, and native themes for each operating system.
- **Modularity and Scalability:** Qt is highly modular. This means the programmer can use only the modules he needs, keeping the application lightweight and scalable. The main module used in the application is Qt Core which provides basic functionalities such as thread management, timers and objects.
- **Event Management:** Qt relies on a robust event management system, the event loop, that allows the programmer to efficiently manage the interaction between the user interface and the business logic. Communication between objects occurs through a mechanism of signals and slots, which facilitates asynchronous and decoupled communication between components.

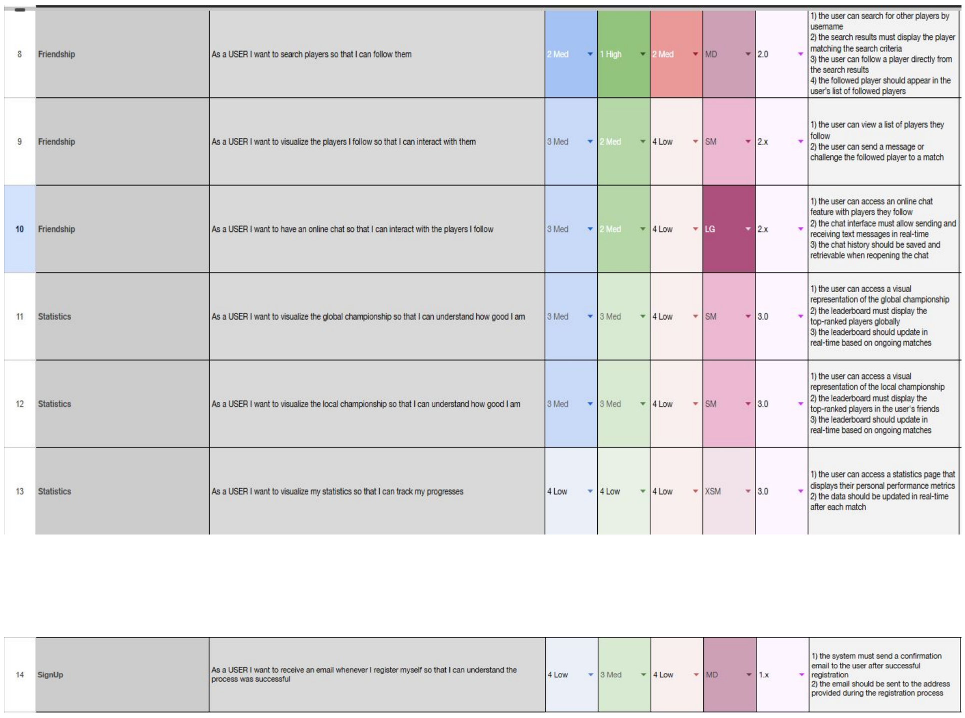
3 Design Techniques

3.1 User Stories

The User Stories are short descriptions of software functionalities from the end-user's perspective. They are used in the context of agile development to capture requirements in a simple and understandable way, without going into technical details.
The structure is composed as follows:
As [type_of_user] I want to [action] so that [goal].
What follows is an image containing the user stories of the application:



1	EPIC	USER STORY	PRIORITY	VALUE	RISK	ESTIMATE	RELEASE	ACCEPTANCE
2	SignUp	As a USER I want to sign up so that I can use the application	1 High	1 High	1 High	SM	1.0	1) the user can access a signup form 2) field validation is required 3) redirect to login page after successful registration
3	LogIn	As a USER I want to log in so that I can enter the application	1 High	1 High	1 High	SM	1.0	1) the user can access a login form 2) the process must authenticate the user's credentials 3) the user must be redirected to the homepage after a successful login
4	Play	As a USER I want to play against random opponents so that I can climb the rankings	1 High	1 High	1 High	XXLG	1.0	1) the user can initiate a match against a random opponent 2) the matchmaking system must pair the user with an opponent of similar ranking 3) the result of the match should impact the user's ranking
5	Play	As a USER I want to have the possibility to withdraw so that I can quit the game	4 Low	3 Med	3 Med	XSM	1.x	1) The user can opt to withdraw from a game in progress 2) the withdrawal action should notify the opponent and record the user's loss 3) the user's ranking should be adjusted accordingly based on the withdrawal
6	Account	As a USER I want to have an account page so that I can visualize my personal information	4 Low	4 Low	4 Low	XSM	1.x	1) the user can access an account page via the application's navigation menu 2) the account page displays the user's personal information
7	Play	As a USER I want to play against the players I follow so that we can have fun together	2 Med	1 High	2 Med	LG	2.0	1) the user can challenge players they follow to a match 2) the result of the match should impact the user's local ranking



8	Friendship	As a USER I want to search players so that I can follow them	2 Med	1 High	2 Med	MD	2.0	1) the user can search for other players by username 2) the search results must display the player matching the search criteria 3) the user can follow a player directly from the search results 4) the followed player should appear in the user's list of followed players
9	Friendship	As a USER I want to visualize the players I follow so that I can interact with them	3 Med	2 Med	4 Low	SM	2.x	1) the user can view a list of players they follow 2) the user can send a message or challenge the followed player to a match
10	Friendship	As a USER I want to have an online chat so that I can interact with the players I follow	3 Med	2 Med	4 Low	LG	2.x	1) the user can access an online chat feature with players they follow 2) the chat interface must allow sending and receiving text messages in real-time 3) the chat history should be saved and retrievable when reopening the chat
11	Statistics	As a USER I want to visualize the global championship so that I can understand how good I am	3 Med	3 Med	4 Low	SM	3.0	1) the user can access a visual representation of the global championship 2) the leaderboard must display the top-ranked players globally 3) the leaderboard should update in real-time based on ongoing matches
12	Statistics	As a USER I want to visualize the local championship so that I can understand how good I am	3 Med	3 Med	4 Low	SM	3.0	1) the user can access a visual representation of the local championship 2) the leaderboard must display the top-ranked players in the user's friends 3) the leaderboard should update in real-time based on ongoing matches
13	Statistics	As a USER I want to visualize my statistics so that I can track my progresses	4 Low	4 Low	4 Low	XSM	3.0	1) the user can access a statistics page that displays their personal performance metrics 2) the data should be updated in real-time after each match
14	SignUp	As a USER I want to receive an email whenever I register myself so that I can understand the process was successful	4 Low	3 Med	4 Low	MD	1.x	1) the system must send a confirmation email to the user after successful registration 2) the email should be sent to the address provided during the registration process

Figure 3: User Stories

3.2 Mockups

Mockups are graphical representations of the application used to design and visualize the user interface (UI) and user experience (UX) before starting development. They show how the interface elements, such as buttons, menus, and windows, will be arranged and help to identify and to resolve design or usability issues before the actual coding. In this section, some of the mockups used as the first prototype of the application are shown. The mockups are realized using Balsamiq software.

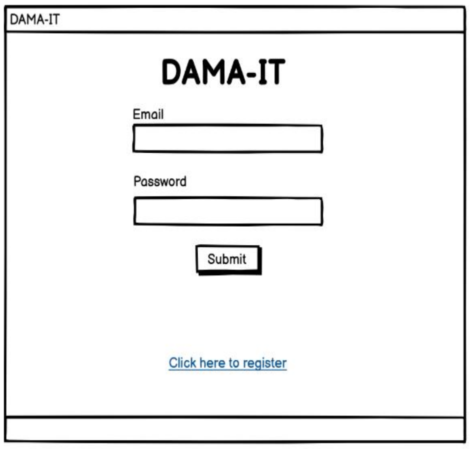


Figure 4: Login page

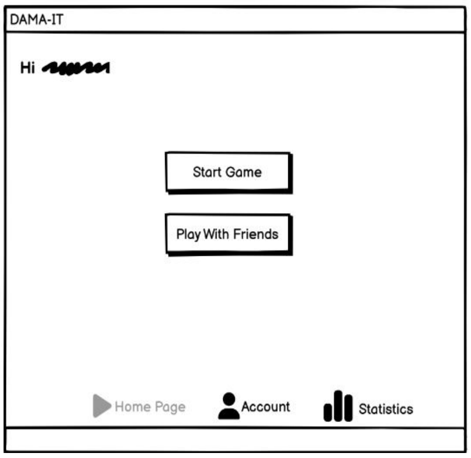


Figure 5: Homepage

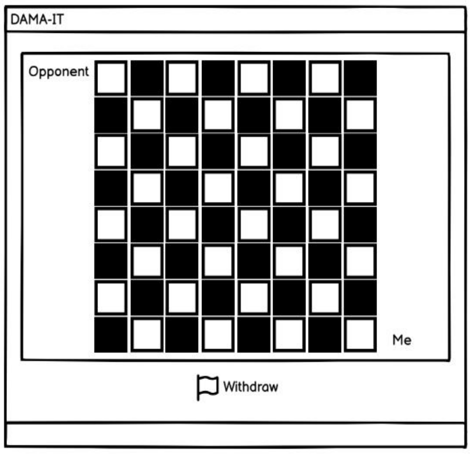


Figure 6: Game page

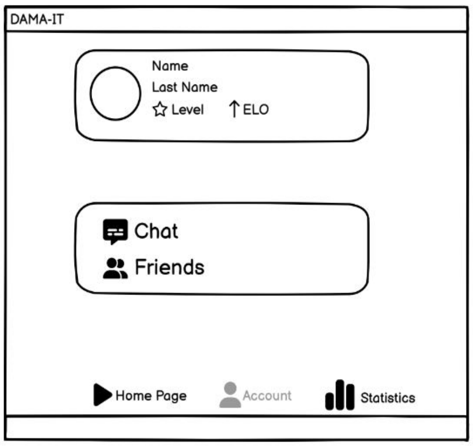


Figure 7: Account page

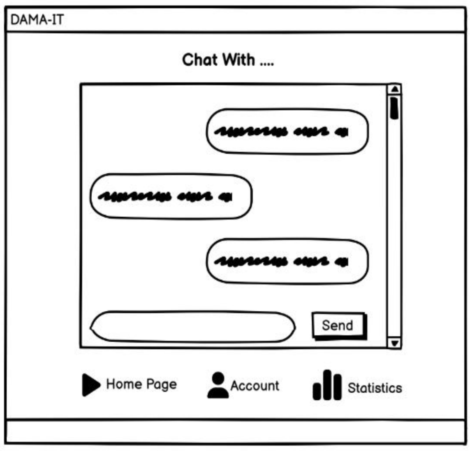


Figure 8: Chat page

3.3 Database

This section shows the ER diagram used for the application database design.

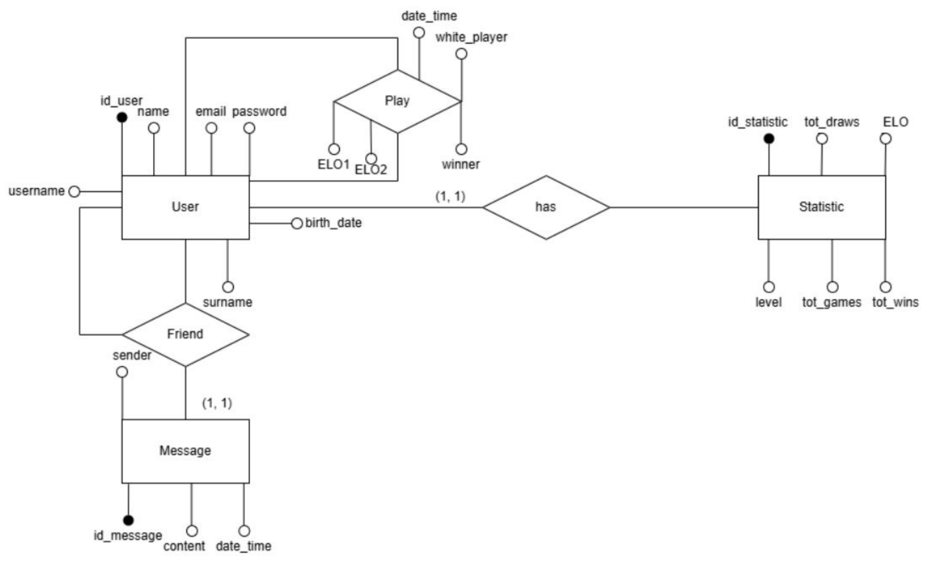


Figure 9: ER schema

The entities are:

- **User**: stores key user's information such as: name, surname, email, password, birth date, username.
- **Statistic**: stores each user's game statistics such as: ELO, total draw, total games, total wins, level.
- **Message**: stores all the exchanged messages among the users.

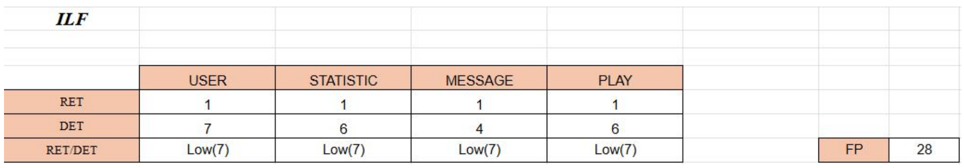
The main relationships are:

- **Play**: stores the played matches among the users.
- **Friend**: for each user stores the players he follows.

3.4 Function Points

Function Points (FP) are a standardized measure used to estimate the size and complexity of a software system based on its functionalities. Function Points quantify the work required to develop a software system by evaluating the functionalities it offers to the end user, rather than the code used to implement it.
The main components are:

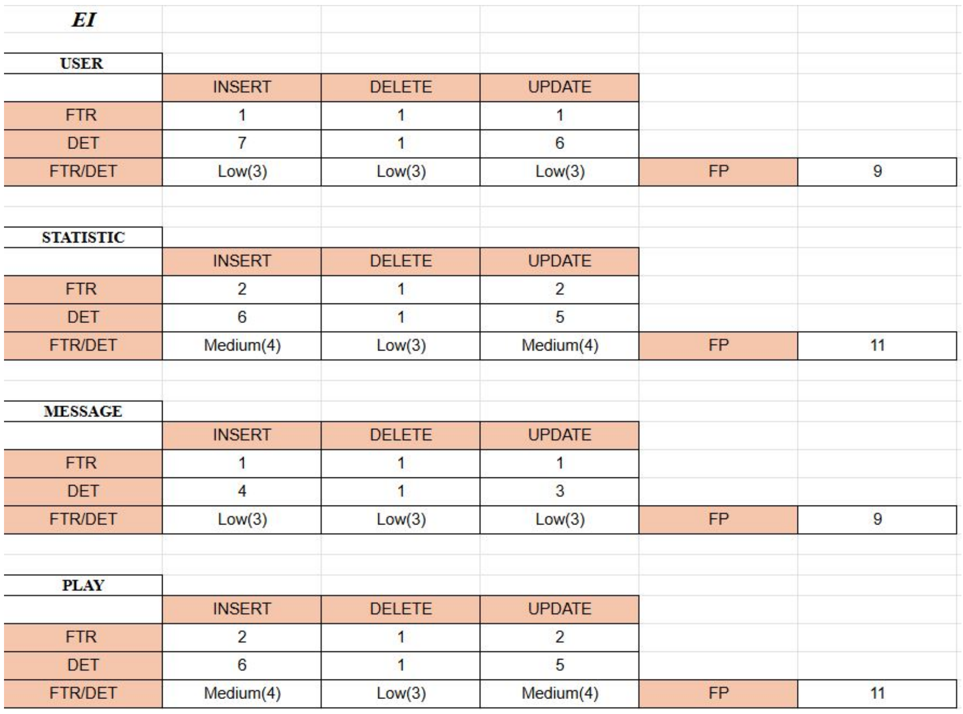
- **External Inputs (EI)**: Data flows or commands that the system receives from the outside (inputs).
- **External Outputs (EO)**: Results produced by the system and sent to the outside, such as reports or messages.
- **External Inquiries (EQ)**: Requests that the system processes to provide data to the user.
- **Internal Logical Files (ILF)**: Data that the system stores and uses internally.
- **External Interface Files (EIF)**: Files managed by other systems that are used by the software. In this project there are not EIFs.



ILF

	USER	STATISTIC	MESSAGE	PLAY		
RET	1	1	1	1		
DET	7	6	4	6		
RET/DET	Low(7)	Low(7)	Low(7)	Low(7)	FP	28

Figure 10: ILF



EI

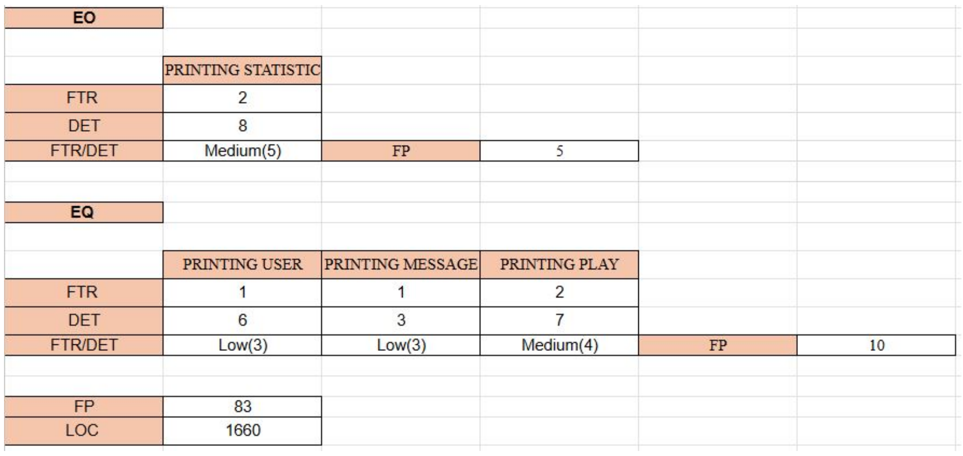
USER					
	INSERT	DELETE	UPDATE		
FTR	1	1	1		
DET	7	1	6		
FTR/DET	Low(3)	Low(3)	Low(3)	FP	9

STATISTIC					
	INSERT	DELETE	UPDATE		
FTR	2	1	2		
DET	6	1	5		
FTR/DET	Medium(4)	Low(3)	Medium(4)	FP	11

MESSAGE					
	INSERT	DELETE	UPDATE		
FTR	1	1	1		
DET	4	1	3		
FTR/DET	Low(3)	Low(3)	Low(3)	FP	9

PLAY					
	INSERT	DELETE	UPDATE		
FTR	2	1	2		
DET	6	1	5		
FTR/DET	Medium(4)	Low(3)	Medium(4)	FP	11

Figure 11: EI



EO			
	PRINTING STATISTIC		
FTR	2		
DET	8		
FTR/DET	Medium(5)	FP	5

EQ					
	PRINTING USER	PRINTING MESSAGE	PRINTING PLAY		
FTR	1	1	2		
DET	6	3	7		
FTR/DET	Low(3)	Low(3)	Medium(4)	FP	10

FP	83
LOC	1660

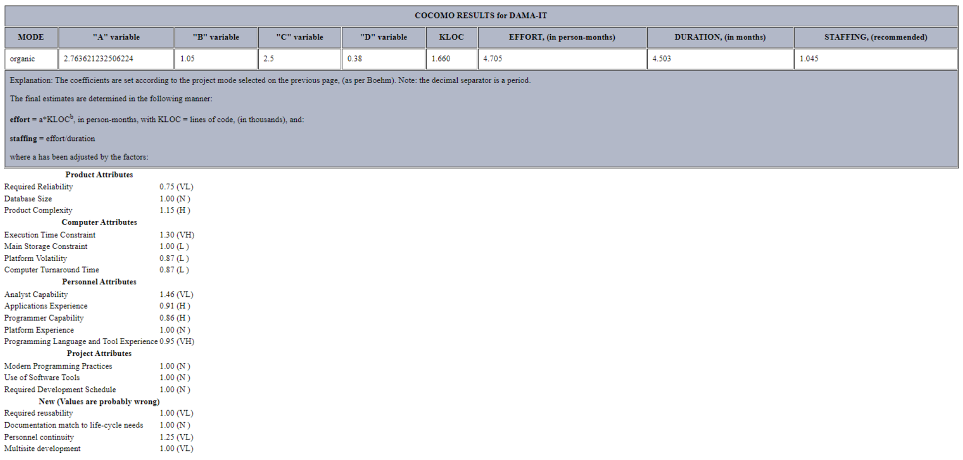
Figure 12: EO and EQ

The total amount of Function Points (FP) is 83, this result is obtained by summing up the single FPs of each components described above. Once computed the Function Points, we used a table to convert them into Line of Code (LOC), taking into account that one FP correspond to 20 LOC in Python (the language we used).

3.5 Cocomo II

COCOMO II (COnstructive COst MOdel II) is a mathematical model used to estimate the costs, the time and the resources required to develope the software product.
COCOMO II considers several constant factors (effort multipliers) that influence the development effort, such as:

- **Product attributes**: Software complexity, required performance, reliability.
- **Computer attributes**: Used to estimate software development effort and cost.
- **Personnel attributes**: Experience and competence of the team, familiarity with the technologies used.
- **Project attributes**: Used tools, time constraints.



COCOMO RESULTS for DAMA-IT								
MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT, (in person-months)	DURATION, (in months)	STAFFING, (recommended)
organic	2.763621232506224	1.05	2.5	0.38	1.660	4.705	4.503	1.045

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	0.75 (VL)
Database Size	1.00 (N)
Product Complexity	1.15 (H)
Computer Attributes	
Execution Time Constraint	1.30 (VH)
Main Storage Constraint	1.00 (L)
Platform Volatility	0.87 (L)
Computer Turnaround Time	0.87 (L)
Personnel Attributes	
Analyst Capability	1.46 (VL)
Applications Experience	0.91 (H)
Programmer Capability	0.86 (H)
Platform Experience	1.00 (N)
Programming Language and Tool Experience	0.95 (VH)
Project Attributes	
Modern Programming Practices	1.00 (N)
Use of Software Tools	1.00 (N)
Required Development Schedule	1.00 (N)
New (Values are probably wrong)	
Required reusability	1.00 (VL)
Documentation match to life-cycle needs	1.00 (N)
Personnel continuity	1.25 (VL)
Multisite development	1.00 (VL)

Figure 13: Cocomo II

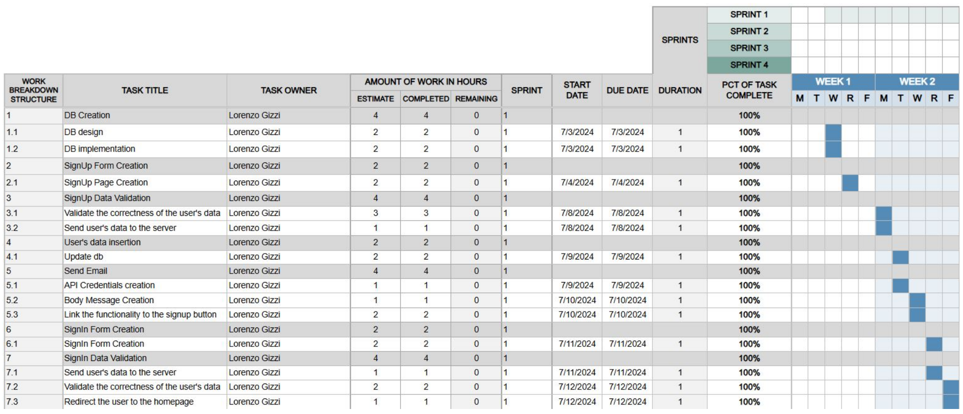
By using the Cocomo II method formulas we computed:

- **Effort**: number of months it would take a person to complete the project. The used formula is $A \times KLOC^B$.
- **Staffing**: is used to plan and manage the personnel needed to complete the project. The formula used is $effort \times duration$.

- **Duration**: refers to the total time required to complete a software project. The formula used is $C \times effort^D$.

3.6 Scrum

SCRUM is an agile framework that relies on a set of principles and practices to manage complex projects. It is designed to be flexible and adaptable, allowing teams to respond quickly to changes and continuously improve. We worked on 2-week sprints. Each user story was divided into multiple tasks, thus defining the backlog. For each sprint, we assigned a certain number of tasks, based on the estimated number of hours needed to complete each of them. We got 4 sprints, for a total of 2 months of estimated work. Below is an extract of the backlog (fig. 14) and also an extract of the Gantt chart (fig. 14).



SPRINTS
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				
			ESTIMATE	COMPLETED	REMAINING						M	T	W	R	F	M	T	W	R	F
1	DB Creation	Lorenzo Gizzi	4	4	0	1				100%										
1.1	DB design	Lorenzo Gizzi	2	2	0	1	7/3/2024	7/3/2024	1	100%			🟦							
1.2	DB implementation	Lorenzo Gizzi	2	2	0	1	7/3/2024	7/3/2024	1	100%			🟦							
2	SignUp Form Creation	Lorenzo Gizzi	2	2	0	1				100%										
2.1	SignUp Page Creation	Lorenzo Gizzi	2	2	0	1	7/4/2024	7/4/2024	1	100%				🟦						
3	SignUp Data Validation	Lorenzo Gizzi	4	4	0	1				100%										
3.1	Validate the correctness of the user's data	Lorenzo Gizzi	3	3	0	1	7/8/2024	7/8/2024	1	100%						🟦				
3.2	Send user's data to the server	Lorenzo Gizzi	1	1	0	1	7/8/2024	7/8/2024	1	100%						🟦				
4	User's data insertion	Lorenzo Gizzi	2	2	0	1				100%										
4.1	Update db	Lorenzo Gizzi	2	2	0	1	7/9/2024	7/9/2024	1	100%							🟦			
5	Send Email	Lorenzo Gizzi	4	4	0	1				100%										
5.1	API Credentials creation	Lorenzo Gizzi	1	1	0	1	7/9/2024	7/9/2024	1	100%							🟦			
5.2	Body Message Creation	Lorenzo Gizzi	1	1	0	1	7/10/2024	7/10/2024	1	100%								🟦		
5.3	Link the functionality to the signup button	Lorenzo Gizzi	2	2	0	1	7/10/2024	7/10/2024	1	100%								🟦		
6	SignIn Form Creation	Lorenzo Gizzi	2	2	0	1				100%										
6.1	SignIn Form Creation	Lorenzo Gizzi	2	2	0	1	7/11/2024	7/11/2024	1	100%									🟦	
7	SignIn Data Validation	Lorenzo Gizzi	4	4	0	1				100%										
7.1	Send user's data to the server	Lorenzo Gizzi	1	1	0	1	7/11/2024	7/11/2024	1	100%									🟦	
7.2	Validate the correctness of the user's data	Lorenzo Gizzi	2	2	0	1	7/12/2024	7/12/2024	1	100%										🟦
7.3	Redirect the user to the homepage	Lorenzo Gizzi	1	1	0	1	7/12/2024	7/12/2024	1	100%										🟦

Figure 14: Gantt chart

Finally, it was possible to create the Burndown chart (fig. 15), which allows you to view the progress of the work.

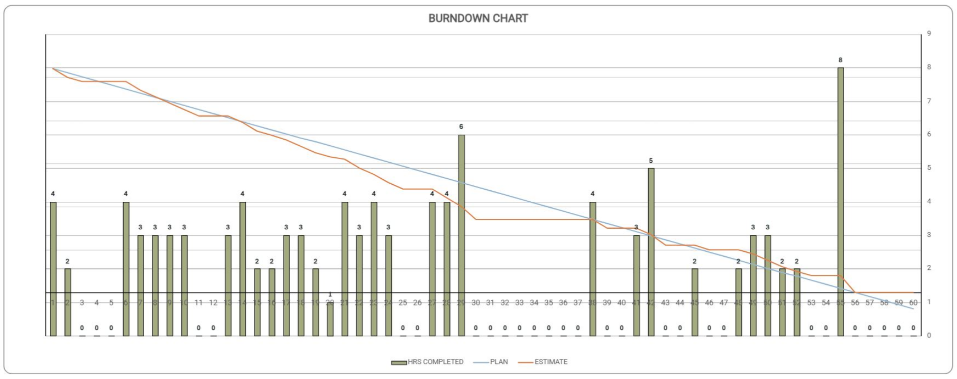


Figure 15: Burndown chart

4 Conclusions

In this report, we have focussed especially on the design and development of a distributed Dama application, implemented by using advanced technologies and design practices to ensure an high-quality product. The usage of those technologies and design techniques helped to create an efficient and scalable distributed Dama application while the use of Docker and RabbitMQ improved the infrastructure management and communication between components, in addition Qt and MySQL provided robust tools for the UI development and the data management.
The adopted design techniques, including user stories, mockups, function points, COCOMO II and SCRUM, played an important role to ensure that the application was well-designed, effectively managed and aligned to the user expectations.