Service Oriented Software Engineering

Collaborative Task Manager

Lluis Barca Pons
lluis.barcapons@studio.unibo.it

June 14, 2024

Contents

1. *Introduction*

1.1 Overview

The Collaborative Task Manager (CTM in advance) is a basic software solution designed to enhance productivity in collaboration across users in any organizational setting. This project is developed with the Jolie programming language [4], distinguished for its robust support for microservices architectures [1].

CTM project architecture leverages a series of interconnected microservices, that will be explained in the following sections, each one responsible for a different set of functionalities. This modular design not only facilitates scalability as organizational needs grow, also ensures system flexibility and changes resilient.

The systems are designed to support real-time data processing and updates, which allow all users to stay informed of task progress and changes at any moment. This is crucial for maintaining up-to-date task statuses and be sure that team members can attach different problems as they occurred.

1.2 Objectives

The CTM is built with different objectives, each one aimed at enhancing the overall performance of task management processes:

- Robust and scalable microservices architecture
- Data integrity
- Concurrent processing
- Clear UI

2. *System Architecture*

The architecture of the CTM system leverages a microservices design pattern to ensure scalability, flexibility, and independent service evolution [3]. This design philosophy is rooted in the principles of Service-Oriented Architecture (SOA) and is tailored to accommodate the dynamic and distributed nature of modern software applications.

2.1 Microservices

The system consists of some microservices, each designed to handle specific aspects of task management functionality. These services operate independently, but each one communicated via a **Main** program:

- **User Management Service:** Manages user creation or deletion. It also supports authentication functionalities to ensure secure access to the system.
- **Task Service:** Responsible for task creation, assignment, status updates, and archival. It handles the core functionality of task management and maintains the integrity of tasks.
- **Notification Service:** Sends and achieve notifications to all user's behaviour. It integrates with both the User and Task Services to provide real-time updates on task changes and alerts.

2.2 Communication

Following the guidelines from the literature on microservices and Jolie, our system architecture emphasizes the importance of decoupled communication through asynchronous message-passing mechanisms, which are crucial for reducing dependencies between services:

- **Asynchronous messaging:** The system uses Jolie's native support for asynchronous communication to facilitate non-blocking interactions between services. This method allows services to remain responsive, as they do not need to wait for responses before continuing execution.
- **Service interfaces:** Each microservice defines its public interface in Jolie, which includes the types of messages it can send and receive. These interfaces are crucial for ensuring that services interact seamlessly and that changes to one service do not adversely affect others.

3. *Requirements*

3.1 Functional

This section describes the specific functional requirements that the CTM system must fulfil. These requirements ensure that the system provides all necessary functionalities to support efficient task management in a collaborative environment.

3.1.1 User registration and authentication

- **User Registration:** Users must be able to register with the system by providing necessary details such as name, email, and password. The system should validate the input data and confirm the registration via `registerUser()` function.
- **Secure Authentication:** The system must support a secure login process using credentials provided at registration. It should also support multifactor authentication to enhance security for user access via `authUser()` and `checkUser()` functions, depending on as we are checking if a new user can be registered or if only want to check if a username is already used.

3.1.2 Task management

- **Task Creation:** Users must be able to create tasks with specific details including title, description, due date, and priority using the `createTask()` function.
- **Task Assignment:** Tasks should be assignable to other registered users. The system should notify the assigned users to each other via `sendNotification()` function, that it's used for sent notifications.
- **Task Updates and Tracking:** The system must provide real-time updates on task status changes, such as from 'in progress' to 'completed'. Users should be able to track the history of all updates due to the notification historial. These task updates are made by `modifyTaskUser()` or `modifyTaskStatus()` functions.
- **Task Deletion:** Users are able to eliminate tasks whenever they want. As others task functionalities, are also registered via notifications. It's possible to delete tasks thanks to `deleteTask()` function.

3.1.3 Notifications

- **Real-time Notifications:** The system must send real-time notifications to users about important events related to their tasks, such as task assignments or status updates via `sendNotification()` function.

3.1.4 Concurrency

Concurrency in the CTM system is a fundamental aspect, designed to allow the system to handle multiple operations simultaneously. The system leverages the concurrency features of Jolie, particularly focusing on synchronized execution where necessary to manage access to shared resources like global variables.

- **Concurrent Execution:** Each microservice is capable of handling requests in a non-blocking and concurrent manner. This is achieved through the use of Jolie's native capabilities for handling concurrent service requests, which ensures that services can scale according to load without performance bottlenecks.

- **Synchronized Access:** To manage access to shared resources efficiently and prevent race conditions, the system employs Jolie's `synchronized()` construct. This is particularly important to maintain data consistency. With this function, we can ensure that only one process can execute the block at any given time.

  - Example of Synchronized Usage: In critical sections of the code where shared resources are accessed or modified, the system uses the `synchronized()` block to ensure that only one process can execute the block at any given time.

    ```
    synchronized(resourceKey) {
        // Code to update shared resource
        sharedResource.update(newData);
    }
    ```

    Here, `resourceKey` is a unique identifier for the resource being accessed, which ensures that all accesses to the resource are serialized, thus avoiding concurrent modification errors.

3.2 Non-functional

The non-functional requirements of CTM define the system's operational qualities and service attributes. These requirements are crucial for ensuring the system's reliability, performance, and user satisfaction.

3.2.1 Data integrity

Data integration in the system is handled through well-defined interfaces and communication protocols, primarily HTTP, which enables synchronous and asynchronous interactions between services. This setup allows the system to maintain a unified data flow, with real-time updates and transactions managed efficiently across different service boundaries. The use of Jolie's native features like synchronous operations and embedded data types ensures that data is not only transmitted securely but also transformed and stored in formats that are optimal for each specific service.

This integration strategy enhances the system's operational efficiency, provides a better user experience by ensuring data accuracy and timeliness, and supports scalability by allowing new services to be added without disrupting existing data flows.

3.2.2 Scalability

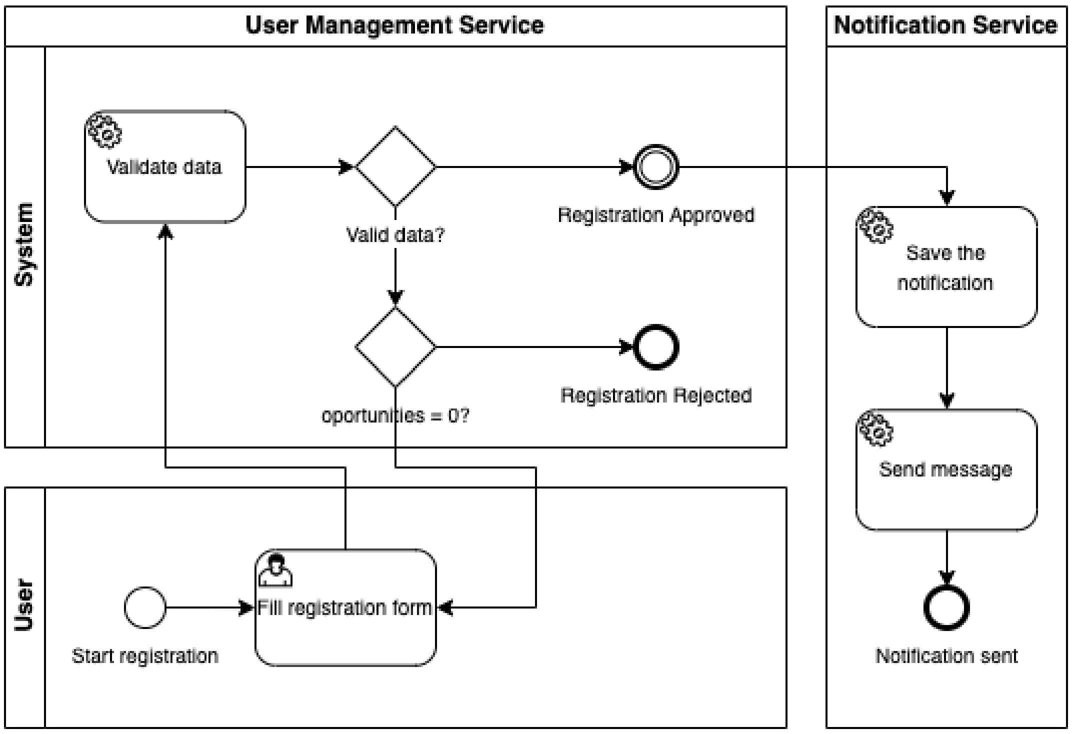
The system's microservices architecture enables exceptional scalability. Services are designed to be stateless, allowing them to scale horizontally as demand increases without significant changes to the architecture. Load balancing techniques distribute incoming requests evenly across multiple server instances, optimizing resource utilization and response times. This setup ensures that the system can handle growth in user numbers and data volume efficiently, maintaining high performance under varying loads.

3.2.3 User interface

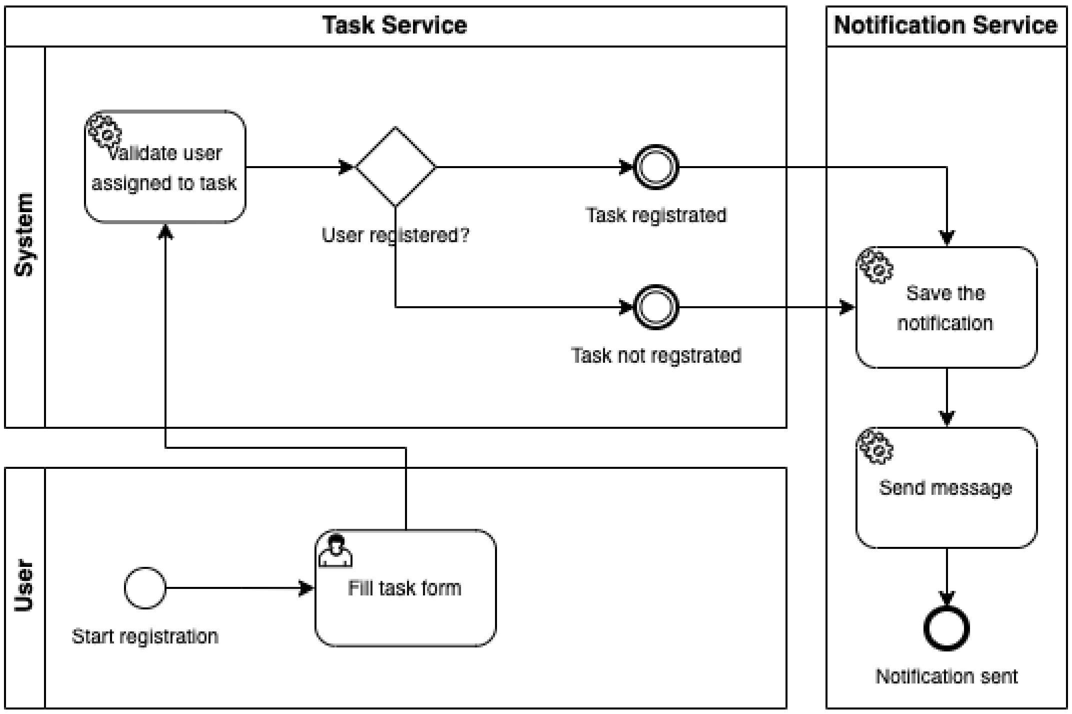
The user interface is designed to be simple and functional, utilizing a terminal-based interface to interact with users. This approach aligns with the system's goal to provide an efficient and straightforward user experience without the overhead of graphical elements.

4. *BPMN Diagrams*

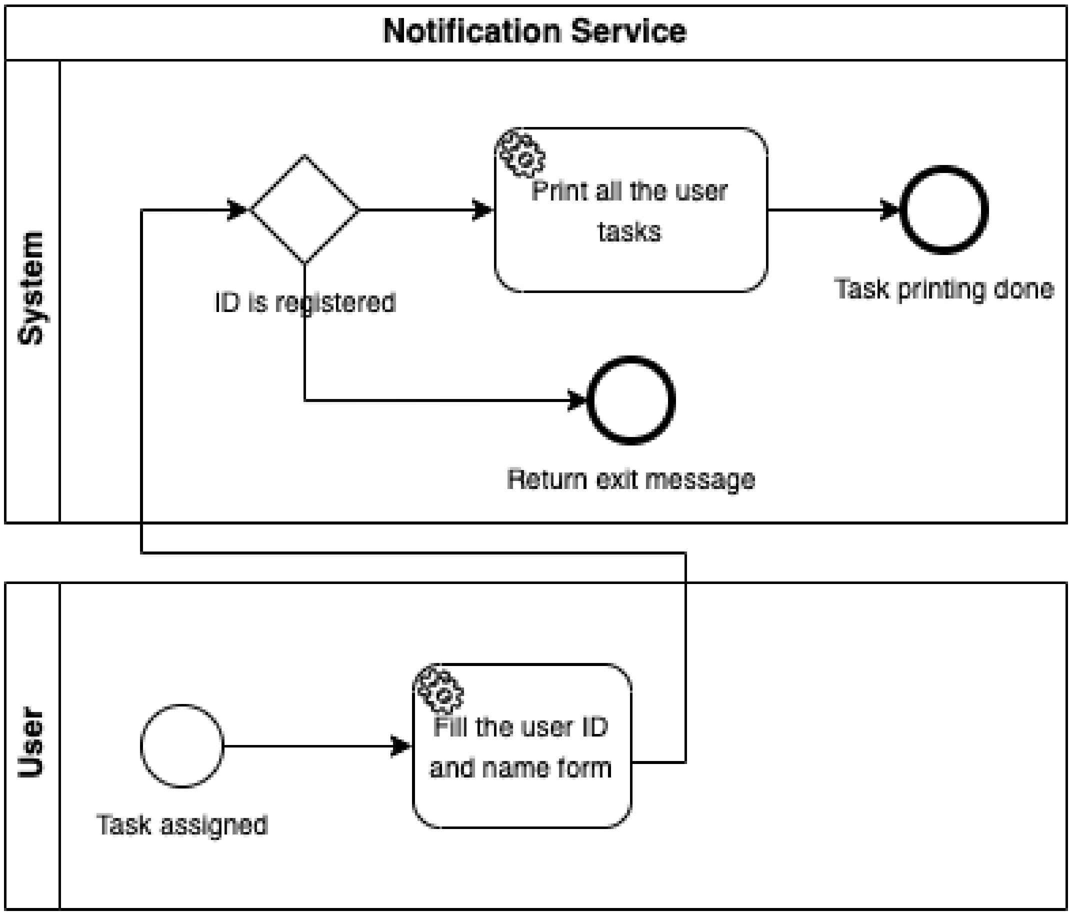
4.1 User Registration Process



4.2 Task Creation and Assignment Process



4.3 Notifications of a user Process



5. *Implementation*

5.1 How it works

First, you have to install Jolie on your machine. You can download it from the official website. Once you have installed Jolie, you have to be sure that the jolie command is added to your `PATH`. You can check it by running `jolie --version` in your terminal.

After that, you can run the project by executing the following commands in the root directory of the project. You should execute each service in different terminals:

```
# Terminal 1
jolie UserManagaementService.ol
```

```
# Terminal 2
jolie TaskManager.ol
```

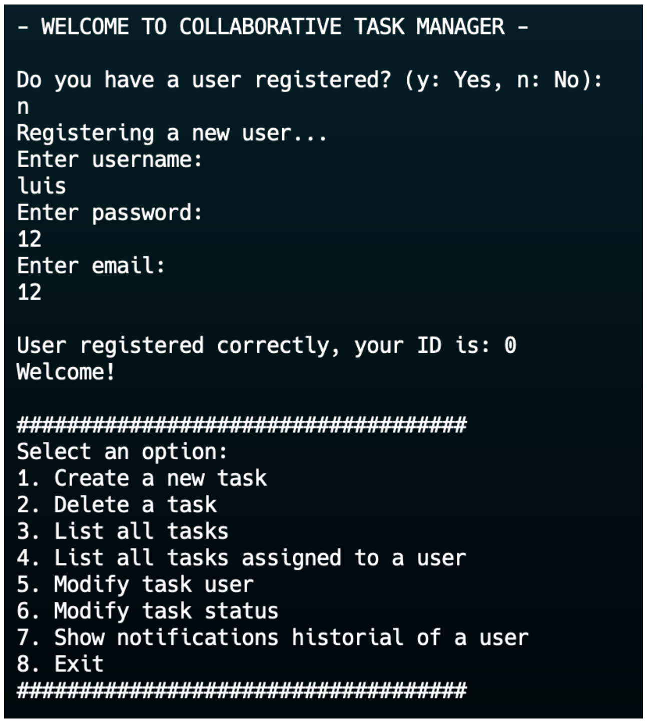
```
# Terminal 3
jolie TaskService.ol
```

```
# Terminal 4
jolie Main.ol
```

In Terminal 4 you could interact with the application. Also, it's recommended to check the documentation [2] for better understanding.

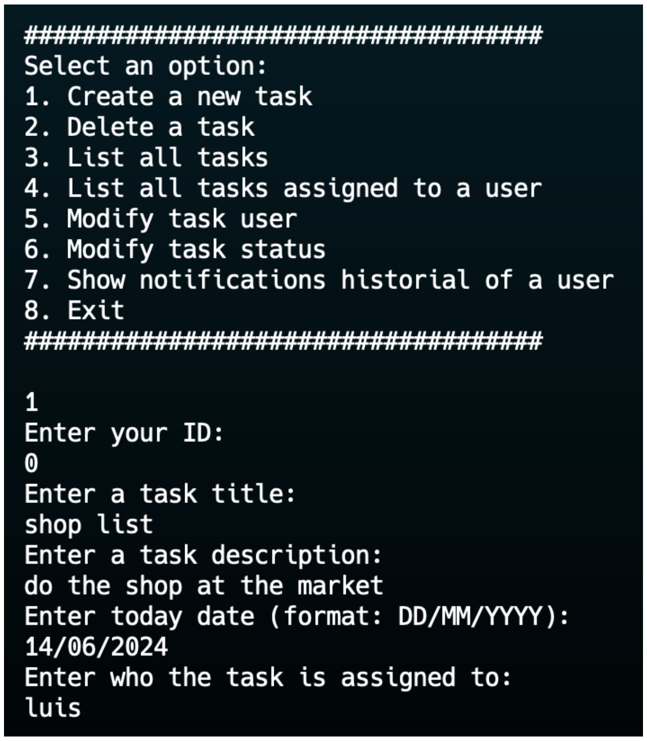
5.2 Execution examples

In the following examples, we will see how the application works in some use cases.



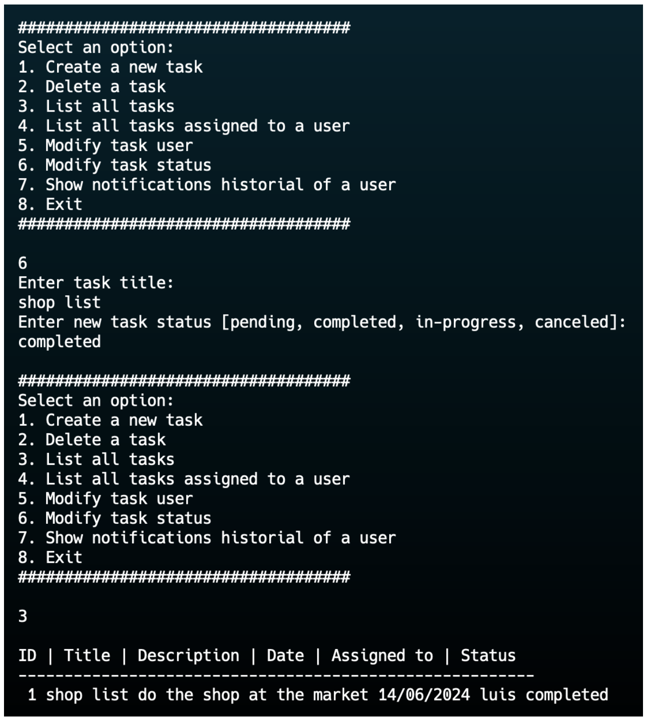
```
- WELCOME TO COLLABORATIVE TASK MANAGER -

Do you have a user registered? (y: Yes, n: No):
n
Registering a new user...
Enter username:
luis
Enter password:
12
Enter email:
12

User registered correctly, your ID is: 0
Welcome!

###################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
###################################
```

In this screenshot, we can see the first possible steps in the application (if the user is not registered). The system takes all the user attributes and save it. Then we can appreciate the menu to interact with the different services.



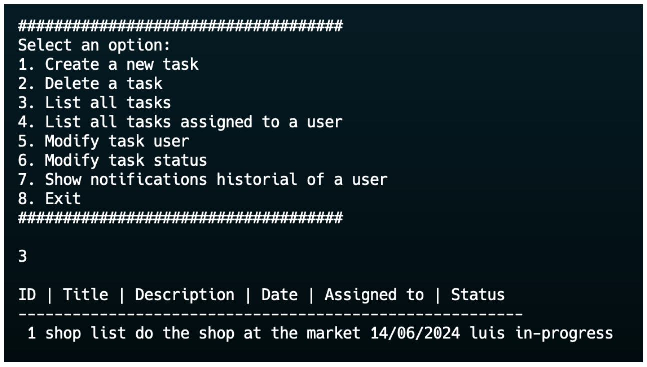
```
###################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
###################################

1
Enter your ID:
0
Enter a task title:
shop list
Enter a task description:
do the shop at the market
Enter today date (format: DD/MM/YYYY):
14/06/2024
Enter who the task is assigned to:
luis
```

Now we want to create a new task, that is created by a user and assigned to the same one or any other. The system ensure that all task information is well saved.



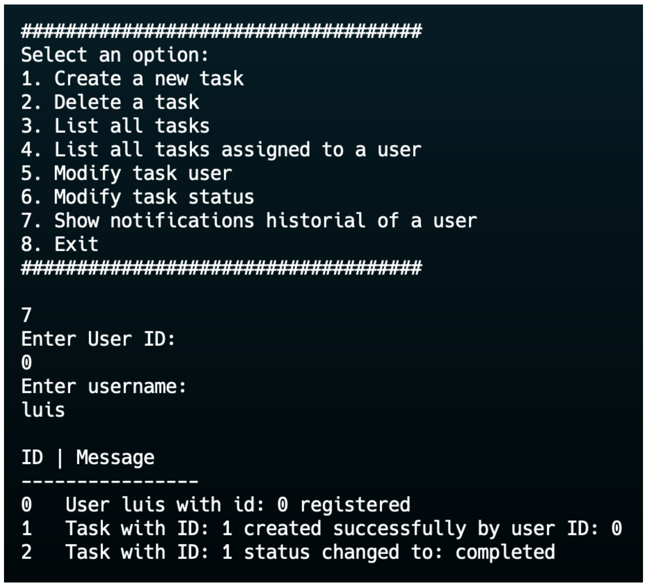
```
####################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
####################################

6
Enter task title:
shop list
Enter new task status [pending, completed, in-progress, canceled]:
completed

####################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
####################################

3

ID | Title | Description | Date | Assigned to | Status
--------------------------------------------------------
 1 shop list do the shop at the market 14/06/2024 luis completed
```

At this moment, we can interact with the system to use different services about tasks. For example, to change the task status, in order to check the progression of the hypothetical project.



```
####################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
####################################

3

ID | Title | Description | Date | Assigned to | Status
--------------------------------------------------------
 1 shop list do the shop at the market 14/06/2024 luis in-progress
```

At this point we can check if our task is created well, and all the information is correctly saved.



```
####################################
Select an option:
1. Create a new task
2. Delete a task
3. List all tasks
4. List all tasks assigned to a user
5. Modify task user
6. Modify task status
7. Show notifications historial of a user
8. Exit
####################################

7
Enter User ID:
0
Enter username:
luis

ID | Message
----------------
0   User luis with id: 0 registered
1   Task with ID: 1 created successfully by user ID: 0
2   Task with ID: 1 status changed to: completed
```

Finally, we can check the session trace or, if we want, the trace of a specific user. In this case, we check all the notifications registered during the session (so, all notifications of all users).

Otherwise, if we want to check the notifications one-by-one in real time, we can look to `NotificationService` terminal, where all notifications are displayed as they are sent.

6. *Conclusions*

The development of the Collaborative Task Management presented a unique set of challenges and learning opportunities, particularly through the adoption of the Jolie programming language. This project required not only learning about a service-oriented programming paradigm, but also learning how to use a language that supports such a paradigm at its core.

The initial phase of the project involved a steep learning curve, as Jolie's service-oriented concepts and syntax were unfamiliar. Mastering Jolie was challenging due to its distinct approach to handling services, concurrency, and communication between distributed components. However, these challenges were instrumental in gaining a deeper appreciation and understanding of microservices architectures.

The project tries to apply the practical application of theoretical concepts, proving that service-oriented architectures can effectively handle task management operations. Looking forward, there is potential to expand the system's capabilities, integrate more advanced features, and explore further enhancements to the user interface, for example.

During the development, I engaged deeply with the Jolie community. My efforts were not limited to merely using the Jolie language but extended to actively contributing to its documentation. I realized some little contributions to contribute in the open source community and help other new Jolie developers start programming with it.

In conclusion, this project not only reached its objective of creating a functional task management system but also contributed to the broader community by enhancing the resources available to Jolie developers. The experience has underscored the importance of persistence and community engagement in overcoming the challenges posed by learning and utilizing new technologies in software development.

Bibliography

[1] Claudio Guidi, Ivan Lanese, Manuel Mazzara, and Fabrizio Montesi. Microservices: a language-based approach. *Present and Ulterior Software Engineering*, pages 217–225, 2017.

[2] Jolie Community. Jolie documentation, 2024.

[3] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Composing services with jolie. In *Fifth European Conference on Web Services (ECOWS'07)*, pages 13–22. IEEE, 2007.

[4] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-oriented programming with jolie. In *Web Services Foundations*, pages 81–107. Springer, 2013.