A Microservice for Multi-Channel Notifications

Capstone Project

José Luís Cunha Rodrigues
Ricardo André Araújo de Matos
Rúben Costa Viana
Tiago Filipe Magalhães Barbosa

Licenciatura em Engenharia Informática e Computação

Tutor at U.Porto: Jácome Miguel Costa da Cunha
Company mentor: José Nuno Lapa Bonifácio
Proponent: Carlos Guilherme Araújo

2022/2023 - Second semester

Contents

1 Introduction

1.1 Environment set

This project was developed in the context of the Capstone Project Curricular Unit of the Informatics and Computer Engineering undergraduation at FEUP and in collaboration with Altice Labs.

With the fast-paced development of the world, technologies, and services in the telecommunications sector have undergone a major evolution. In order to manage the networks, services, and operational tasks, telecommunication service providers have resourced to use platforms known as Operations Support Systems (OSS). These systems help manage a high volume of data produced, which has to be monitored to maintain service quality and guarantee operational effectiveness. The objective of the OSS department is to offer companies technology that raises the standard of services delivered to society[1].

The capacity to alert clients and employees in real time of any network and service problems is one of the essential needs for an OSS platform. These alerts provide the teams with the chance to take quick action and avoid service interruptions. Yet, as collaboration technologies have grown in popularity, users want to receive these notifications over a variety of platforms like WhatsApp, phone calls, SMS, Microsoft Teams, and email.

It is also crucial for companies to have a fast response time to failures that might happen. With this project, machines will be able to contact the appropriate entities when a failure happens in the product they are in. For instance, by contacting a repairman when your internet fails.

In order to satisfy these needs, Altice Labs has suggested creating a microservice that can send notifications across several channels to inform users and products of network and/or service problems.

The social impact of this project is substantial since it immediately enhances people's lives by guaranteeing that telecommunications services are effective, dependable, and continuous.

1.2 Objectives and expected results

The objective of the project is to create a microservice that enables the notification of network and/or service problems to clients or employees. This microservice should support plugins for different methods of sending notifications, including WhatsApp, Microsoft Teams, email (SMTP), SMS (SMPP), and voice calls (HTTP REST). The team should evaluate the advantages and disadvantages of synchronous and/or asynchronous APIs. The key goal is to implement at least two plugins, with WhatsApp being mandatory.

This work aims to support several features, including the selection of the plugin to use, a list of recipients (which can be one or a group), a map of optional parameters (each plugin may have specific needs, such as attachments in emails), and the return of execution results. The microservice will be distributed with Docker containers and should be ready for deployment in Kubernetes, integrating with the other products in the Altice Labs' Operation Support Systems Stack, whether on-premises or in public clouds.

The microservice should be included in CI/CD pipelines, supporting static code analysis, security vulnerability scans, and end-to-end testing.

The project's goal is to provide an efficient, reliable, and scalable solution that enhances the Altice Labs OSS Stack's capabilities, ultimately contributing to the company's competitiveness in the telecommunications market.

1.3 Structure of the report

This report is outlined in 2 different main sections. First, we will discuss the methodology used and the main activities that were intrinsic to the development of the project. In section 2, we will discuss how we organized and projected the development of the project, enumerating the distinguished activities that lead to our final state result. After this, in section 3, we will take a look at the solution we found, explain the requirements, the architecture, and the validation of our solution. We will also enumerate the technologies used and explain their role in the resulting product.

2 Methodology used and main activities developed

2.1 Stakeholders, roles and responsibilities

The developer team was composed of 4 students from the Bachelor of Informatics and Computing Engineering at FEUP:

José Luís Cunha Rodrigues	`up202008462@fe.up.pt`
Ricardo André Araújo de Matos	`up202007962@fe.up.pt`
Rúben Costa Viana	`up202005108@fe.up.pt`
Tiago Filipe Magalhães Barbosa	`up202004926@fe.up.pt`

This team worked under the monitoring of professor Jácome Cunha (`jacome@fe.up.pt`) and Altice Labs mentor José Nuno Lapa Bonifácio (`jose-n-bonifacio@alticelabs.com`). The proposal of the project was made by Carlos Guilherme Araújo (`carlos-guilherme-araujo@alticelabs.com`).

All of the students had the same role of developer so most of our focus was to deal with the technical aspects of the project. We would plan the sprints every 1-2 weeks and usually work individually on the assigned tasks. Constant communication was in place all throughout the project and we even used techniques like pair programming to make sure that every team member had a clear understanding of the codebase and could actively contribute to its development.

Moreover, we would be weekly instructed by the company mentor. As a stakeholder, he had a better insight into the product's goals and could better explain to us the direction our development should take. This meant he took a major role while planning the sprints and shaping the final product.

Still, on the company side, we would occasionally communicate with the project's proponent who also played the role of stakeholder.

Finally, the last involved party in this project was the mentoring professor at the university. He took a more distanced role, making sure the project was running according to plan but giving us space to solve the proposed problems by ourselves. Nevertheless, he was constantly available to help us with any technical difficulties that appeared.

2.2 Methodology used

We employed an agile methodology, conducting weekly meetings to gather feedback and devise plans for the next iteration. This meant that the need for an initial comprehensive plan was reduced, as different requirements could change as the project grew. The weekly meeting held with the company mentor had the role of monitoring development and planning in which direction we wanted to grow. As our understanding of the assignment developed with it, feature suggestions would also come from the development team. We would additionally plan the individual tasks of each team member for the next sprint.

We diligently conducted tests and typically deployed a release to the main branch every week before our meeting, ensuring we received valuable input from our stakeholders.

In order to maintain the organization of our project, we used `git` to do version control and hosted the project on `GitHub`.

This project adhered to a continuous integration and continuous deployment (CI/CD) approach. A pipeline was implemented to automate the processes of building, testing, and static and vulnerability analyses. We also made use of `SonarCloud` to detect vulnerabilities and check for the overall quality of our code (Codes Smells, Reliability, Code duplication, ...), and Github's `Superlinter`, to make static analysis of the code.

2.3 Activities developed

As previously mentioned, we had a recurring meeting every week. The first of these meetings was held on the 10th of February with all the involved parties. According to what was established at this initial meeting, the team paid a visit to the company's headquarters on the 10th of March. On this day, we got to meet with the stakeholders. We were also provided with a better insight into the company's operations which gave us an even better understanding of the real-world applications of the assignment.

Several demos were held in some of the meetings with the company mentor, namely when a major feature was implemented. On the 23rd of May, we presented the project to the proponent to get some validation on the implementation.

Besides the initial meeting, we had a subsequent session with the professor on the 27th of April. At this point, we summarized the development and the teacher made sure we were according to plan. The remaining communication with the tutor was done via email.

To better understand the project's development we can take a look at the Gantt chart in Figure 1. It is dived according to the four main stages Initiation, Planning, Execution, and Closing.

Firstly, the moments following the first meeting were used to better understand the assignment and make a provisional initial plan. During this time, we tried to access each individual's experience with the technologies.

The Planning stage helped us better outline the key features we wanted to implement. As the collective understanding of the technology grew, we were able to discuss the architecture of the program. We also started researching the different targeted services, in order to understand the available interfaces provided.

As the outline became more clear-cut, we began the longest phase of the project: Execution. Following an initial setup, the first plugin implemented was WhatsApp. As mentioned earlier, this was a priority requirement and thus carried out first. Furthermore, different team members began working on separate plugins as follows: Microsoft Teams, Email, and SMS. Note that we quickly surpassed the expected goal of two plugins and were then able to focus on additional features. A simple front-end application was developed for demonstration purposes. Moreover, some extra components include Scheduled notifications, Workload limits, and Authentication. We also used Twilio to finish the list of suggested plugins in the project proposal. An inherent component of this phase was creating different kinds of tests and analyses to ensure the accuracy and quality of our work. Finally, besides the continuing improvement of existing plugins, we worked on deploying the application to Kubernetes.

In the closing stage of the project, we leveraged feedback from the demo to refine and fine-tune the product, incorporating valuable suggestions to improve user experience.

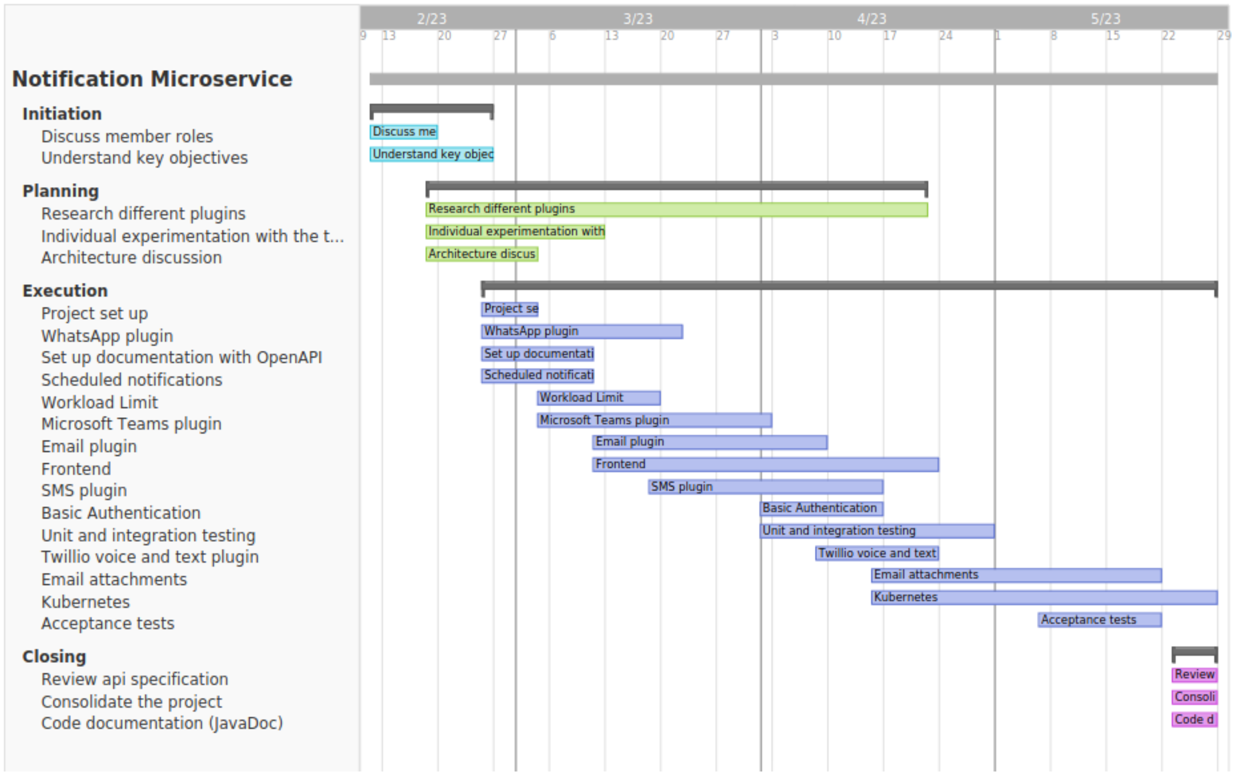


Figure 1: Gantt chart.

3 Development of solution

The final product consists of a Kubernetes-ready microservice that allows the sending of notifications through different types of channels. This section will go over our implemented solution and explain both the initial requirements and our implemented solution.

3.1 Requirements

After the first couple of meetings with the stakeholders, we outlined the requirements for the project. The key specifications are listed as follows:

- Implement at least two plugins with one of them being WhatsApp. A plugin, in this context, is an integration with a communication platform.
- Allow for the selection of the plugin to use. The service should provide a clear way to select the communications channel(s) used to send the message.
- Should support the use of WhatsApp groups. This is a fundamental specification of the project and should be taken into account while researching APIs to interact with the platform.
- Requests should expect a variety of parameters to easily customize alerts. These include message templates, the date to send, and the attachment of files.
- Return the execution result - the outcome can be classified as either a success or a failure, depending on whether the message was successfully sent or the notification was effectively created.

We are pleased to report that all of the mentioned functional requisites were fulfilled. As this goal was achieved, we shifted our focus to additional features that carried value for the project's stakeholders. These are enumerated below.

- Scheduling of notifications. This can be useful when a service is expected to be down at a certain time.
- Comprehensive list of supported plugins: Microsoft Teams, E-mail (SMTP), SMS with SMPP server, SMS and voice calls with Twilio.
- Basic authentication with user and password in the request, which can be changed to suit Altice Labs' own authenticator.
- Different kinds of WhatsApp messages: Media, Location, Links, Text. This makes sure a user does not lose the platform's functionality by using our service.
- Message templates to personalize the visual of the notification for some plugins. As an example, a user can select a previously built-in structure for an email that aligns with the message's context.
- A simple front-end interface to interact with the service. This was created for demonstration purposes only, as the service is not targeted for direct human interaction.

It is also crucial to highlight the inclusion of non-functional requirements, which are equally important. All these initial requirements were accomplished, as listed below.

- The project should be developed using the Quarkus framework.
- Ready to be distributed with Kubernetes.
- Usage of CI/CD pipelines.

Note that a detailed overview of the technologies is outlined below, in section 3.3.

In this matter, it is also worth mentioning that we have added some important requirements, some arose from our weekly discussions with the proponent:

- Documentation, using Guava for generating Javadoc.
- Selenium test using the front-end.
- Health checks and metrics for the availability and proper functioning of various components.
- Swagger UI - The documentation for the created Rest API.
- Workload limit provider to prevent abuse of the API usage.

Table 1: Advantages and Disadvantages of each Architecture

Strategies	**Advantages**	**Disadvantages**
Central endpoint	• Easier to use for simple interactions • Simplifies the implementation of generic plugin features • Easier to send multiple channel messages	• More convoluted implementation • Harder to scale • Harder to use the specificity of each plugin
Plugin specific endpoints	• More modular approach • Easier to interact with plugin-specific functions	• Brings some implementation overhead • Hinders multiple plugin interaction

3.2 Architecture

Regarding the architecture, we had two different approaches to consider. One option was to create separate endpoints for each plugin, while the other approach involved designing a single endpoint that would serve as a gateway for all the plugins.

Undoubtedly, both approaches had their advantages and disadvantages. Our team and the proponents discussed this topic, marking the advantages and disadvantages of each one as listed in Table 1. Because both alternatives provided strong advantages that we could not let go, we decided on a hybrid approach: keep the central endpoint to facilitate multiple plugin usage while at the same time providing plugin-specific endpoints that give the user more versatility to explore specific functions. From a technical standpoint, merging the strategies did not create a big overhead as most code could be shared. The way we ensured this required some technical expertise as depicted hereafter.

Our idea for the unique endpoint was to design it in a way that would permit the re-utilization of the code in other products and the rapid ease of use for the user. At the core of our implementation, we used the Decorator Design pattern [2]. This provided the capability to maintain the independence of each plugin while at the same time making it easier to combine them into a single component.

In order for the endpoint to know which plugins to call, we made use of a factory implementation. It receives a list of strings, representing the notifications services (or plugins) and transforms it into the wrapped notification object. This leaves space for future work ideas, we could for instance create a different logic of call, calling only the service if the wrapped object failed to send the notification. However, we decided not to explore those ideas as they don't seem to be of great value for the project in question and that logic can be easily passed to the client of the product. The architecture of the notifier is depicted as described in Figure 2.

3.3 Technologies

All the notifications created are stored in a database. For the Microsoft Teams plugin and the WhatsApp plugin, we also saved the groups available to send messages. The database used was PostgreSQL, which is run in a separate Docker container from the service.

We have employed a diverse range of technologies, which will be enumerated beneath.

Quarkus Lightweight and fast Java framework designed for cloud-native and container-based applications.

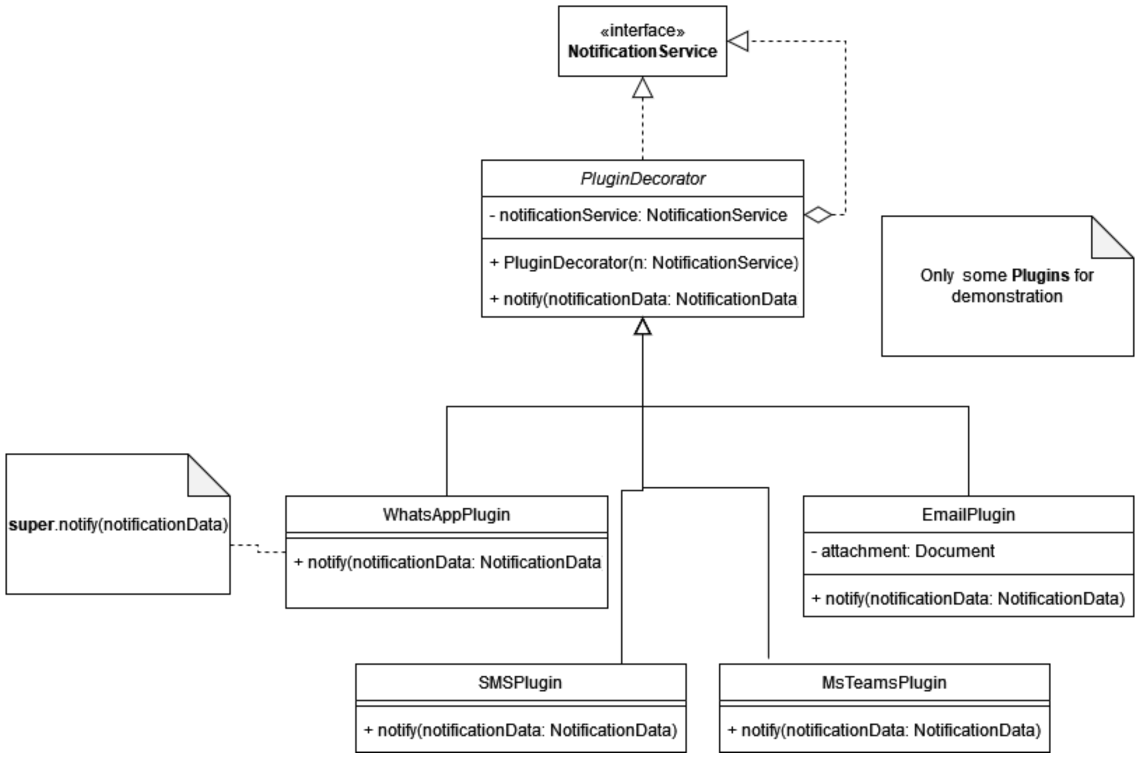

«interface»
Notification Service
PluginDecorator
- notificationService: NotificationService
+ PluginDecorator(n: NotificationService)
+ notify(notificationData: NotificationData
Only some Plugins for demonstration
super.notify(notificationData)
WhatsAppPlugin
+ notify(notificationData: NotificationData
EmailPlugin
- attachment: Document
+ notify(notificationData: NotificationData)
SMSPlugin
+ notify(notificationData: NotificationData)
MsTeamsPlugin
+ notify(notificationData: NotificationData)

Figure 2: Architecture of the Notifier.

Kubernetes Open-source system for automating deployment, scaling, and management of containerized applications.

Hibernate ORM that provided an object-oriented approach to interact with relational databases.

PostgreSQL Relational database used.

Selenium Used to test through the front-end interface.

Junit 5 Facilitated the creation and execution of automated tests to ensure software quality.

SonarCloud Employed to scan for vulnerabilities, code smells, etc.

REST Used both on our API implementation and by interacting with the set of public APIs supporting our plugins: WhatsApp, Twillio, etc.

Docker It simplifies the process of building, packaging, and distributing software, making it easier to deploy applications across various environments. Used to run the postgreSQL database.

3.4 Solution developed

As previously stated, the primary objective of this project revolves around enhancing the real-time communication capabilities of Altice Labs' Operations Support Systems (OSS), specifically regarding network or service-related issues. Accordingly, our project is specifically designed to facilitate machine-to-machine communication rather than human interaction. This means that we did not end up with a very visual product, making it harder to showcase. Therefore, some API endpoints will be described in this section. To get a better insight into the latter, please consult Annex A where we outlined the OpenAPI documentation. Furthermore, we will also provide example requests to certain endpoints in a JSON format.

Through the careful development of custom plugins compatible with diverse platforms a cohesive communication framework is established within Altice Labs' product ecosystem. This section presents a comprehensive analysis of the solution, delving into its tangible applications and the potential it holds for reshaping Altice Labs' operational landscape.

Usage

The configuration of all plugins can be conveniently managed through environment variables. Configuration plays a vital role in systems of this nature, making it an essential aspect valued by the stakeholders.

As previously mentioned, the microservice incorporates a basic authentication mechanism that facilitates the mapping of notifications to the respective requesting users. Consequently, in order to initiate any plugin call, it is imperative to provide proper identification credentials.

To simplify the presentation and demonstration of our project to others, we opted to develop a user-friendly front-end application, in accordance with Figure 3. The application is solely intended for demonstration purposes, serving no other functional use. Thus, it is important to note that not all features of the plugins are accessible through the front-end application, which is available on `/view`.

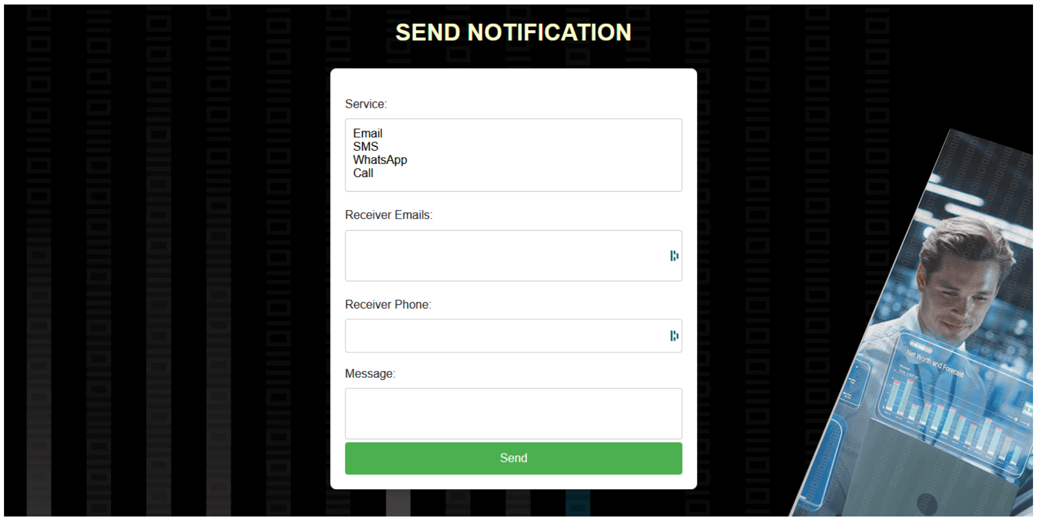


Figure 3: Demonstration web page

WhatsApp Plugin

The central focus of the project revolved around the development of the WhatsApp plugin. Our primary task was to identify suitable APIs that would enable the sending of group messages. However, this requirement significantly limited the available options as the official Meta API is very limited. After researching the topic, we narrowed down our choices to two APIs that met the requirements: Maytapi and Twilio. Ultimately, we opted to utilize Maytapi due to its additional set of features that offered exciting possibilities to explore. One such feature was the ability to send locations, which was deemed highly valuable for this product. Consider the scenario of a product transmitting its location to a repairman, enabling swift and efficient troubleshooting and resolution.

Maytapi imposes a monthly fee of 30 euros per phone, granting a generous allowance of 6,000 messages per day, surpassing our project's requirements. However, one of the challenges we encountered during development was the substantial cost associated with most of the APIs employed. Due to the pricing constraints, we resorted to leveraging the offered free trial by Maytapi, which spans a duration of three days.

Listing 1 showcases an example of how to send a WhatsApp message using the `/notifier(POST)` endpoint:

Listing 1: Sample usage of notifier for WhatsApp

```
{
  "notificationServices": ["whatsapp"],
  "message": "This is a service alert!",
  "receiverGroup" : "4",
}
```

It's worth mentioning that the `notificationServices` parameter presented in the previous request body enables the user to send notifications using various plugins. For instance, sending the same message to a WhatsApp group and to an email (Listing 2).

Listing 2: WhatsApp and email notification

```
{
  "notificationServices": ["whatsapp", "email"],
  "receiverEmails": ["up202000000@g.uporto.pt"],
  "message": "This is a service alert!",
  "receiverGroup" : "4",
}
```

Sending a WhatsApp text message can also be achieved using the `/whatsapp/message/text` endpoint. In this scenario, the JSON data required is even simpler, in accordance to Listing 3.

Listing 3: Simpler whatsApp JSON data

```
{
  "message": "This is a service alert!",
  "receiverGroup" : "4",
}
```

The aforementioned JSON data (Listing 3) sends a message to a group named ”Grupo Altice Labs”. As per the stakeholders' request, teams can be created using the appropriate endpoint. Additionally, it is possible to include or remove individuals from the group.

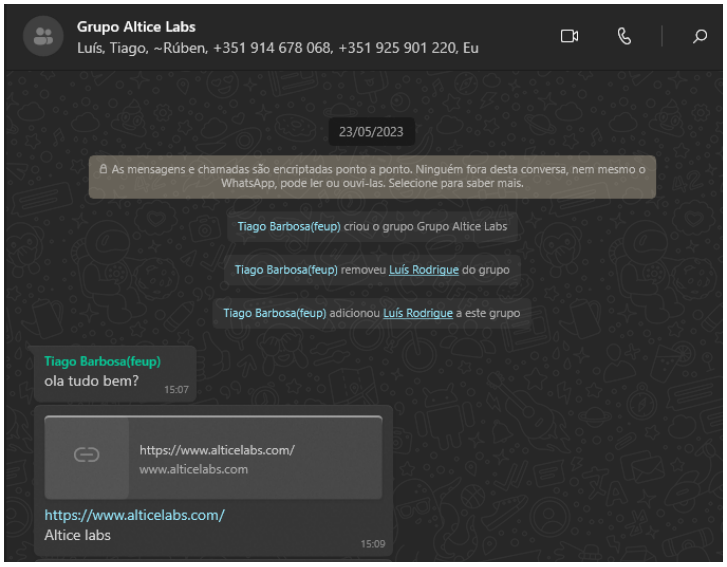
The usage of IDs to identify the respective groups was a requirement put forth by the project stakeholders to ensure clarity and consistency when using the diverse plugins. To create the previous group, a simple request can be made (Listing 4).

Listing 4: Creating a group request body

```
{
    "phoneList": ["351987654321","351123456789"],
    "groupName" : "Grupo Altice Labs"
}
```

To put it all together, Figure 4 illustrates the result from the combined features mentioned above.

As previously stated, this plugin has several additional features that are currently accessible. Due to the comprehensive nature of the API specifications associated with these features, including them in their entirety within this report would prove exceedingly extensive. Consequently, it is strongly advised that the reader consults the Swagger UI (in annex A), which provides a thorough elucidation of each parameter pertaining



Grupo Altice Labs
Luís, Tiago, ~Rúben, +351 914 678 068, +351 925 901 220, Eu

23/05/2023

As mensagens e chamadas são encriptadas ponto a ponto. Ninguém fora desta conversa, nem mesmo o WhatsApp, pode ler ou ouvi-las. Selecione para saber mais.

Tiago Barbosa(feup) criou o grupo Grupo Altice Labs

Tiago Barbosa(feup) removeu Luís Rodrigue do grupo

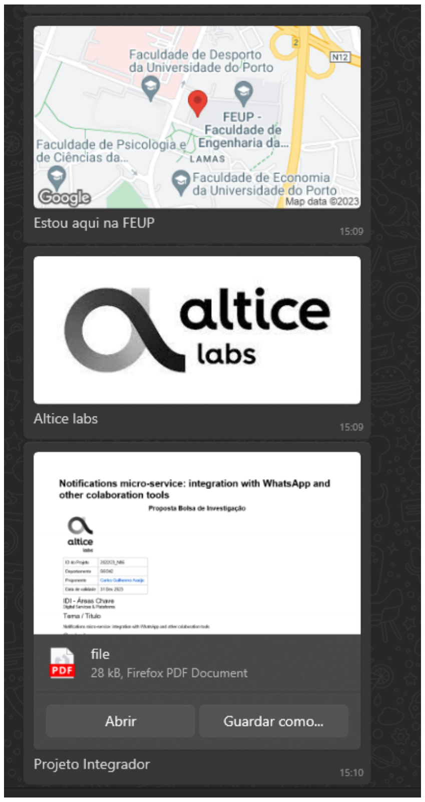
Tiago Barbosa(feup) adicionou Luís Rodrigue a este grupo

Tiago Barbosa(feup)
ola tudo bem? 15:07

https://www.alticelabs.com/
www.alticelabs.com

https://www.alticelabs.com/
Altice labs 15:09

(a) Group management and Message sending.


Faculdade de Desporto da Universidade do Porto
FEUP - Faculdade de Engenharia da...
Faculdade de Psicologia e de Ciências da...
LAMAS
Faculdade de Economia da Universidade do Porto
Google
Map data ©2023
altice labs

Estou aqui na FEUP 15:09

Altice labs 15:09

Notifications micro-service: integration with WhatsApp and other colaboration tools

file
28 kB, Firefox PDF Document

Abrir
Guardar como...

Projeto Integrador 15:10

(b) Specific media messages

Figure 4: Showcase of implemented WhatsApp interaction

to these features. Figures 10 and 11 illustrates a selection of those features, which are accessible for both individual and group messaging.

Microsoft Teams Plugin

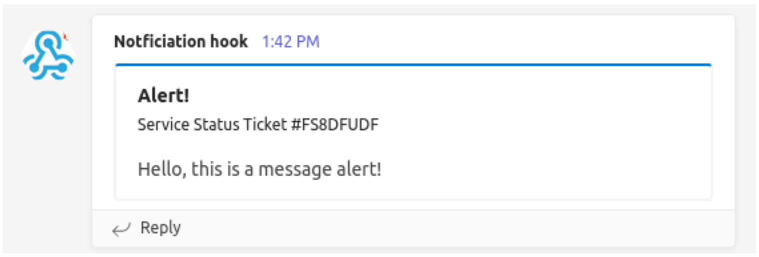
The Microsoft Teams plugin implementation turned out to be challenging. By nature, this is not a very permissive service as it relies on private groups and invitations to allow for communication. Our first attempt at integrating the platform was through the Azure API. We soon realized that it did not provide the necessary functionalities required. Luckily, Microsoft provides a Webhook interface for MsTeams' channels that was a good fit for our use case. To use this, the user should first set up a URL from the platform. Then, it should be added to the service's database, so it can be used to send messages to a specific channel. This plugin allows the usage of message templates. Further details are described on Swagger UI, such as a usage guide. A sample of usage is portrayed on Listing 5, along with the result in Figure 5.

Email Plugin

For the email plugin, we used the Quarkus mailer and configured it to use Gmail's SMTP server. As mentioned earlier, the SMTP server configuration can be conveniently managed with environment variables. Within this plugin, we had the option to choose between a synchronous or asynchronous mailer service. We opted to use Reactive Mailer, an asynchronous mailer service. Hence, the user won't receive immediate confirmation of

Listing 5: Sending template message on microsoft teams

```
{
    "receiverTeams":[3],
    "message":"Hello, this is a message alert!",
    "ticketId":"FS8DFUDF",
    "template":"ticket"
}
```


Notficiation hook 1:42 PM

Alert!

Service Status Ticket #FS8DFUDF

Hello, this is a message alert!

Figure 5: Message sent

whether the email was successfully sent. Although this is a limitation, we think that it is a good assumption and highly improves performance.

To introduce an additional set of features to this plugin, we incorporated the capability to send emails with templates (Qute templates). An interesting fact learned about CSS in emails is that in order for it to be rendered correctly, the CSS styles must be inline within the email content. Listing 6 illustrates the request body utilized for transmitting an email. The template used can be seen in Figure 6.

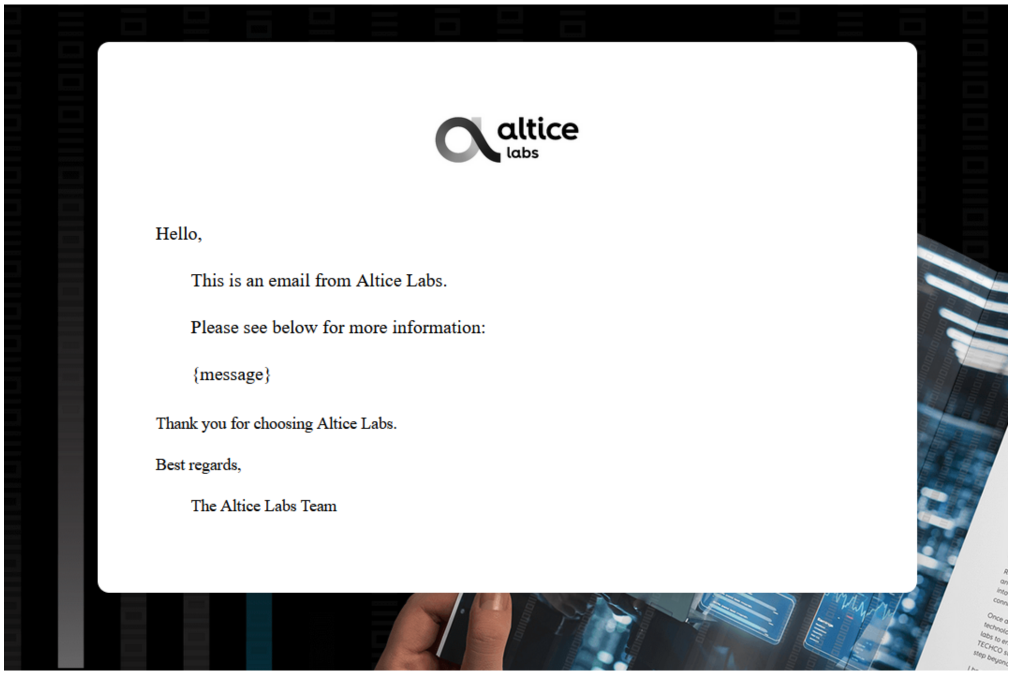
Listing 6: Send an Email using a template

```
{
  "notificationServices": ["email"],
  "notificationData": {
    "mailTemplateId" : "1",
    "dateToSend": "2023-05-24T16:59:00",
    "receiverEmails": ["up202000000@g.uporto.pt"],
    "subject": "Monty Python is good",
    "message": "Ola isto e um email",
    "attachments": [...]
  }
}
```

It is worth mentioning the `subject`, `mailTemplateId` and `attachments` parameters. Three specific parameters of the mail plugin.

Another noteworthy aspect is the presence of the `dateToSend` parameter, which relates to the previously discussed feature of message scheduling. The schedule of notifications is available for all plugins and raised a few questions. For instance, what happens when the Kubernetes node goes down? Does the scheduling job survive? And how does the client know if the message was successfully sent?

To solve the first issue the application rescheduled unsent scheduled messages on startup, after a failure or shutdown, by looking at the correspondent `dateToSend` parameter and verifying if the date as passed. Obviously, this raises another question, if the notification was not sent on the appropriate time should we send it nevertheless? By default, we decided not to send it, however, we see it as a possible future feature.


Hello,

This is an email from Altice Labs.

Please see below for more information:

{message}

Thank you for choosing Altice Labs.

Best regards,

The Altice Labs Team

Figure 6: An email template

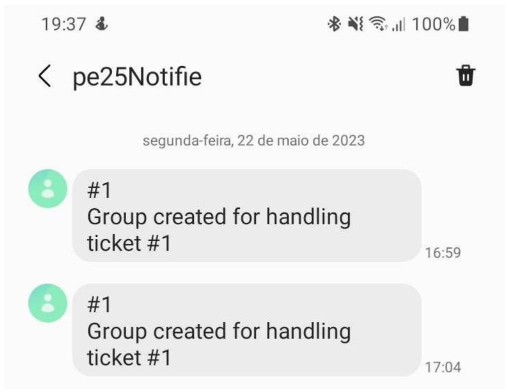
The second problem is harder to resolve, one should know how to connect to the product that emitted the ticket in order for it to be addressed, possibly using push notifications. Alternatively, instead of relying on push notifications, the client can fetch the microservice after the designated send time. For example, asking if a notification due 5 minutes ago has been sent or not. However, this is not a definitive solution. The client can be occupied with other tasks or even shut down. Every client can check the notifiers created by checking the `/notifier` endpoint (`GET`). It is also possible to check the scheduled notifications with `/notifier/toSend` (`GET`)

SMPP Plugin

As the name indicates, this plugin uses the SMPP protocol to send text messages. To facilitate the implementation of this plugin we used JSMPP, a library that implements the SMPP protocol in Java. All the parameters passed to the protocol are configurable environment variables as requested by the stakeholders. To test the plugin, we made use of a free SMPP server. The host and port are also configurable, thus Altice only has to change the appropriate parameters to run this plugin with their own server. Refer to Listing 7 and Figure 7 for an example of the usage of this plugin.

Listing 7: Send an SMS

```
{
  "notificationServices": ["sms"],
  "notificationData": {
    "phoneList": ["351987654321","351123456789"],
    "message": "#1 Group created for handling ticket #1",
  }
}
```


19:37

100%

pe25Notifie

segunda-feira, 22 de maio de 2023

#1
Group created for handling ticket #1
16:59

#1
Group created for handling ticket #1
17:04

Figure 7: Some messages using a SMPP server

Twillio Plugins

Research and experimentation played significant roles in our project. With that in mind, we decided to explore the APIs from Twillio. Namely the SMS and voice call APIs. To utilize these APIs, we obtained a free trial number, which served as the source for sending SMS messages and making voice calls. It's important to note that these plugins may hold lower priority for a company like Altice Labs, as they likely have their own SMPP servers and voice call services in place. Thus, there is no need to rely on a costly external service.

The request body is similar to the SMPP server and hence won't be demonstrated here. For more information refer to the Swagger UI, annex A, or Javadoc documentation.

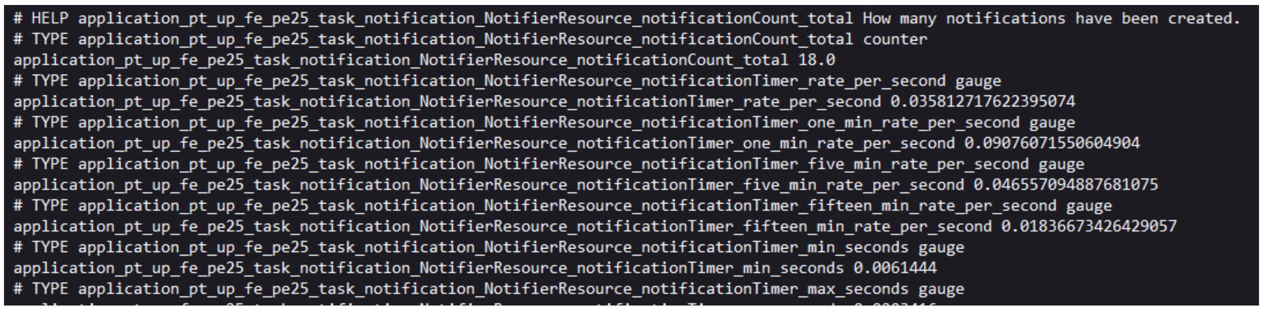
3.5 Validation

Purpose of Validation

Validation was a way to ensure that our project complied with the requirements and expectations that had been set. This entailed observing any particular requirements specified in the project's goals or specifications, as well as industry best practices, coding conventions, and security measures. We could spot any deviations from these standards through thorough testing and validation and fix them to guarantee that the system satisfied the desired quality and was in line with the project's stated goals.

Please note that we have not conducted any scalability tests for our product. Given the reliance on external APIs, performing load testing would be both expensive and impractical. Although we could have simulated API calls for the testing process, we chose not to do so.

To consistently monitor the up-time and performance of our service, we use SmallRye Metrics. This enables us to measure the response time of our endpoints to various calls and effectively identify any potential issues within our product. Figure 8 shows an example of these metrics. SmallRye provides various metrics, but we will only highlight a few. Some of the metrics used seem interesting. For instance, it offers percentile measurements, such as determining the duration at which 95 percent of the measurements were faster.



```
# HELP application_pt_up_fe_pe25_task_notification_NotifierResource_notificationCount_total How many notifications have been created.
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationCount_total counter
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationCount_total 18.0
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_rate_per_second gauge
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_rate_per_second 0.035812717622395074
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_one_min_rate_per_second gauge
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_one_min_rate_per_second 0.09076071550604904
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_five_min_rate_per_second gauge
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_five_min_rate_per_second 0.046557094887681075
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_fifteen_min_rate_per_second gauge
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_fifteen_min_rate_per_second 0.01836673426429057
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_min_seconds gauge
application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_min_seconds 0.0061444
# TYPE application_pt_up_fe_pe25_task_notification_NotifierResource_notificationTimer_max_seconds gauge
```

Figure 8: A small set of SmallRye metrics

Quality Assurance

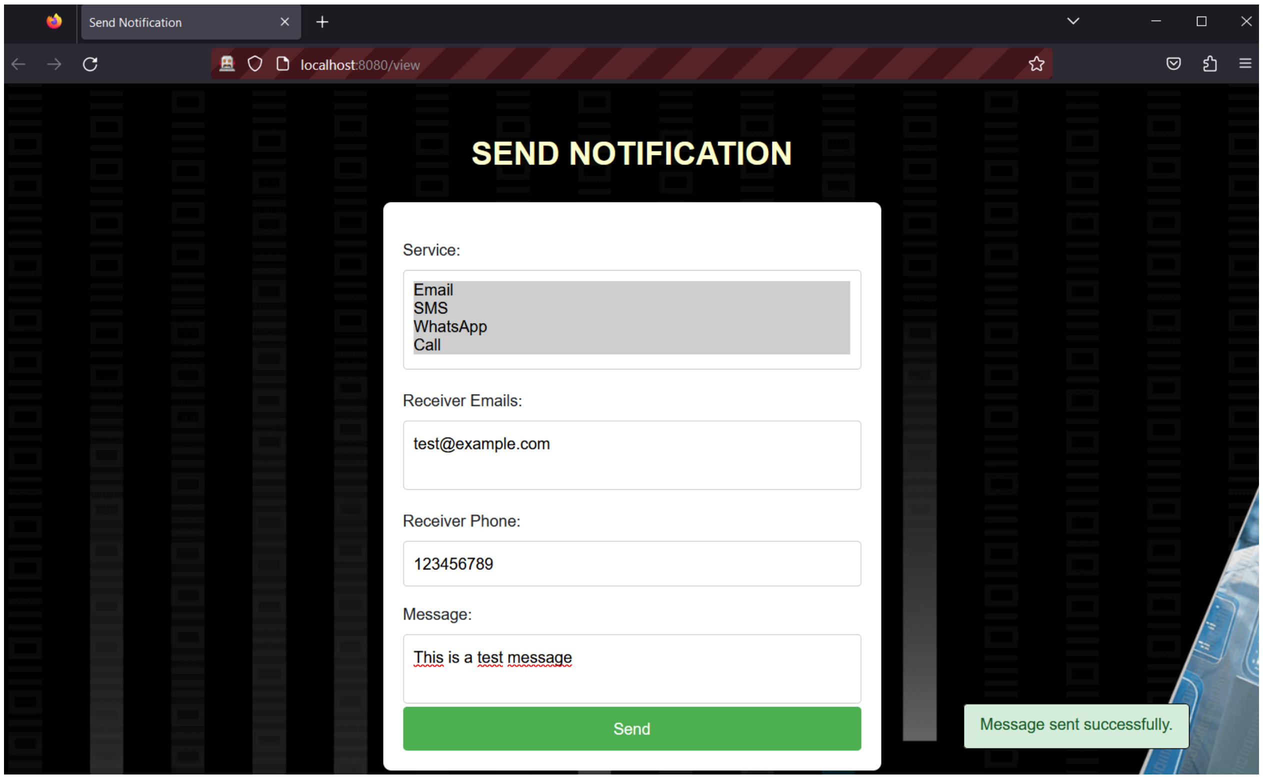
For the development process to produce stable and maintainable software, maintaining code quality and spotting possible problems early on were essential. By automating code analysis and delivering useful insights, we integrated SonarCloud with GitHub in our project to improve quality assurance.

SonarCloud was able to automatically analyze the code in each pull request. It discovered code smells, coding standards violations, probable flaws, and vulnerabilities through static code analysis. This automatic analysis gave us quick feedback, enabling us to fix problems before merging our code.

We were also able to identify code quality problems and vulnerabilities early in the development process. By identifying issues early on, we were able to maintain code quality across the whole codebase and avoid the accumulation of technical debt. The work needed to resolve problems later in the development cycle was lessened because of this proactive approach.

Testing and Validation Techniques

We used a thorough testing strategy that comprised both unit testing and integration testing. For some components, unit tests were made, concentrating on isolated functionality. To ensure smooth communication, integration tests were carried out to validate interactions and integrations with external systems (only done in the WhatsApp plugin). We made sure our project was accurate and dependable by using testing frameworks like JUnit and Selenium.



SEND NOTIFICATION

Service:
Email
SMS
WhatsApp
Call

Receiver Emails:
test@example.com

Receiver Phone:
123456789

Message:
This is a test message

Message sent successfully.

Figure 9: Selenium test

Security and Authorization

We put in place fundamental systems for authentication and authorization in order to guarantee secure communication and safeguard sensitive data. Role-based access control was implemented through our integration with the JPA identity provider.

The reliance of our product's system security CIA triad (Confidentiality, Integrity, and Availability) on external APIs or services/servers is a significant aspect to consider. Therefore, it is vital that these services exhibit a high level of confidence and can be securely used on a large scale. In this regard, we have made efforts to utilize well-known services within the community, such as Maytapi, Twilio, Gmail, and others, as they are perceived to be more reliable and trustworthy compared to lesser-known alternatives.

Validation with the stakeholders

The product was continually validated with the stakeholders. Each requirement was demonstrated to the company mentor when implemented, ensuring we had a quick feedback and were able to move forward. As an example, when the WhatsApp plugin was ready, we tested it during the meeting by exchanging messages, ensuring it was according to the requirements. When we were nearing the final delivery, a demonstration of the product as a whole was conducted with the proponent. At this point, we got a second appraisal and were able to both confirm the implementation and find missing details that were fixed thereafter.

4 Conclusions

4.1 Achieved results

The notification microservice has been successfully developed, representing a significant achievement for the project. It was designed to address the challenge of providing real-time notifications to users informing them of network and/or service problems.

One of the main achievements was the integration of various methods of sending notifications, comfortably surpassing the initial proposal of two plugins. In addition to integration with WhatsApp, which is a widely used communication channel, other popular means such as Microsoft Teams, email, SMS, and voice calls were also incorporated. For each method, plugins have been created that allow the choice and proper configuration according to the specific needs of each situation. The plugins were developed taking into account the best practices of integration with each platform, ensuring a smooth and efficient notification delivery experience.

An important aspect of the final product is the flexibility offered to users in defining the recipients of the notifications. Notifications can be sent to a single recipient or to groups, allowing an efficient approach to communicating with different stakeholders. The additional features implemented ensure the adaptability of the microservice to the specific needs of each type of notification.

The implementation of was done following best practices in microservice architecture and considering its cloud-native nature. It was distributed in containers and ready for Kubernetes deployment, which offers scalability, robustness, and simplicity of management in many contexts, including on-premises and public clouds. The microservice has also been integrated with CI/CD pipelines, enabling static code analysis, security vulnerability scanning, and end-to-end testing. This integration provides an automated and efficient approach to development, ensuring code quality and service stability.

In conclusion, we have successfully met all the specified requirements and even exceeded them to some extent. The ability to send notifications through several popular channels, along with advanced customization features and the adoption of modern architecture, make the microservice a valuable contribution to the enhancement of telecommunications services and the improvement of the quality of service provided to society.

The individual contributions of each member are depicted in Table 2. These were evaluated both qualitatively and quantitatively, ensuring that each team member played a meaningful role and fulfilled their assigned responsibilities. These achievements and individual contributions are evidence of the project's success and of the team's collective effort to achieve the proposed goals.

Table 2: Individual member contributions

Team Member	**Contribution (%)**
José Luís Cunha Rodrigues	25%
Ricardo André Araújo de Matos	25%
Rúben Costa Viana	25%
Tiago Filipe Magalhães Barbosa	25%

4.2 Lessons Learned and Their Impact on Project Development

While this project was being developed, many lessons were discovered that benefited the team and improved the knowledge they possessed. Some of the key takeaways are listed below.

The significance of teamwork The success of the project depended on the team's ability to work together. Through efficient communication, knowledge sharing, and mutual support, it was possible to face the challenges more effectively and achieve satisfactory results.

Mastery of technologies and tools The project required mastery of various technologies and tools, such as Java, Quarkus, Docker, Kubernetes, and others related to microservices development and integration with communication platforms. We learned how to use these technologies properly, exploring their functionalities and understanding their advantages and disadvantages.

Flexibility and adaptation During the development process, unexpected challenges arose that required flexibility and adaptation. We learned to deal with changing requirements, technical problems, and other unforeseen situations, seeking creative solutions while maintaining focus on the project's objectives. The usage of an agile methodology was very important for the success of the assignment.

Good development practices It was essential to apply good software development practices, such as the use of design patterns, integrated testing, and static code analysis. Through these practices, we were able to ensure code quality, system maintainability, and early detection of potential problems.

Importance of documentation and knowledge sharing During the project, we recognized the importance of properly documenting the work performed and sharing the knowledge acquired. This facilitated collaboration among team members, allowing a better understanding of the project and assisting in future maintenance or improvements.

These lessons contributed to the professional and personal growth of each team member, strengthening technical skills and promoting greater awareness about the importance of collaboration, flexibility, and the application of best practices in software project development.

By reflecting on the lessons learned, we can use this acquired knowledge in future projects, avoiding past mistakes and always seeking to improve our efficiency and quality of delivery. Through this continuous learning process, we are prepared to take on new challenges and make even greater contributions to the accomplishment of upcoming initiatives.

4.3 Future Work

Upon completion of this project, we identified some ideas for improvements and future work that could be explored to further enhance the product. These ideas can serve as guidelines for future iterations of the project or as suggestions for possible related projects:

Additional Plugins Although the microservice currently supports various sending methods, such as WhatsApp, Microsoft Teams, email, SMS, and voice calls, there is a possibility to further expand the list of plugins. For example, we could consider integration with other collaboration platforms or messaging services, allowing users to choose their communication preferences.

Improved configuration flexibility Currently, the microservice supports the selection of plugins, recipient list, and optional parameters. However, we could explore additional configuration options, such as notification priorities or advanced message format customization. This would allow further adaptation to the specific needs of each use case.

Enhanced monitoring and metrics Consider including monitoring and metrics collection capabilities in the microservice. This would allow monitoring of the performance, usage, and effectiveness of the notifications, providing valuable insights for continuous improvements.

Integration with alerting and incident management systems Explore the integration of the notifications microservice with existing Altice alerting and incident management systems. This would make it easier to synchronize notifications with the incident handling workflow, enabling a faster and more efficient response to identified problems.

Additional testing and security enhancements To guarantee the resilience and dependability of the microservice, invest in more thorough testing, including more unit, integration and load tests. To safeguard the integrity and confidentiality of the transferred information, we should also strengthen our security measures like the existing authentication and encryption.

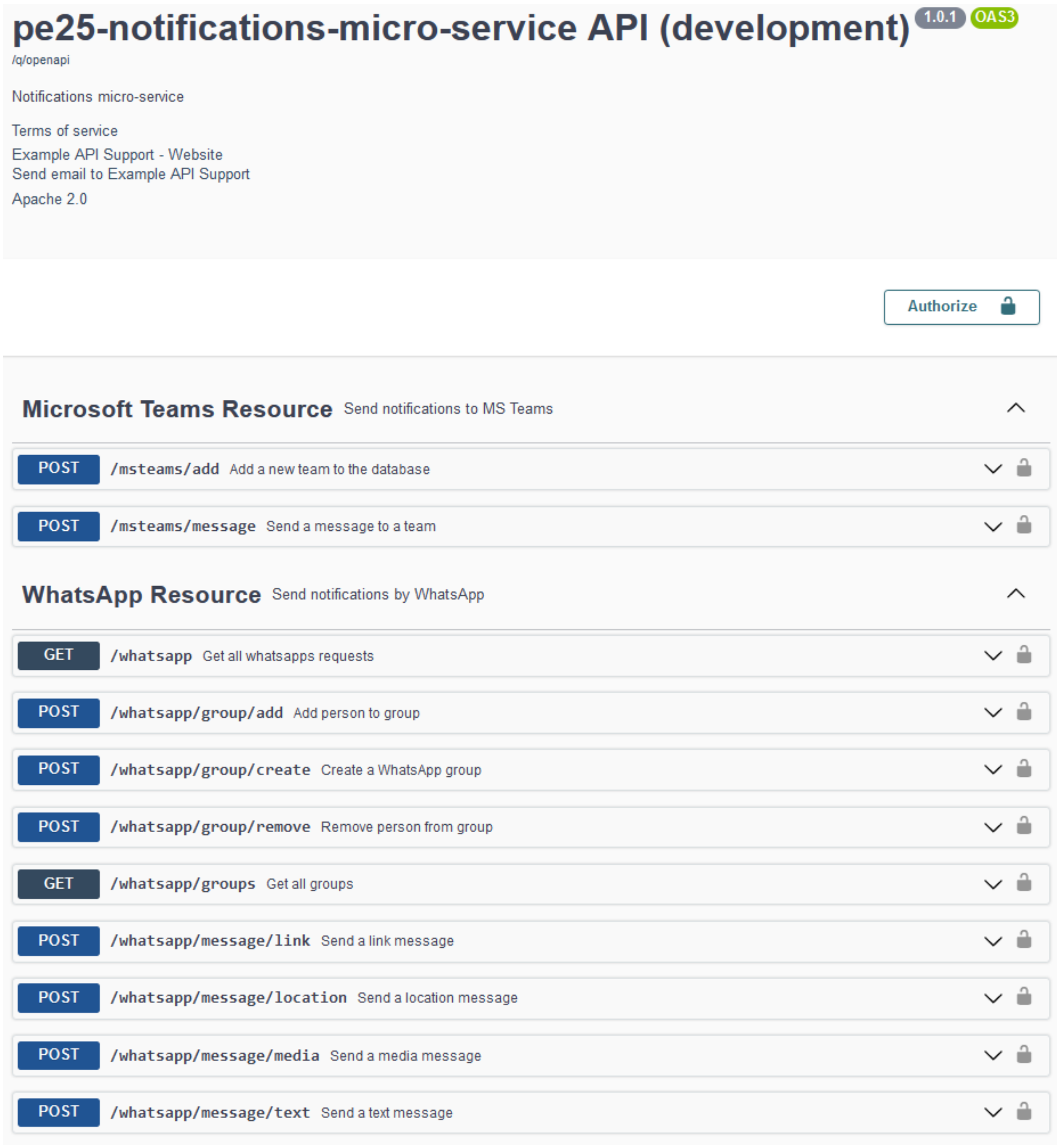
These are just a few ideas for further projects that might help the product's improvement. Each of these concepts may be investigated in further detail while taking into consideration the unique demands of the company and the users.

References

[1] Paul Kirvan. operational support system (OSS). *Networking*, 5 2023.

[2] Refactoring.Guru. Decorator. `https://refactoring.guru/design-patterns/decorator`. [Online; accessed on June 19, 2023].

A Open API



pe25-notifications-micro-service API (development) 1.0.1 OAS3

/q/openapi

Notifications micro-service

Terms of service

Example API Support - Website

Send email to Example API Support

Apache 2.0

Authorize

Microsoft Teams Resource Send notifications to MS Teams

POST `/msteams/add` Add a new team to the database

POST `/msteams/message` Send a message to a team

WhatsApp Resource Send notifications by WhatsApp

GET `/whatsapp` Get all whatsapps requests

POST `/whatsapp/group/add` Add person to group

POST `/whatsapp/group/create` Create a WhatsApp group

POST `/whatsapp/group/remove` Remove person from group

GET `/whatsapp/groups` Get all groups

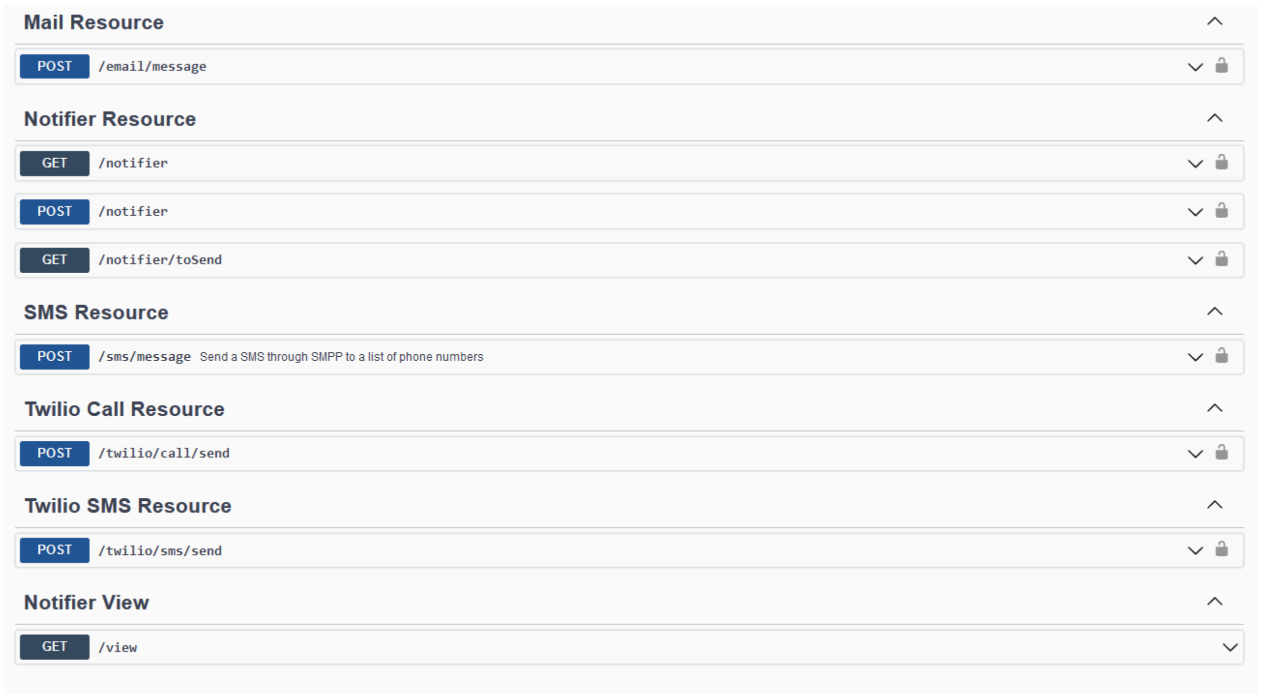
POST `/whatsapp/message/link` Send a link message

POST `/whatsapp/message/location` Send a location message

POST `/whatsapp/message/media` Send a media message

POST `/whatsapp/message/text` Send a text message

Figure 10: OpenApi endpoints 1



Mail Resource

POST /email/message

Notifier Resource

GET /notifier

POST /notifier

GET /notifier/toSend

SMS Resource

POST /sms/message Send a SMS through SMPP to a list of phone numbers

Twilio Call Resource

POST /twilio/call/send

Twilio SMS Resource

POST /twilio/sms/send

Notifier View

GET /view

Figure 11: OpenApi endpoints 2