BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

SYSTEM FOR MANAGING CORPORATE DATA INTEGRATION

SYSTÉM PRO SPRÁVU FIREMNÍ DATOVÉ INTEGRACE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR AUTOR PRÁCE	MICHAL ŘEZNÍK
SUPERVISOR VEDOUCÍ PRÁCE	Ing. RADEK KOČÍ, Ph.D.

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148861

Institut:	Department of Intelligent Systems (UITS)
Student:	**Řezník Michal**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**System for Managing Corporate Data Integration**
Category:	Information Systems
Academic year:	2022/23

Assignment:

1. Study the issue of creating scalable applications based on the principles of middleware, service-oriented architecture, microservices, etc. Learn about the Spring framework and the ReactJS UI library.
2. Analyze the data integration needs in a real company and learn about the current state of the art.
3. Suggest necessary modifications, mainly to introduce user control over data flows.
4. Implement the proposed modification using Java, Spring Framework, and ReactJS technologies.
5. Test the resulting system and discuss its possible further development.

Literature:

- M. Deinum, D. Rubio, J. Long. Spring 5 Recipes: A Problem-Solution Approach, 4th Edition. Apress, 2017. ISBN-13: 978-1484227893
- ReactJS: Getting started. Online, https://reactjs.org/docs/getting-started.html, září 2022.

Requirements for the semestral defence:
The first two points.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Kočí Radek, Ing., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	3.11.2022

Abstract

This thesis aims to design and implement an application interface that will become part of a micro application on the Onsemi integration platform. The thesis introduces the strategies used during the development of the system, the tools used, and the system's design. The Onsemi middleware platform caters for more than 3000 data flows per day and ensures the corporation's successful day-to-day running. Middlware platform must gurantee data delivery for each integration. Some data integration flows can produce a token representing an outgoing datafile stored in the file system. The token represents this state by moving around the server's file system on which the application catering for data delivery is currently running. The main purpose of the interface is to make it easier for middleware team members to manage failed data integrations, which they must deal with along with developing the platform and providing new integrations. At the end of the thesis, the implementation in the company's system is described together with an evaluation of the benefits of the developed solution.

Abstrakt

Cílem této práce je návrh a implementace aplikačního rozhraní, který se stane součástí mikro aplikace patřící do integrační platformy firmy Onsemi. Práce se nejprve věnuje představení strategií používaných během vývoje systému, použitým nástrojům a návrhu samotného designu systému. Middleware platforma Onsemi obstarává více než 3000 datových toků denně a zajišťuje úspěšný každodenní chod korporátu. Tato platofrma garantuje doručení dat pro každou jednu integraci. Datové integrační toky mohou vytvářet token představující odchozí datový soubor uložený v souborovém systému. Každý jeden z daných toků vytváří během spuštění token, který se po dobu běhu toku stává jakousi reprezentací jeho aktuálního stavu. Token reprezentuje tento stav pomocí pohybu v souborovém systému serveru, na kterém zrovna běží aplikace obstarávající asynchronní doručení dat. Hlavním úkolem daného rozhraní je ulehčit správu neúspěšných datových integrací pro členy middleware týmu, kterými se musí zabývat společně s vyvíjením platfromy a zajišťováním nových integrací. Na konci práce je popsána implementace do systému firmy společně se zhodnocením přínosu vypracovaného řešení.

Keywords

user interface, web application, middleware, OpenApi, microservices, service-oriented architecture, ReactJS frontend, Spring Framework, Java

Klíčová slova

uživatelské rozhraní, webová aplikace, middleware, OpenApi, microslužby,
architektura orientovaná na služby, ReactJS frontend, Spring Framework, Java

Reference

ŘEZNÍK, Michal. *System for Managing Corporate Data Integration.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Kočí, Ph.D.

System for Managing Corporate Data Integration

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Radek Kočí, Ph.D. The supplementary information was provided by the Onsemi middleware team. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Michal Řezník
May 9, 2023

Acknowledgements

I want to thank my supervisor, Ing. Radek Kočí, Ph.D., for his guidance and willingness to help while writing this thesis. Furthermore, I would like to thank the company Onsemi, especially their middleware team, for their patience and for allowing me to work with them. And last but not least to my family and my girlfriend for their huge support.

Contents

Chapter 1

Introduction

Middleware, as such, is the layer that mediates communication. In practice, it is a bridge that can deliver data to the user or application from another application or system. Without middleware, a developer must create a special module for each component communicating with the application. In large companies, it is important for the overall operation because it mediates data transfers and the operation of distributed applications simultaneously. One of these companies is Onsemi.

Onsemi manufactures and supplies semiconductors and integrated circuits. It is an American company that has been growing rapidly in recent years. For this reason, it is also important that the middleware functions are as reliable as possible when acquiring or adding new parts to a system.

This work aims to design and implement an interface to the application that integrates the different data streams, allowing the user to manage the individual runs of these flows. It would save middleware team members' time, which can be spent on maintenance or extending the platform.

Change would increase the efficiency of daily operations that are currently hindered by the inability of users to manage files representing a queue with individual data flow runs. It means that users have to seek assistance from system administrators, who are members of the middleware team, whenever there is an issue. This disrupts both the users and the administrators work, with several minutes wasted daily managing individual runs and coordinating with flow owners. This fact does not fit the philosophy of the platform, which tries to be built on the principle of Platform as a Service, which means that the user should be able to perform all tasks without assistance.

To address this issue, an interface will be created for users to manage their queues from their flows on the file system. This solution will not only free up administrators time but also empower users to manage their own flows.

At the beginning of the work, to achieve the desired results, it is necessary to familiarize oneself with the platform and study the issues related to its development and functioning. After researching and creating a needs analysis, the next step is to create an interface design. The design is then implemented into the middleware of the platform. The thesis also discusses the testing methods used in the development process.

The ending includes an evaluation of the benefits to the company along with a statement of the degree of achievement of the objective.

Chapter 2

Analysis and Specification of Requirements

Requirements analysis and specification is a crucial first step in software engineering, which commences once the project's feasibility is established. This process involves communication with the customer for data collection, analysis, and correction. The output of this phase is typically a Software Requirements Specification (SRS) document [28], which system analysts usually prepare for larger projects.

In the case of Onsemi, the initial phase involved getting acquainted with the platform they had been developing for 12 years. This platform is an integral part of the company's system, and the company provided a complete confluence, including architecture descriptions and user tutorials mainly intended for managers. After getting up to speed with the necessary basics, I was assigned some aesthetic tasks, such as modifying the logging in some micro-applications or tweaking the frontend.

Once I had completed these initial tasks, the team architect presented me with the project brief. The brief initially contained an outline of the final product that would make the team members' jobs easier. After consultations and fine-tuning, this outline gradually evolved into a complete architecture design of the module [22]. To support this design, we created a use case diagram, which was consulted with various team members with different views on the interface design. After evaluating these views, we selected an interface design that delegated all responsibility and works to the data flow owner, despite concerns about their ability to manage the runs effectively. This was chosen over creating an admin-only interface that would only speed up server access and operations over tokens. From this diagram, we extracted the system requests and created a first draft of the interface design.

2.1 Current System

When developing a larger platform, it is crucial to set specific requirements. These requirements are related to both the functionality itself and the features that are not functional. In our case, we are trying to add a module to the Delivery engine microservice, which caters for delivery that transfers many data. Currently, there is a problem that has not been addressed due to the workload of the administrators, specifically token management. Tokens are small files that represent a given integration run. And if there's any problem, like with the queue. The user can't do anything about it and has to contact the middleware team that manages the platform. This thesis solves this problem. Choosing the proper requirements at the beginning of development is necessary to make everything work as it should. So that there is no affectation on the existing system and, at the same time, everything works as required.

2.2 Requirements

Each software requirement represents a task or condition imposed on the software [13], and requirements can be categorized in several ways. For this thesis, I have divided them into two categories: functional and non-functional, which represent different groups of requirements.

The process of requirements gathering and analysis involves several steps, starting with communicating with stakeholders, who are interested parties [29]. We gather information from stakeholders and analyze it to eliminate any unclear, incompatible, or conflicting requirements.

Once the requirements are recorded, we create documentation such as the above use case diagram. The final stage is to manage the requirements throughout their lifecycle, which may involve merging or modifying them.

Since this product will be used by management employees, developers, and other IT users, it was essential to consider their needs and design an interface that is simple and intuitive. After designing and discussing the use case diagram with users, we refined the requirements and categorized them as functional or non-functional.

2.2.1 Functional Requirements

The first group of requirements are functional requirements. These define the software's features to be developed along with the functions it will perform. It is, therefore, essential that they are clearly defined for the development team.

The following functional requirements were selected for this project to extend Delivery engine micro application of the Onsemi middleware platform:

For users:

- Display the current state of the file system in counts for the owned data flow.
- View tokens in a specific state (folder) for a specific (owned) interface.
- Perform operations on the token list, such as moving and deleting.
- Get a specific token and the ability to work with it as needed. This includes resetting the number of launch attempts, moving the token, or deleting it.

Administrators can perform the same operations as users. The only change is that they are not limited to their own data flows but can manage all of them in a given environment.

In order to display these requirements as simply as possible, a use case diagram was created that corresponds to the system's functional requirements. This diagram was used as an outline for creating individual endpoints in the API.

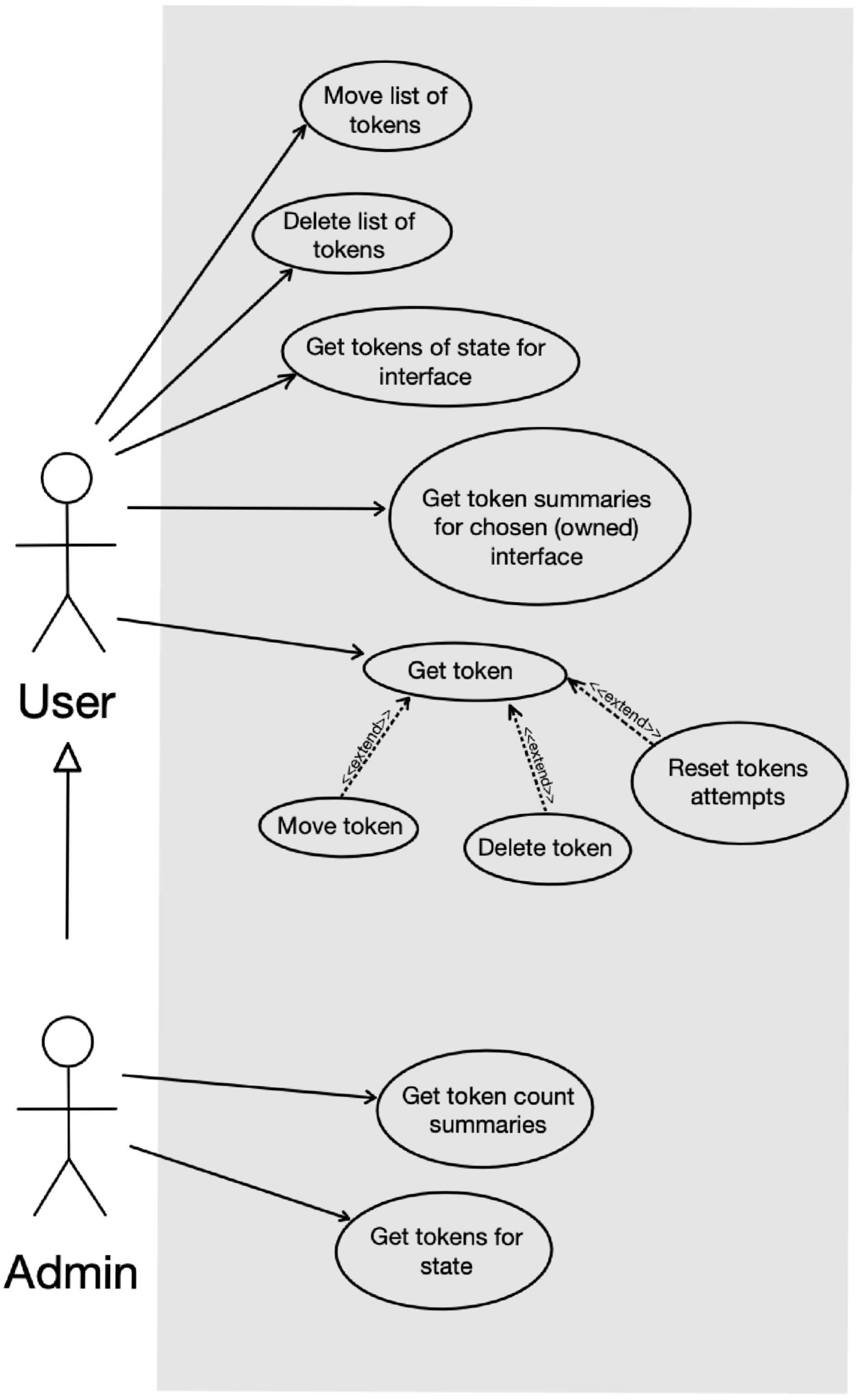


Figure 2.1: Design of a use case diagram for the newly designed application interface

2.2.2 Non-functional Requirements

The second major group of requirements consists of non-functional requirements, which do not specify the system's behaviour in different situations but rather define criteria that can be used to judge whether a development project has succeeded. Non-functional requirements take the form of features, not specific system functions, in different situations.

The following non-functional parameters have been selected for the software under development:

- Maintainability: This requirement determines the software's ability to be repaired without affecting other system parts. If the system is correctly divided into modules, one module can be repaired without affecting the others. Proper documentation and a well-designed architecture are essential to avoid unnecessary interdependencies..
- Scalability: Scalability is essential for systems with the potential for load growth. It is a property that indicates the ability of the system to increase or decrease its performance depending on the load. This is particularly important for businesses that are expected to grow.
- Reliability: Reliability is one of the main requirements in software development. It is defined as the probability of success at a given time.
- Extensibility: This is a characteristic that refers to the capacity for future growth, such as adding new functionality or modifying existing functionality without interfering with the system's current functionality.

Chapter 3

The Application Platform

Onsemi's [25] application middleware platform is significant for the day-to-day running of the company, as communication between the various systems of the corporation is indispensable. Therefore, despite hindsight, everyone is happy with the decision to develop their platform rather than buying it from an external supplier. The platform connects the business systems of the corporation, such as the sales systems, with the production systems, accounting or, for example, it can predict when the selected product will be able to be shipped thanks to the information obtained from production. Its benefits can also be quantified in data, with over 3,000 data streams containing approximately 2000 GB of data being transferred daily. Thus, even though the platform is not visible to the average user, it is a pillar that makes the company's stable growth possible.

Supported transfers are from database to database, between files, from file to database or vice versa. Other options are for example using the cloud, messaging services, various rest API, exchange and much more. So-called generic interfaces mainly trigger these transfers. An interface is called a data flow in a corporate environment. As already indicated, there are two kinds of interfaces. Generic interfacing can be created by the user himself thanks to a user interface that is used for that, and it is part of one of the micro applications. It is possible to select inputs, outputs, transmission, and data properties during the creation. The second, numerically smaller group are interfaces that are written manually. Mainly, these custom interfaces cater to higher-level logic that the generic ones no longer contain. For these interfaces, data collection and communication with the end owner of the interfacing are already required. These interfaces usually cause more problems than the generic ones, so developers usually try to cater for data flows differently, for example, by combining generic interfaces rather than creating special ones.

The system consists of 4 environments, each represented by various servers. The unique environments are ranked in ascending order of importance. First is the DEV environment, which is purely for developing enhancements and new features. Then there is Integration Testing, in abbreviated INT, an intermediate layer between the pure development and user testing environments. UAT, that is User Acceptance Testing. It is a user-testing environment, which is just below production. Because of rights and access, the testing process usually requires the owner to create sample input and run the actual interface. The result can then be discussed with the developer if something needs improvement. The last and most important is the PROD production environment. This is where the individual enhancements are pushed to during the 4-week release train, continuously moving the software up a level in the CD/CI process [2].

The entire system runs on the time of the company's headquarters, which is Phoenix in the United States.

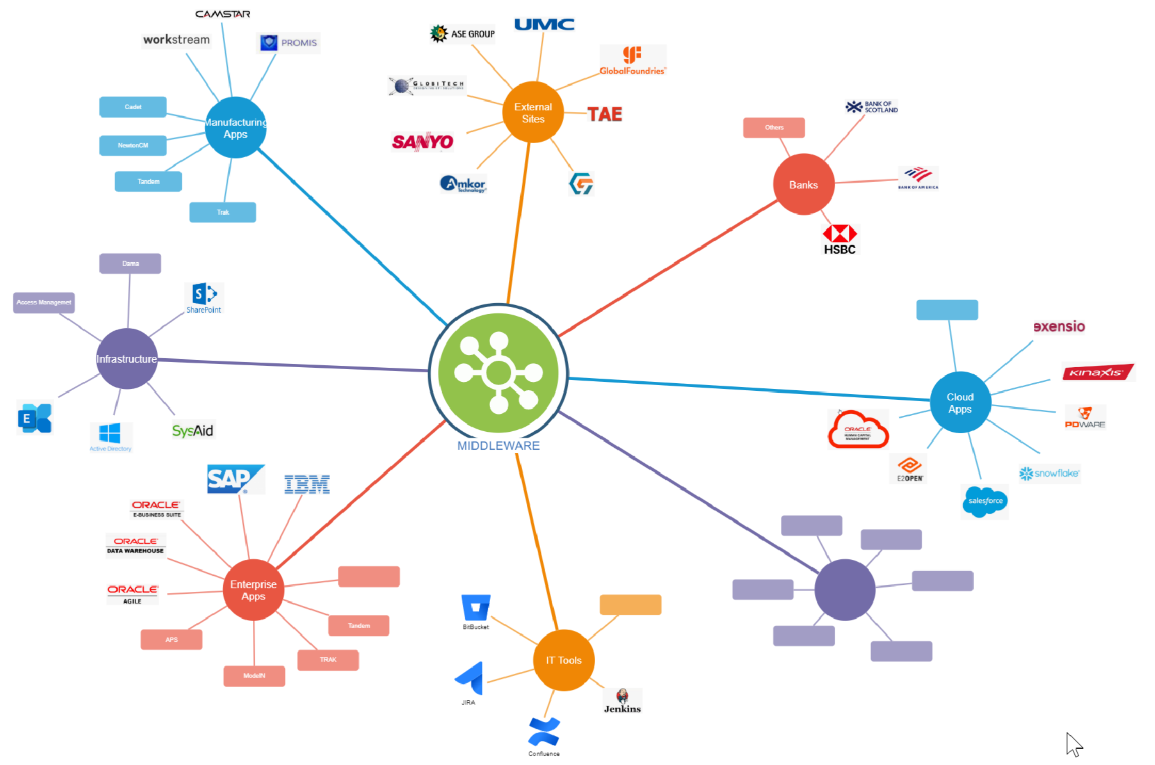


Figure 3.1: Diagram of the integrations handled by the 0nsemi platform, Source: Onsemi

3.1 Architecture

As observed, modern information systems are increasingly demanding, particularly regarding scalability and speed. This trend is mainly driven by the rapid growth of companies, which raises the crucial question of whether traditional monolithic architectures can still compete with modern microservices-based architectures [24].

In 2009, Netflix faced this challenge due to the high demand for streaming content, even though the concept of microservices was still unknown at the time. Nevertheless, Netflix became one of the first major companies to successfully replace its monolithic architecture with a microservices-based architecture, thus becoming a pioneer of this now widely used architecture.

3.1.1 Monolith

The monolithic model [10] creates a system as a single unit independent of other applications. One large monolithic code base ties all the functions together, making it significantly more challenging to modify and update the code.

However, monoliths are still used for smaller projects due to the ease of code management and deploying the entire project simultaneously.

The main reasons to use a monolith are:

- Easy development and deployment
- Simple testing and debugging

However, the disadvantages vastly outweigh the advantages for larger projects. These include:

- Scalability - One component can't scale on its own
- Reliability - One failure can affect the whole system
- Flexibility - Development is constrained by the technologies used in the monolith
- Deployment - Redeploying the whole project with every single change may slow down development.

3.1.2 Microservices

Microservices architecture [11] is built on distributed components, each with separate deployments, updates, logic, and a goal that it performs, thus contributing to the overall system goal.

Architecture solves many of the problems that large companies have with software development because the system consists of units that work independently, so it is possible to scale a unit independently or develop a service that does not affect the rest of the system.

One more crucial feature in development makes many companies apply this architecture today. This is the ease of updating code and accelerating release cycles in continuous integration and delivery.

This architecture is trendy nowadays mainly for the following reasons:

- Scalability - Can add another instance of this service to the cluster
- Deployment - Independent deployment of individual features
- Technology flexibility
- Reliability - A change can be deployed without risking crashing the entire system.

On the other side, there are also some disadvantages:

- Development management and overheads - If development is properly managed, the increased complexity makes development faster and more efficient. Along with this, there is a need to coordinate teams working on different services.
- Challenging deployment - The actual transition to this architecture can be almost unattainable for companies running on monolith.

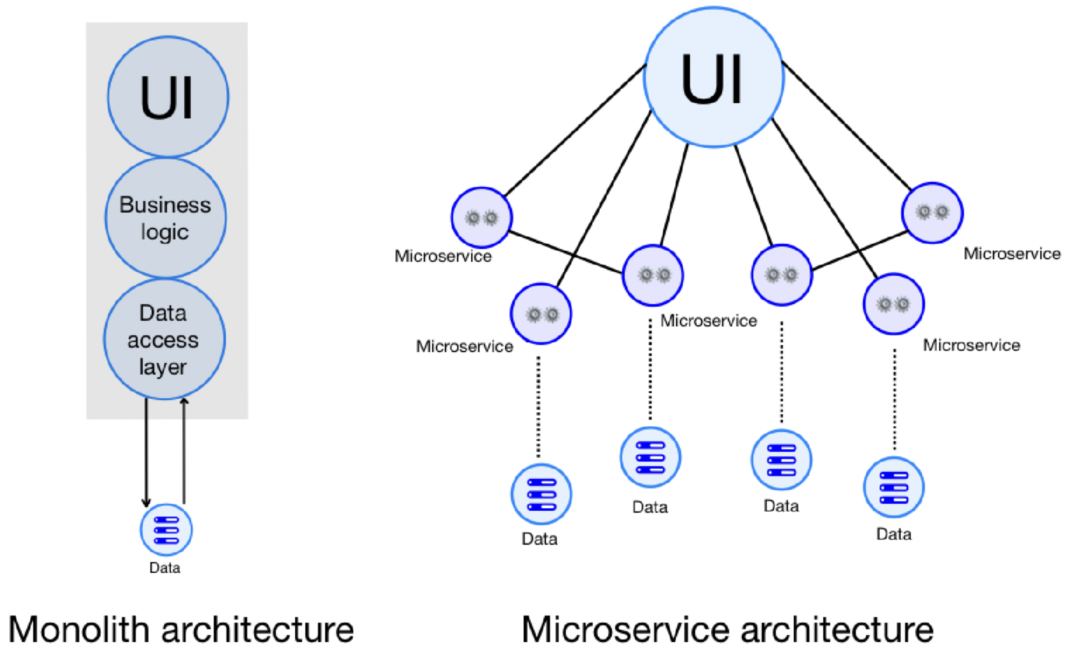


Figure 3.2: Comparison of monolithic and microservices-based architectures

3.1.3 Cluster Architecture

This architecture [27] includes how companies design their systems, and for larger systems, it is an ideal way to increase resilience to outages and increase the availability of their services. In a cluster architecture, individual components, which can be physical or virtual, are called nodes. These nodes communicate with each other. If one of these components fails, the others can continue their functionality without system outages, leading to higher service availability.

In our case, the Consul tool manages and coordinates the clusters. This tool allows administrators to monitor and adjust the configuration itself and much more.

3.2 Development Process

The right approach to software development is critical to the overall development process. The comprehensive set of methodologies and processes determines how a product is developed. Today, many approaches vary in different aspects, starting from planning to the delivery of the software. It is impossible to label any approach as the right one, as each one is suited to a different kind of project. This is why choosing the right approach at the beginning of any development is essential. Some of the most well-known approaches [19] are the waterfall mode, the agile approach, DevOps or Lean.

Onsemi's middleware team uses an agile approach enriched with CD/CI processes to develop its platform. In addition, they also use it to develop individual data integrations.

By using these practices, the team can develop the platform very quickly. Which, together with the involvement of appropriate tools, is also relatively easy and efficient. Which also reduces the overall cost of developing and maintaining the platform.

3.2.1 Agile

Within software development [7], the Agile approach focuses mainly on the continuous delivery of functionality. This approach makes it easier for developers to react quickly to changes and adapt the product to their needs. It is, in fact, an iterative process at the end of which there is always feedback from the customer or developers on which the next step is based.

3.2.2 CD/CI

Continuous Delivery and Continuous Integration are very well suited to the agile approach. Its use and advantages are demonstrated in the article [15], while also connecting it to the concept of DevOps [20]. The point is that these tools enable continuous delivery that is focused on code quality. These tools are used to automate testing, deployment and verification of code. In agile development, they are useful in the effort to deliver new features that are tested and deployed quickly. This process eliminates human mistakes to a greater extent and saves the developers themselves the time they would otherwise spend performing monotonous tasks.

3.3 Platform Services

The Onsemi platform is designed based on microservices architecture for the reasons mentioned above. Thanks to this feature, the team can release bug fixes and work on improvements daily, which they are able to release every 4 weeks.

This process is called Continuous Deployment, and together with the Continuous Integration process, it introduces a series of steps that need to be carried out to create a quality software release. This versioning is done on each micro-application, and the Jenkins service is used to manage and build it.

These are the leading and most essential parts of the middleware platform that relate to the software being developed:

3.3.1 Runtime Engine

The main task of this microservice is to start individual interfaces. These interfaces are stored as Java archive files, aka JARs, which the service executes if the Scheduler decides to. The java development kit runs these individual interfaces. Because a large number of JAR files run, approximately 3000 per day, much emphasis is placed on garbage collection.

The runtime engine could be called this system's core due to its interfacing spooling function. It communicates with all other microservices.

DE light

It is a library that performs only a simple task: token creation. The runtime engine always initiates this action at the start of each interface. It is essential for our work because of precaching, where each created token will be cached to avoid non-correlation with the real file system as much as possible..

3.3.2 Dashboard

This service is significant for every user because it is a place to view all the data integrations.

The introductory page provides a file hierarchy of all interfaces created for administrators and a list of owned interfaces for regular users in the selected environment. This is a critical feature precisely because administration outside the Dashboard is not possible. The user can test a given interface, view logs and modify its startup properties—for example, triggers, cascading actions or scheduled operations.

The Dashboard also contains an interface on which users can create a generic interface using a wizard. Few users still use this option due to a lack of technical knowledge. As a rule, the way it works is that the requirements are given to the administrator, who considers what interface type it will be. Once created, he sets the interface as visible to other users and promotes its environment to that intended for user testing or "UAT". Once the user approves the functionality, the interface can be deployed to the production servers. However, administrators try to mystify this process and teach users to create their own interfaces and operate the platform without communication.

This is the first place administrators look if one of the integrations fails because there is also a listing of another service called "Logging Service", which logs all the information about the runs of each interface and service. Due to the simplicity of the interface and the creation of contrasting logs containing unexpected behaviour, this is the fastest way to investigate the problem.

The application will also include the interface being developed as part of this work.

3.3.3 Scheduler

The Scheduler has a critical role in the platform, which controls the execution of single interactions. Data integrations can be triggered in various ways, remotely - API, database plugins, by clients, manually or timely scheduled by a Scheduler. The integrations themselves can be triggered periodically or based on events. One of these events is, for example, a cascading trigger after another integration has run out of time or, based on, for example, the fulfilment of one of the preconditions of the so-called precondition, which can be different. Typically, this may be, for example, a condition that the input data is in place. Ideally, it can notify the runtime engine when the interface is running, but it can save time and resources otherwise if one of the conditions is not met.

3.3.4 Delivery Engine

Delivery Engine takes care of data delivery. The different interfaces are described in the introduction of chapter 3. Interfaces are accessible to users through the Dashboard application, where they can be configured and managed.It is essential to guarantee the correct order when delivering files. If this is not necessary, it is possible to deliver multithreaded.

We need to guarantee data delivery, but any error could occur while the file is being processed, so the logic of moving tokens between folders in the file system was introduced to represent the transfer state.After data processing, the service creates a token on the server's file system where it is running. The token represents the current delivery status by moving between folders. If an attempt fails, it will reduce the number of attempts and try to run again in 5 minutes. On successful completion, the token ends up in the "DONE" folder, likewise on failure.

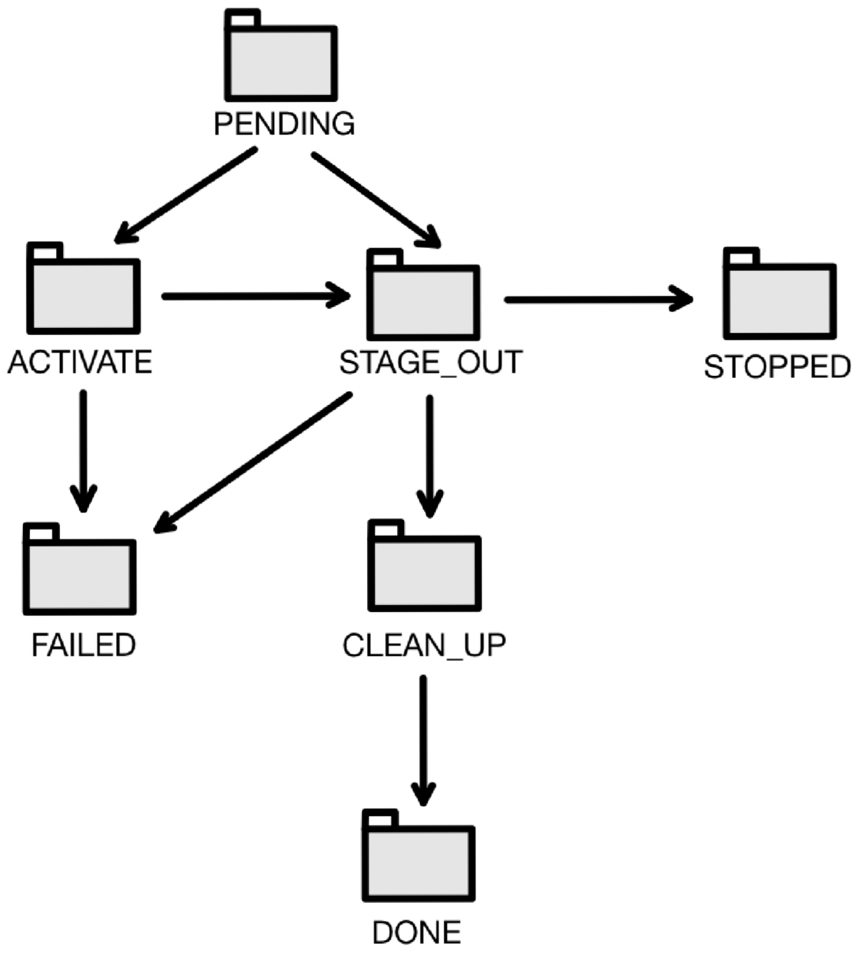


Figure 3.3: Simplified token lifecycle diagram

As the paragraph above states, the token movement corresponds to the actual state. Thus, the individual states correspond to processing-related states:

- PENDING - Runtime is ready for scheduling.
- ACTIVATE - Runtime is currently executing.
- STAGE_OUT - Currently waiting for output file to be transfered.
- STOPPED - Stop command was issued during execution.
- FAILED - It was not possible to perform all the actions.
- CLEAN_UP - Deleting temporary folders created at runtime.
- DONE - All actions were successfully completed.

Our application interface will take care of the management of these tokens.

3.3.5 Logging service

This service creates and stores logs from the interface runs together with logs from the work of individual services.

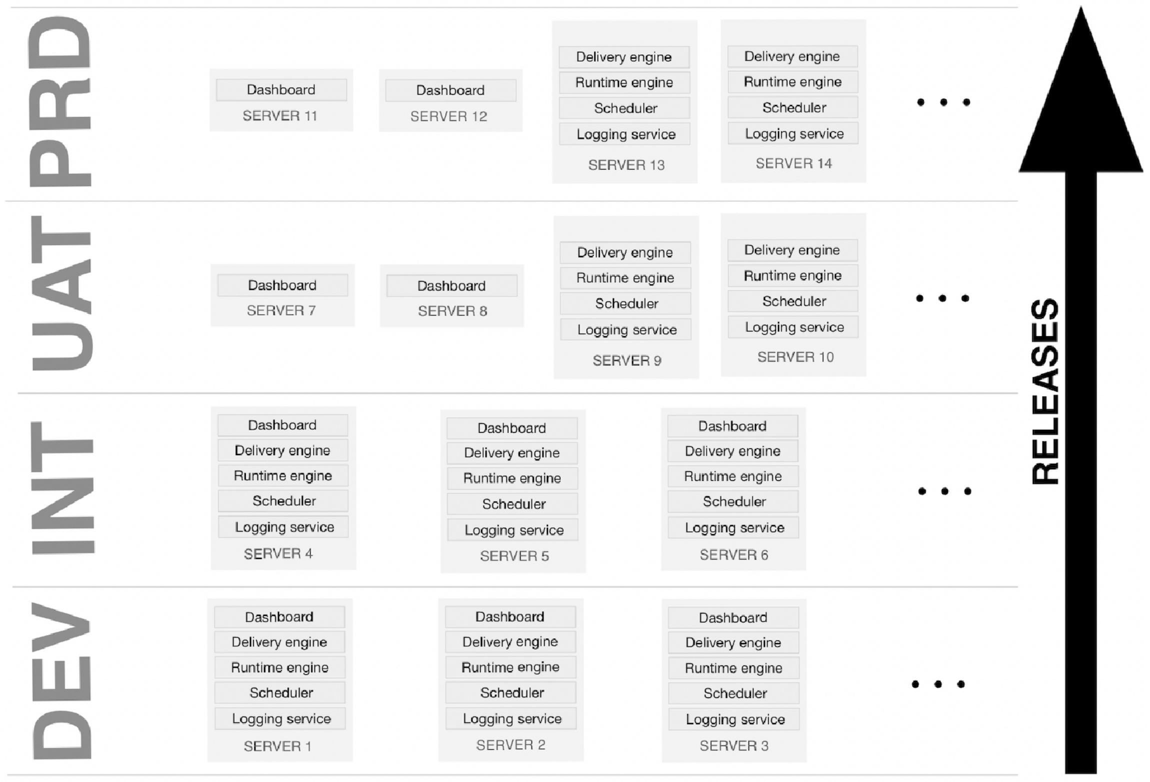


Figure 3.4: Simplified scheme of clusters used in Onsemi

3.4 Communication

Communication between services in a microservice-oriented architecture is critical because communication in the system must be as efficient as possible [3]. This architecture's fundamental division of communication is synchronous and asynchronous communication.

The difference between these two communications is that a message is sent during asynchronous communication, but the response is not waited for but is processed by another service entirely independently. While synchronous communication consists of calling an application interface in another service, synchronous communication is about waiting for a response.

Each communication in this architecture has its advantages and disadvantages:

For asynchronous communication, the main advantages are for example:

- Lower failure rate - if the service fails, the sender continues to send
- Faster response time - reduced waiting for a response

- Load - uploaded requests act as a queue

However, the disadvantages include:

- Latency when the queue is full
- Higher complexity

As stated in the [33], data dissemination between microservices should always be asynchronous if possible. The main reason for this is the goal of the architecture itself, which is to have services available even if other services become unavailable. At the same time, as synchronous dependencies grow, the overall response time for applications grows. Therefore, creating a kind of HTTP request chain between services is undesirable.

APIs are used in communication between middleware platform applications along with the OpenApi specification.

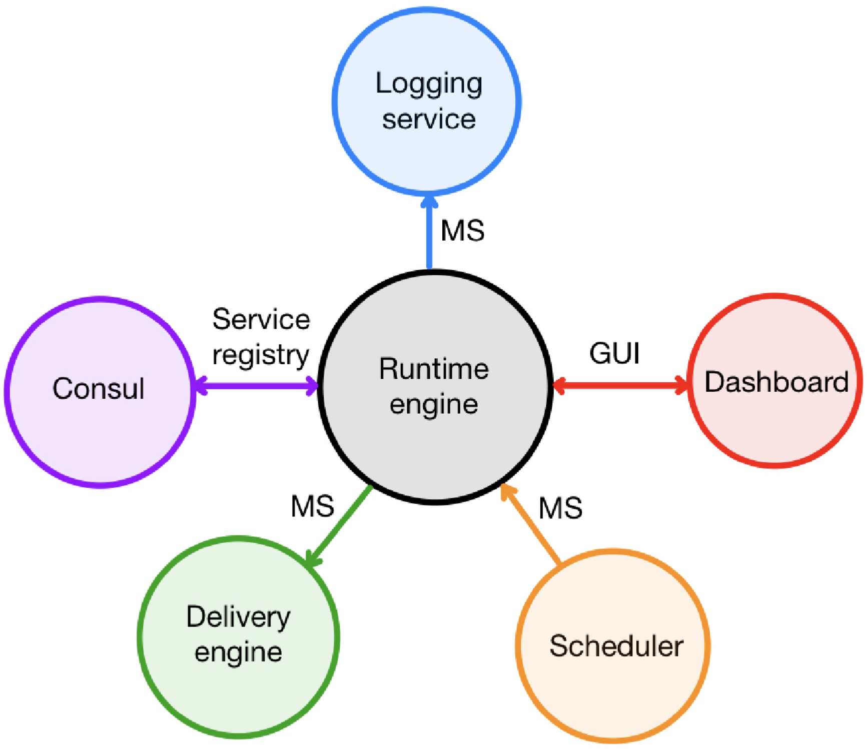


Figure 3.5: Diagram of basic communication between platform microservices

3.4.1 Swagger

This is a framework [38] that is used to define interfaces according to the OpenApi specification. It covers the complete software lifecycle, from design to documentation to deployment. It is a toolset that can make developing and working with software easier when adequately annotated.

When used in its full scope, the tool can significantly accelerate software development in larger teams.

Swagger Editor

This tool is used in the design phase. It allows the creation of Swagger UI, thanks to which it is possible to test annotated endpoints through structuring using annotations in the source code [23].The documentation itself then allows testing and shows the responses from the service.

In addition to the interactive documentation, this is also a building block for subsequent code generation and API work.

It also allows documentation of data structures.

Swagger CodeGen

The main task of SwaggerCodeGen [26] is to use the generated documentation to create a skeleton of either a client that will call the annotated service or an application to implement the service.

3.5 Development tools

The development of platforms like this one must include several additional tools, which the Onsemi team has time-tested thanks to their experience in the industry and time spent developing their platform. These tools are used during development to facilitate the extraction of value and manage the code and the development. The tools mentioned below are an essential part of the platform.

3.5.1 Consul

Consul [14] is a multi-network software. It can be used to solve problems associated with the operation of microservices. With this tool, it is possible to discover the state of microservices, service networking and overall traffic management. Consul can be deployed individually or together. The procurement team can use this tool to manage individual microservices and servers in the cluster.

It provides, among other things, several services that are key to its users:

- Health Check - Provides individual microservices with a check that determines whether the requested microservice is active in the cluster. If any of the services fail, Consul notifies the other services.
- Key-Value store - This is a distributed key and value store that can be used by individual microservices.
- Service Discovery - It is an interface that allows services to find others. For example, it can get an address for subsequent communication.
- Service Mesh - This is a configuration for microservices, which can thus use Consul to control traffic routing or traffic limits using the Traffic Management feature. It also handles securing communication with encryption or user authentication, for example.

3.5.2 Jenkins

It is an open-source tool that is used in Continuous Integration (CI) and Continuous Deployment/Delivery (CD) processes during software development [9]. It is a very widespread tool that is used for automation, such as building code, testing it or even deploying it. Even though other services offer similar features today, Jenkins still remains a big player in this field because it has certain advantages that are especially crucial in the case of large projects.

- Open-source - As mentioned in the first paragraph, this is open-source software. Which means it is completely free.
- Scalability - Allowing scaling across multiple servers is great for cluster architecture, where you would have to build and test separately on each server.
- Flexibility and expandability - Probably the most fundamental feature. This allows extended instances with custom scripts or plugins from the Jenkins library.

Thanks not only to these features, Jenkins is great for CI/CD processes in microservices architecture [18], where the emphasis is on the fast and reliable deployment of changes.

3.5.3 Maven

Maven is a project management tool. Maven takes care of building a project based on the dependencies specified. It manages this process by using a configuration file called "Project Object Model", abbreviated as POM, which contains information about the project along with its dependencies and modules. At the same time, Maven provides a central repository that contains thousands of plugins that can be downloaded to a local repository where these plugins can be run with each project start.

The tool automatically downloads the dependencies that are in the POM file, which results in saving time spent manually downloading individual dependencies. Maven can make software development faster and more efficient, as its described in [34].

Parent

In addition to the classic dependencies, there is a parent section in the POM file. This section contains the configuration that can be shared across the projects that share it.

3.5.4 Atlassian tools

It is an Australian software company. Their products are tools that are used to improve teamwork. With these tools, it is possible to improve the software development process or project management. Due to its high robustness and performance, the company's products are trendy. In the case of 0nsemi, a large part of the work management, the following tools are used:

Confluence

This tool is used to manage information between team members [6]. Specifically, it includes various documentation and manuals that make the work of team members more efficient. This tool works on the principle of creating and editing individual pages that can contain text, images, tables and others. The tool also supports the creation of these pages in a tree structure, which improves the clarity of information. Of course, it is also possible to set up different accesses, which can be divided by level. For example, confluence can also be integrated with JIRA, a project management tool.

Jira

Jira is used for project management and task management [5]. Its main task is to plan and then manage the work. It does this by creating a so-called ticket that contains a task that needs to be completed. This task is then assigned to someone and gradually changes its status until it is handed over for testing. This process aims to reduce task errors as much as possible and make the overall team work more efficiently. Among other things, it allows working with individual versions of a project, which leads to the ability to track changes in single versions by summarizing individual tickets. Like Confluence, JIRA allows integration with other tools, which leads to a kind of environment for working on projects.

BitBucket

This platform is used to manage Git or Mercurial repositories, as described in [12]. Bitbucket offers provides code management and collaboration tools for projects. It allows developers to create new repositories or comment on code and manage them. These features make it very easy to collaborate on code among team members. Providing feedback on code is also an essential part of development, which is also one of the critical features of the software. Another feature is version control, which is indispensable in almost every project and allows tracking changes in each version. This is another Git repository management tool, and its main advantage over the others is a feature outlined throughout this section: its integration into the environment created by the Atlassian tool combination.

In addition, plugins are used that allow integration with Jenkins. Then, when the code is added, it runs the Jenkins Job configured. As a rule, this involves building, running tests and then deploying. This integration again increases the automation level, saving time for the platform developers themselves.

3.6 Data serialization formats

The most commonly used formats are Extensible Markup Language or XML and *Javascript* Object Notation or JSON. JSON and XML are both highly used serialization formats with advantages and disadvantages, as described in this article [4].

In the case of XML, it is a markup language used to describe the structure of objects. In comparison, JSON is a data exchange format that can be used in any language. In the case of XML, it is an inferior option for sharing data between applications compared to the second format because it consumes more memory during processing. Writing and reading JSON is generally easier than XML because it uses fewer characters. This also makes it more efficient to transfer over the network. Another reason Onsemi's middleware platform uses JSON is to parse large amounts of data more efficiently.

The above formats are the main reasons why Onsemi mainly uses JSON format, along with the Jackson processor, to work with the data.

3.6.1 Jackson

This is a prevalent open-source library for working with JSON format, and this book is dedicated to this topic [16]. It is used to map Java objects to format and back. It is a very efficient way because it supports different types of data structures. There are three modules that the library contains. The first is the basic module that handles writing and reading bypasses in JSON format. The following module is the Databind module, which takes care of serializing Java objects to JSON and vice versa without parsing. The last of these modules annotates classes and attributes for automatic serialization and deserialization.

3.7 Security

In this section, we'll look at why security is an important part of the design of that system. This mainly includes the protection of corporate and user data, which by name implies that the data that each endpoint acquires will not be misused or compromised. If this were to happen, sensitive information could be leaked, or functionality could be abused.

As such, the functionality could also be misused to attack the application itself. For this reason, endpoint security is an important element in protecting the application. Endpoints, due to their function in the communication between client and server, can fall victim to many types of attacks. Some of the most notorious attacks described in [21] include the following:

- Man-in-the-middle - this attack targets the communication itself and obtains sensitive data. This may include, for example, user data.
- SQL injection - this involves problematic or incorrect filtering of input. Resulting in the attacker being able to execute malicious code and/or obtain sensitive data.
- DDOS attack - this attack targets the overload of a given service using many requests from different devices. This causes an overload that makes it impossible for even legitimate users to enter. This attack results in slowing down or crashing the service.

There are many more attacks, like brute-force or cross-site attacks. Therefore, endpoint security must include various measures such as SQL injection filtering and data encryption. Each attack, as mentioned above, needs a different form of solving, and it is, therefore, essential to choose suitable security.

Other vital processes in security are authentication and authorization. Using a combination of these two processes prevents data leakage or malicious behaviour by a user who is not authorized to access the data.

Authentication

This process is used to authenticate the user. Its purpose is to prevent unauthorized access to the system. Simply that the user who logs in to the system is actually behind the device. This process is usually performed using a username and password. For higher security, biometric authentication is emerging.

Authorization

This is a process that is handled after the authentication process, where it is possible that the user has the right to enter the system itself but may no longer have them for reading certain documents or for other operations. The goal is to control the access possibilities within the organization, where based on the role, individual rights are modified as part of the movement within the system. As with authentication, this process is concerned with denying access to data, but in this case, it is a denial based on the rights of an already known user.

3.7.1 User Authentication and Authorization Methods

There are some methods for implementing authentication and authorization processes. Where data and systems need to be protected, they must be accessed only through authenticated users with the necessary rights. In this section, we will therefore discuss the methods that are described collection [8] and that are commonly used.

JWT - JSON Web Token

This is a standard for the reliable exchange of information between parties. The JSON token plays an important role here, as it is used to authenticate the user and access certain parts of the system.

This mechanism is used so that when the user logs in, he receives a JWT, which is then used for further operations that the user will perform on the server.

OAuth

This standard works by having users share their data with third-party applications that they do not have to enter directly.

It works because the user first logs in to the provider's site, the most common being Facebook and Google. After the user gives consent, the provider returns a token to the app to access his or her data.

BasicAuth

This method works by authenticating with a password and username. It is prevalent when accessing resources within a website or API.

When this method is used, it is required in every operation where security is implied. The method is susceptible to man-in-the-middle attacks due to the format in which user data is sent.

Nevertheless, BasicAuth remains in use today, especially for easier and faster user identity authentication.

Form-based HTML auth

This method shows the approach by which the website secures access to resources. It works by filling out a form and getting user credentials. A final authentication mechanism on the server then follows this.

Despite being highly susceptible to attacks, this authentication method is widely used.

SAML

This is a standard that is used to exchange authentication and authorization data. With this method, users can log in only once and gain access to all applications.

The application is an identity provider in a central location within the organization. After authentication, it sends a SAML token to the application. After authenticating it, the application gets the user's data and rights. SAML is much used in corporate environments.

Chapter 4

Design of the system

The chapter is dedicated to the design of the application. This chapter aims on creation application design based on the requirements set out in the chapter 2. The main task of the chapter was to look into the requirements and create a summary that can be implemented. It is about introducing a new interface to the application. Adding new communication on such a platform is challenging, especially debugging it properly. Therefore, it was necessary to consider the placement of the data transfer objects and how the actual communication will occur. Ultimately, the choice was made to connect the microservices using the Swagger CodeGen tool, which can create a client that mediates the communication when adequately annotated.

From the beginning, it was essential to understand what we would be working with to achieve the requirements. This part was followed by determining what endpoints would need to be created in our application interface and how they would work. Theoretically, this was just a list of relatively simple functions that would ultimately perform tasks over the system when combined with the frontend.

The software design was consulted with the architect of the platform.

4.1 Architecture

Architecture design is a crucial part of designing new software [32], because a poorly designed API can harm the whole process. Therefore, this section will describe the final communication layout to make it as suitable for our case.

The case we are discussing is necessary based on future sustainability. Essential elements are, for example, the ability to respond quickly to customer requests or the ability to scale together with application management.

Another important task was creating a user through which client communication will work. Fortunately, this possibility is already implemented in the platform. This eased the authentication process and set up communication between microservices in the case of the application interface. The last section that had to be addressed in the architecture design case was to limit the possibility of negative impact on the operation of already established services as much as possible.

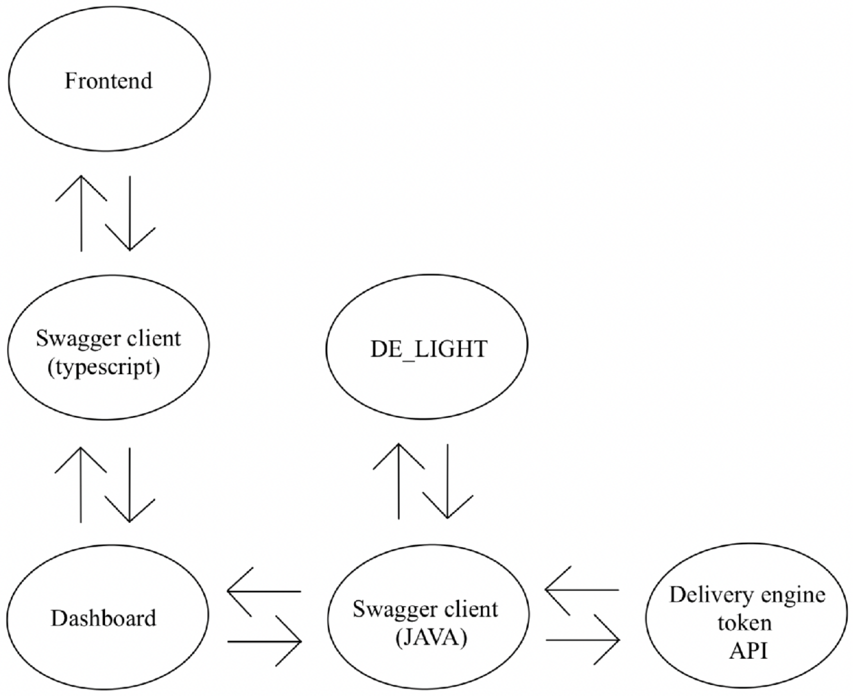


Figure 4.1: Design of communication between microservices and the frontend of the system

4.2 Entities and Models

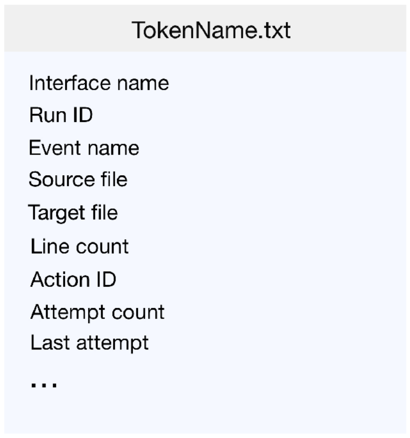
Each system needs to create models and entities to implement the new API. These will be used as data representations within the functioning of the application programming interface. Properly designing models and entities is integral to creating a successful system.

The models that need to be created in the system include the following:

- MoveTokenRequest - This is an object that is intended for the case of a request to change the state of the token. This object contains three properties: the token's name and its current and new state.
- NodeNameAwareResponse - this object returns a response enriched with information about the server the response is coming from. Thus, there are two properties, one containing the content of the response from the server and a string with a string.
- StateSummary - an entity representing the individual states between which the token moves. Practically, these are folders on the file system.

There are also several entities in our system:

- State - this is an entity that represents the individual states between which the token moves. Practically, these are folders on the file system.
- WorkItem - this is the representation of the token.


TokenName.txt

Interface name
Run ID
Event name
Source file
Target file
Line count
Action ID
Attempt count
Last attempt
...

Figure 4.2: The content of the token that is serialized to the WorkItem

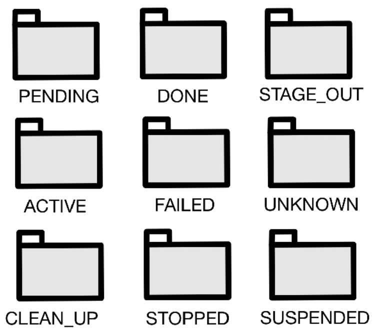


Figure 4.3: Initialized objects in state

4.3 Application Programming Interface

In this section, we will discuss the process of designing an application interface that will allow interaction with the final application. This interface will be crucial for the entire application. Every application interface, must meet certain characteristics that should be properly thought out during the design process, because a bad design can affect existing parts of the system.

The API itself should be based on the functional requirements that we have defined in the chapter 2. We have defined that each user will only have access to his integrations for which he is the owner. We cannot solve this problem because the Dashboard interface already works this way and only allows administrators to view all interfaces. This makes the authentication issue relatively easy to solve.

Furthermore, we know that the Swagger framework, currently being implemented in various platform interfaces, will be used for communication between services. This framework will also provide interactive documentation and an environment to test individual endpoints.

The main thing that needs to be designed is the endpoints themselves. As mentioned, these endpoints need to be designed based on the endpoints we have already defined.

Apart from that, however, the properties mentioned above need to be designed for each endpoint:

- URLs - The URL address for each endpoint should be easy to read and should describe the functionality of the endpoint. The consequence is easier work with the API, easy understanding for developers what the endpoint is for, etc.

- HTTP Methods - Each of the endpoints must have an HTTP method selected, such as the GET method, which is used when retrieving data from the server, or the POST method, which sends data to the server in the payload.

- Parameters - We must choose the correct type and number of parameters that will then be used in the business logic.

- Responses - Deciding what format and type of response to return to the client. The response should contain all the elements that will need to be used afterwards. And have a format that is easy to work with.

For these reasons, the following endpoints were designed using entities that we have already designed or already existed in the system:

GET methods:

- `getSummary()` - This endpoint will provide a main window, displaying a list of all possible folders (states) that we can access. Next to them, the number of tokens inside will be displayed in brackets, making the work more transparent. Parameters are not needed here, as the primary view is displayed. The response will be in JSON format, and in addition to the sheet containing objects of type StateSummary, it will also contain what server the response came from for future reference.

  This endpoint is more of a test, as there is no requirement in the current model to display all interfaces from all servers in the environment. In the future, it could extend into the Admin section, where there would be aggregated tokens from all servers and interfaces.

- `getSummaryForInterface(String interfaceName)` - This endpoint is the one that will be displayed first to the real user. It is actually a filtered version of the previous one. Its only parameter will be a string containing the name of the interface for which the StateSummary list will be returned and the name of the server.

- `getTokensForState(String state)` - AAs with the last pair of endpoints, we start with the one that is a general version of the previous one. Its function is to return tokens that are in a specific state. Its only parameter is a string containing the name of the state. Its larger purpose rests on the ability to extend the API beyond the interface directly for that interface. The response is again in JSON format and is a list of work items, which are models created based on the token, again containing server information, among other things.

- `getTokensForInterface(String state, String interface)`
  - This Endpoint will be the one that will be called when the user selects which state he wants to enter. As the definition implies, the function's parameters will be a string containing the name of the state to step into and the name of the interfacename that is selected. We will return a list of WorkItem that corresponds to the tokens in the given state.

POST methods:

- `resetAttempts(String tokenName, String state)` - This endpoint will be tasked with resetting the attempts of a given token on completion. If, for some reason, a token has lost all attempts, which could be due to system overload, a crashed server, or perhaps an error related to the input data. The number of attempts is defined during the integration creation, and the default value is 5. Thus, on failure, the run is stopped and restarted after 5 minutes, up to 5 times. The function parameters are two strings, one containing the name of the token, and the other containing the name of the state in which the token is located. The response is a CommonReplyBean that contains the success or failure status of the action.

- `sendToken(MoveTokenRequest request)` - This will be an action that will move tokens between states. Its parameter is the MoveTokenRequest described in the section above. This action may be required, for example, when moving tokens from some folders to DONE, which is used for tokens that are already completed. The response is again a CommonReplyBean that informs about the success or failure of the operation.

- `deleteToken(String tokenName, String state)` - This endpoint will remove the token. Even though this is an extreme case that is not a commonly used practice, there are still cases where this endpoint will be used. However, it must be flagged with a warning in the future front end. Here again, we have string-type parameters, one is again the name of the token to be targeted, and the other is the state where it is..

- `deleteTokens(List<String> tokensList, String state)`,
  `sendTokens(List<String> tokensList, String state)`
  - Both endpoints have the same function: extending the functionality with and operation on token lists. Hence the functionality is not different, only one of the parameters is different, and that is that there is a token list instead of just one. The answer is the same as the original functions, namely CommonReplyBean, determining the resulting state of the call.

4.4 Frontend Design

For API design for a system such as this, the behaviour of the front end must be as close as possible to the expected behaviour from the user. This expectation will match the behaviour of a file system known from different operating systems. This is achieved by choosing suitable icons and working with the hierarchy.

Users must be able to find intuitively a given token or group of tokens and then perform operations that display important data, moving tokens, deleting them and, last but not least resetting attempts.

It is also necessary to think about where the design will be implemented. In this case, it is the window within each integration. This window will be reachable in the details of each integration. Thus, it is also necessary to think about how the graphical page will look together with the rest of the already implemented frontend of the page.

Therefore, because of the users' diversity, the interface's main requirement is intuitiveness. Intuitive control and design is the key to a proper interface design.

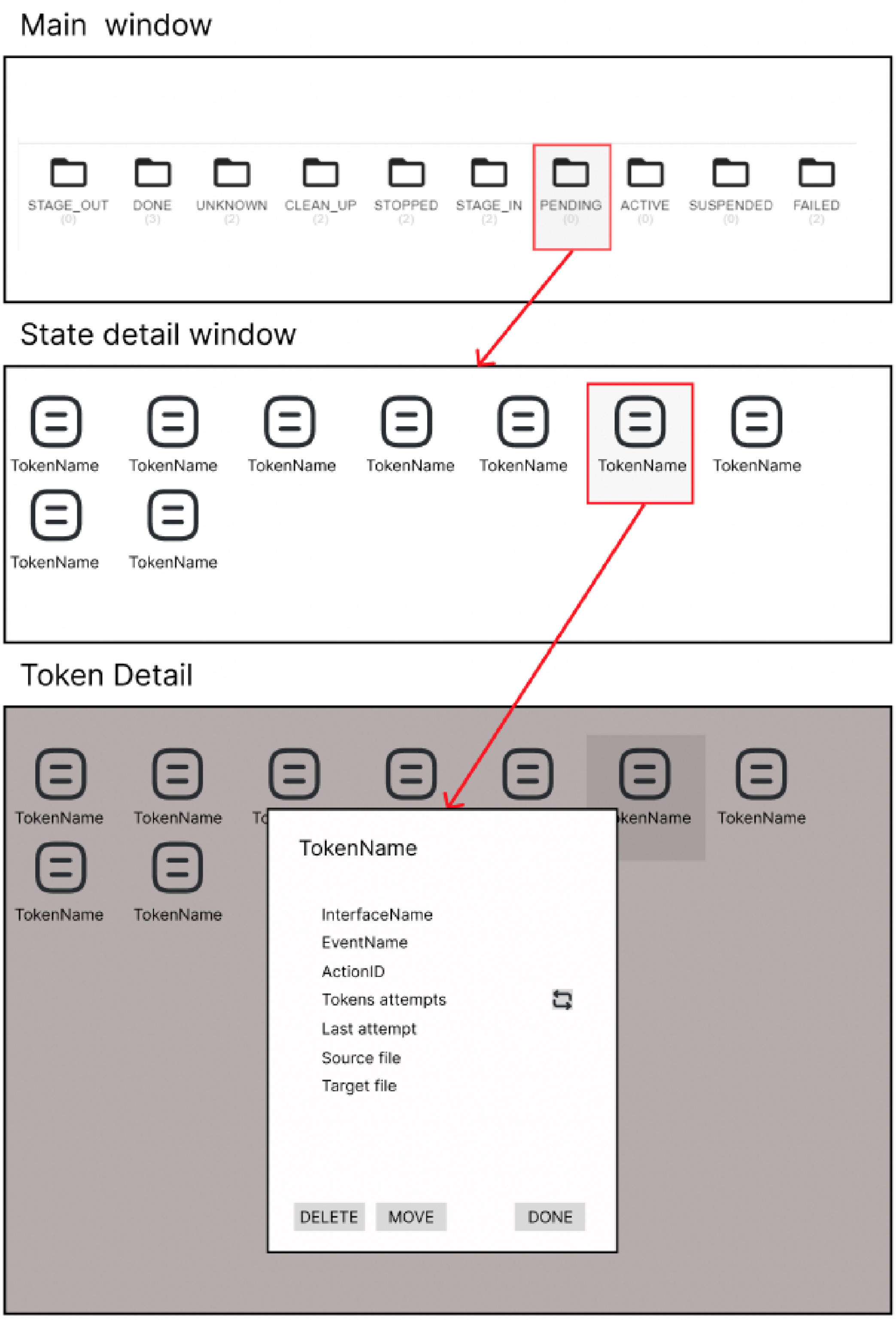


Figure 4.4: Design of the graphic interface for Dashboard

4.4.1 Figma

For the design and presentation of the frontend, the Figma was used. Figma is a graphical tool that is free of charge in its basic version and serves users to design web or mobile applications. One of its main advantages is that it is a cloud-based tool, which allows working anywhere. Its functions and uses are also described in the collection [17]

It has been used to create wireframes containing preliminary graphic designs. Once created, this design was consulted with other, more experienced platform developers.

4.4.2 Security

Platform security refers to the specific steps that each system must complete to ensure the platform's security [31]. Thus, the implementation involves many tools and technologies together with a methodology that, as a whole, creates a whole that should implement as many measures as possible to eliminate the possibility of security risks.

We will address the security used by the platform. Basically, it is the Legacy security framework. It is actually an outdated security framework that is largely not up to date with current standards, which can result in increased security risks for the company.

The company's middleware platform uses Spring Security to modernize its various components. It is a comprehensive framework for implementing security features. Specifically, for example, authentication or authorization. Data linking occurs specifically with the Legacy framework, which serves as an additional layer of security. Thus, it creates various additional rules and constraints for the system. Spring Security can also be combined with other authentication methods. In our case, this is the BasicAuth method. It serves as a primary authentication based on username and password.

Even though this combination requires a more complex configuration, the platform achieves a sufficient level of security. That ensures that all layers are fully functional and that different methods are used for different system parts. Even though, for example, BasicAuth is a primary method that might seem susceptible to attacks of various types, such as brute force, this risk is drastically reduced due to its inaccessibility from the public network.

Therefore, if we would like to log in here, it is necessary to be on a local network or be connected using Remote Access Security, abbreviated RAS. That means that the attacker would have to be inside the network for such an attack. High requirements such as creating strong, regularly changed passwords are already a standard that Onsemi also uses.

Chapter 5

Implementation

This section follows the software design chapter, so it is in reality. So we will focus on implementing APIs that are used for communication between applications and systems. This can be done using the interface that is defined here. We address the implementation by defining our designed functions and their associated operations.

As such, API implementation is crucial for larger systems where there is communication between different systems or applications. It is also used to integrate different systems that use different technologies. In our case, it is an API that will communicate with microservices. Specifically, it will be the Dashboard for communicating with the frontend and performing operations defined within the functional requirements and the DE_light library, which is part of the Runtime engine, which will be called during token creation as part of the precaching. In the previous chapters, we designed the operations that will be implemented. At the same time, the technologies are decided within the corporate system.

The implementation itself is implemented in a gradual process of Continuous Integration, from the basic functions to the functions providing the required operations. Part of our implementation is to ensure zero impact on the existing system, which is achieved through integration testing and continuous upgrading of the environment in which the system resides. This impact is essential, especially in terms of performance, which is important in delivery platforms. This is why efficiency is vital for any related development. Another way this can be achieved besides code optimization is, for example, scalability. Scalability is crucial for growing systems, as the adaptability to different numbers of users and increasing command processing requirements can be devastating.

Therefore, in this chapter, we will implement APIs so that the ability to perform all operations even under load is achieved without system affectation. Among other things, all the tools that were used besides those mentioned in the 3.5 section will be described here. These tools were used primarily because of their use in the platform. It also discusses the problems that occurred during the implementation—moreover, ways of solving them operatively.

As part of the overall development process, I received feedback on the code in the code review of the BitBucket application. This process was helpful, especially in improving the quality of the code.

5.1 Models and Entities

The designed models and entities from the previous chapter had to be created to meet the requirements and formal properties of proper implementation. In the implementation, it was necessary first to create models that will be used in the responses to the individual endpoints of our proposed application programming interface.

We first started with the basic building block of most of our responses, named `NodeNameAwareResponse`. This found its use mainly because of the information returned by the individual responses from the server. This functionality will find its use mainly in case of working from the Dashboard, where the knowledge of which of the servers a given token comes from will be very important for the creation of the subsequent request, which will have to send a request to a specific one of them from the environment obtained from the aggregation of all servers.

The next implemented already created are very specific and are more about dealing with specific situations. One such example is the `StateSummary` object, which is used within a list, and its parameters are the `Stateit` is in, along with the number of tokens it contains. It will thus be used to get the current state on the main window. Alternatively, for example, the `MoveTokenRequest` object was created and this is used within a single command, namely creating a command to move a token to another state.

5.2 Cache

Even though cache creation was not planned in the design phase, during the gradual integration into the system, it turned out that the gradual loading of folders slows down the Delivery engine to a greater extent, so it was necessary to create another object that creates an accurate representation of the current file system content on the server. Then every movement is also projected into this object. Hence the add, delete, and move methods are created in this class to keep it current.

The first setup and then the periodic refresh is taken care of by `TokenCacheSetter`, which scans each state and stores information about it in the cache, which is then used. As I mentioned, it also takes care of the automatic refresh thanks to the `@Scheduled` annotation, which runs the function periodically according to the time not set in the consul tool.

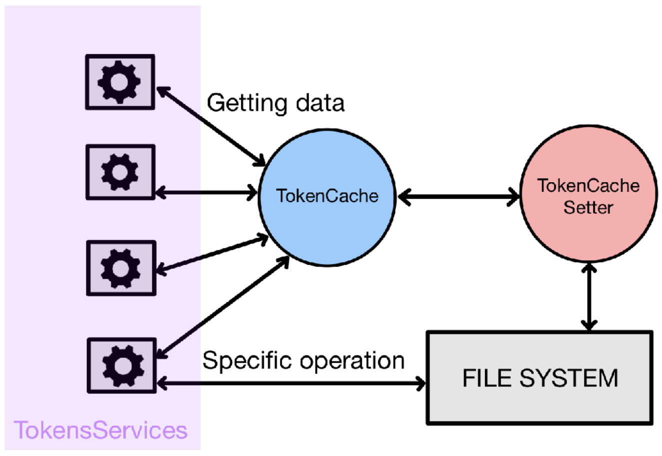


Figure 5.1: A way to solve a performance problem by creating a cache.

5.2.1 Performance Problem

Another essential piece of information that changed only as part of the implementation was the use of the DONE state. The first change became apparent right after the release of the first version of the API, which encountered a problem with a higher number of tokens the moment it tried to list the DONE folder and, thus, the request timeout. This problem was solved the first time by adding the pagination feature.

However, with the addition of the cache to our implementation, it was no longer the only one needed, so it was agreed with the administrators that the usage of the DONE folder itself is pretty small, so it will be sufficient to store the `StateSummary` itself, which will only show the user that the folder exists. We will only store the number of tokens that are in the folder.

The final problem that brought development to the point where the DONE folder itself was omitted entirely and not even displayed to the user was a coincidence problem, where the script that usually takes care of creating the hierarchy in the DONE folder so that the number of tokens in it does not grow infinitely failed. Creating a folder and inserting older tokens would prevent it from listing them. Unfortunately, this failure was also not detected immediately, but only when the version containing the autorefresh cache went into production. Thus, the moment the setter attempted to list the value of a given folder, it would generally store the number of tokens. He could not achieve a result within that because he was trying to count millions of tokens, and every 3 minutes, he would try again. This led to a drastic slowdown of the entire product delivery even though the API was not used yet. Thus, the decision was made and removing the DONE state from the cache and not allow users to interact with that state. It already has no use case for moving or doing anything with tokens in the DONE state.

5.3 Important parts of the project

New software is a crucial success factor for many companies. It can deliver critical efficiencies and improve overall operations across the board. Even though it is a crucial process and many companies devote a significant amount of their attention to it, it is a complex process. Working in a larger team requires some overhead, such as planning and coordination.

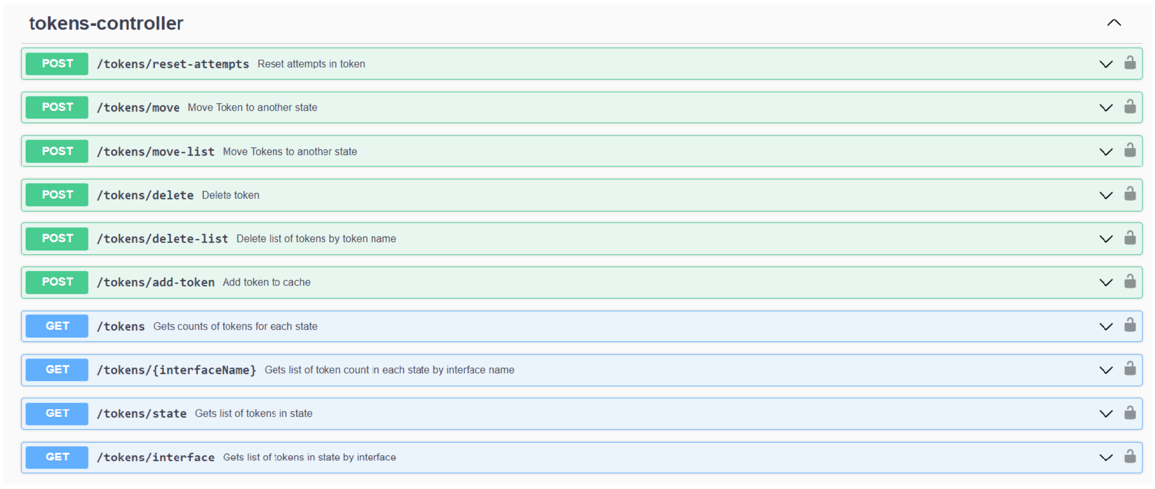
This section is dedicated to the essential parts of the system developed. These core modules play an important role in overall functioning. The modules listed below interact with each other to form the overall system.

5.3.1 Delivery Engine API

It would be easier to find something more important to the functionality of our system than the basic API. It is the basic building block for further work and was also the first part to start working on after the theoretical preparation. The development process primarily started from the design phase, when it was decided what endpoints we would design and what tasks they would perform. During the development process, many problems arose, and many of the early assumptions were wrong and had to be adapted to the environment where we would implement the system.

As with the rest of the platform, Swagger was used along with *Java* to facilitate testing of the API being created during implementation.

The implementation was also accompanied by learning about previously unknown features of each tool, which are essential for platform development in larger companies. During this part of the development, the Models and Entities used there were developed in parallel.



tokens-controller

```
POST  /tokens/reset-attempts  Reset attempts in token
POST  /tokens/move  Move Token to another state
POST  /tokens/move-list  Move Tokens to another state
POST  /tokens/delete  Delete token
POST  /tokens/delete-list  Delete list of tokens by token name
POST  /tokens/add-token  Add token to cache
GET   /tokens  Gets counts of tokens for each state
GET   /tokens/{interfaceName}  Gets list of token count in each state by interface name
GET   /tokens/state  Gets list of tokens in state
GET   /tokens/interface  Gets list of tokens in state by interface
```

Figure 5.2: Created endpoints that ensure the functionality of the system.

5.3.2 Swagger Client

After completing the first version and writing tests dedicated to the created interface, it was necessary to start thinking about generating the client itself, which will be used for communication between the Delivery engine and Dashboard and, as it turned out, between `DE_Light` and `Delivery Engine`. The process started with figuring out what arguments to choose for the generation. The swagger-codegen itself moved pretty quickly, and over the phase where different options were tried, the documentation of the Swagger itself changed, and more options were added.

After some time of testing the client, it was decided to add it to Jenkins. The setup was relatively smooth thanks to the interface that Jenkins provides. The next stage was to change the architecture in which the client is generated. The change is that only the un-generated part of the client is uploaded to BitBucket itself, which is generated on the server every time the Delivery engine is run. This is achieved by a `swagger-codegen-maven-plugin`, which is part of the Maven plugin library. This led to the automation of the generation process, which was inevitable.

The not generated code, which contains the configuration of the client in question, had to be created to set the values necessary for communication correctly. The first version of the application contained an interceptor, which was there because of the work of the `DE_Light` library. Here, due to the cluster architecture, it is not allowed to call a load-balanced endpoint. It would have resulted in not being able to target a specific server either for getting data using GET type commands, whereas in a dashboard where data would be aggregated would break the whole concept. Thus, none of the endpoints would be able to achieve an operation, such as accessing a specific file system where a given token is expected. That is because running individual interfaces and thus creating tokens on specific server file systems works based on load balancing.

5.3.3 Dashboard API

The first important task within this API was to deal with load balancing, where it was inevitable to achieve the ability to access a specific server using the `@OverrideHost` annotation. In our case, the procedure was that we initially overwrote the postProcessAfterInitialization function, which belongs to the BeanPostProcessor. Thanks to this, we could mark those beans that contain our annotation by adding advice as an interceptor.

This interceptor implements the following:

- **Method Interceptor** - This is an interceptor that is meant to define the behaviour to be executed before or after a method call.

  In our case, this was to take advantage of the fact that at the moment of invocation, the `ThreadLocal` variable is set to the value of the parameter annotated by the @OverrideHost annotation, the function from which the invocation is made. It also takes care of removing the value after completion.

- **ClientHttpRequestInterceptor** - This interceptor allows you to define the behaviour to be performed before sending an HTTP request or after receiving an HTTP response.

  In our case, it performed the function of the interceptor itself, it worked by breaking the request that came in here down into individual URI parts and modifying it based on the circumstances of the call. For example, the lack of context in various microservices played a role here. Fortunately, thanks to the platform environment, it was possible to get the information from the sprint or from the URL of the original call.

Then it was necessary to start the aggregation itself, which was done quite specifically for each endpoint in the case of GET requests. It was done by creating a unique bean used in the aggregation to target each server specifically.

5.3.4 Frontend

This section will describe the frontend creation for our system, which was proposed in the previous section. It is an integral part of our application as it is a graphical interface that allows the user to interact with the functions of our application programming interface and display information. The frontend includes all the elements that the user can see.

When creating the frontend, the main requirement is knowledge of technology. In this case, knowledge of the *React framework* and *Typescript* programming is at the core. In the implementation, it was necessary to emphasize the intuitiveness and design of the interface.

In this section, we discuss the implementation process, which in the first moments took place in the Storybook application, where the individual components were tested. Which are sort of basic building blocks that are modular. Our API consists of two Token and state, both of which have different roles. When you click on a particular state, it aggregates the server data and returns the tokens corresponding to that interface for all servers. These two components were created first.

This was followed by the creation of the actual windows that display the response based on the tokens and the creation of a window that contains the detail of the token along with the action options associated with the token. This was completed by implementing modal windows that are used to confirm the actions. The implementation used hooks and states,

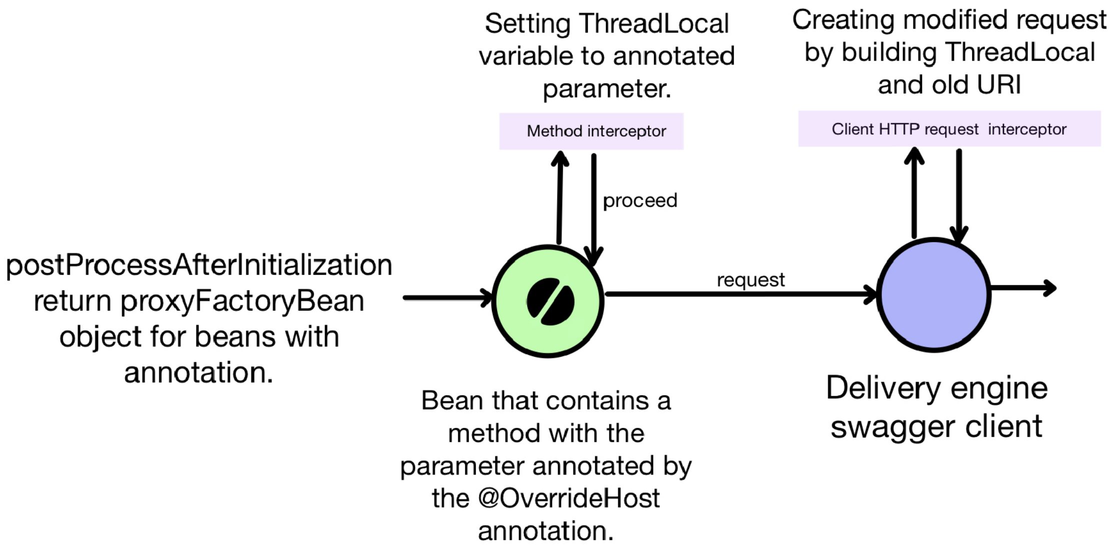

Setting ThreadLocal variable to annotated parameter.
Method interceptor
proceed
Creating modified request by building ThreadLocal and old URI
Client HTTP request interceptor
postProcessAfterInitialization return proxyFactoryBean object for beans with annotation.
request
Bean that contains a method with the parameter annotated by the @OverrideHost annotation.
Delivery engine swagger client

Figure 5.3: The way the request reaches the specific server.

which are essential elements of React. The Material UI library was used for styles and pre-made components, which made the creation process more manageable.

5.4 Technologies Used

In order to connect the new API to the system, it was necessary to get acquainted with the technologies used in addition to the system itself and its functioning. The correct language and tools are essential elements in any development.

This section will be dedicated to describing the technologies and tools I used within the implementation of the new API. These technologies were chosen based on technologies that the rest of the platform uses or it is a custom in the platform development environment to use it.

Therefore, the following list contains the technologies primarily used in the process of creating and writing code. These technologies were used to integrate this interface and create a functional and seamless system.

5.4.1 Java 8

Java [36] is a widely used programming language. Java 8 first release was in 2014. In this time, it has brought several innovations. For example, it has significantly impacted data processing, allowing for use in working with large amounts of data. The next feature was the support for parallel data processing. These are some reasons why Java 8 is still a prevalent choice in application development.

One of the main competitors is *C#*, developed by Microsoft and part of the *.NET Framework*. Even though the choice depends to some extent on the preferences and specific features of the chosen project, there are differences in the language choice that cannot be ignored. One of Java's leading and significant advantages is that it is a multiplatform language, thanks to which an application developed in this language can run on different operating systems, unlike the other one that runs on the .NET Framework and is therefore connected to windows. Another advantage is, for example, multi-threaded programming, or we can also count among the advantages the broad community of developers that Java 8 has.

These and many others are the advantages that make many companies, including the middleware team at Onsemi, choose Java to develop their own applications.

5.4.2 Spring Framework

Another technology that the team uses in their platform and that had to be familiarized with in order to start the implementation was Spring [37].This framework aims to reduce the dependencies between components and make testing and maintenance of the platform more accessible. Spring, as such, works on the principle of IOC or "Inversion of control", which changes the application's flow control so that objects in the system are created and processed by the framework and are also assigned dependencies on other such objects.

The technique Spring framework uses for IOC is DI or "Dependency injection", a process where an object's given dependencies are created externally instead of the object creating them itself. This increases the modularity of the application.

The framework is modular. That is, it offers modules that are designed to be integrated into different applications. Here is a description of the most critical modules found in the framework.

- Spring Core - As you can tell from the name, this is the module that contains the core of the Spring Framework, so it contains DI, IOC and other features.
- Spring MVC - Provides MVC architecture for building applications. It separates modules for logic and appearance.
- Spring Data - This module caters for the work of data warehousing. This module allows you to switch between the individual data handlers without having to change the code.
- Spring Security - This module takes care of authentication and authorization. Examples include use in website security, single-user account management, and more.

Thus, Spring Framework is used to facilitate the development of robust enterprise applications for Java. This works because of the practices mentioned above that Spring uses.

5.4.3 React

This library is used to create user interfaces for web applications like ours. It was released in 2013 by Facebook. The library is open-source, making it easier to manage application states and create state-based interfaces. It can also be used with libraries and frameworks like Redux or Angular. This library is described in the book [35].

In React, the functioning is based on components. These blocks can nest within each other and be assembled into more complex clusters of components. As such, components are written in *Javascript*, and their output indicates how the output will be displayed.

React itself also uses JSX, which stands for *Javascript* XML, which serves as an extension to clickable *Javascript*, and allows for syntax extensions so that HTML-like code can be written. This is then compiled into plain *Javascript* code. Among other things, it includes additional features such as hooks or lifecycle methods, for example.

5.4.4 TypeScript

It is a programming language which is a superstructure of the JavaScript language. The extension consists of several significant additions that improve the previous language. TypeScript is compiled into pure *JavaScript* at compile time, which runs on all browsers that support it.

Typescript provides many extensions, the most important of which include type specification, class inheritance, and others.By using this extension, it is possible to achieve a lower error rate during development, increasing the application's overall reliability. At the same time, it achieves ease of working in a team.

5.4.5 StoryBook

This tool is used to create and test isolated components. The whole process is shown in [1] so that the developer can create components, which are then displayed and tested in an isolated environment. That makes it unnecessary to run the entire application.

In our case, it is also used because of the support for React. At the same time, various testing tools can be integrated.

All in all, we can say that it serves mainly developers, for whom it simplifies the work spent on development.

Chapter 6

Testing

This section is the last one, and it deals with one of the essential parts of the development process, namely testing. The goal of testing is to ensure that the functionality currently in the development phase is working correctly and fulfils the specific requirements defined at the beginning of the development. Through this process, deficiencies or bugs can be detected early, resulting in a reduced likelihood of problems in the future.

Many methods of testing can be performed on a system under development. The most basic ones, which are also the easiest, are tests that are performed manually. These tests serve as a sort of baseline or as tests of specific situations. However, this testing is not efficient. An alternative option is automated tests. These, although in this case, require some knowledge regarding the technology, are faster and have the possibility of repeatable execution, which leads to saving time by manual execution.

At the beginning of the build, it is a good idea to create a good test plan covering what will be tested to achieve the lowest possibility of errors. At the same time, this plan aims to ensure the highest quality and reliability of the tests. This is achieved by choosing different types of tests and complexities. There are many types of tests within the test plan. There are, for example, tests for integrations, performace, security and much more. For our work, we used only Unit tests to test if important functionality works correctly.

As mentioned here, proper testing is the key to successful deployment into production. It is through this that software can be achieved that can be safely uploaded into a production environment with the maximum possible elimination of future bug patterns. Ultimately, testing will also reduce the cost of having to spend on fixing specific patches or defects and possible functionality modifications.

6.1 Testing new functionality

In our case, when the development started from the API, which we implemented in a microservice called the Delivery Engine, and we also implemented a swagger in parallel with the individual endpoints, it was relatively easy to run them. Thus, there was continuous manual testing. This was also because, quite often, thanks to code reviews from various colleagues, there was a re-discussion regarding the individual endpoints.

However, each section was followed by writing automated tests written in Groovy and using the *Spock framework* [30]. This testing was done based on testing the essential parts of the system so that they would show possible affections in future development. Overall, this testing also fits into the CD/CI 3.2.2 process. Their use in these processes works so that they can regularly integrate into the system in the Continuous Integration process to see if changes affect the rest. At the same time, Continuous Deployment has a role in these tests because when code is automatically integrated into the production environment, it must be checked that the system continues functioning as it should.

Among the tested parts is TokenCache, which is a kind of information backbone for working with the file system for performance reasons. It is therefore important that all functions that cater for operations on top of it work as reliably as possible. Another such is TokenController, these tests are important for example to check for correct return codes in different scenarios. Other testing has already been done by users.

This testing is also related to teaching users how to use the functionality correctly. It is mainly about the possibility of token removal, which despite its minimal use, many users choose as the first choice for solving their problems. For this reason, the interface currently does not find itself on the production but only in an environment dedicated to testing new user integrations. That should avoid the targeted removal of essential production data by unfamiliar users.

Unfortunately, during these tests, the code turned out to contain a bug and therefore the API was pulled from the servers at the time of submission, where it was slowing down the existing system.

6.2 Testing process

As mentioned above, the development process itself was mainly accompanied by manual tests, which were very inefficient and always followed by the creation of automated tests at the end of the first version. In the case of our development, these are purely Unit tests that accompany each build. These tests are designed to avoid as much as possible the possibility of interference.

Currently, with the launch of the Delivery engine, a group of tests is run to check the work of the Cache and Controllers. Other tests are at the Dashboard level, so again, unit tests. This checks if the Aggregation service works correctly. The next object tested is the interceptor that connects to the correct server. There the function that caters for the valid URL is checked.

Further tests were performed by deploying on a non-production environment, and as mentioned above, they were unsuccessful because they revealed problems that were not detected during local development.

Chapter 7

Conclusion

The aim of this application was to get acquainted with the technologies related to the development of middleware platforms, together with the principles on which corporate systems that usually depend on these platforms operate. Another element of the assignment was also to learn about the technologies that are used. All of these requirements were met within the first chapters of this thesis. For these phases, the middleware team of Onsemi itself was heavily involved and took the place of the tutor in this phase.

Based on this experience, a discussion was then started about the needs for the development and the future extensibility of the platform. After familiarising myself with the basic functionality and operation of the current system, a plan for the future of the platform itself was outlined to me. Whereby with the increasing demands on middleware due to the company's growth, any opportunity to hand over the overhead of their data to users is beneficial. Also, to avoid overloading administrators with tasks that are primitive. For this reason, too, the topic of my thesis was decided.

My work has been honestly directed throughout the development by the programmers of the Onsemi team, who regularly prepared code reviews and gave me feedback and planned further development with me, also thanks to this, all the points of the assignment we set during the assignment were achieved.Unfortunately, due to an error that occurred during deployment in the corporate environment, the application was withdrawn and is therefore out of the environment at the time of submission of this work and needs to be fixed.

Nevertheless, created application facilitates the middleware team's work after its implementation in the system. Users will be able to manage their tokens. The presence of this API in the system's user interface will save the administrators several tens of minutes every day. At the same time, communication time will be saved, which is less efficient than using a text-based communication platform and takes more time than solving the problem itself.

A large part of the work is devoted to the application platform itself. This was because it was a source of considerable development slowdown, as it took an uninitiated programmer time to understand the overall workings. The main reason for this was to implement parts of our module so that, as far as possible, there was no slowdown or affectation of the system. The implementation as such on the local device went fine, unfortunately, it failed just before the submission of this work. It is, therefore, currently out of service. However, it is a matter of a short time to get this interface back up and running and then integrate it back into the corporate environment.

Bibliography

[1] Agutu, M., Shilman, M. and Nguyen, D. *Designing with Storybook: Create scalable design systems that empower your team to build great user experiences.* Pragmatic Bookshelf, 2021. ISBN 978-1680507035.

[2] Ahmed, S., Awan, M. A., Ikram, A. and Ullah, S. Continuous deployment and continuous integration: A systematic review on the latest developments in the literature. *IEEE Access*. IEEE. 2019, vol. 7, p. 85808–85822.

[3] Ali, N., Gao, L., Wang, X. and Li, L. Asynchronous Communication Between Microservices: The Service Mesh. *IEEE Cloud Computing*. IEEE. 2020, vol. 7, no. 6, p. 64–71.

[4] Alizadeh, E., Razak, S. A., Shojafar, M. and Gani, A. Performance evaluation of JSON and XML for big data storage and processing. *Journal of Big Data*. Springer. 2015, vol. 2, no. 1, p. 1–18.

[5] Atlassian. *Jira* [https://www.atlassian.com/software/jira]. 2002. Accessed: 2022-04-22.

[6] Atlassian. *Confluence* [https://www.atlassian.com/software/confluence]. 2004. Accessed: 2022-04-22.

[7] Beck, K. and al. et. *Manifesto for Agile Software Development*. Agile Alliance, 2001.

[8] Boateng, R. and Siaw, F. User Authentication and Authorization in the Digital Age. In: *Handbook of Research on Human-Computer Interfaces, Developments, and Applications*. IGI Global, 2018, p. 79–98.

[9] CI, J. *Jenkins* [https://www.jenkins.io/]. 2021. Accessed: April 22, 2023.

[10] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M. and Pricl, S. Monolithic to microservices: An experience report from the banking domain. *Journal of Systems and Software*. Elsevier. 2017, vol. 131, p. 81–92.

[11] Fowler, M. Microservices: a definition of this new architectural term. *Martinfowler.com*. 2014, vol. 25, p. 2014.

[12] Gallo, M. How to Use Bitbucket: A Guide for Beginners. *The Startup*. April 2020. Available at: https://medium.com/swlh/how-to-use-bitbucket-a-guide-for-beginners-94339b0a9f9c.

[13] Gorschek, T. and Wnuk, K. Software requirements prioritization. *IEEE software*. IEEE. 2006, vol. 23, no. 2.

[14] HashiCorp. *Consul Documentation* [https://www.consul.io/docs/]. 2021. Accessed: April 22, 2023.

[15] Humble, J. and Farley, D. Continuous delivery and the world of devops. *IEEE software*. IEEE. 2010, vol. 28, no. 3, p. 18–21.

[16] Jackson, F. *Jackson in Action*. Shelter Island, NY: Manning Publications, 2019. ISBN 978-1-61729-372-5.

[17] Jankovičová, J. Figma: Collaborative interface design tool. In: IEEE. *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. 2018, p. 1–4.

[18] Jespersen, C. and Bergman, S. Continuous integration and delivery of microservices using Jenkins CI. In: IEEE. *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2016, p. 516–522.

[19] Karlsson, J. and Ryan, K. Software development: Agile vs. traditional. *IEEE Software*. IEEE. 2001, vol. 18, no. 5, p. 104–105.

[20] Kim, G., Humble, J., Debois, P. and Willis, J. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[21] Kolias, C., Kambourakis, G., Gritzalis, S. and Stergiopoulos, G. Taxonomy of cyber attacks: A comprehensive study of types, motives and techniques. *Journal of Network and Computer Applications*. Elsevier. 2017, vol. 88, p. 1–10.

[22] Larman, C. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004.

[23] Li, X., Zhu, X., Shen, B., Gao, S. and Liu, Y. Swagger Editor: A Graphical Tool for OpenAPI Definitions. *IEEE Software*. IEEE. 2017, vol. 34, no. 5, p. 87–91.

[24] Newman, S. *Building Microservices: Designing Fine-grained Systems*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015. ISBN 1491950358, 9781491950357.

[25] onsemi. ON Semiconductor. *Onsemi.com*. 2022. Available at: https://www.onsemi.com/company/about-onsemi.

[26] Shah, M. Generating client-side code with Swagger: a complete walkthrough. *Journal of Open Source Software*. The Open Journal. 2017, vol. 2, no. 12, p. 269.

[27] Sharifloo, A. and Tizhoosh, H. R. Cluster-based Architecture for High Availability and Scalability of Web Applications. *Journal of Network and Computer Applications*. Elsevier. 2016, vol. 76, p. 84–94.

[28] Sommerville, I. *Software engineering*. Pearson Education Limited, 2015.

[29] Sommerville, I. and Sawyer, P. *Requirements engineering: a good practice guide*. John Wiley & Sons, 1997.

[30] Spock Framework Contributors. *Spock Framework* [https://spockframework.org/]. 2023. Accessed: April 17, 2023.

[31] Stallings, W. and Brown, L. *Computer Security: Principles and Practice*. Pearson, 2017.

[32] Stefanikova, A. System architecture design: A practical overview. *Procedia Engineering*. Elsevier. 2016, vol. 161, p. 1196–1201.

[33] SysAid. Microservices Architecture: Why Asynchronous Communication is Better. *SysAid Tech Blog* [https://www.sysaid.com/blog/sysaid-tech/microservices-architecture-asynchronouscommunication-better]. April 2021. Accessed: April 22, 2023.

[34] Tahchiev, M. and Mitev, N. Building Microservices with Spring Boot and Spring Cloud. *Spring: Developing Microservices with Spring Boot*. Apress. 2019, p. 35–55.

[35] Todorov, S. and Kovačević, J. *React: Up & Running: Building Web Applications*. Sebastopol, CA: O'Reilly Media, 2016.

[36] Ullenboom, C. *Java 8 - The Complete Reference, Ninth Edition*. New York, NY: McGraw-Hill Education, 2014. ISBN 978-0-07-180855-2.

[37] Walls, C. *Spring in Action: Covers Spring 4*. Manning Publications, 2016.

[38] Wang, X., Li, Z., Li, X., Li, Y. and Wu, Z. A survey on microservice API documentation. *Journal of Systems and Software*. Elsevier. 2020, vol. 170, p. 110727.

Appendix A

Contents of the physical medium

The included SD card contains, in addition to the work itself, the most important files that have been created to add new functionality to the system.

- text.pdf - PDF file containing this bachelor thesis
- \source - source codes
- \models - models used in system
- \tests - snit tests used