VSscan

This document has been largely trimmed to a small, digestible report (original size: 106 pages).
Please take into account that the translation has been made by Microsoft Office Translation tool, some lexical errors may still be present.

TOC

Introduction

Context

The production of this report is part of the evaluation framework of the above-mentioned apprentice candidate's end-of-apprenticeship work.

Evaluation

- The applicant is evaluated on his/her project report and presentation.
- The in-house trainer and experts are available to the candidate throughout the evaluation process.
- The candidate's assessment is based on a catalogue of criteria made available to him beforehand.

Workspace

In order to meet the deadline, the candidate has at his disposal a workspace at the training site with all the necessary tools.

Remarks

In this report, the terms application, service, microservice, vsscan, and vsscan-svc all mean the same thing; referring the vsscan-svc microservice developed by the candidate.

Analysis of the specifications

Objectives

The objective of the vsscan project is to create a backend component allowing the applications of the Switzerland State of Valais/Wallis to simplify the process of uploading documents by users. This component will allow users to scan documents using their mobile phones and upload them directly to the platform, without having to go through complex file exchanges.

Project Management

Planning

Gantt and Kanban framework of the AGILE methodology.

Session Management

Session management is done via tokens called **JWT (Json Web Token).**

The application of JWTs is, at best, done according to the best practices defined by the IETF RFCs:
JWT: RFC 7519 – JSON Web Token (JWT) (ietf.org)
JWK: RFC 7517 – JSON Web Key (JWK) (ietf.org)
JWS: RFC 7515: JSON Web Signature (JWS) (rfc-editor.org)
JWE: RFC 7516 – JSON Web Encryption (JWE) (ietf.org)
JWA: RFC 7518 – JSON Web Algorithms (JWA) (ietf.org)
JWT best practices: RFC 8725 - JSON Web Token Best Current Practices (ietf.org)

What is a JWT?

A JWT can be perceived as a bank check:

- It contains a **payload** (payload that contains the amount of the check, the different parties, and other useful information) that is **dated and signed**.

A JWT is composed of a **header**, a **playload,** and a **signature**.
Each section is Base64 encoded. Example:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1MTYyMzkwMjJ9.tbDepxpstvGdW8TC3G8zg4B6rUYAOvfzdceoH48wgRQ

Header

In a header there are typically 2 attributes:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- The algorithm that signed the JWT.
- The type of structure that the JSON object represents.

Reference: Different types of structures: RFC 7519 – JSON Web Token (JWT) (ietf.org)

Payload

The attributes of the payload are called **claims**.

{
"sub": "1234567890",
"name": "John Doe",
"IAT" means 1516239022
}

Unique identifier of the JWT.

iss	Issuer	Issuer of the JWT.
sub	Subject	Recipient of the JWT.
aud	Audience	Intended recipients of the JWT.
exp	Expiration time	Token expiration time.
nbf	Not before	Time before which the JWT is rejected.
iat	Issued at	Time when the JWT was issued.
jti	JWT ID	Unique identifier of the JWT.

Signature

The signature includes:

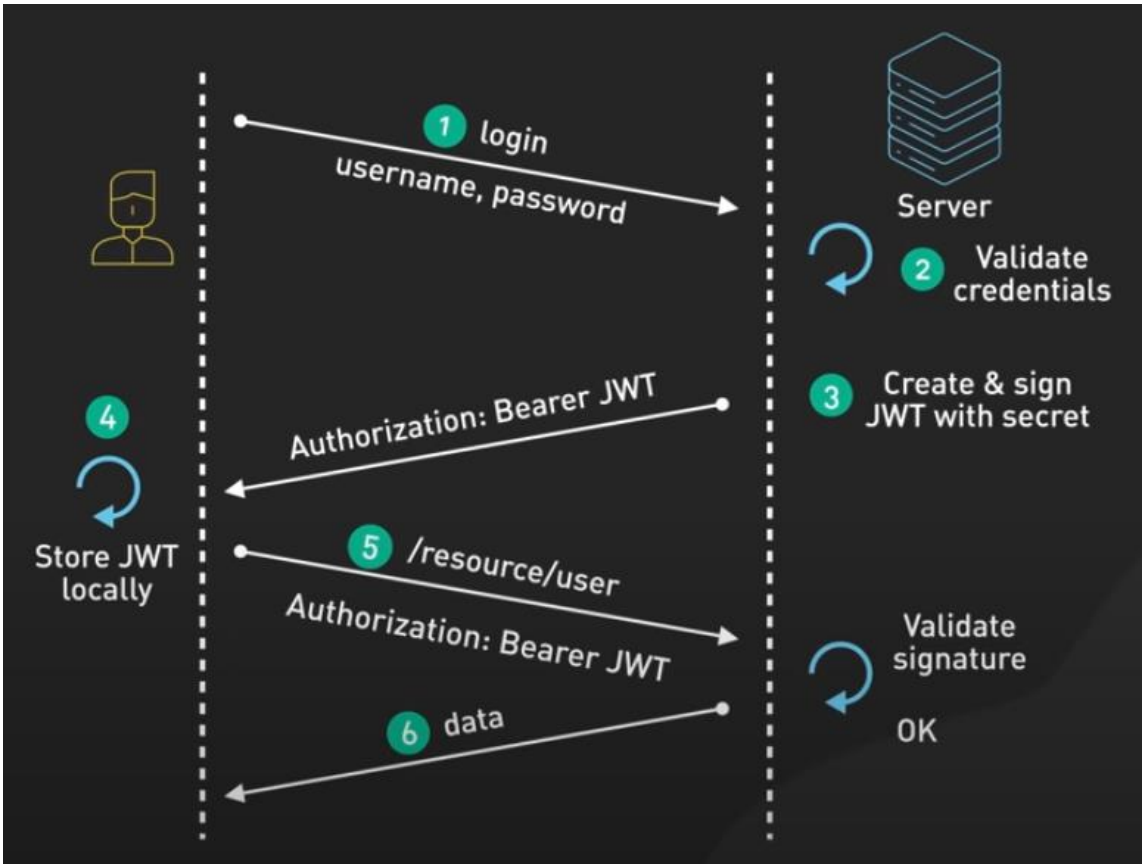
- The Base64 encoded header.
- Plus, the Base64 encoded payload.
- Plus, the key from a digital signature algorithm.
- All encrypted by a digital signature algorithm.

There are 2 types of signature algorithms:

1. Symmetric algorithms (generates a secret key).
2. Asymmetric algorithms (generates a key pair).

Life cycle of a JWT

1. The user requests a JWT from the authentication server.
2. The server validates the user's credentials.
3. The server creates, signs, and sends a JWT:
   eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
4. The client stores the JWT securely.
5. The JWT now acts as a passport. As soon as the client wants to communicate with the server, it must now pass its JWT in the header of its request.
6. This allows the server to verify the authenticity and integrity of the JWT.



Reference: bytebytego.com

Actually, the creation of a JWT is done using the JWS or/and JWE:

- **JWS (JSON Web Signature)** ensures the integrity of the payload, which means that it verifies that the contents of the JWT have not been tampered with during transmission.
- **JWE (JSON Web Encryption)** provides encryption, ensuring that the JWT payload remains confidential and cannot be read by unauthorized parties.

What is a JWK?

A **JWK (JSON Web Key)** is a JSON structure that represents a cryptographic key.
(the public key of an asymmetric encryption key pair, or a secret key).

What is a JWA?

The IETF JSON Web Algorithm (JWA) **specification** describes the digital signature and encryption algorithms that can be used by a JWS and JWE.

To sum up

A JWT (token), JWK (key), JWS (signature), JWE (encryption) and JWA (algorithm) are all JSON structure standards that allow each to describe different specifications useful for creating a JWT.

Advantages / Disadvantages

Feature	Disadvantages
The JSON format is easy to read	Sensitive data in the payload needs to be encrypted.
Stateless authentication	- Vulnerable to man-in-the-middle attacks. - Impersonation.
JWS, JWE security	- Too large of a payload size if too much information is included. - Vulnerable to brute-force attacks if weak/obsolete algorithms are used.
Ease of use	Incorrect usage if poorly implemented.

Risks

HTTPS	Transmit JWTs via HTTPS to prevent interception.
Signature validation	Verify the JWT's signature to ensure it was issued by a trusted authority and has not been tampered with.
Expiration	Include an expiration time in JWTs to limit their validity period.
Short-lived token	Issue short-lived JWTs with relatively short expiration times to reduce the window of opportunity for attackers. Implement token refresh mechanisms to obtain new tokens.
Sensitive information	Avoid including sensitive information (such as passwords or personally identifiable information) in the JWT payload.
Access control	Verify that the JWT contains the necessary claims (iss, sub, aud, exp) before granting access to protected resources.
Algorithms	Verify that the encryption or signing algorithms are not obsolete. Verify that the JWT algorithm is correct and unmodified.
Keys	Protect the keys used to sign JWTs to prevent unauthorized access or misuse.
Secured token storage	Store JWTs securely on the client-side, using mechanisms such as HTTP-only cookies, local storage, or session storage.

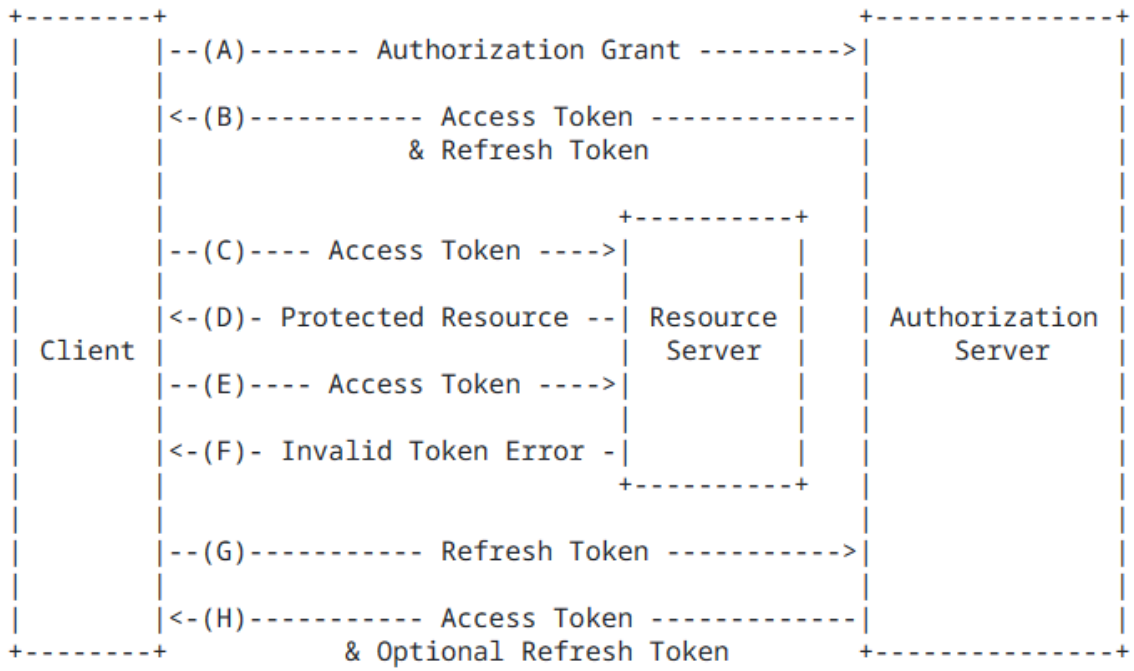
Key ID	Verify that the kid in the JWT header does not allow SQL injection.

0Auth2.0

The implementation of a JWT is done under the specifications defined by the "0Auth2.0 authorization framework" protocol which allows a third-party application to obtain limited access to an http service.
Reference: RFC 6749 - The OAuth 2.0 Authorization Framework (ietf.org)

It defines the management of **access tokens** and **refresh tokens** , both of which are represented as JWTs:

- An **access token** represents credentials that allow access to protected resources.
- A **refresh token** represents credentials used to obtain an access token.



Source: RFC 6749 - The OAuth 2.0 Authorization Framework (ietf.org)

A. **The client requests an access token** by authenticating itself to the authorization server and presenting its access rights.
B. **The authorization server authenticates the client** and validates its credentials and **issues an access token and refresh token**.
C. **The client makes a request for protected resources** from the resource server **by presenting the access token**.
D. The resource server validates the access token and processes the request.
E. Steps (C) and (D) repeat until **the access token expires**.
If the client knows that the access token has expired, it goes to step (G); otherwise, it makes another protected resource request.
F. Because **the access token is invalid**, the resource server returns an invalid token error.

G. The client requests a **new access token** by authenticating to the authorization server and **presenting the refresh token**.
H. The authorization server **authenticates the client**, **validates the refresh token,** and issues a new access token.
The session ends when the refresh token expires.

Peculiarities

1. Typically, a refresh token is never sent to the resource server.
   But within the framework of this project, it was concluded that only one microservice would be to be built and therefore in charge of authentication, authorization and file transactions.
2. Normally, an endpoint (access point of an API) is available to list the different JWKs of a JWKSet (set of JWKs), so that the resource server can validate the integrity and authenticity of the JWT.
   However, this approach requires the time-consuming installation of security measures. In addition, the application of this standard is used with the generation of asymmetric key pairs.
   If a public key is compromised, it will not be able to generate valid JWTs.
3. The claim sub must match the identifier of the resource server.
   Since we don't have multiple resource servers, we'll use the claim sub as the session ID.
   Reference: RFC 9068 - JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens (ietf.org)

Implementation in vsscan

As specified in the specification: "The JWT token is secured via a symmetric key, stored as a JWK in the database."

- Access tokens will therefore be signed by a symmetric key using the HS256 signature algorithm.
- The symmetric key is stored as a JWK and persisted in the database.
- A symmetric key is generated for each session.
- A refresh token is signed using the same symmetric key.
- The access token has a short validity of 5 minutes.
- The refresh token is valid for 1 hour (session validity period).

Uploading files

Limitations PostgreSQL

Since the files are backed up in a PostgreSQL database, the limitations must be defined:

Postgres specs

Maximum size for a database? unlimited
Maximum size for a table? 32 TB
Maximum size for a row? 400 GB
Maximum size for a field? 1 GB
Maximum number of rows in a table? unlimited
Maximum number of columns in a table? 250-1600 depending on column types
Maximum number of indexes on a table? unlimited

Source: PostgreSQL: Documentation: 16: Appendix K. PostgreSQL Limits

- Only dealing with small file sizes, we won't have any problems with backing them up.

Quarkus Limitations

The default size for file transfers is 10MB per file. Which is reasonable.
Source: All configuration options - Quarkus
The real limitation we need to consider is the behavior of our application during file transfers.

- If Person A sends a large amount of files, the process allocated to my application crashes until the operation completes. Once the operation is complete, Person B can send their files.

To overcome this problem, the quarkus team has released a new execution model that they call the "Reactive Execution Model".
Knowing that this type of model has been around for a long time now.

Worker threads VS I/O threads

There are two main changes to this delivery model:

1. The use of non-blocking I/O threads rather than worker threads.
2. The use of an event loop.

The worker thread architecture allows the processor to easily handle compute-intensive operations. On the other hand, these same threads are bad when running I/O tasks.
I/O threads are only used for information input and output, but are bad at computational operations that require CPU resources.

Process

A process is a program that runs on the operating system. It has its own memory and cannot access the memory of other programs. Within a process, only one task can be performed at a time.

Hearts

A processor has sockets (physical location on the motherboard), cores (physical component on the processor), and logical processors (comes from Intel's hyper-threading technology, where 1 core has 2 logical processors).

To summarize, the number of operations that a processor can perform in parallel depends on the number of logical processors in the processor.
Source: What is Hyper-Threading? - Intel

Threads

A thread is a logical entity that we've associated a task with, and it shares the resources of a process.
The threads that quarkus uses are threads created at the operating system level. This means that it is up to the OS to choose how to distribute threads between the different cores.
The operating system also has its own threads, which take care of interfacing the hardware components and bringing the data to the applications and vice versa.

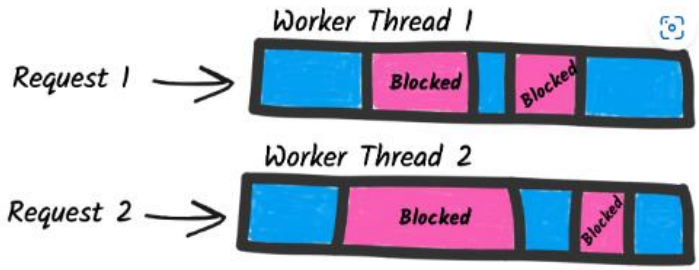
Sockets

Sockets are instances that correspond to endpoints through which processes can communicate (for example, the network interface controller that allows communication with the hardware (previous point)).
Example: TCP socket, UDP, ...
These sockets are different from physical sockets. Physical sockets are simply sockets to plug our processors into.

Blocking or non-blocking

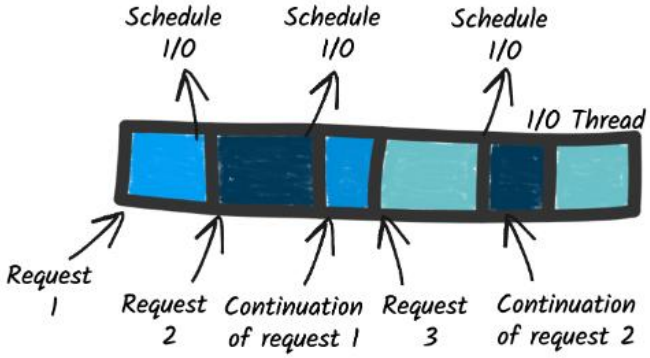
When making a request to a third-party service, we must wait for the response from that service:

- While waiting for our response, a worker thread will be blocked until it has received the response to the request.



Source: Quarkus Reactive Architecture - Quarkus

- While waiting for our response, an I/O thread will be put into a sleep state and will be woken up when the data is ready to be retrieved. This means that it doesn't consume CPU resources while it sleeps.



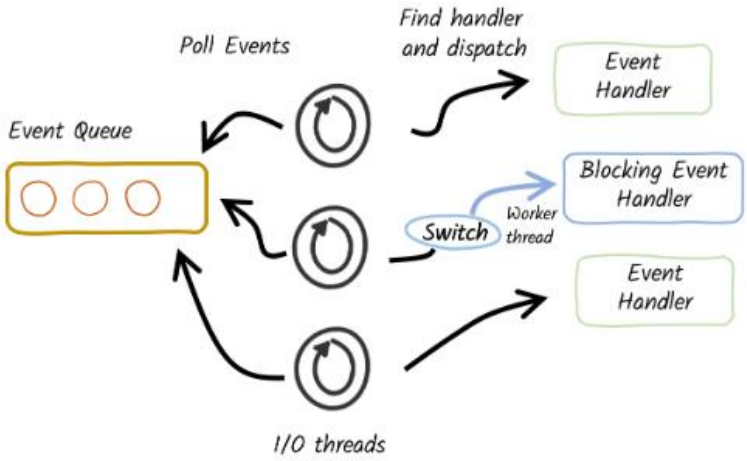
Source: Quarkus Reactive Architecture - Quarkus

Event loop

The event loop is a loop that is constantly spinning and looking for threads to finish their task.

- When a thread finishes its task, it sends an identifier to a waitlist, which will be taken into account by the event loop and will be distributed in a call stack (at the application level) when its turn comes.
- Using this identifier, the event loop notifies the program to which it tells where to find the result of the query.

> The loop does not rotate all the time, and its state is managed at the kernel level, but that is beyond the scope of this report.



Source: Quarkus Reactive Architecture - Quarkus

Advantages:

1. There's no need to monitor and maintain the status of thousands of tasks running in the background.
2. We can handle more concurrent connections.

Risks:

1. If the call stack (at the application level) is blocked with a task that includes intensive operation, no further action can be performed.
2. Computationally intensive tasks should always be distributed to worker threads.

Analogy of a reception

If we go to the doctor, we must wait in line and wait our turn at the front desk.
When it is the patient's turn, he must fill out a paper and indicate his information, identity, etc.

The problem:

If every patient must do the same, the queue is blocked at each patient.

- If we add a receptionist, we can halve the number of patients waiting, etc. This is called the "Thread Per Request" model.
  But this model consumes a lot of resources since each request must be allocated sufficient resources to the process so that the threads can take care of the task. You also need to delete and create a lot of threads at once.

The solution:

- A limited number of receptionists were set up, each of whom was responsible for welcoming their first patient. The receptionist gives the paper to fill out to the first patient and then moves on to the next patient without waiting for the first patient to finish entering this information.
- When the first patient has finished filling out the paper, the receptionist takes care of receiving the paper and puts it on the list of patients ready to be checked by the doctor.
  This is called an event loop with non-blocking I/O threads.
  - The event loop is the system that retrieves the list of patients who have completed their paper and transmits the information to the doctor.
  - Receptionists act as threads.
  - The doctor fits our application.
  - The non-blocking concept corresponds to the fact that receptionists do not wait.
  - And the competition is that receptionists can "take care of multiple patients."

To use the Reactive Execution Model, we just need to use the "Reactive" (or "Rest" since 3.8) extensions of quarkus. This way, client file uploads won't block our micro-service.
Source: Quarkus Reactive Architecture - Quarkus

Authentication

The authentication chapter explains how different services communicate with each other in a secure manner.

This chapter is intrinsically linked to the management of sessions , both of which are part of the security discipline of Identity and Access Management (IAM).
The IAM solution used in the Valais is Redhat SSO, which is an application using the **Keycloak authentication protocol**.

- The creation of the identity of the vsscan microservice and the management of access to this microservice is managed by the project manager.

Introduction (Keycloak)

Keycloak is an open-source tool for integrating stable and convenient IAM services into applications.
It includes the standard **0Auth2.0**, **OpenID,** and **SAML** protocols. This means that keycloak can handle SSO**,** authentication**,** and **authorization** processes.

0Auth2.0 and OpenID Connect

0Auth2.0

0Auth **only allows** devices, APIs, and servers to communicate via access tokens.

OpenID

OpenID is an **additional layer of identity on top of the OAuth 2.0** security layer.

The OpenID transaction procedure is the same as the OAuth 2.0 authorization workflow. The significant difference is an "id-token" instead of an access token that allows user **authentication**.

Id-token vs access token

Since oidc (OpenID Connect) is based on 0Auth 2.0, oidc will also provide access tokens.

- An id-token is represented as a JWT.
- Access tokens are used as bearer tokens.
  A bearer token means that the bearer (who holds the access token) can access the authorized resources without any other identifications (**not necessarily a JWT**).
- An id-token represents **who** the person is, whereas an access token (bearer token) represents what that person has access to.

Process Differences

- In 0Auth 2.0, when a user wants to log in, they will be redirected to the login page, or a new pop-up page will appear for authorization.

- In OpenID, whenever a user wants to sign in to a third-party app, they must enter their **OpenID credentials** into those third-party apps. After that, the third-party app will redirect the user to the OpenID provider to confirm the login process.

Practical examples:

0Auth 2.0
If you've already signed up for a new app and agreed to let it **automatically search for new contacts** via Facebook or your phone contacts, then you've probably used 0Auth2.0.
This standard provides secure delegated access. This means that an application can **take actions or access a server's resources on behalf of the user**, without the user having to share their credentials. It does this by allowing the identity provider **to issue tokens to third-party applications** with the user's approval.

OpenID Connect
If you've used your **Google account to sign in to apps** like YouTube or Facebook to sign in to an online shopping cart, you're familiar with this authentication option. OpenID Connect is an open standard that organizations use to **Authenticate users**. The IdP use it so that users can log in to the IdP, and then go to other websites and apps **without having to log in or share their login information**.

SAML
You've probably already experienced SAML authentication in your work environment. For example, it allows you to **Connect to the intranet** or the IdP of your business, and then **Access to a wide range of services** without having to re-enter your credentials. SAML is an XML-based standard for **exchanging authentication and authorization data between identity providers and service providers** to verify the user's identity and permissions, and then grant or deny access to services. This is commonly known as the ancestor of oidc and still largely used.

Summary

OAuth 2.0
Used for authorization.

OpenID Connect
Used for authentication.

SAML
Used to exchange authorization and authentication data between IdPs and the SP.

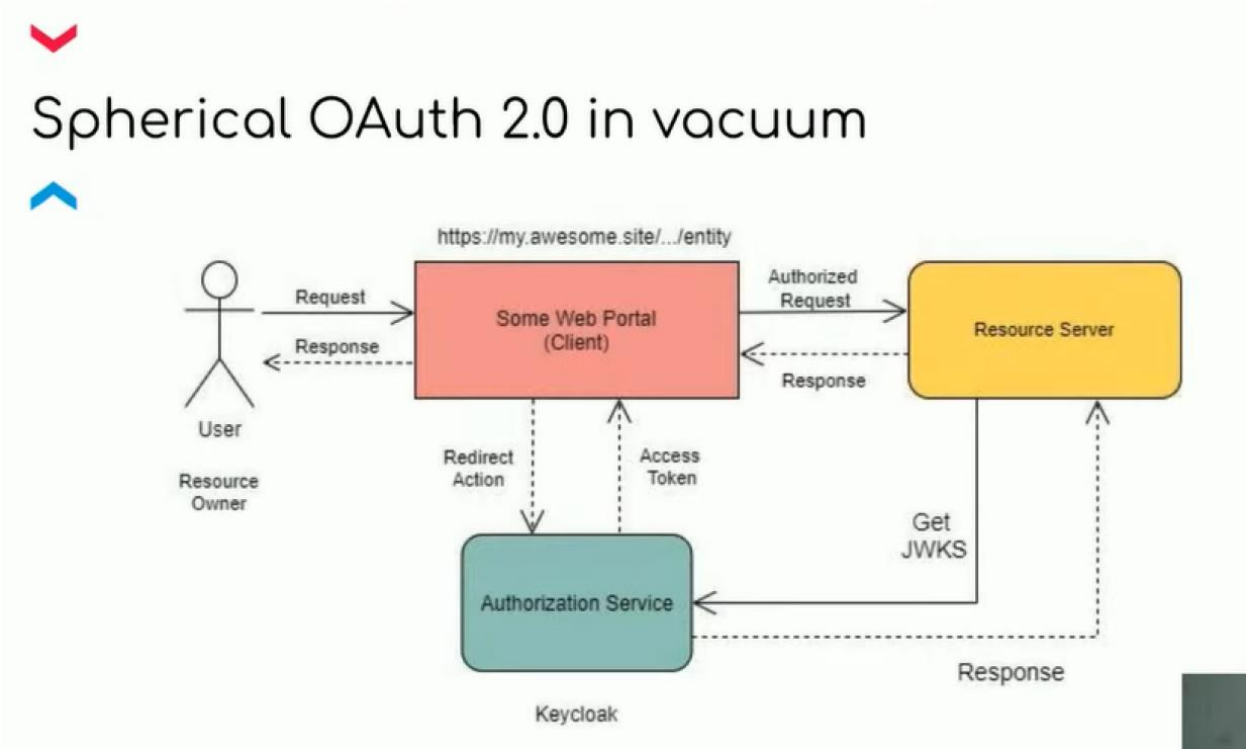
Id-token
JWT containing user information.

Access token
A bearer token that manages the user's access.

Identity provider (IdP)
Manages and stores credentials.

Service Provider (SP)
Trust and authorize IdPs.

Keycloak



A micro-service of some kind.[How to architect OAuth 2.0 authorization using Keycloak | Enable Architect (redhat.com)]

1. The user logs in through the portal.
2. Then, it requests resources from a client.
3. Keycloak validates the client and returns an access token if the client does not have one.
4. The client uses this token to request resources from the resource server.
5. The resource server looks for the JWK associated with the token to validate its signature.
6. If the token is valid, the resource server returns the information to the client.
7. The client returns the information to the user.

The JWKS requires a key ID in the header of the access token.

Mall Analogy

Keycloak has domains, users, groups, clients, and roles. Keycloak SDK has built-in methods that allow you to make HTTP requests from your microservices.

Keycloack

Think of Keycloak as a shopping mall with departments containing stores.

Customers

Think of any department in the mall as a customer. When you log in to keycloak, you are logging in to a certain department.

Kingdoms

Think of any store in the mall as a kingdom.

Roles

Consider the customers, the cashiers, the people who serve the customers... like roles.

- Cashiers have one role, and customers have another. It is therefore necessary to differentiate users by department and by store.
- For example, if someone can take things for free from the first store, it doesn't mean they can take things for free from the second store.

Login URL

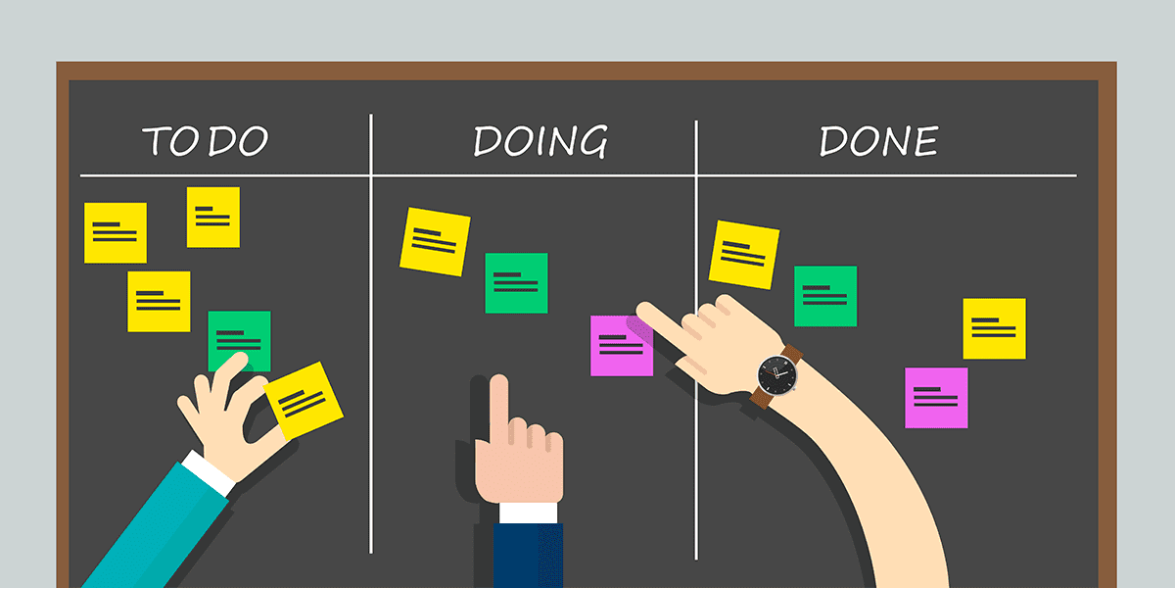
A standard keycloak URL looks like this:
https://someDomain/auth/realms/myRealm/protocol/openid-connect/auth?client_id=myClient...

Planning

Organization

Kanban

As mentioned in the successes and failures section, my ambition was to use the Kanban framework of the agile methodology, which emphasizes a so-called "just-in-time" organization, which allows to indicate the status of tasks in progress.



Source: Kanban: o que é e como aplicar? – Novidá (novida.com.br)

I implemented this method by performing separations in this order:

- **Research** (todo)
- **Realization** (doing/done)
- **Test**
- **QA**
- **Errors**

- The colors red, orange, and gray only indicate a particular status of the task (blocked task, additional time for a task and absent candidate respectively).

Effort & Groups

An organization by type of task has also been carried out to separate the functionalities to be developed.

- This approach offers multiple advantages:
    1. Decouple component integration for testing.
    2. Reorder groups and not each task if reordering applies.
    3. Know how much effort it takes to achieve each feature.

Difference Analysis

Time saved: ~21 hours
Wasted time: ~18 hours
Nb: These calculations have been rounded several times, hence their imprecision, but it gives a sufficiently precise idea of the implementation of this project.

Technical Conclusion

The project was very interesting from a technical point of view. I learned a lot about how to use and implement JWTs and how to sign them correctly.
The threads part of quarkus was really interesting and testing its (reactive) implementation in a microservice was something I had never done before. Moving from theory to practice was more complex since security issues must be taken into account at each iteration.

The project issuer perfectly measured the effort required to complete the project since I was able to complete all the tasks within the planned 88 hours.
In fact, I am writing this conclusion just a few hours before the submission of this report.
The support within the SCI was perfect, my questions were answered without too long a waiting time.

I honestly think that the skills I learned on session management, deployment and how to handle file uploads will be useful in my future activities in the field.

Personal Conclusion

The writing of the report was quite long and it must be said that going from development to writing and then improving the code and the report was quite difficult. I understand why these tasks are separated in business. I think that the definition of the specifications, their analysis AND the management of tasks should not be included in the implementation part of the project. This requires too much introspection in terms of technology and the management of resources, time and effort for the candidate.

The candidate should be able to concentrate on the realization of the project and the organization of it; not temporal, but directed until the complete success of the tasks. Thus, he would have sufficient availability to bring a quality solution to the requested project, and even, allow an evolution of the project and make his creativity, his common sense, the relevance of his reflections speak and understand the full extent of the actions as an application developer without being put under pressure to make a work relevant, fair, well thought out, in a limited time, for which he must define the limits, the needs, the relevance of his remarks, the alternatives and in addition meet structural criteria such as "order the glossary in alphabetical order".
I find that the expectations of the experts are too demanding and irrelevant to define a spectrum of global evaluation common to all projects, including all candidates.

Balance sheet

The work was quite interesting technologically speaking, of course writing a few short conclusions and a balance sheet is not enough to explain the whole context in which the apprentice candidate must meet the requirements presented to him. It was a performance job. There are no personal or intellectual qualities to be provided in this Individual Practical Work, from which one could expect certain reflections from the candidate on his evolution by pushing the comparison on the theory he has seen in class and the final work he has had to submit. A very academic look, but a beneficial and rewarding experience.