*République Tunisienne*
*****
*Ministère de l'Enseignement Supérieur et de la Recherche Scientifique*
*****
*Université de Monastir*
*****
*Institut Supérieur d'Informatique et de Mathématiques de Monastir*
*****
*Département Informatique*

**N° d'ordre : L46_INFO**

# Memory of End of Studies Project

***Presented in order to obtain***

## National Bachelor's Degree in Computer Science

***Specialty :***

## Software Engineering and Information Systems

***By***

*Saif Eddin Nairi*

# Development of a consultation process and passage of administrative Endorsement

***Presented on 22/06/2024 in front of the jury composed of :***

| | |
|---|---|
| **Mrs : Samah Hbaieb** | **President** |
| **Mr : Ridha Jarray** | **Member** |
| **Mr : Sami Bihiri** | **ISIMM Supervisor** |
| **Mr : Mejdi Maaloul** | **Professional Supervisor** |

*Année Universitaire : 2023 / 2024*

# Resume

**This work is developed as part of our end-of-studies project to obtain a bachelor's degree in computer science. It consists of designing and creating a digital platform for the management of consultations and administrative amendments to insurance contracts.**

**This platform, intended for insurance agents, offers functionalities allowing simplified and optimized management of contract consultations and administrative modifications.**

**Mots clés : Angular, Spring Boot, PostgreSQL, JWT, REST API.**

# Abstract

**This project is part of our final year endeavor to obtain a bachelor's degree in computer science. It involves designing and developing a digital pathway for the management of insurance contract consultations and administrative amendments.**

**The platform, intended for customers and insurance agents, provides functionalities that simplify and optimize the handling of contract consultations and administrative changes.**

# Acknowledgments

*As a preamble, we would like to express our thanks to those who provided us with their help and who contributed to the development of this report.*

*We would like to thank Mr.**Sami Bhiri**, our supervisor at **ISIMM**, for his support, for his encouragement and the help he gave us throughout. throughout our work.*

*We also thank the **members of the jury** for having accepted to examine and judge our work.*

*I also want to express my gratitude to my professional supervisor at **VERMEG** Mr.**Mejdi Maaloul**, who allowed me to benefit from his broad knowledge and valuable advice during this internship. I express to him all my feelings of gratitude and appreciation.*

*Finally, we want to express our gratitude to all our professors at **ISIMM** for the training they have carefully provided us throughout our university studies.*

# Contents

# List of Tables

# List of Figures

# *General Introduction*

Currently, the management of administrative endorsements in the insurance sector is often conducted using traditional, manual, and paper-based interactions among various stakeholders. This method poses several challenges, particularly in terms of efficiency, traceability, and time management, leading to delays and sometimes errors in the processing of contract amendments.

In this context, our end-of-studies project aims to develop an innovative digital solution to optimize and simplify the management of consultation pathways and the processing of administrative endorsements. This system will allow for smooth interactions between insurers and their clients, thus facilitating rapid and secure modifications to existing contracts.

Our report details the development of this system across four main chapters. The first chapter, Context and Objectives, presents the general framework of the project and defines the specific objectives and needs it seeks to fulfill. The second chapter, Analysis and Specification of Requirements, examines in detail the functional and non-functional requirements, establishing the foundation of what will be built. The third chapter, Design, describes the software architecture and technological choices, illustrated by relevant diagrams. Finally, the fourth chapter, Implementation, covers the development and implementation of the system, as well as its deployment and testing performed to ensure its functionality.

Through this project, we hope not only to effectively meet the current needs for managing administrative endorsements but also to pave the way for future improvements and extensions of the system.

# Chapter 1

# *Context of the project*

## 1.1 Introduction

In this first chapter, we will start with the general context of the project and the emerging idea. Next, we will go over certain existing solutions so we can further refine the work to be done. We end this chapter by exploring different development methodologies in order to choose the one that is best suited for our needs.

## 1.2 General context of the project

### 1.2.1 Framework of the project

This work is part of our graduation project for obtaining a diploma in Computer Science in Higher Institute of Informatics and Mathematics of Monastir. We conducted our internship at Vermeg over the course of three months.

### 1.2.2 Presentation of the host company

The story of **VERMEG** began in 2002 independently from parent company **BFI**, the latter is a specialist in software and digital solutions for the banking, capital markets and insurance sectors. In addition to its standard software solution offerings meeting evolving digital needs, VERMEG offers tailor-made solutions based on its own tools and professional expertise. The VERMEG network includes more than 550 clients which are management companies, private banks, insurance companies and curators present in 40 countries with more than 1600 employees.
VERMEG provides several innovative software intended for the finance and insurance sector.
Over the years, VERMEG has grown through internal growth as well as strategic acquisitions:

- **In 2011** :VERMEG acquired the regulatory information business of Sofgen, in particular the main part known on the American regulatory market as IDOM USA.

- **In 2014** :VERMEG acquired the Belgian firm BSB specializing in financial software for the insurance and private banking sectors.

- **In 2018** :VERMEG acquired Lombard Risk, an international provider of software solutions for risk management, regulatory information and collateral management.[1]


Figure 1.1: VERMEG

### 1.2.3 Activities and products

- Vermeg's activities are primarily structured around four key areas:
  - Banking
  - Capital markets
  - Insurance
  - Digital transformation
- Vermeg offers a rich range of products, including:
  - Solife: A leading insurance contract management system designed by and for insurers. It is a comprehensive solution for managing life insurance, savings, retirement, and provident contracts, both individual and collective.
  - Megara: A modular service range that covers post-market processing. It relies on cutting-edge technology and an innovative three-view architecture: the operations view, the client view, and the market view. This approach facilitates the integration of new markets and meets client requirements without impacting ongoing operations.
  - Soliam: A centralized solution that tracks client investments in real-time. It is software for managing multiple accounts and custodians, offering comprehensive management of front-office and middle-office processes for asset managers, private bankers, and family office managers.
  - Palmyra: A solution that enables the development of applications and digital architecture. Its main goal is to automatically generate an entire application from a simple UML model.
  - Agile Reporter: A strategic solution that addresses cost and risk reduction, efficiency optimization, integrated regulatory change management, direct process control, and regulatory information assurance worldwide.[2]

### 1.2.4 Vermeg's clients

Our clientele spans a wide range of companies operating in the finance sector, from small family office managers to large national and international enterprises.

- Financial institutions:
  - Banque de France
  - Bank of England
  - Caceis
  - Societe Generale
  - Commerzbank
- Insurance companies:
  - Cardif Lux Vie
  - MAIF
  - AXA
  - Milleis

### 1.2.5 Assignment service

To realize our project, we were welcomed into the "Solife R&D" department. The team in this department is dedicated to developing solutions for insurance companies. Specifically, this department is responsible for the Solife solution, a platform dedicated to the administration and management of life insurance policies.

## 1.3 Solife digital

This section presents the existing solution in Solife digital and its limitations.

### 1.3.1 Analysis Solife digital

In order to choose a reliable application that meets clients' needs, we first had to review the existing REST web services in Solife to understand their limitations.

Indeed, Solife offers its clients, aside from the Solife backend solution (developed entirely with the Java framework), a digital interface (partly based on REST web services and partly created using the Palmyra framework).

The existing Solife digital solution is based on REST APIs. It facilitates customer and broker interaction with Solife through a very simple and effective interface. This digital solution meets several needs However, to date, it does not cover all the functionalities available in the Solife backend.

### 1.3.2 limitations of the existing

The main limitation is the lack of a mid-office (insurance manager) interface. This type of user interface is reserved for internal core business collaborators. This deficiency can lead to several critical issues:.

- **Limited Functionality for Managers:** Without a tailored interface, managers may not have access to specialized tools and features that support their specific tasks, such as overseeing claims, managing policies, and monitoring customer interactions. This can hinder their ability to perform efficiently and effectively.
- **Data Accessibility Issues:** Managers often need to analyze large volumes of data to make informed decisions. Without a manager-specific interface that can integrate and present this data effectively, it can be challenging to gain insights, identify trends, or address issues proactively.

### 1.3.3 Proposed Solution

to improve an insurance website that doesn't have a specific interface for insurance managers, here are some solutions we can consider:

- **Create a Manager-Specific Dashboard:** Develop a customized dashboard that gives insurance managers a complete overview of important metrics. This dashboard should include information on active claims, new policies, customer feedback, and performance indicators.
- **Provide Enhanced Data Management Tools:** Give managers advanced tools for managing and analyzing data. These tools should make it easy for them to sort, filter, and analyze data effectively. Consider including dynamic search options, custom report builders, and automated alert systems for critical metrics.
- **Automate Workflow Tasks:** Streamline common management tasks by automating them. For example, the creation of insurance contracts with well-defined investment strategies. This will not only reduce the workload on managers but also minimize the chances of human errors.

By implementing these solutions, we could optimize the insurance website for managers, making their job easier and more efficient.

## 1.4 Preliminary study

in the following section , we will present the preliminary study that we carried out before starting our project. The objective of this study is to highlight certain essential theoretical concepts in order to facilitate understanding of the business aspect of our application.

### 1.4.1 Insurance Market

The insurance market represents a crucial sector of the global economy, providing financial protection against various risks and unforeseen events. It is important to identify certain technical terms used in the field of insurance.

#### 1.4.1.1 Key Insurance Concepts

- **Insurance Contract:** An agreement where the insurer promises to compensate the insured in case of a loss, according to the agreed terms. In return, the insured commits to pay a premium, either initially or regularly, to benefit from financial protection in case of damage.
- **Insurance Policy:** A printed contractual document establishing the relationship between the insured and the insurance company. It specifies the terms, coverage's, commitments, and mutual rights of both parties involved.
- **Insurer:** An entity, either an individual or a legal entity, committed to providing protection to its clients against life's uncertainties. Its role is limited to intervening upon the declaration of claims and assuming the resulting financial responsibility.
- **Insured:** The person primarily involved in the contract, who may not necessarily be the policyholder. The insured is required to specify the amount of capital to be insured from the outset.
- **Beneficiary:** The person designated by the policyholder at the time of subscribing to a life insurance contract, or at a later date, to receive the accumulated capital on the contract in case of the insured's death. To specify the beneficiaries' identity clearly, the insured completes what is called a "beneficiary clause."
- **Policyholder:** The person designated to subscribe to a life insurance contract on behalf of the insured. This person may be the insured themselves or another individual, such as a spouse or parent. The policyholder is responsible for providing the essential financial and personal information to assess the risks and determine the premium cost.
- **Endorsement Policy:** An endorsement in an insurance policy is an amendment or addition to the existing terms of an insurance contract. An endorsement may change various aspects of the original policy, such as adding or deleting coverage, or updating the policy to reflect current conditions or regulations. It can be issued during the policy term, at the time of purchase, or at renewal. In insurance terms, endorsements are used to tailor the insurance policy to fit specific needs without needing to issue a new policy.
- **Investment Strategy:** The variety of available financial instruments inevitably leads to a diversity of investment choices. For this reason, insurance has established strategies to follow. Each strategy includes specific financial instruments. The factors influencing the investment strategy are numerous, the main ones being age, income, and goals to be achieved.

#### 1.4.1.2 Investment Insurance and Traditional Insurance

Insurance constitutes an agreement between an individual or an entity (the insured) and an insurance company. The insurer commits to compensating for the losses or damages incurred by the insured in exchange for the payment of a premium. Examples of products for traditional insurance include:

- Automobile Insurance

- Home Insurance
- Life Insurance: It can be traditional if the insurer's role is solely to guarantee the payment of a lump sum or an annuity to a designated beneficiary in the event of the insured's death.

Investment insurance is a form of insurance that allows investors to place their money in various investment products such as mutual funds, stocks, or bonds, while also benefiting from protection against losses in value. Generally, investment insurance combines life insurance with an investment product. In cases of lower-than-expected returns or a decrease in the portfolio's value, the insurance offers a certain level of protection for the initial capital invested. It primarily aims to enable investors to enjoy the benefits of investment products while limiting the risk of capital loss.
Insurance contracts can be fixed capital contracts (these products offer life insurance protection and guarantee a fixed interest rate for a specified period) or variable capital contracts (investors typically have the option to choose from a selection of funds offered by the insurance company).

### 1.4.2 Financial Operations and Strategies of an Insurance Contract

#### 1.4.2.1 Financial Operations

In Solife digital, we are primarily interested in four major endorsements:

- **Single Premium Top-Up or Free Payment (Avenant de versement libre):** This is an option offered by certain life insurance contracts allowing policyholders to add a significant sum of money to their initial contract in a single payment. This additional sum is then invested according to the same terms as the initial contract, providing the opportunity for the policyholder to generate additional gains.
- **Switch (Avenant d'arbitrage):** This action involves the partial or total transfer of an investment from one unit-linked (UC) account to another UC account within a life insurance contract. This operation is generally carried out to readjust the insured's portfolio according to their investment objectives, market conditions, or risk tolerance.
- **Partial Withdrawal (Rachat partiel):** This refers to the withdrawal of a sum of money by the policyholder from the surrender value of their insurance contract.
- **Surrender (Rachat total):** This is a decision made by the investor to prematurely terminate their investment contract and recover its total value. This operation may be subject to surrender charges and can also have tax implications.

#### 1.4.2.2 Investment Strategies

Subscribing to a life insurance contract requires the establishment of an investment strategy aimed at enabling autonomous management of the client's portfolio. The availability of a wide range of financial instruments inevitably leads to a diversity of investment choices. Consequently, individuals who choose to invest their savings or financial resources must follow a structured process in a defined sequence. This choice of strategy

can be influenced by several factors, including age, income, and goals to be achieved. (Figure 1.2) presents the three types of strategies in Solife:


Figure 1.2: Investment Strategies

- **Free Strategy (Stratégie libre):** This is the preferred mode of management as it offers complete freedom in the choice of investment supports and arbitrages. It is recommended to become familiar with the workings of life insurance beforehand.
- **Profiled Strategy (Stratégie profilée):** The allocation of funds among cautious, balanced, and dynamic investment profiles is determined based on your profile and risk tolerance.
- **Managed Strategy (Stratégie pilotée):** The policyholder delegates the management of their life insurance contract to an advisor, who makes investment decisions based on the investor's profile. An initial discussion with the advisor is conducted to define the investor's requirements and return objectives.

## 1.5 Methodology adopted

In the following section, we will compare the classic and Unified Process methodologies. We will then focus on **2TUP** as the chosen methodology.

### 1.5.1 Classical methodologies

Classical methodologies, such as the V cycle model (Figure 1.5) are divided into phases that are identified from the beginning of the project. When a phase is validated, the next one begins. As a result, these methodologies limit the returns to the previous steps.

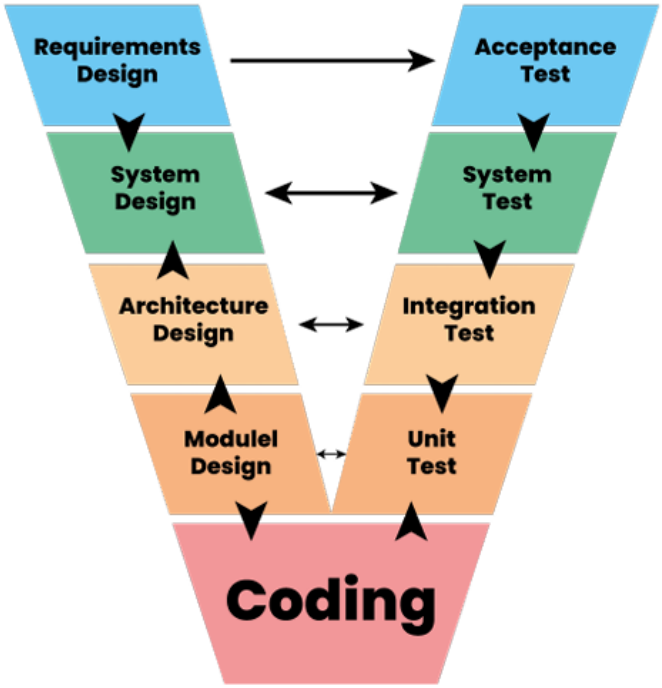



Figure 1.3: Representation of the V model methodology

### 1.5.2 Unified Process (UP)

Unified Process [3] is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation. The system is developed incrementally over time, iteration by iteration (Figure [**?**]), and thus this approach is also known as iterative and incremental software development. The iterations are spread over four phases where each phase consists of one or more iterations.

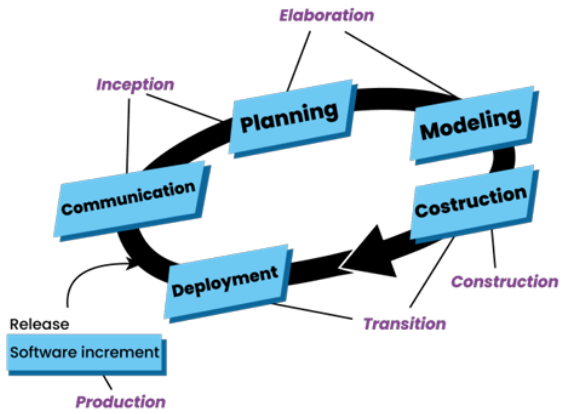



Figure 1.4: Representation of the UP methodology

### 1.5.3 Comparison of development methodologies

| | Advantages | Disadvantages |
|---|---|---|
| Scrum (Agile) | – Early functionality makes customer happy.<br>– Small teams are more efficient.<br>– Requirements flexibility allows for change. | – Hard to manage something that isn't well defined.<br>– A lot of time is spent in reviews.<br>– The requirements change a lot. So time is spent on redesign.<br>– Difficult for complex projects. |
| XP (Agile) | – Code is simple and allows for improvement.<br>– Constant testing makes XP more agile.<br>– Error avoidance through pair programming. | – Relatively large time investment.<br>– A lot of time is spent in reviews.<br>– The requirements change a lot. So time is spent on redesign.<br>– Difficult for complex projects. |
| RUP (Unified Process) | – Delivering exactly what the customer wants.<br>– The development time required is less due to reuse of components.<br>– Use case driven. | – The development process is too complex and disorganized.<br>– The team members need to be expert in their field. |
| 2TUP (Unified Process) | • Roles are well defined.<br>• Make a room for risk management.<br>• Complete process and well assisted by tools. | – Less efficient in small projects.<br>– Validation runs late which presents a risk of higher costs in case errors were detected. |

Table 1.1: Comparison of development methodologies

### 1.5.4 Adopted methodology: 2TUP

Most projects that use traditional approaches encounter delays and budget overruns because they attempt to forecast how things should progress according to a predetermined schedule. For these reasons, we have opted to use the unified 2TUP methodology.
The 2TUP proposes a development cycle that separates technical from functional aspects and suggests a parallel study of the two functional branches (application) and the technique (implementation).
This process is structured around three branches and places great emphasis on technology and risk management. The application presents at the technical level a certain complexity given the need to provide a phase for the study of the technical requirements that must be taken into consideration. The objective of our project is to develop a web application based on Angular, Spring Boot, and Oracle. New technologies that we have not yet mastered.
Furthermore, our project consists in defining the functional needs of the application that is to be developed, which is why we have oriented ourselves towards the use of the 2TUP methodology .

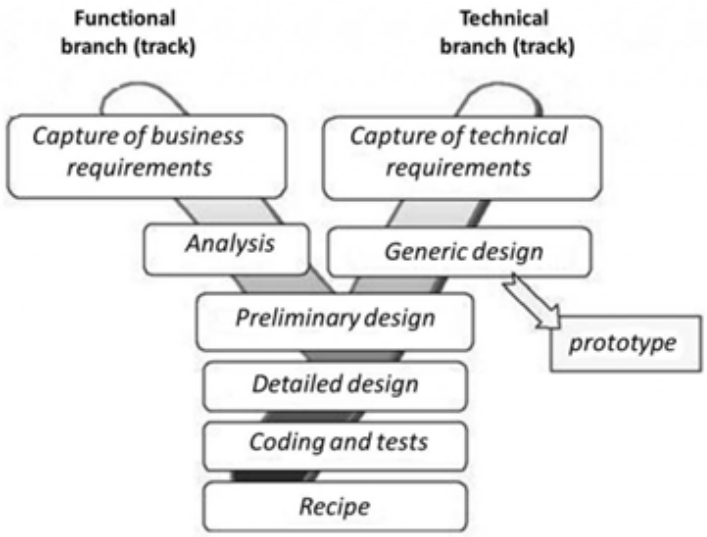



Figure 1.5: The development stages of 2TUP

## 1.6 Conclusion

We presented our project's framework in this chapter. We investigated some existing solutions in order to fine-tune the subject of our project, identify the functionalities that must be implemented and we have clarified the general concepts necessary for understanding our project. Finally, we presented the development methodology that we have adopted.

# Chapter 2

# *Requirement Analysis and Specification*

## 2.1 Introduction

A good specification is a key to successful design. In fact, a detailed description of the needs facilitates the development of the project and the communication between clients and developers. The specification provides a translation of user needs into conceptual and technical documentation useful informal contexts. In this chapter, we will discuss the functional and non-functional requirements for this application, then we will present the semi-formal specification, and finally, we will talk about the technical constraints

## 2.2 Functional requirements

### 2.2.1 Identification of Actors

- **Insurance Manager :**
  This actor is responsible for overseeing the insurance-related operations within the system . This actor interacts with the system to ensure that all insurance transactions are conducted accurately and efficiently.

### 2.2.2 Functional Requirements

- **consult Policy** : The ability to consult created policies and their endorsements.
- **Create New Subscription** : The ability to create insurance contracts (policy) with multi-pocket products.
- **Login** : insurance agent enter their username and password to access their manager space.
- **manage Endorsement** : The system must allow insurance agents to create, modify, and apply endorsements to existing policies.
- **Change Account Password** : the system allow user to change password of the account.
- **Switch Language** : the user can change language to English or french .

## 2.3 Semi-formal specification

### 2.3.1 General Use Case diagram

A use case diagram is a graphical description of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has. Figure 2.1 presents the general use case diagram of the dashboard. A detailed description of each use case can be found in the section 2.1 .

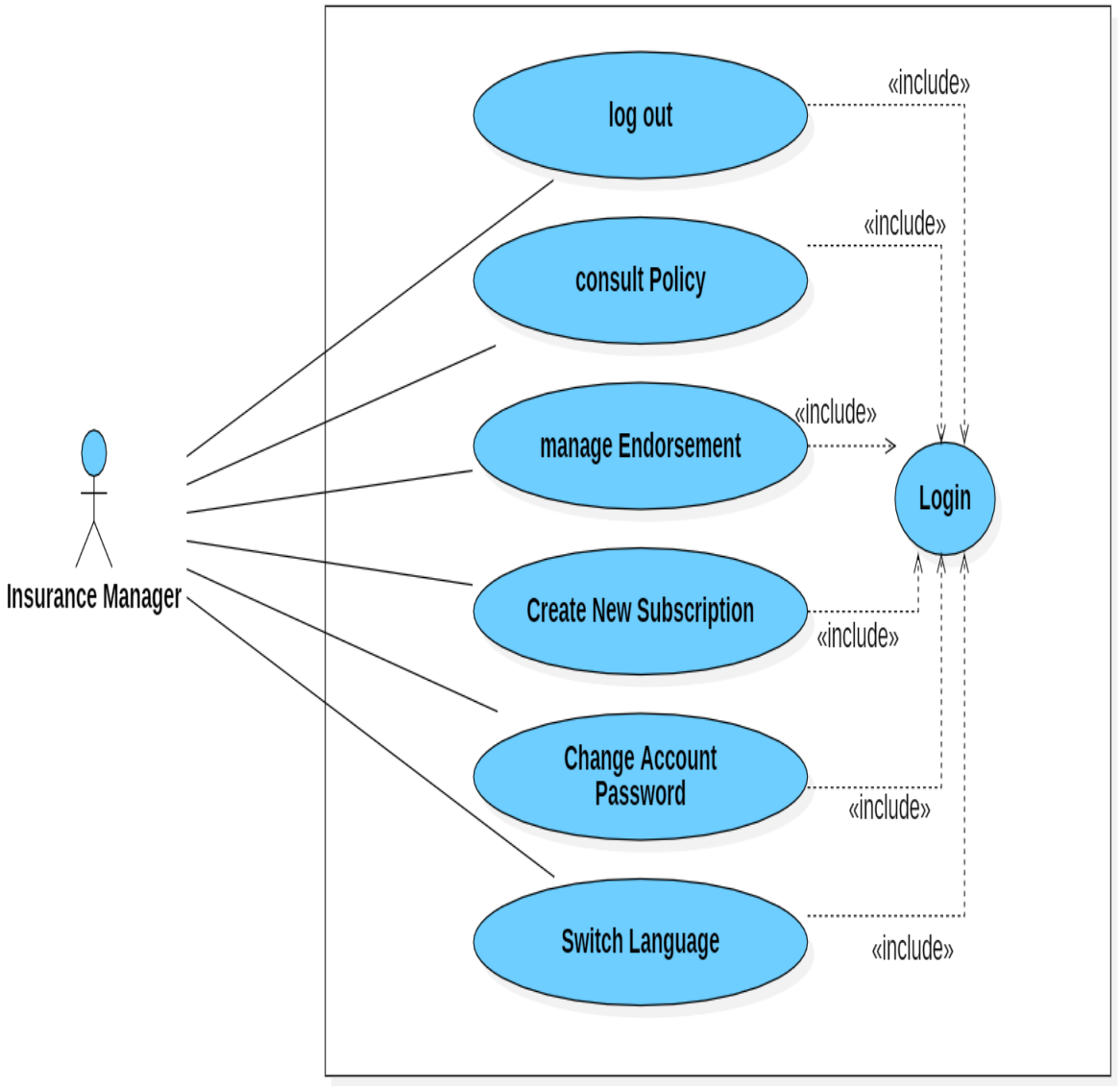



Figure 2.1: General Use-case diagram.

### 2.3.2 Detailed Use Case diagrams

**Use case «Consult policy ».**
This use case details Access and review the details and status of an existing insurance policy.The figure 2.2 graphically describes this use case

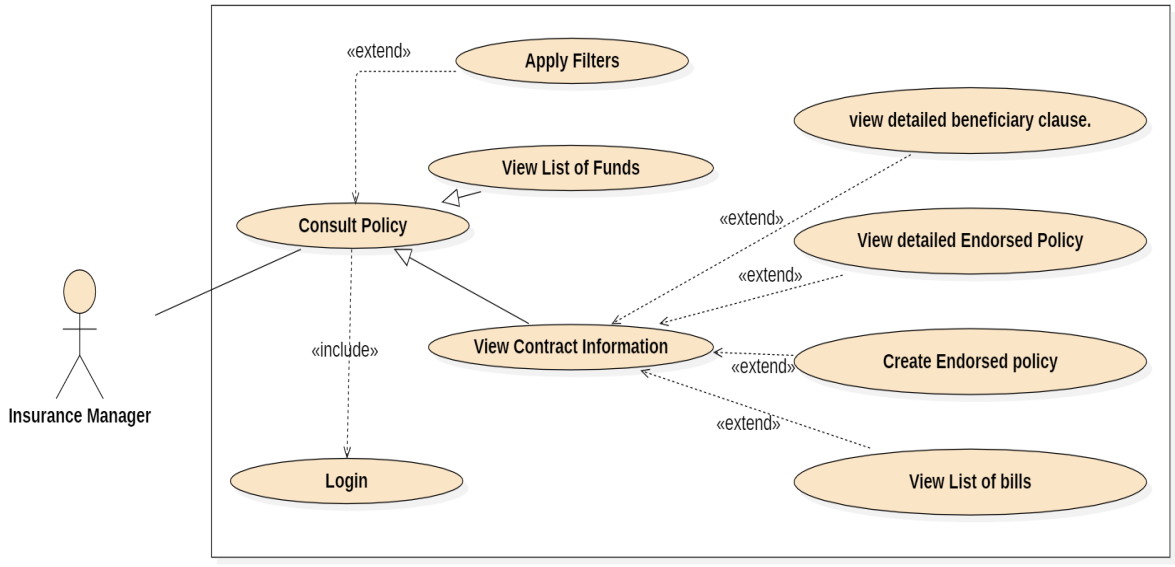



Figure 2.2: Consult Policy use case diagram

**Sequence diagram «Consult policy ».**
In the following, Figure 2.3 presents the sequence diagram for the analysis "Consult policy" accompanied by a structured textual description.

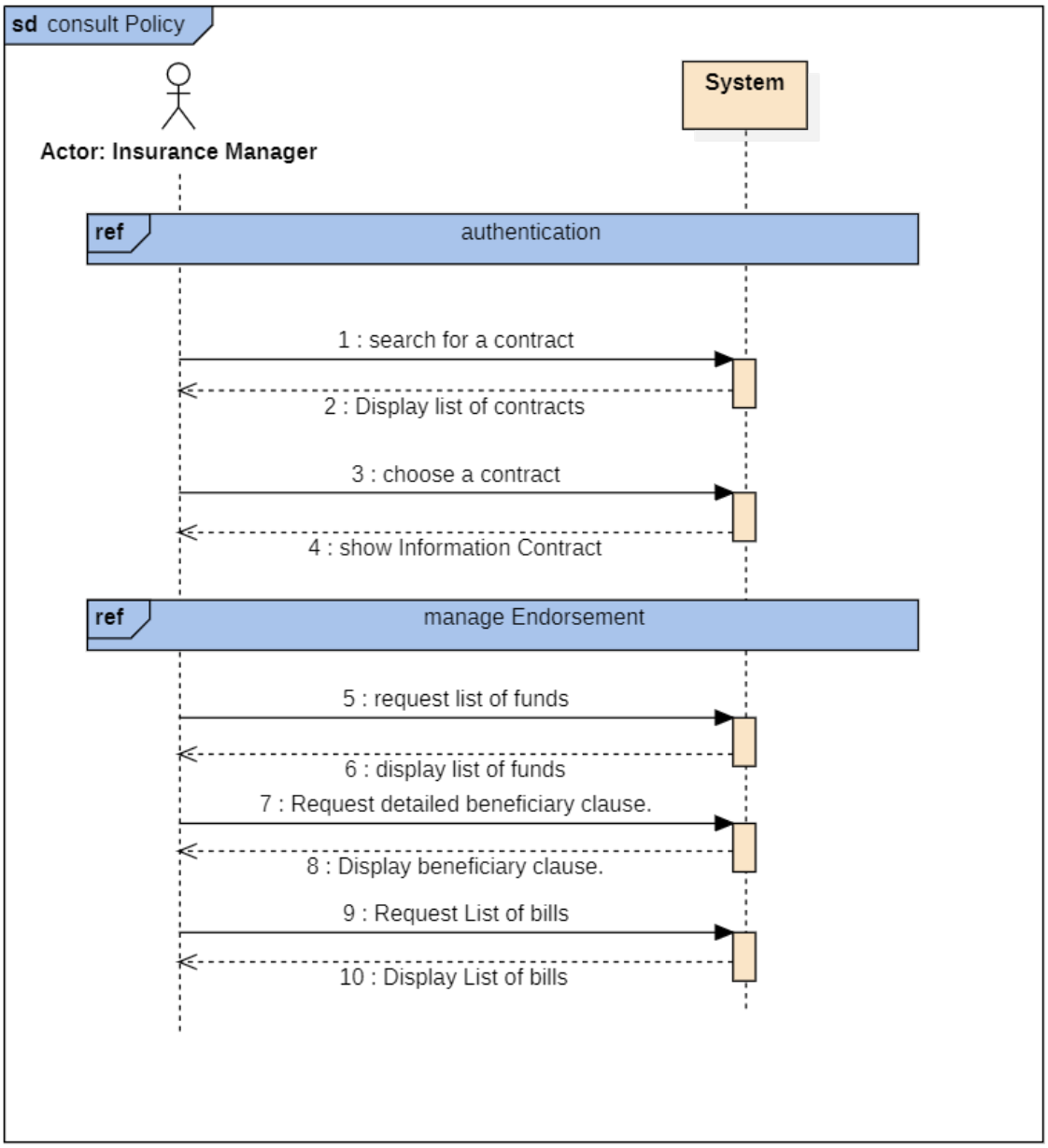



Figure 2.3: Consult Policy sequence diagram.

| **Title** | Consult an Insurance Policy |
|---|---|
| **Actors** | Insurance Manager |
| **Summary** | The actor inputs the policy identifier along with the optional date of consultation. |
| **Pre-Condition** | – Actor is authenticated.<br>– The consultation operation is selected. |

| | |
|---|---|
| **Post-Condition** | – The contract information is displayed. |
| **Main Scenario** | 1. The actor enters the data related to the Contract.<br>2. The actor validates the operation.<br>3. The system verifies the input.<br>4. The system displays a status of 200 OK with the contract information.<br>5. The actor requests detailed beneficiary clause.<br>6. The system displays a status of 200 OK with Beneficiary Clause.<br>7. The actor requests detailed List of bills.<br>8. The system displays a status of 200 OK with Beneficiary Clause.<br>9. The actor Manage Endorsement.<br>10. The system displays a status of 200 OK with Manage Endorsement. |
| **Exception Scenarios** | a. Date of consultation before the creation date of the policy:<br>– A 400 status from the web service is displayed.<br>– The system displays an error message with the incorrect date.<br>b. Policy number not found in Solife:<br>– A 400 status from the web service is displayed.<br>– The system displays an error message with the incorrect number.<br>c. Date of consultation later than the current date:<br>– A 400 status from the web service is displayed.<br>– The system displays an error message indicating that the date is incorrect. |

Table 2.1: Description diagram «Consult policy »

**Use case «Manage Endorsement».**
This use case details managing Endorsements in insurance contracts. Figure 2.4 graphically describes this use case

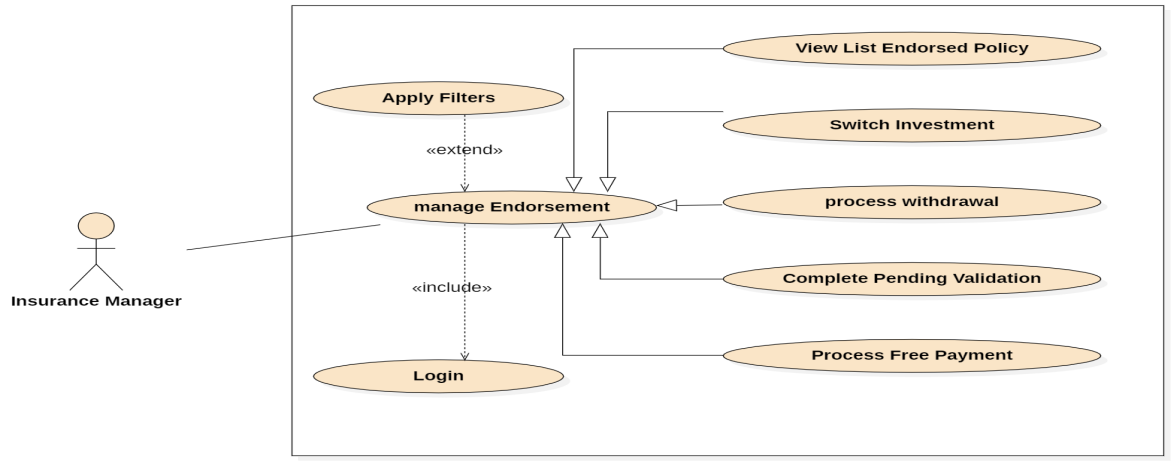



Figure 2.4: Manage Endorsement use case diagram

**Sequence diagram «Manage Endorsement».**
In the following, Figure 2.5 presents the sequence diagram for the analysis "Manage Endorsement" accompanied by a structured textual description.

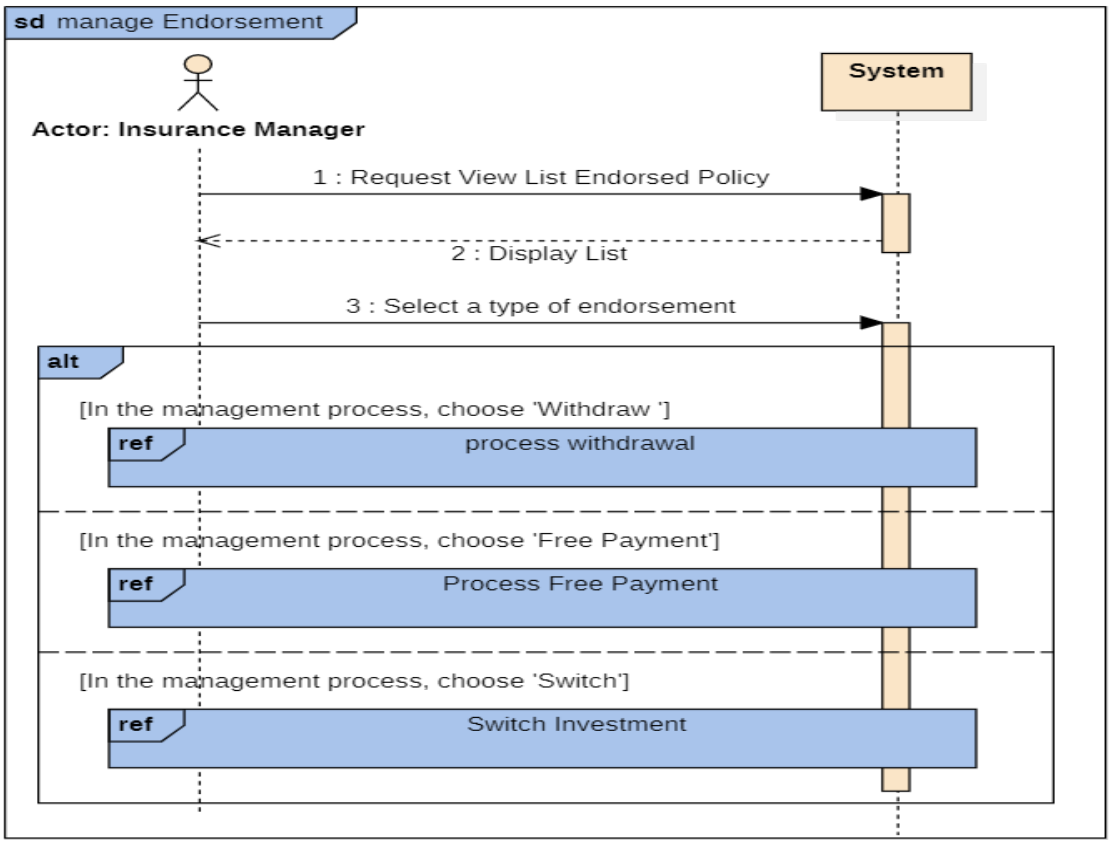



Figure 2.5: Manage Endorsement sequence diagram.

| **Title** | Manage Endorsement |
|---|---|
| **Actors** | Insurance Manager |
| **Summary** | The actor enters information relating to the desired financial transaction |
| **Pre-Condition** | – Actor is authenticated.<br>– Selected an Insurance contract<br>– Selected Creation Operation or view list Endorsement |
| **Post-Condition** | Endorsement completed. |
| **Main Scenario** | 1. The Actor requests to view the list of endorsed policies.<br>2. The system displays a status of 200 OK with the list of endorsed policies.<br>3. The Actor enters data relating to the type of endorsement.<br>4. The Actor validates the operation.<br>5. The system displays status 200 OK with the identifier of the endorsement proposal carried out. |
| **Exception Scenarios** | If the Actor doesn't fill in the required form, the system displays an error. |

Table 2.2: Description diagram «Manage Endorsement»

**Use case «Create New Subscription».**
This use case allows the actor to create a new insurance contract for clients. Figure 2.5 graphically describes this use case

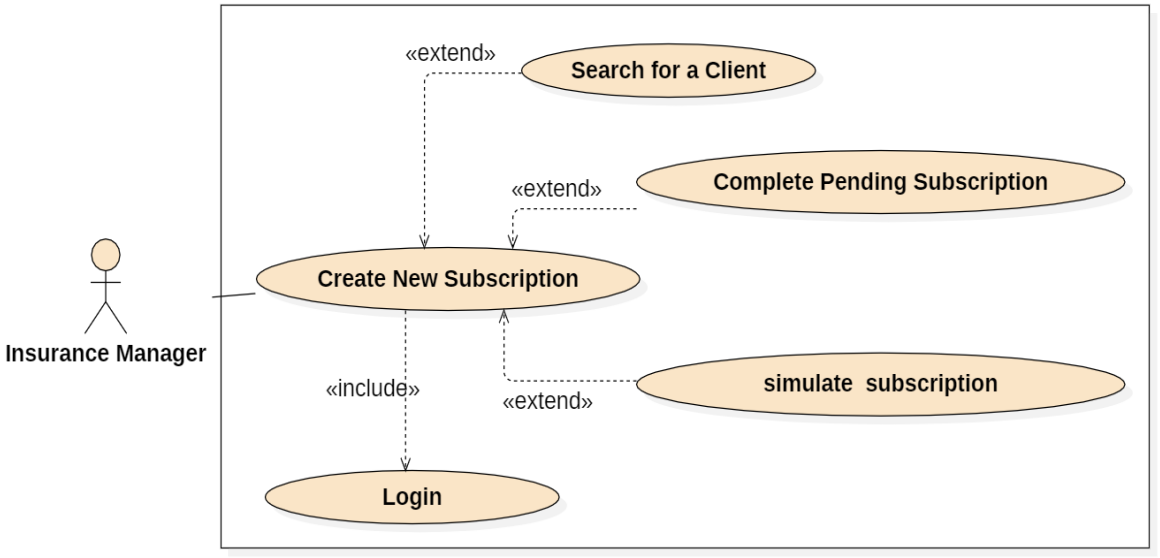



Figure 2.6: Create New Subscription use case diagram

**Sequence diagram «Create New Subscription».**
In the following, Figure 2.5 presents the sequence diagram for the analysis "Create New Subscription" accompanied by a structured textual description.

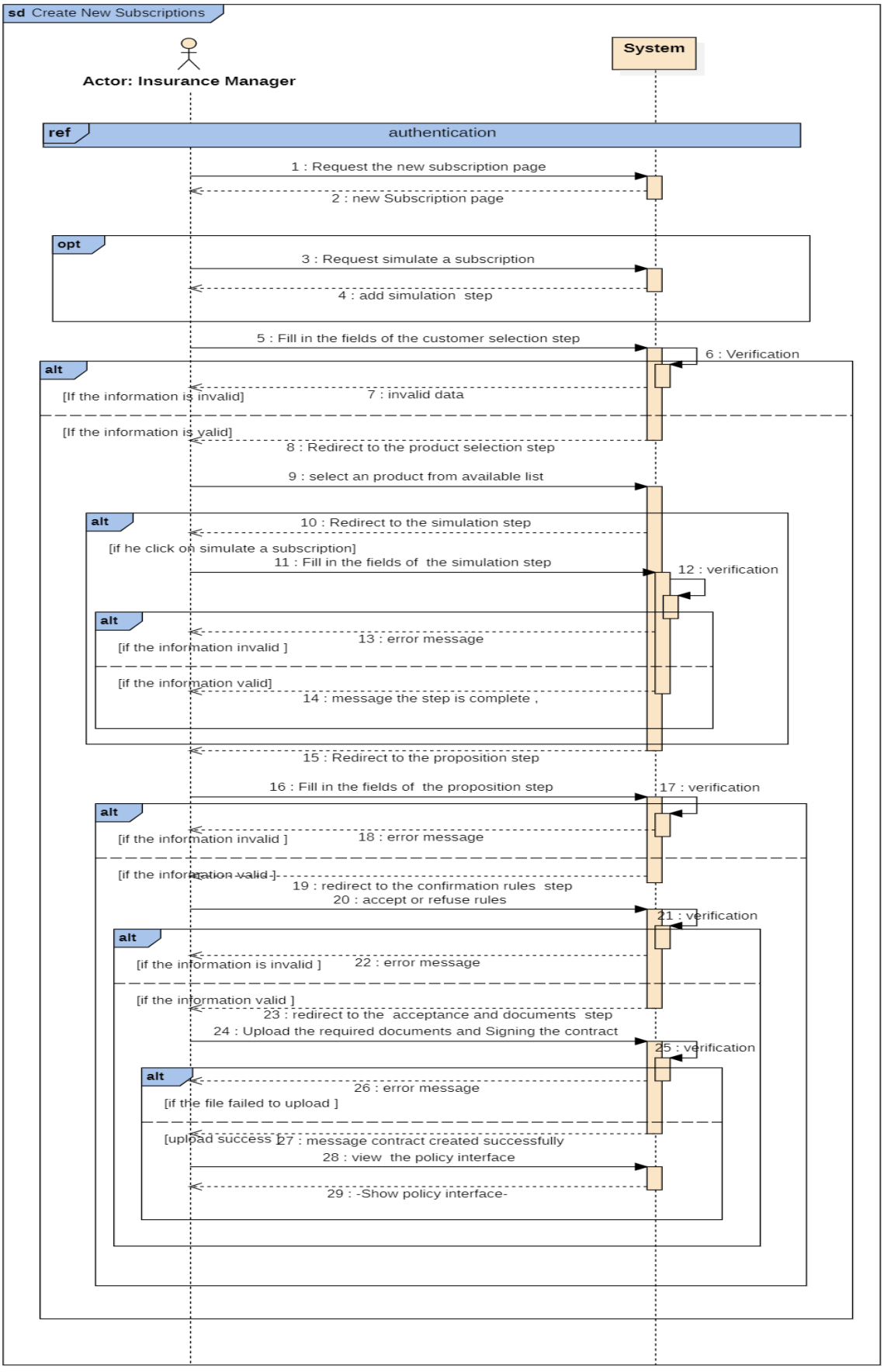



Figure 2.7: Create New Subscription sequence diagram.

## 2.4 Non-Functional requirements

- **Security:**
  - The system must ensure the confidentiality of beneficiary and contract information.
  - Implementation of security mechanisms to protect sensitive data.
- **Integration with the Solife backend.**
- **Performance:**
  - The management of beneficiary clauses must be performed efficiently, even with a large number of contracts and beneficiaries.
- **Scalability:**
  - The system should be designed to allow the addition of new functionalities or the modification of existing features without disrupting overall operations.
- **Reliability:**
  - The system must be reliable, minimizing the risk of data loss or service interruption.
- **User Interface Usability:**
  - The user interface should be user-friendly and intuitive to facilitate navigation and use by end users.
- **Documentation:**
  - Comprehensive documentation should be provided to facilitate understanding of the system, its integration, and its maintenance.
- **Regulatory Compliance:**
  - The system must comply with current regulations regarding data protection and the management of life insurance contracts.
- **Availability:**
  - The system must be reliably available, minimizing unplanned downtime.

## 2.5 Detailed Security Requirements

Since this application will handle very critical and sensitive information, the best security practices should be followed to protect against unauthorized access and minimize the threat of an attack.

### 2.5.1 Passwords

Data breaches are common, and even the largest companies often are affected by them. A password data breach can be devastating to application users, especially if they use the same password on more than one site. To minimize the damage from a potential data breach, some steps must be taken when storing passwords. **Password hashing**

In cryptography, a hash function is a mathematical algorithm that maps data of any size to a bit string of a fixed size. We can refer to the function input as a message or simply as input. The fixed-size string function output is known as the hash or the message digest. As stated by OWASP, hash functions used in cryptography have the following key properties:

- It's easy and practical to compute the hash, but *difficult or impossible to re-generate the original input if only the hash value is known.*
- It's difficult to create an initial input that would match a specific desired output. Thus, in contrast to encryption, hashing is a one-way mechanism. The data that is hashed cannot be practically restored.

This is perfect for storing passwords since we only need to verify that the password is valid when logging a user in, we can now hash the password and compare the hashes instead. Many hashing algorithms exist today for hashing passwords .

#### Minimum password length

an attacker could use a brute force attack to crack the password. This means we must limit the password length to prevent such an attack.

**Characters in password:** Characters [a-z]: ☑ Characters [A-Z]: ☑ Characters [0-9]: ☐ Characters [!@#$%^&*()+_]: ☐

**Password cracking speed:** 1 Billion per second **Processing elements:** 1

| | | Time to crack (max) |
|---|---|---|
| No of characters: | 52 | |
| No of passwords (5 digits): | 380,204,032 | 0.38 secs |
| No of passwords (6 digits): | 19,770,609,664 | 19.77 secs |
| No of passwords (7 digits): | 1,028,071,702,528 | 17.13 mins |
| No of passwords (8 digits): | 53,459,728,531,456 | 14.85 hours |
| No of passwords (9 digits): | 2,779,905,883,635,712 | 32.17 days |
| No of passwords (10 digits): | 144,555,105,949,057,024 | 4.58 years |
| No of passwords (11 digits): | 7,516,865,509,350,965,248 | 238.20 years |
| No of passwords (12 digits): | 390,877,006,486,250,192,896 | 12386.42 years |

Figure 2.8: Time to crack password based on number of characters.

The minimum password length shouldn't also be too long as it will discourage the user from memorizing the password and instead saving it in a file or writing it down. Taking everything into consideration we decide to use a minimum of 10 characters for passwords in our application.

### 2.5.2 Authentication

Authentication is the process of recognizing a user's identity. It is the mechanism of associating an incoming request with a set of identifying credentials. There are multiple ways to implement authentication, but in this application, we are using JWT.

#### JSON Web Tokens

**J**SON **W**eb **T**oken is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret with the HMAC algorithm.

#### Authentication with JWT?

When it comes to authentication, a JWT is a token that's issued by the server. The token features a JSON payload part that contains information specific to the user who is authenticated. This token may be employed by clients when requesting resources from an API so that the APIs can identify the user represented by the token, and take user-specific action.
JWTalso contains a signature. This signature is created by the server that issued the token and is used to verify the integrity of the token, which means another user cannot be authenticated just by tampering with the payload (unless they know the secret key used to generate the signature). Usually when the user logs in successfully a JWT is generated on the server and returned to the client, then the client would use that in the 'Authorization' header field to access private resources.

```
{
    Authorization: "Bearer <JWT>"
}
```

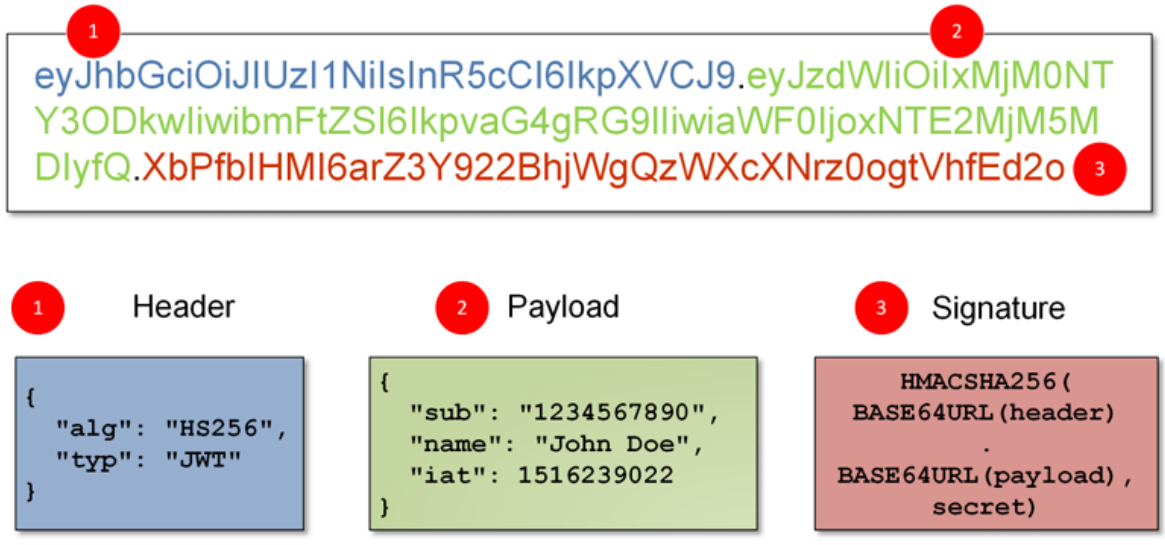



Figure 2.9: The format of a JWT.

### Why JWT and not server-side sessions?

With server-side sessions, the token must be saved on the server side, most likely in a database. Every time a user makes a request, a database query must be done to verify the session, which can become a bottleneck.

On the other hand, JWT is stored on the client side and does not require a database query with every request, which helps increase performance and response speed while reducing database size. JWT authentication enables scaling and microservices since it doesn't require a centralized database to store sessions.

### Challenges of using JWT authentication

JWT authentication like all protocols has its challenges, for example:

- Securing token transport between client and server.
- Securing token storage on the client side.
- Invalidating tokens.
- Dealing with stolen tokens.

### Securing JWT transportation

Sending tokens over an encrypted channel can solve the issue of securing the transportation. Therefore, all tokens must be sent via HTTPS.

### Securing JWT storage on the client side

Tokens on the client-side can be stored in application memory, with short expiration times to prevent token theft.

### Invalidating JWT & Dealing with stolen tokens

Unfortunately, JWT is not verified using a centralized database. This means that once a token is issued, it can only be invalidated when its time expires. However, we can address this issue by setting a short expiration time for the JWT and introducing deny-lists and refresh tokens.

### Refresh & Access JWT

To solve the issue of token invalidation, we can introduce a 2-token system:

The first token is the Access JWT, which is similar to the traditional JWT. It will have a short expiration time and will be stored in application memory. This token will be used to access resources on resource services.
We will introduce a second token, called a Refresh Token. This token will have a long

expiration time and will be stored in local storage or an HttpOnly cookie. This token will not be sent to resource services but will be sent to a special authentication service and will be used to refresh the access token. This authentication service will also allow us to invalidate refresh tokens.

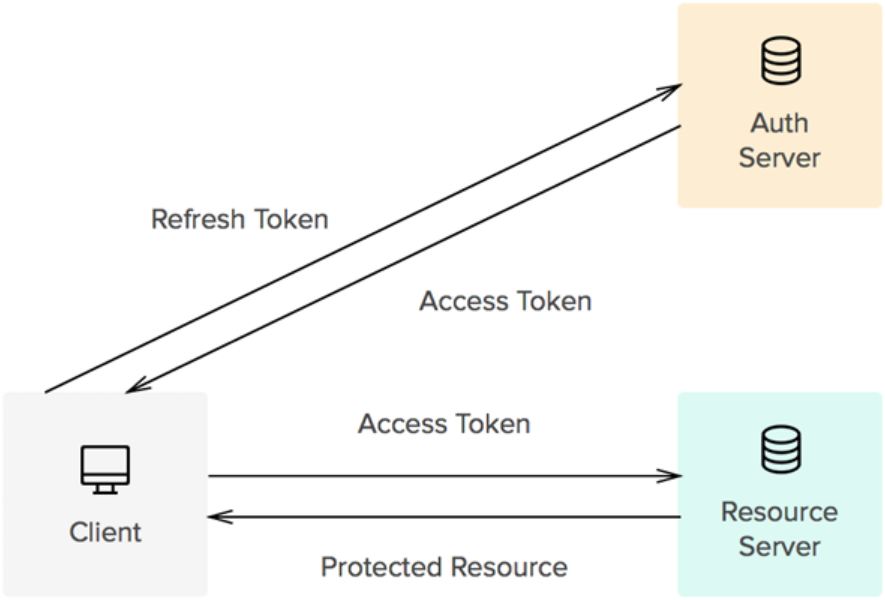



Figure 2.10: Access and refresh tokens.

### Keycloak

In addition to the native handling of JWTs, we employ Keycloak, an open-source Identity and Access Management solution, to further streamline and secure user authentication processes. Keycloak serves as a dedicated authentication server that integrates seamlessly with our JWT-based system. It provides a robust set of features including user federation, identity brokering, and social login. Keycloak simplifies the management of user identities by allowing for the centralized administration of roles, session management, and user permissions. With Keycloak, issuing, managing, and invalidating JWTs becomes more efficient, allowing for enhanced security measures such as multi-factor authentication and single sign-on (SSO). The server handles the generation and verification of JWTs, ensuring that all security tokens adhere to current security protocols and standards. This integration not only enhances the security architecture of our application but also improves the user experience by providing a smooth and secure authentication flow.

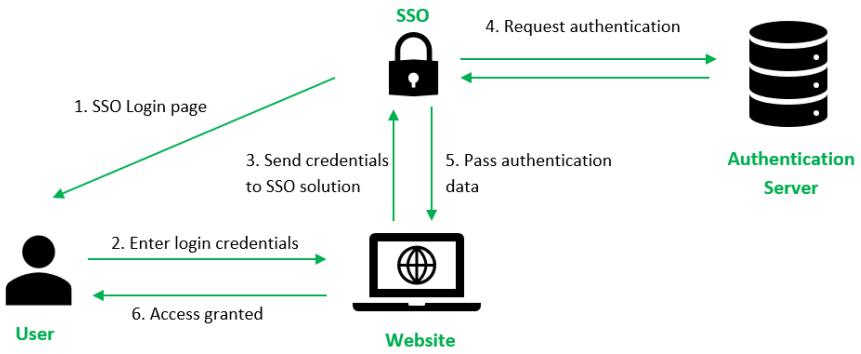



Figure 2.11: Single Sign On (SSO).

## 2.6 Technical requirements

| | |
|---|---|
| **Programming languages** | Java, Typescript (NodeJs) |
| **Frameworks and Packages** | Angular, Spring Boot |
| **DataBases** | Oracle Database, PostgreSQL |

Table 2.3: Technical constraints..

## 2.7 Conclusion

After studying the semi-formal specifications and specifying the different functional and non functional requirements of this application, we will start planning the conception of this project by talking about the architecture and exploring the static and dynamic views of this application.

# Chapter 3

# *Conception*

## 3.1 Introduction

If you fail to plan, you plan to fail. In this chapter, we will talk about the application architecture, then we will present the detailed conception of the dynamic and the static view of this application using a collection of diagrams.

## 3.2 Application Architecture:

After choosing the 2TUP methodology, the design approach will be in line with the architecture of the application.

### 3.2.1 3-tier architecture:

The 3-tier architecture [4] is a logical model of application architecture that aims to separate three software layers within the same application. The application is modeled as a stack of three layers whose role is clearly defined:

- **Data presentation:** Corresponding to the display, the workstation output, and communication with the client.
- **Business data processing:** Corresponding to the implementation of all management rules and application logic.
- **Access to persistent data:** Corresponding to data that is intended for permanent Storage.

### 3.2.2 MVC Architecture

MVC stands for Model-View-Controller, a software design pattern commonly used to implement user interfaces, data, and controlling logic. This pattern divides the application into three interconnected components:

- **Model:** Represents the data and business rules. It handles data processing, persistence, and defines methods for database access.
- **View:** Represents the user interface. It displays data retrieved from the model and forwards user actions to the controller.

- **Controller:** Analyzes and processes user requests, generating responses to HTTP requests from website visitors.

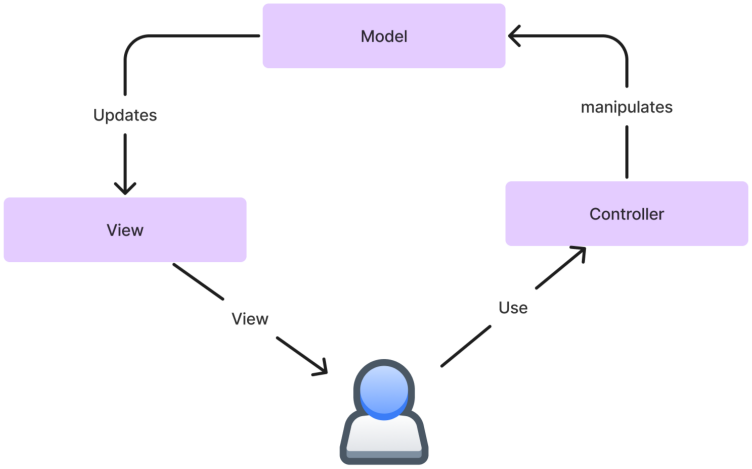



Figure 3.1: MVC Model.

### 3.2.3 Adopted architecture:

In our assignment, we've got followed the Model-View-Controller (MVC) structure, renowned for facilitating fast and parallel improvement. This architectural sample is mainly effective in segregating the person interface, commercial enterprise logic, and statistics model, which permits for independent improvement and maintenance of each factor. Here's how we have mapped our generation stack to the MVC structure:

- **Model:** The Model, answerable for coping with the facts, common sense, and policies of the utility, is carried out the usage of PostgreSQL. This database system shops and retrieves the statistics based on the interactions that come from the controller, ensuring information integrity and supplying important facts-associated operations.

- **View:** The View, which presents information to the person and sends consumer commands to the controller, is built with Angular. Angular gives a robust framework for developing dynamic and responsive person interfaces, making it ideal for dealing with the presentation layer in our software.

- **Controller:**The Controller, which accepts enter and converts it to instructions for the version or view, is treated by using Spring Boot. Spring Boot acts because the intermediary that manages the float of statistics between the PostgreSQL database (Model) and the Angular front-quit (View), and additionally carries commercial enterprise common sense and safety validation.

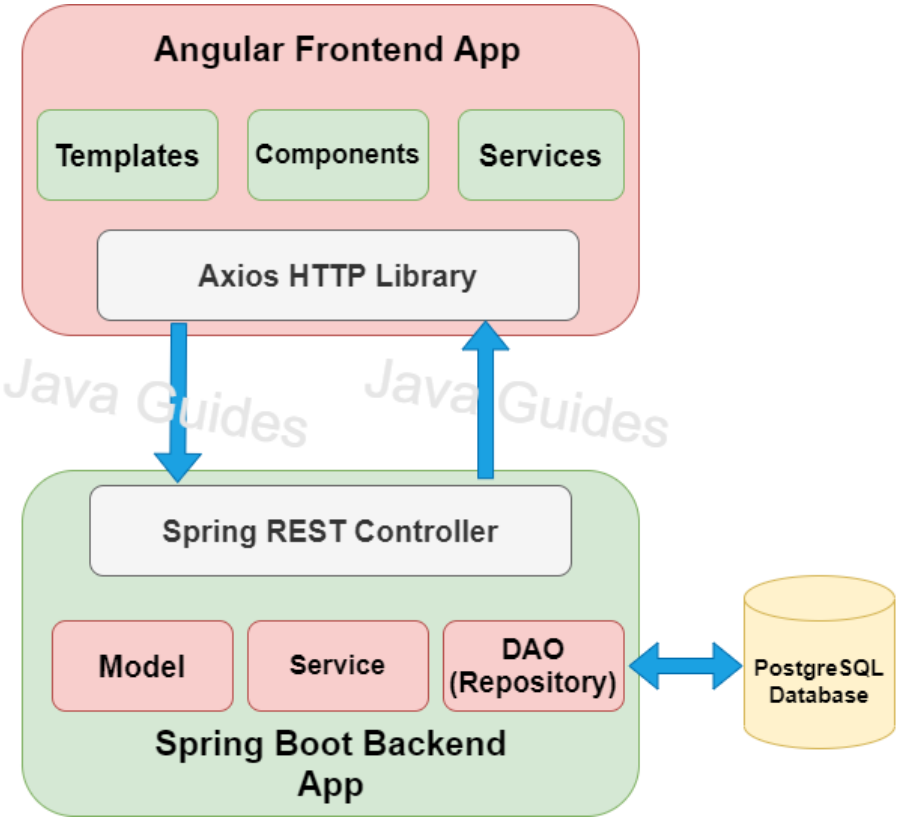



Figure 3.2: Architecture of Our System.

This structure now not best supports the green separation of concerns but also complements the scalability and maintainability of our platform with the aid of allowing every issue to be developed and tested independently.

## 3.3 Database Conception:

In the following, we will present the conceptual data model and the physical data model.

### 3.3.1 Conceptual Data Model (CDM) :

conceptual data model identifies the different business entities manipulated by our application, as well as the associations that link them.

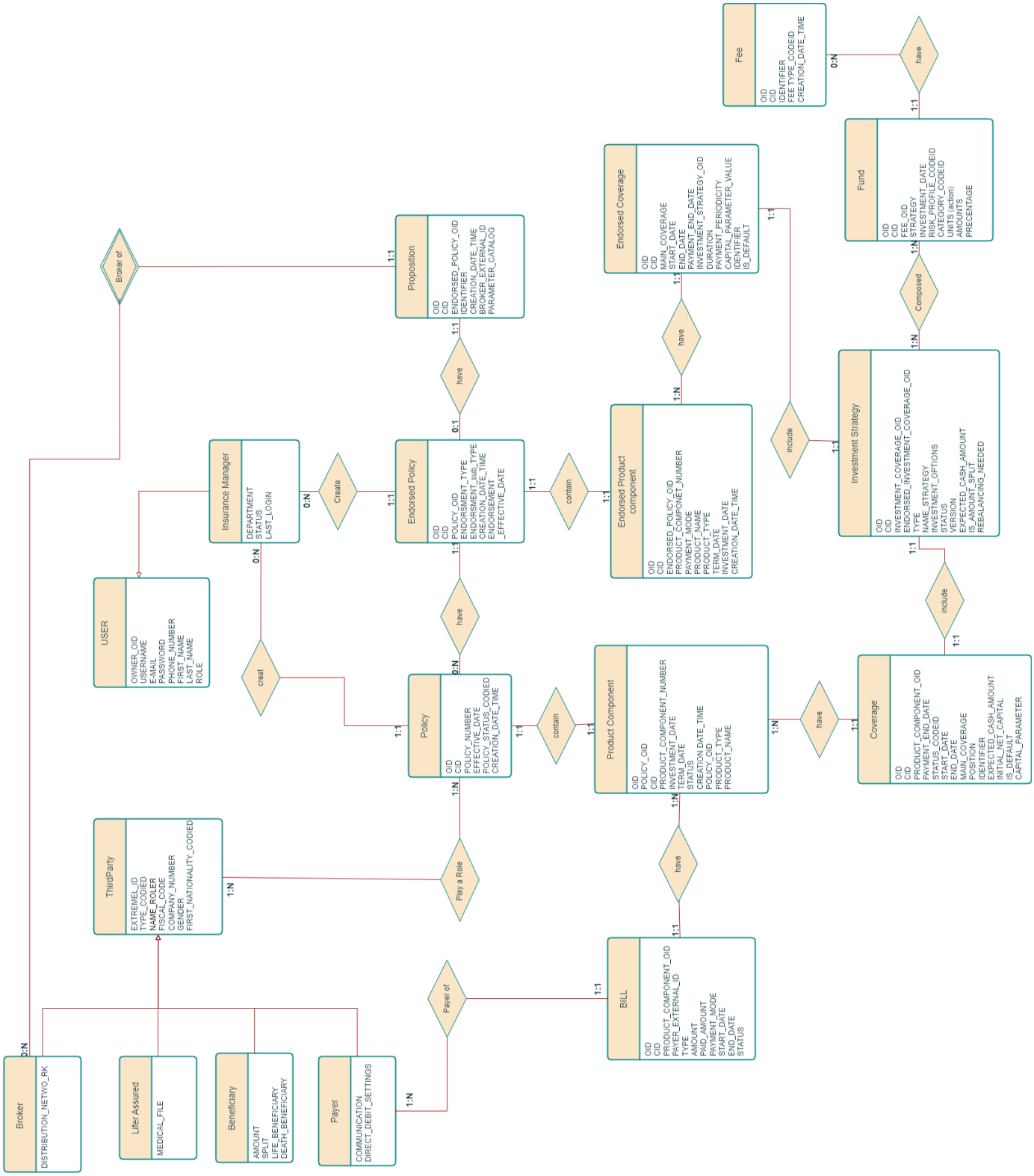



Figure 3.3: Conceptual Data Model (CDM).

| Collection | Description | Attributes |
|---|---|---|
| **User** | Models any platform user | • **OWNER_OID**: the user ID.<br>• **USERNAME, E-MAIL, PASSWORD**: login credentials.<br>• **PHONE_NUMBER, FIRST_NAME, LAST_NAME**: the user's contact details.<br>• **ROLE**: Specific role or title within the USER. |
| **Insurance Manager** | A specialized user role responsible for managing insurance policies and processing endorsements, crucial for maintaining policy integrity and updates. | • **STATUS**: Active, inactive, suspended.<br>• **LAST_LOGIN**: Timestamp of their last login.<br>• **DEPARTMENT**<br>Inherits attributes from User. |
| **Policy** | Represents an insurance contract between the insurer and the insured, detailing the terms, coverage, and conditions of the insurance agreement. | • **OID**:The ID of policy.<br>• **CID**: the class ID of policy.<br>• **POLICY_NUMBER**<br>• **EFFECTIVE_DATE**: date from which policy is effective.<br>• **POLICY_STATUS**: Inforce/Terminated/Cancel.<br>• **CREATION_DATE _TIME** |
| **ThirdParty** | Represents external entities like brokers, healthcare providers, or other companies that interact with the insurance system. | • **EXTREMEL_ID**: the ThirdParty ID.<br>• **TYPE_CODIED**<br>• **NAME**<br>• **FISCAL_CODE**<br>• **COMPANY_NUMBER**<br>• **GENDER**<br>• **FIRST_NATIONALITY** |

| Collection | Description | Attributes |
|---|---|---|
| **Endorsed Policy** | Details modifications or history of policy to existing policies, which are crucial for updating policy terms in response to changes in coverage. | • **OID**: the object ID of Endorsed Policy.<br>• **CID**: the class ID of Endorsed Policy.<br>• **ENDORSEMENT_TYPE**: its type of action like withdraw, free payment, manage investment.<br>• **ENDORSEMENT_sub _TYPE**<br>• **CREATION_DATE _TIME**<br>• **ENDORSEMENT_EFFECT IVE_DATE** |
| **Proposition** | Offers or proposals that are usually generated by brokers or agents to upsell or adjust existing policies to better fit customer needs. | • **OID**: the object ID of Proposition.<br>• **CID**: the class ID of Proposition.<br>• **IDENTIFIER**<br>• **CREATION_DATE _TIME**<br>• **BROKER_OID**<br>• **PARAMETER_CATALOG** |
| **Beneficiary** | Individuals or entities designated to receive the policy benefits upon the occurrence of the event insured against, such as the death of the Life Assured. | • **AMOUNT**: the benefit amount to be received.<br>• **SPLIT**: the percentage of benefit amount to be received<br>• **LIFE_BENEFICIARY**<br>• **DEATH_BENEFICIARY**<br>• **Inherits attributes from ThirdParty.** |
| **Broker** | Specialized third parties that facilitate the sale and management of insurance policies, often acting as intermediaries between insurers and clients. | • **DISTRIBUTION _NETWO_RK**<br>• **Inherits attributes from ThirdParty.** |

| Collection | Description | Attributes |
|---|---|---|
| **Payer** | Refers to the party responsible for paying premiums or fees associated with an insurance policy. | • **COMMUNICATION**<br>• **DIRECT_DEBIT _SETTINGS**<br>• **Inherits attributes from ThirdParty.** |
| **Life Assured** | The individual whose life or health is covered under the insurance policy, it is the primary subject of life insurance contracts. | • **MEDICAL_FILE**<br>• **Inherits attributes from ThirdParty.** |
| **Bills** | Handles billing and payment details for premiums and other charges related to insurance policies. | • **OID**<br>• **CID**<br>• **PAYMENT_MODE**<br>• **STATUS**<br>• **AMOUNT**<br>• **PAID_AMOUNT**<br>• **START_DATE**<br>• **END_DATE** |
| **Product Component** | Details specific components or features of a policy, such as various types of coverage or investment options associated with the policy. | • **OID**<br>• **POLICY_OID**<br>• **CID**<br>• **PRODUCT_COMPONENT _NUMBER**<br>• **INVESTMENT_DATE**<br>• **TERM_DATE**<br>• **STATUS**<br>• **CREATION_DATE_TIME**<br>• **PRODUCT_TYPE**: investment or traditional life.<br>• **PRODUCT_NAME**<br>• **PAYMENT_MODE**: The last configured payment mode which can be (cash, cheque, etc.). |

| Collection | Description | Attributes |
|---|---|---|
| **Coverage** | Generic entity that details the insurance protection provided under a policy, covering various risks associated with the policyholder. | • **OID**<br>• **CID**<br>• **PAYMENT_END_DATE**<br>• **STATUS_CODEID**<br>• **START_DATE**<br>• **END_DATE**<br>• **MAIN_COVERAGE**<br>• **POSITION:** the order of coverage on product<br>• **IDENTIFIER:** the name coverage like death accident<br>• **CAPITAL _PARAMETER_VALUE**<br>• **IS_DEFAULT:** boolean set true if coverage mandatory<br>• **INITIAL_NET_CAPITAL**<br>• **EXPECTED_CASH _AMOUNT** |
| **Investment Strategy** | Describes the methodologies and approaches used to manage the investments. | • **OID**<br>• **CID**<br>• **TYPE:** free strategy or Profiled Strategy or Managed Strategy<br>• **NAME_STRATEGY**<br>• **INVESTMENT_OPTIONS**<br>• **STATUS**<br>• **VERSION:**<br>• **EXPECTED_CASH _AMOUNT**<br>• **IS_AMOUNT_SPLIT**<br>• **REBALANCING_NEEDED:** boolean if set true then periodically balance the investment strategy. |
| **Fee** | Manages various types of fees associated with transactions within the insurance system. | • **OID**<br>• **CID**<br>• **IDENTIFIER:** name of fee<br>• **FEE_TYPE_CODEID**<br>• **CREATION_DATE _TIME** |

| Collection | Description | Attributes |
|---|---|---|
| **Fund** | Represents investment funds that are linked to specific investment strategies, providing a mechanism for achieving specified investment objectives. | • **OID**<br>• **CID**<br>• **RISK_PROFILE_CODEID**<br>• **CATEGORY_CODEID**<br>• **UNITS:** number of Shares<br>• **AMOUNTS**<br>• **PERCENTAGE** |
| **Endorsed Coverage** | Modifications to existing coverage terms that are documented to adjust the policy's terms or conditions. | • **OID**<br>• **CID**<br>• **MAIN_COVERAGE**<br>• **START_DATE**<br>• **END_DATE**<br>• **IDENTIFIER**<br>• **CAPITAL _PARAMETER_VALUE**<br>• **VERSION:**<br>• **PAYMENT_END _DATE**<br>• **PAYMENT_PERIODICITY**<br>• **IS_DEFAULT:**<br>• **REBALANCING_NEEDED:** boolean if set true then periodically balance the investment strategy. |

Table 3.1: Entities of the conceptual data model

### 3.3.2 Logical Data Model (LDM) :

Figure 3.3 below illustrates the relational model, which is a logical data model specifying a schema for a relational database. This schema includes tables, the fields of each table and their properties, the primary key of the tables, foreign keys ensuring the links between the tables, and the integrity constraints on these links (Relational Data Model).

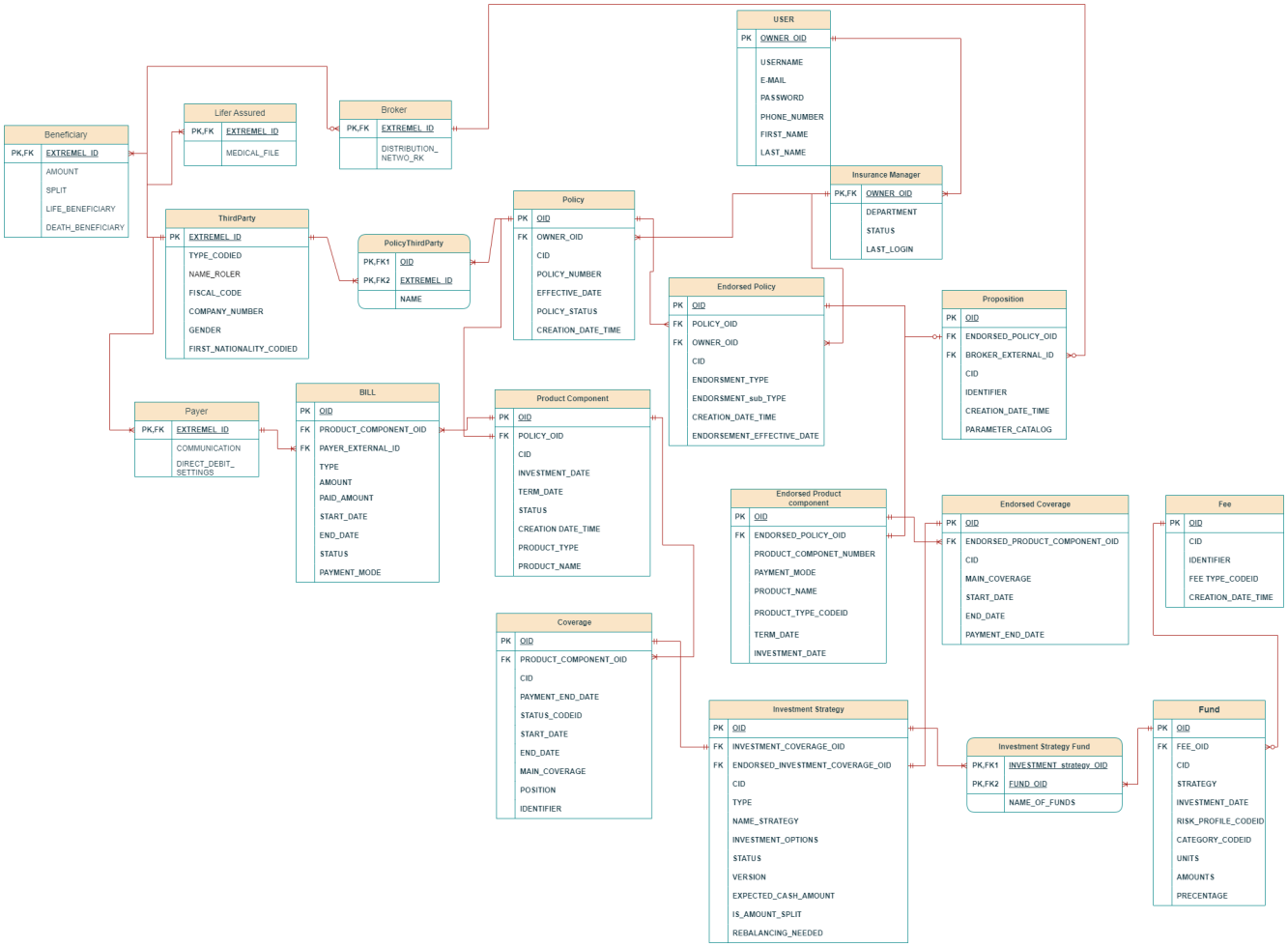



Figure 3.4: entity relational diagram.

In our case, we have two many-to-many relationships between the tables *Fund*, *Investment Strategy*, *Third Party*, and *Policy*, which result in the creation of two new tables: *PolicyThirdParty* and *InvestmentStrategyFund* .
Inheritance is then transformed into relationships:

- *Insurance Manager/User*
- *Broker/Third Party*
- *Life Assured/Third Party*
- *Beneficiary/Third Party*
- *Payer/Third Party*

This is done by adding the primary key of the parent classes into each of their child classes as both a foreign key and a primary key.

Finally, one-to-many relationships are also transformed into relations by adding the primary key of the table on the 'one' side into the table on the 'many' side as a foreign key.

## 3.4 Software Design

This part presents the software design of our application based on the UML. While respecting the constraints of our adopted architecture, we will build a static view as a class diagram and a dynamic view as a sequence diagram.

### 3.4.1 Static View: Class Diagram

The class diagram is one of the structural diagrams in UML. It is the most important diagram in object-oriented modeling. It is used in software engineering to present the classes and interfaces of systems as well as the various relationships between them. Figure illustrates the diagram.
Since we've adopted the MVC architecture, the explanation of the class diagram will be made in three parts.

***Part Model:***

This part consists primarily of the backend where data models and database interactions are defined using Spring Boot and PostgreSQL.

- «Model»Layer: These are the domain entities of our system representing the data that will be stored in or retrieved from the PostgreSQL database.
- «Repository»Layer: This layer handles the data access logic. In Spring Boot, this typically involves interfaces extending "JpaRepository" or "CrudRepository" for each entity , utilizing Spring Data JPA for ORM.

***Part Controller:***

This part deals with receiving HTTP requests and sending responses, using controllers in Spring Boot.

- «Controller»Layer: Controllers in Spring Boot act as the entry point for handling incoming HTTP requests. Each controller manages different aspects of the application.
- «Service»Layer: This is the intermediary layer that provides business logic and data manipulation. It typically interacts with the Repository layer to fetch, store, or update data.

***Part View:***

The view part is handled by Angular on the frontend, displaying data and interfacing with the user.

- «Component»Layer: Angular components are the building blocks of the UI, handling user interactions and data display. Examples include components like "Home", "Policy", "NewSouscription", which correspond to different parts of the application.
- «Service»Layer: Angular services are used to separate out common functionalities like data calls to the backend. Services that handle all HTTP communications with the backend APIs provided by Spring Boot.

Figure 3.5: Diagram class.

### 3.4.2 Dynamic view: Sequence diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

#### 3.4.2.1 Sequence diagram: Process Withdrawal

The figure3.6 Illustrates the Process Withdrawal sequence diagram.
The withdrawal process happens when the Insurance Manager creates an endorsed policy and chooses the withdrawal type.

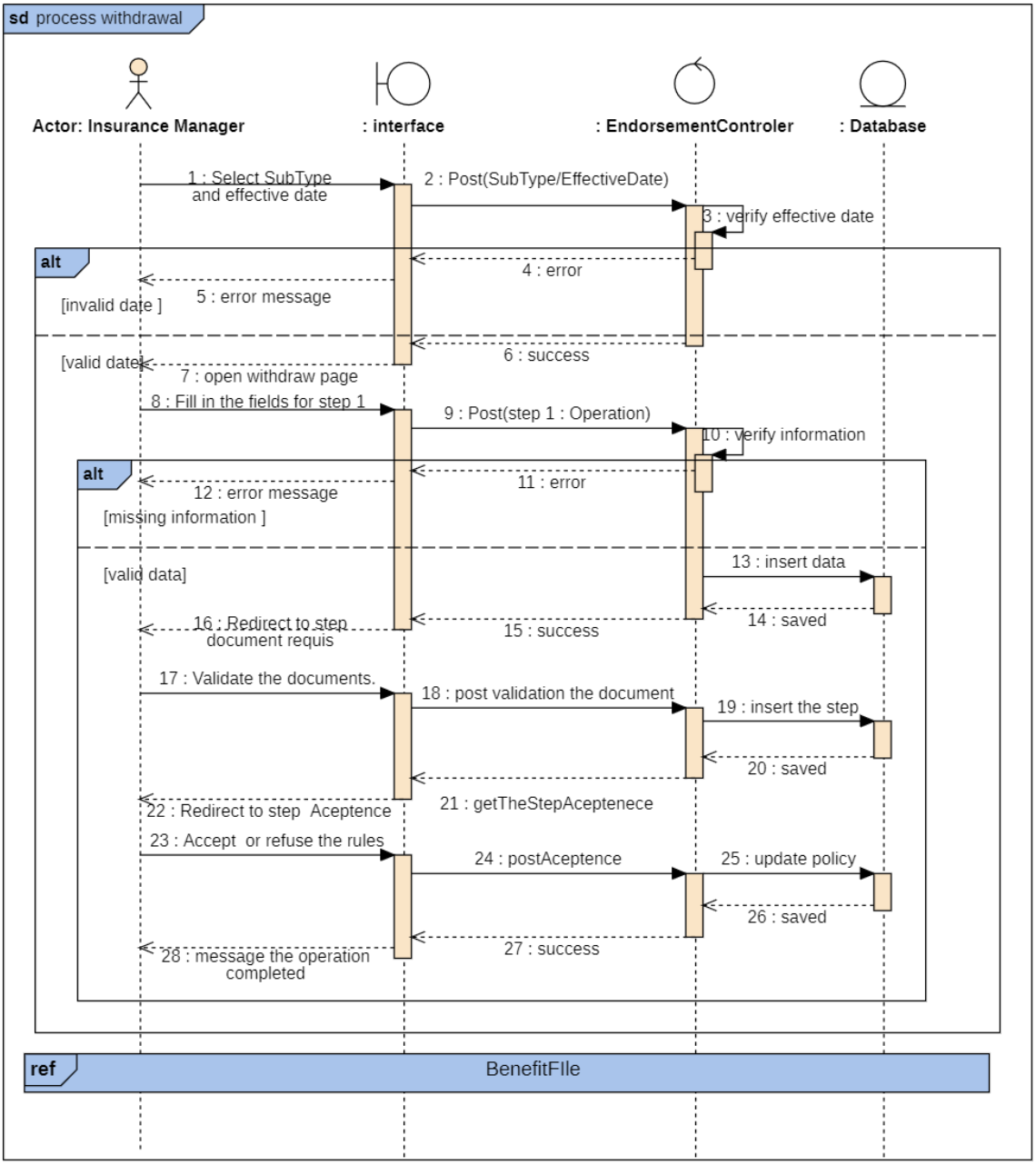



Figure 3.6: Sequence diagram: Process Withdrawal.

#### 3.4.2.2 Activity Diagram : Process Free Payment

The figure3.7 Illustrates the Switch sequence diagram.
The Free Payment process happens when the Insurance Manager creates an endorsed policy and chooses Free Payment type.

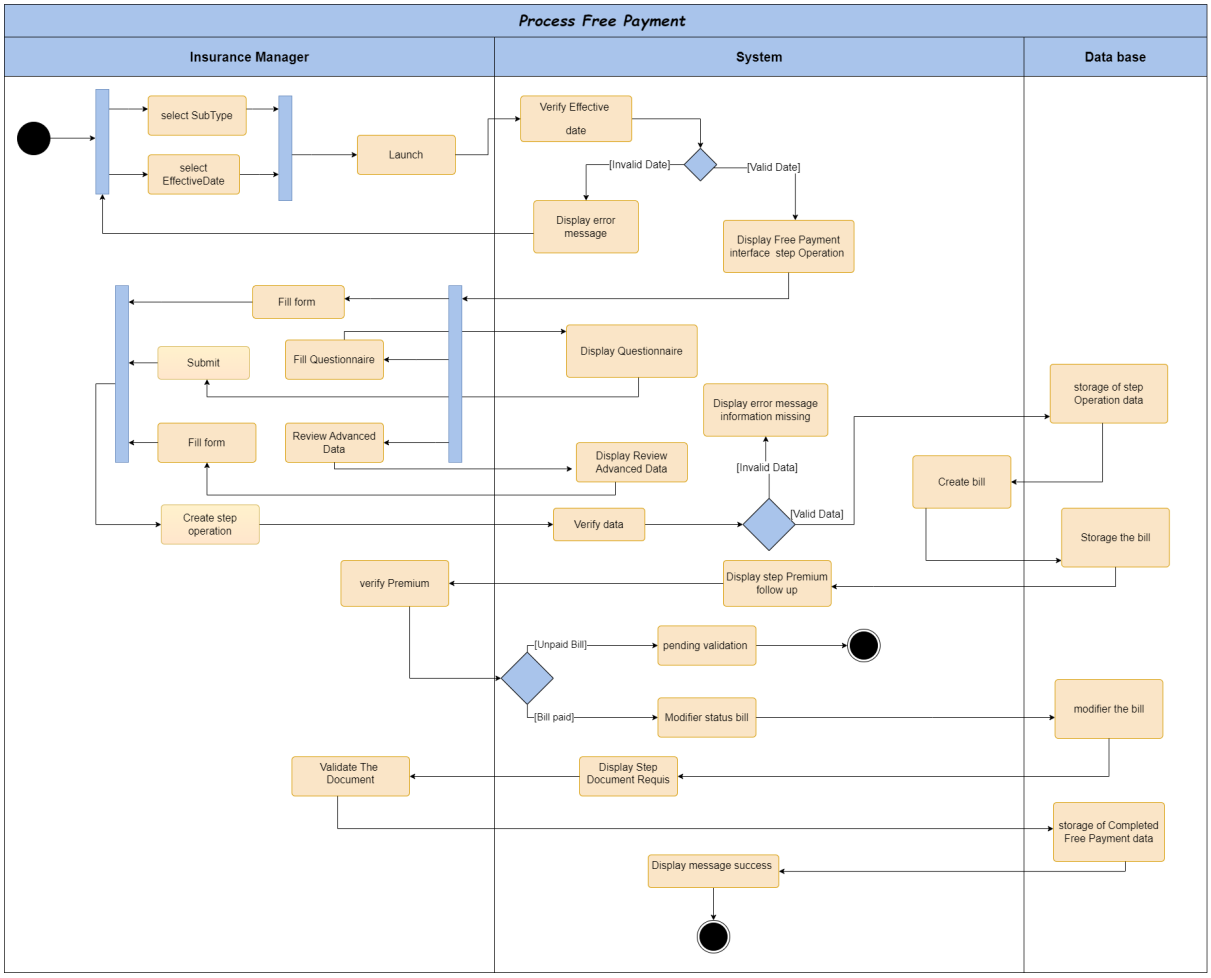



Figure 3.7: Activity Diagram : Process Free Payment

#### 3.4.2.3 Activity Diagram : Switch Investment

The figure3.8 Illustrates the SWitch sequence diagram.
The Switch process happens when the Insurance Manager creates an endorsed policy and chooses Switch type.

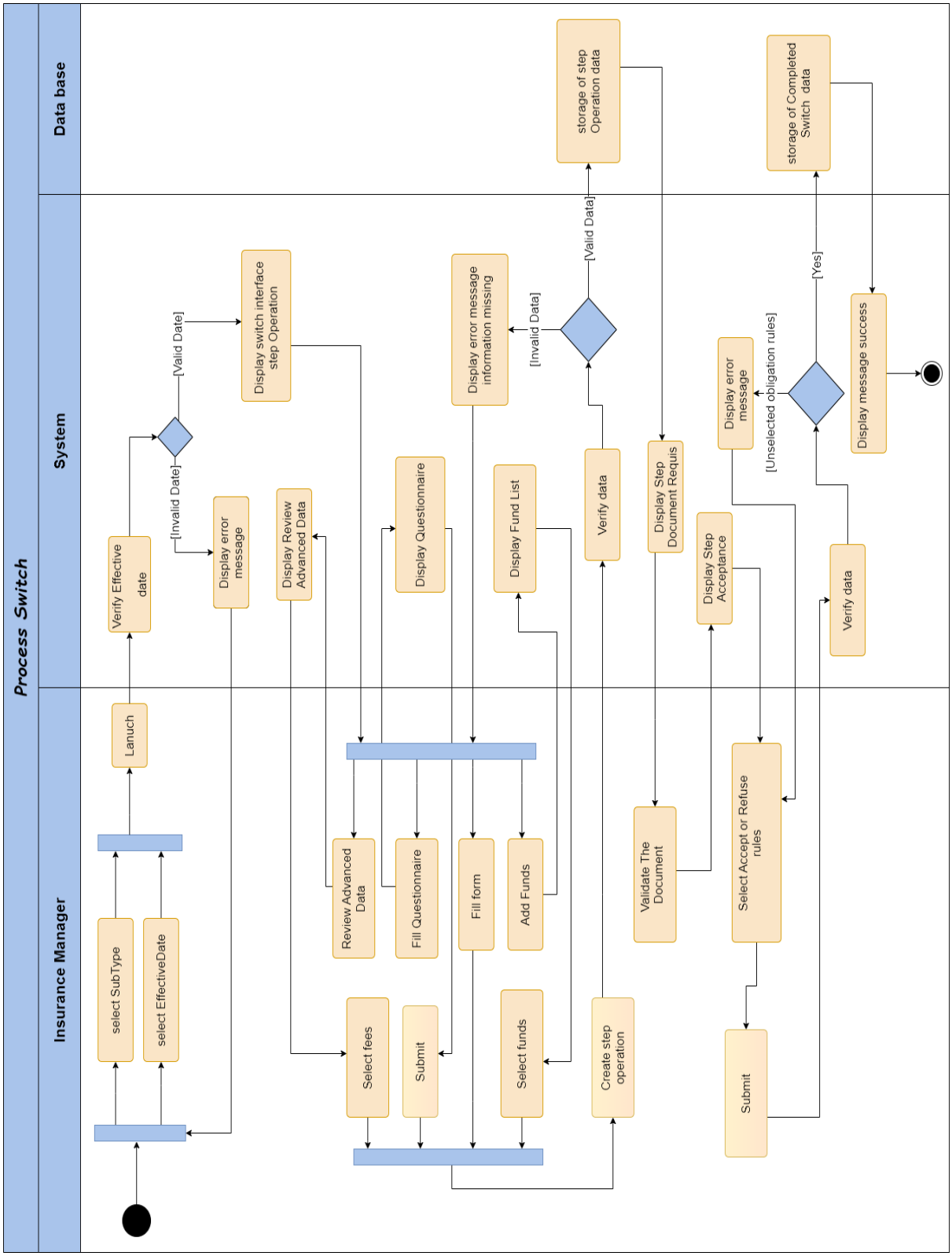



Figure 3.8: Activity Diagram : Switch Investment.

# Chapter 4

# *Realization*

## 4.1 Introduction

In this chapter, we will present the hardware and software development environment, the programming languages, the frameworks, the libraries, and tools used, and finally showcase the final application we developed.

## 4.2 Development environment

### 4.2.1 Hardware environment

For the development of this project, we used a machine provided by the host company with the following specifications:

| | |
|---|---|
| **Computer** | Lenovo |
| **Processeur** | i7-8665U |
| **RAM** | 16GB |
| **Hard disk** | 512GB SSD |
| **Operating system** | Windows 10 Enterprise |

Table 4.1: Hardware environment

### 4.2.2 software enviorment:

**IntelliJ IDEA**

-is a source-code editor developed by JetBrains for Windows, Linux, and macOS. It features intelligent code completion, debugging, refactoring, and support for various programming languages. Additional features include syntax highlighting, version control integration, and a wide range of plugins to extend its functionality.

-We used IntelliJ as the IDE for SpringBoot throughout the development phase.

**Visual Studio Code- Code Editor**

-Visual Studio Code is a source-code editor created by Microsoft for Windows, Linux, and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

-We used Visual Studio Code as the IDE for Angular throughout the development phase.

**Git**

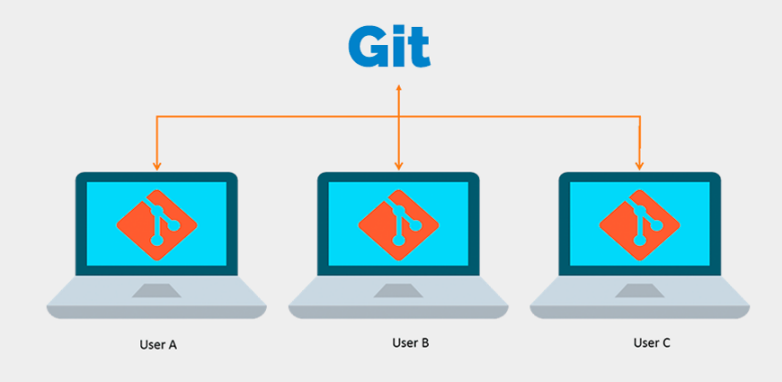



Figure 4.1: Git.

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).Since we needed to collaborate on this project, git was a must to use software for us.

**GitHub**

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.
GitHub was of great help for us, we especially used the private repository, the issues, and the task management features.

**Postman- API testing tool**

-While building a RESTful API, you need to be able to constantly test the new features and manually call the API, for this task we used Postman. Postman is the collaboration platform for API development, used by 15 million developers and 500,000 companies worldwide. Postman is an elegant, flexible tool for building connected software via APIs quickly, easily, and accurately.

-We used Postman to test our Rest APIs

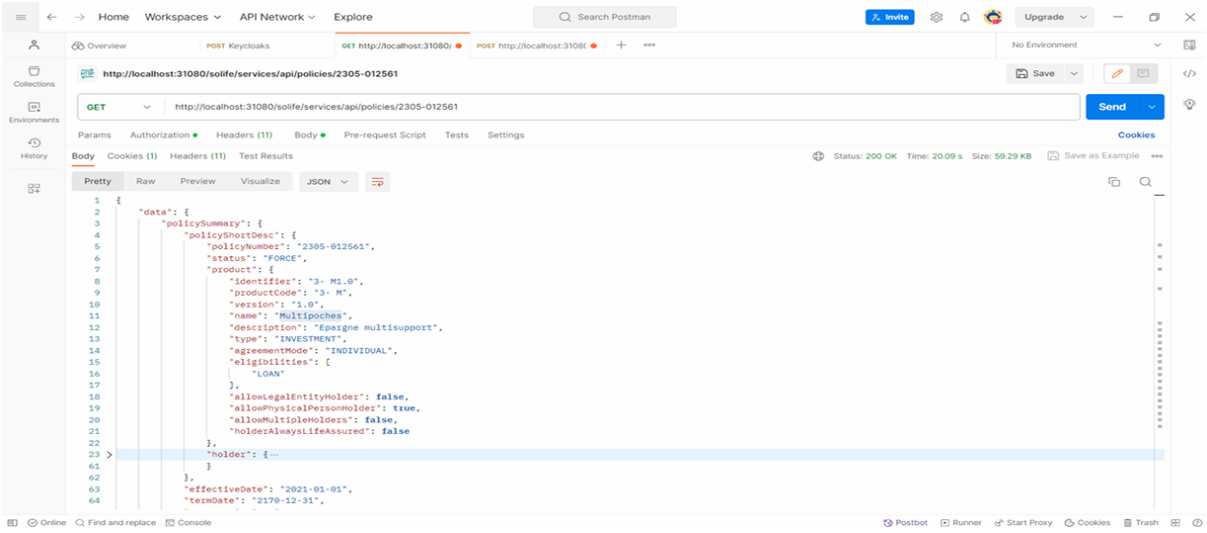



Figure 4.2: Test of the nominal scenario for consulting the policy.

**Overleaf LaTeX editor**

To write the project report, we needed a tool to help us collaborate and write LaTeX easily, we decided to use overleaf for this task.
Overleaf is a collaborative cloud-based LaTeX editor used for writing, editing and publishing scientific documents. It partners with a wide range of scientific publishers to provide official journal LaTeX templates, and direct submission links.

**StarUML**

-is a software modeling tool used for creating and managing UML (Unified Modeling Language) diagrams. It supports various UML diagrams, such as class diagrams, sequence diagrams, and use case diagrams. StarUML aims to support agile and concise modeling, and it is compatible with Windows, macOS, and Linux. Features include code generation, reverse engineering, and extensibility through plugins.

- Creation of UML diagrams such as use case diagrams and sequence diagrams.

**Draw.io**

-Draw.io is a free online application that allows you to draw diagrams or flowcharts.

-Create the entity/relationship diagram, relational schema, class diagram, and Activity diagram of our application.

**keycloak**

is to create the users who have access to our Solife API layer. figure 4.3 shows the different users created on Keycloak.

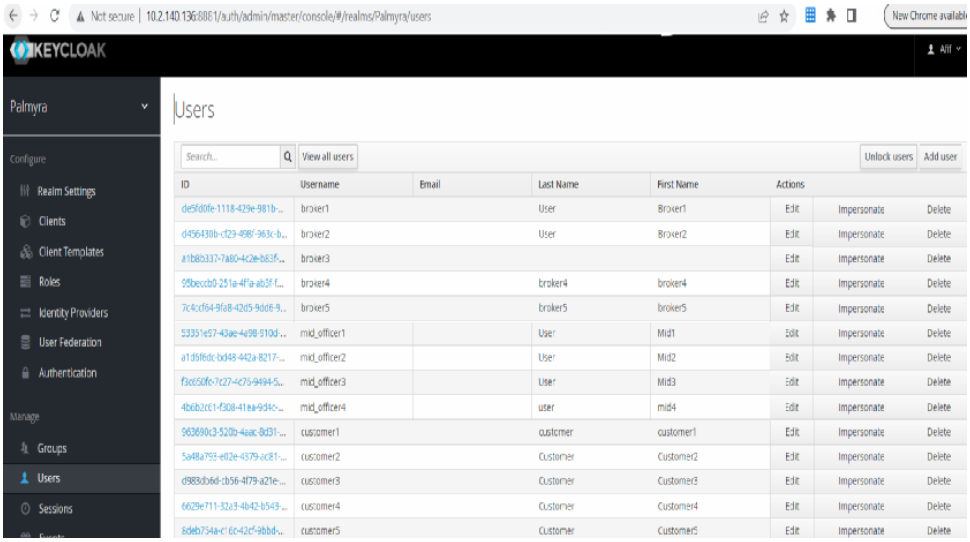



Figure 4.3: keycloak Solife.

### 4.2.3 Programming languages

**JavaScript**
JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. We have used JavaScript in the FrontEnd.

**HTML**
The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

**CSS**
Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

**Java**
Java is a high-level, class-based, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is designed to have as few implementation dependencies as possible, making it a key tool for building platform-independent applications. Java is widely used for developing mobile applications, web applications, enterprise software, and large systems. It features automatic memory management, a robust standard library, and strong security features.

### 4.2.4 Frameworks, Libraries, and run-time environments

#### Framework Spring

Spring is described as a framework for building Java applications. We can use Spring to create any type of Java application (e.g., standalone, web, or JEE applications), unlike many other frameworks (such as Apache Struts, which is limited to web applications).

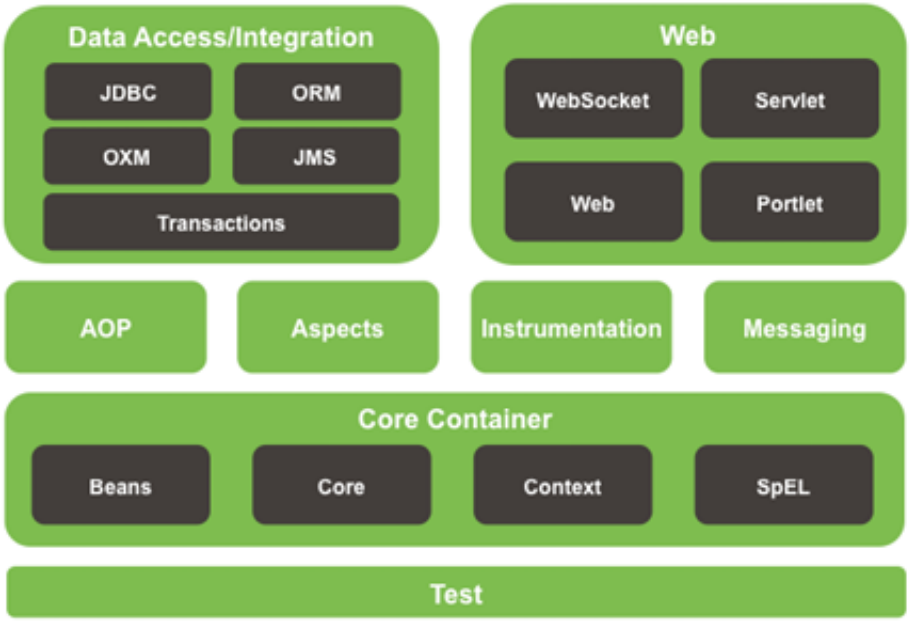



Figure 4.4: Spring Boot Architecture.

#### Angular

Angular is a TypeScript-based free and open-source single-page web application framework run on Node.js. It is developed by Google and by a community of individuals and corporations.

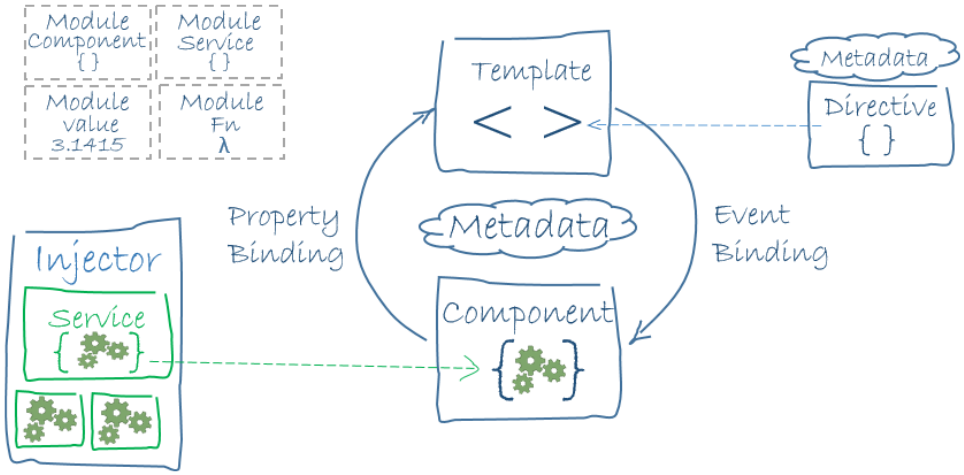



Figure 4.5: Angula Architecture.

**PostgreSQL**

PostgreSQL, often referred to as Postgres, is a powerful, open-source relational database management system (RDBMS). Known for its robustness, extensibility, and standards compliance, PostgreSQL supports a wide range of advanced data types and sophisticated querying capabilities.

### 4.2.5 source code confidentiality.

Rue du lac Neuchâtel
1053 les berges du lac
Tunis-Tunisie
Tél : (+216) 71 160 600
Fax : (+216) 71 963 878

www.vermeg.com

VERMEG

Tunis, le 10 Juin 2024

**Attestation de confidentialité de code source**

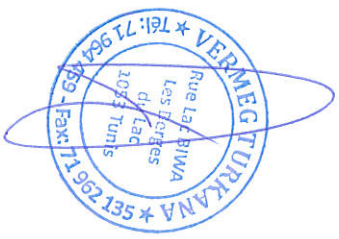
Monsieur le président de jury,

Dans le cadre de son stage PFE, **M Saif Eddin Nairi** a eu accès en lecture ou en écriture à des éléments des applications qui sont à la propriété de Vermeg et/ou de l'une de ses filiales, et/ou d'une société sœur et/ou d'un partenaire : codes source des programmes informatiques, documentations, documents de spécification, modèles informatiques UML, scripts de test, schémas informatiques, notes internes relatives aux applications etc…

Il s'engage à ne procéder en aucun cas à des copies, divulgations à des tiers ou diffusions de ces éléments par aucun moyen que ce soit : clés USB, disques externes, CDs/DVDs, réseaux sans fil, internet ou tout autre moyen).

L'ensemble de ces éléments doit être protégé par tout moyen pour préserver tous les droits de Vermeg.

Signature de L'encadreur



## 4.3 Application Interfaces

In the next part, we will give an overview of the website interfaces illustrating the different use cases already seen in the previous chapter.

### 4.3.1 Login page interface

Figure 4.6 showcase the login page for the application,

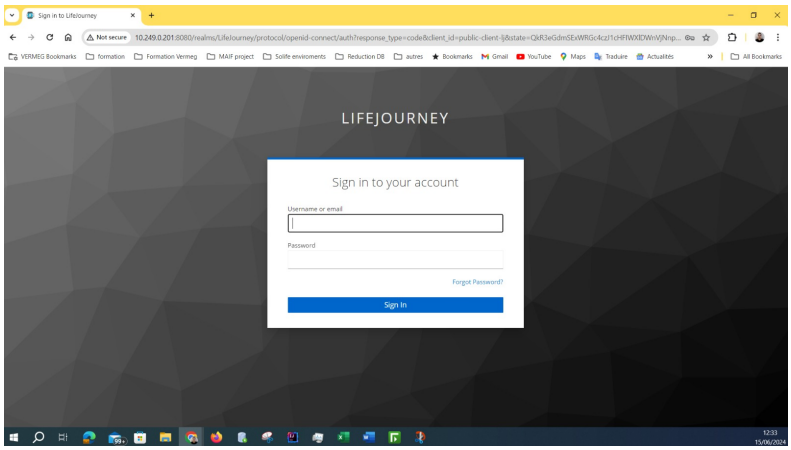



Figure 4.6: Login page interface.

### 4.3.2 Home page interface

When the login is successful, the user is directed to the home screen of the application. Figure 4.7

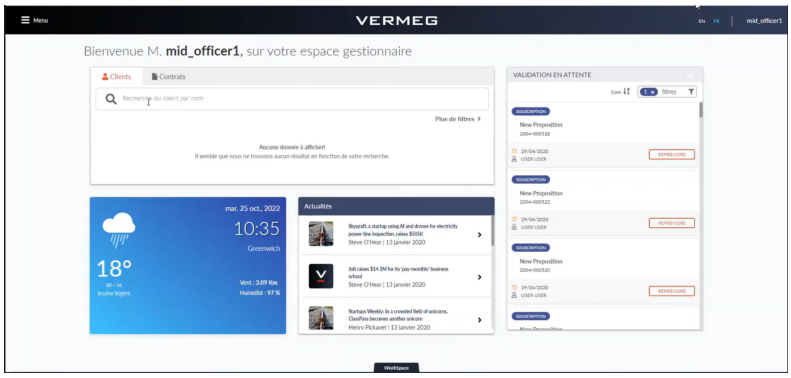



Figure 4.7: Home page interface.

### 4.3.3 Dashboard interface

Figure 4.8 showcases the dashboard interface of the management platform, displaying various modules such as policy search, new subscriptions, and validations pending. This interface centralizes crucial information, enhancing navigation and user efficiency.

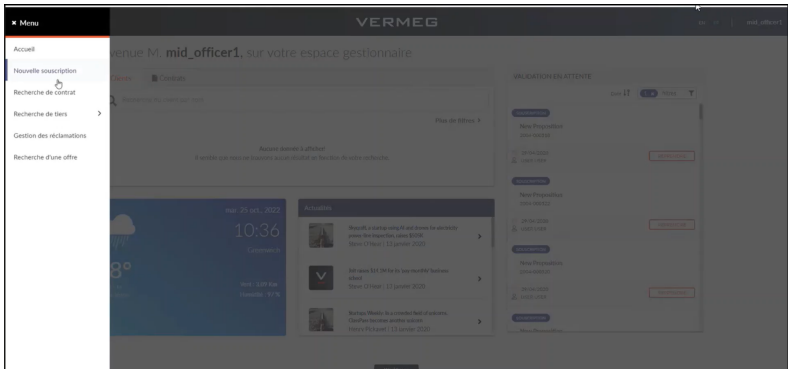



Figure 4.8: Dashboard interface.

### 4.3.4 Policy search interface

Figure 4.9displays the policy search interface, where users can efficiently query insurance contracts by various criteria such as contract number, policyholder name, status, and product type, streamlining the management of insurance portfolios.

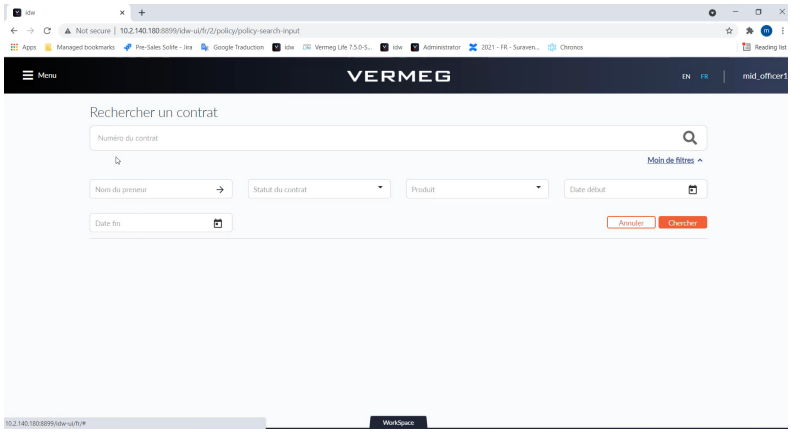



Figure 4.9: Policy search interface.

### 4.3.5 Policy interface

Figures 4.10 illustrates the policy interface, where users can view comprehensive details of individual insurance contracts, track payment schedules, manage documents, and review the status of policy endorsements, streamlining policy administration and oversight.

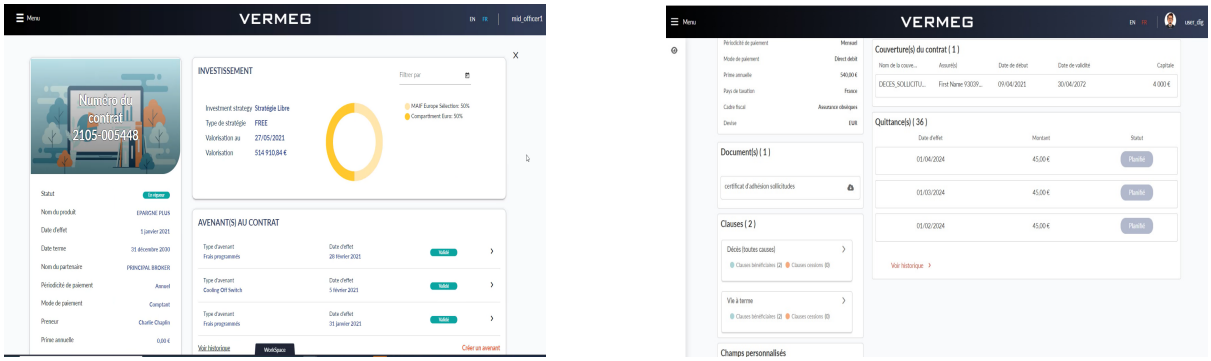



Figure 4.10: Policy interfaces.

### 4.3.6 Manage Endorsed Policy interface

Figures 4.11 displays the endorsement management interface, where users can easily track and manage modifications to insurance contracts, including operations like withdrawals, Switch, and premium top-ups, enhancing flexibility and control over policy adjustments.

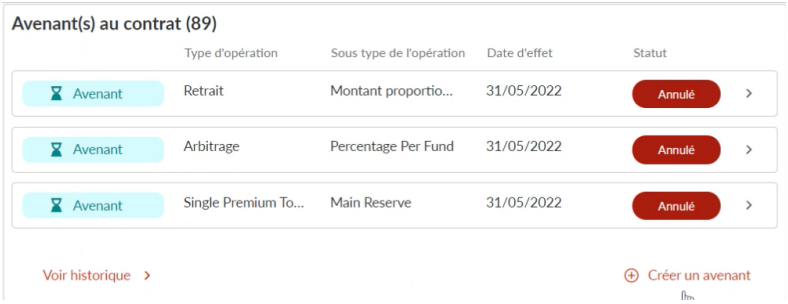



Figure 4.11: Manage EndorsedPolicy interface.

#### 4.3.6.1 Withdrawal interface

figure 4.12depict the withdrawal request interface, enabling manger to initiate and manage fund redemption requests efficiently, with options to specify the withdrawal type, amount, and to comply with regulatory conditions.

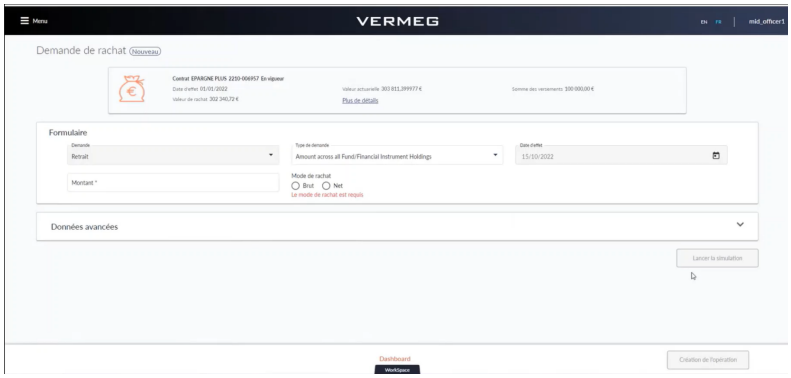



Figure 4.12: withdrawal interface.

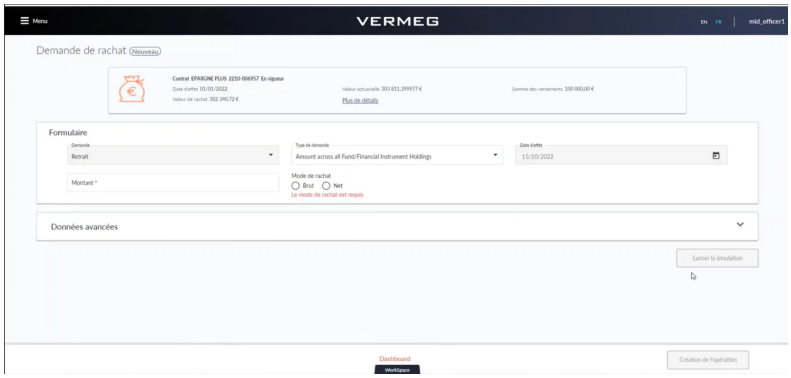



Figure 4.13: Steps withdrawal.

#### 4.3.6.2 Free Payment interface

Figure 4.14 showcases the interface for initiating a single premium top-up, allowing users to freely add Money to their existing insurance policies for enhanced coverage or investment growth.

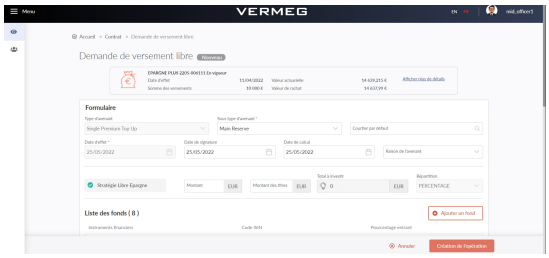



Figure 4.14: Free Payment interface.

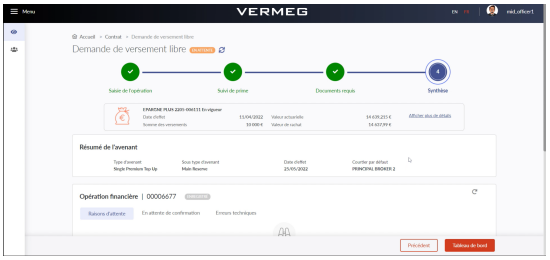
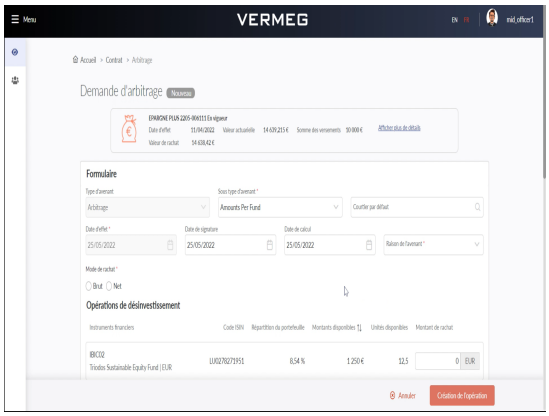



Figure 4.15: Steps Free Payment.

#### 4.3.6.3 Switch interface

Figure 4.16 displays the switch interface for arbitrage endorsements, facilitating the reallocation of investments within different unit-linked accounts in a life insurance policy to align with the insured's changing investment goals or market dynamics.



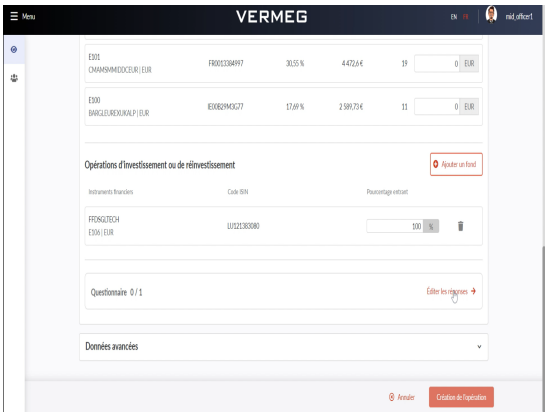




Figure 4.16: Switch interfaces.

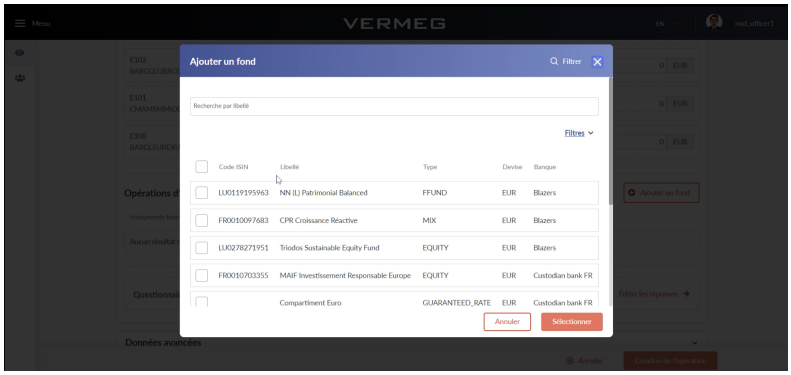



Figure 4.17: list Fund.

### 4.3.7 New Souscription interface

Figures 4.18 illustrates the first step in the new subscription process, where users initiate the creation of an insurance contract by selecting a client and optionally bypassing the simulation stage, facilitating a tailored approach to contract customization.

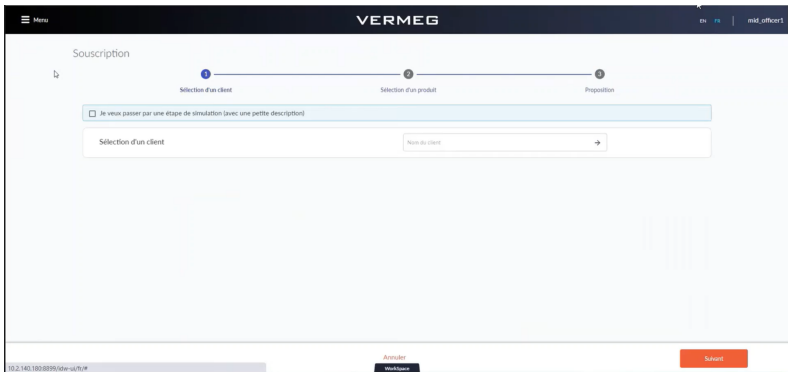



Figure 4.18: New Souscription interface.

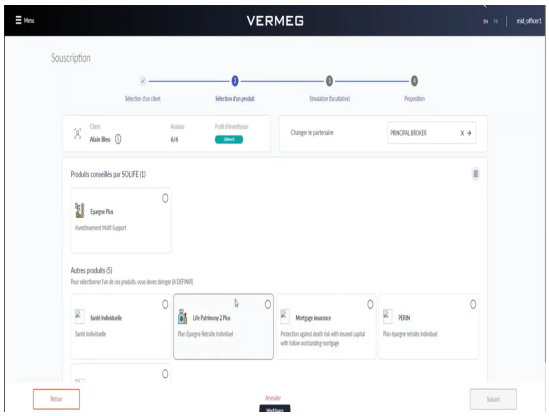



Figure 4.19: newSouscription step2

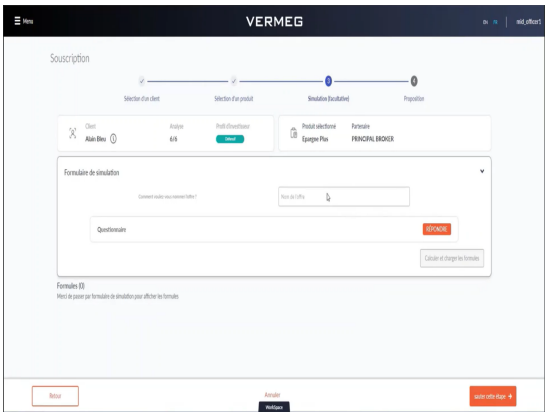



Figure 4.20: newSouscription step3

Figure 4.21: Subscription Workflow.

# *Conclusion*

This project was undertaken as part of our graduation requirement for the Computer Science degree at our university, focusing on the development of a digital pathway for consulting and managing administrative Endorsement in insurance contracts. The project allowed us to integrate theoretical knowledge with practical application by developing and deploying a digital solution that facilitates the management of insurance contracts.

Throughout this project, we collaborated closely, which not only enhanced our teamwork skills but also improved our communication abilities—key competencies in any software development environment. Regular meetings with our project Supervisor provided us with invaluable guidance, helping us overcome technical challenges and ensuring the project met all specified requirements.

The system we developed is designed to interact with the Solife backend system through RESTful APIs, which we designed and implemented to handle various administrative tasks such as Endorsement contract and Policy consultations. The flexibility of our design allows for future enhancements and integration with additional modules as new requirements emerge.

The practical experience gained from this project is immense. It not only allowed us to understand the complexities involved in developing real-world software solutions but also prepared us for the challenges we might face in our future careers as software developers. The project also underscored the importance of adaptability and continuous learning, which are crucial in the ever-evolving field of technology.

In conclusion, the successful completion of this project marks not just an academic achievement but also a significant step forward in our readiness to engage with and contribute to the tech industry. The application is now functional and deployed, ready for use by the insurance company to streamline their processes and enhance their service delivery.

# Webographie

[1] Vermeg:. https://www.vermeg.com/fr/.

[2] Produits vermeg:. https://www.vermeg.com/fr/produit-solife/.

[3] Unified process:. https://en.wikipedia.org/wiki/Unified_Process.

[4] Mvc architecture:. https://bit.ly/3MS1WGd.