République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Sousse

Institut Supérieur des Sciences
Appliquées et de Technologie
de Sousse

DEPARTEMENT INFORMATIQUE

RAPPORT DE PROJET DE FIN D'ETUDES

En vue de l'obtention du:
Diplôme National d'Ingénieur Informatique
Option: Génie Logiciel-Architecture Logicielle

Conception et développement d'une Lightning application de gestion des utilisateurs

Elaboré par :

Aslene OUAZE

Soutenu le ...../**.....**/**.....** **devant le jury :**

Président :	**Nom & Prénom Président**	**ISSAT de Sousse**
Examinateur :	**Nom & Prénom Examinateur**	**ISSAT de Sousse**
Encadrant :	**Nom & Prénom Encadrant**	**ISSAT de Sousse**
Encadrant Industriel :	**Mme. Rim Ben Abdesslem** **M.Arous Mandhour**	**TECHLEAD**

Année Universitaire : 2022 / 2023

Code Sujet: FI-

Thanks

At the end of this work, I would like to express my sincere thanks to all the people who have contributed, directly or indirectly, to its success.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Dedication

I dedicate this work to:
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

To all of you,
I dedicate this work.

Résumé

Ce travail a été développé dans le cadre d'un projet de stage de fin d'études qui a été réalisé au sein de la Société TECHLEAD. Ce projet consiste à concevoir et développer une lightning application dynamique pour le CRM Salesforce pour permettre aux administrateurs et aux directeurs de communautés de gérer leur communauté et ses utilisateurs.
Notre application donne également la possibilité de consulter l'historique de connexion des utilisateurs et un tableau de bord synthétique visualisant les KPI ainsi que la mise à disposition d'un Chatbot intelligent à l'administrateur.

Mots clés
Application Lightning, Gestionnaire de communauté, Chatbot, LWC, JS, CSS, Apex, Aura, SLDS, SOQL, SOSL, Architecture MVC

Summary

This work was developed as part of an end-of-study internship project which was achieved within the TECHLEAD company. This project involves designing and developing a dynamic lightning application for the Salesforce CRM to enable administrators and community managers to manage their community and its users.
Our application also gives the possibility of consulting users' connection history and a synthetic dashboard visualizing the KPIs as well as providing a smart Chatbot to the administrator.

Keywords
Lightning application, Community Management, Chatbot, LWC, JS, CSS, Apex, Aura, SLDS, SOQL, SOSL, MVC Architecture

Contents

List of Figures

List of Tables

General Introduction

In today's digital world, where businesses are increasingly relying on cloud-based services and software,such as the Salesforce Platform, efficient management of user accounts has become crucial. This area demonstrates multiple important investments thanks to the increasing number of interested developers in this platform who offer an infinity of useful applications.

These applications are accessible everywhere and through multiple devices as long as that device is connected to Salesforce, such availability is provided thanks to the robust web and mobile infrastructure of the Salesforce platform.

That's why we wanted to create a lightning solution for individuals and enterprises who want to manage and monitor their user's activity within Salesforce and we took community users as a base point. Our application will provide multiple information and statistics about each user as well as enable modifications to such information, it will also provide useful KPI charts and a smart chatbot solution for administrators and community managers.

Our end-of-study project entitled "Salesforce Community Management Lightning Application" concludes our summer training as a computer engineer.

The project was carried out over six months, within the company TECHLEAD. This report summarizes the stages of realization of this project. Its purpose is to situate the context of the project, to describe the resulting application, the methods, and tools used as well as the results obtained.

This report follows the following organization:

The first chapter is entitled "General Framework of the Project", which is an introductory

chapter presenting the host company, the problem, the solution proposed, and the objectives of the project, a study of the existing and the process of development of our application.

The second chapter, ”Specification of needs”, is used to identify the actors of our application and then to specify the functional and non-functional needs. functionalities to which our application must respond, making it possible to identify its main features.

The third chapter, ”Conception”, serves to describe the conceptual diagrams and the architecture applied to our proposed solution.

The fourth and final chapter, ”Realization”, illustrates the realization of our project through the presentation of the environment and the development tools as well as the visualization of the results of our work through the main application interfaces.

Finally, we end the report with a general conclusion in which we recapitulate the work carried out and we present the prospects.

Chapter 1

General framework of the project

Introduction

We begin this chapter with a general presentation of the organization of the reception. We will then detail the context and the problem of our project and the proposed solution, which will attempt to resolve the inconveniences that already exist. We will finish this chapter by describing the schedule of our internship through the Gantt chart as well as the development process.

1.1 *Presentation of the host organization*

TechLead is a Tunisian IT engineering company whose mission is to design and implement Salesforce solutions for companies to improve their productivity, profitability, and market adaptability.

The company supports its clients throughout the life cycle of their projects, from consulting to the complete implementation of the solution and up to the transfer of skills. The company's young, dynamic, and versatile team assists clients in all stages of implementing their Salesforce solution to better interact with customers, partners, and prospects. The company offers many services such as:

- **Accompaniment:** The company's Salesforce advisors help clients implement and develop your Salesforce solution. They can intervene in the audit and analysis of needs, the design, the integration of data, and the configuration of clients' projects quickly and efficiently.
- **Support:** The company's experienced developers can provide assistance and maintenance to clients' projects in the various administration or development needs with good availability and responsiveness.
- **Salesforce Training:** The company provides training tailored to client's needs and helps them use and leverage the capabilities of Salesforce.

Figure 1. *Host organization TECHLEAD*

1.2 *Project presentation*

In this part, we put our work in its general context. first, we present the context. Second, we present the problem and the reasons that led us to suggest this topic. Third, we present the solution proposed to solve the problem. Finally, we will describe the objectives of this project.

1.2.1 Context

Salesforce is a highly customizable advanced CRM, it stores customer data, gives processes to nurture prospective customers, and provides ways to collaborate with other workers. [**1**]
Salesforce comes with a lot of standard functionality, or out-of-the-box products and features

that clients can use to run their business. Here are some common things businesses want to do with Salesforce and the features Salesforce gives that support those activities:[**1**]

You need to:	So we give you:
Sell to prospects and customers	Leads and Opportunities to manage sales
Help customers after the sale	Cases and Communities for customer engagement
Work on the go	The customizable Salesforce mobile app
Collaborate with coworkers, partners, and customers	Slack, Chatter, and Communities to connect your company
Market to your audience	Marketing Cloud to manage your customer journeys

Figure 2. *Salesforce features, Source: [**1**]*

The platform also helps clients move fast. Part of that speed comes from replacing tasks that clients are used to doing by hand with more streamlined processes.[**2**]

The platform's goal is to make big changes with minimal effort and to solve mistakes that impact the buyer using dynamic expandable interfaces using additional extensions.

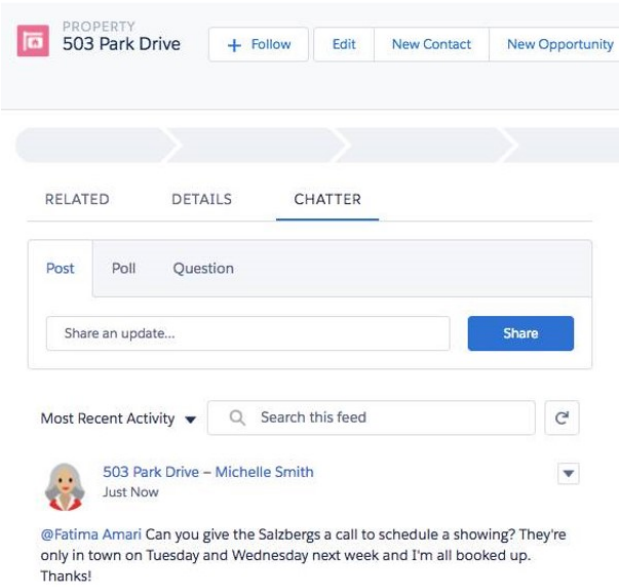

Figure 3. *Chatter extension in Salesforce, Source: [**2**]*

Here are a few use cases for different departments:

For employees who work in...	Customize the platform for...
Finance	• Budget management • Contract management • Pricing
Product	• Warranty management • Preproduction testing • Product ideas and innovation
Supply Chain	• Procurement • Vendor management • Logistics
Ops	• Asset and facilities management • Merger and acquisition enablement • Business agility

Figure 4. *Usecases for Salesforce, Source: [2]*

Salesforce is a cloud company. Everything to offer resides in the trusted, multitenant cloud.[**3**]

The Salesforce platform is the foundation of the services. It's powered by metadata and made up of different parts, like data services, artificial intelligence, and robust APIs for development. [**3**]

All the apps sit on top of the platform. The prebuilt offerings like Sales Cloud and Marketing Cloud, along with apps built using the platform, have consistent, powerful functionality.[**3**]

Everything is integrated. The platform technologies like predictive analytics and the development framework are built into everything to offer and everything to build.[**3**]

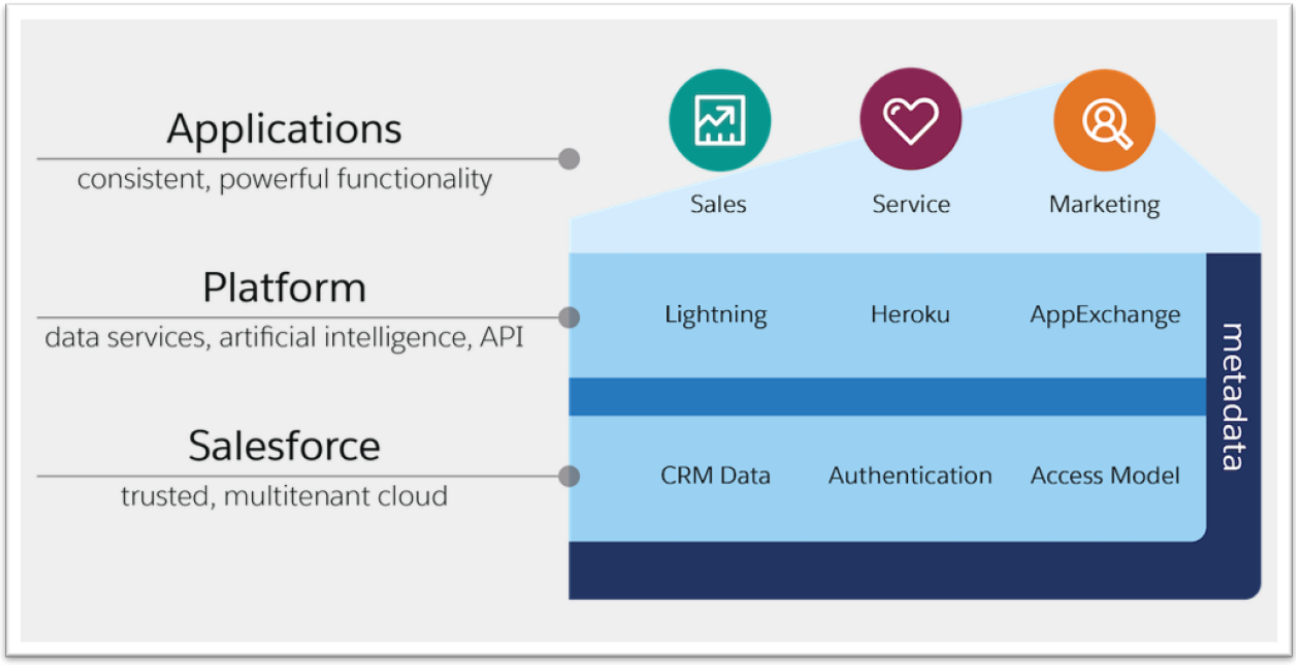

Figure 5. *Salesforce's architecture, Source: [**3**]*

1.2.2 Problem

In Salesforce, each user is identified by a unique username and profile. Along with other settings, the profile determines what tasks a user can perform, what data they can view, and how they can use the data.

As a Salesforce admin, you manage users in your organization. In addition to creating and assigning users, user management includes managing permissions and licenses, delegating users, and more

Users are managed in the community through a Salesforce interface and over several stages which makes it difficult and time-consuming.

1.2.3 Proposed solution

The main objective of our work is to design and develop a powerful tool to facilitate the task of managing the users of the community.

This application is intended to offer any organization a simple and effective means to manage the

”administration” part, then the connection history part, and finally provide a synthetic dashboard visualizing the KPIs as well as a smart Chatbot solution for the administrators and community managers.

1.2.4 Objectives

Ensure user satisfaction by ensuring:

Functional objectives:

1. **Choice of the community:** Select the community to which we will manage the users
2. **Manage users:** The system must allow users to be managed with the functionalities of activation, deactivation, modification, and consultation of the list of users via a data table. Creation of different filters that allow us to facilitate the navigation of the list of users.
3. **Manage connection history:** Create a chart that shows the number of user connections per day, week, year, or according to a well-determined date. This allows the administrator to modify the user's license
4. **Offer a synthetic dashboard**
5. **Offer a smart Chatbot solution**

Non-functional objectives:

1. **Security:** Access to information is only possible after verification of privileges and access rights, for example Authentication, Redirections.
2. **Ergonomics and user-friendliness:** The application will provide a user-friendly and easy-to-use interface that does not require any prerequisites, so it can be used by all types of users (even non-computer specialists).
3. **Extensibility and maintainability:** The architecture of the application will allow the evolution and maintenance (addition or deletion or update) at the level of its various modules in a flexible manner.
4. **Performance:** The application must be efficient, i.e. the system must react within a period that does not exceed 5 seconds, whatever the action of the application.
5. **Availability:** The application will be available on 24/24, and 7/7 except during the maintenance period.

Technical objectives:

1. Organization of the application according to the MVC architecture: a software architecture model that separates the representation of information from the user's interaction with it.

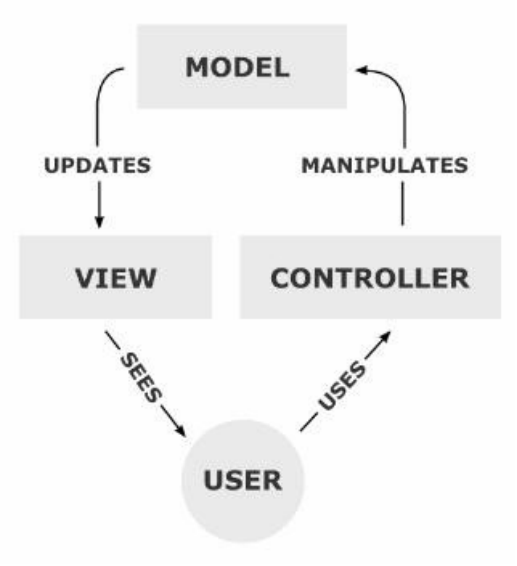

Figure 6. *MVC Architecture, Source: [**6**]*

2. Using the Framework "LWC".
3. Using the programming language "Apex".
4. Using the library "SLDS".
5. Using the Salesforce Object Query Language "SOQL/SOSL".

1.3 *Study of the existing*

In this part, we analyze and criticize the existing applications currently, through the following table propose a solution that solves their drawbacks.

Application	**Pros**	**Cons**
HubSpot	- User-Friendly Interface - Ability to create custom roles and assign specific permissions to users based on their roles. - Ability tracks user activity, allowing businesses to monitor user behavior.	- Interface can be complex, particularly for businesses with a large number of users. - Pricing structure is based on the number of contacts in a business's database, which can make it more expensive for businesses with larger teams. - There may be a learning curve when it comes to managing users and configuring access control.
ZenDesk	- Provides user analytics, allowing businesses to monitor user behavior. - Allows businesses to set up custom notifications for user actions, such as ticket creation or update. - Provides tools for user collaboration, such as shared views and comments.	- May not offer as much granularity as some businesses require. - Interface may not offer as much customization as some businesses require. - Pricing structure is based on the number of agents, which can make it more expensive for businesses with larger teams.
Conga	- Offers customizable workflows. - Offers Centralized user management system. - Provides tools for user collaboration, such as document sharing and commenting	- Limited third-party integrations - Pricing structure can be more expensive for businesses with larger teams. - User management interface can be complex.

Table 1. *Study of the existing*

1.4 *Development process*

A software development process is a set of related activities followed by a team led to the production of the software within the organization. It consists of a detailed plan describing how to develop, design, test, deploy, and maintain the product. [**4**]

1.4.1 Incremental development

In this context, we adopt the process of incremental development as an approach to the realization of our project. According to this process, the customer's needs are specialized, the software is globally designed, then the realization is done by an increment of functionalities.[**4**] Each increment is considered an executable part of the final system. These increments are successively integrated into the final product and at each stage, the software is tested, operated, and maintained as a whole.[**4**]

Implementing the software by increment makes it possible to take into account the risk analysis to facilitate the detection of errors at the earliest according to customer feedback and to reduce time and cost of production, which helps in the realization of software quality. [**4**]

1.4.2 Provisional schedule of tasks

A Gantt chart is a graphical tool that represents the management of the project over time, which facilitates its implementation.

Indeed, the internship within TECHLEAD will run for a period of 4 months. The following timeline illustrates a provisional schedule set early in development, representing the main stages leading to a functional solution that meets the criteria defined by previously mentioned specifications.

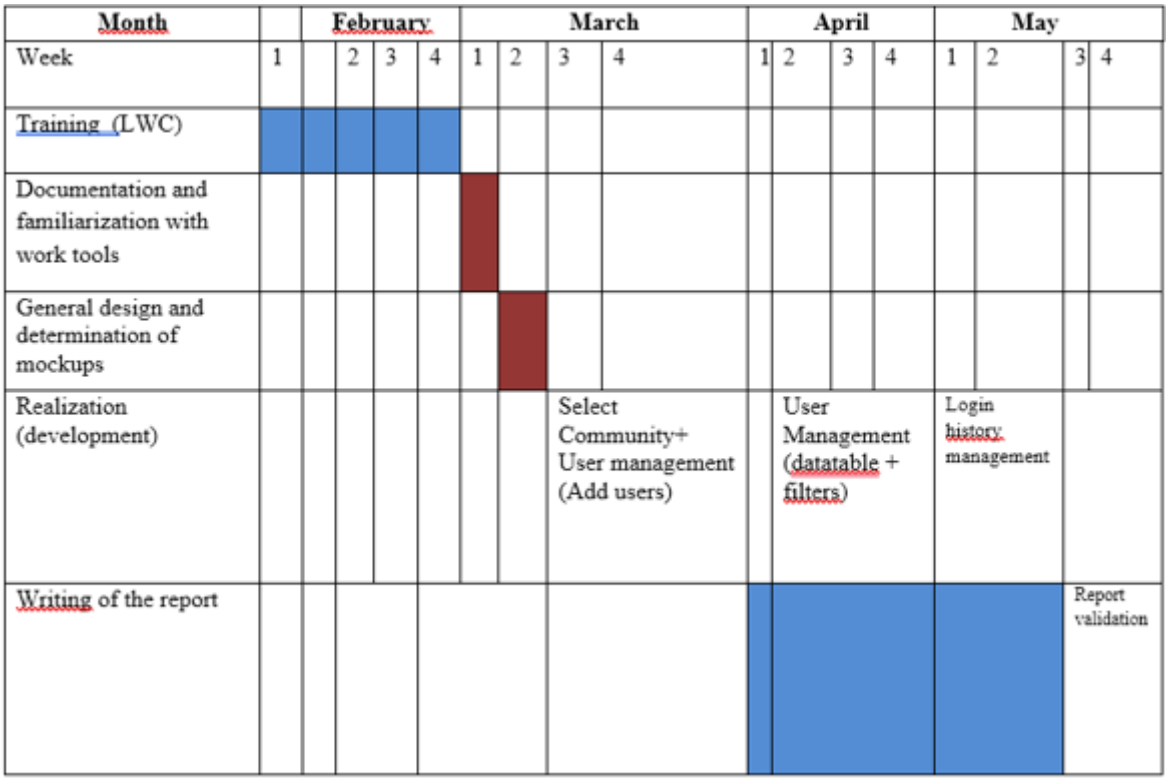

Figure 7. *Gantt diagram*

Conclusion

In this chapter, we have introduced the context of our project by representing the host organization in the first place. Secondly, we have introduced the sales platform and its features. Thirdly, we have cleared up the problem. Then, we described the proposed solution and the objectives to be achieved. After that, we analyzed the existing applications. Lastly, we have depicted the advancement of activities throughout the project according to the adopted development process. In the next chapter, we will specify the functional requirements and the non-functional needs.

Chapter 2

Specification of needs

Introduction

In this chapter, we will identify the actors, then we will specify the functional and non-functional needs that the proposed solution must meet. Finally, we present the use case diagrams explaining our application's main functionalities.

2.1 *Identification of actors*

An actor represents an external entity that interacts directly with the system. It can be either a human person or a system. We distinguish two types of actors, the main actor, and a secondary actor. Indeed, a principal actor obtains an observable result of the system while a secondary actor is asked for additional information.

Main actors

Community Manager & Salesforce Administrator

Both the community manager and the Salesforce Administrator are the main users of our application and should be able to :

- Choose the community that he has the right to access and manage.
- Consult users of a specific community inside a well-organized table with pagination options.
- Filter said users by name, user-name, Salesforce account name, status (active or not active), and Salesforce profile.
- Consult and update the details of each user.
- Activate users within a specific community.
- Deactivate users within a specific community.
- Send a ”Welcome to community” email to a specific active user.
- Send a ”Reset password” email to a specific active user.
- Consult a bar chart that shows the number of logins of each user within the selected community.
- Filter chart results by the period between two specific dates.
- Consult details about each user displayed in the chart.
- Update the Salesforce user license for each user displayed in the chart.
- Consult users failed login attempts to a specific community inside a well-organized table with pagination options.
- Filter said login attempts by name, user-name, status(Invalid password, No community access, etc...) and event date and time.
- Consult detailed information about each login attempt.
- Send a security warning email to the account owner about the login attempt event.
- Access a Chatbot that provides information about the selected community or the Salesforce organization.

- Access a synthetic dashboard displaying community KPIs (Key Performance Indicators).

In case when the community manager or the Salesforce Administrator has a Salesforce role he will be also able to :

- Add one or multiple users at once to a specific community.

Secondary actors

Salesforce System

This actor is the system, previously developed and deployed in a cloud server by the Salesforce organization, interactable through our application, and it's responsible for :

- Sending an automatic "Welcome to community" email upon adding a new member to the community by our application user.
- Sending an automatic "Welcome to community" email upon activating a previously deactivated user by our application user.
- Generating reset password URL upon sending a "Reset password" email to a community member by our application user.
- Tracking users' successful and failed login attempts and saving them to the organization database.

2.2 *Functional Needs*

Functional needs are expressed by the user of the application which makes it possible to identify the functionalities of this application.
In our case, the functional needs are:

- Choice of the community:
  Select the community to which we will manage the users.

- Manage users:

  The system must allow users to be managed with the functionalities of activation, deactivation, modification, and consultation of the list of users via a data table. Creation of different filters that allow us to facilitate the navigation of the list of users.

- Manage connection history:

  Create a chart that shows the number of user connections per day, week, year, or according to a well-determined date. This allows the administrator to modify the user's license.

- Offer a synthetic dashboard:

  Allowing the system administrator / the community manager to visualize the KPIs (Key Performance Indicators) of his organization/community.

- Manage failed login attempts:

  Allowing the system administrator / the community manager to consult details about failed login events as well as send security warning emails to concerned users.

- Access a smart Chatbot:

  Allowing the system administrator / the community manager to request pieces of information from a smart digital assistant.

2.3 *Non-functional Needs*

Non-functional requirements represent the characteristics of the system. They relate to the constraints to be taken into consideration to set up an adequate solution.
For our application, the non-functional requirements are:

- Security:

  Access to information is only possible after verification of privileges and access rights, for example Authentication, Redirections.

- Ergonomics and user-friendliness:

  The application will provide a user-friendly and easy-to-use interface that does not require any prerequisites, so it can be used by all types of users (even non-computer specialists).

- Extensibility and maintainability:

  The architecture of the application will allow the evolution and maintenance (addition or deletion or update) at the level of its various modules in a flexible manner.

- Performance:

  The application must be efficient, i.e. the system must react within a period that does not exceed 5 seconds, whatever the action of the application.

- Availability:

  The application will be available on 24/24 and 7/7 except during maintenance.

2.4 *Use case diagrams*

In this section, we will highlight the system's functionalities to be from the functional needs mentioned above based on the UML(Unified Modeling Language) diagrams which group together all the system's use cases.

2.4.1 Global use case diagram

The following figure illustrates the global use case diagram of our application

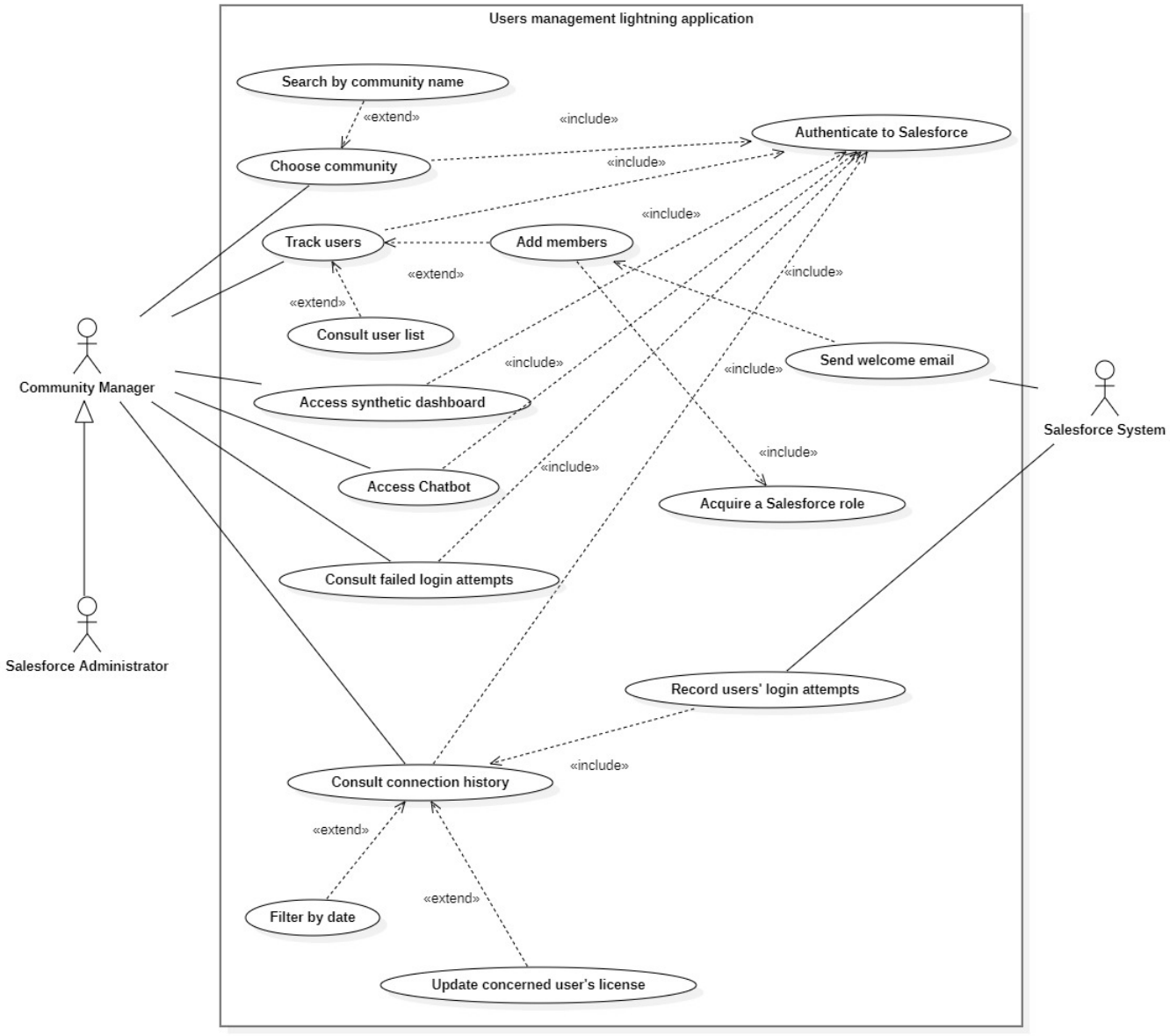

Figure 8. *Global use case diagram*

2.4.2 Use case refinement

In this section, we will detail the main use cases.

2.4.2.1 Consult user list use case refinement

In our application, the community manager and the system administrator can access a data table containing the full list of users within their respective communities.

The following figure shows the Consult user list use case diagram.

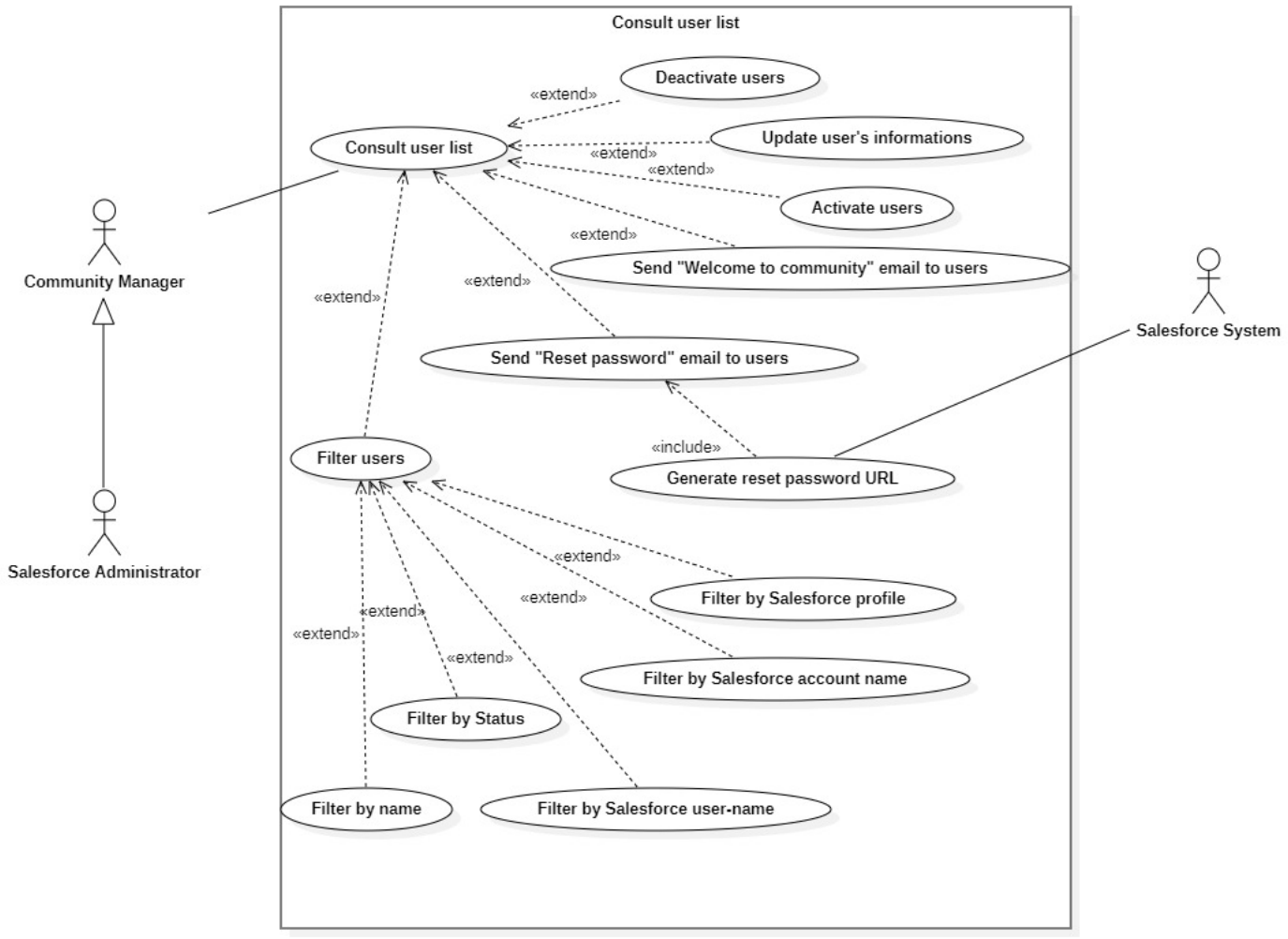

Figure 9. *Consult user list use case diagram*

The following table details the tasks to be performed by the user to manage the user list.

Summary	
Title	Consult user list
Objectif	Allowing the community manager or the system administrator to manage his community members
Actors	Community manager, System administrator
Description of sequences	
Pre-condition	- User should be authenticated to Salesforce - User should be the owner of at least one community
Post-condition	- Community members are well organized
Normal scenario	1. User accesses the user list interface 2. User selects one or many community members 3. User clicks the ”activate/deactivate” button for selected members 4. The system prompts the success of the operation 5. The user clicks the ”Send welcome email” button for the selected members 6. The system prompts the success of the operation 7. The user clicks the ”Send reset password” button for the selected members 8. The system prompts the success of the operation 9. The user clicks the ”show details” button for one member 10. The member information interface pops up 11. User updates the selected member information and complies 12. System prompts the success of the operation
Alternative scenario	1. License limit exceeded when activating members: The system sends an error message 2. Sending a welcome email to a deactivated member: The system sends an error message 3. Changing member username to an existing one: The system sends an error message
Non-functional constraints	1. The interface must be ergonomic 2. Error messages should be understandable and clear

Table 2. *Consult user list use case*

2.4.2.2 Add members use case refinement

In our application, the community manager and the system administrator can add one or multiple members at once to their respective communities.

The following table details the tasks to be performed by the user to add members to his community.

Summary	
Title	Add members
Objectif	Allowing the community manager or the system administrator to add one or multiple members at once to his community
Actors	Community manager, System administrator
Description of sequences	
Pre-condition	- User should be authenticated to Salesforce - User should be the owner of at least one community - User should have a Salesforce role
Post-condition	- New member(s) are added to the owner's community
Normal scenario	1. User accesses the add members interface 2. The user fills the displayed form with information about the new member 3. The user clicks the " add another member" button 4. A new empty form, identical to the previous one, is displayed 5. The user fills the new form with information about the next new member 6. The user clicks the "clone member" button 7. A new form containing the same values as the previous form is displayed 8. The user clicks the "delete form" button 9. The selected form is deleted from the screen 10. The user clicks the "submit" button 11. System prompts the success of the operations
Alternative scenario	1. Empty fields: The system sends an error message: You must complete all the required fields 2. The email entered is not valid: The system sends an error message describing the validation conditions for this field 3. Duplicate Salesforce user-name: The system sends an error message
Non-functional constraints	1. The interface must be ergonomic 2. Error messages should be understandable and clear

Table 3. *Add members use case*

2.4.2.3 Consult synthetic dashboard use case refinement

In our application, the community manager and the system administrator can access a dashboard containing features that helps them to manage their respective communities.
The following figure shows the Consult synthetic use case diagram.

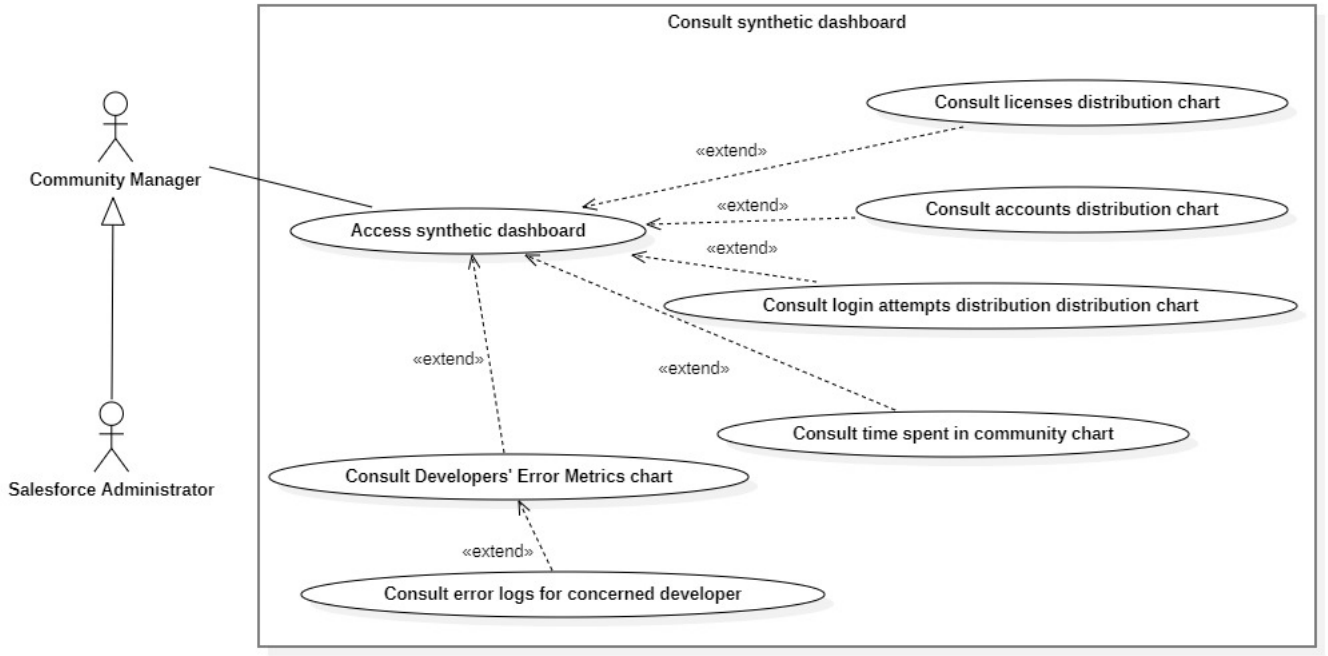

Figure 10. *Consult synthetic dashboard use case diagram*

The following table details the tasks to be performed by the customer to consult the synthetic dashboard in our application.

Summary	
Title	Consult synthetic dashboard
Objectif	Allowing the system administrator to access extra features that helps him to manage his communities
Actors	System administrator, Community Manager
Description of sequences	
Pre-condition	- User should be authenticated to Salesforce - User should be the owner of at least one community
Post-condition	- System administrator is well informed about his respective community's KPIs.
Normal scenario	1. User accesses the dashboard interface 2. Multiple KPI charts pops up 3. User may click a bar from the developers' Error Metrics Chart to access error logs specific to the selected developer
Alternative scenario	
Non-functional constraints	1. The interface must be ergonomic

Table 4. *Consult synthetic dashboard use case*

2.4.2.4 Consult the failed login attempts use case refinement

In our application, the community manager and the system administrator can manage failed login attempts committed by their community's members as well as send security warning emails containing details about the event to concerned members.

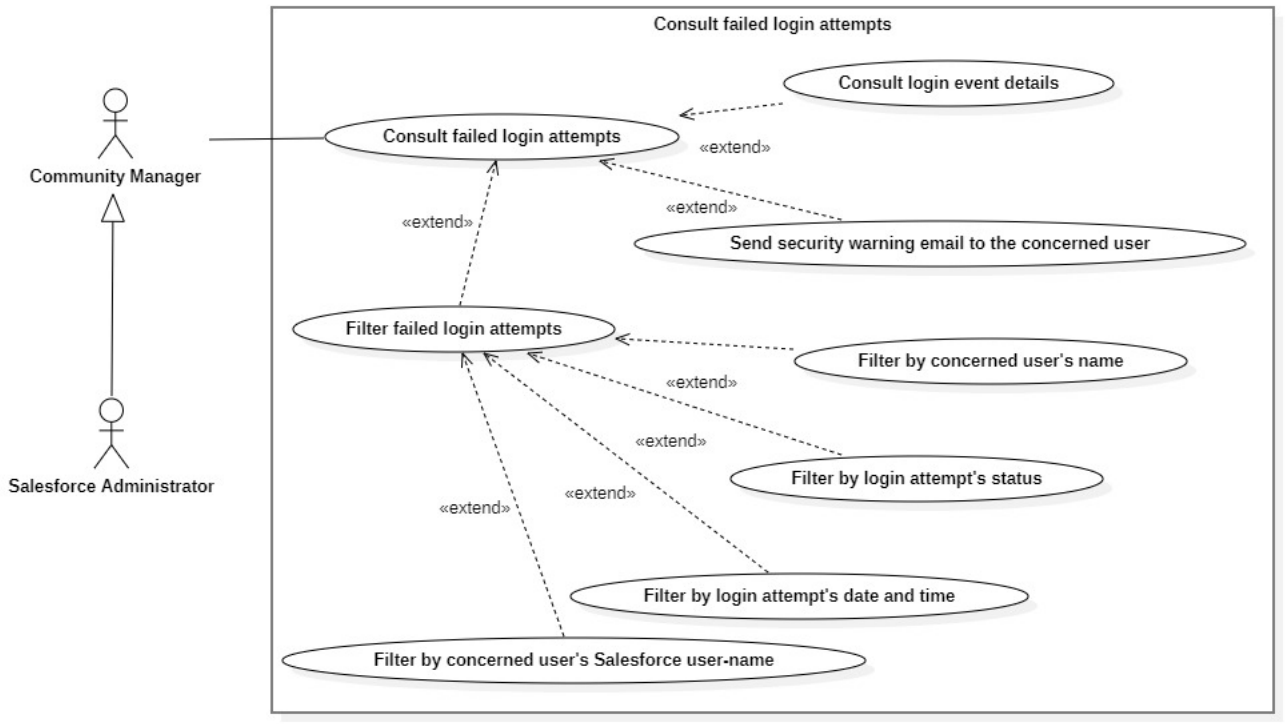

Figure 11. *Consult failed login attempts use case diagram*

The following table details the tasks to be performed by the customer to consult the failed login attempts in our application.

Summary	
Title	Consult failed login attempts
Objectif	Allowing the system administrator to access failed login attempts and inform concerned users about them
Actors	System administrator, Community manager
Description of sequences	
Pre-condition	- User should be authenticated to Salesforce - User should be the owner of at least one community
Post-condition	- Users are well-informed about their account security status.
Normal scenario	1. User accesses the failed login attempts interface 2. The user clicks the ”show details” button for a specific login event 3. The ”show details” interface for the selected login event pops up 4. The user clicks the ”send security warning email” button for the specific login event 5. The system prompts the success of the operation 6. The concerned user receives the warning email, containing details about the failed login attempt, in his email
Alternative scenario	1. Server error when sending the warning email: The system sends an error message received from the server
Non-functional constraints	1. The interface must be ergonomic 2. Error messages should be understandable and clear

Table 5. *Consult failed login attempts use case*

2.4.2.5 Access Chatbot use case refinement

The following figure illustrates the use case diagram ”Access Chatbot”

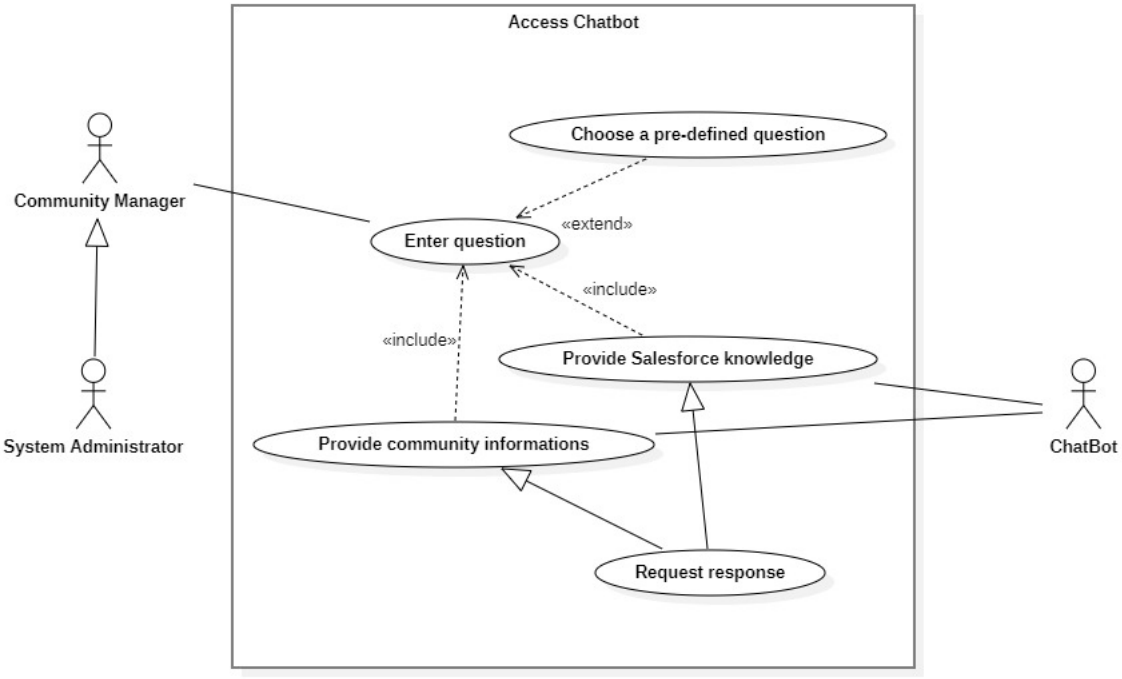

Figure 12. *Access Chatbot use case diagram*

The following table details the tasks to be performed by the customer to Access the Chatbot within our app.

Summary	
Title	Access Chatbot
Objectif	Trigger a Chatbot that responds to user requests while providing informations about the current community
Actors	System administrator, Community manager, Chatbot
Description of sequences	
Pre-condition	- User should be authenticated to Salesforce - User should be the owner of at least one community
Post-condition	- User question is answered and the response is displayed on the screen.
Normal scenario	1. User accesses the Chatbot interface 2. User types a question or selects a predefined question. 3. The chatbot is launched upon receipt of a request 4. The chatbot analyzes the request 5. The chatbot prepares the response 6. The chatbot sends the response
Alternative scenario	1. An error message is displayed if the question is not recognized by the system.
Non-functional constraints	1. The interface must be ergonomic 2. Error messages should be understandable and clear 3. The ChatBot must always be available.

Table 6. *Access Chatbot use case*

Conclusion

In this chapter, we have described the specification phases of the needs of our developed application to identify the different actors as well as the features and services that our application must provide.

We have detailed these features with use case diagrams. The next chapter will be devoted to the conception phase.

Chapter 3

Conception

Introduction

After identifying the functional and non-functional needs and the main functionalities of our application. We begin this part of the conceptual study by describing the general architecture of our system and it's detailed internal modeling through class and sequence diagrams.

3.1 *Global Architecture*

In this section, we will give an overview of the definition of the the architectural pattern was chosen to model our application and we will detail the projection of this pattern on our application.

3.1.1 Definition of the Model-View-Controller design pattern

Our chosen architecture of work for this project will be the MVC architecture since it is recommended by the Salesforce developer's community.

The model-view-controller (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information.[**5**]

The pattern requires that each of these be separated into different objects:

- **The model** (for example, the data information) contains only pure application data; it contains no logic describing how to present the data to a user. [**5**]
- **The view** (for example, the presentation information) presents the model's data to the user. The view knows how to access the model's data but does not know what this data means or what the user can do to manipulate it. [**5**]
- **The controller** (for example, the control information) exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.[**5**]

Thanks to these principles, the components of the application become easy to manage, test, modify, and reuse.

The following figure explains furthermore the role and the idea behind each component of the MVC pattern:

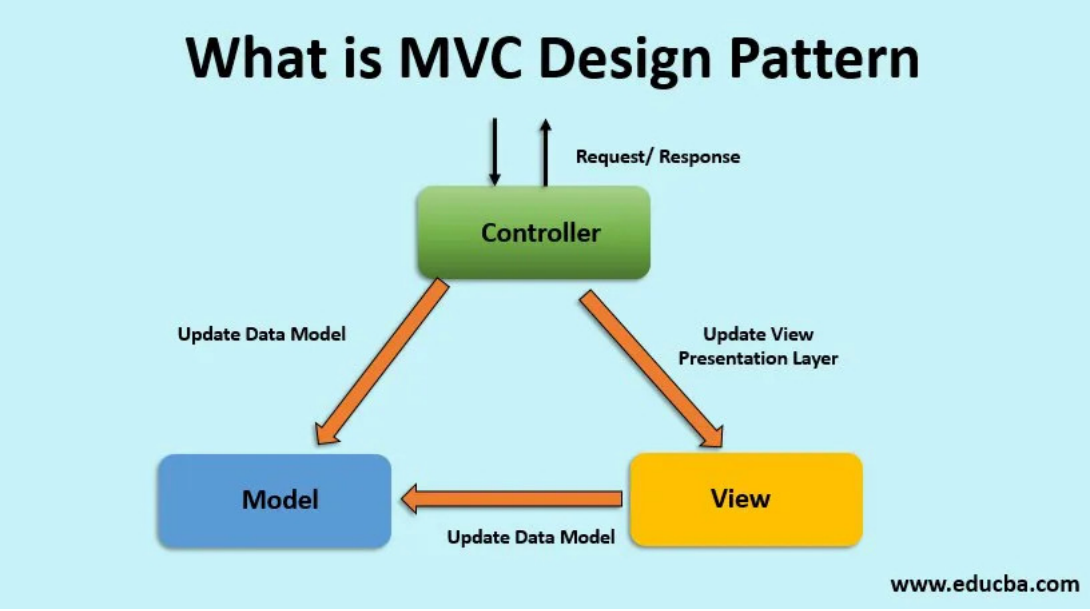

Figure 13. *Understanding MVC Design Pattern, Source: [**7**]*

3.1.2 Application of the ”MVC” pattern

We explain in this paragraph the details of the projection of the pattern MVC on our app.
The following figure represents the class diagram which illustrates the binding between entities (classes). Indeed, this diagram translates the content of the **Model** component, and it's based on the schema that is already established within the Salesforce infrastructure.

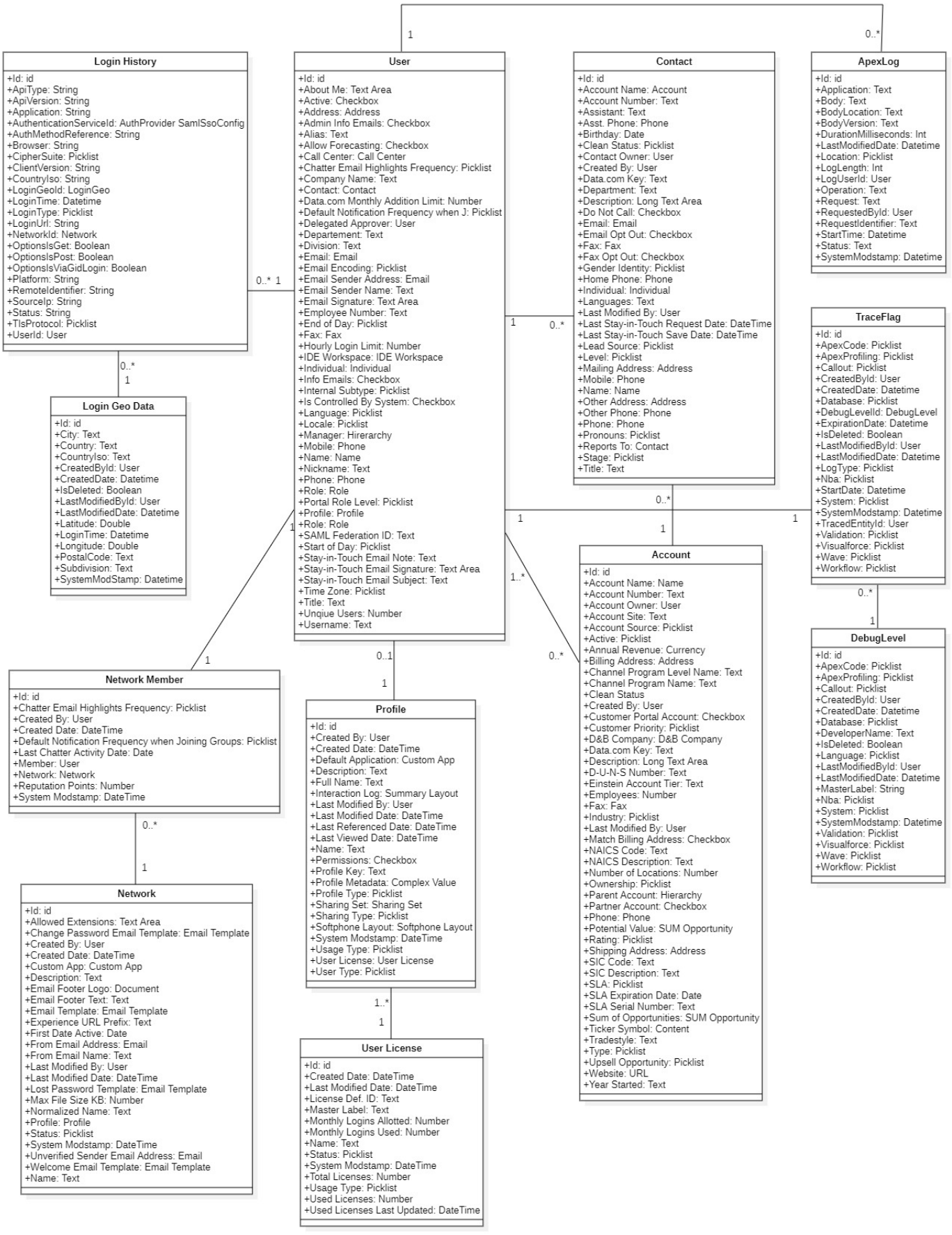

Figure 14. *Class diagram representing the "Model" component*

Description of classes representing our Model:

- **User:**

  This class represents our application's Salesforce user who is the main focus of the administrator and the community manager when working within our app. This class's most important attributes in our application are:

  - Id: the unique identification key for the user
  - Active: determines the status of the user (active or not active)
  - Address: the full address of the user
  - Alias: the alias of the user, usually composed of the first letter of his first name and his last name concatenated together
  - Contact: a lookup field that connects the user to its corresponding Salesforce contact which is necessary to have to allow the user to access a community
  - Email: the email of the user
  - Email Encoding: the encoding of the user's email (for example UTF-8), useful for sending appropriate emails to the user
  - Is Portal Enabled: determines if the user can access a community or not
  - Language: the preferred interface language of the user that will control the user interface of the accessed community
  - Locale: the geolocation locale id of the user (example Arabic(Tunisia))
  - Name: the full name of the user which is composed automatically by the Salesforce system by concatenating his first name and his last name
  - Profile: a lookup field that connects the user to its corresponding Salesforce profile which controls the level of access provided to the user (for example System Administrator which gives full access to all Salesforce objects and configurations)
  - Role: the relative company role of the user (for example CEO), this field is required to be able to add a user to a community through our app

- Time Zone: the timezone of the user (for example GMT+02:00)
- Username: the unique Salesforce user name if the user usually identical to the user's email

- **Contact:**

  This class represents the Salesforce contacts connected to the community user in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the contact
  - Account Name: a lookup field that represents the Salesforce unique account connected to the contact
  - Contact Owner: a lookup field that represents the user owning the Salesforce contact
  - Created By:: a lookup field that represents the user who created the Salesforce contact
  - Email: email of the contact
  - Last Modified By: a lookup field that represents the last user to modify details of the contact
  - Name: the full name of the contact

- **Account**

  This class represents the Salesforce account connected to the community user in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the account
  - Account Name: Salesforce account name
  - Account Owner: a lookup field that represents the user owning the Salesforce account
  - Created By: a lookup field that represents the user who created the Salesforce account

- Customer Portal Account: determines if the account can be connected to the community user
- Last Modified By: a lookup field that represents the last user to modify details of the account
- Parent Account: a self-lookup field that represents the parent account of this account through the Salesforce hierarchy

- **Profile**

  This class represents the Salesforce profile connected to the community user in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the profile
  - Created By: a lookup field that represents the user who created the Salesforce profile
  - Full Name: the full name of the profile that represents its usage context
  - Last Modified By: a lookup field that represents the last user to modify details of the profile
  - Name: the name of the Salesforce profile that represents the main way of identifying the profile through other objects
  - User License: a lookup field that represents the Salesforce purchased user license connected to the profile

- **User License**

  This class represents the Salesforce purchased user license connected to the community user in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the user license
  - Name: the name of the Salesforce user license that represents the main way of identifying the user license through other objects
  - Total Licenses: represents the total number of purchased user licenses dedicated to community users

- Used Licenses: represents the number of used user licenses which determines if the administrator or the community manager can add more users to his community depending on how many purchased user licenses are left

- **Network**

  This class represents the community created by the administrator or the community manager in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the network
  - Created By: a lookup field that represents the user who created the Salesforce community
  - Created Date: represents the date and time of the creation of the Salesforce community
  - Description: a brief description of the community set by its creator to summarize its purpose
  - From Email Address: represents the email that will be displayed when a user receives an email through our application
  - From Email Name: represents the name that will be displayed when a user receives an email through our application
  - Last Modified By: a lookup field that represents the last user to modify details or the content of the community
  - Last Modified Date: represents the date and time of the last activity within the community
  - Lost Password Template: represents the email template that will be used when sending a "Reset password" email through our application
  - Name: the name of the Salesforce community that represents the main way of identifying the community through other objects
  - Profile: a lookup field that represents the Salesforce profiles that are allowed to access the community

- Welcome Email Template: represents the email template that will be used when sending a ”Welcome to community” email through our application

- **Network Member**

  This class represents the users that are considered members of a Salesforce community in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the network member
  - Created By: a lookup field that represents the user who added the new member to the Salesforce community
  - Member: a lookup field that represents the user representing the member of the community, this field is used to access all the details about the user within the Network Member object
  - Network: a lookup field that represents the community that the member is considered a part of, this field is used to access all the details about the community within the Network Member object

- **Login History**

  This class represents the login event details that are recorded by Salesforce regarding users' activity and will be used to track the activities of the members of any Salesforce community in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the login history
  - Browser: represents the commercial name and the version of the web browser that the login vent happened from (for example Chrome 112)
  - CountryIso: the first two letters of the country where the login event happened from (for example TN stands for Tunisia)
  - LoginGeoId: a lookup field that represents the geographic location connected to the login event
  - LoginTime: the exact date and time of the login event

- NetworkId: a lookup field that represents the community that the user wanted to access when the login event was recorded
- Platform: represents the commercial name and the version of the platform that the login vent happened from (for example Windows 10)
- SourceIp: the IP address of the user that tried to access the community
- Status: the state of the login attempt (for example Invalid Password)
- UserId: a lookup field that represents the user concerned with the login event, this field is used to send security warning emails, within our application, to users concerned with a failed login attempt

- **LoginGeo**

  This class represents the login event's geographic location details that are recorded by Salesforce regarding users' activity and will be used to track the geographic location of the activities of the members of any Salesforce community in our application. This class's most important attributes in our application are:

  - Id: the unique identification key for the loginGeo
  - City: represents the city when the login event happened (for example Mahdia)
  - Country: represents the country when the login event happened (for example Tunisia)
  - CountryIso: the first two letters of the country where the login event happened from (for example TN stands for Tunisia)
  - Latitude: represents the geographic latitude of the login event
  - Longitude: represents the geographic longitude of the login event

- **TraceFlag**

  This class represents the tracked users and entities regarding monitoring logs for programming errors and bugs committed by developers in the community and will be used to register said users so that their error logs will be monitor by the administrator. This class's most important attributes in our application are:

- Id: the unique identification key for the traceFlag
- StartDate: represents the tracking event's starting date for the created trace flag
- ExpirationDate: represents the expiration date for the created trace flag
- DebugLevelId: a lookup field that connects the traceflag to the Debulevel object in Salesforce , in our application this field will have the default value of the SFDC DevConsole's Id
- LogType: determines the type of logging that will be returned when requesting it, in our application it will take the default value of "USER _ DEBUG" since we will be tracking users error logs
- TracedEntityId: represents the id of the tracked user

- **DebugLevel**

  This class represents a configuration setting that determines the level of detail for debugging and logging activities in the system. This class's most important attributes in our application are:

  - Id: the unique identification key for the debugLevel
  - MasterLabel: represents debugLevel main label that will be used in our application to retrieve the id of the object and connect it to the TraceFlag object

- **ApexLog**

  This class represents the detailed logs of the execution of all Apex code ran by the tracked user within the community, and it will be used in our application to provide said logs the administrator. This class's most important attributes in our application are:

  - Id: the unique identification key for the apexLog
  - Body: represents the plain text containing execution traces, debug operations and outputs and errors provided by the called Apex methods
  - LogUserId: a lookup field representing the user who ran the Apex code which generated the ApexLog

- StartDate: represents date and time marking the beginning of the Apex code's execution
- Status: represents the title of the main event's log (for example out of bound exception)

The classes we have defined in the "Entities" layer are related to the German business field, employable in various types of commercial applications, and independent of our application. Thus, we made the principle of independence between the business rules and the application rules.

For the rest of the layers of the Clean architecture pattern, we will describe the dependencies between these layers by giving an example of a connection between a set of classes belonging to these layers. The following figure represents the general dependence in our case between the layers of the clean architecture pattern.

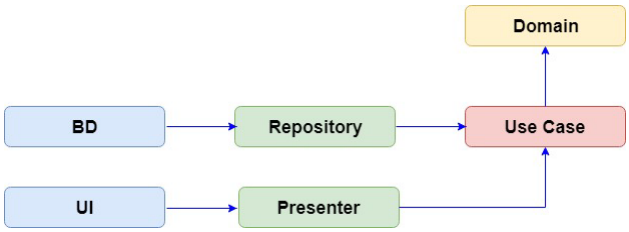

Figure 15. *General dependency diagram*

- The domain layer (Entities) is independent of all layers.
- The application layer (Usecase) depends on the domain layer and is independent of other layers. This layer prepares responses to user queries using entities.
- The adapters interface layer depends on the application(Usecase) layer. This layer transmits the user request and the necessary data to its processing at the Use Cases layer and retrieves the response and structure for the User Interface component.
- The user interface (UI) component of the "External technologies" layer depends on the interface layer adapters through the Presenter an n pass the user's request to it and retrieves the response.

- The database component of the ”External technologies” layer allows the Use Case layer to retrieve data through a Gateway(Repository) belonging to the Interface Adapter layer.

The following figure shows an example of a detailed dependency between layers of the clean architecture pattern in the case of our application. These dependencies link a set of classes belonging to the different layers. These classes realize the example of the functionality of displaying the list of users.

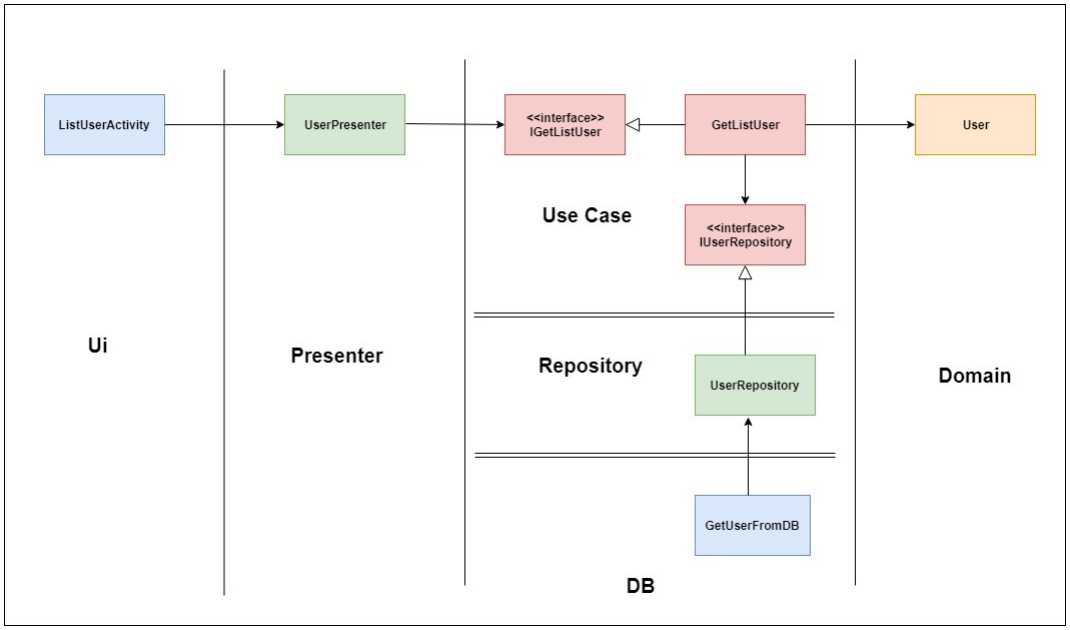

Figure 16. *Detailed dependency diagram in case of functionality print the list of users*

According to the figure, the Interface Adapter layer (UserPresenter and UserRepository) communicates (passing and retrieving data) with the Use layer boxes by implementing or instantiating its IGetListUser and IUser- Repository. These interfaces represent the ”ports” of the Use Cases layer. This example shows how we realized the principle of independence from the core of the application of the technologies used. No instance of the adapters interface layer or outer layer exists in the app’s core.

3.2 *Dynamic view of the application*

In this section, we will describe the internal dynamic of our application using sequence diagrams and activity diagrams

3.2.1 Detailed sequence diagram of the ”register” use case

The following figure illustrates the detailed sequence diagram of the use case ”Register”. This use case starts with opening a registration interface for the user. Then, the user enters his coordinates and confirms them. The system displays an error message if the email already exists otherwise it redirects the user to the home page.

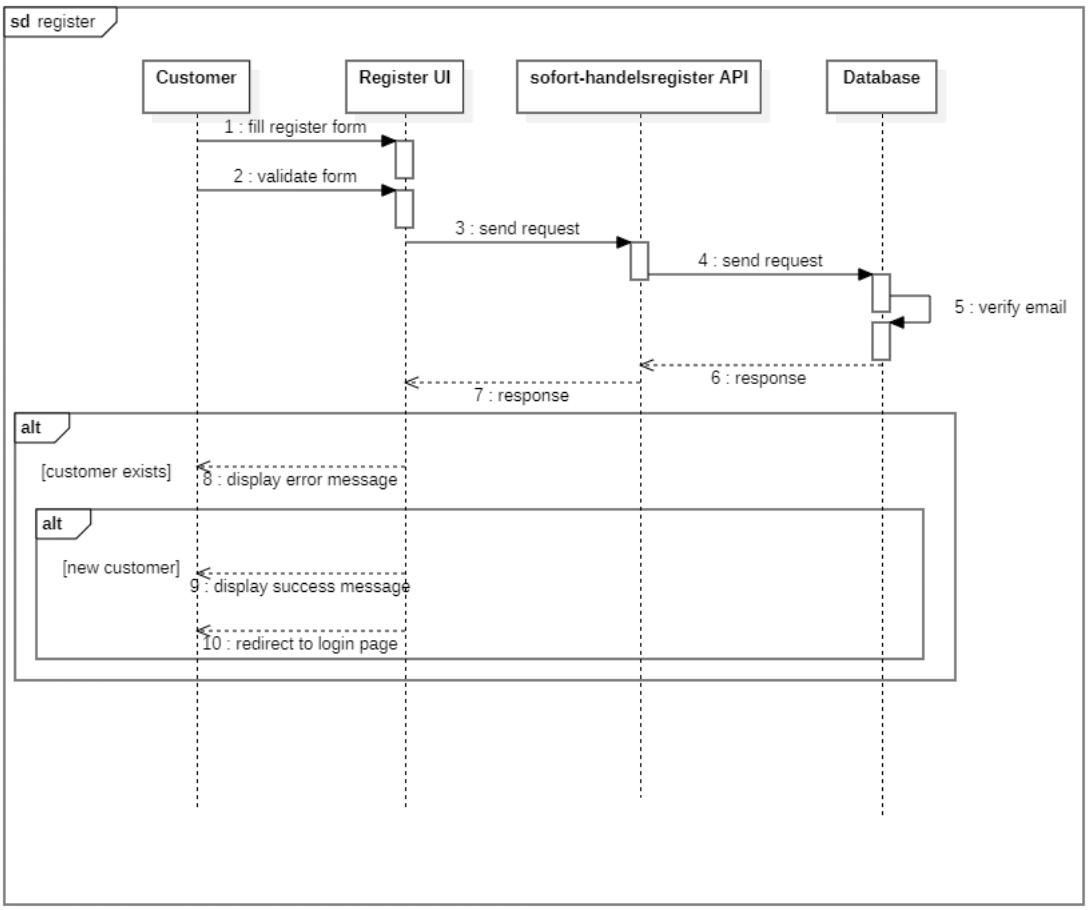

Figure 17. *Detailed sequence diagram of the ”register” use case*

3.2.2 Detailed sequence diagram of the ”login” use case

The following figure illustrates the detailed sequence diagram of the use case ”login”. This use case starts with opening a login interface for the user. Then, the user enters his email and password then confirms them. The system displays an error message if the email or the password is incorrect otherwise it redirects the user to the home page. the user may also choose

to log in using a social media account (Google, Facebook, LinkedIn, Xing), by taping the dedicated button to each social media which will redirect him to the interface provided by that specific social media API, the user will be redirected to the home interface upon successful social media login.

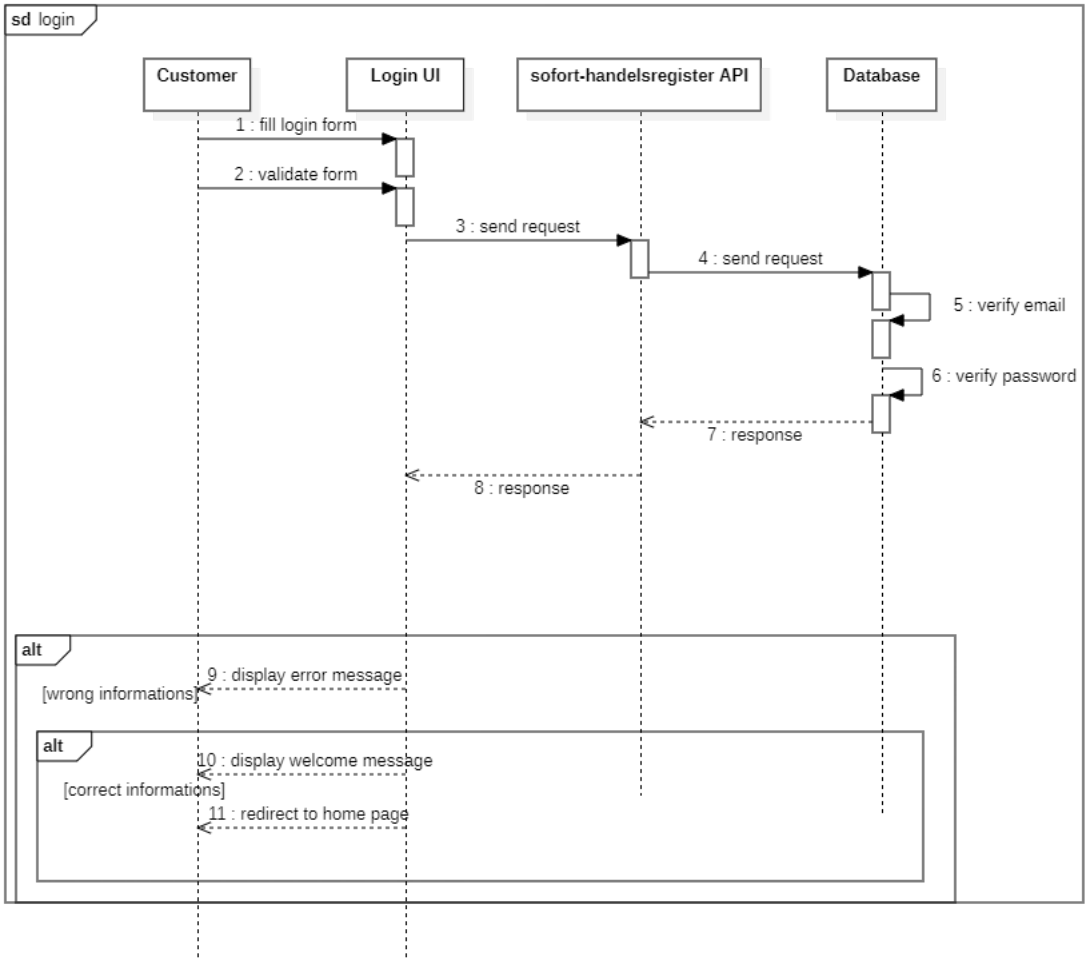

Figure 18. *Detailed sequence diagram of the "login" use case*

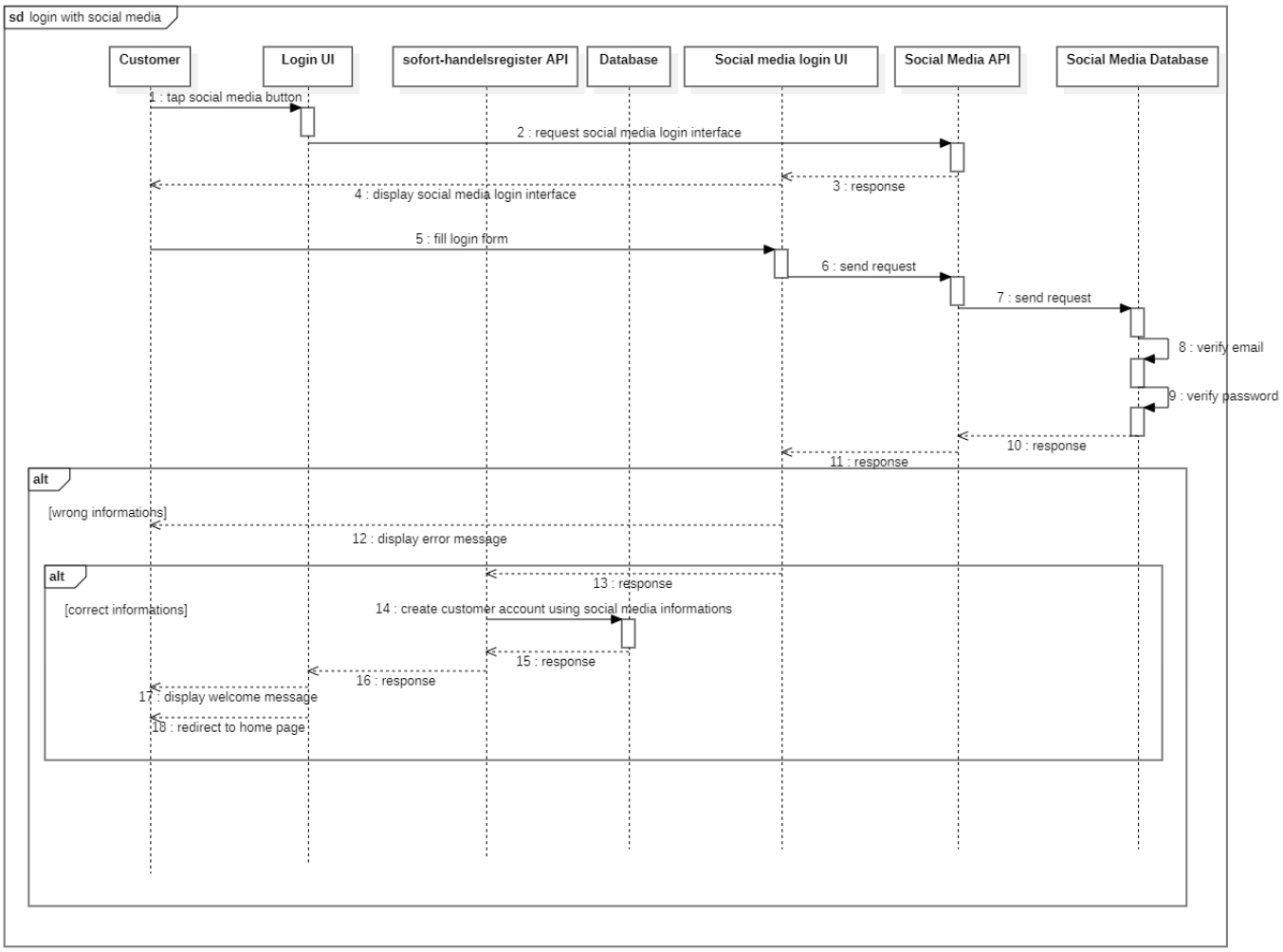

Figure 19. *Detailed sequence diagram of the ”login with social media” use case*

3.2.3 Detailed sequence diagram of the ”search company” use case

The following figure illustrates the detailed sequence diagram of the use case ”search company”. This use case starts with opening an interface for the company’s search for the user. Then the user enters the name of the wanted company’s name or the first letters of its name, he may also add some filter to the results using a different interface (by state, zip code, legal form, capital, branch), upon confirming his request the interface will prompt an interactive list of results where the user may tap one of them to access more detailed information about it.

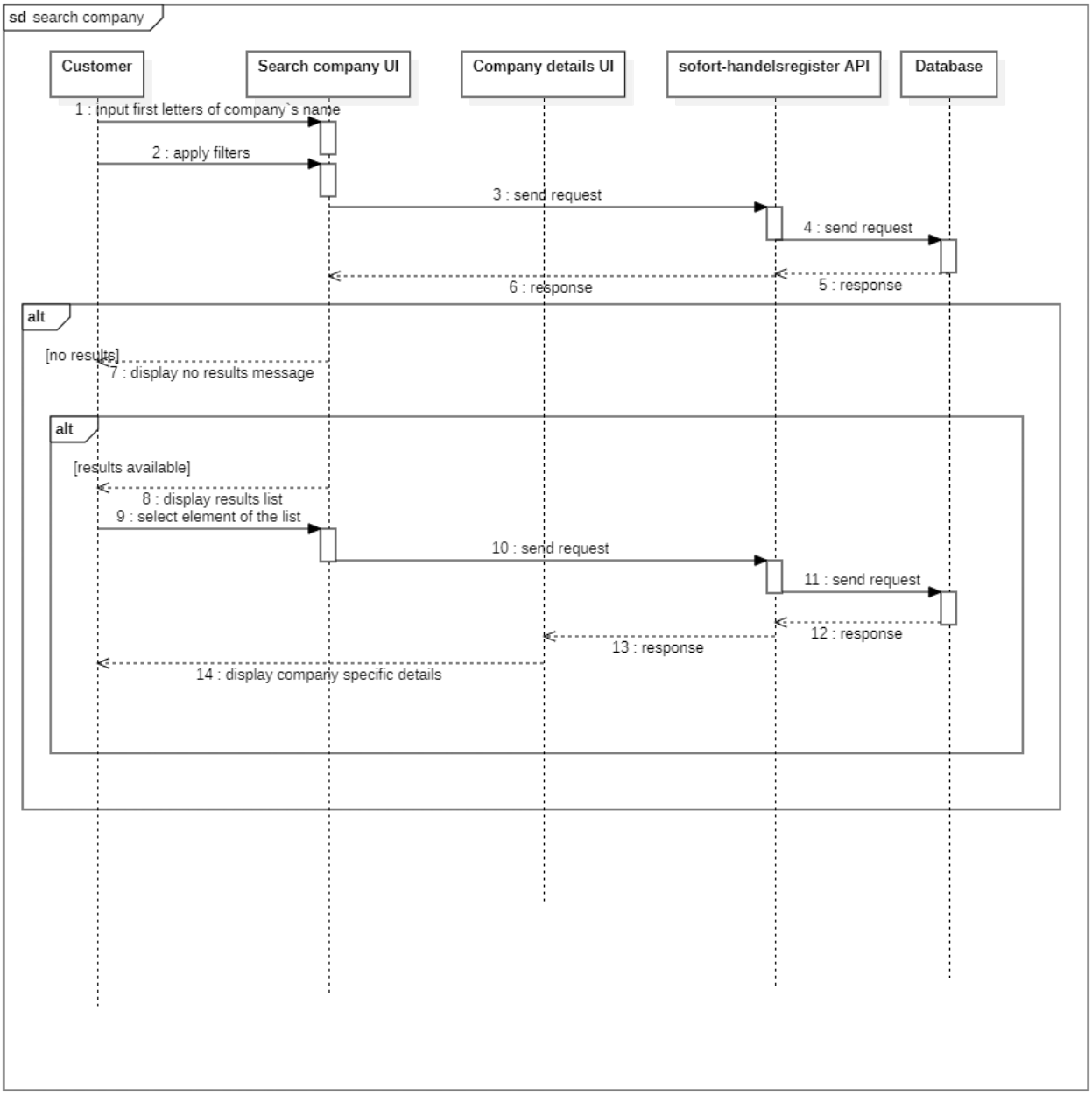

Figure 20. *Detailed sequence diagram of the ”search company” use case*

3.2.4 Detailed sequence diagram of the ”follow company” use case

The following figure illustrates the detailed sequence diagram of the use case ”follow company”. This use case starts with opening an interface containing specific company details for the user. Then the user may tap the follow button to add that company to his tracked companies list and start receiving notifications about it, these notifications may be deactivated later in the tracked companies interface.

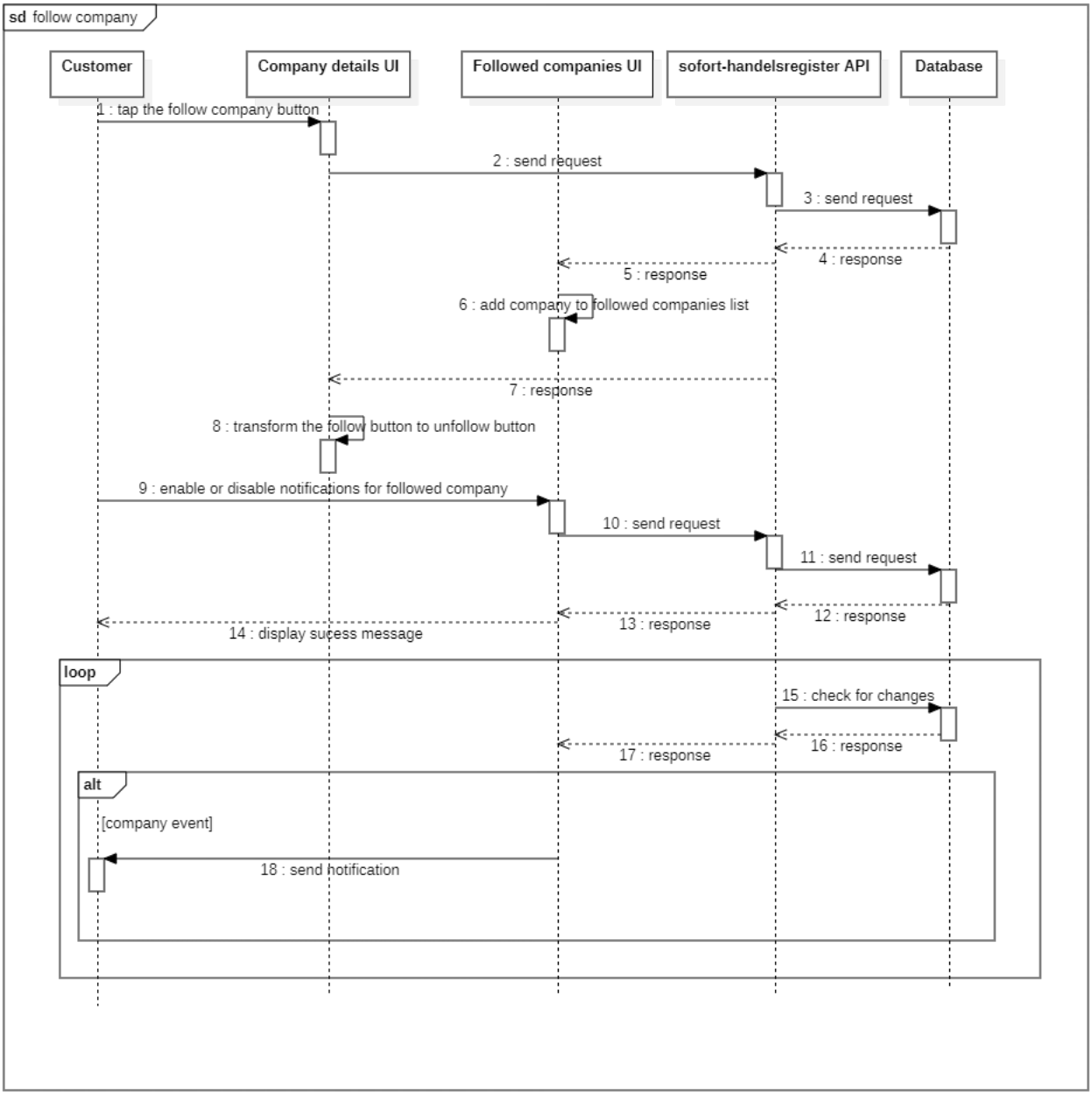

Figure 21. *Detailed sequence diagram of the ”follow company” use case*

3.2.5 Detailed sequence diagram of the ”purchase documents” use case

The following figure illustrates the detailed sequence diagram of the use case ”purchase documents”. This use case starts with opening an interface containing specific company details for the user. Then the user may scroll down to the documents section and add wanted documents to his cart, after that he may checkout and finalize his purchase process after filling in his credit

card details. As a result, the purchased documents are added to the user's inventory interface and he may download them.

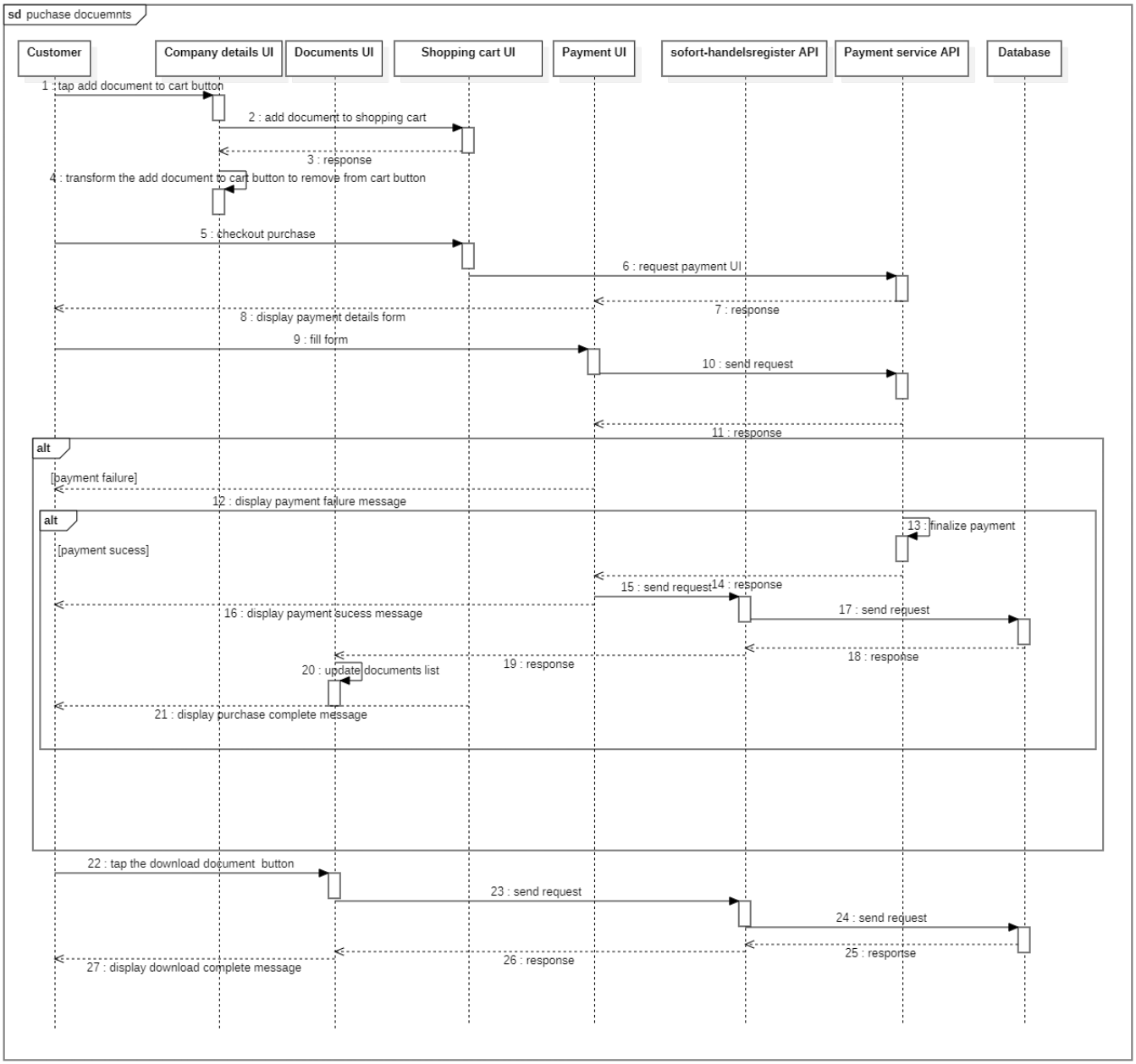

Figure 22. *Detailed sequence diagram of the "purchase documents" use case*

Conclusion

This chapter presents one of the most important phases of the process of development of a project: design subdivided into the overall design and detailed design.

We presented a Clean Architecture architecture pattern and we explained its application in our case. We have described the dynamic view of the system through a set of sequence diagrams and diagrams of activities.

The following chapter will deal with the realization of the project illustrated by screenshots of different interfaces. We will describe the environment of work and the tools used too.

Bibliography

[1] Get Started with the Salesforce Platform. [trailhead.salesforce.com].

[2] Discover Use Cases for the Platform. [trailhead.salesforce.com].

[3] Understand the Salesforce Architecture. [trailhead.salesforce.com].

[4] Oumayma LAZREG : Rapport PFE, Conception et développement d'une application mobile intelligente de e _ consulation et d'estimation des maladies. (ISSAT de Sousse), 2019.

[5] Model-View-Controller design pattern. [help.hcltechsw.com].

[6] Model–view–controller [en.wikipedia.org].

[7] What is MVC Design Pattern? [www.educba.com].