Tunisian republic
Ministry of Higher Education
and Scientific Research
University of Sousse

Higher Institute of Applied
Sciences and Technology
of Sousse

COMPUTER DEPARTMENT

# *END OF STUDIES PROJECT REPORT*

***In order to obtain:***
***National Diploma in Computer Engineering***
*Option: Software engineering-Software architecture*

# Conception and development of a Lightning community management application

**Developed by:**

**Aslene OUAZE**

**Defended on 10/07/2023 in front of the jury :**

| | | |
|---|---|---|
| **President :** | **Ms. Maha KHEMAJA** | **ISSAT Sousse** |
| **Examiner:** | **Mr. Houssemeddine CHTIOUI** | **ISSAT Sousse** |
| **Supervisor:** | **Dr. Farah BARIKA KTATA** | **ISSAT Sousse** |
| **Industrial supervisors** | **Ms. Rim BEN ABDESSLEM**<br>**Mr. Mandour AROUS** | **TECHLEAD** |

**Academic Year:** 2022 / 2023

Subject Code: FI-GL23-082

# Dedication

I would like to dedicate this modest work, the fruit of my efforts and my success:

**To my very dear father**, as a token of my affection and my gratitude for all the sacrifices made for my education
and to whom I owe my success.

**To my very dear mother**, who has never ceased to surround me with her affection, support, and love.

**To my dear sister**, who has always surrounded by her support and affection, thank you for always being there for me,
to help me get through the tough times.

**To my best friends**, who have proven that it is love that weave family ties and not blood, for the memories we have shared,
to whom I wish good luck and success.

To all my family and all those I love,

To all of you,
I dedicate this work.

# Thanks

At the end of this work, I would first like to thank
**GOD**, for empowering me to be who I am today,
**(Al-hamdullah)**.

Then, I would like to express my sincere thanks to all the people who have contributed,
directly or indirectly, to its success.
My special thanks go to:

**Ms. Farah BARIKA KTATA**, teacher at ISSAT Sousse,
for her supervision, assistance, and rigor, which helped me a lot in the
drafting of this report.

**Ms. Rim BEN ABDESSLEM** and **Mr. Mandour AROUS** Developers at TechLead,
**Ms. Noura ELABED**, head of TechLead,
for their guidance, advice, and support.
Thank you for the invaluable experience and the positive impact you have had on my career.

My sincere gratitude to all **ISSAT** faculty, for the training they gave us.

I end these thanks by warmly greeting the members of the jury for
the honor they do me by agreeing to judge 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, HTML, 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 clients' needs and helps them use and leverage the capabilities of Salesforce.


*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 for suggesting 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 1.** *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.

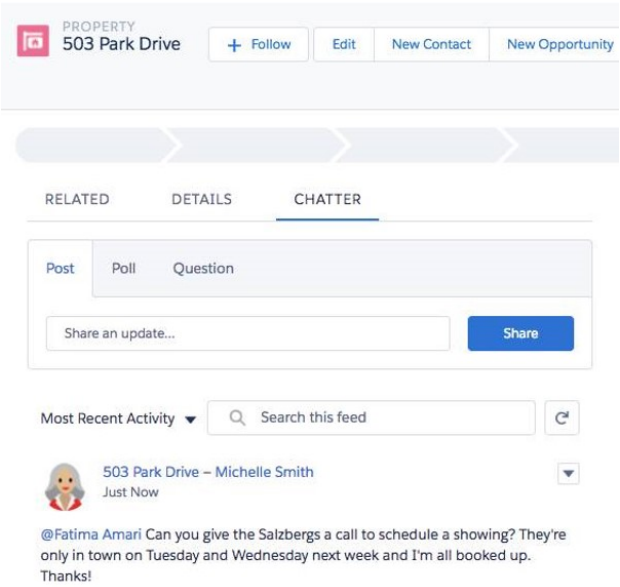


**Figure 2.** *Chatter extension in Salesforce, Source: [**2**]*

Here are a few use cases for different departments:

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

**Figure 3.** *Usecases for Salesforce, Source: [**2**]*

Salesforce is a cloud company. Everything to offer resides in the trusted, multitenant cloud. 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. 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. Everything is integrated. The platform technologies like predictive analytics and the development framework are built into everything to offer and everything to build.[**3**]

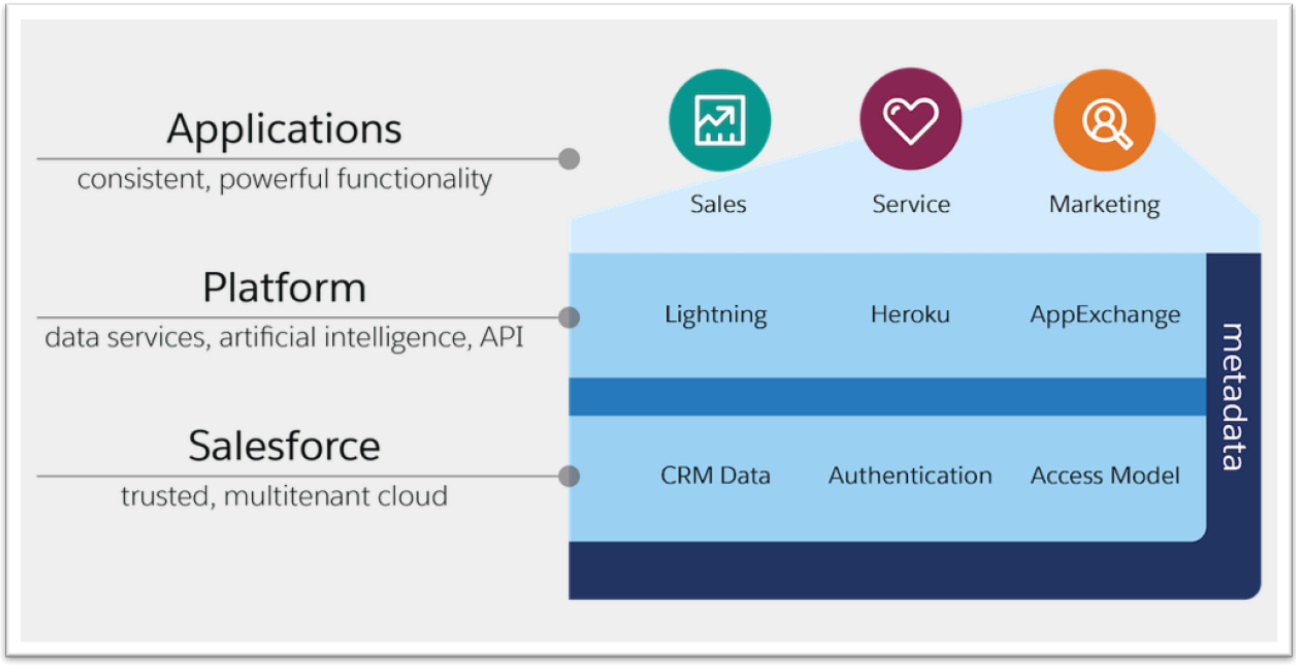


**Figure 4.** *Salesforce's architecture, Source: [**3**]*

### 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, making it difficult and time-consuming.

This problem becomes even more apparent when dealing with the base Salesforce interface where there are so many places to go when trying to manage even the simplest tasks regarding users and their communities for example:

- If the administrator wants to manage all his users he will have to access the Salesforce setup home menu but when accessing his community users he will have to access the community administration interface that takes so much time to load.
- If the administrator wants to track members' connection history, it cannot be done without using third-party extensions or applications.
- If the administrator wants to add multiple members at once to his community, it cannot be done without wasting time navigating between interfaces.

### 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 maintenance.

**Technical objectives:**

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

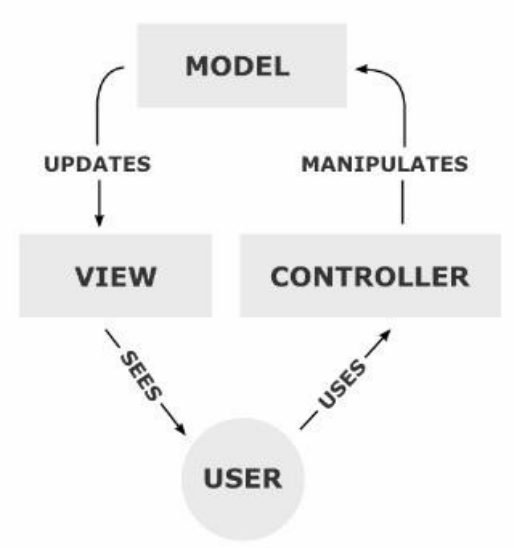


**Figure 5.** *MVC Architecture, Source: [**6**]*

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 table 1, we propose a solution that solves their drawbacks.

While searching for existing solutions for our main problem we came across 3 of the most popular applications and Salesforce extensions specializing in users and community management:

- **HubSpot:** HubSpot is a popular marketing, sales, and customer service platform that integrates with Salesforce.

*HubSpot Logo*

HubSpot uses Salesforce’s user management features to allow organizations to manage their teams and control access to different parts of the application.

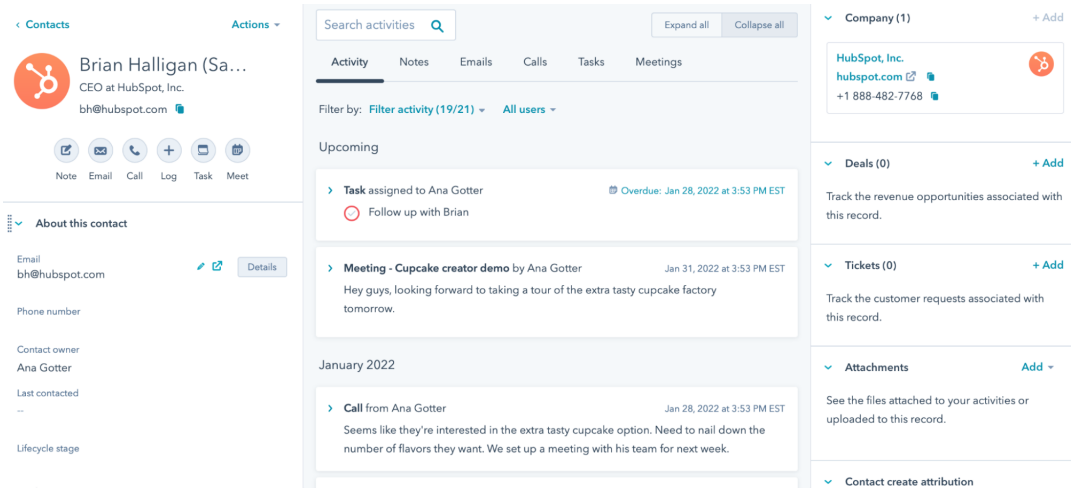


**Figure 6.** *HubSpot user interface*

- **ZenDesk:** ZenDesk is a customer support platform that integrates with Salesforce.

*ZenDesk Logo*

ZenDesk uses Salesforce's user management features to allow organizations to manage their support teams and control access to different parts of the application.

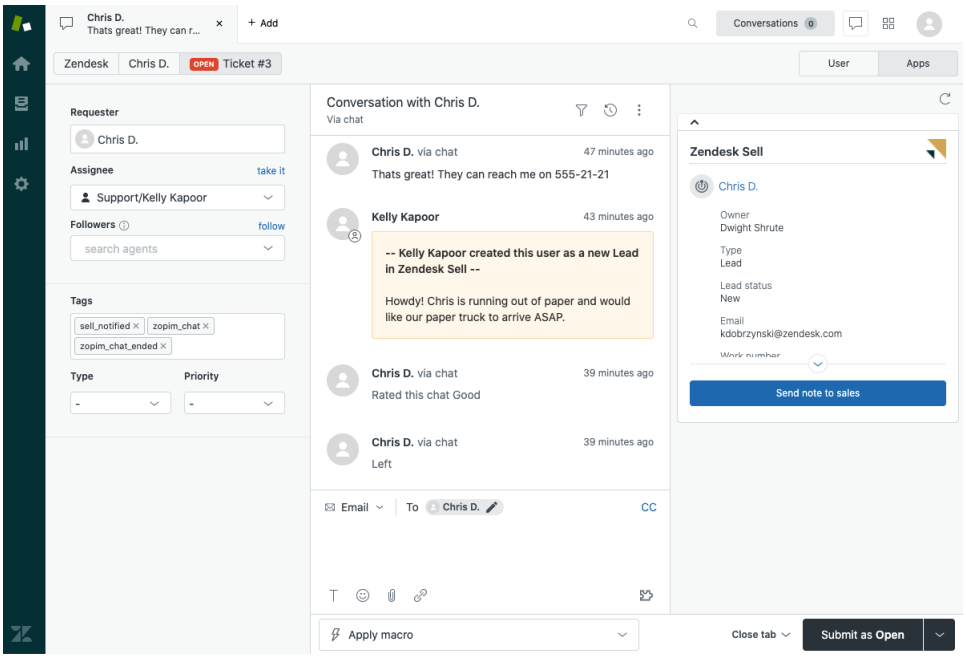


**Figure 7.** *ZenDesk user interface*

- **Conga:** Conga is a document generation and reporting platform that integrates with Salesforce.

*Conga Logo*

Conga uses Salesforce's user management features to allow organizations to manage their teams and control access to different parts of the application.

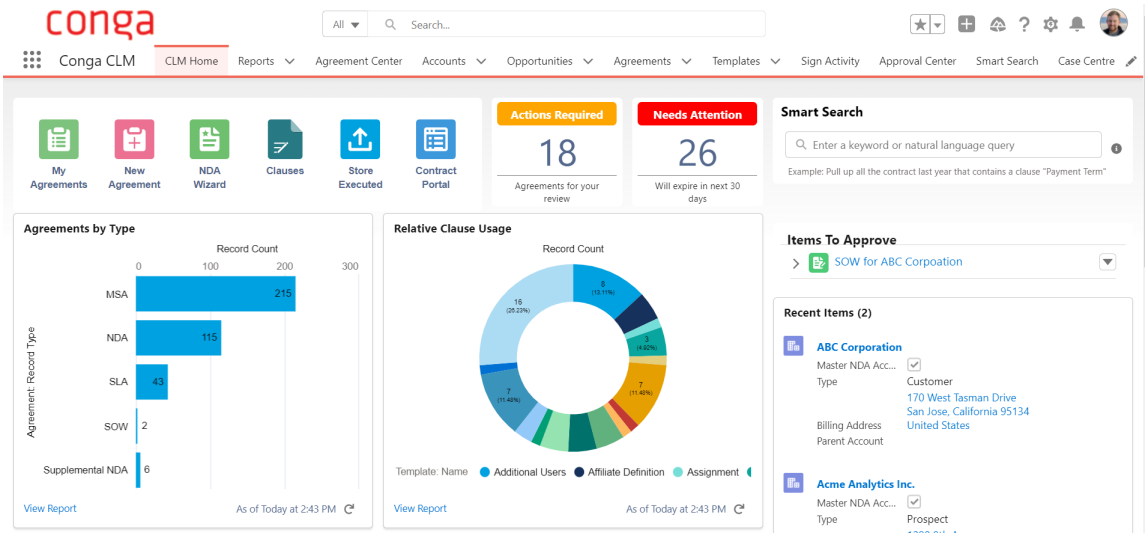


**Figure 8.** *Conga user interface*

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

1. **Iterative Process:** Incremental development follows an iterative process where the development is divided into small, manageable phases. Each phase focuses on delivering a specific set of features or functionalities. This approach allows for frequent feedback and collaboration with stakeholders, ensuring that their evolving requirements are met throughout the development process.
2. **Faster Time-to-Market:** By breaking down the development into smaller increments, we can release functional versions of the application more quickly. This enables organizations to deploy and start using essential features earlier, gaining a competitive advantage and generating value sooner.

3. **Adaptability to Changing Requirements:** The business landscape is dynamic, and requirements can change rapidly. Incremental development allows for flexibility and adaptability, as it enables us to respond to changing needs and incorporate new features or adjustments easily. It reduces the risk of building a solution that doesn't align with evolving business requirements.
4. **Risk Mitigation:** Incremental development minimizes risk by addressing potential issues early in the development cycle. With each iteration, we can identify and rectify any problems, ensuring that the final product meets quality standards. It also allows for better risk management and course correction during the development process.
5. **Scalability and Maintainability:** Incremental development makes it easier to manage the scalability and maintainability of Salesforce applications. By building and releasing smaller increments, it becomes simpler to identify and resolve scalability issues as the application evolves. Additionally, the iterative nature of development allows for ongoing improvements and enhancements, ensuring the application remains maintainable and adaptable in the long term.

### 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 4 months. The figure 9 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.

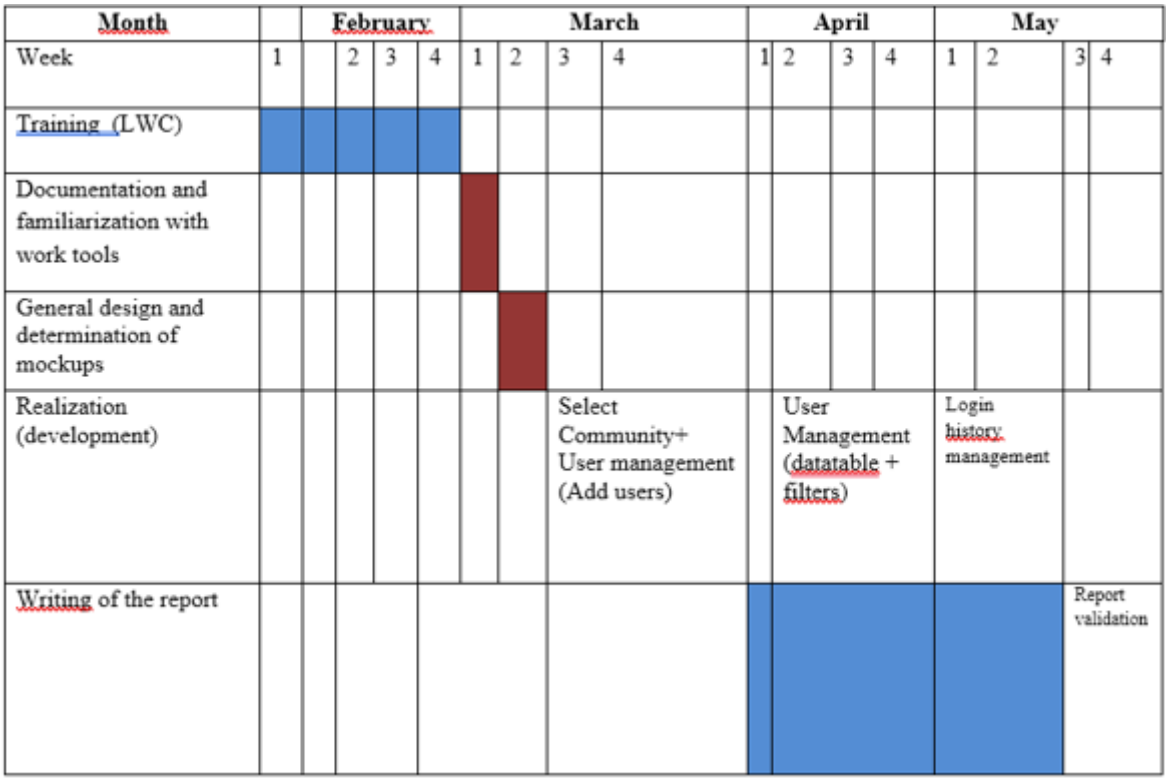


**Figure 9.** *Gantt diagram*

## *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 showing each user’s logins 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 to a specific community at once.

### 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 figure 10 illustrates the global use case diagram of our application

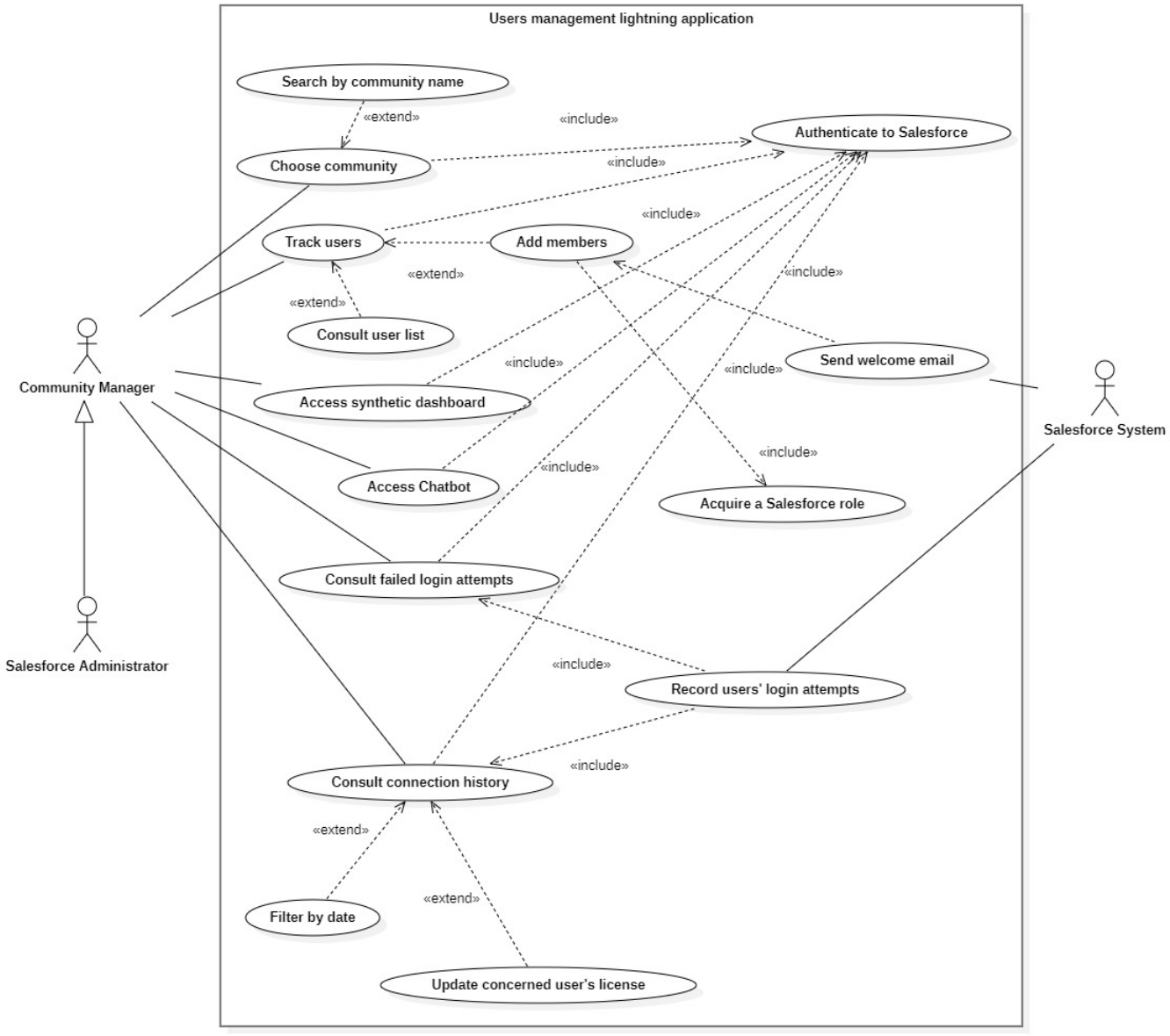


**Figure 10.** *Global use case diagram*

Our application will provide both the community manager and the system administrator ,as long as they are authenticated to Salesforce, the ability to:

- Search for and select the community that they want to manage.
- Track community members by either consult the full user list or add new members, as long as they own a Salesforce role, where the Salesforce system will send an automatic welcome email to the new community member.

- Access synthetic dashboard illustrating the KPI (Key Performance Indicators) of the community.
- Access a smart Chatbot that will provide useful Salesforce information to the user.
- Consult and manage failed login attempts committed by community members and previously recorded and stored by the Salesforce system.
- Consult and filter members' connection history previously recorded and stored by the Salesforce system, as well as update their Salesforce license to match their level of activity.

### 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 figure 11 shows the Consult user list use case diagram.

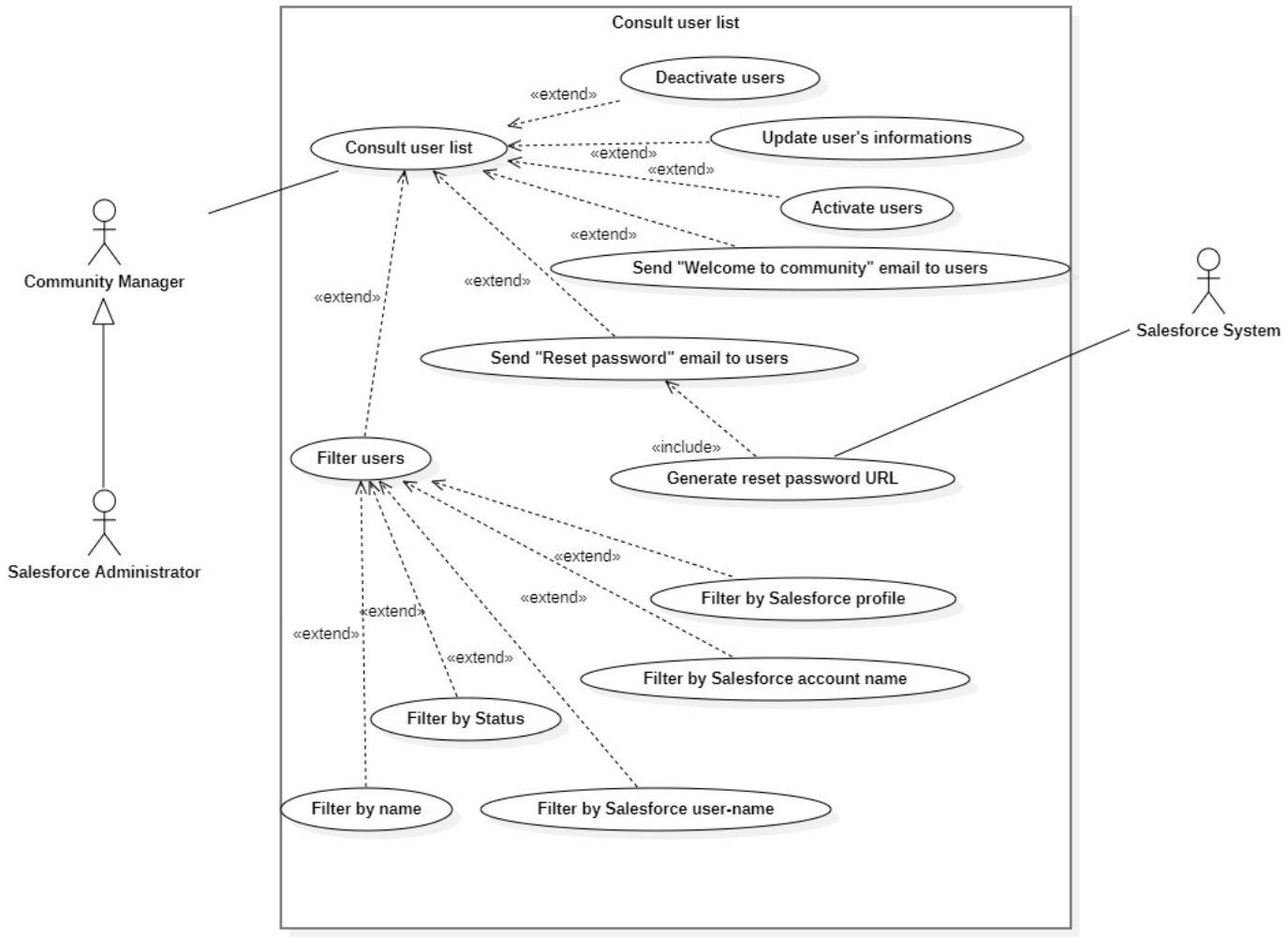


**Figure 11.** *Consult user list use case diagram*

The table 2 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<br>- 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<br>2. User selects one or many community members<br>3. User clicks the ”activate/deactivate” button for selected members<br>4. The system prompts the success of the operation<br>5. The user clicks the ”Send welcome email” button for the selected members<br>6. The system prompts the success of the operation<br>7. The user clicks the ”Send reset password” button for the selected members<br>8. The system prompts the success of the operation<br>9. The user clicks the ”show details” button for one member<br>10. The member information interface pops up<br>11. User updates the selected member information and complies<br>12. System prompts the success of the operation |
| Alternative scenario | 1. License limit exceeded when activating members: The system sends an error message<br>2. Sending a welcome email to a deactivated member: The system sends an error message<br>3. Changing member username to an existing one: The system sends an error message |
| Non-functional constraints | 1. The interface must be ergonomic<br>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 system administrator can add one or multiple members to their respective communities at once.

The table 3 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<br>- User should be the owner of at least one community<br>- 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<br>2. The user fills the displayed form with information about the new member<br>3. The user clicks the " add another member" button<br>4. A new empty form, identical to the previous one, is displayed<br>5. The user fills the new form with information about the next new member<br>6. The user clicks the "clone member" button<br>7. A new form containing the same values as the previous form is displayed<br>8. The user clicks the "delete form" button<br>9. The selected form is deleted from the screen<br>10. The user clicks the "submit" button<br>11. System prompts the success of the operations |
| Alternative scenario | 1. Empty fields: The system sends an error message:<br>You must complete all the required fields<br>2. The email entered is not valid: The system sends an error message describing the validation conditions for this field<br>3. Duplicate Salesforce user-name: The system sends an error message |
| Non-functional constraints | 1. The interface must be ergonomic<br>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 figure 12 shows the ”Consult synthetic dashboard” use case diagram.

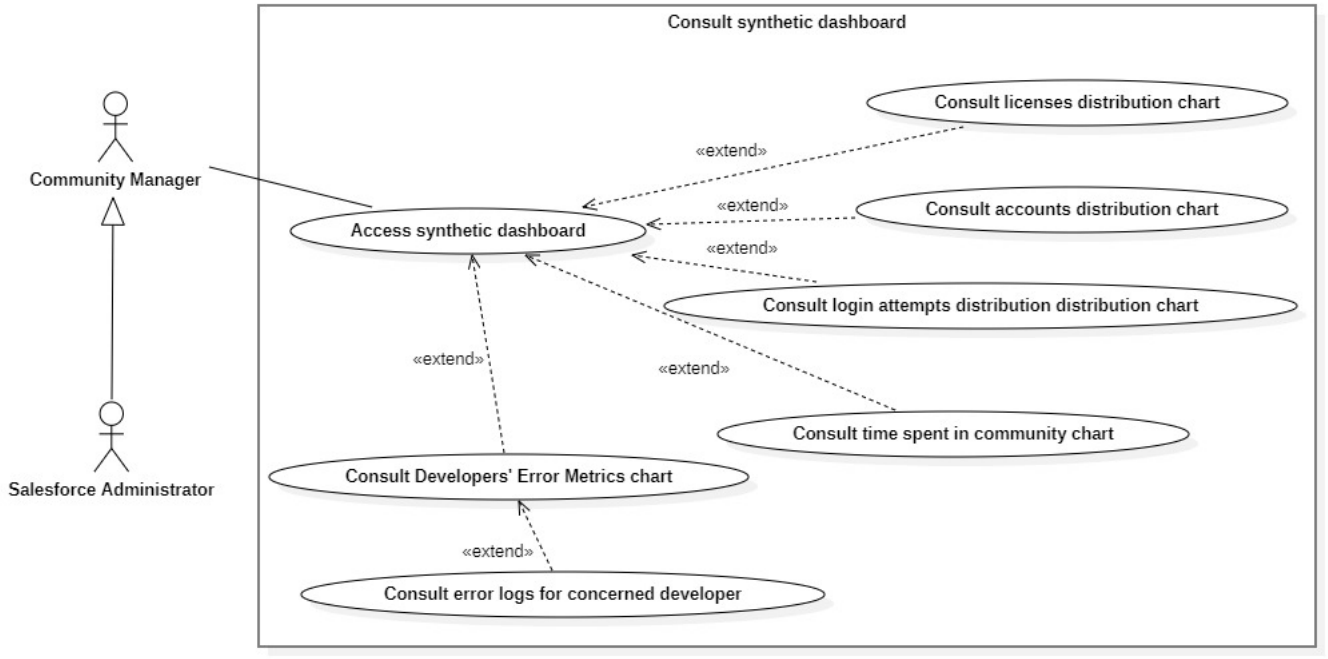


**Figure 12.** *Consult synthetic dashboard use case diagram*

The table 4 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<br>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<br>- User should be the owner of at least one community |
| Post-condition | - System administrator is well informed<br>about his respective community's KPIs. |
| Normal scenario | 1. User accesses the dashboard interface<br>2. Multiple KPI charts pop up<br>3. Users may click a bar from the developers' Error Metrics Chart<br>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.

The figure 13 shows the "consult failed login attempts" use case diagrams.

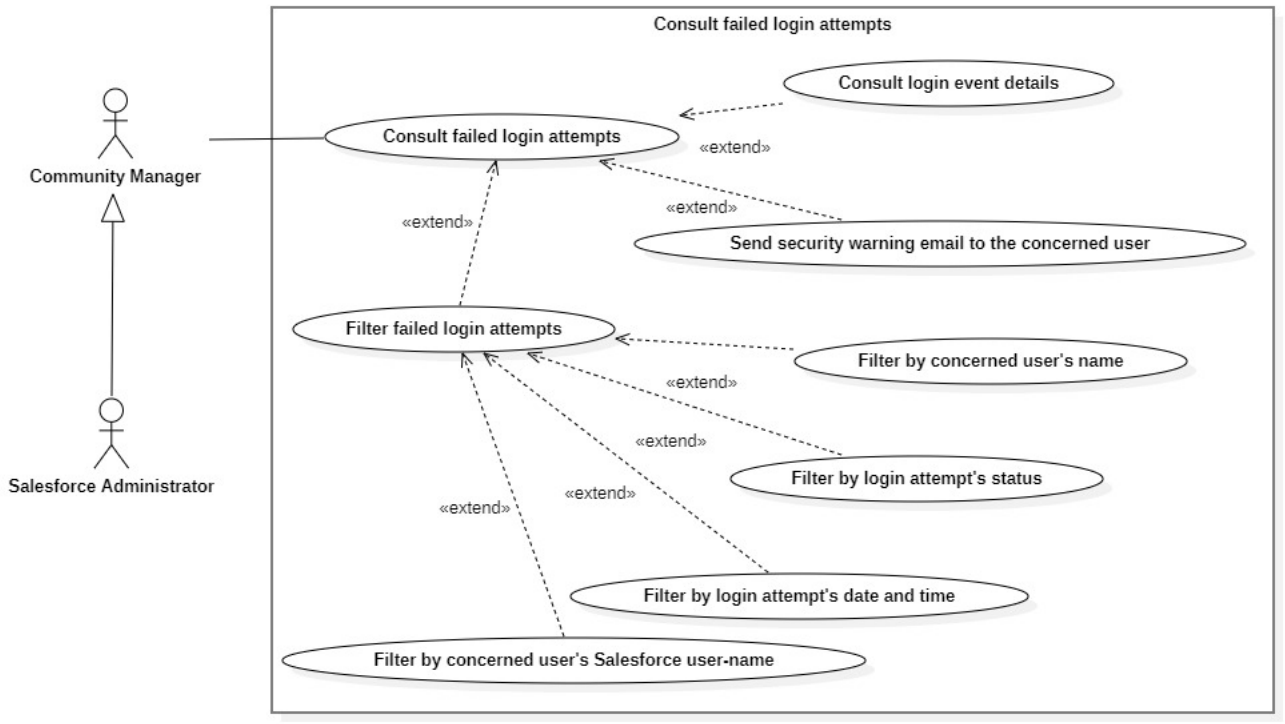


**Figure 13.** *Consult failed login attempts use case diagram*

The table 5 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<br>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<br>- 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<br>2. The user clicks the ”show details” button for a specific login event<br>3. The ”show details” interface for the selected login event pops up<br>4. The user clicks the ”send security warning email” button for the specific login event<br>5. The system prompts the success of the operation<br>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<br>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

In our application, the community manager and the system administrator can access a smart Chatbot interface to benefit from a Salesforce digital assistant that will provide information about Salesforce and its features as well as provide information about the current community. The figure 14 illustrates the use case diagram ”Access Chatbot”

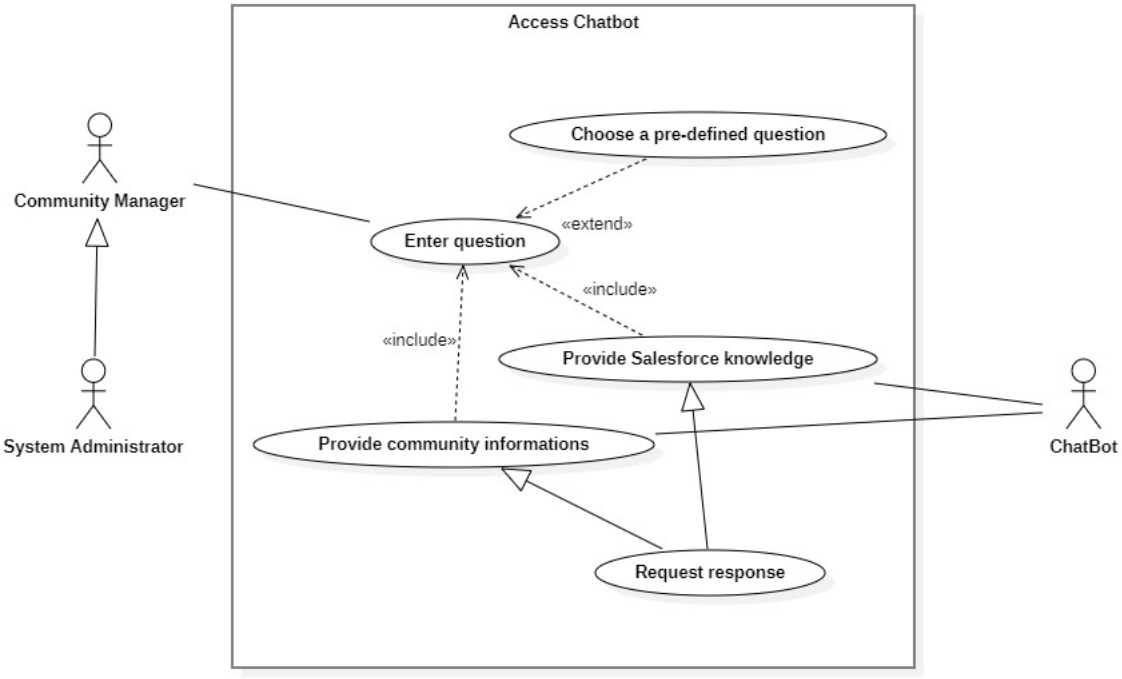


**Figure 14.** *Access Chatbot use case diagram*

The table 6 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<br>while providing informations about the current community |
| Actors | System administrator, Community manager, Chatbot |
| **Description of sequences** | |
| Pre-condition | - User should be authenticated to Salesforce<br>- User should be the owner of at least one community |
| Post-condition | - User question is answered<br>and the response is displayed on the screen. |
| Normal scenario | 1. User accesses the Chatbot interface<br>2. User types a question or selects a predefined question.<br>3. The chatbot is launched upon receipt of a request<br>4. The chatbot analyzes the request<br>5. The chatbot prepares the response<br>6. The chatbot sends the response |
| Alternative scenario | 1. An error message is displayed if the question<br>is not recognized by the system. |
| Non-functional constraints | 1. The interface must be ergonomic<br>2. Error messages should be understandable and clear<br>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.
- **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.
- **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 figure 15 explains furthermore the role and the idea behind each component of the MVC pattern:

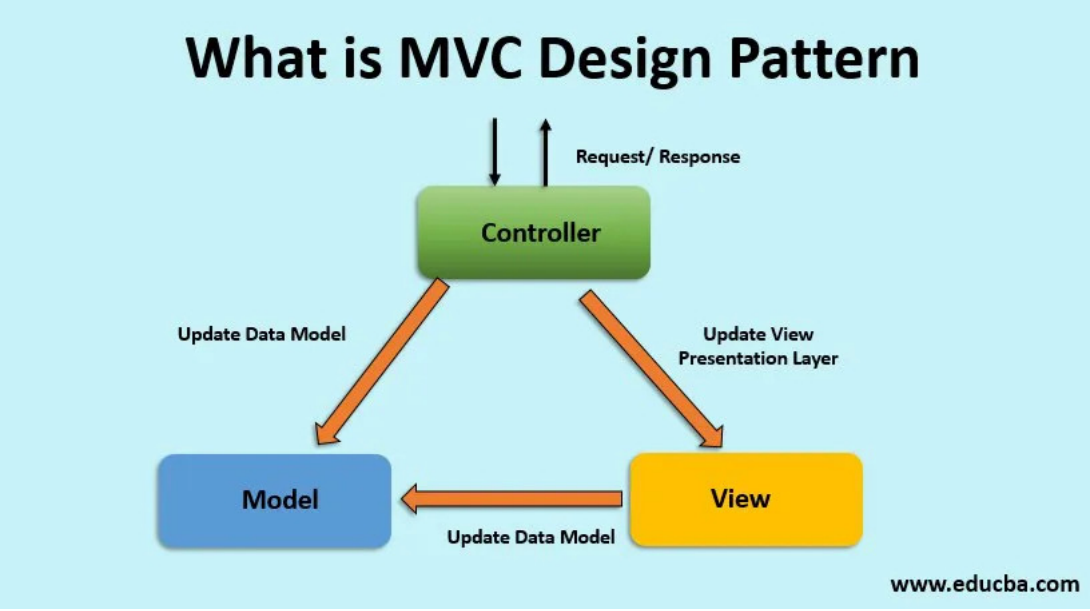


**Figure 15.** *Understanding MVC Design Pattern, Source: [**7**]*

### 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 figure 16 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.

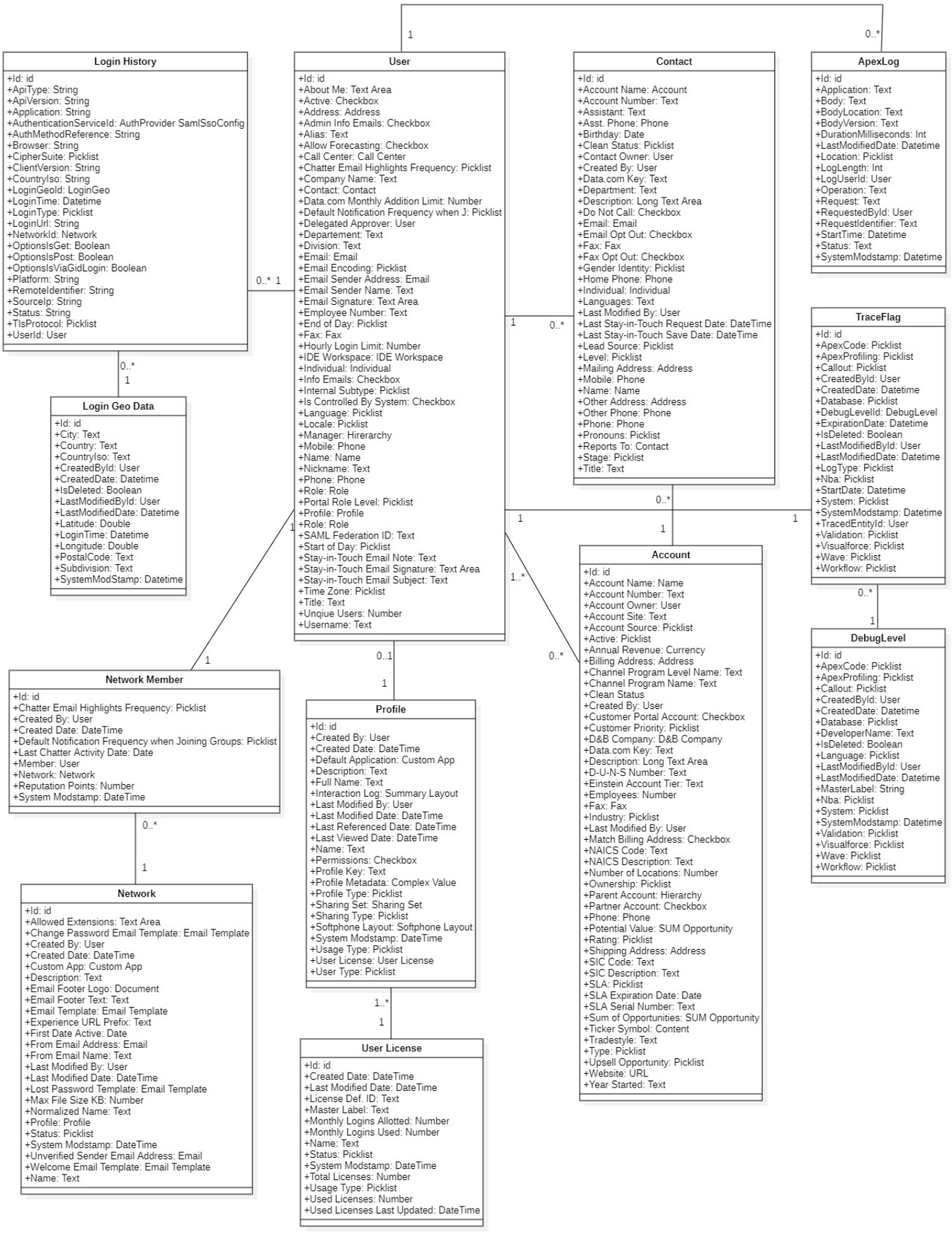


**Figure 16.** *Class diagram representing the "Model" component*

Description of classes representing our Model:

- **User:**

  This class represents our application's Salesforce user, who is the main focus of the administrator and community managers main focus 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 our application's administrator or the community manager. 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 monitored 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 run by the tracked user within the community, and it will be used in our application to provide said logs to 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 the 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 ”Model” component are related to the Salesforce object infrastructure, and employable in various types of features in our application. Thus, we made the principle of independence between the business rules and the application rules.

## 3.2 *Dynamic view of the application*

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

### 3.2.1 Detailed sequence diagram of the ”Choose community” use case

The figure 17 illustrates the detailed sequence diagram of the use case ”Choose community”. This use case starts with opening a community list interface for the user. Then, the user may enter his searched community name or a few letters of it. The system displays an error message if there are no communities found with the entered name otherwise it displays found communities. Then the user may choose to access one of the displayed communities, then the system redirects the user to that community dashboard.

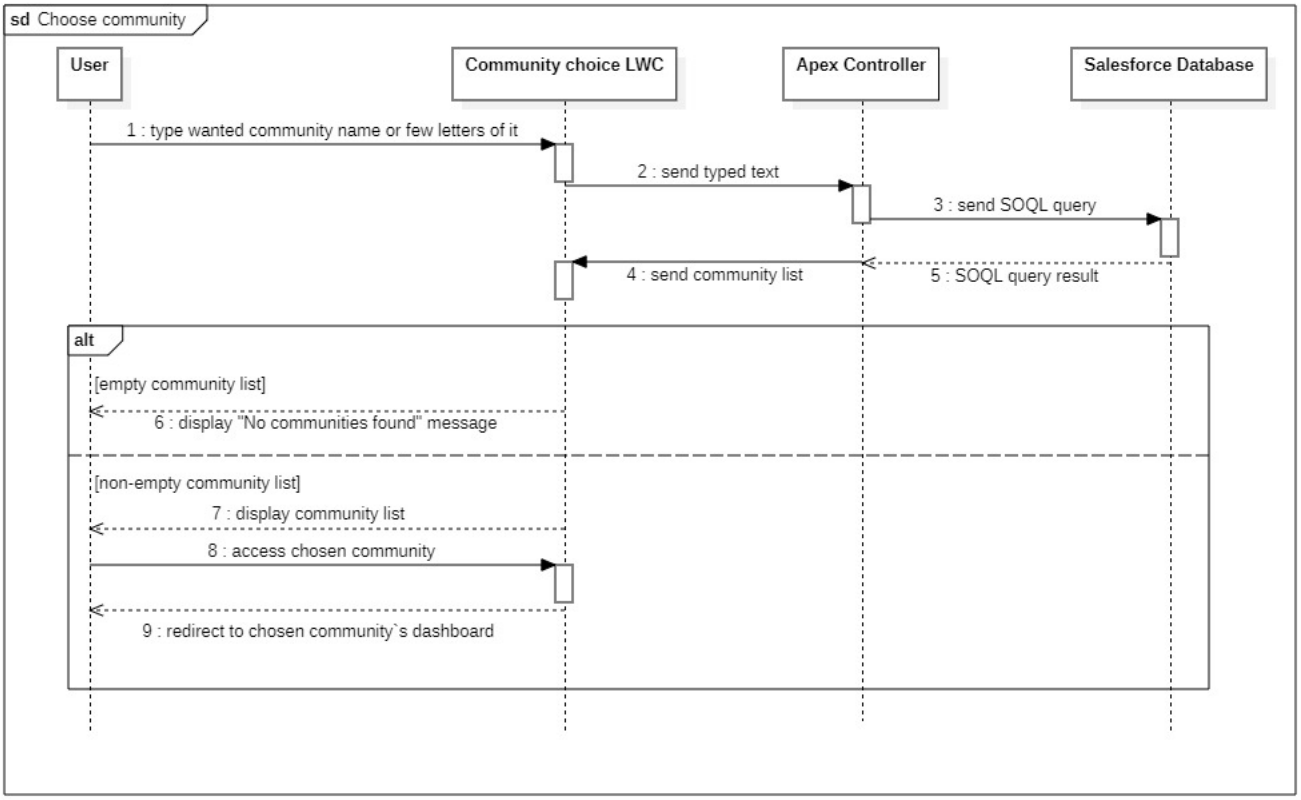


**Figure 17.** *Detailed sequence diagram of the ”Choose community” use case*

### 3.2.2 Detailed sequence diagram of the ”Access synthetic dashboard” use case

The figure 18 illustrates the detailed sequence diagram of the use case ”Access synthetic dashboard”. This use case starts with opening a dashboard interface specific to the community chosen by the user. The user may consult the account distribution chart, licenses distribution chart, login attempts distribution chart, time spent in community chart, and developers’ error metrics chart where he will be able to select a specific user to monitor their error logs.

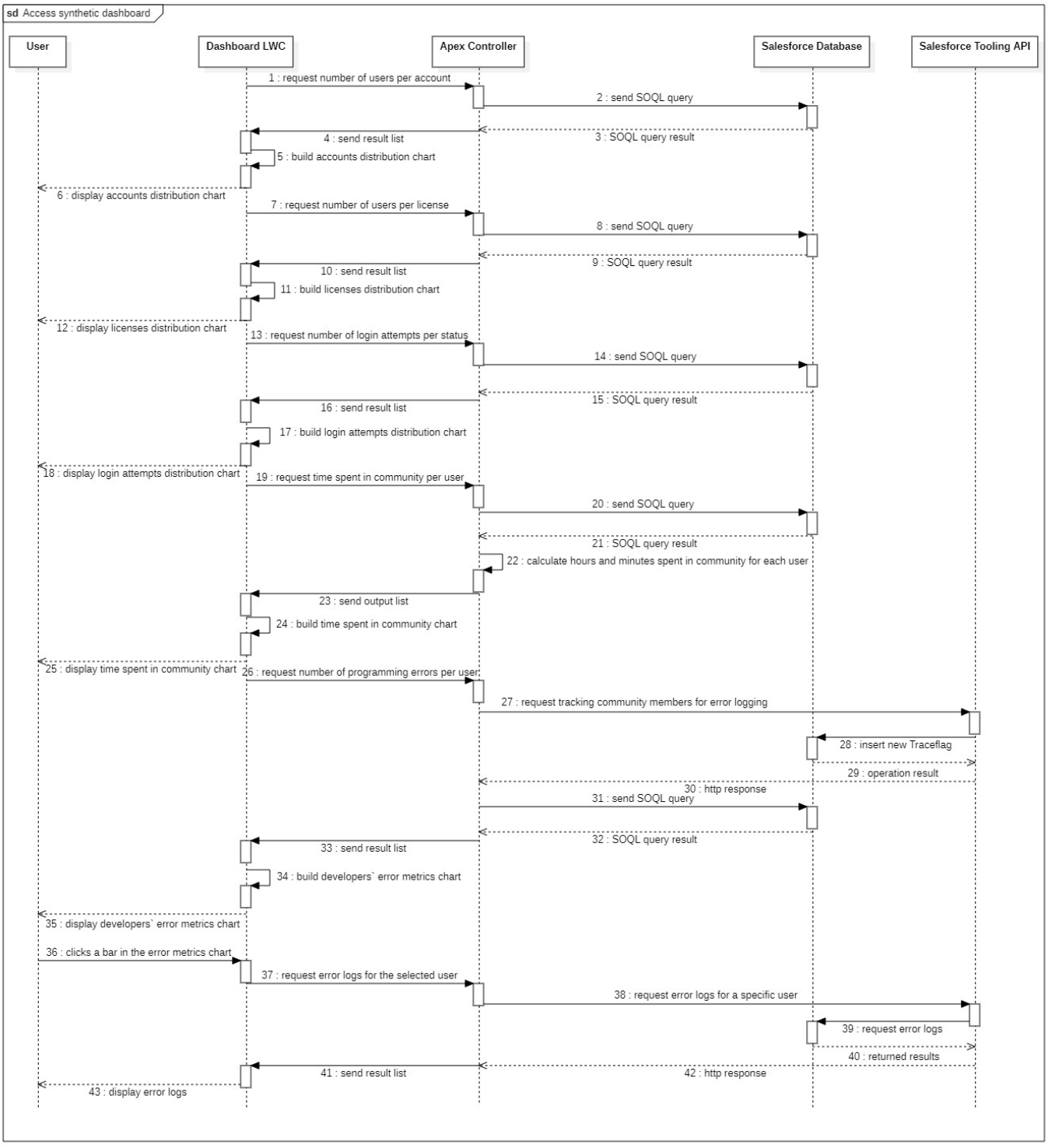


**Figure 18.** *Detailed sequence diagram of the ”Access synthetic dashboard” use case*

### 3.2.3 Detailed sequence diagram of the ”Consult users List” use case

The figure 19 illustrates the detailed sequence diagram of the use case ”Consult users List”. This use case starts with opening an interface displaying a data table containing information about the user’s community members. Then the user may search for a specific user using his Salesforce user name, full name, or his email address, he may also filter given results by account name, user status (active or not active), profile name, or account name. For the displayed members the user may choose to send welcome emails, send reset password emails to selected members as well as update their current status and their information.

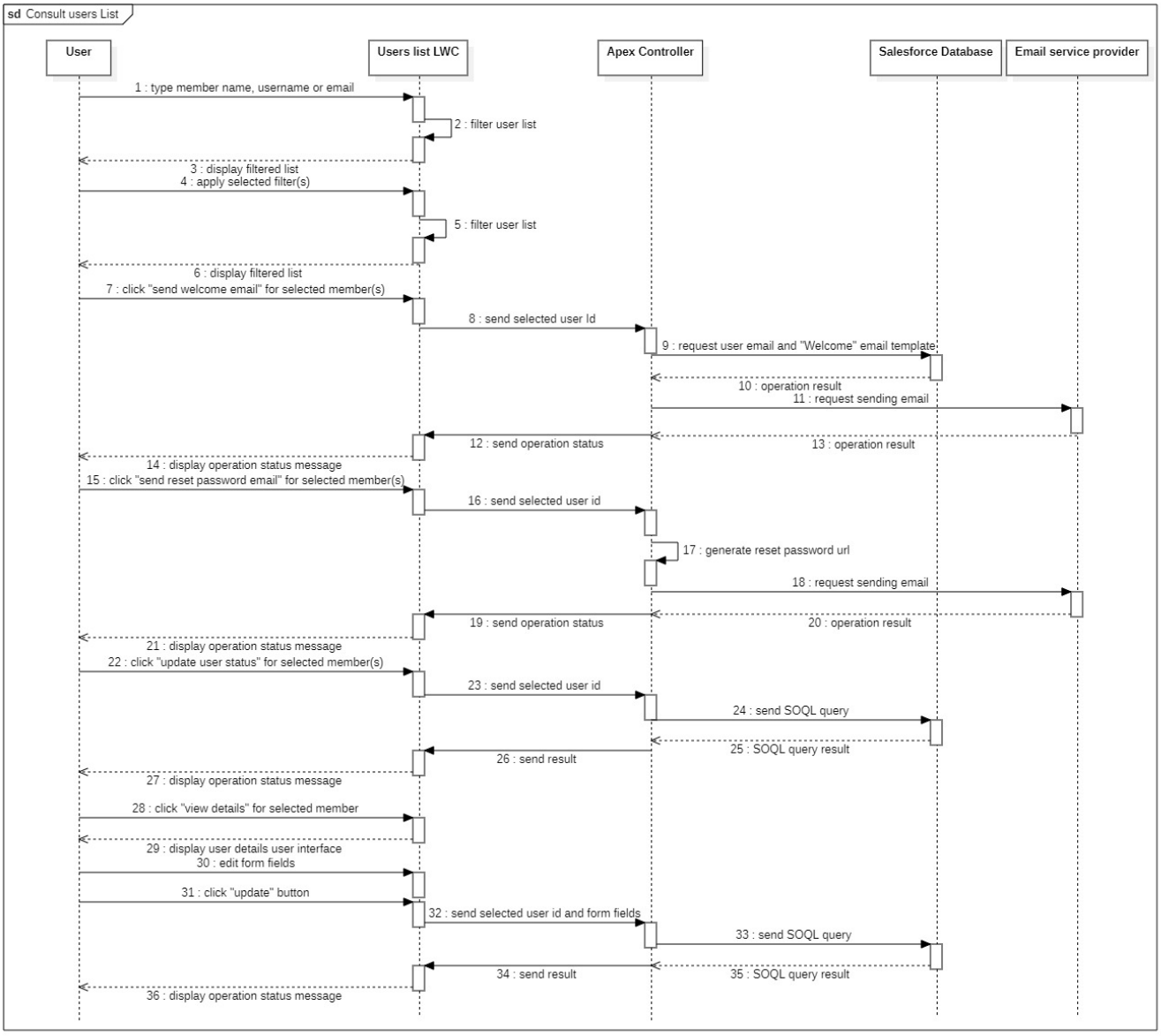


**Figure 19.** *Detailed sequence diagram of the ”Consult users List” use case*

### 3.2.4 Detailed sequence diagram of the ”Add members” use case

The figure 20 illustrates the detailed sequence diagram of the use case ”Add members”. This use case starts with opening an interface displaying an empty form containing input fields representing required and optional information about the new community member, the user may choose to add another empty form, add another form containing the same values as the previous one or delete the last form. Afterward, the user may choose to submit all forms where the system will prompt an error message in case one or more required fields are missing, the type user name already exists or the license limits are exceeded, otherwise, the system will prompt the success of the insert operation and will refresh the users' list.

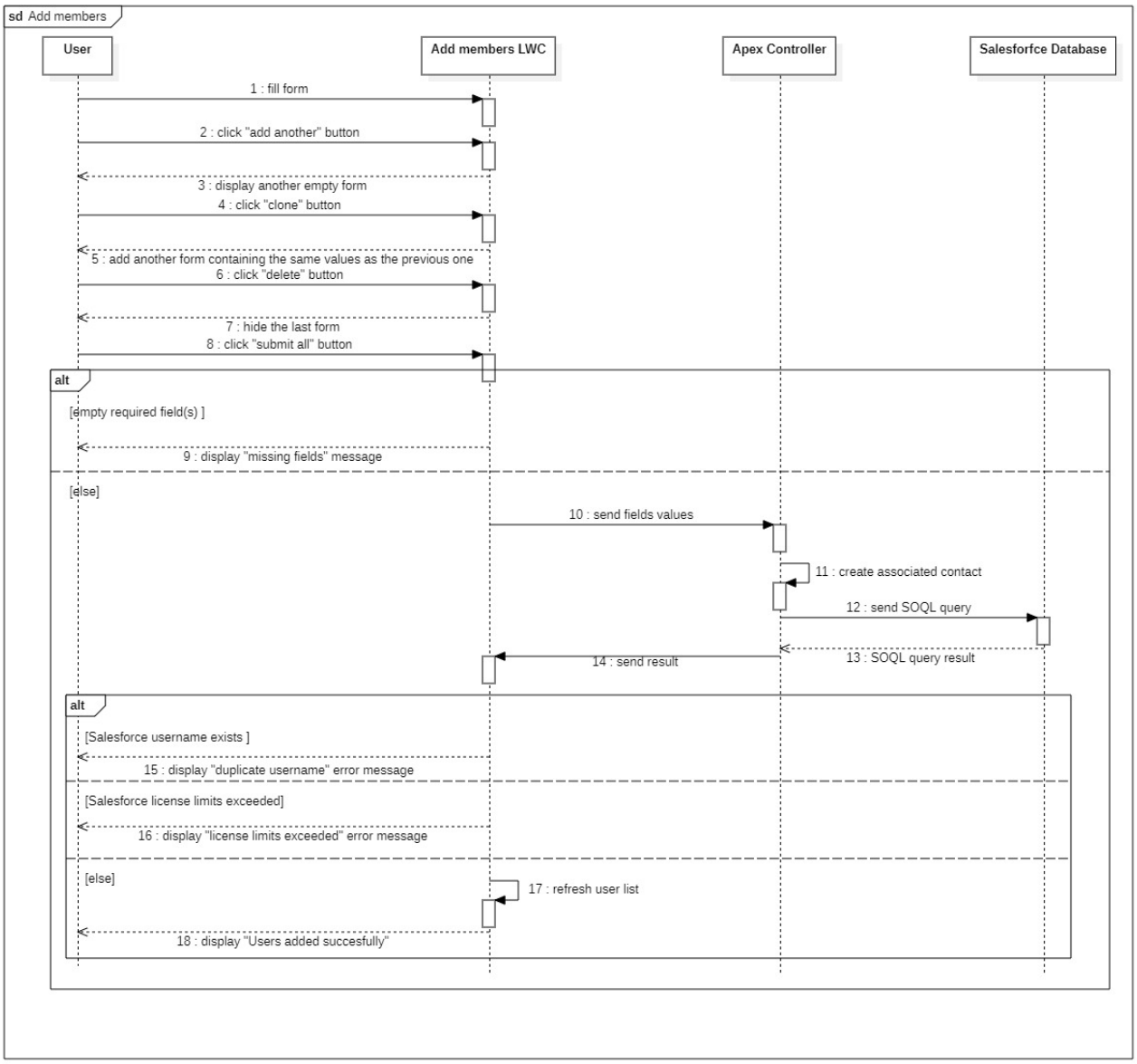


**Figure 20.** *Detailed sequence diagram of the ”Add members” use case*

### 3.2.5 Detailed sequence diagram of the ”Consult failed login attempts” use case

The figure 21 illustrates the detailed sequence diagram of the use case ”Consult failed login attempts”. This use case starts with opening an interface displaying a data table containing information about the failed login attempts committed by the user’s community members. Then the user may search for a specific login event using the member’s Salesforce user name or his full name, he may also filter given results by login event status (invalid password, no community

access, etc) or by its date and time. For the displayed events the user may choose to send welcome security warning email to the concerned member or consult details about the full login event.

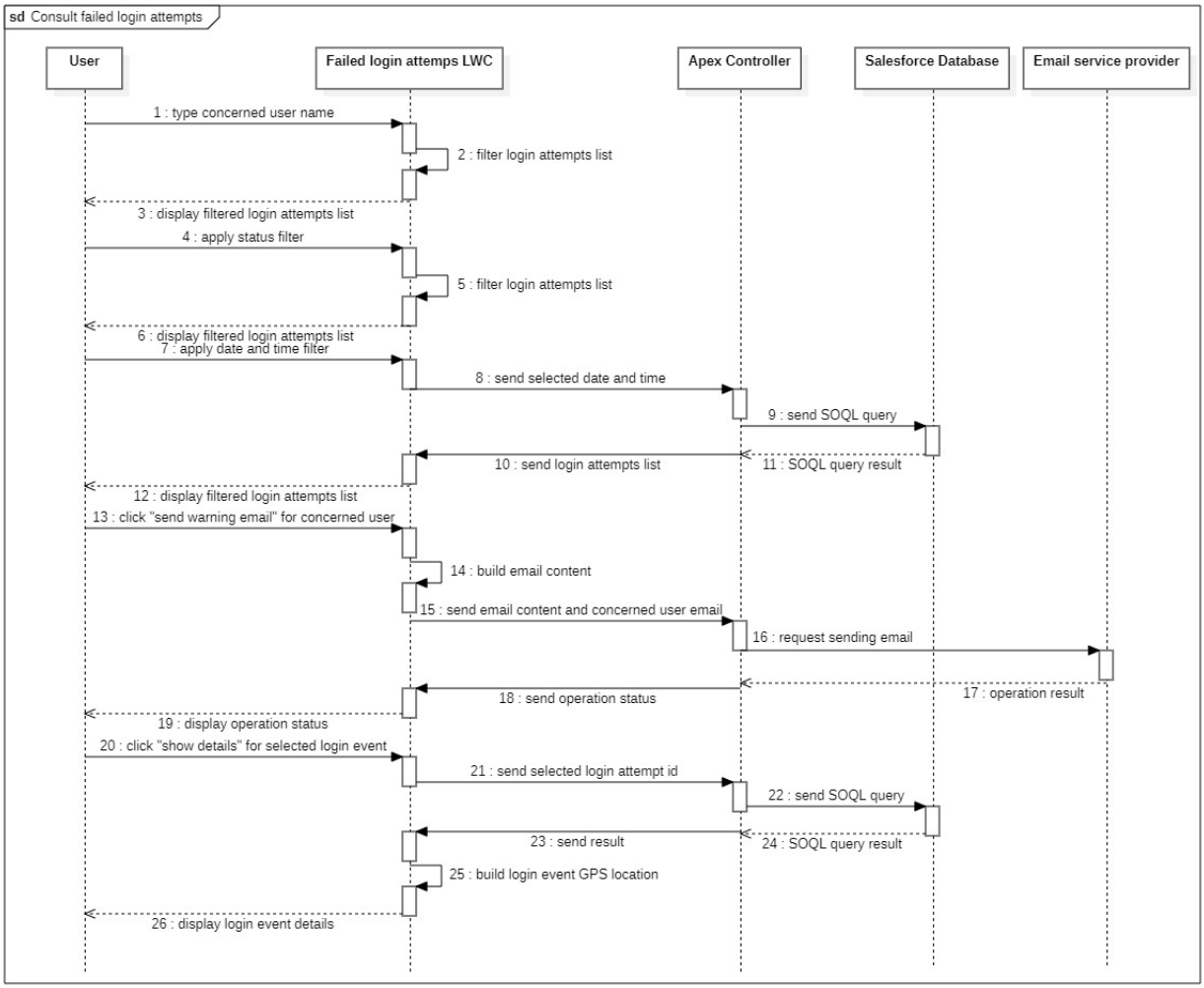


**Figure 21.** *Detailed sequence diagram of the ”Consult failed login attempts” use case*

### 3.2.6 Detailed sequence diagram of the ”Consult connection history” use case

The figure 22 illustrates the detailed sequence diagram of the use case ”Consult connection history”. This use case starts with opening an interface containing a bar chart displaying the number of logins for each member, the user may filter displayed results by date and time or by time range (yesterday, last week, or last month). For the displayed results the user may select

a specific member to access further information about him where he will also be able to update his Salesforce user license.

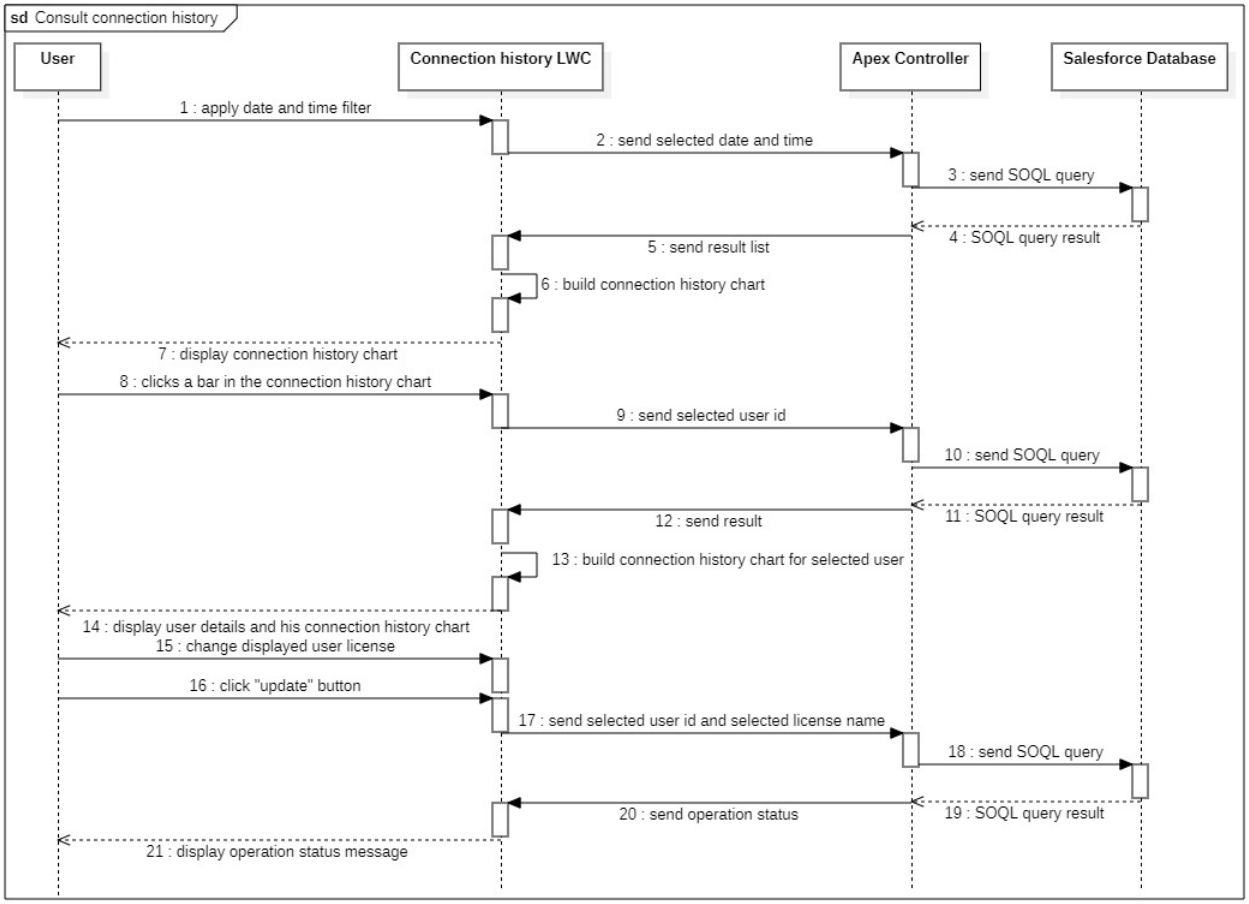


**Figure 22.** *Detailed sequence diagram of the ”Consult connection history” use case*

### 3.2.7 Detailed sequence diagram of the ”Access Chatbot” use case

The figure 23 illustrates the detailed sequence diagram of the use case ”Access Chatbot”. This use case starts with opening a chat interface where the user may type any question that he wants to ask the digital assistant, he may also select one of the predefined questions proposed by the system, then the system will prompt the answer given by the Chatbot or display an error in case there is one. Next, the user will be asked for feedback regarding the given answer and he will be able to click the like or the dislike button, the given feedback will be saved in the current Salesforce organization database and will be exploited for future updates of the Chatbot.

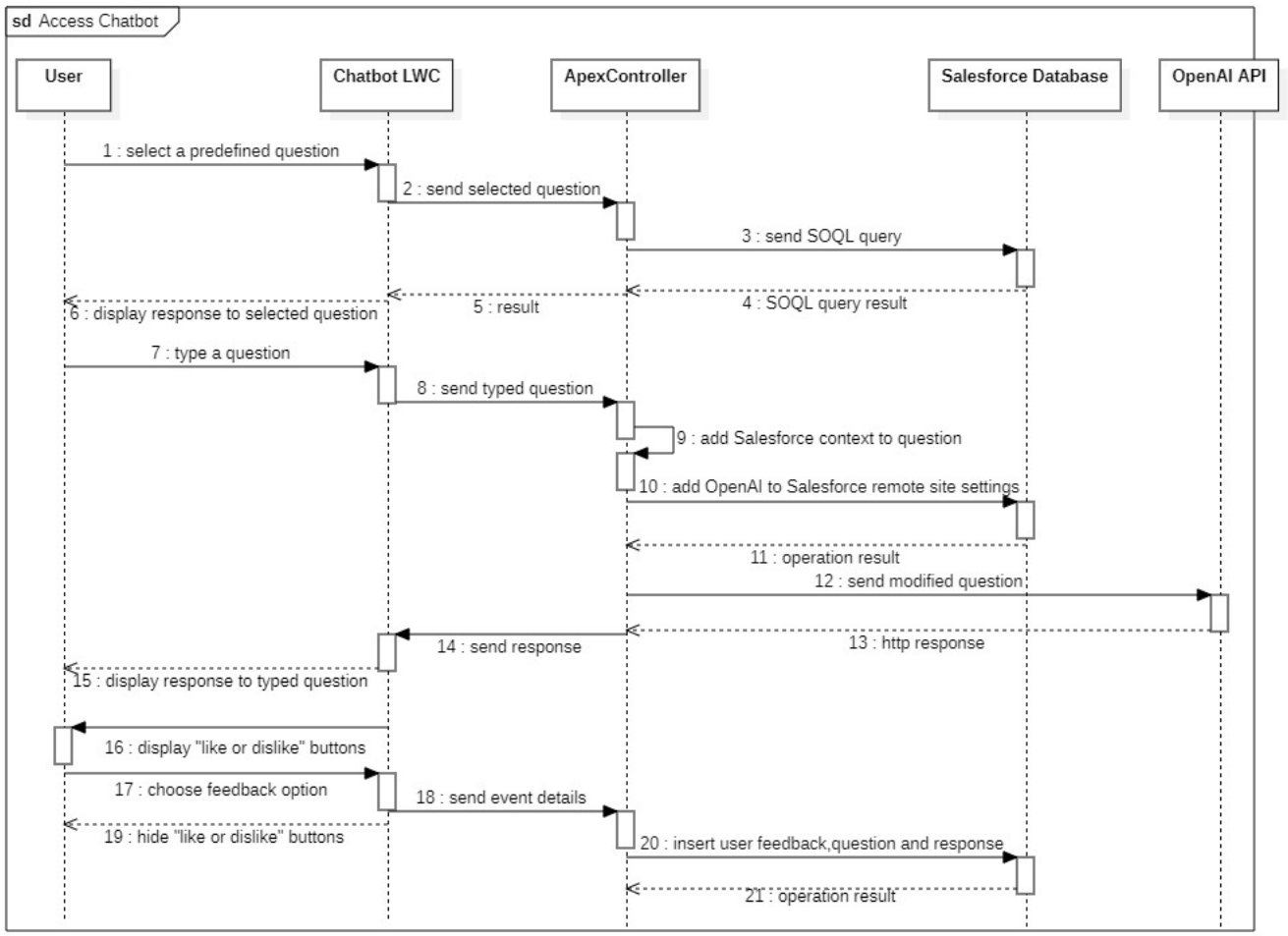


**Figure 23.** *Detailed sequence diagram of the "Access Chatbot" use case*

## *Conclusion*

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

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

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

Chapter 4

# Implementation

## *Introduction*

In this chapter, we describe the working environment used during the implementation of our application. We will also describe its physical layout using a deployment diagram. Then, we detail the work carried out and the results obtained using a set of screenshots representing the interfaces of the different features of our app.

## 4.1 *Technical specification*

In this section, we will present the technical choices relating to the hardware and software environment that contributed to the realization of our application.

### 4.1.1 Hardware environment

During the different stages of our project, i.e. documentation, code implementation and testing, we had:

- A laptop computer with the following configuration:
  - Brand: Lenovo;
  - Processor: AMD Ryzen 3 3200U @ 2.60 GHz ;

  - Graphical processor: Radeon Vega Mobile Gfx;
  - RAM: 12 GB;
  - Hard disk: 1TB;
  - System type: Windows 64-bit operating system.

- A desktop computer with the following characteristics:
  - Brand: Lenovo;
  - Processor: Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz 3.91GHz ;
  - Graphical processor: NVIDIA GeForce GT 1030
  - RAM: 8 GB;
  - Hard disk: 500 GB + 300 GB;
  - System type: Windows 64-bit operating system.

### 4.1.2 Software environment

Throughout the development phase, we used these software tools and the following programming languages and frameworks:

#### 4.1.2.1 Software tools

During the implementation of our project, we used the following software :

**Visual Studio Code**

Visual Studio Code is a popular source code editor developed by Microsoft. It provides a powerful and customizable environment for software development across various programming languages. Known for its lightweight design and extensive plugin ecosystem, Visual Studio Code offers features such as syntax highlighting, code completion, debugging capabilities, and version control integration. It supports multiple operating systems and offers a user-friendly

interface.

Visual Studio Code allows mainly the JS/HTML/CSS files in our application as well as the XML configuration files for each LWC component, in addition to Cls files that represent the Apex Controllers and the files representing the Aura applications.

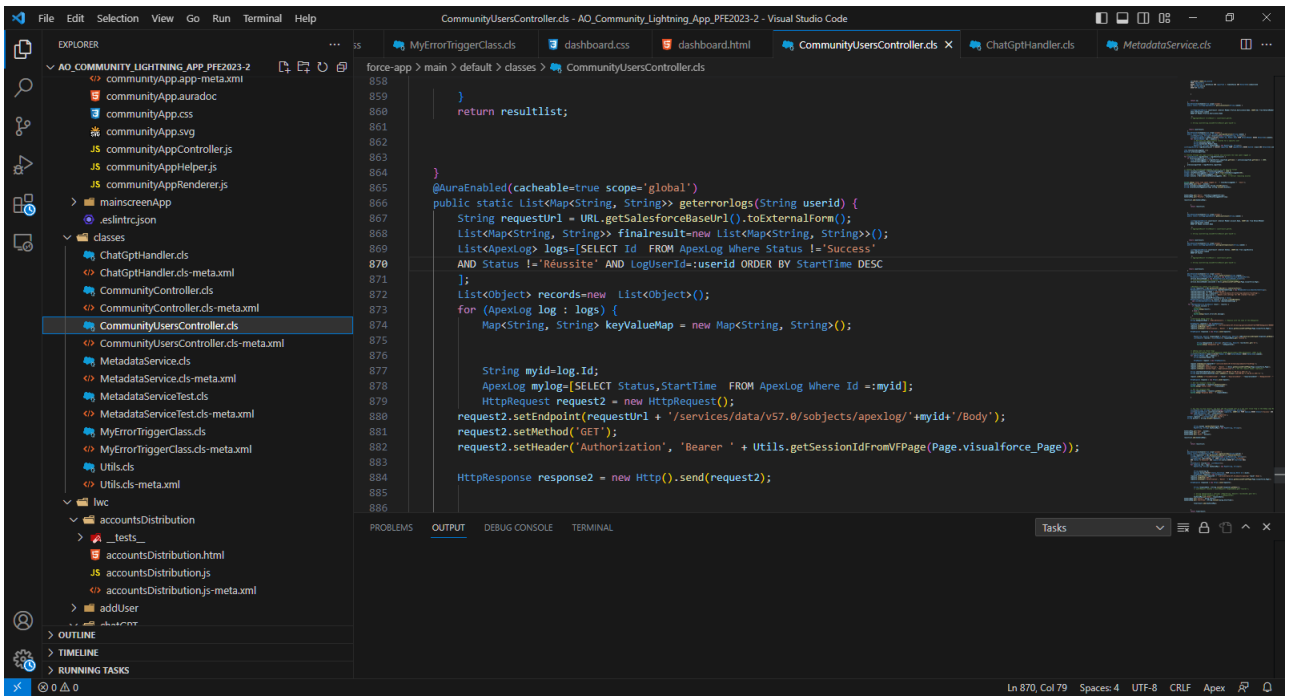


**Figure 24.** *Visual Studio Code*

### Salesforce Extension Pack for Visual Studio Code

The Salesforce Development Extension Pack offers a collection of powerful tools specifically designed for developers working on the Salesforce platform. Integrated seamlessly with the lightweight and highly extensible VS Code editor, these tools empower developers with a comprehensive set of features tailored for various aspects of Salesforce development. With specialized functionalities for managing development orgs, including scratch orgs, sandboxes, and DE orgs, as well as robust support for Apex, Aura components, and Visualforce, the extension pack significantly enhances the development experience and productivity for Salesforce developers.

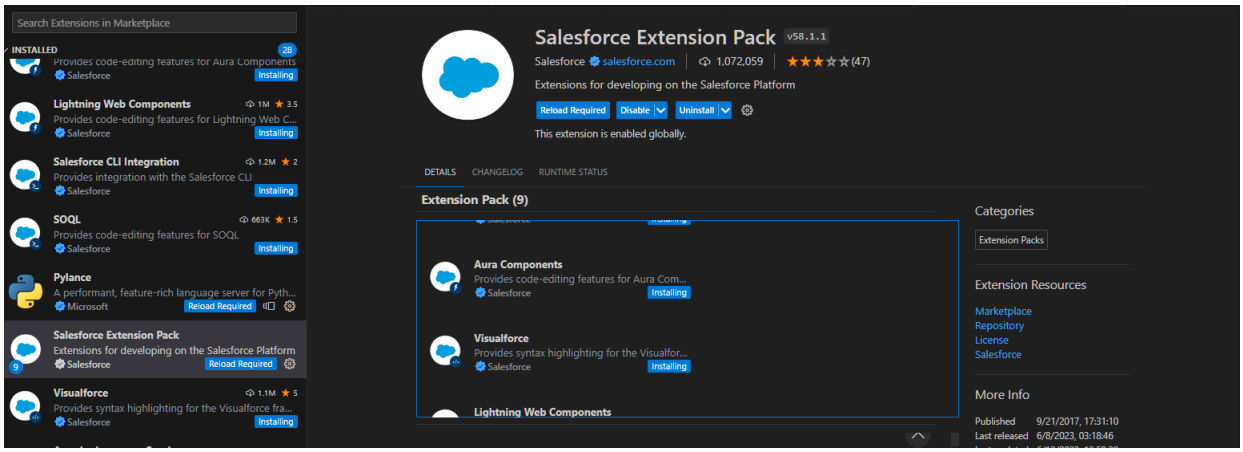


**Figure 25.** *Salesforce Extension Pack*

### Salesforce CLI (sfdx)

The Salesforce CLI is an essential command line interface that streamlines the development and automation processes for Salesforce orgs. It offers a wide range of capabilities, allowing developers to:

- Consolidate Development Tools: By utilizing the Salesforce CLI, developers can bring together all the necessary tools required for efficient development and execute commands seamlessly within their Salesforce org.
- Source Synchronization: The CLI enables developers to synchronize source code between local environments and scratch orgs, ensuring that changes made in the development environment are accurately reflected in the org.
- Org Management: Developers can create and manage orgs effortlessly using the CLI. This includes provisioning new orgs, configuring various settings, and handling administrative tasks.
- Data Import and Export: With the CLI, developers can easily import and export data to and from their Salesforce orgs, facilitating data migration, data backup, and data integration processes.

- Test Creation and Execution: The CLI provides functionalities for creating and executing tests, allowing developers to ensure the quality and reliability of their code through automated testing procedures.

- Package Creation and Installation: Developers can utilize the CLI to create packages that encapsulate specific components or functionalities within their Salesforce orgs. These packages can be easily installed in other orgs, simplifying the deployment and distribution of applications.

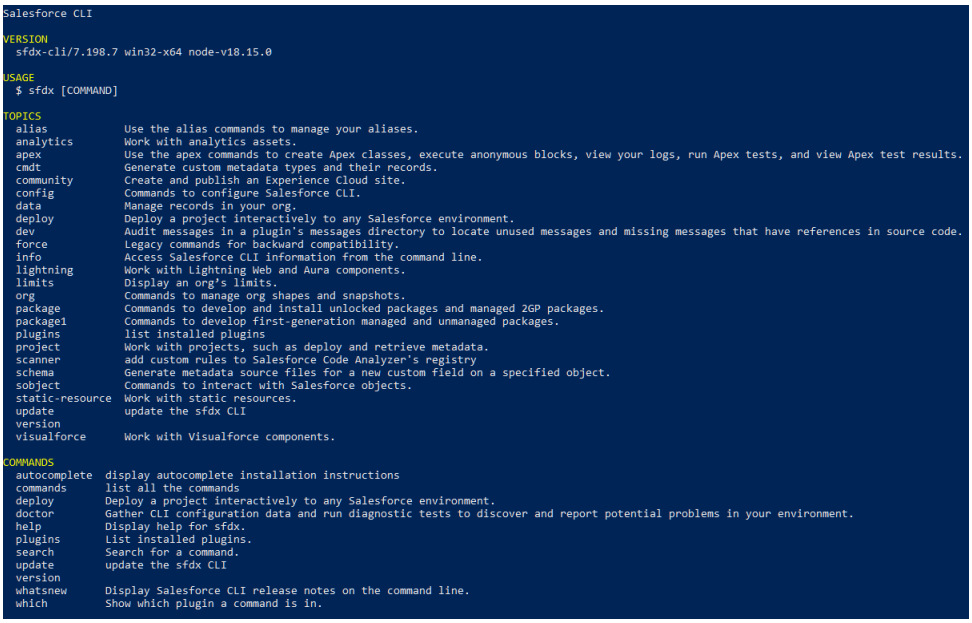


**Figure 26.** *Salesforce CLI*

### Salesforce Inspector

Salesforce Inspector is a highly useful productivity tool designed specifically for Salesforce administrators and developers. It offers the capability to inspect data and metadata directly within the Salesforce user interface (UI), enhancing the efficiency and convenience of working with Salesforce. With the added metadata layout, we were able to gain enhanced visibility and control over our Salesforce environment, making it easier to navigate, manipulate, and analyze data and metadata that is hidden by usual means.

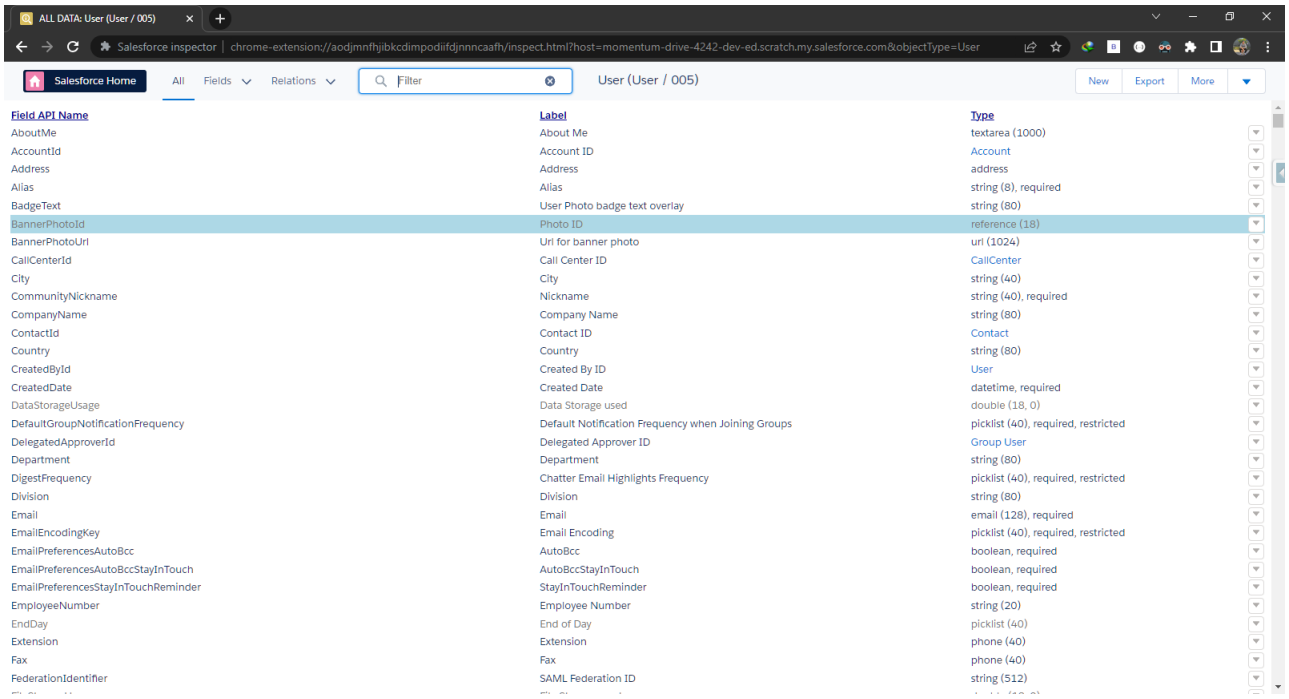


**Figure 27.** *Salesforce Inspector*

### Maven Tools for Salesforce

Maven Tools is an all-encompassing collection of Salesforce developer tools, purposefully designed to cater to the needs of Salesforce technical consultants. With its unique architecture, Maven Tools revolutionizes the delivery process of Salesforce implementations. This comprehensive toolkit offers a wide range of features, including an intelligent query editor, REST console, event hub, and much more, empowering developers to execute changes with increased speed and effectiveness.

In our application, we used Maven Tools for accessing the Salesforce tooling API.

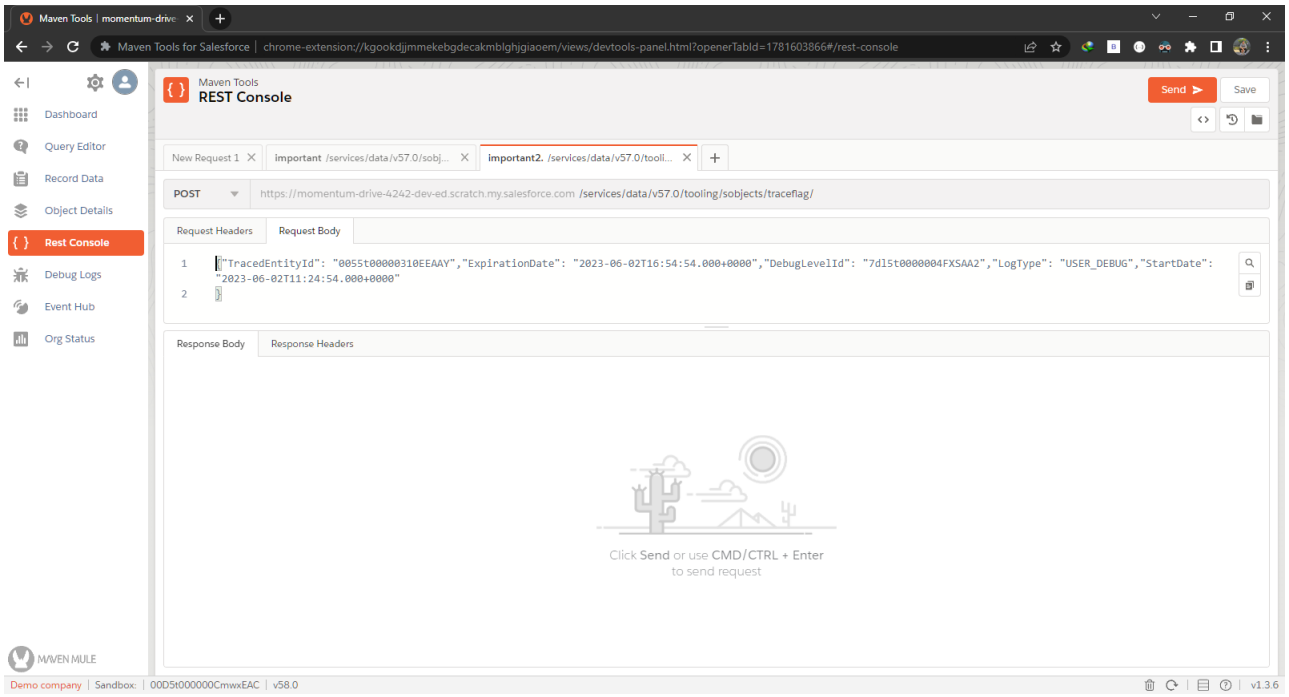


**Figure 28.** *Maven Tools*

### StarUML

StarUML is a versatile software engineering tool designed for system modeling, employing various modeling notations including the Unified Modeling Language (UML), Systems Modeling Language (SysML), and classical modeling notations. Developed and published by MKLabs, StarUML offers comprehensive modeling capabilities and is compatible with Windows, Linux, and macOS operating systems.

In our application, we used StarUML to establish UML diagrams for this report.

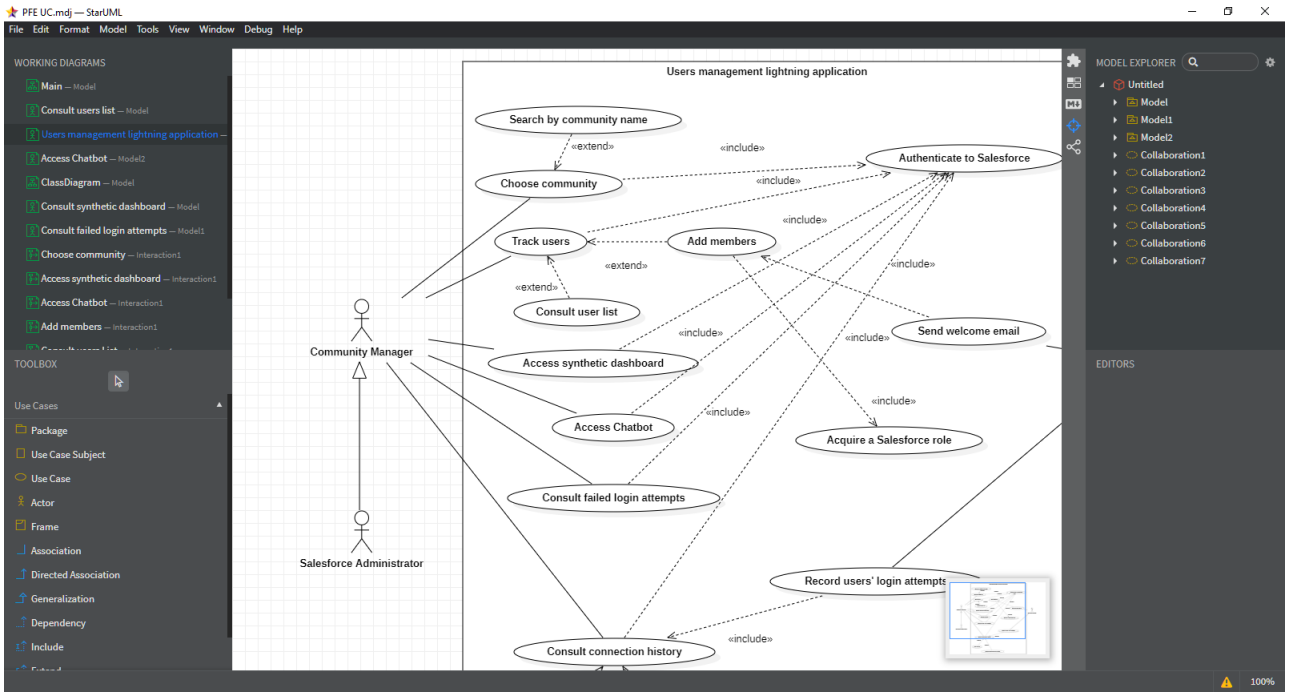


**Figure 29.** *StarUML*

#### 4.1.2.2 APIs and libraries

**Salesforce Tooling API**

The Tooling API is a valuable resource for developers looking to create custom development tools or applications for Lightning Platform applications. It offers a range of capabilities that facilitate efficient and interactive development experiences. One of the advantages of the Tooling API is its ability to retrieve smaller pieces of metadata using SOQL (Salesforce Object Query Language). This enables developers to fetch specific metadata components, resulting in improved performance compared to retrieving larger sets of metadata.
In our application, Salesforce tooling API was used to enable our application to interact with private Salesforce objects like Traceflag and DebugLevel.

**OpenAI API**

The OpenAI API is a versatile tool that can be utilized for a wide range of tasks involving natural language understanding and generation, as well as code-related operations. With its

capabilities, the OpenAI API enables users to perform tasks such as language translation, text summarization, sentiment analysis, ChatBot development, and more.
In our application, OpenAI API was used to power up our Salesforce digital assistant and enable it to understand natural language.

**Libraries**

| **Library** | **Usage** |
|---|---|
| Salesforce Metadata Service | - Automate remote site settings configuration.<br>- Automate custom Salesforce object for chatbot feedback. |
| LWC's wire | - Establish a connection between the lightning web component and an Apex method, a function in a JavaScript module, or a wire adapter. |
| LWC's API | - Create a public property or method that can be accessed by parent components. |
| Chart Js | - Build responsive and interactive charts from given Salesforce data. |

**Table 7.** *Libraries*

#### 4.1.2.3 Programming languages and frameworks

**Apex**

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on Salesforce servers in conjunction with calls to the API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.[**8**]

**LWC(Lightning Web Components)**

Lightning Web Components is a framework that leverages core Web Components standards and focuses on delivering optimal performance within Salesforce-supported browsers. By

utilizing the native capabilities of modern browsers, Lightning Web Components achieves a lightweight structure that excels in performance. It primarily utilizes standard JavaScript, HTML, and CSS, making it accessible and familiar to those already experienced in web development.

### Aura

Aura components are the self-contained and reusable units of an app. They represent a reusable section of the UI and can range in granularity from a single line of text to an entire app. The framework includes a set of prebuilt components. For example, components that come with the Lightning Design System styling are available in the lightning namespace.[**9**]

### SOQL

SOQL (Salesforce Object Query Language) serves as a powerful query language dedicated to retrieving data from Salesforce databases. While sharing similarities with SQL (Structured Query Language), SOQL is specifically designed for querying Salesforce data. Using SOQL, developers can efficiently query and retrieve data from both standard and custom objects within the Salesforce platform. This includes accessing records, and fields, and establishing relationships between different objects.

### SLDS

The Salesforce Lightning Design System (SLDS) represents a comprehensive set of guidelines and resources offered by Salesforce to easily create visually consistent and user-friendly interfaces within Salesforce applications. It also provides developers with a rich assortment of CSS styles, design patterns, and components. These resources can be utilized to construct custom Salesforce applications, including Lightning Web Components (LWC) and Aura components.

**XML**

The XML (Extensible Markup Language) allows the developer to create and share information formats via networks. This standard is used to describe data in our application as well as configure component visibility in Salesforce.

**JSON**

JSON(JavaScript Object Notation) is a lightweight data-interchange format that is easy to read and write and frequently used by developers. Within our application, it was mainly used for communicating with external APIs.

### 4.1.3 Deployment diagram of the application

The deployment diagram models the physical architecture of a system. This diagram shows the topology of the software elements that are deployed over the material elements. The figure 30 represents the deployment diagram of our application.

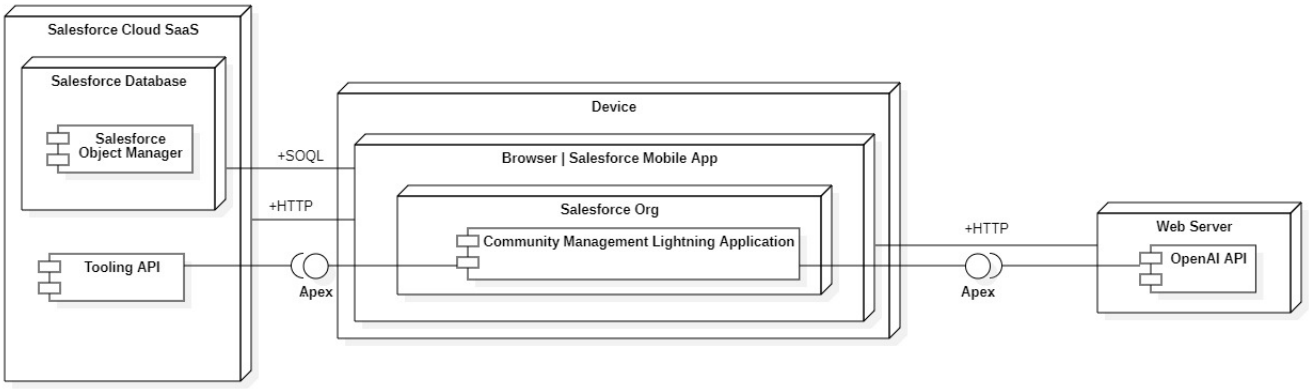


**Figure 30.** *Deployment diagram of the application*

Our application's deployment diagram is composed of 3 Nodes:

- Device: the physical device that is used to access the user's Salesforce org containing our lightning application

- Web Server: the web server embedding the OpenAI API responsible for powering up our ChatBot
- Salesforce cloud SAAS(software as a service): the provider of the Salesforce database that enables our application to create, read, update, or delete Salesforce records' data and the provider of the tooling API that enables our application to automate org configuration and similar operation.

The interactions between software components are:

- between our application and OpenAI API via HTTP requests. The exchange of data is in JSON format through the REST(representational state transfer) APIs, the resulting data is exploited through Apex.
- between our application and Salesforce Database via SOQL queries. The exchange of data is in sObjects (Salesforce objects) format.
- between our application and Tooling API via HTTP requests. The exchange of data is in JSON format through the REST(representational state transfer) APIs, the resulting data is exploited through Apex.

## 4.2 *Application interfaces*

In this part, we present the different features of our application by presenting the graphical interfaces produced:

### 4.2.1 Launching the application

The figure 31 represents the landing page of the application where the user can choose to access one of his owned communities or search for a specific one by its name using the provided search bar.

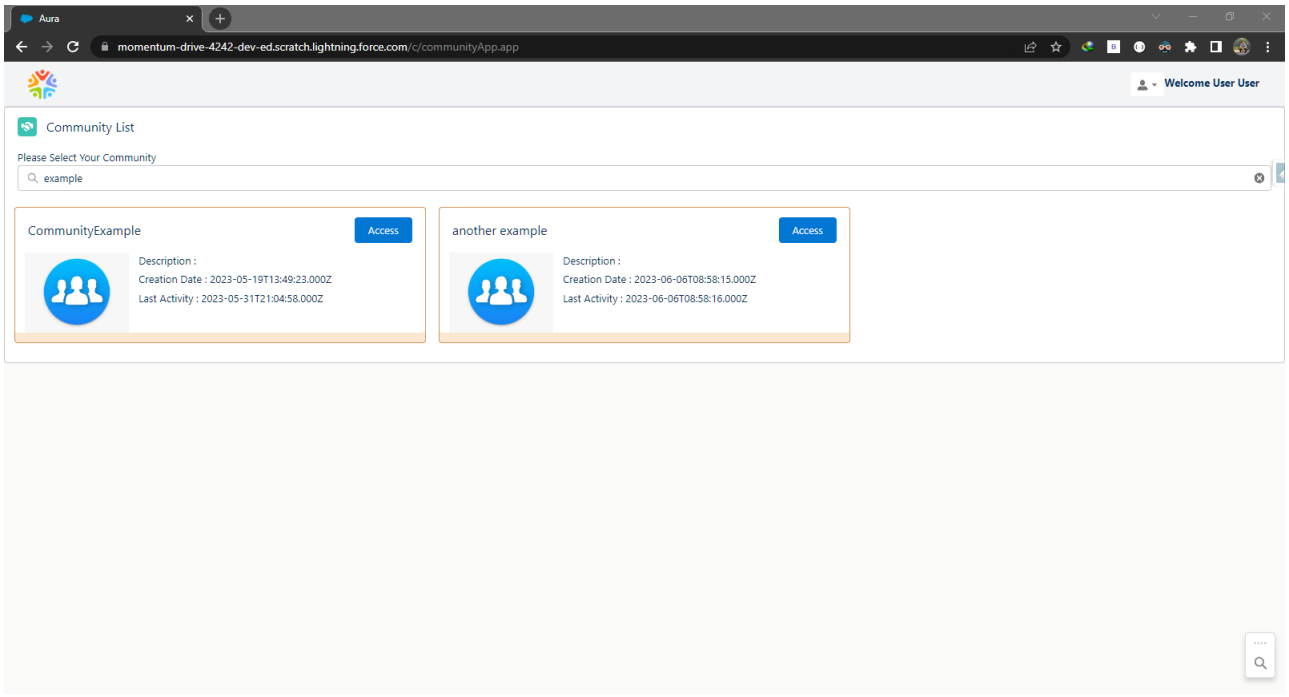


**Figure 31.** *Community selection interface (landing page of the application)*

### 4.2.2 Community Dashboard

The figures 32, 33, 34, 35 represent the community dashboard, which shows up upon accessing the selected community by the user, containing KPIs charts that help the user monitor his community.

The figure 32 represents respectively the accounts distribution chart, Salesforce user licenses distribution chart, and login attempts status distribution chart for the current community.

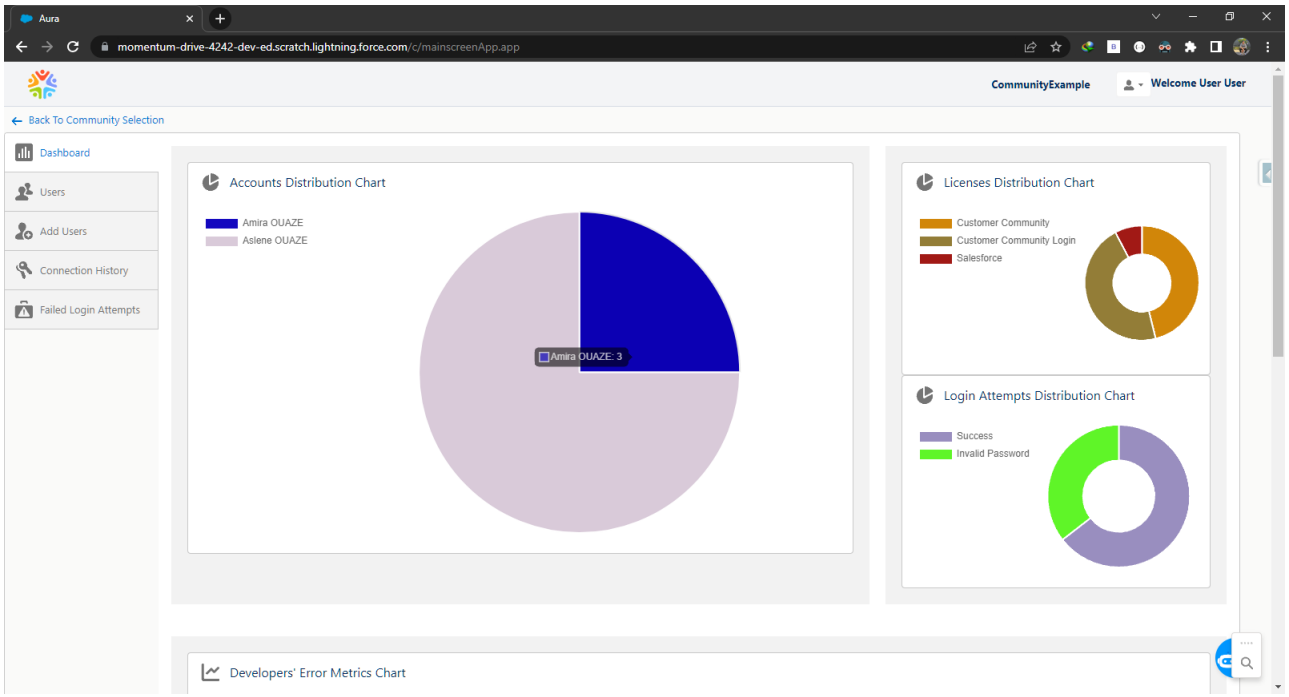


**Figure 32.** *Community dashboard interface (distribution charts)*

The figure 33 represents the time spent in the community chart that shows the number of hours and minutes spent by each member when surfing the current community.

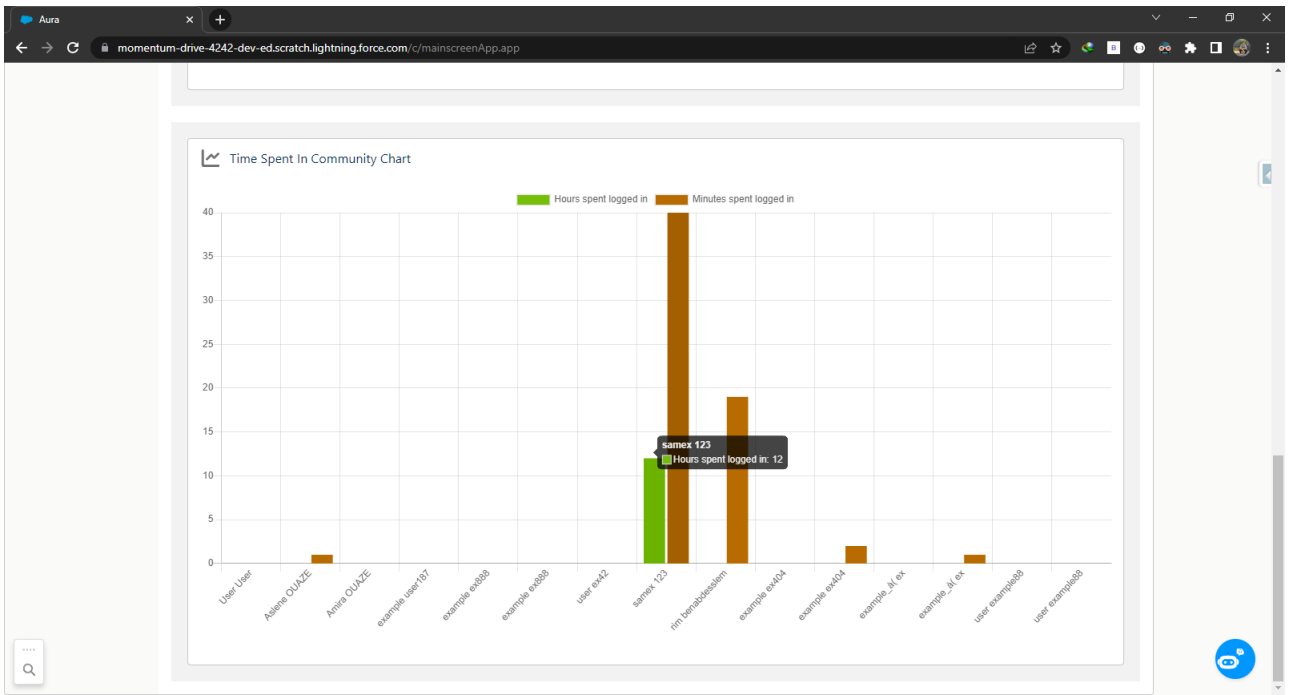


**Figure 33.** *Community dashboard interface (time spent in the community chart)*

The figure 34 represents the community developers' error metrics chart that shows the number of programming errors faced by each developer working in the current community.

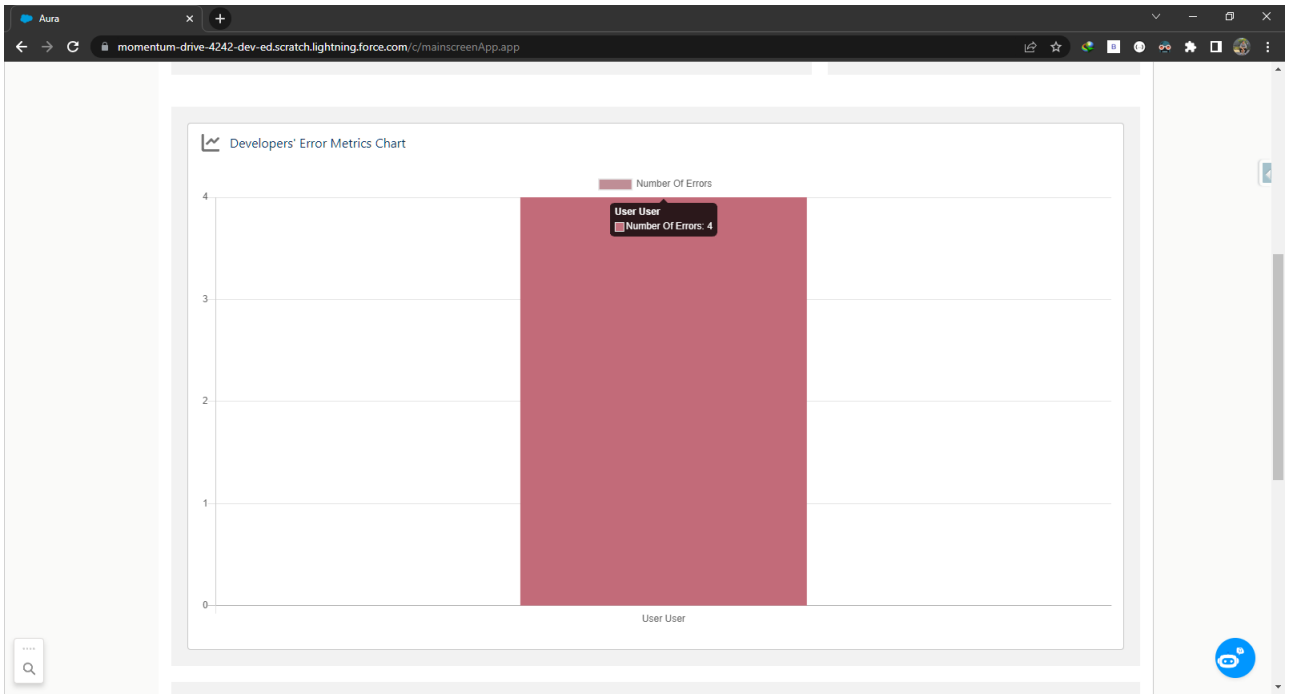


**Figure 34.** *Community dashboard interface (developers' error metrics chart)*

Upon clicking on one of the bars displayed in the previous chart the complete error logs specific to the selected developer will be displayed as shown in the figure 35.

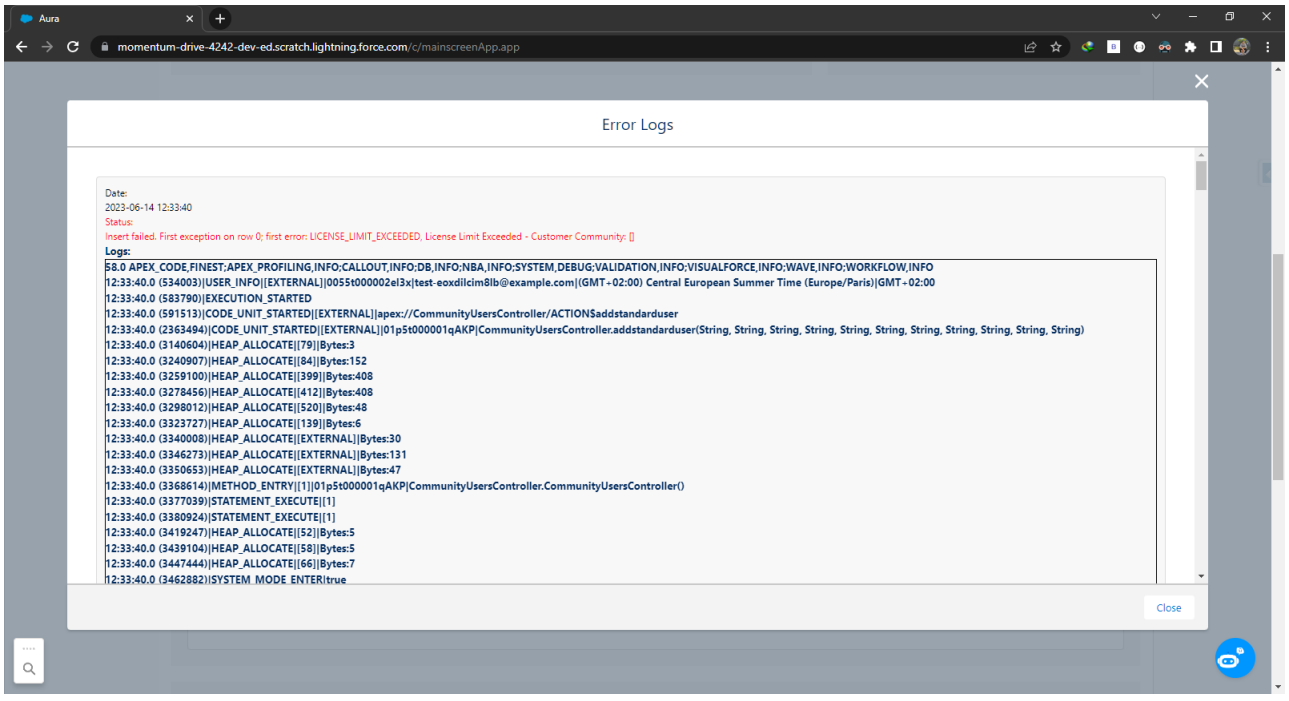


**Figure 35.** *Community developer error logs interface*

### 4.2.3 User list

The figure 36 represents the community members list where the user can navigate through his community members' information using given pagination options. He may also filter displayed members by user status (active or inactive), account name, or profile label using the given combo boxes or search for a specific user using his Salesforce user name, full name, or email using the provided search bar. The user can perform actions on one or multiple members, these actions can be activating/deactivating selected members, sending welcome emails to selected members, or sending reset password reset emails to selected members.

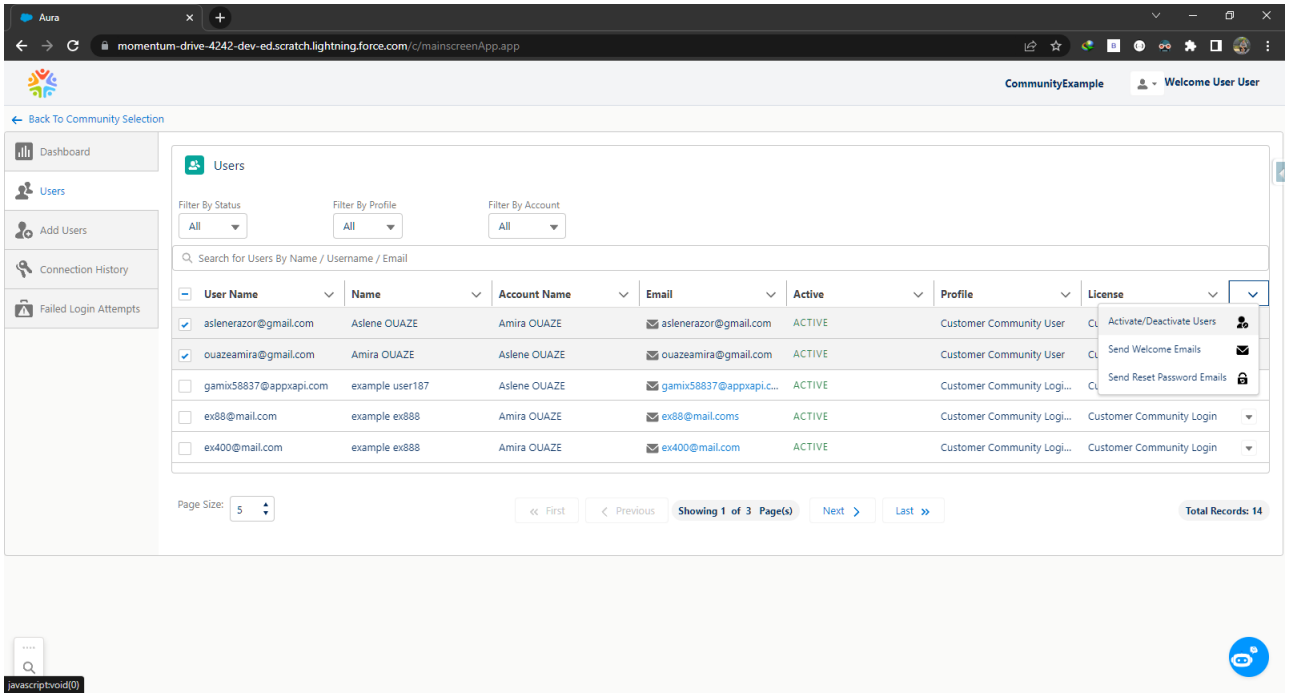


**Figure 36.** *User list interface*

The user may also consult details about a specific user as well as update said details as shown in the figure 37, the show details action is only selectable when selecting one community member.

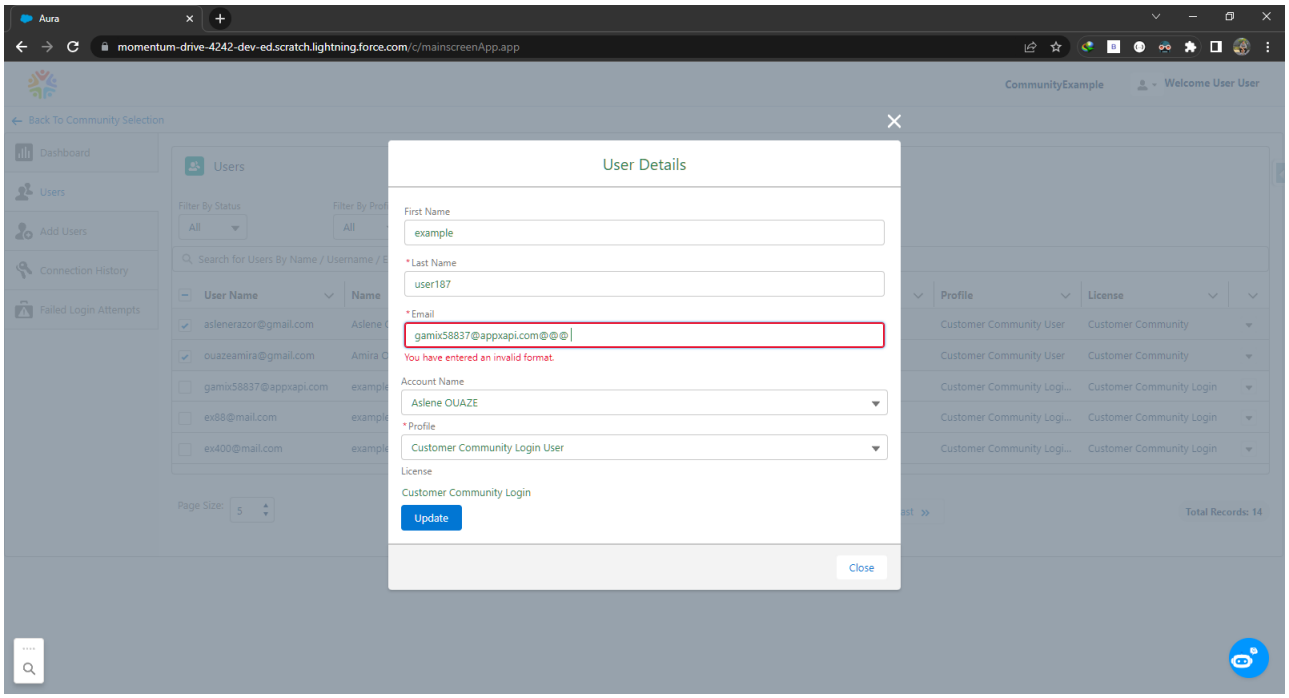


**Figure 37.** *User details interface*

### 4.2.4 Add users

The figure 38 represents the add users form interface where the user can add one or multiple users using the displayed form fields. The user can fill out the initial given form and press the ”submit all displayed forms” button where the system will prompt any error message if it exists or the success of the operation otherwise. The user is also able to clone the previously filled form, add a new empty one, or delete the latest one using the given buttons.

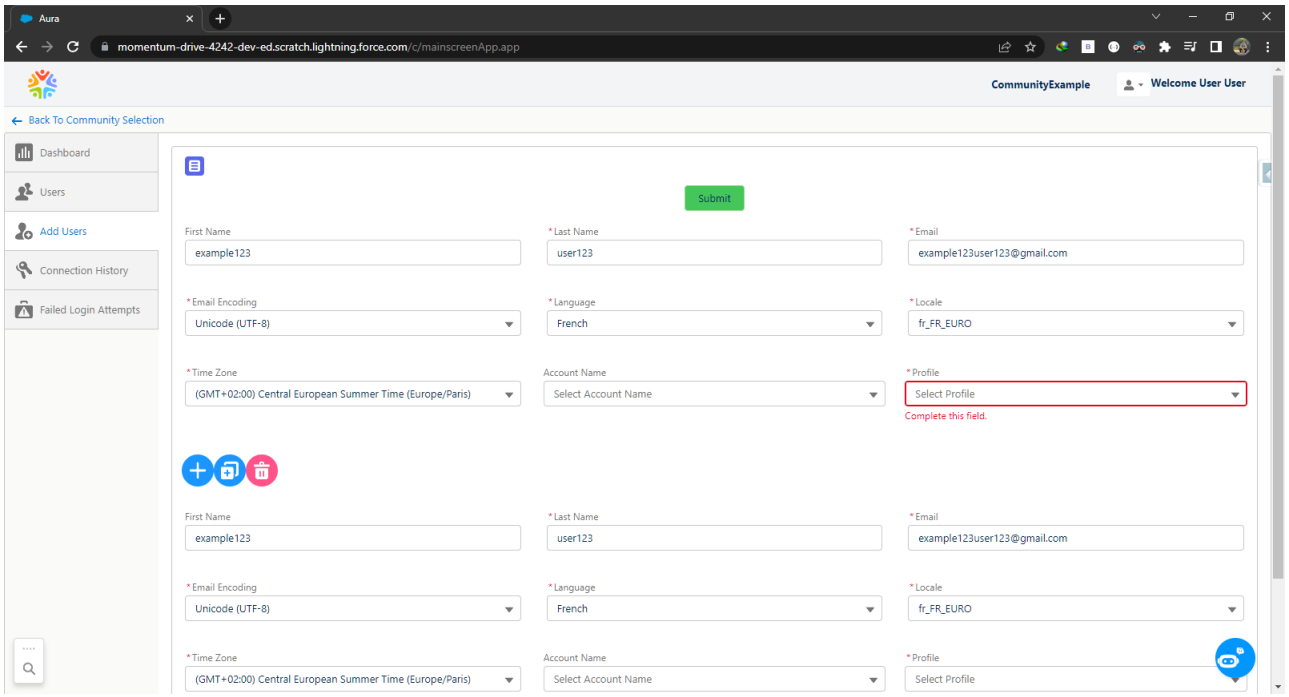


**Figure 38.** *Add users interface*

### 4.2.5 Connection history

The figure 39 represents the connection history interface that shows the number of logins for each user based on the given ”from” and ”to” dates and time or the time range filter where the user can interact with both of these filters to change the displayed chart.

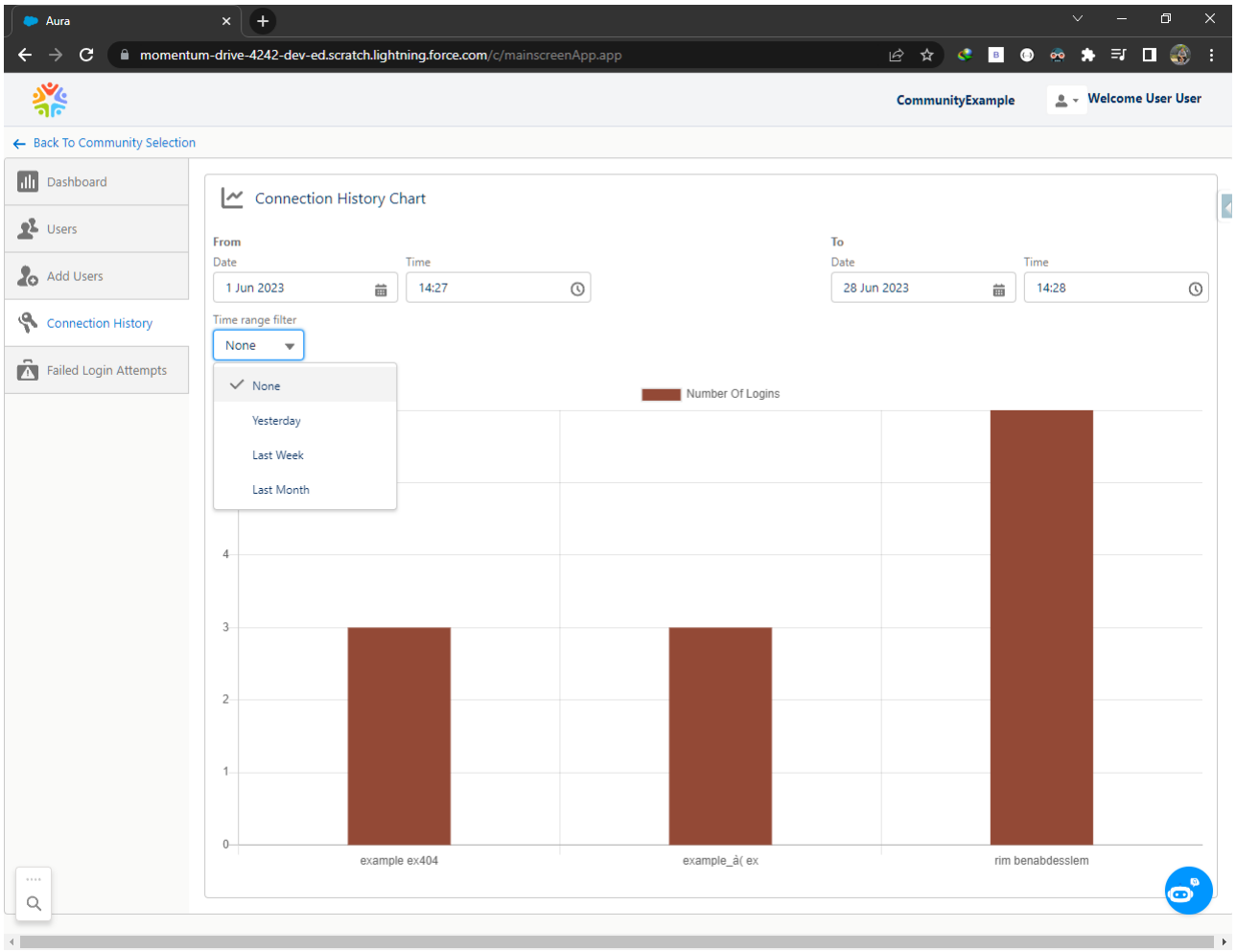


**Figure 39.** *Connection history interface (chart)*

The user may also click one of the displayed bars on the displayed chart to access brief details about the concerned user and update his Salesforce user license as shown in the figure 40.

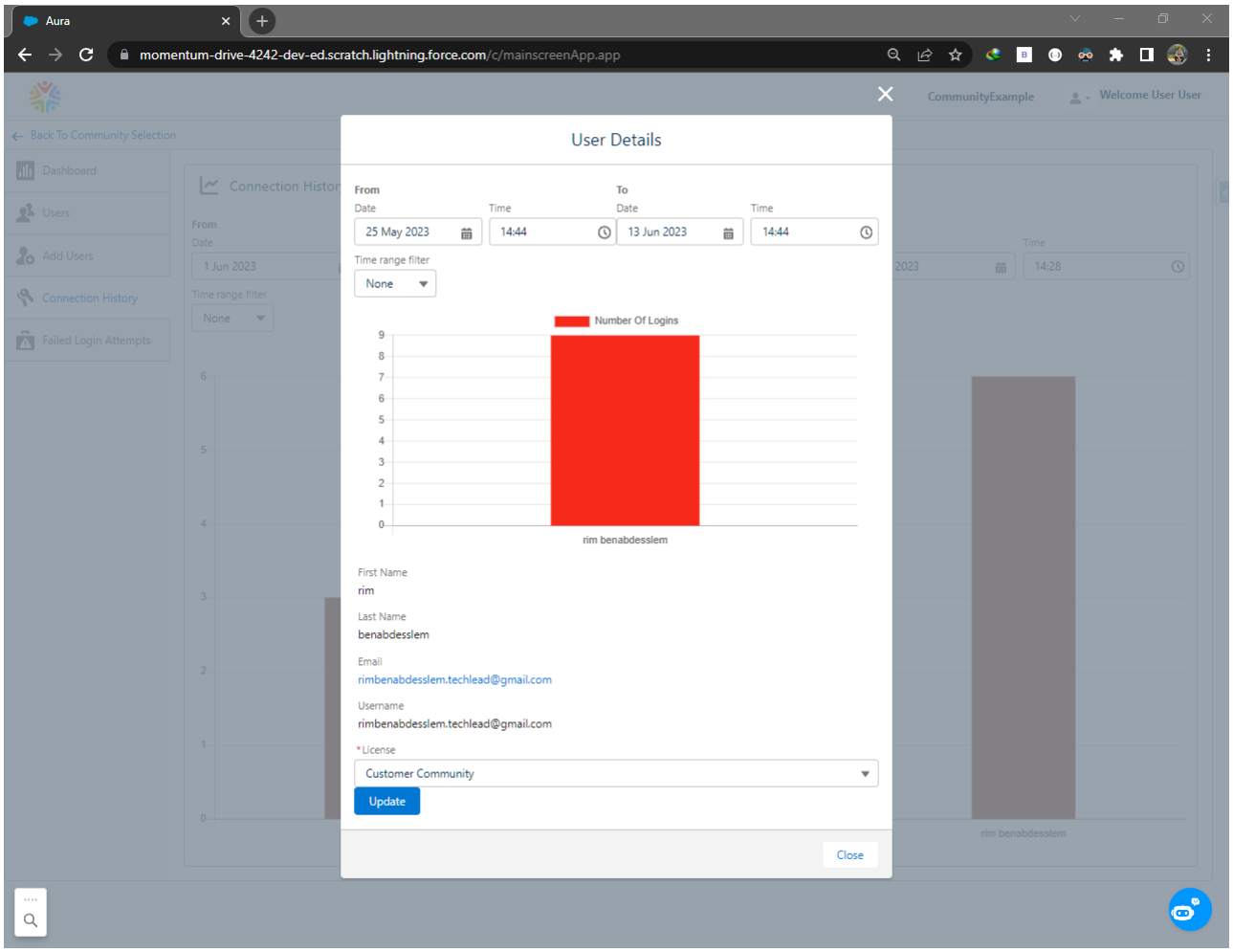


**Figure 40.** *Connection history interface (user details)*

### 4.2.6 Failed login attempts

The figure 41 represents the community members' failed login attempts list where the user can navigate through his community login events using given pagination options. He may also filter displayed events by status (invalid password, no community access, etc) using the given combo box or search for a specific event using the concerned user's Salesforce user name or full name using the provided search bar, he may also filter displayed results by date and time using the provided "from" and "to" filters.

The user can also choose to show more details about a specific login event or send a security warning email to the concerned user using the given action menu.

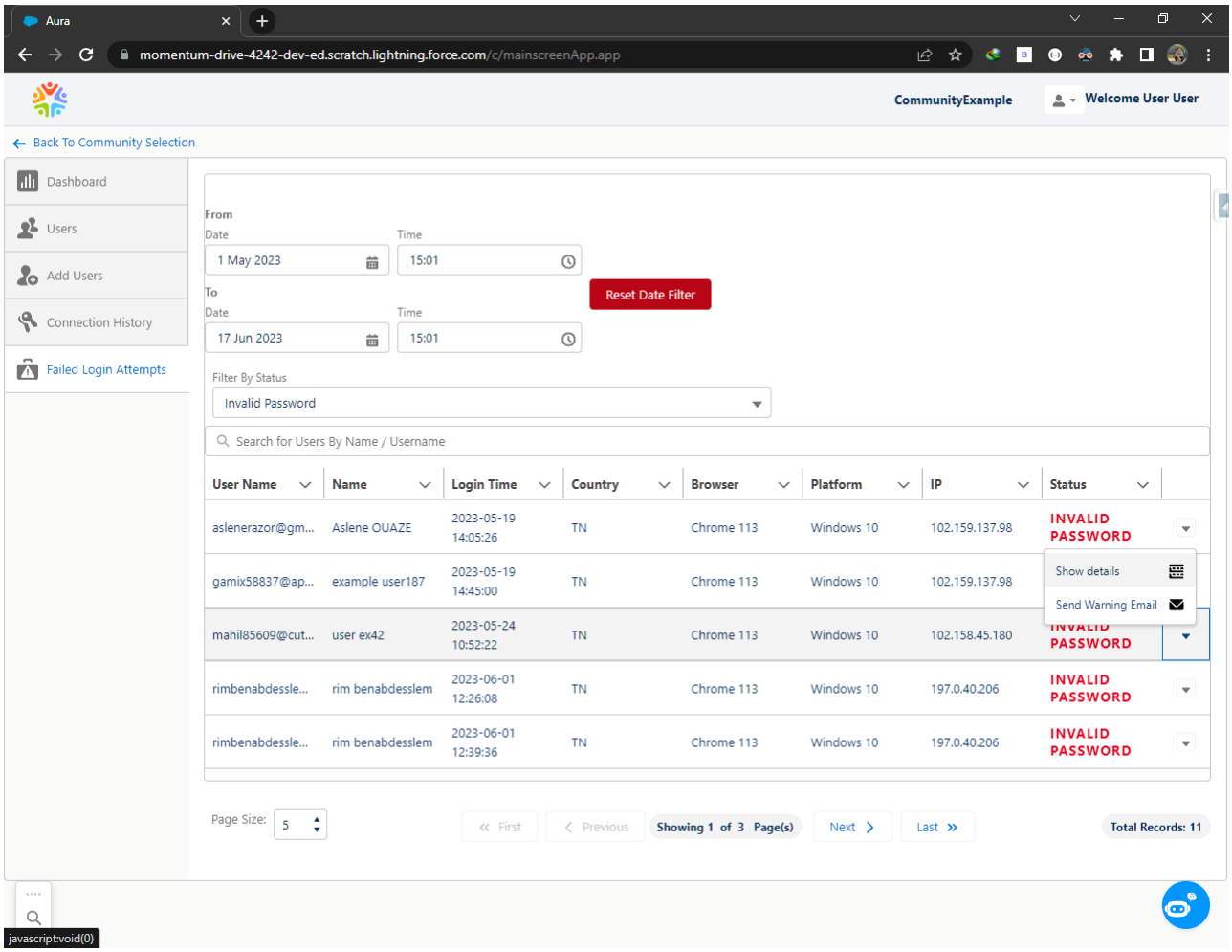


**Figure 41.** *Failed login attempts interface (event list)*

The figure 42 represents the selected login event's details interface where the user can navigate the map displaying the event's location in addition to consulting further details about it.

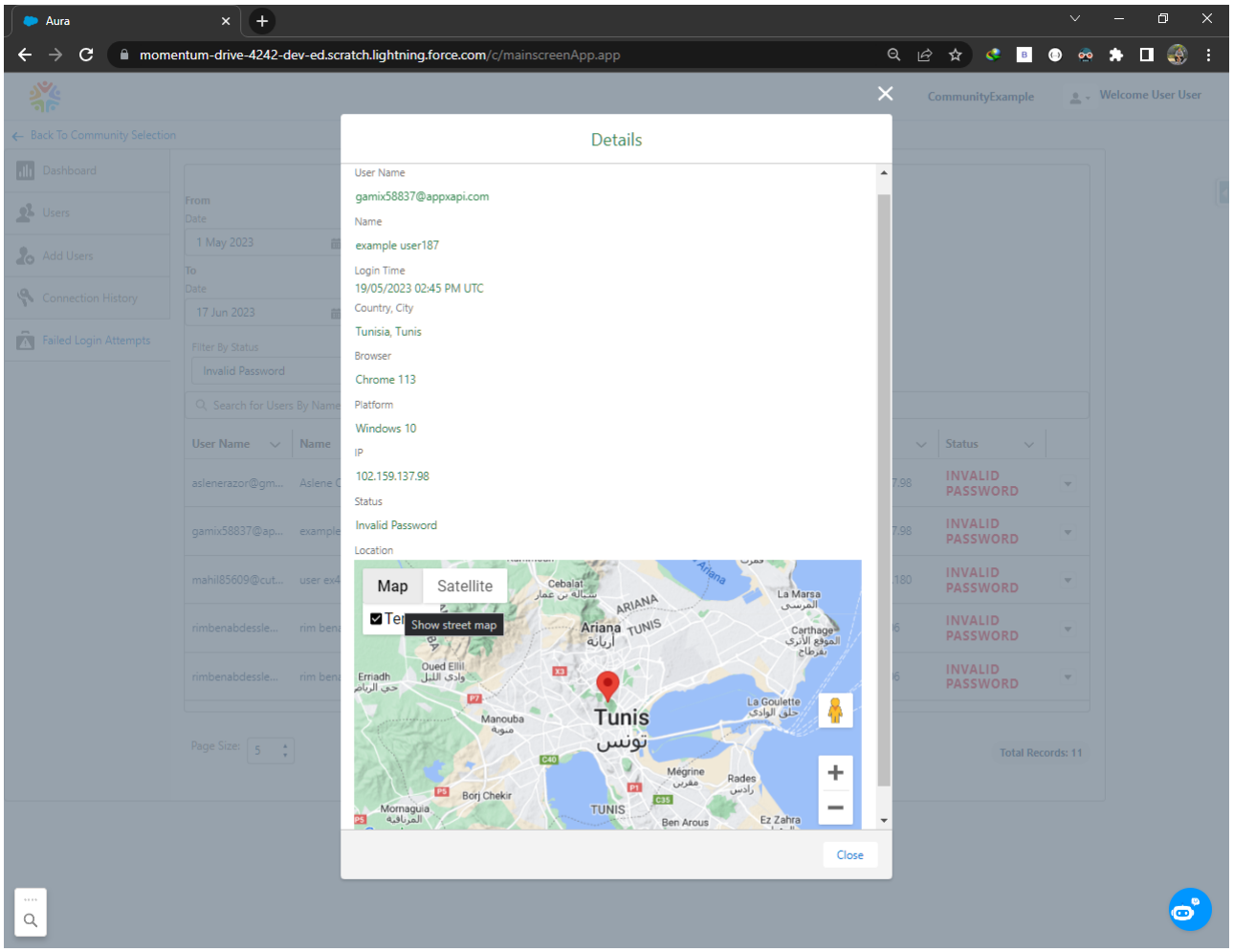


**Figure 42.** *Failed login attempts interface (event details)*

### 4.2.7 Chatbot

The figure 43 represents the Chatbot interface which is accessible from anywhere on the main screen of our application using the floating Chatbot button. In this interface, the user may choose one of the predefined questions or type his question, upon submitting the question the answer will be displayed on the screen where the user may choose to give positive or negative feedback to the developers for future improvements.

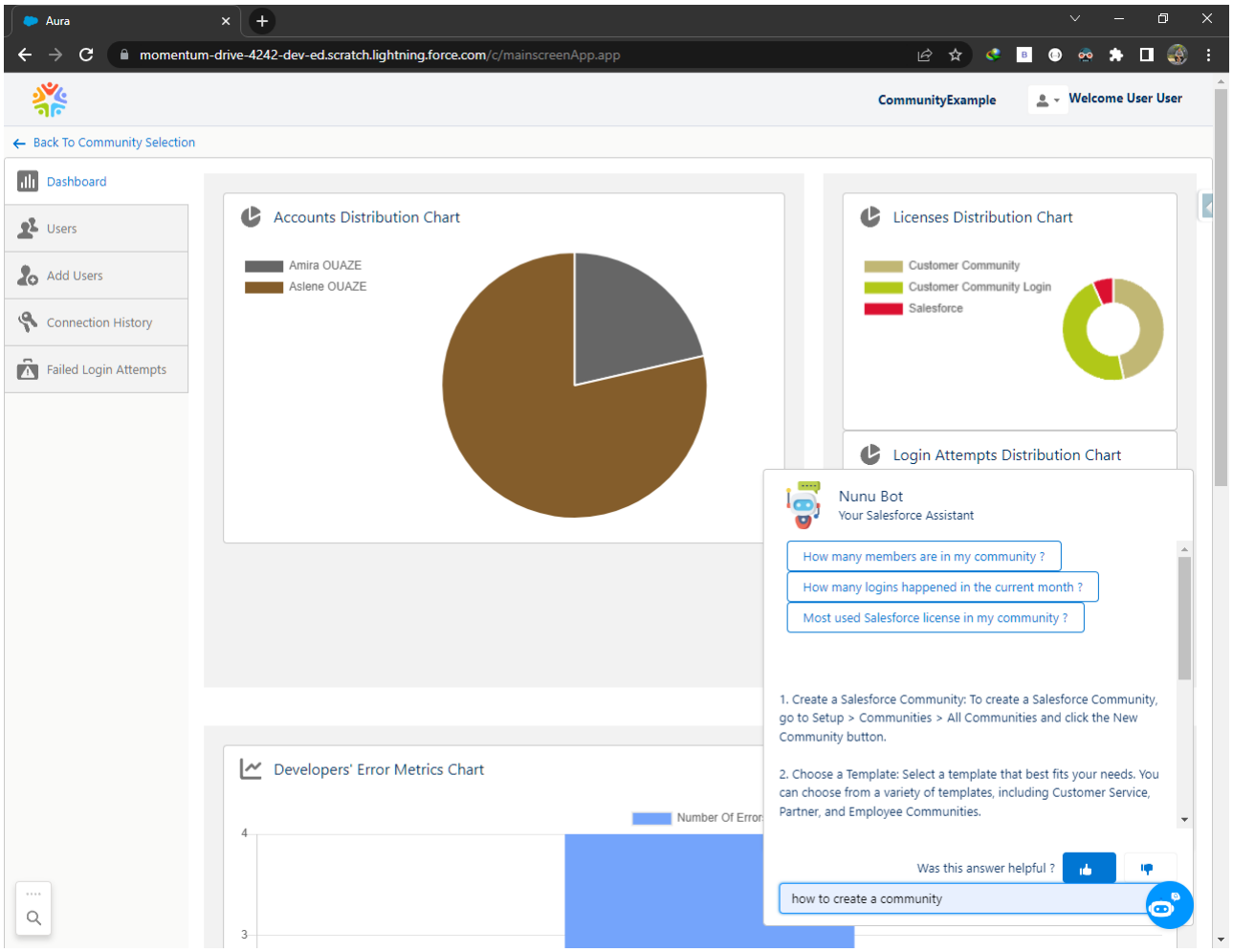


**Figure 43.** *Chatbot interface*

## *Conclusion*

In this chapter, we presented the technical specification of our application through the introduction of hardware and software environments. Ultimately, we have described the features of our application, illustrated by screenshots of its user interfaces. We end our report with the general conclusion and the different prospects.

# Conclusion and perspectives

Admittedly, the diversity of lightning applications is an enrichment world of Salesforce extensions, where everything goes through these applications to facilitate tasks, bring more comfort, and gain in time and money.

The objective of this report is to present the application carried out during our end of studies internship project within the company TechLead. Our application facilitates the task of managing the members of Salesforce communities for community managers and administrators. The app carried out also integrates a smart Salesforce digital assistant and a synthetic dashboard for community directors.

This internship allowed us to improve our technical Salesforce knowledge given that we used a variety of technologies associated with said platform. On the other hand, during this internship, we got used to professional life and teamwork as well as time management regarding software development. To conclude, our realized application can be extended by further improving the Chatbot feature, as well as adding an event notification system. It is also possible to make our application multilingual in future updates.

# Bibliography

[1] Get Started with the Salesforce Platform. [trailhead.salesforce.com] (visited on 22/05/2023).

[2] Discover Use Cases for the Platform. [trailhead.salesforce.com] (visited on 24/05/2023).

[3] Understand the Salesforce Architecture. [trailhead.salesforce.com] (visited on 24/05/2023).

[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] (visited on 25/05/2023).

[6] Model–view–controller [en.wikipedia.org] (visited on 26/05/2023).

[7] What is MVC Design Pattern? [www.educba.com] (visited on 01/06/2023).

[8] What is Apex? [developer.salesforce.com] (visited on 01/06/2023).

[9] Lightning Aura Components Developer Guide [developer.salesforce.com] (visited on 02/06/2023).