IG4 2023/2024 - Software Engineering Practices

Subject:

"SUBSTRACK: The Subscription Management Application"
https://github.com/nicolas-goyon/SEProject

Deliverable n°2
Use cases analysis

CURREA Juan Esteban
GOYON Nicolas
HANNOU Assia
RAGUE David

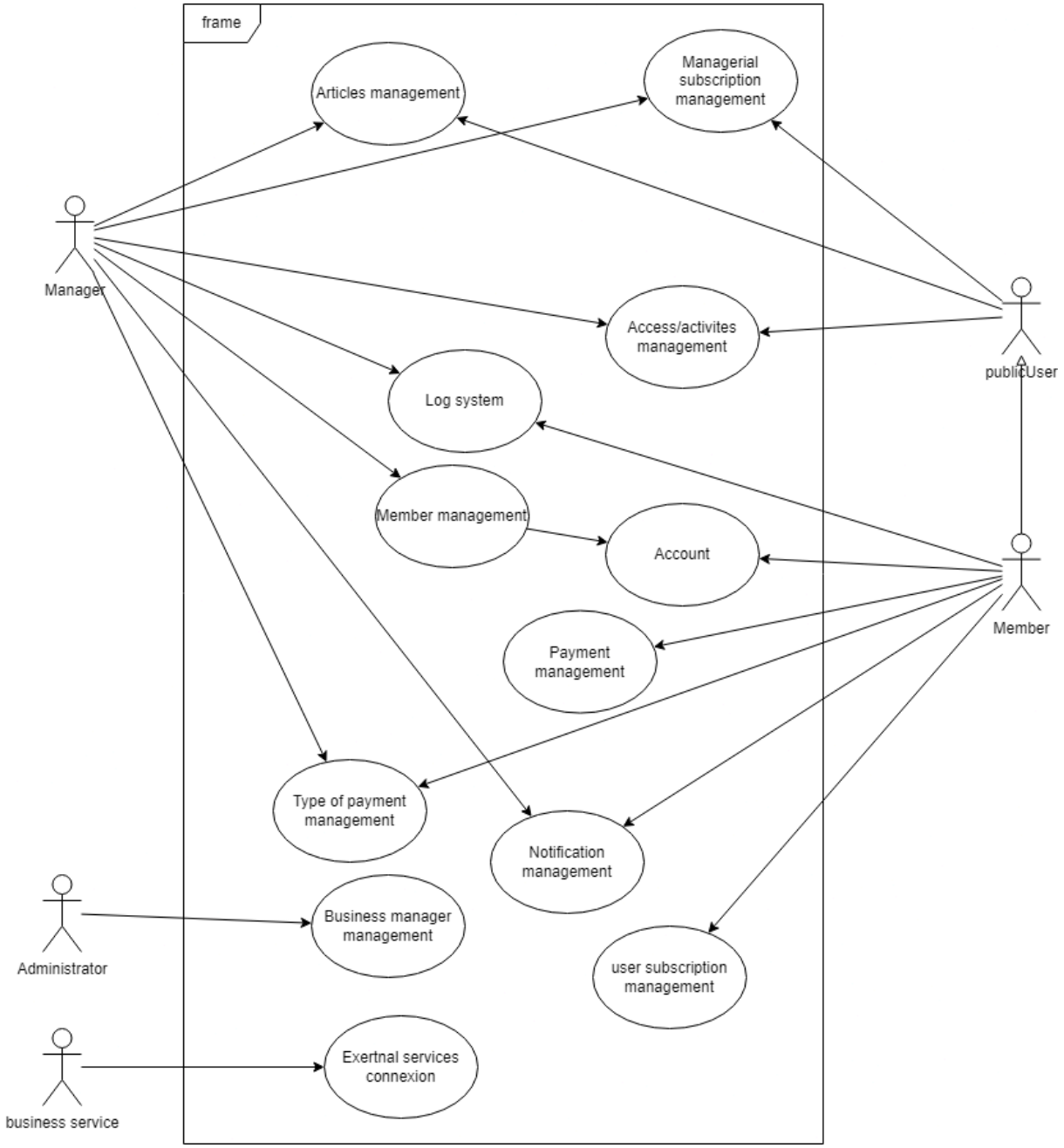
Summary

Introduction

The focus of this project is on developing software designed for subscription management within a gym-like setting.

The software, named SubsTrack, aims to achieve key objectives, including the effective management of basic functions such as Member management (both self-management and management by designated managers), plans and access control managed by appointed personnel, and subscription and payment processes. The implementation of various functionalities will ensure a cohesive user experience. Another significant feature is the integration of an external service, like a badge door, which will verify if the user has the required access permissions for the designated area.

General diagram



Use Cases

Users :

- Member
- Business Manager
- Business Administrator

1. Account management(login/register)

2. Member Management :
    a. create a member
    a. update a member
    b. view a member
    c. delete a member

3. User subscription management:
    a. view his/her subscription
    b. delete his/her subscription
    c. subscribe to a plan

4. Managerial Subscription Management:
    a. create a subscription plan
    b. view all subscriptions plans
    c. modify subs management
    d. delete a subs plans
    e. view the details of a subscription plan

5. Payment management
    a. validate the payment
    b. cancel the payment
    c. select a payment method

6. Type of Payment management
    a. add type of payment
    b. view type of payment
    c. remove type of payment
    d. update type of payment

7. Business manager Management:
    a. create a BM member (leur donner certains droits)
    b. update a BM member
    c. view a BM member
    d. delete a BM member

8. Articles Management
    a. create an article
    b. view an article
    c. update an article
    d. delete an article

9. Notifications
    a. Create notifications
    b. View notifications
    c. Recurrent notification services (ref : diagrams.net)

10. External service connexion
    a. Access to the API by an external service

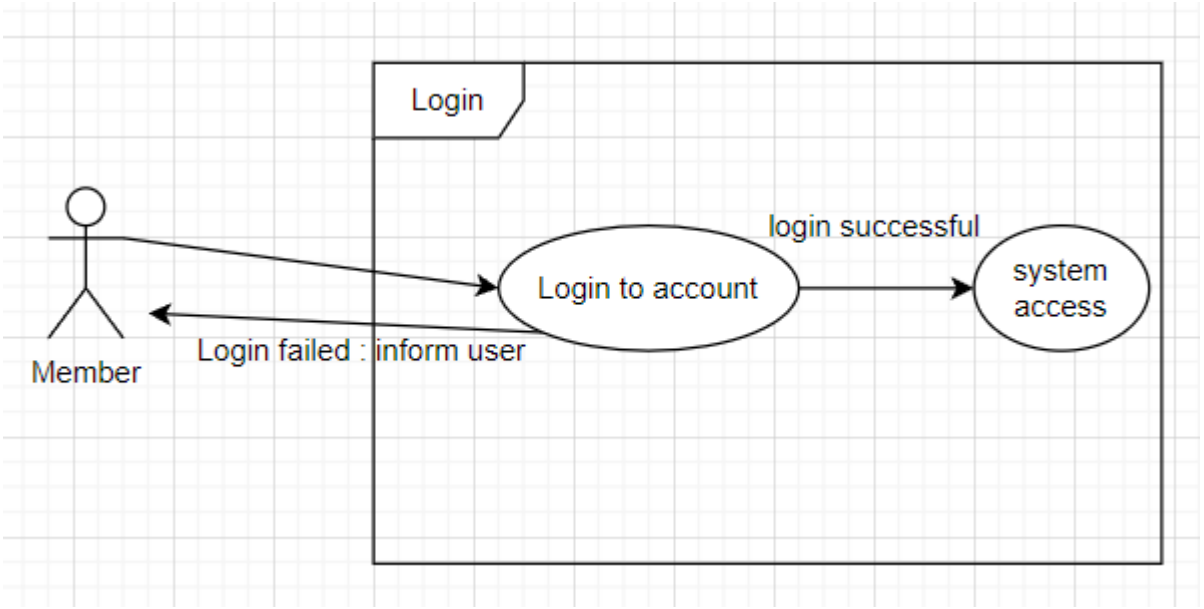
11. Log system
    a. view entrance/exit to history
    b. add entrance/exit to history

12. Access/activites management
    a. create an access for an activity
    b. view an access for an activity
    c. update an access for an activity
    d. delete an access for an activity

Use cases detailed

1.Login

1.0 Diagram



1.1. Brief Description

This use case describes how a user logs into the Subscription Management Application.

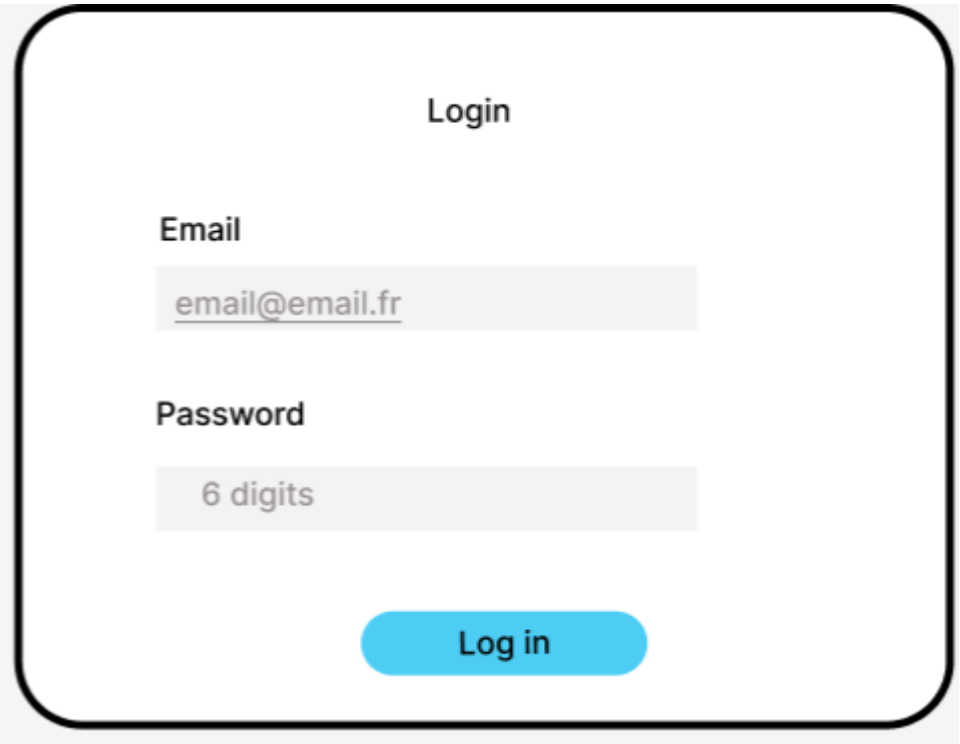
1.2. Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to Login to the Subscription Management Application .

1. The system requests that the actor enter his/her email and password
2. The actor enters his/her email and password.

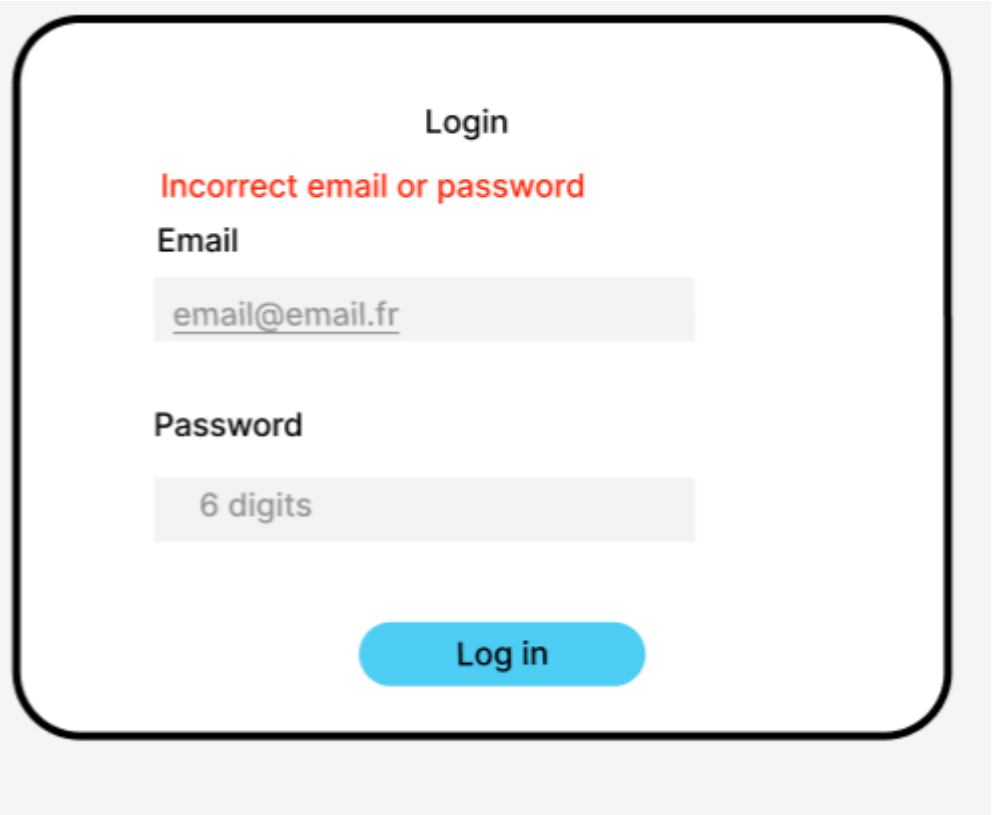
3. The system validates the entered email and password and logs the actor into the system.



1.2.2 Alternative Flows

1.2.2.1Invalid Email/Password

If in the **Basic Flow**, the actor enters an invalid email and/or password, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.



1.3 Special Requirements

None.

1.4 Pre-Conditions

None.

1.5 Post-Conditions

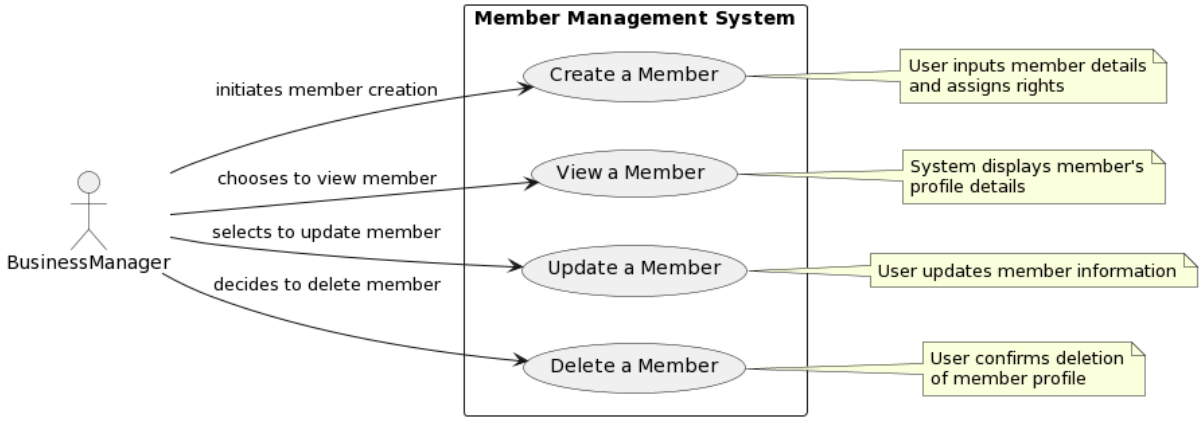
If the use case was successful, the actor is now logged into the system. If not, the system state is unchanged.

1.6 Extension Points

None.

2. Member Management

2.0 Diagram



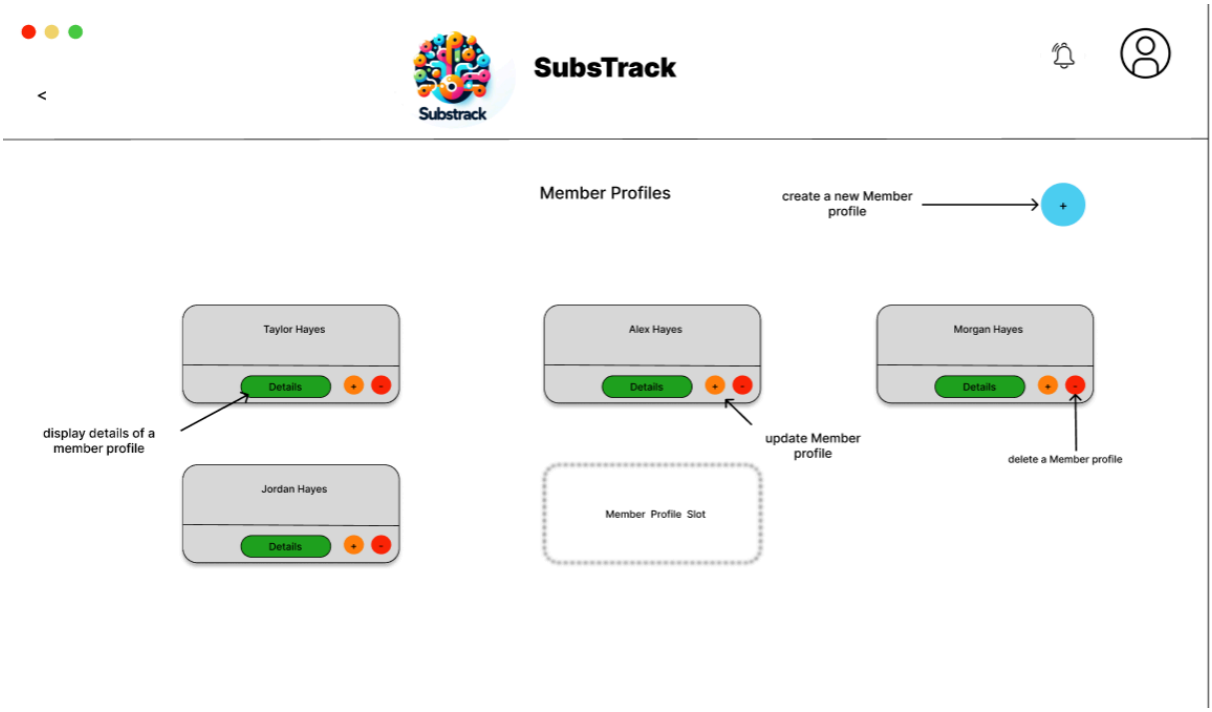
2.1 Brief Description

This use case describes the process of Member Management within a system, including creating, updating, viewing, and deleting member profiles, along with assigning roles.

2.2 Flow of Events

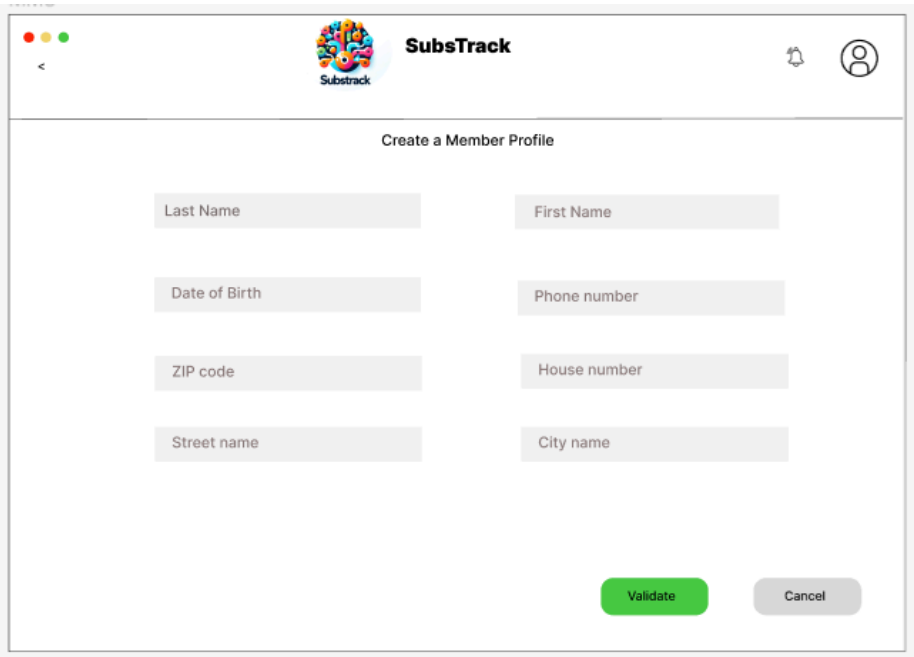
2.2.1 Basic Flow

- Initiation: The process starts when an authorized user or administrator wishes to update a member's profile.
- Identification: The system prompts for the member's ID or unique identifier.
- Data Retrieval: Upon ID submission, the system retrieves and displays the current member information.
- Modifications: The user makes necessary changes to the member's profile, such as updating contact details, membership status, or other personal information.
- Confirmation & Update: The user submits the changes. The system validates and updates the member's profile.



2.2.1.1 Create a Member

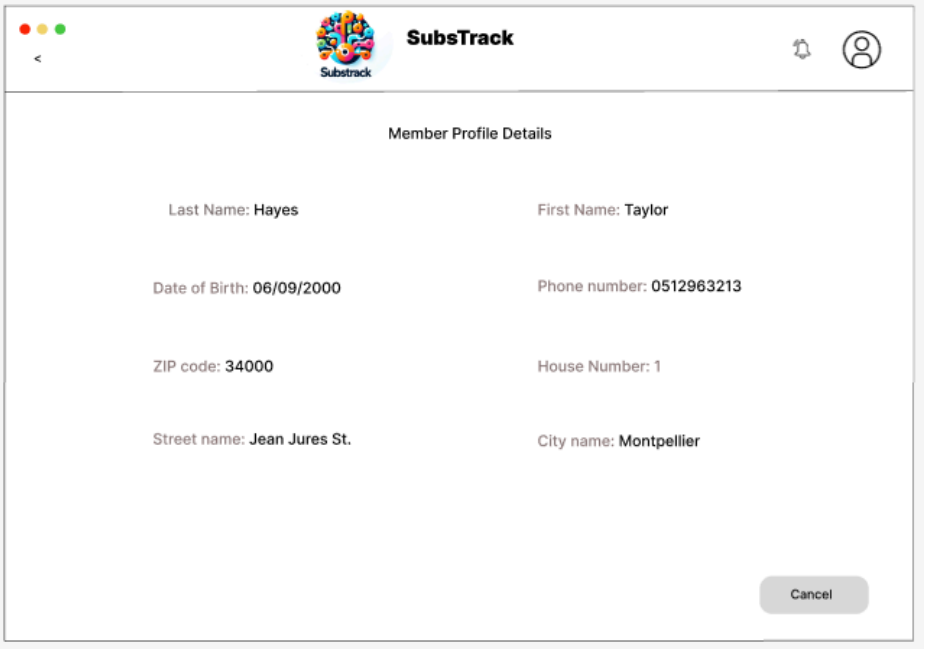
- The system requests details such as name, contact information, and the specific rights to be assigned.
- Once the user inputs the information, the new member is added to the system, and the rights are assigned accordingly.



2.2.1.2 View a Member

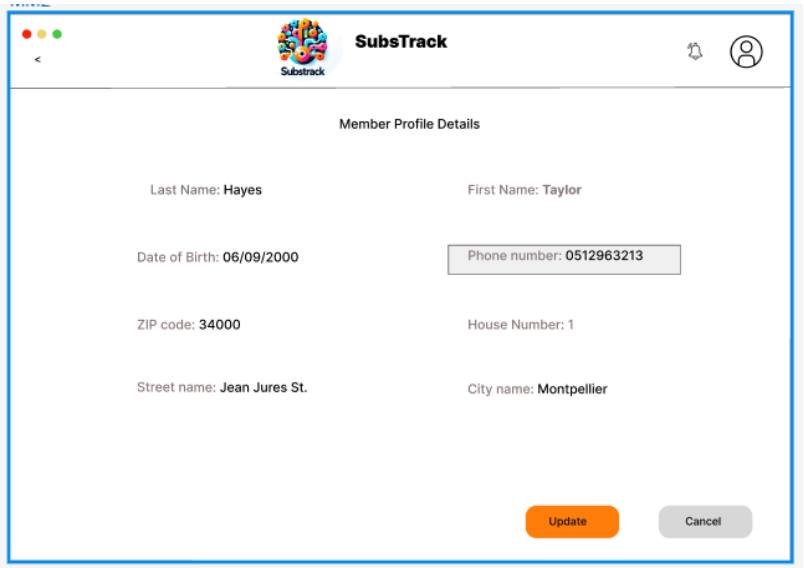
- The user selects the member profile to view.

- The system retrieves and displays the selected member's details.



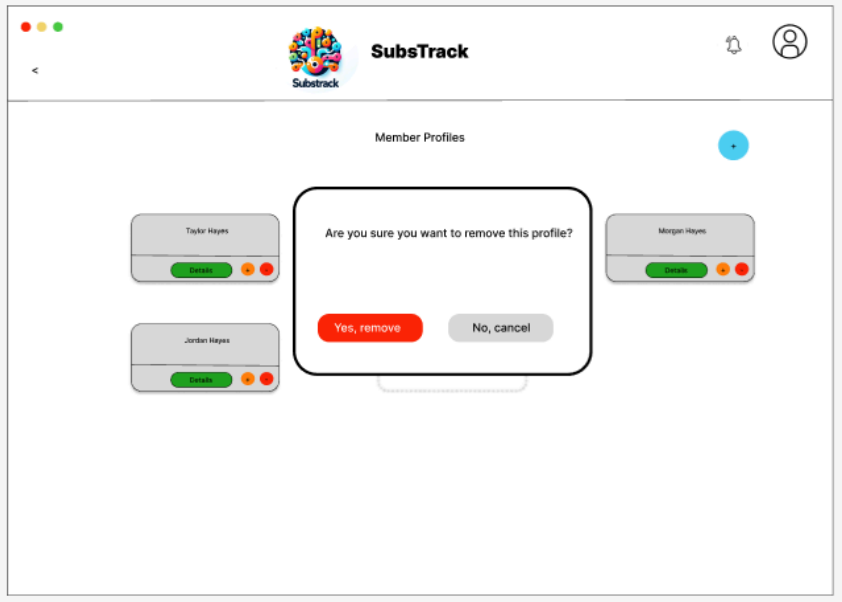
2.2.1.3 Update a Member

- The system asks for the member ID to be updated.
- The user enters the ID, and the system displays the member details once authorized.
- Changes are made as needed, and the system updates the member's profile.



2.2.1.4 Delete a Member

- The system asks for confirmation to delete a member..
- If the user selects yes, changes are made and the system updates the member's profile.



2.2.2 Alternative Flows

- **Invalid Member ID**:
  - If the member ID is not found, the system displays an error. The user can re-enter the ID or cancel the operation.
- **Unchanged Profile:**
  - If no changes are made, the system doesn't update the profile and informs the user.

2.3 Pre-Conditions

- User must be logged in with adequate privileges to modify member data.

2.4 Post-Conditions

- The selected member's profile is displayed to the user.

2.5 Special Requirements

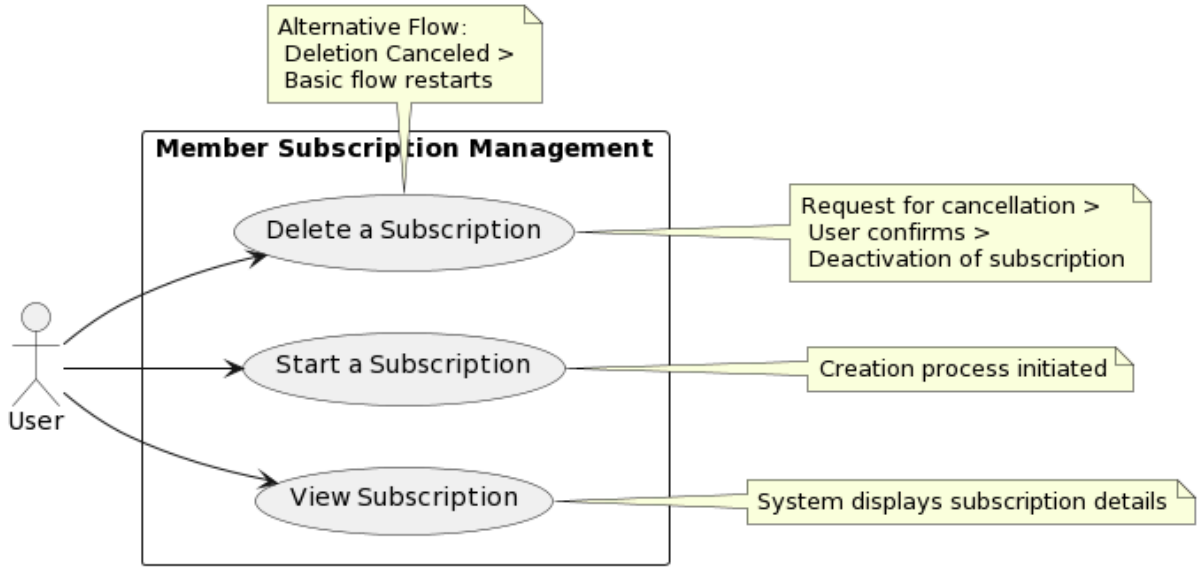
None

2.6 Extension Points

None

3. Member Subscription Management

3.0 Diagram



3.1 Brief Description

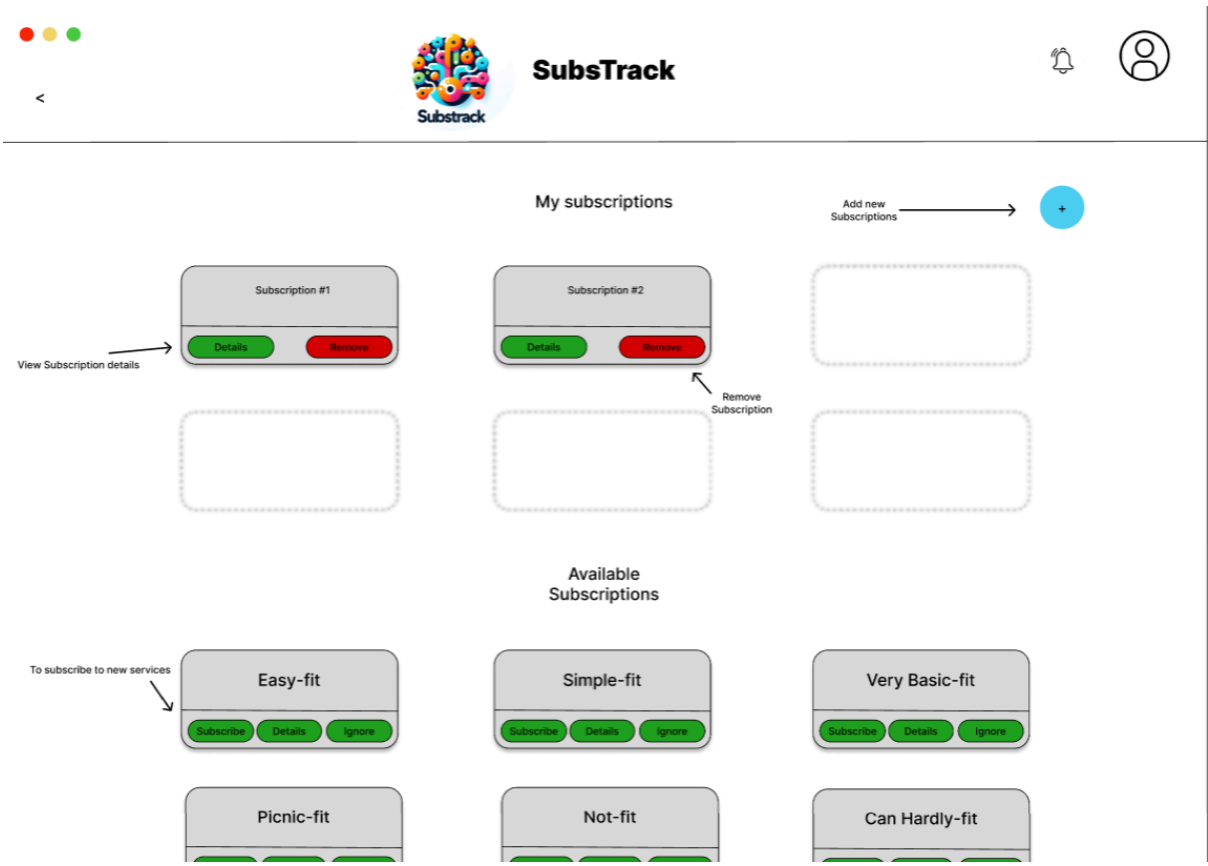
This use case describes the process of users managing their own subscription, including subscribing, updating, viewing, and deleting a subscription.

3.2 Flow of Events

3.2.1 Basic Flow

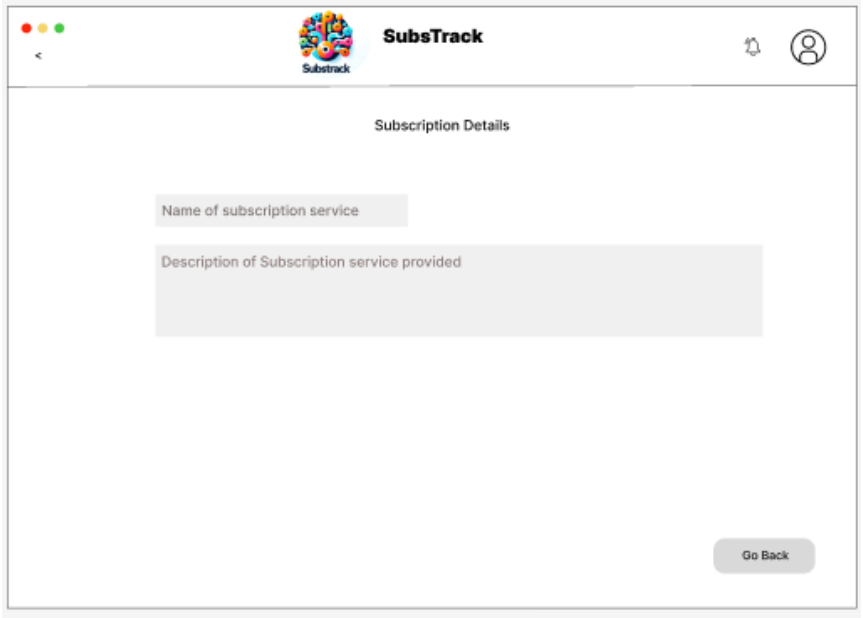
The process begins when a user requests to manage their subscriptions.

- The system prompts the user to select an operation: Create a subscription, view subscription, update subscription, and delete subscription..
- Based on the selection, the corresponding sub-flow is executed:
    - If "Start a Subscription”, the creation process is initiated.
    - If "View a subscription" is selected, the system displays member subscriptions.
    - If "Delete a subscription" is chosen, the deletion process begins.



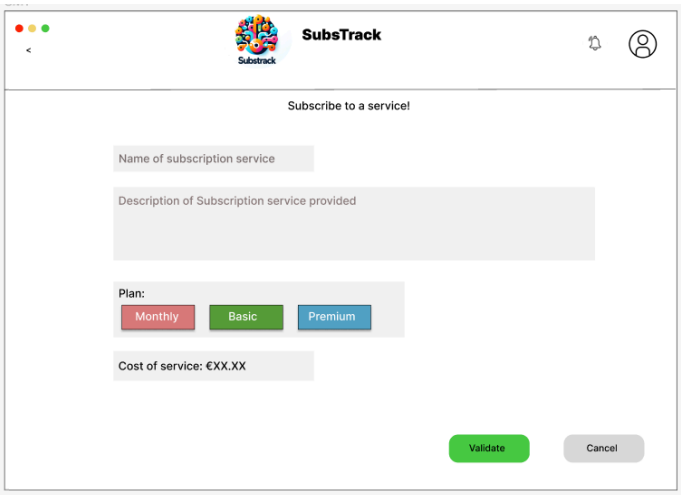
3.2.1.1 View Subscription

- Initiation: User chooses to view their subscription details.
- Display: The system displays the user's current subscription information, including plan details, duration, and any other relevant information.



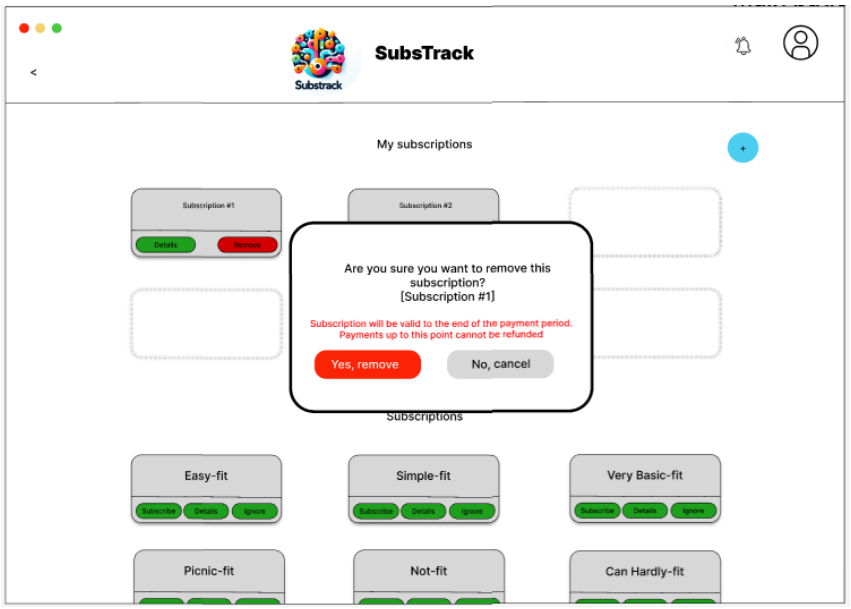
3.2.1.2 Join Subscription

- Selection: User selects the option to join a subscription.
- Plan Options: The system presents available subscription plans.
- Confirmation: The system updates the subscription details and confirms the changes to the user.



3.2.1.3 Delete a subscription

- Request for Deletion: The process starts when the user chooses to delete their subscription.
- Confirmation: The system requests confirmation from the user to proceed with the deletion.
- Deletion Process: Once the user confirms, the system processes the cancellation and deactivates the subscription.



3.2.2 Alternative Flows

- **Deletion Canceled**:
    - If the user decides not to delete a subscription, the operation is canceled, and the basic flow restarts.

3.3 Special Requirements

None

3.4 Pre-Conditions

- The user must be logged in with adequate privileges to modify member data.

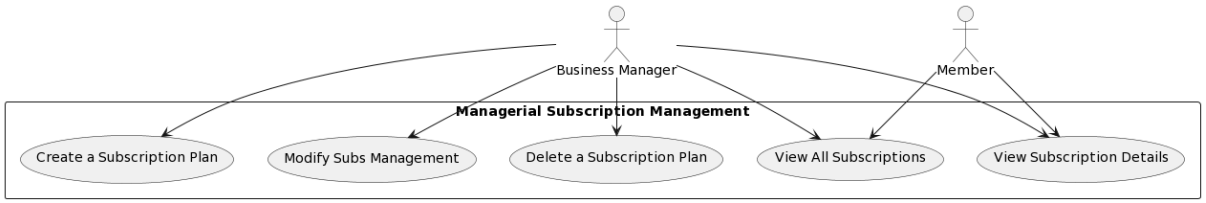
3.5 Post-Conditions

- Depending on the operation, a member subscription is either created, updated, viewed, or deleted. If the operation fails, there is no change in the system.

3.6 Extension Points

None

4. Managerial Subscription Management



4.1 Brief Description

It helps the business manager to manage the different client subscriptions easily.

4.2 Flow of events

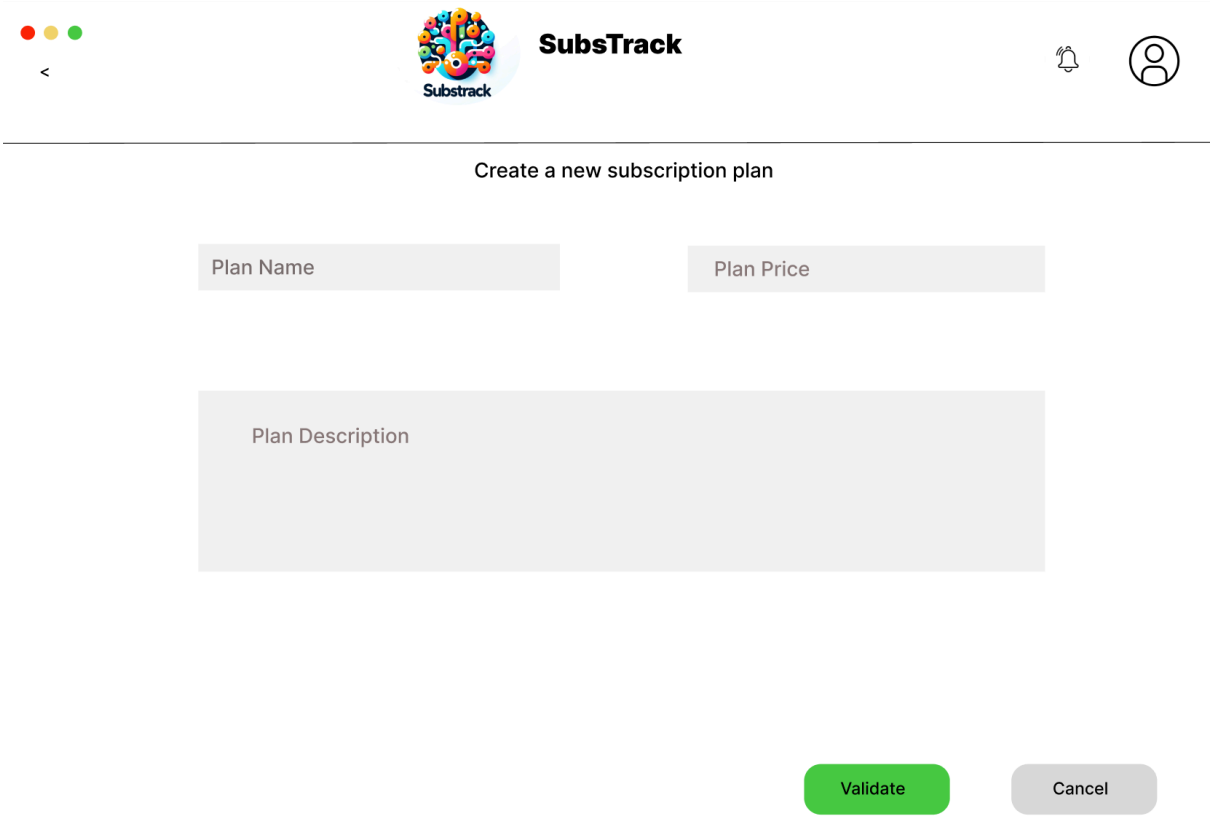
4.2.1 Basic Flow

This process starts whenever a manager decides to manage a client's subscription. It gives the manager the following options:

a. Create a subscription plan: If chosen, the process to create a new subscription plan is initiated.
b. View all subscription plans: If chosen, all the existing subscription plans will be shown.
c. Modify subs management: If chosen, modify the plan of a member.
d. Delete a subs plan: If chosen, the deleting option would appear for the existing plans.
e. View the details of a subscription plan: If chosen, the details of the chosen subscription plan would be shown.

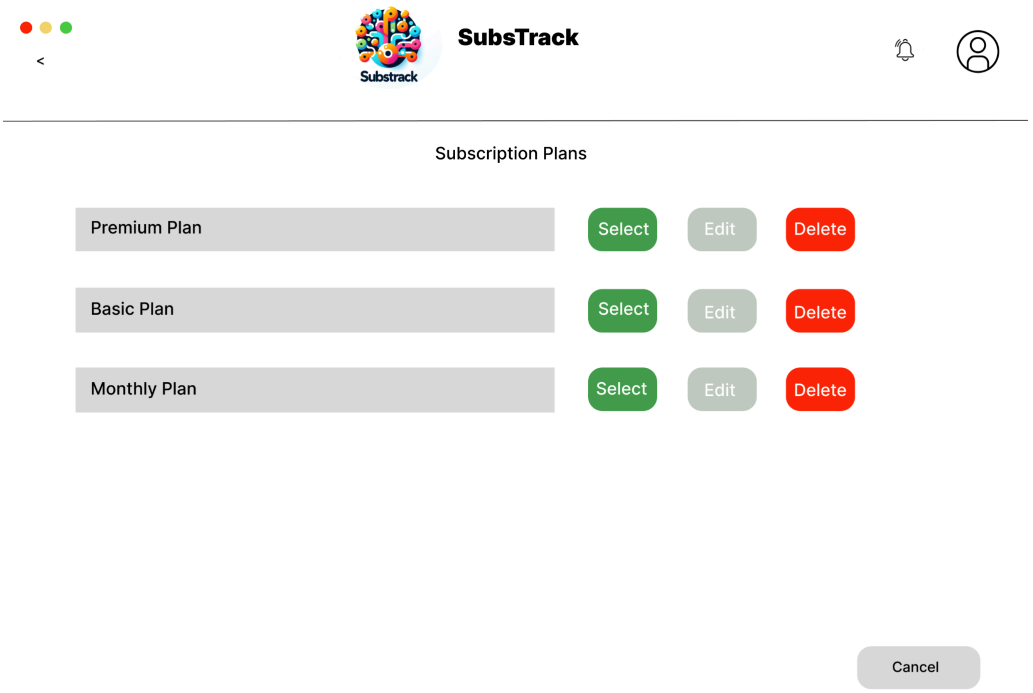
4.2.1.1 Create a subscription plan

- The system prompts the administrator to input details such as plan name, pricing, description and activities.
- Once the administrator provides the necessary information, the new subscription plan is created and stored in the system.



4.2.1.2 View all subscription

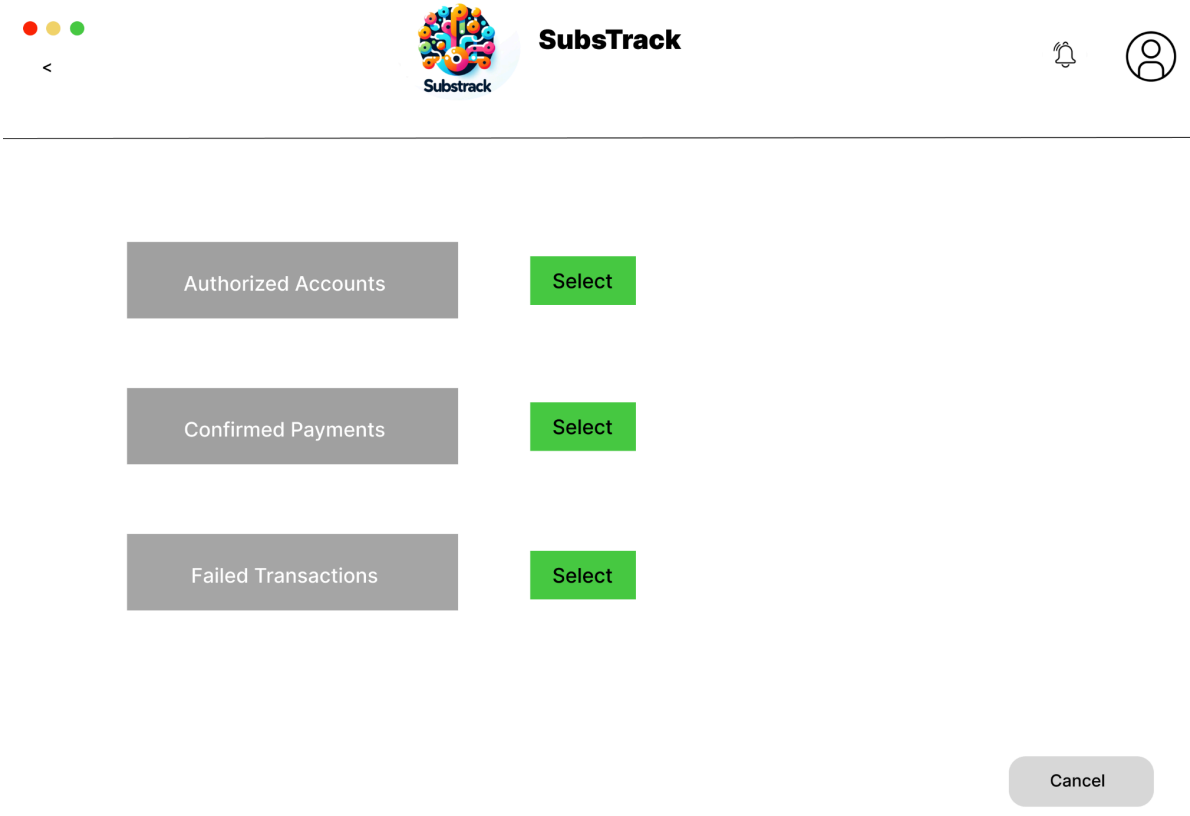
- The system displays a comprehensive list of all existing subscription plans.
- Administrators can review and access detailed information about each subscription plan in the system.



The "Edit" and "Delete" buttons will only be shown to authorized personnel

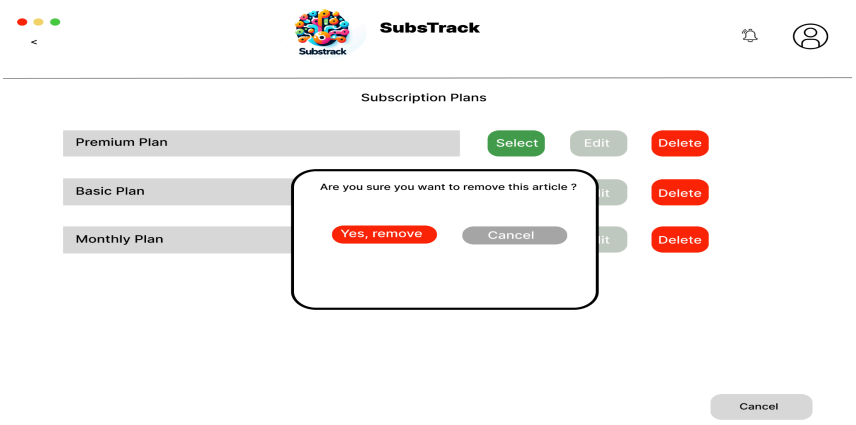
4.2.1.3 Modify subs management

- Administrators can access a set of tools to modify subscription-related settings and configurations.
- Upon making modifications, the system updates the subscription management parameters accordingly.



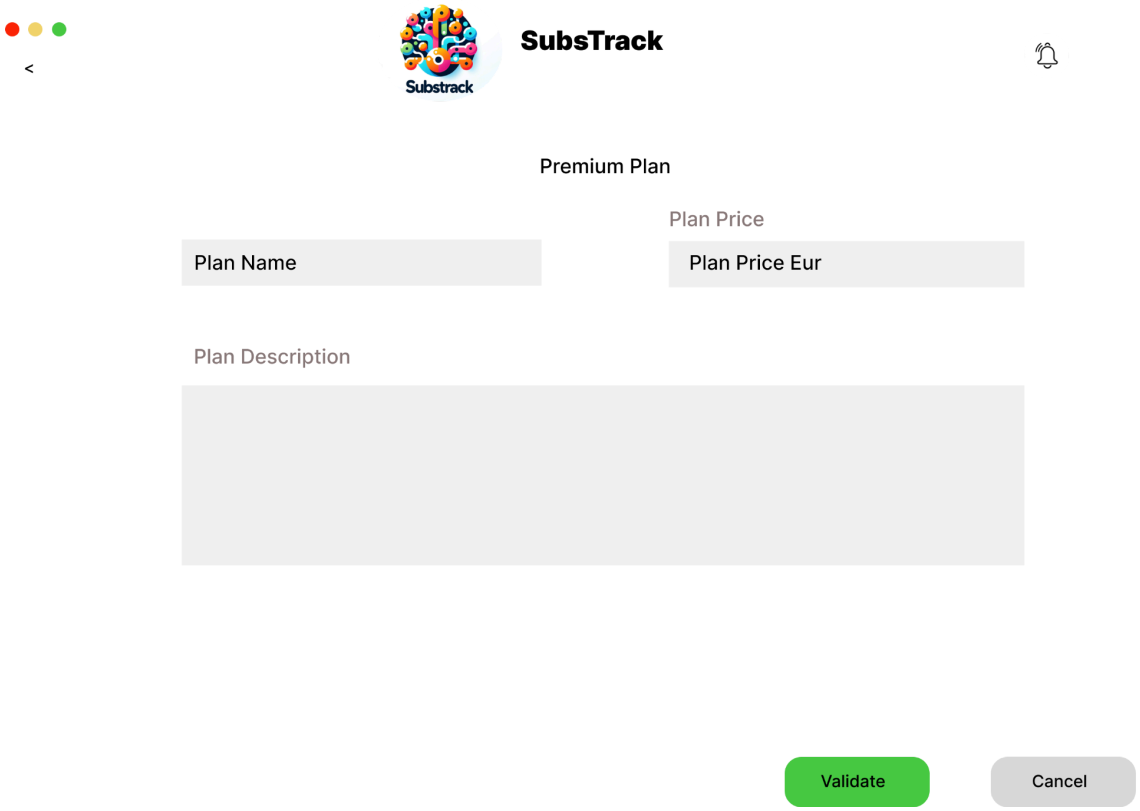
4.2.1.4 Delete a subscription plan

- The system provides administrators with the option to delete a specific subscription plan.
- Upon confirmation, the system removes the selected subscription plan from the database.



4.2.1.5 View the details of a subscription plan

- Administrators can select a specific subscription plan to view detailed information.
- The system presents comprehensive details about the selected subscription plan, including its features, pricing, and other relevant information.



4.2.2 Alternative Flow

Incorrect/Invalid Login:

- The system will show an error regarding the login. The user can either try again or cancel the process.

4.3 Special Requirements

- The system must enforce secure access and authentication for business manager.
- Subscription data should be stored securely with appropriate encryption measures.

4.4 Pre-Conditions

- Business manager must be authenticated before accessing managerial subscription management features.
- The system database must be operational and accessible.

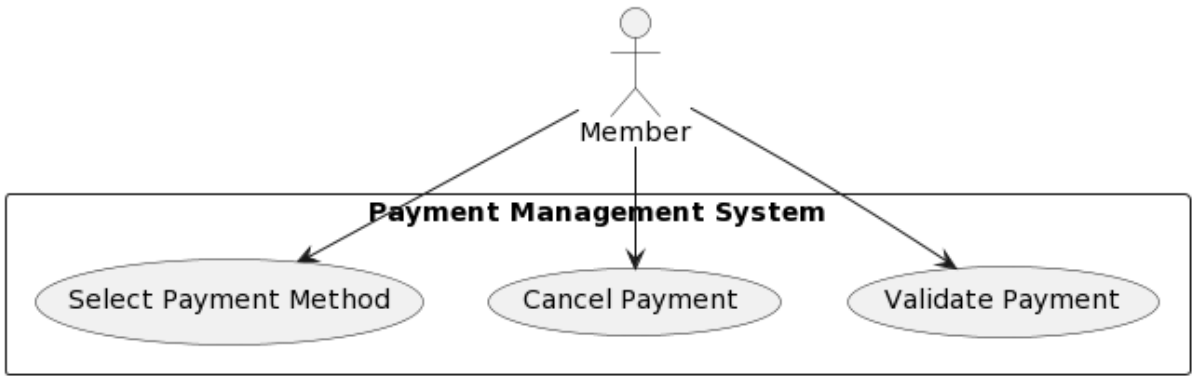
4.5 Post-Conditions

- Updated subscription data is reflected in the system after successful creation, modification, or deletion of a subscription plan.
- Business manager receive feedback after completing subscription management tasks.

4.6 Extension Points

- The system should be scalable to accommodate additional subscription-related functionalities in the future.
- Integration with external systems for advanced analytics or reporting capabilities.

5. Payment Management



5.1 Brief Description

This use case allows the users to manage his payment whenever he wishes to place an order after validating his subscription. This includes validating or canceling a payment and selecting a payment method.

5.2 Flow of Events

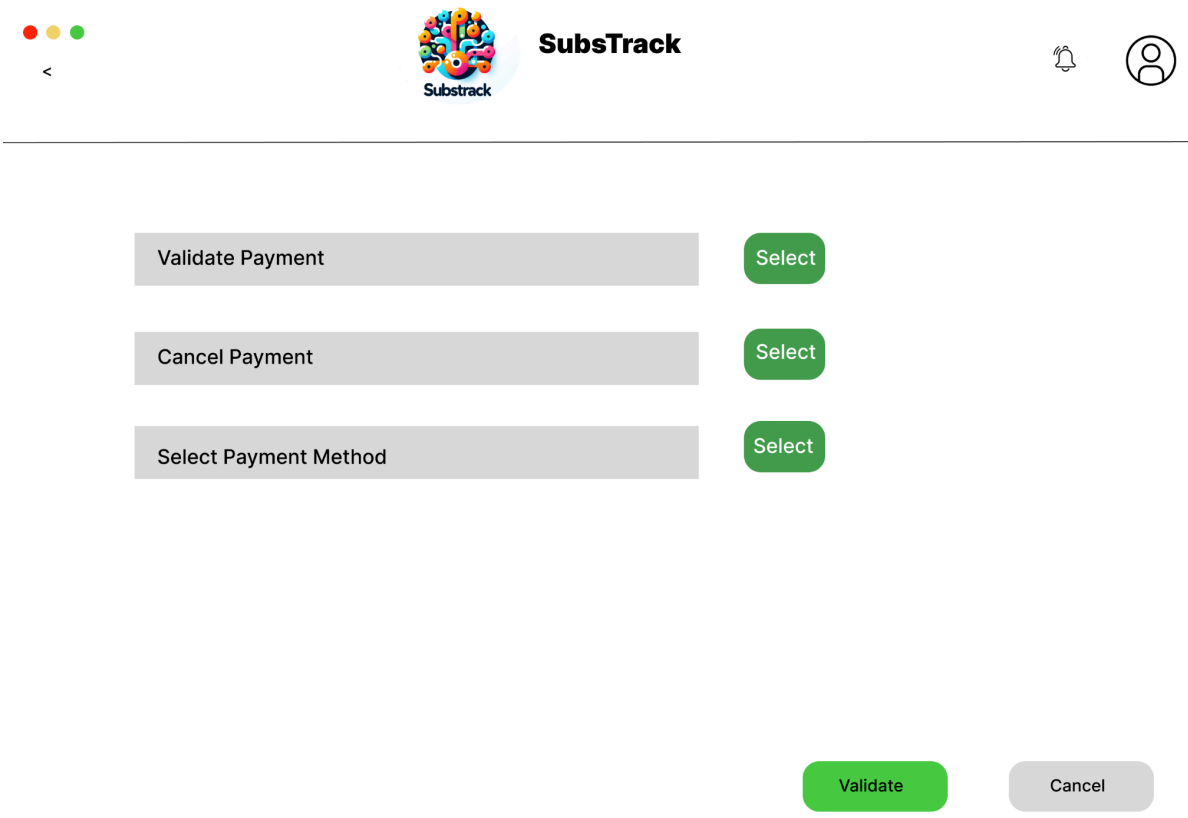
5.2.1 Basic Flow

This use case starts when the user wishes to validate or cancel a payment after validating his subscription plan.

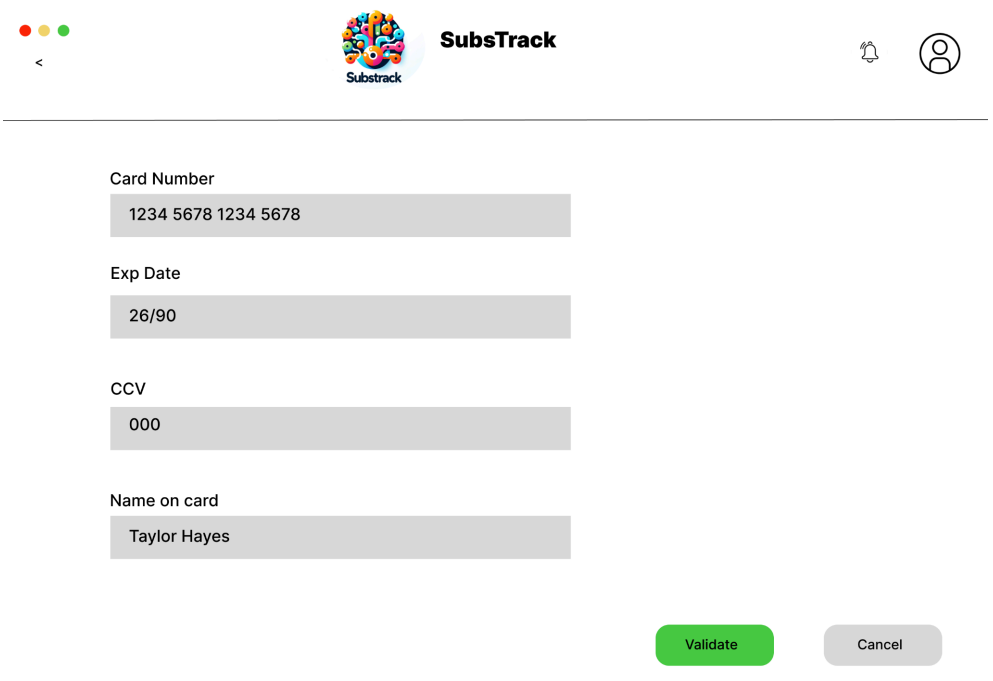
1. The system requests that the user specify the function he/she would like to perform, either :
    - Validate the payment
    - Cancel the payment
    - Select payment method

2. Once the user provides the requested information, one of the sub flows is executed.

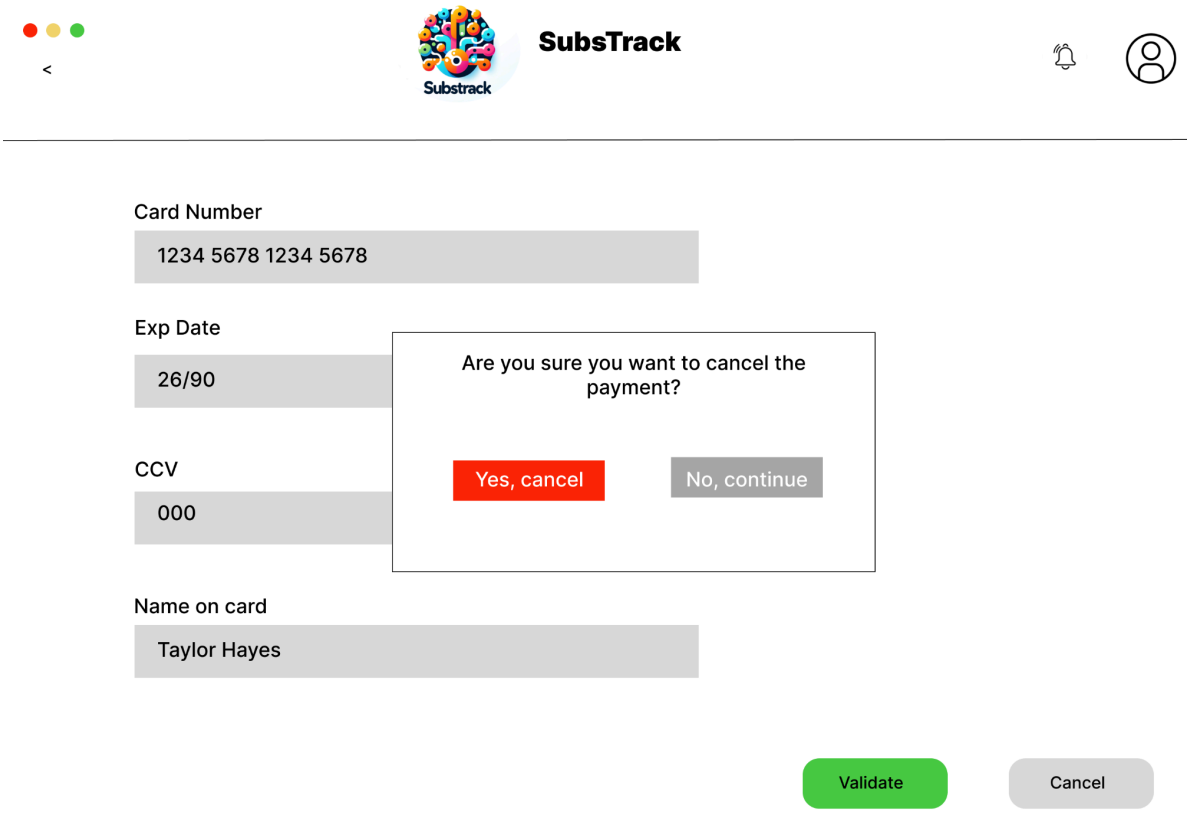
- If the user selects “Validate the payment“, the Validate the payment subflow is executed.
- If the user selects “Cancel the payment”, the Cancel the payment subflow is executed.
- If the user selects “Add payment method“, the Select payment method subflow is executed.



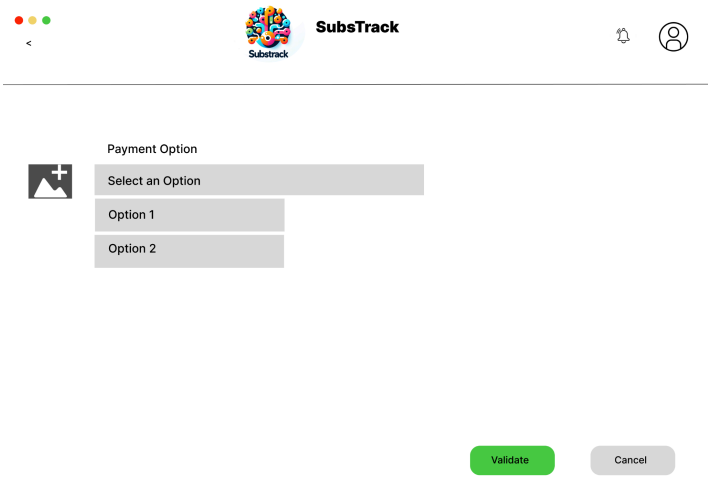
- *Validate the Payment*
  - The user initiates the payment validation process.
  - The system verifies the payment details and ensures they meet the required criteria.



- *Cancel the Payment*
  - The user has the option to cancel an ongoing payment transaction.
  - The system cancels the payment and updates the transaction status.



- *Select a Payment Method*
  - The different payment options are shown to the user.
  - The user chooses a payment method from available options.
  - The system processes the payment using the selected method.



5.2.2 Alternative Flow

Problem with a payment:

- If there is any problem that makes the payment invalid, the system will show an error. The user can either try again or cancel the payment process.

5.3 Special Requirements

- The system must ensure the security and confidentiality of payment information.
- Integration with secure and reliable payment gateways is required.

5.4 Pre-Conditions

- Users must be logged in before initiating any payment-related operations.
- The system must have access to the payment methods.

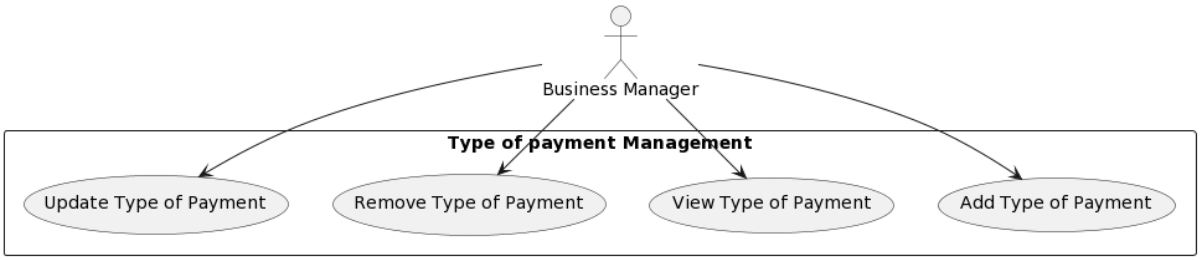
5.5 Post-Conditions

- The system updates the payment status after validating the transaction.
- Users receive confirmation after completing a successful payment.

5.6 Extension Points

- The system should be scalable to add different payment methods in the future.

6. Type of Payment management



6.1 Brief Description

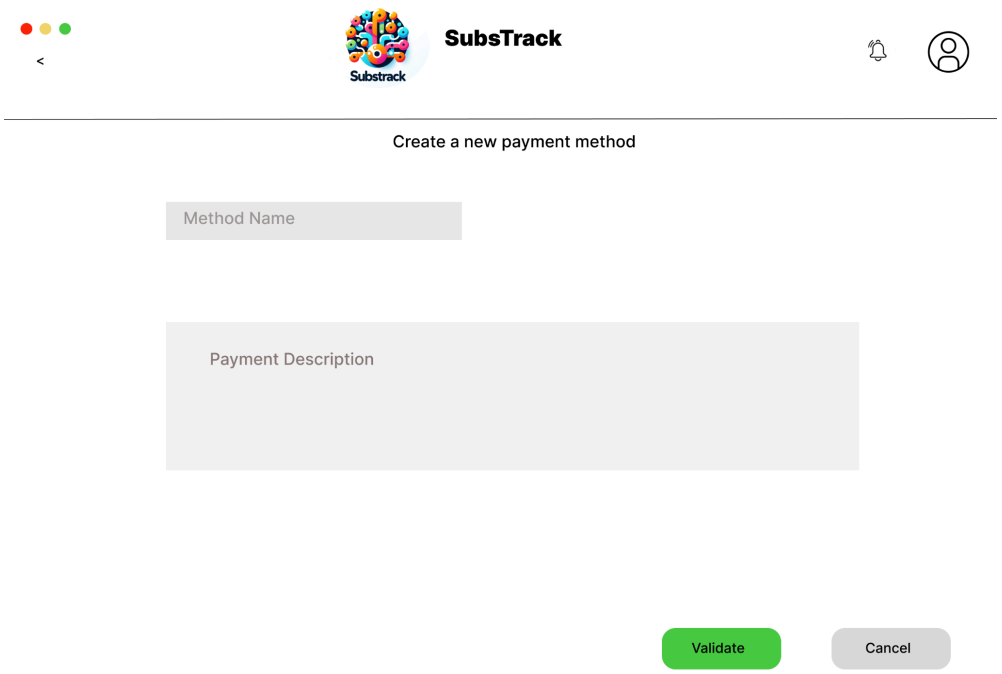
It allows the business manager to add new payment types, view existing types, remove outdated ones, and update information related to specific payment methods.

6.2 Flow of Events

6.2.1 Basic Flow

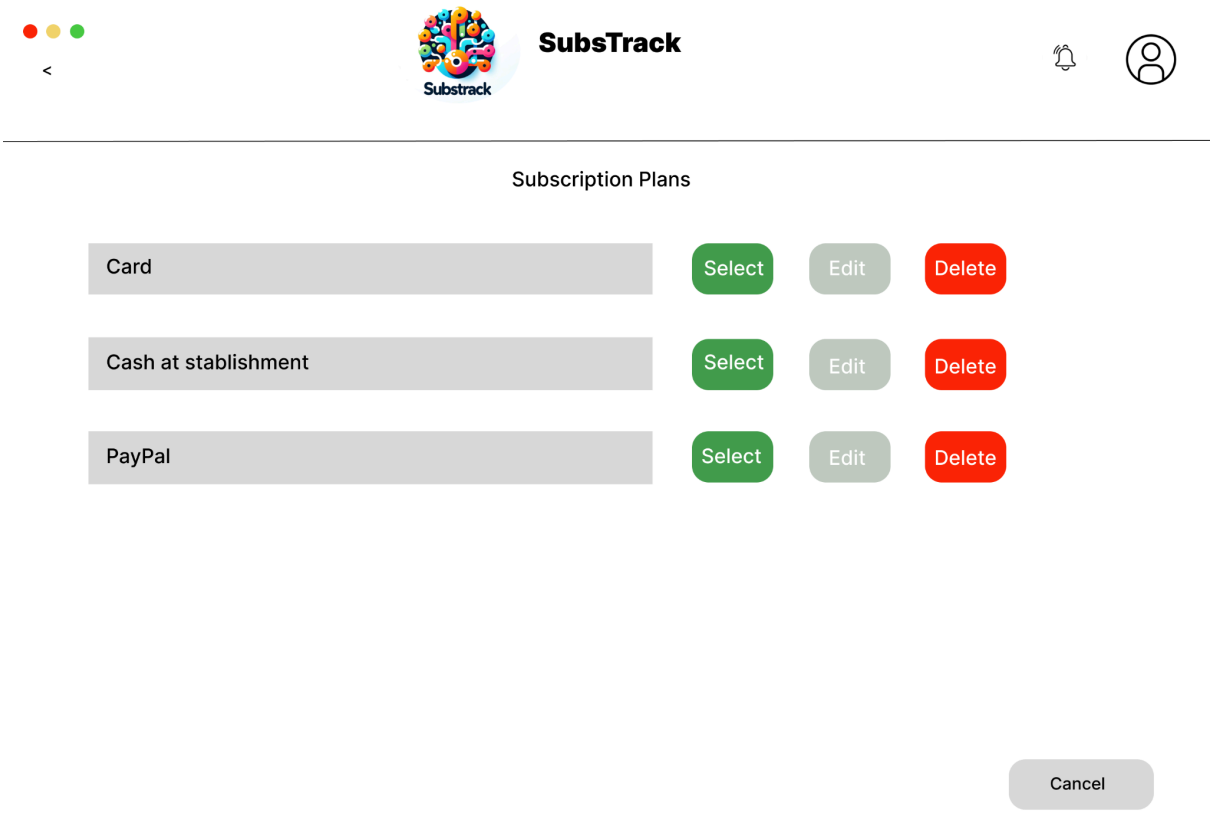
6.2.1.1 Add type of payment

- Business manager initiate the process of adding a new type of payment.
- The system asks to give details such as the name and specifications of the new payment type.
- The system adds the new payment type to the system.



6.2.1.2 View type of payment

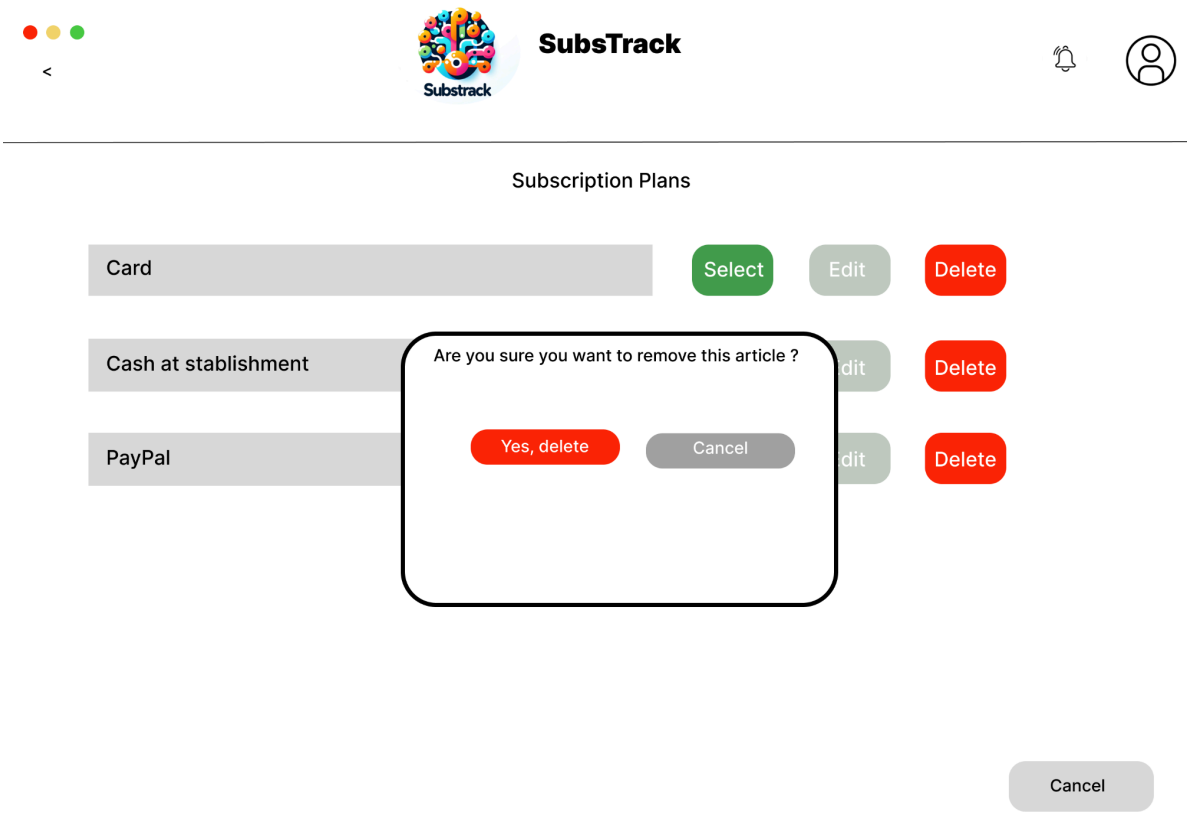
- User asks for the payment methods available
- The system presents a list of payment types with their respective details.



The "Edit" and "Delete" buttons will only be shown to authorized personnel

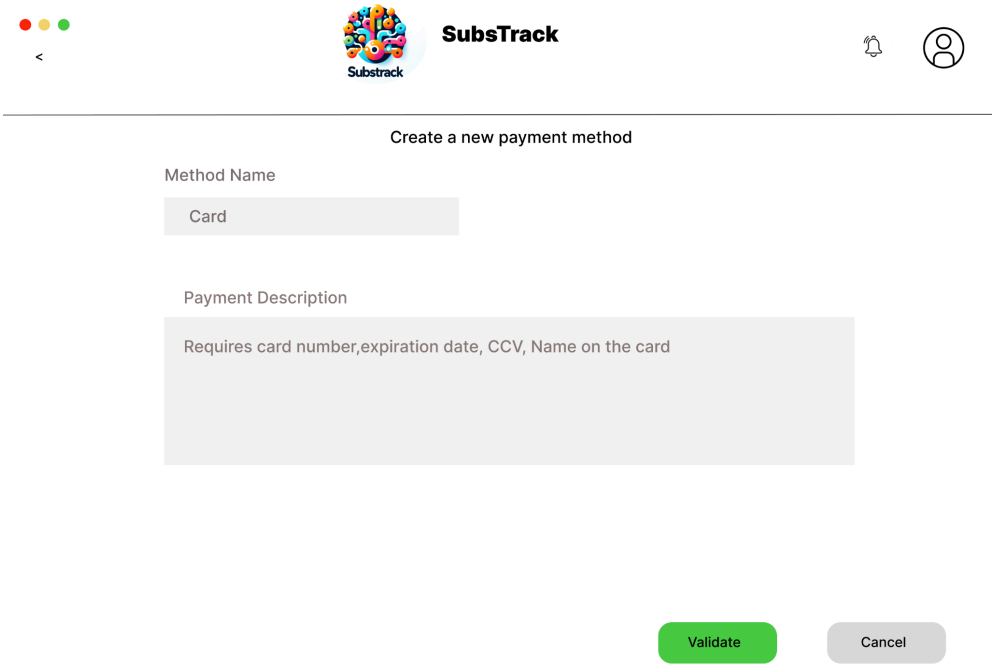
6.2.1.3 Remove type of payment

- Authorized personnel have the option to remove payment types.
- The system asks the users to confirm the removal.
- The system deletes the specified payment type from the system.



6.2.1.4 Update type of payment

- Users can modify information related to a specific payment type.
- The system allows users to update details such as the name or different specifications of the payment type.



6.2.2 Alternative Flow

- **Invalid Login:**
    - If the login used does not belong to an authorized person or has any mistake, the system will let the user know about it. The user can cancel the process or try again.

6.3 Special Requirements

- Access to the Type of Payment Management system should be restricted to authorized users.

6.4 Pre-Conditions

- Users must be logged in with an approved account to manage types of payment.

6.5 Post-Conditions

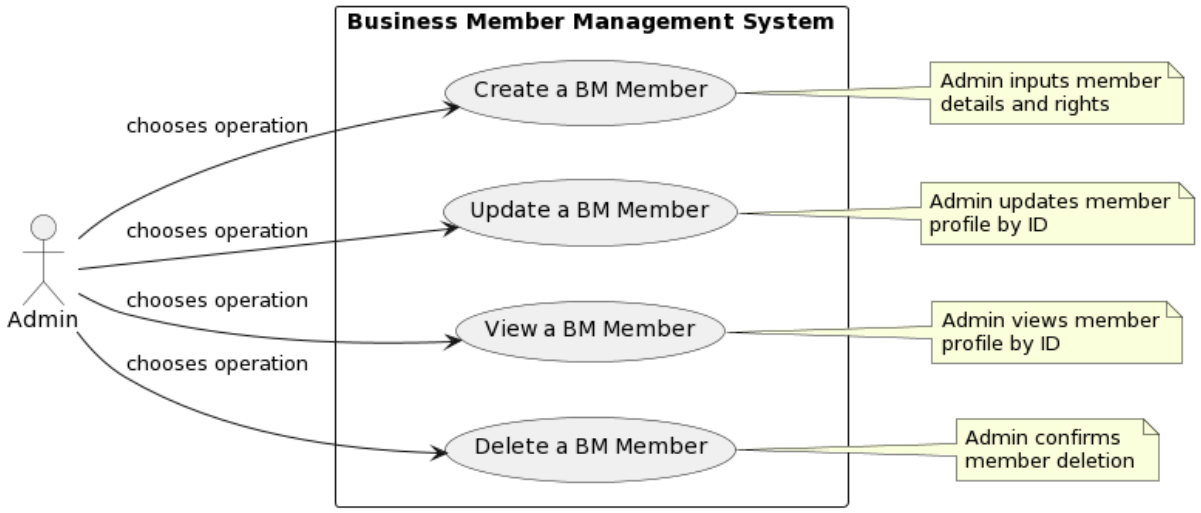
- The system updates the list of payment types after successful addition, removal, or update.
- Users receive confirmation after completing the type of payment successfully.

6.6 Extension Points

- The system should be scalable to accommodate additional attributes or features for payment types.

7. Business Member Management

7.0 Diagram



7.1 Brief Description

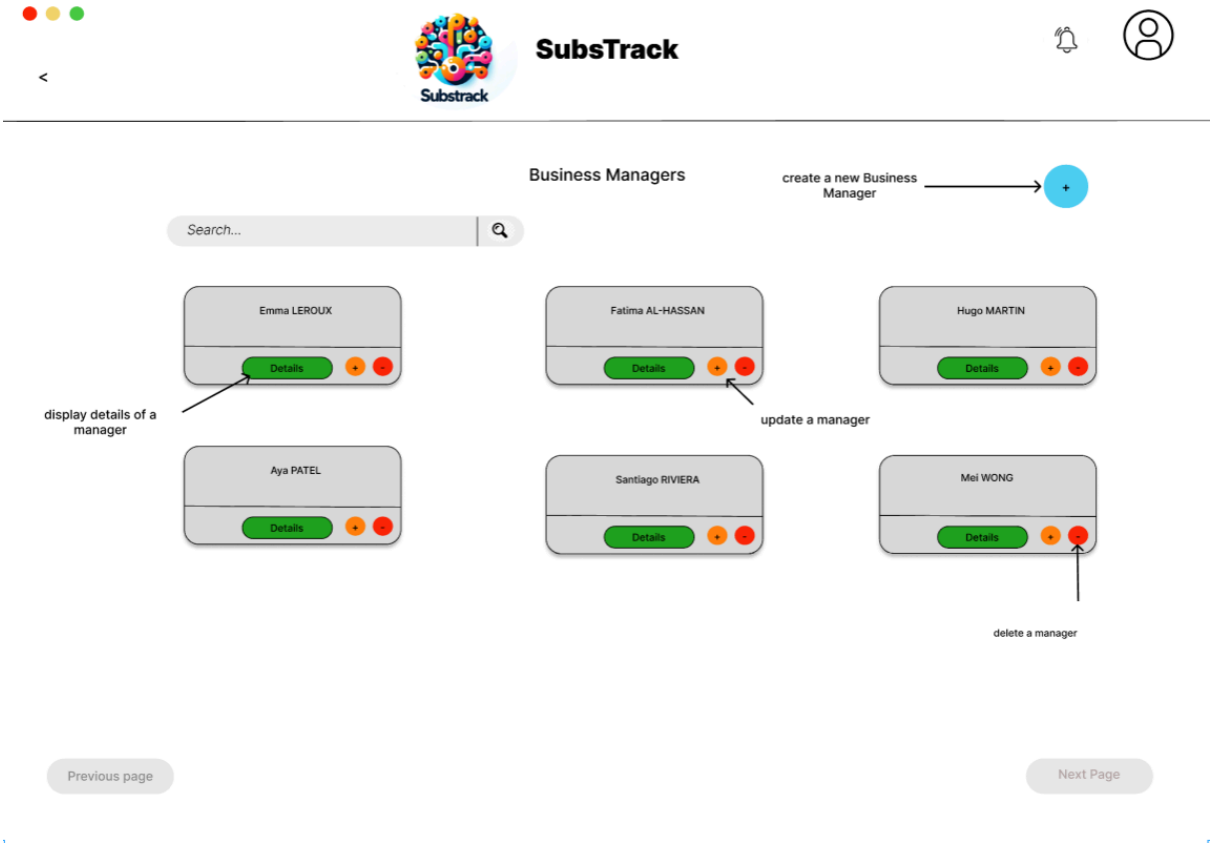
This use case describes the process of managing business members within a system, including creating, updating, viewing, and deleting business member profiles, along with assigning specific rights to them.

7.2 Flow of Events

7.2.1 Basic Flow

The process begins when a system administrator wishes to manage business member profiles.

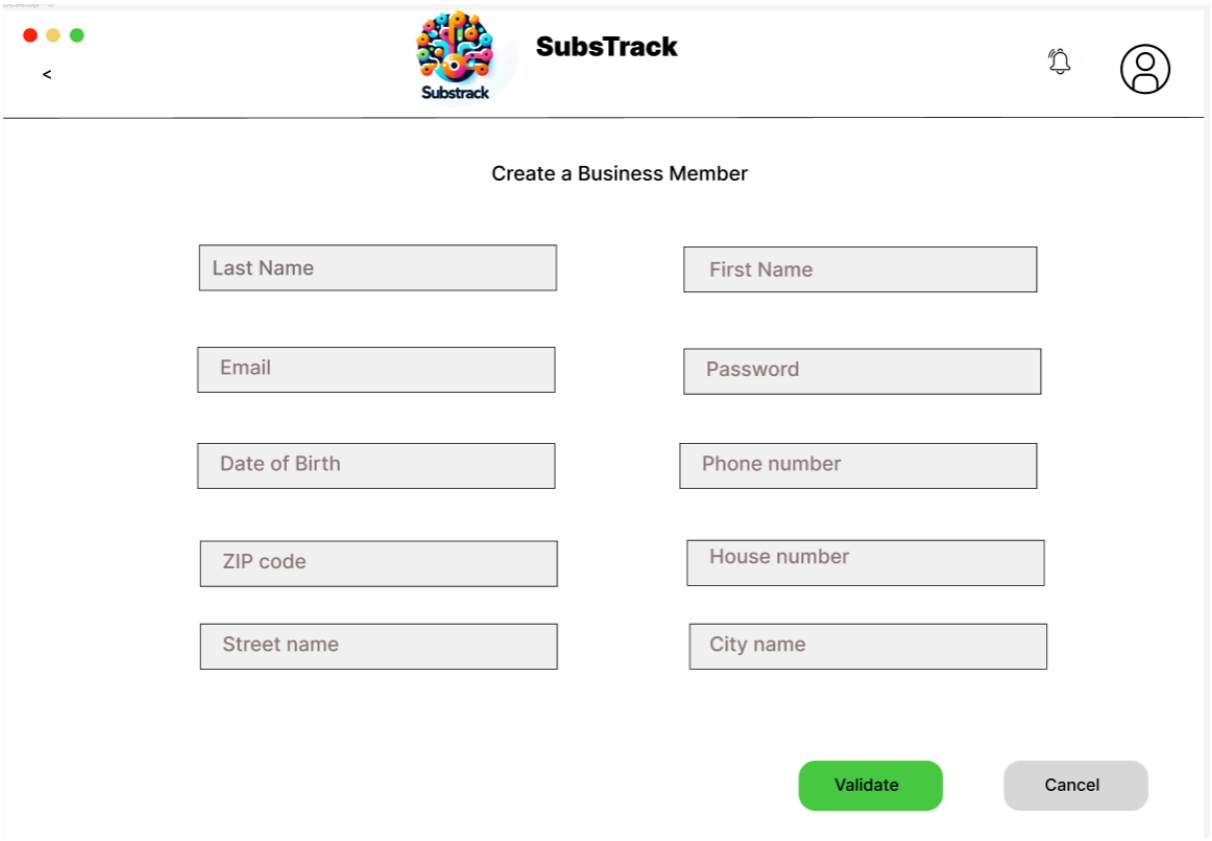
- The system prompts the administrator to select an operation: Create a “Business Manager” Member, Update a “Business Manager” Member, View a “Business Manager” Member, or Delete a “Business Manager” Member.
- Based on the selection, the corresponding sub-flow is executed:
    - If "Create a BM Member" is selected, the creation process is initiated.
    - If "Update a BM Member" is chosen, the system proceeds to update a member.
    - If "View a BM Member" is selected, the system displays member profiles.
    - If "Delete a BM Member" is chosen, the deletion process begins.



The “update”, “delete”, and “create” buttons are will only be shown to authorized personnel

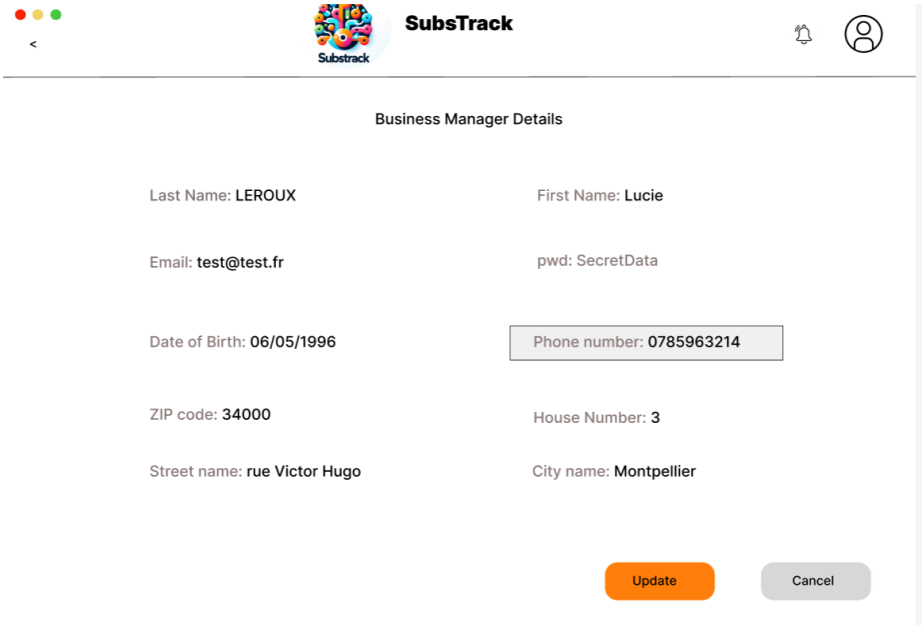
7.2.1.1 Create a BM Member

- The system requests details such as name, contact information, and the specific rights to be assigned.
- Once the administrator inputs the information, the new business member is added to the system, and the rights are assigned accordingly.



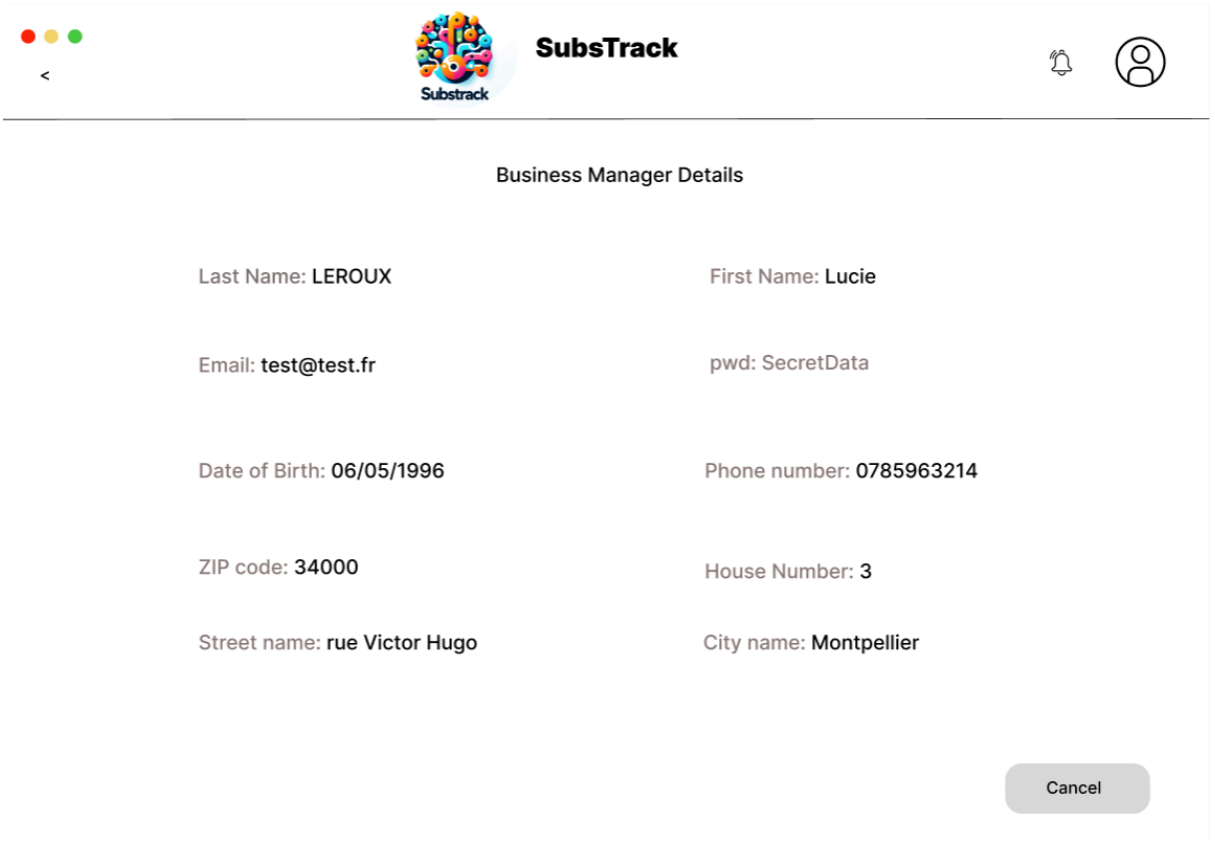
7.2.1.2 Update a BM Member

- The system asks for the member ID to be updated.
- The administrator enters the ID, and the system displays the current member details.
- Changes are made as needed, and the system updates the member's profile.



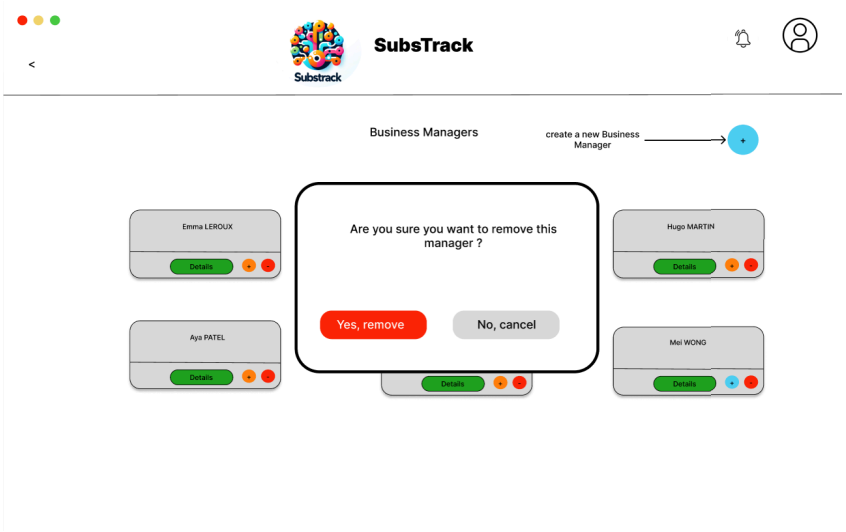
7.2.1.3 View a BM Member

- The administrator selects the member profile to view.
- The system retrieves and displays the selected member's details.



7.2.1.4 Delete a BM Member

- The system requests the member ID for deletion.
- After the administrator inputs the ID, the system asks for confirmation to delete.
- Once confirmed, the system removes the member's profile.



7.2.2 Alternative Flows

- **Invalid Member ID**:
  - If a member with the specified ID does not exist, the system displays an error. The administrator can then enter a different ID or cancel the operation.
- **Deletion Canceled**:
  - If the administrator decides not to delete a member, the operation is canceled, and the basic flow restarts.

7.3 Special Requirements

- Rights and permissions assignment functionality.
- Secure handling of personal and sensitive member data.

7.4 Pre-Conditions

- The administrator must be logged into the system with sufficient privileges to perform these operations.

7.5 Post-Conditions

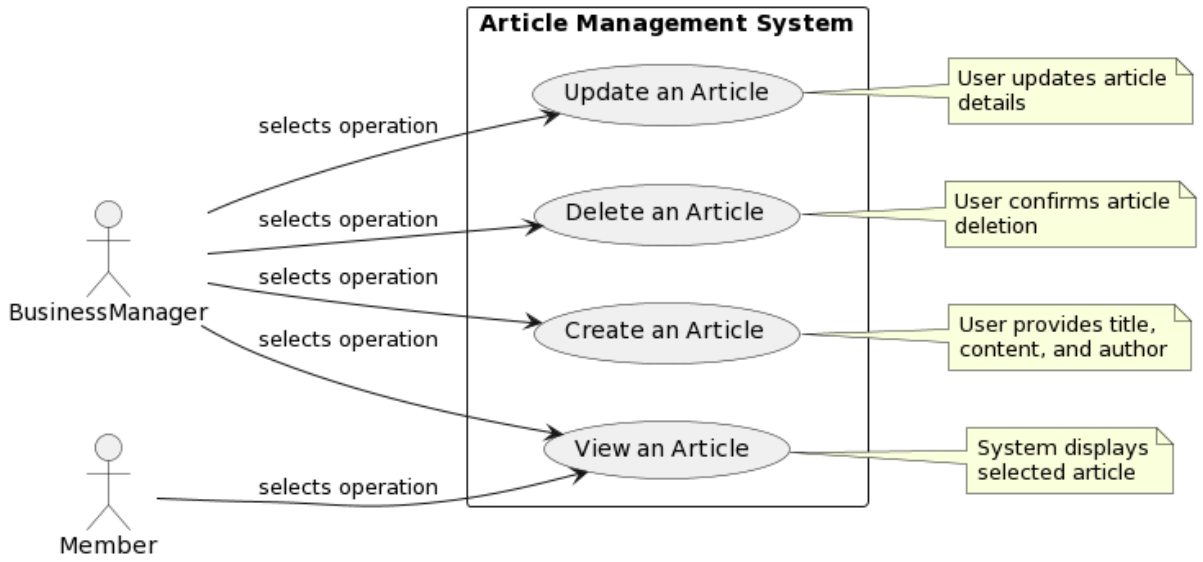
- Depending on the operation, a business member's profile is either created, updated, viewed, or deleted. If the operation fails, there is no change in the system.

7.6 Extension Points

- Integration with other business systems for extended member management capabilities.
- Linking member profiles to specific business roles or departments.

8. Articles Management

8.0 Diagram



8.1 Brief Description

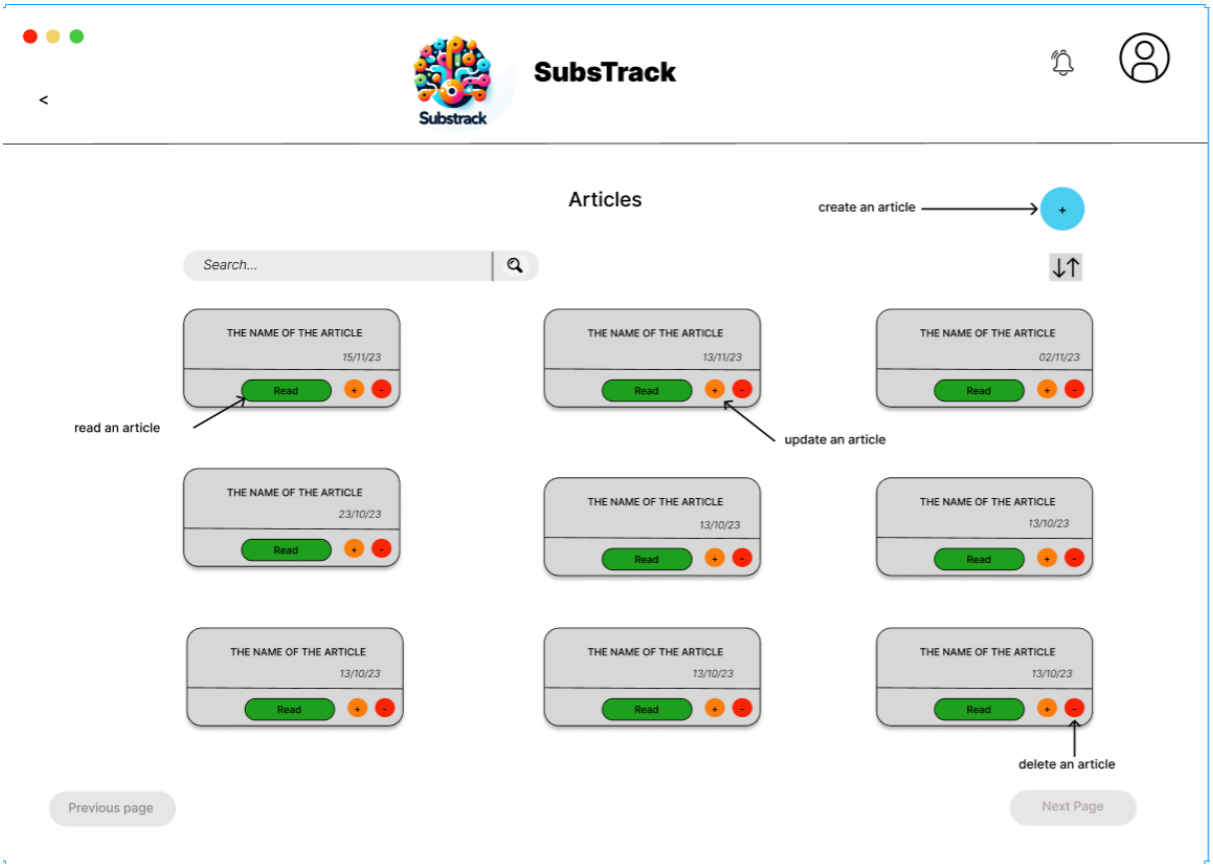
This use case involves managing articles within a system, encompassing creating, viewing, updating, and deleting articles.

8.2 Flow of Events

8.2.1 Basic Flow

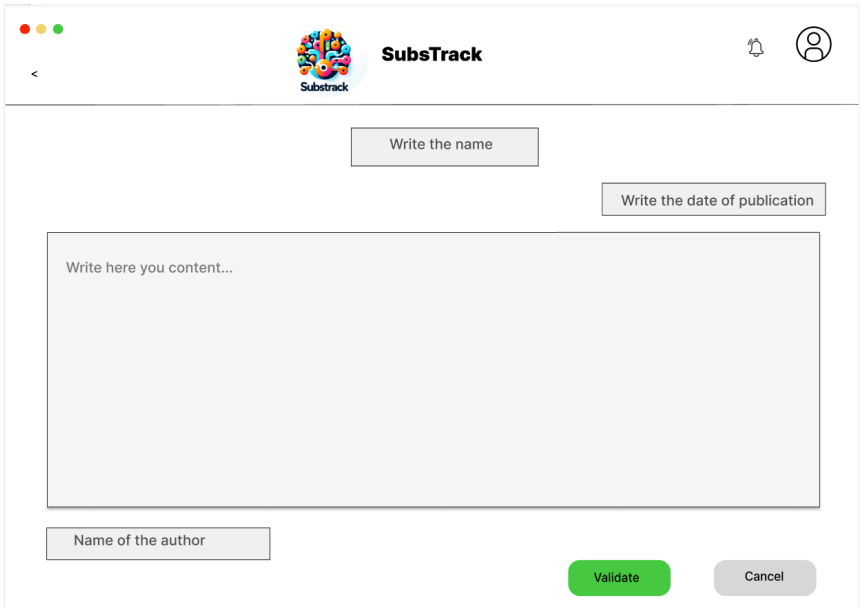
This use case initiates when a user wishes to perform any article management task.

- The system asks the user to select the desired operation: Create an Article, View an Article, Update an Article, or Delete an Article.
- Depending on the user's choice, the relevant sub-flow is executed:
    - If "Create an Article" is chosen, the system proceeds with the article creation process.
    - If "View an Article" is chosen, the system displays existing articles.
    - If "Update an Article" is selected, the system initiates the update process.
    - If "Delete an Article" is chosen, the system starts the deletion process.



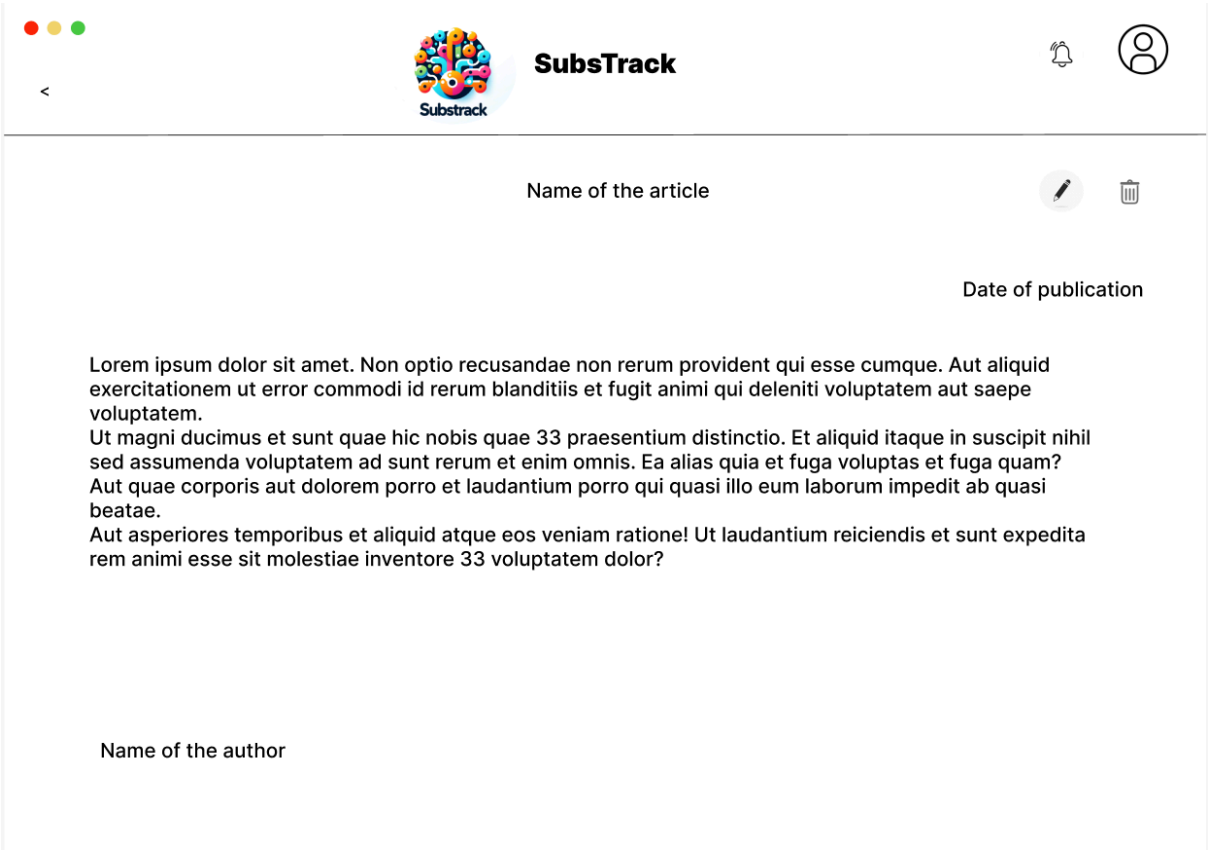
8.2.1.1 Create an Article

- The system requests details such as the article's title, content, and author.
- Once the user provides the information, the article is added to the system, and the user is notified of the successful creation.



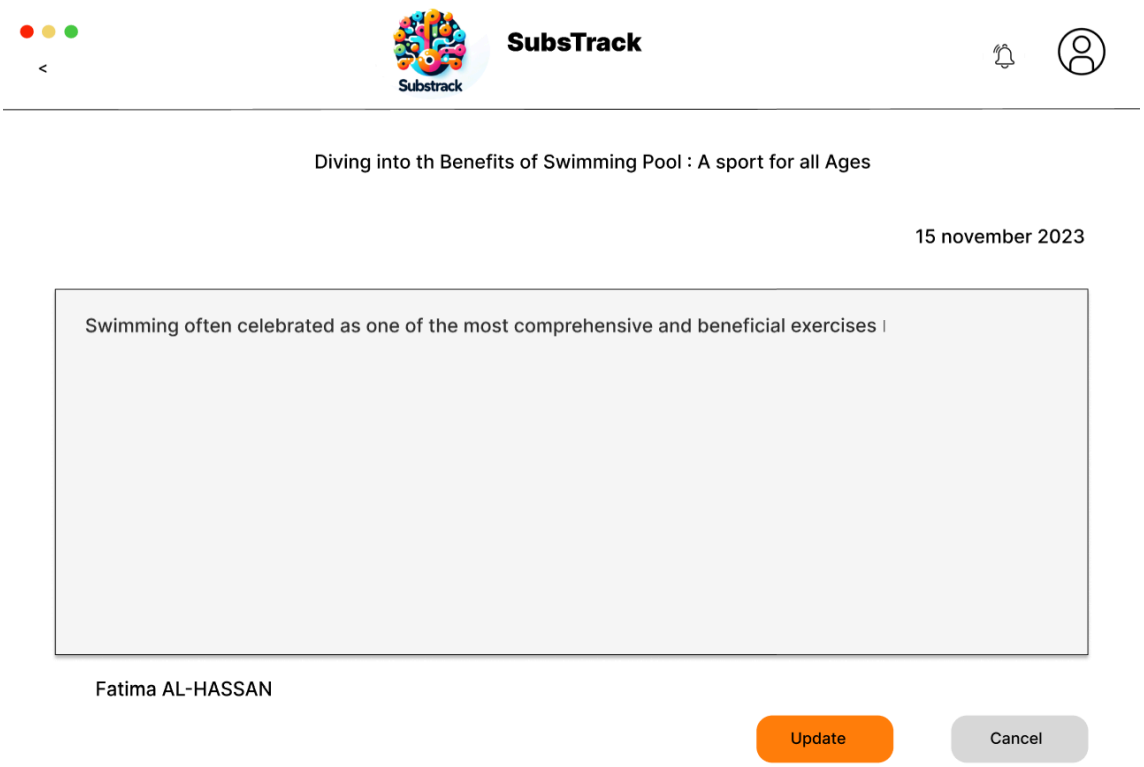
8.2.1.2 View an Article

- The user specifies the article to view.
- The system retrieves and displays the selected article.



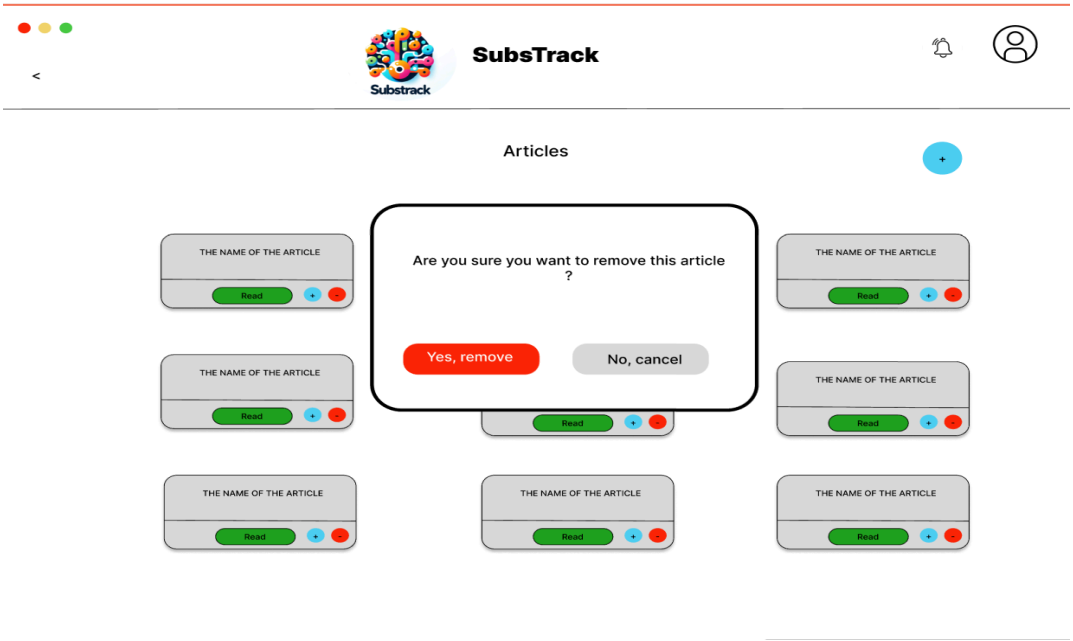
8.2.1.3 Update an Article

- The system asks for the article ID to be updated.
- The user enters the ID, and the system displays the current article details.
- The user makes changes as needed, and the system updates the article information.



8.2.1.4 Delete an Article

- The system requests the article ID for deletion.
- After the user inputs the ID, the system asks for confirmation to delete.
- Once confirmed, the system removes the article and notifies the user.



8.2.2 Alternative Flows

- **Invalid Article ID**:
    - If an article with the specified ID does not exist, the system displays an error. The user can then enter a different ID or cancel the operation.
- **Deletion Canceled**:
    - If the user decides not to delete an article, the operation is canceled, and the basic flow restarts.

8.3 Special Requirements

- Rich text editor for article creation and updates.
- Version control for tracking changes in articles.

8.4 Pre-Conditions

- The user must be logged into the system and have the necessary permissions for the chosen operation.

8.5 Post-Conditions

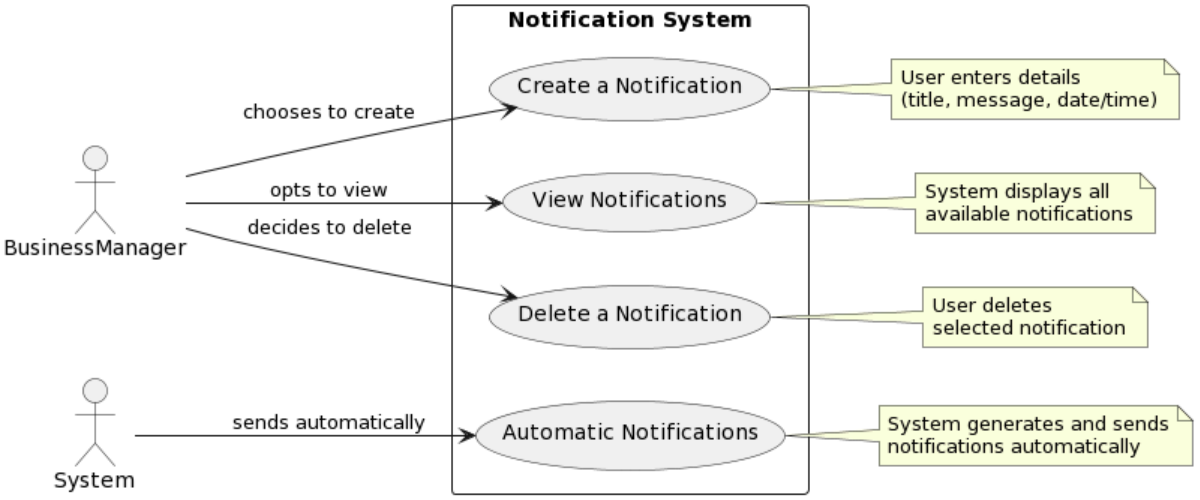
- Depending on the operation, the article is either created, displayed, updated, or deleted. If the operation is not successful, the system remains unchanged.

8.6 Extension Points

- Integration with a tagging system for categorizing articles.
- Linking articles to user profiles or other content within the system.

9. Notifications

9.0 Diagram



9.1 Brief Description

This use case outlines how users and the system interact with the notification system. It includes user-initiated actions such as creating, viewing, and deleting notifications, as well as system-initiated automatic notifications.

9.2 Flow of Events

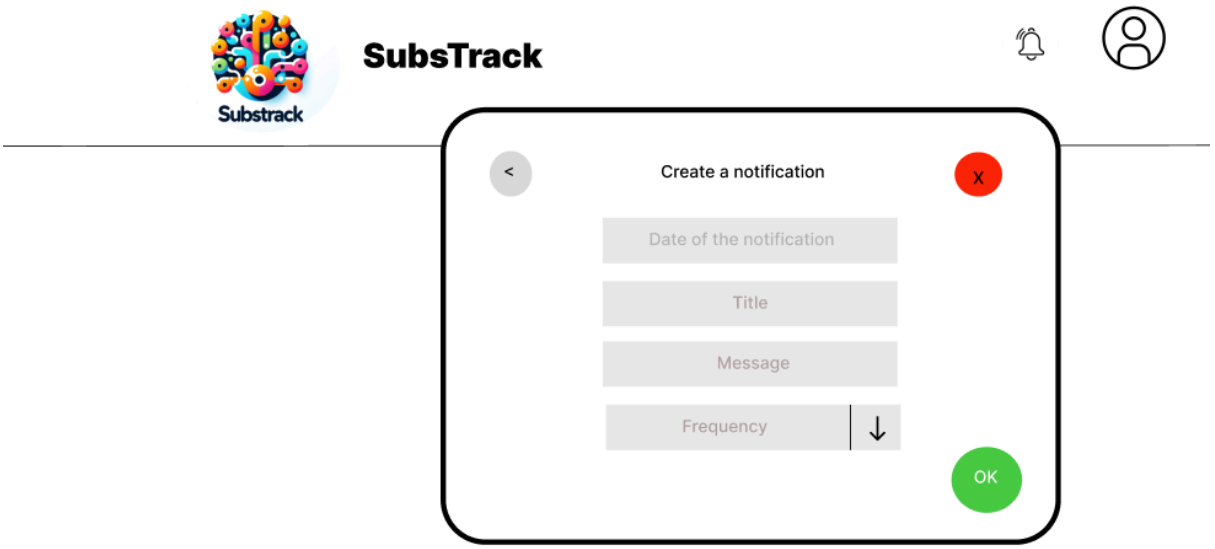
9.2.1 Basic Flow

- Creating Notifications:
  - Initiated when a user decides to create a notification.
  - The system requests details like title, message, date/time, and frequency.
  - The user provides this information.
  - The system records and confirms the creation.
- Viewing Notifications:
  - The user opts to view their notifications.
  - The system displays all available notifications.
- Deleting Notifications:
  - The user chooses to delete a specific notification.
  - The system removes the selected notification and confirms the deletion.
- Automatic Notifications:
  - Initiated by the system based on predefined criteria or triggers.
  - The system generates and sends notifications automatically without user input.

9.2.1.1 Create a Notification

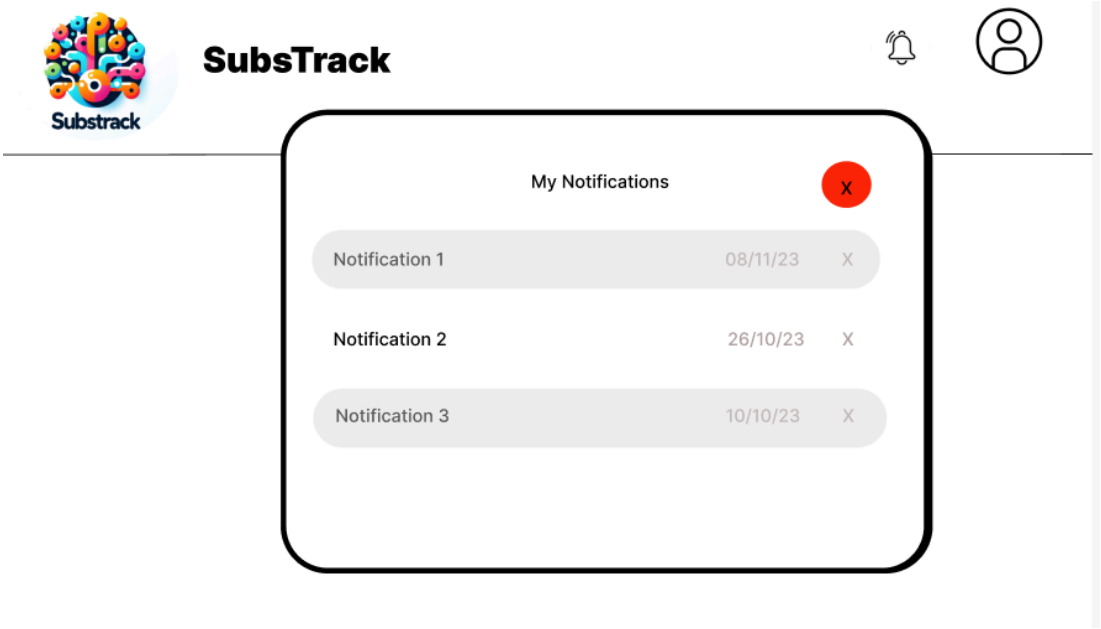
- The system prompts the user for necessary details.

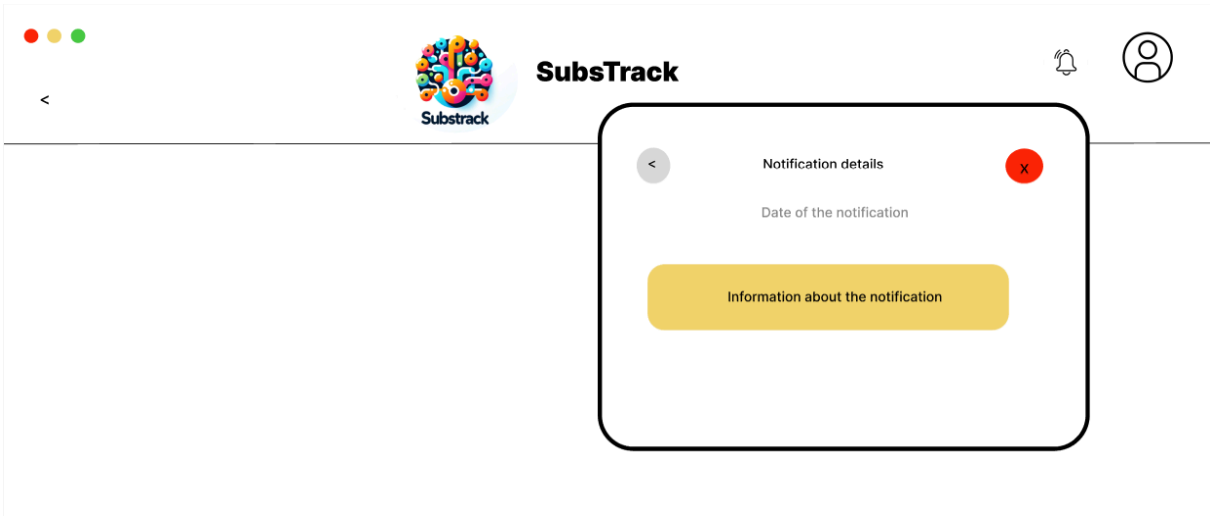
- Upon submission, the notification is added to the system.



9.2.1.2 View a Notification

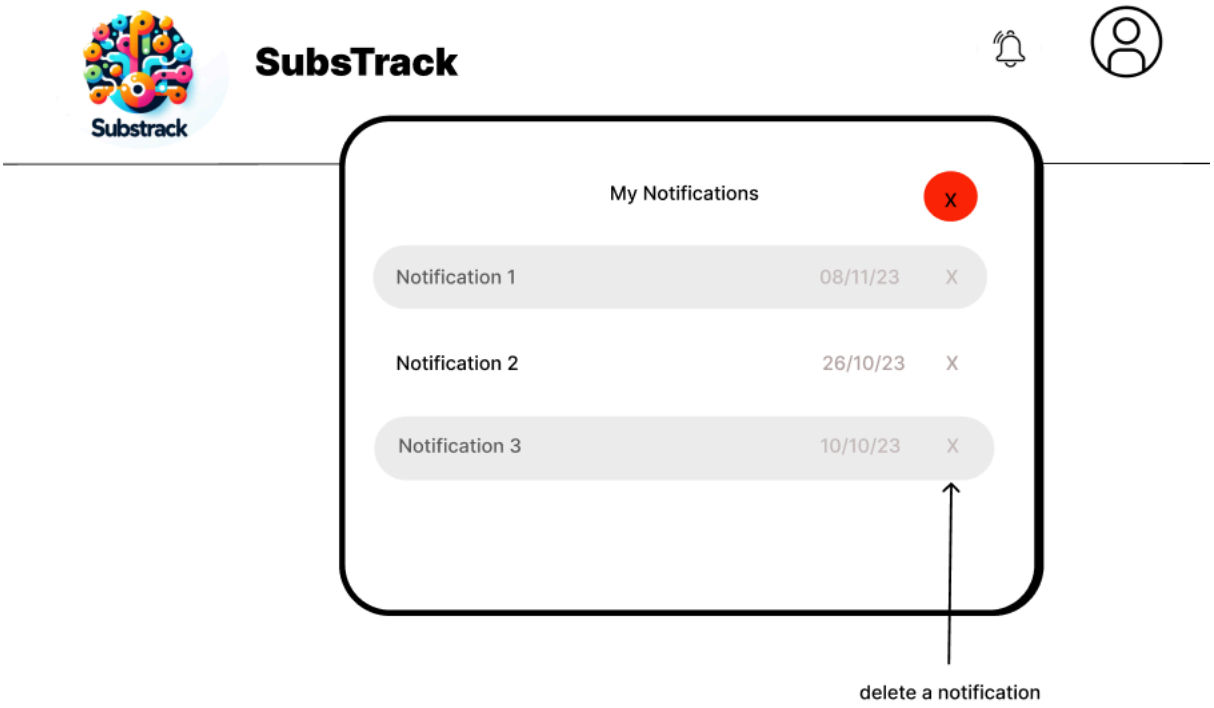
- The user requests to view notifications.
- The system retrieves and displays all available notifications.





9.2.1.3 Delete a Notification

- The user selects a notification to delete.
- The system confirms the deletion and proceeds.

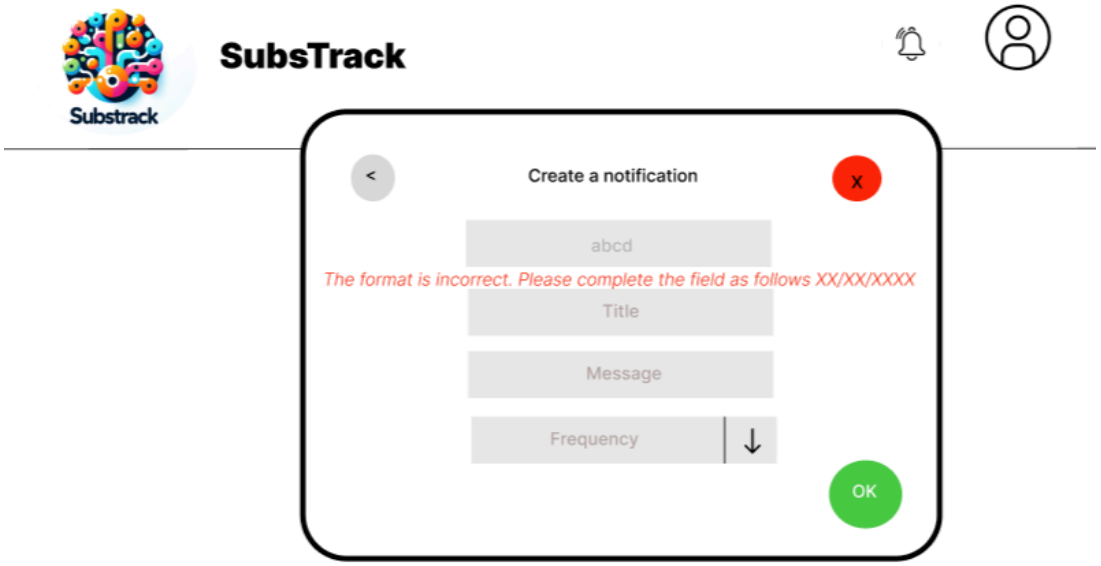


9.2.1.4 Automatic Notifications

- The system identifies a trigger for sending a notification.
- The notification is automatically generated and sent to the relevant users.

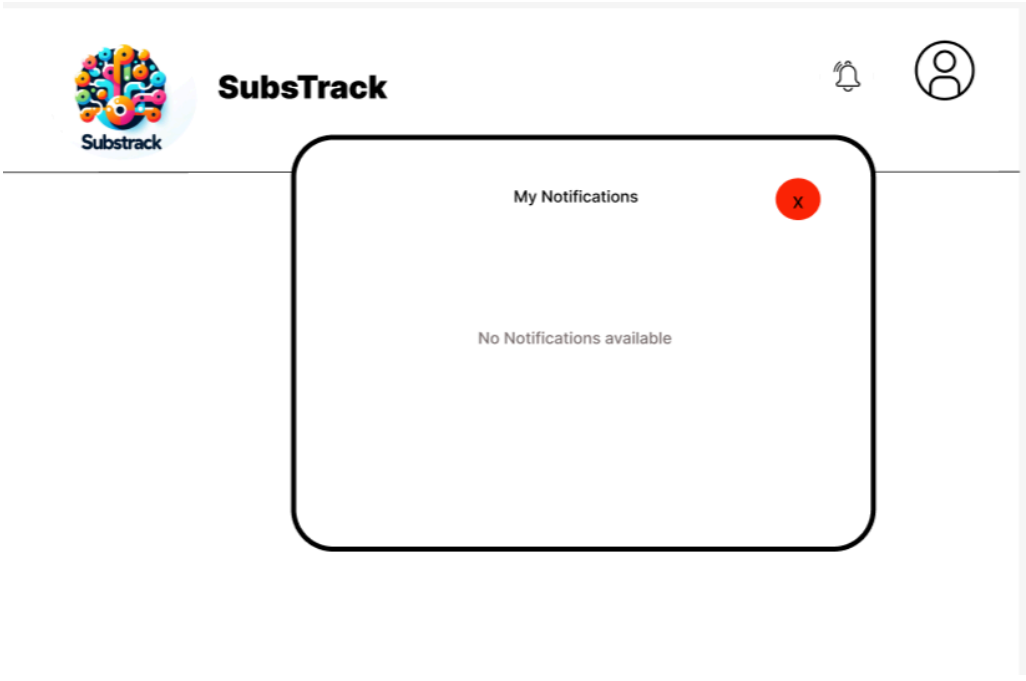
9.2.2 Alternative Flows

- Creation Error: If incorrect details are entered, the system shows an error. The user can correct or cancel the process.



- No Notifications Available:

  If no notifications are available, the system notifies the user.



9.3 Special Requirements

- Integration with the system calendar for scheduling notifications.
- Customizable options for recurrent notifications and automatic notification triggers.

9.4 Pre-Conditions

- Users must be authenticated to access notification functionalities.

9.5 Post-Conditions

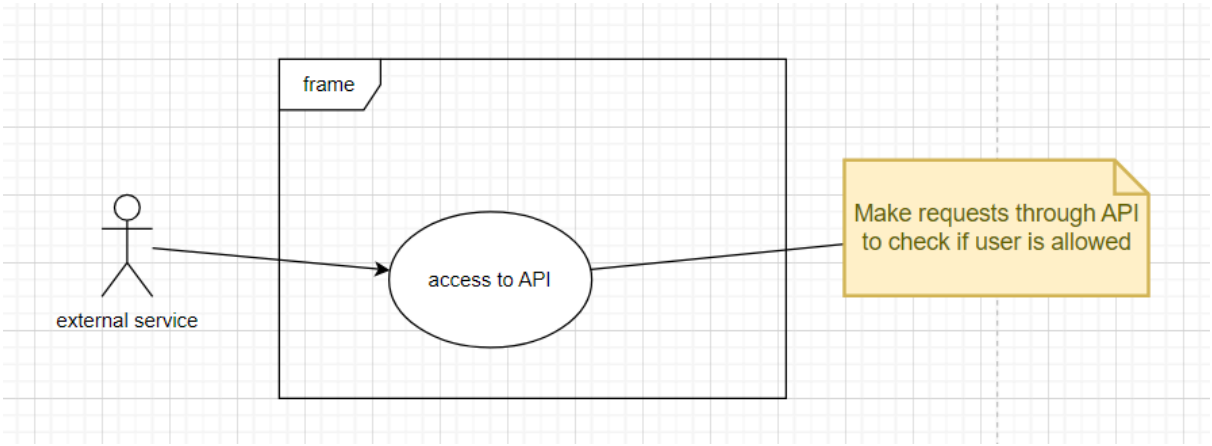
- Successful execution results in the creation, viewing, deletion, or automatic sending of notifications. If unsuccessful, the system remains unchanged.

9.6 Extension Points

- Integration with other system modules for event-specific or condition-triggered notifications, such as meeting reminders or system alerts.

10. External service connexion

10.0 Diagram



10.1 Brief Description

Having the ability to use the subscription for external purposes, like for a subscription that gives a keycard to access a room.

10.2 Flow of Events

10.2.1 Basic Flow

1. External HTTP request
2. Search through users in the software to check if the subscription gives access to the requested activity/access.
3. Send response

10.2.2 Alternative Flows

None

10.3 Special Requirements

The software needs a Server-side that have an API.

10.4 Pre-Conditions

The request need to contains the userId and accessId.

10.5 Post-Conditions

Returns :

- 200 if success and allow access
- 401 if the userId is not allowed

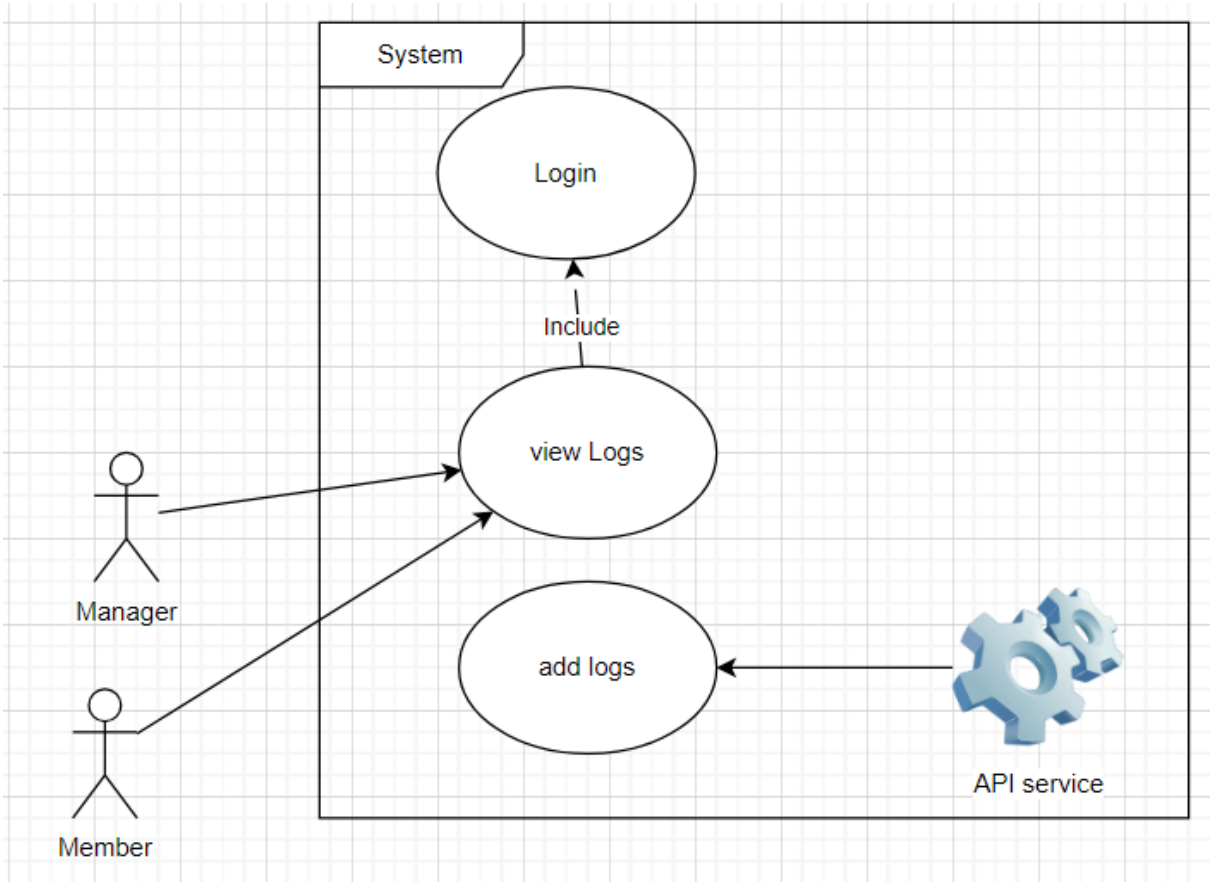
- 404 if the userId and/or the accessId is not found

10.6 Extension Points

The API could take other requests in the future, to enable reusage of the database access/activites so it can be used in other software.

11. Log history

11.0 Diagram



11.1 Brief Description

The manager and members can see the logs, no one can change or remove it for safety features. The API service can add logs depending on entry/exits.

11.2 Flow of Events

11.2.1 Basic Flow

1. Services are adding logs when their is and entry or exit
2. Members/Manager login
3. See logs :
    a. Manager can see members logs
    b. Members can see their logs

11.2.2 Alternative Flows

None

11.3 Special Requirements

None

11.4 Pre-Conditions

The member or Manager needs to be logged in.

11.5 Post-Conditions

None

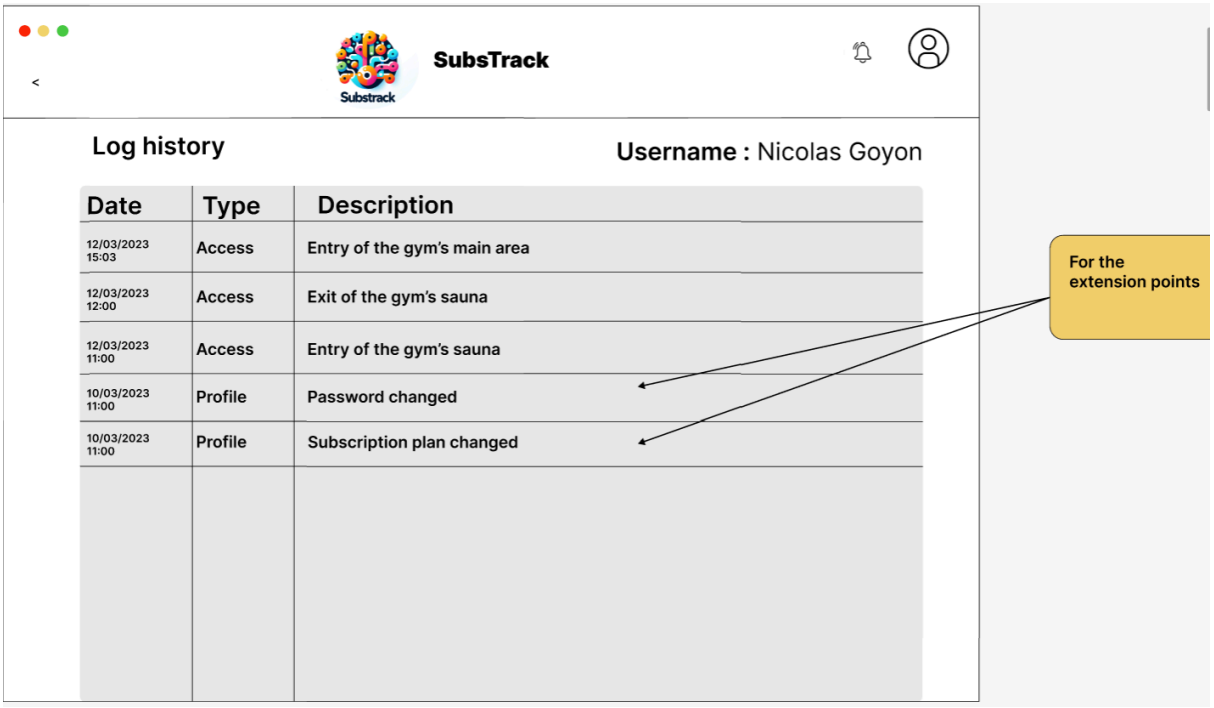
11.6 Extension Points

Logs can be extended to :

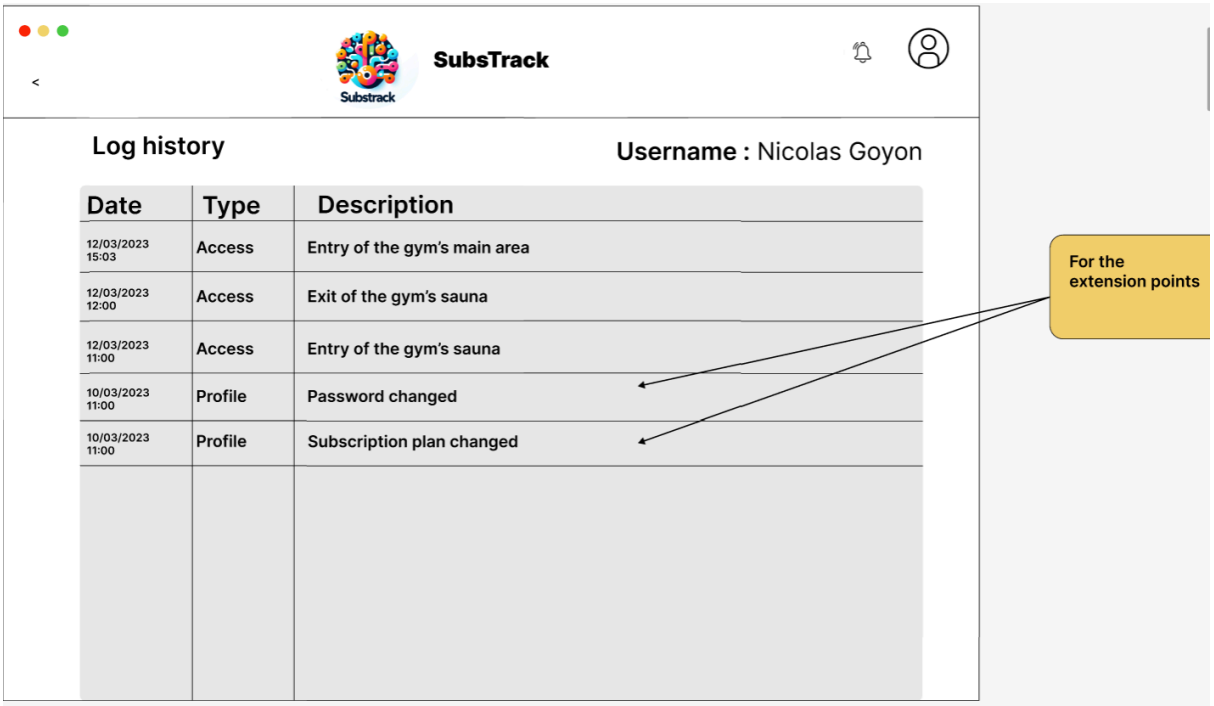
- Changes in every member profile editing
- Same for managers
- Changes in subscription plan
- Changes in access/activities

11.7 Mock-up

11.7.1 Mock-up when a manager see someone's logs.

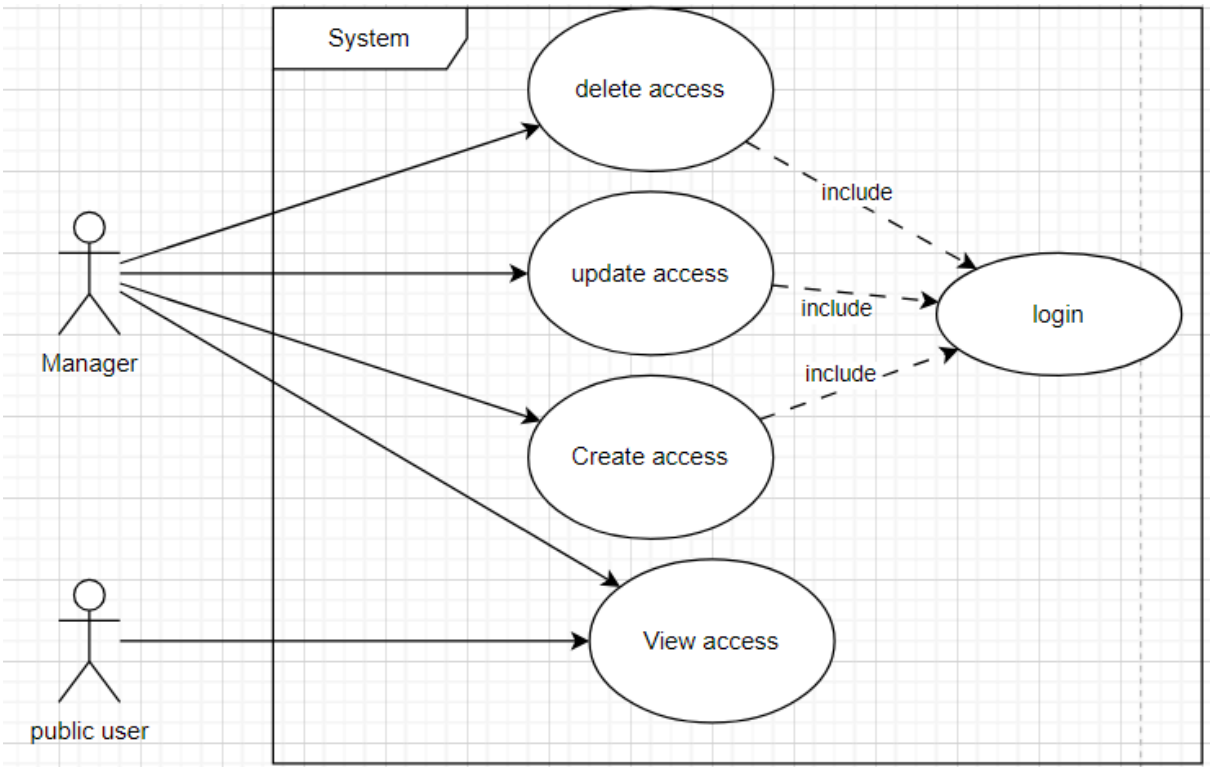


11.7.2 Mock-up when someone sees his logs



12. Access/Activities management

12.0 Diagram



12.1 Brief Description

Managers can do all aspects of a CRUD of the management of access/activities. These will also link to the update of subscriptions, these will cover multiple access/activites.

12.2 Flow of Events

12.2.1 Basic Flow

12.2.1.1 Create an access

- The system prompts for essential access details such as :
    - Name of the access
    - Description
- Once the manager submits the information, the access is added to the system.

12.2.1.2 View a access

- Either public user or manager can view access :

- the public user’s view is to be aware of what access/activities are available.

12.2.1.3 Manage access

- The manager can update any access.
- The manager can delete any access.

12.2.2 Alternative Flows

None

12.3 Special Requirements

None

12.4 Pre-Conditions

For the Update, Create, Delete operations, the manager needs to be logged.

12.5 Post-Conditions

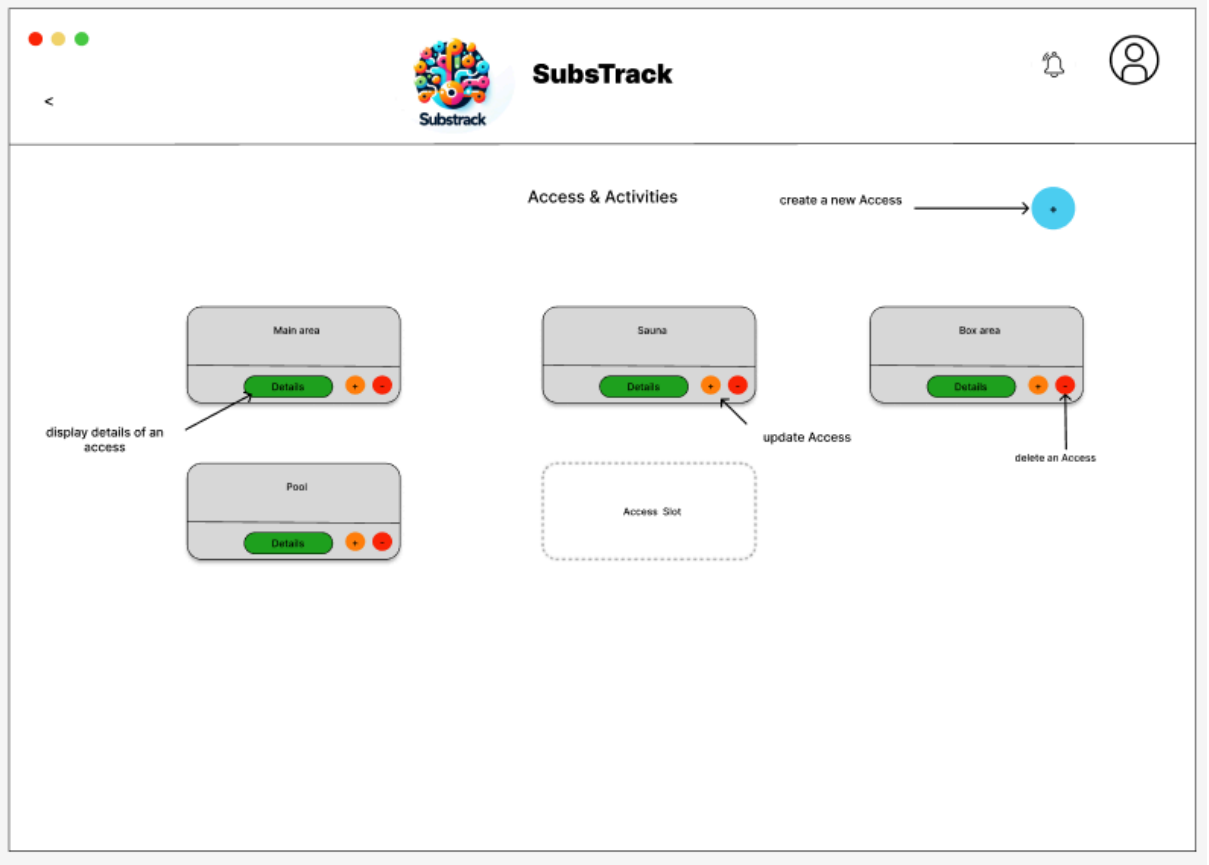
None

12.6 Extension Points

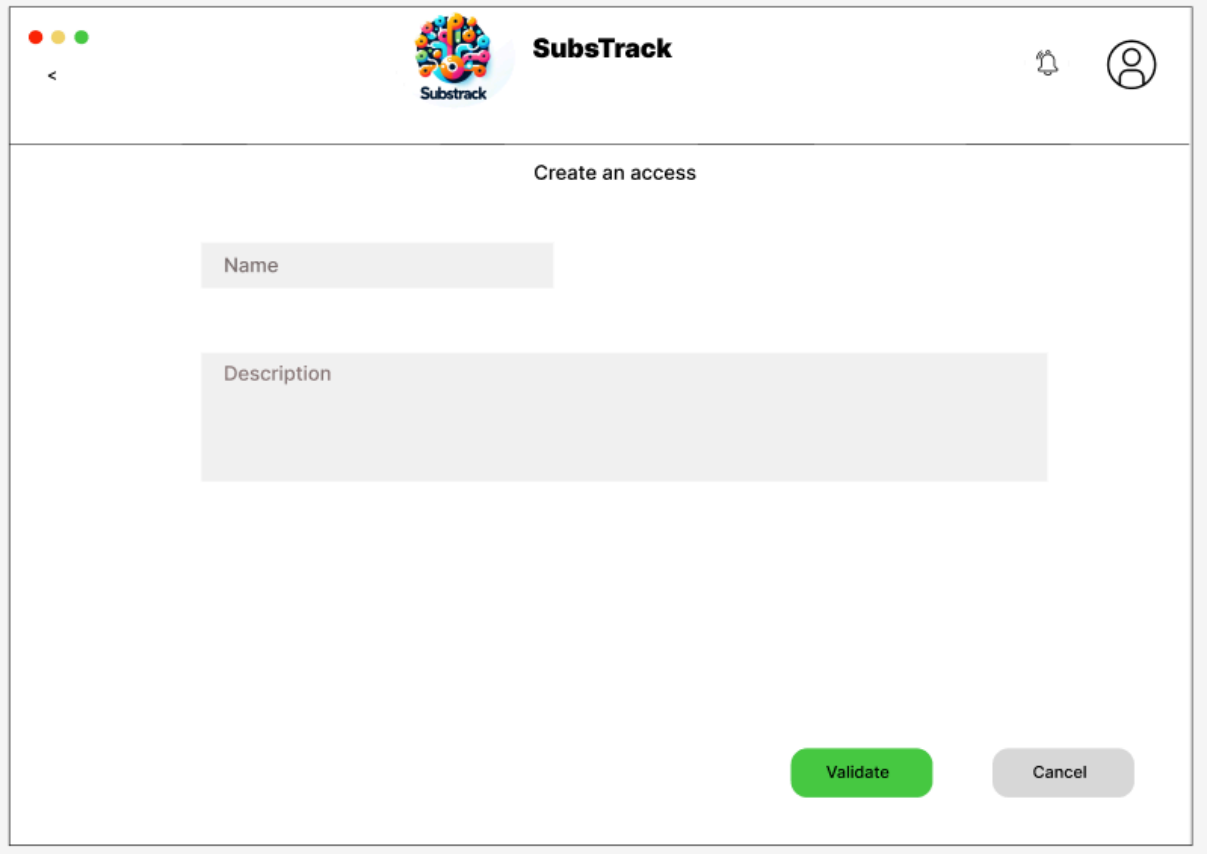
None.

12.7 Mock-up

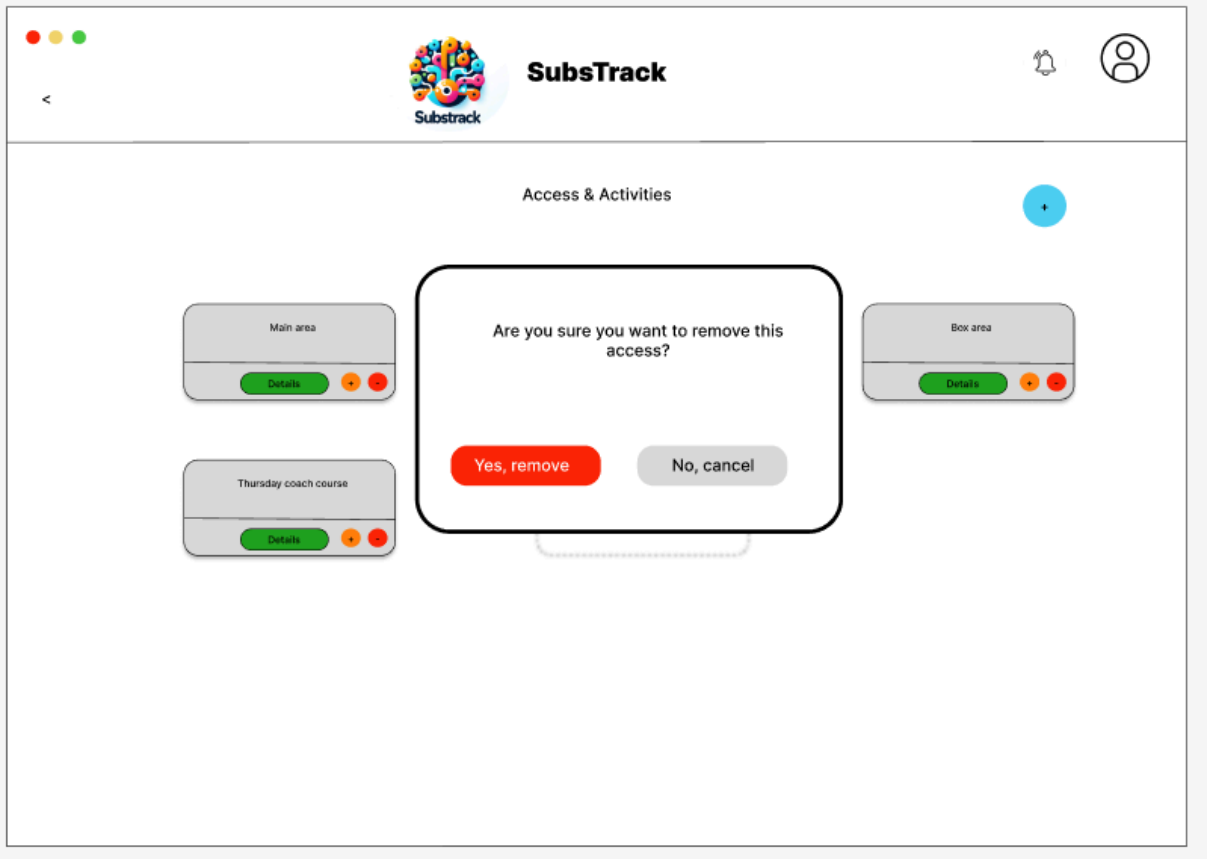
12.7.1 Manager see all Access



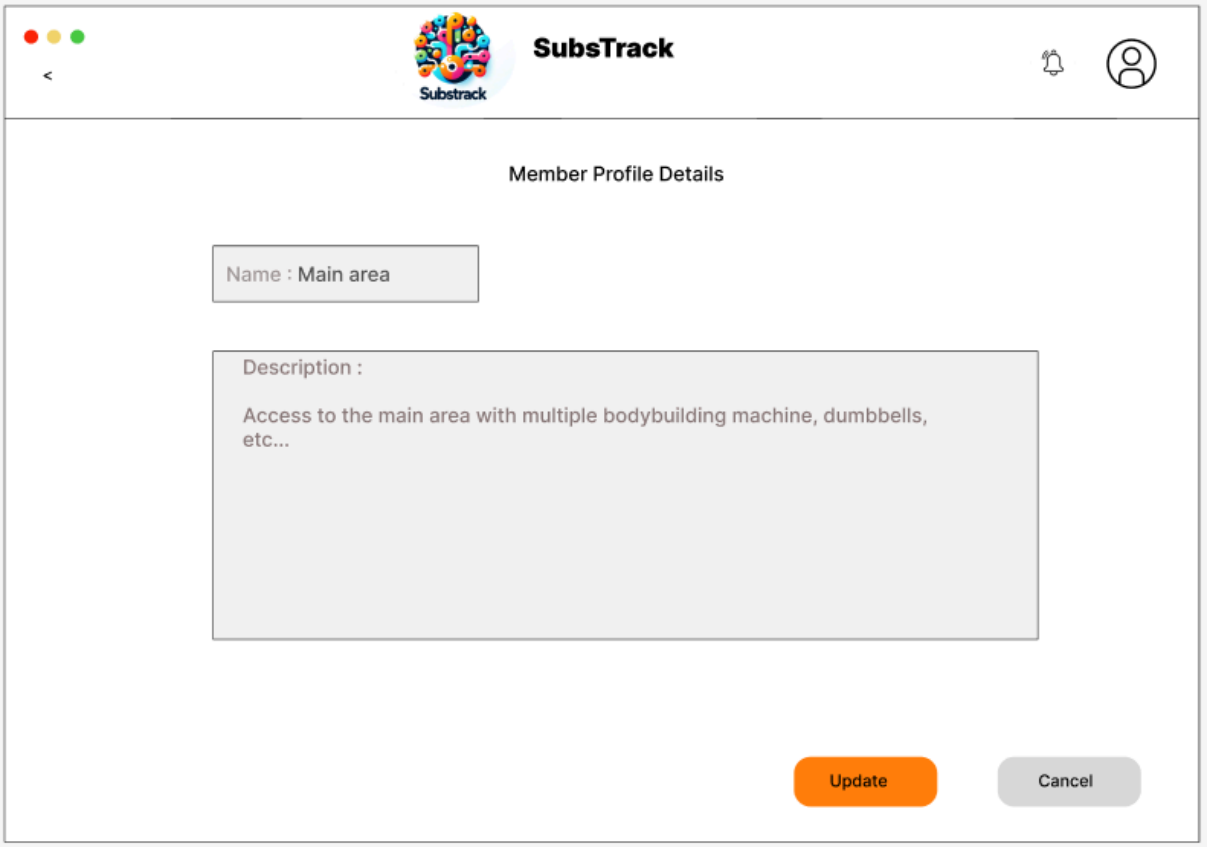
12.7.2 Creating a new Access



12.7.3 Removing an access



12.7.4 Updating an access



Progress of the project

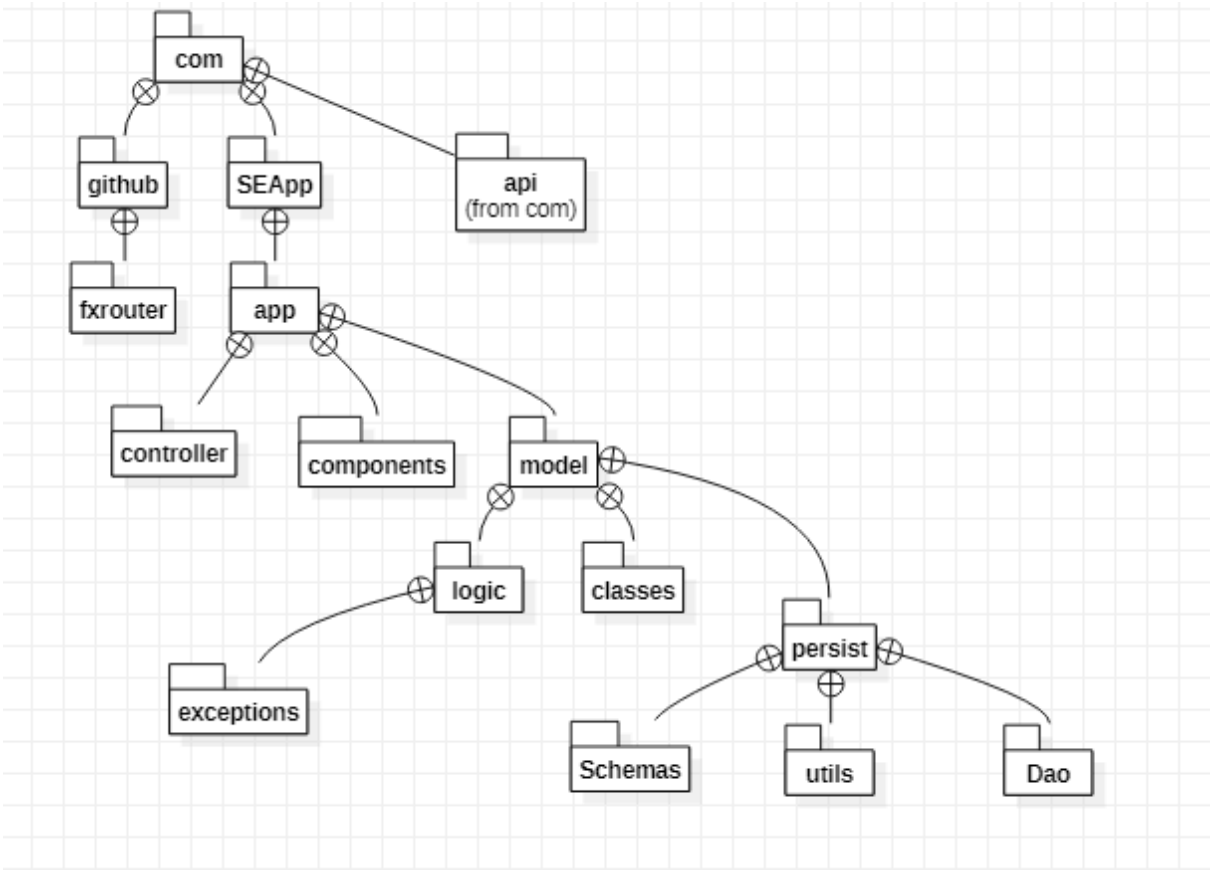
Use case work

Here is what have been done for each use case by each :

1. Login
    a. **Design** : Everyone
    b. **Code** : Nicolas
    c. **Tests** : *Not done (no unit tests)*
    d. **Database** : Assia
2. Member management
    a. **Design** :
        i. **Register** : Juan
        ii. **Member management** : Assia
    b. **Code** : Nicolas
    c. **Test** : Nicolas
3. Member subscription management
    a. **Design** : Nicolas
    b. **Code** : Nicolas
    c. **Test** : Nicolas
4. Plan management (managerial subscription)
    a. **Design** : Nicolas
    b. **Code** : Nicolas
    c. **Test** : Nicolas
5. Payment
    a. **Design** : Assia
    b. **Code** : Nicolas
    c. **Test** : Nicolas
6. Payment type management
    a. **Design** : Assia
    b. **Code** : Nicolas
    c. **Test** : Nicolas
7. Manager Management
    a. Design : Nicolas
    b. Code : Nicolas
    c. Test : Nicolas
8. Article Management : *Not done*
9. Notification : *Not done*
10. External services
    a. Design : Nicolas
    b. Code : Nicolas
    c. Test : Nicolas
11. Accesses Management
    a. Design : Nicolas

b. **Code :** Nicolas
c. **Test :** Nicolas

Global Architecture



The overall structure can be described as follows: within the "com" package, each distinct app or cloned dependency is housed. Due to modifications made to the FXRouter dependency, it was necessary to clone it and incorporate it into the source. The primary app is located in the "SEApp" package, while the API is situated in the "api" package. The API serves as an HTTP server for external services and must exist within the same project to share the same .env file and access the identical database.

Within the "SEApp," the "app" package contains all the source code, encompassing "controllers" for FXML controllers, "components" for the Java code of reusable components, and "model" for business logic. In the "model," one can find "logic" for Facades and Exceptions classes, "classes" for business classes, and "persist" for everything related to the database.

Further, in the "persist" package, there is a "Schema" for various table schemas (including table names and column names), "Dao's" for all data access objects and associated abstractions, and directly within "persist," the "AbstractDaoFactory" with the "PostgresDaoFactory."

Side use case project improvements

Nicolas implemented several code enhancements, including:

- Improving the base of the reusable component "Header," which was subsequently refined by Assia.
- Developing "ListDisplay" and "GridDisplay" reusable components to simplify the display of collections of objects, incorporating buttons for interaction with the main controller.
- Employing and fixing FXRouter to align with the current project requirements.
- Establishing a Docker-compose directory for a local Docker database during the development phase.
- Creating the "PostGres" class, which encompasses basic CRUD methods with generic "UpdateOperands" and "WhereOperands" to standardize database calls. This class ensures proper use of JDBC objects and guarantees their closure. All database access occurs through this class to enforce correct JDBC usage and implement queries that prevent SQL injections.
- Implementing a "Password Encryption" class to encrypt and verify passwords in a centralized area of the software, preventing code duplication and ensuring consistent password encryption.
- Utilizing DotEnv and a .env file to avoid hard-coded values in the code and consolidate configuration values in a single file.
- Introducing the "UserFacade" abstract class to encapsulate "AdminFacade," "ManagerFacade," and "MemberFacade." This approach centralizes login management in a single controller using abstraction, eliminating redundant or spaghetti code.
- Utilizing various class objects such as "Member," "Plan," etc., within controllers to ensure consistency in data passed through each method. For example, the "Member" object is passed through methods, and considerations about whether the password is encrypted are only required during member creation. The "Member" class itself handles the encryption, eliminating the need to encrypt the password again when read from the database.
- Leveraging GitHub CI to attempt to ensure that tests are completed when pushing changes to the main branch. Initially, the CI setup faced challenges, but eventually, a functional CI system became attainable.
- Introduce a "bigTransaction" context that allows for the option of rolling back or committing transactions. This context is primarily designed for testing purposes, enabling interactions with the database as intended without modifying the data. It ensures data consistency even in the presence of errors, preventing alterations to the database in case of failures.

Assia made significant enhancements to the CSS and diligently reviewed all FXML files to ensure they met a minimum quality standard.

Details of technologies used

FXRouter

FXRouter is simple, here are the main information needed to work with FXRouter :

- FXRouter is a singleton that handle the changes between controllers
- Bind() : the stage to the Singleton
- When() : a route is called, load a fxml
- GoTo() : a specified route

General project progress

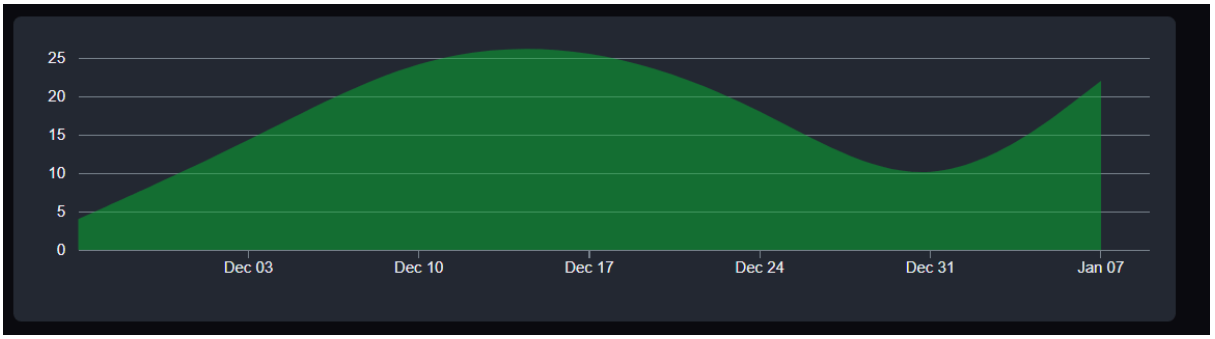
Initially, from the project's inception through the use case analysis phase, Assia exhibited high motivation and significantly contributed to the project's advancement. The entire group was actively engaged during the login phase. Following the completion of the login phase, Nicolas consolidated the work and extensively refined a substantial portion of the login use case to ensure a cohesive and polished beginning. In the subsequent weeks, there was a decline in activity, primarily due to exams. During the holidays, Nicolas completed the design work initiated earlier, developed and tested some components. Towards the end of the holidays, Assia resumed the work, displaying considerable effort and receiving support from Nicolas to finalize certain sections.

Additionally, at the outset, the database was hosted remotely on Azure Cloud Database. This decision was prompted by time constraints, necessitating a swift creation of a database without delving into considerations such as Docker. However, a few weeks later, we introduced a Docker container with MSSQL, maintaining compatibility with Azure's SQL syntax. Yet, due to performance issues and challenges in managing the Docker container, Nicolas subsequently transitioned the database to Postgres.

Github contribution

Disclaimer: GitHub contributions may not accurately reflect our actual work.

In terms of general contributions measured by commits:



For individual contributions within each commit (including lines of code added/removed) :

It's worth noting that the significant fluctuations in the number of additions and deletions are often attributed to the StarUML file. Occasionally, it is saved as a single line, while at other times, it may be saved as multiple thousand lines of code.

It's important to highlight that Assia's contributions may not fully represent her work due to her tendency to make larger commits, as opposed to Nicolas, who tends to commit more frequently with each stable step within an iteration. This difference in commit style can influence the granularity and visibility of individual contributions.

Challenges and Lessons Learned

We encountered communication challenges, leading Nicolas to establish a GitHub project with a Kanban board and a roadmap to monitor progress and organize the agenda. While Assia effectively utilized and maintained the GitHub project, David and Juan struggled to keep their progress updated and failed to communicate their issues or questions within the group. They reached out to Assia through direct messages but did not share their concerns with the entire group for collective assistance.

From this experience, we've learned valuable lessons for future projects. Emphasizing the importance of maintaining the GitHub project up to date is crucial, and communication should be conducted transparently within the entire group. Additionally, project or phase managers may consider proactively seeking feedback by asking questions more frequently to foster a collaborative and communicative environment.