# MindMerge — Deliverable e 1

Gabriele Benetti, Gioele Bernardini, Luca Fossa Crescini, Luca Sartore

June 23, 2024

## Contents

**1 Domain, Objectives** **4**

**2 SWAT Analysis** **5**
2.1 Quick recap . . . 5

**3 Functional Requirements** **6**
3.1 Authentication and account management . . . 6
3.2 Organization management and fee Requirements . . . 6
3.3 Task structure . . . 6
3.4 User interface . . . 6
3.5 Permission system . . . 6
3.6 Report system . . . 7
3.7 Notification system . . . 7

**4 Non-functional Requirements** **7**
4.1 Performance . . . 7
4.2 Reliability . . . 7
4.3 Usability . . . 7
4.4 Scalability . . . 8
4.5 Security . . . 8

**5 Use Case Diagram** **9**
5.1 Actors structure . . . 9
5.2 Authentication structure . . . 10
5.3 Owner structure . . . 11
5.4 Assignee and Manager structure . . . 12

**6 Context Diagram** **13**
6.1 Diagram . . . 13
6.2 Subordinate systems . . . 13
6.3 Actors . . . 14

**7 Components Diagram** **14**
7.1 Data Base Manager . . . 15
7.1.1 Diagram . . . 15
7.1.2 Descriptions . . . 16
7.2 Notification Manager . . . 16
7.2.1 Diagram . . . 16
7.2.2 Descriptions . . . 16
7.3 Account Manager . . . 17
7.3.1 Diagram . . . 17
7.3.2 Descriptions . . . 17
7.4 LLM Prompter . . . 18
7.4.1 Diagram . . . 18
7.4.2 Descriptions . . . 18
7.5 Task Tree Navigator . . . 18
7.5.1 Diagram . . . 18
7.5.2 Descriptions . . . 18

7.6 Report Manager . . . 19
7.6.1 Diagram . . . 19
7.6.2 Descriptions . . . 19
7.7 Task Manager . . . 21
7.7.1 Diagram . . . 21
7.7.2 Descriptions . . . 21
7.8 Organization Manager . . . 22
7.8.1 Diagram . . . 22
7.8.2 Descriptions . . . 22
7.9 Front End . . . 23
7.9.1 Diagram . . . 23
7.9.2 Descriptions . . . 24

**8 Class Diagram** **24**
8.1 Data Transfer Objects . . . 25
8.1.1 Diagram . . . 25
8.1.2 Description . . . 26
8.2 Data Base Manager . . . 27
8.2.1 Diagram . . . 27
8.2.2 Description . . . 28
8.3 Services Base Module . . . 28
8.3.1 Diagram . . . 28
8.3.2 Description . . . 28
8.4 Notification System . . . 29
8.4.1 Diagram . . . 29
8.4.2 Description . . . 29
8.5 Task Manager . . . 30
8.5.1 Diagram . . . 30
8.5.2 Description . . . 30
8.6 User Manager . . . 31
8.6.1 Diagram . . . 31
8.6.2 Description . . . 31
8.7 Organization Manager . . . 32
8.7.1 Diagram . . . 32
8.7.2 Description . . . 32
8.8 Report Manager . . . 32
8.8.1 Diagram . . . 32
8.8.2 Description . . . 33

## Abstract

In today's rapidly evolving business landscape, the importance of advanced technological tools becomes increasingly evident. The current market is characterized by swift and unpredictable changes, fueled by a myriad of factors such as technological advancement and the complex global geopolitical landscape. In this already intricate scenario, economic fluctuations and the growing expectations of consumers demand significant efforts from companies to remain competitive and meet the evolving needs of the market. All things considered, many uncertainties arise:

- *How* will the market change as these models become more efficient and effective?
- *What* must companies do to remain competitive in the evolving market?
- *What* opportunities exist today for companies to harness these technologies?

These were the uncertainties and presumptions of the present-day world.

What follows is our response.


## MindMerge

A web application focused on optimizing workflow. The app allows users to create work schedules and tasks to assist businesses in dividing workloads. The flagship feature is the system - integrated with modern language models - that provides *fully automated reports* on personnel productivity, based on notes and comments provided by employees themselves in an easy and flexible manner.

We believe that the best way to showcase our project is through the *experiences* of the end users it is aimed at. Various actors are involved in the software:

- Google Authenticator: the system dedicated to user access through Google credentials.
- OpenAI/Google server: the agent responsible for the operation of our flagship feature, namely the automatic creation of reports.
- MongoDB: the DBMS tasked with storing employee notes, tasks, subtasks, etc.

Of course, the end users cannot do without these agents. The end users can be divided into three separate subroles, interconnected through a **hierarchical** relationship regarding system read/write permissions, as we'll explore later on:

- *Employer*
- *Manager*
- *Executor*

Based on that, users might be looking for different goals on our platform:

- As an **employee**, I want improved communication and collaboration tools to enhance work efficiency.
- As a **team member**, I want tasks to be scheduled immediately based on objectives, with immediate acknowledgment from other teams, to reduce time spent in meetings.
- As a **team member**, I want a system to track tasks and deadlines, to ensure adherence to project timelines.
- As a **project manager**, I want a mechanism to manage internal project priorities effectively.
- As a **project manager**, I want the system to be scalable and adaptable to large projects with hundreds of collaborators.
- As a **project manager**, I want tools to help manage the workload of team members, to eliminate bottlenecks.
- As a **manager**, I want to receive detailed reports on the work completed, to enhance management effectiveness.
- As a **manager**, I want tools to evaluate employee efficiency effectively.

# 1 Domain, Objectives

Regarding the application domain, we have chosen **Business Productivity** for our project. Our decision was influenced by several factors. Certainly, the growing trend of public interest in AI has been a determining factor. The related market opportunities are promising, and B2B has historically always been promising. While our limited and relatively inexperienced team has certainly prevented us from implementing advanced features for our software, it has not impacted the choice of the system to develop.

1. **Enhanced Workflow Management**: Improve workflow management through detailed reports, easily obtainable through AI implemented within the software.
2. **Workload Optimization**: Optimize team members' workload by providing a real-time overview of current tasks and deadlines.
3. **Deadline Compliance Support**: Support adherence to deadlines through constant and real-time monitoring of tasks.
4. **Project Priority Management**: Effectively manage internal project priorities through a clear hierarchy of tasks to be completed.
5. **Communication Enhancement**: Improve communication and collaboration among employees by providing easy access to information for managers and colleagues.
6. **Scalability and Adaptability**: Ensure software scalability for projects of any size through a hierarchical structure of tasks and subtasks.
7. **Immediate Task Scheduling**: Schedule tasks immediately to reduce time spent in meetings, in an intuitive and practical manner.
8. **Employee Efficiency Evaluation**: Evaluate employee efficiency to optimize human resource utilization, eliminating the need for paper reports and individual interviews.

# 2 SWAT Analysis

## 2.1 Quick recap

The development of software must incorporate a comprehensive evaluation of the *economic environment* in which it will operate. Within this framework, the **SWOT** analysis emerges as a vital instrument for gaining a profound understanding of the surrounding landscape, enabling the formulation of a cohesive strategy to tackle challenges and leverage opportunities.

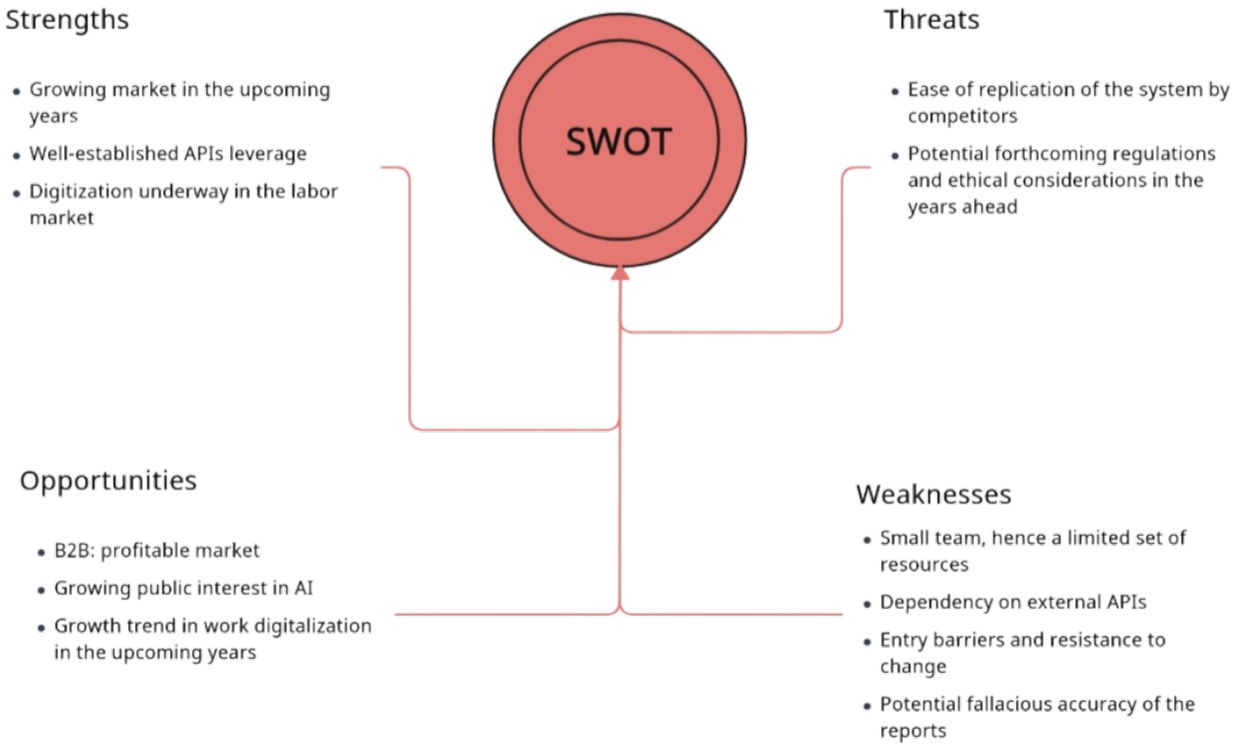



Figure 1: SWOT Analysis Diagram: visual representation of the Strengths, Weaknesses, Opportunities, and Threats associated with our software project

Here is a brief summary of the points outlined.

1. **Strenghts**:
   Predictions indicate that the LLM market is expected to grow by around **36%** by 2030[1]. Having reliable and established APIs is a big plus as they offer more flexibility and are easy to incorporate. The ongoing shift towards digitalization in the job market provides opportunities for innovation and improving how companies manage their human resources.

2. **Weaknesses**: Moving on to weaknesses, our project has a few. Firstly, our team is small both in terms of finances and people. Depending too heavily on external APIs, even if they're reliable, is risky because our system relies heavily on them. Marketing our product and breaking into the market, which is competitive and growing, could be challenging. *Resistance to change* from employees at client companies is another hurdle. Lastly, ensuring the reliability of our reports and avoiding mistakes is crucial; if reports are inaccurate, it affects the usefulness and saleability of our product.

3. **Opportuinities**: The business-to-business (*B2B*) market has always been profitable, mainly because buyers are often other businesses rather than individual consumers. It is estimated to grow by **19%** by 2031[2]. LLMs are becoming more recognized in the public eye, which can work in our favor. Plus, ongoing trends in digitalization and innovation in our sector offer good opportunities.

4. **Threats**: Our project faces some potential risks. Firstly, there is the chance that competitors could easily copy our idea in the near future. Additionally, **future regulations** on AI could pose an economic threat. If restrictions are

[1]source: grandviewresearch.com
[2]source: straitsresearch.com

placed on LLMs, a significant part of our system would be affected. There is also the concern that *ethical debates* around work might become a problem, especially if there is widespread distrust or bans on AI in Western countries.

# 3 Functional Requirements

## 3.1 Authentication and account management

- An user shall be able to perform a **login** operation
- An authenticated user shall be able to perform a **logout** operation
- An authenticated user shall be able to completely **delete** his *account*

## 3.2 Organization management and fee Requirements

- The owner shall be able to **pay a subscription** for enabling its *Organization*
- The system shall be able to restrict edit *permissions* if the subscription is not paid
- The system must allow users to **create** an *Organization*, and therefore becoming its *owner*
- The owner must be able to **add and remove** users to his *Organization*
- The system must allow Organizations to assign tasks for their *assignees*

## 3.3 Task structure

- Each task should be capable of having **multiple** subtasks in a *tree structure*
- Each task must have exactly **one** *manager*
- Each task must have **at least one** assignee
- Each task should have a **title**
- Each task should have a **description**
- Each task should have a **priority option**
- Each task should have a **deadline option**
- Each task should have a **state attribute**
- Each task should have a *manually managed* **notes section**
- Each task should have the possibility to be **deleted** by its *managers*

## 3.4 User interface

- Users shall be able to **see** all the tasks *assigned* to and/or *managed by them* in a *tree structure*
- Each user shall be able to **edit** all the tasks that are *assigned to and/or managed* by him

## 3.5 Permission system

- The system must implement a **permission mechanism**
- The permission mechanism shall be **customizable** by each task
- The permission system must implement **read/write** permissions
- The system must *enforce* task permissions **to each user**

### 3.6 Report system

- The application must provide a **manual report** option
- The manual reports *must* be done by users
- The application *must* provide an **automatic report** option
- The automatic reports *must* be *based* off of user notes and reports
- The automatic reports *must* be created by the system using *LLM technology*
- The system must allow managers to *request* reports **manually**
- The system must allow managers to *request reports* **automatically**
- The system must implement a mechanism that allow managers to *schedule* **manual or automatic** report requests
- When a manager *schedules* a report he shall be able to set a **time** and a **frequency**
- When a manager *schedules* a report he shall be able to **delete** it
- A manager shall be able to see a list *containing* all the reports of a specific task

### 3.7 Notification system

- The system *must* implement a **Notification dashboard**
- Each user can *mark* messages as **read** or even **delete** them
- The notification dashboard shall be **available** for both *assignees and managers*
- The notification dashboard shall show **report requests** issued by *managers*
- The notification dashboard shall show **task completion** pings to each appropriate user that have *manually enabled notifications* for a specific task
- The system shall allow its users the option for **enabling** notifications
- The system shall allow its users the option for **disabling** notifications

## 4 Non-functional Requirements

### 4.1 Performance

- The software should implement tools for **performance monitoring**
- The automatic report generation should take **at most** *3 seconds* for each level of the task tree
- Notifications should be sent **in less than** *5 minutes* from the happening of the triggering event

### 4.2 Reliability

- The system should have **minimum downtime** and at least 99% uptime
- The system should implement a way to **avoid collisions** if two or more users try to edit the same task simultaneously
- The automatic report generation should implement a system to **prevent chatbot hallucination**
- The system should have **redundant storage** to avoid data loss in case of an accident

### 4.3 Usability

- The software and the UI should be **easy** to understand and **friendly** to use
- The automatically generated report should be **as accurate as possible**

### 4.4 Scalability

- The system should be able to manage **at least** *100.000* users simultaneously
- The system should be able to manage **at least** *10.000.000* non-simultaneous users
- The system should be able to manage a task tree with **at least** *1000* tasks
- The system should be able to manage a task tree with an height of **at least** *10* levels

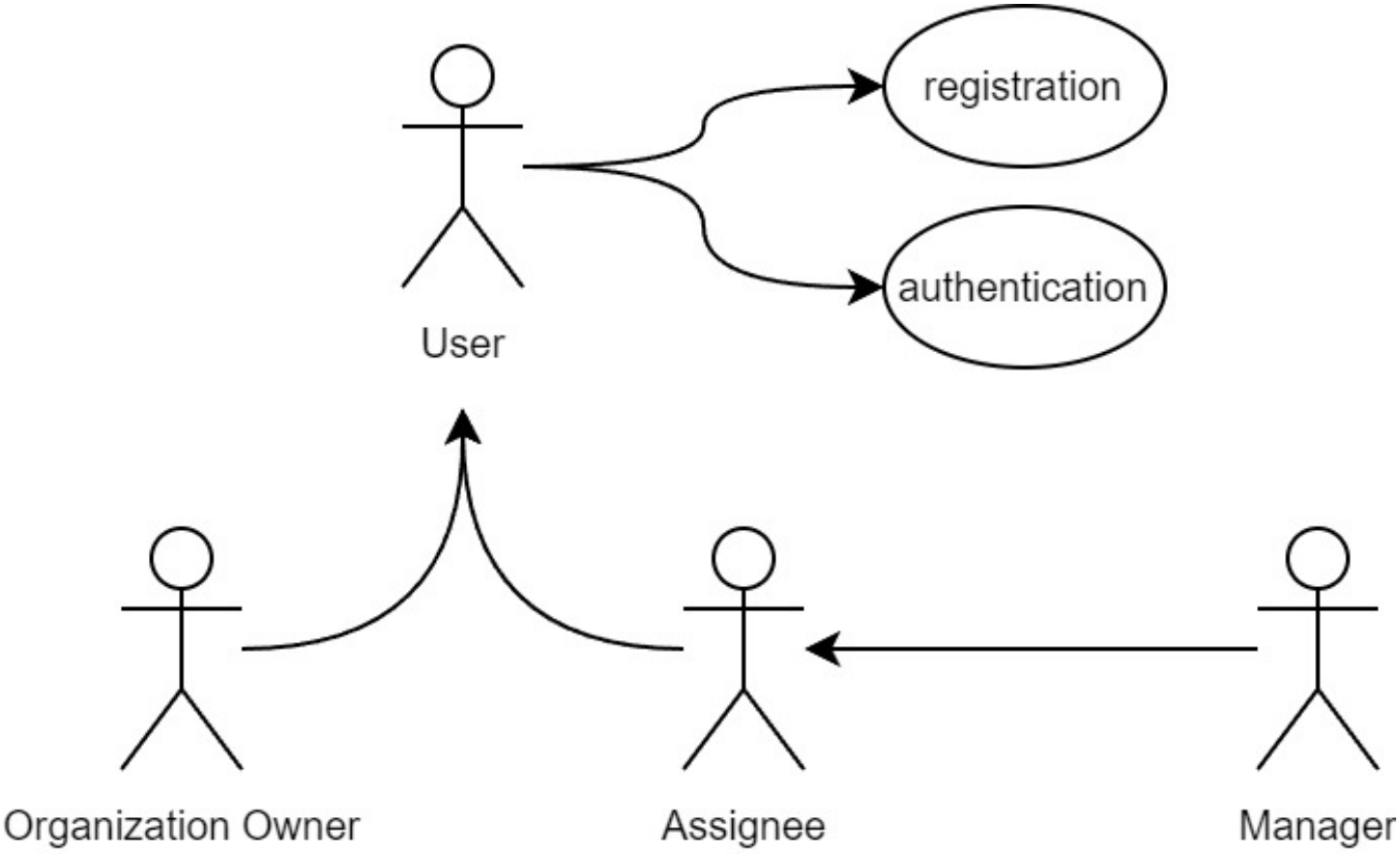
### 4.5 Security

- The system should allow users to see and edit **only** tasks they are entitled to use
- Authentication procedure should rely on **strong encryption** mechanisms
- The password recovery system should not compromise security
- A company's data should be encrypted in the database in order to **avoid data leaks** in case of security breaches

# 5 Use Case Diagram

Within this section, we discuss the *Use Case Diagram* of the system, outlining the diverse *capabilities* available to each actor when utilizing MindMerge.

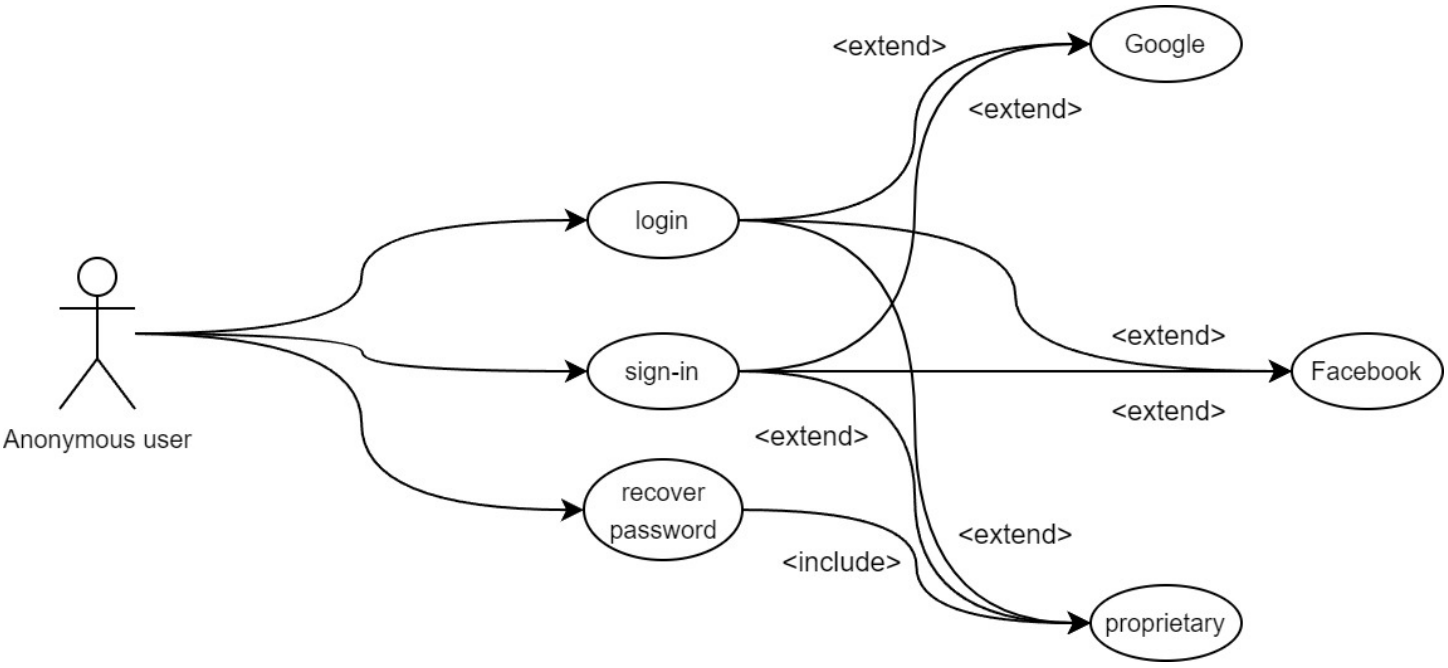
## 5.1 Actors structure




### Description

In our system, a critical part is the **registration** and the following **authentication**. This concept keeps valid for *every kind of user* within the platform. In the system, *3* type of user are made available. The most important one is the **Organization Owner**. It is responsible for the *deployment* of the entire organization and managing every user contained in it. More of that will be discussed later on. The other roles are the **Assignee** and the **Manager**. The assigne is considered to be the last in the logistic chain, responsible for completing tasks and managing its own notes for the *AI report system* to work. Finally, the Manager actor *inherits* everything from the assignee, but he is granted some major *capabilities*, such as managing tasks, creating subtasks and all things related to them.

## 5.2 Authentication structure



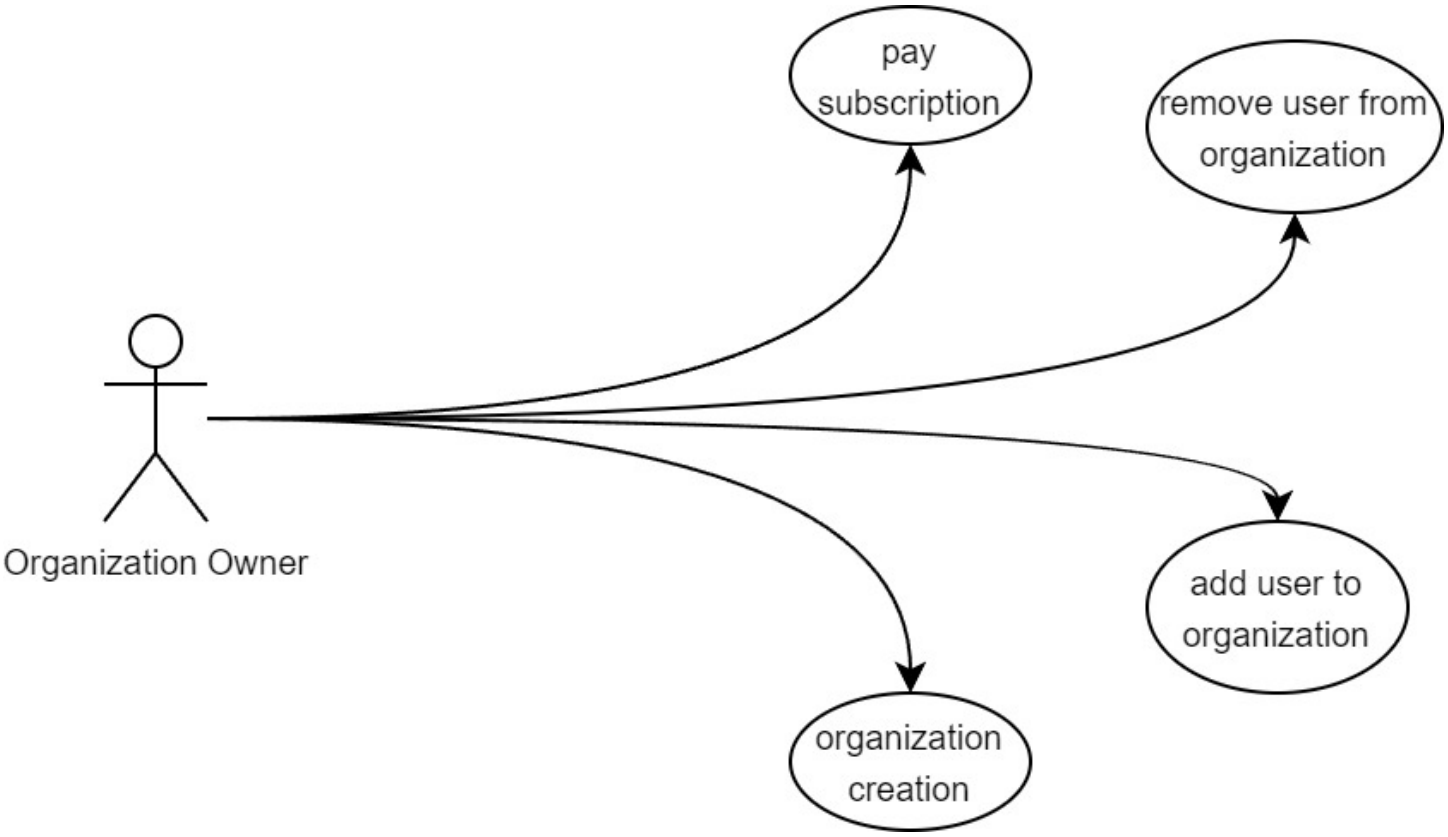

### Description

There are multiple methods for end-users to *access* the platform. Mainly, users can utilize **Google** and **Facebook** services, which allow them to conveniently access the platform using their already existing accounts. Additionally, a **proprietary system** for account creation and management will be provided.
Here is the authentication structure flow:

- An *anonymous* user initiates a login request.
- A subsequent sign-in request is made, utilizing the sign-in service mentioned above.
- Once authenticated, the user is granted access to the platform.

In the event that a user forgets their credentials, a *password recovery system* will be available. The system will send an email to the user containing a recovery link.

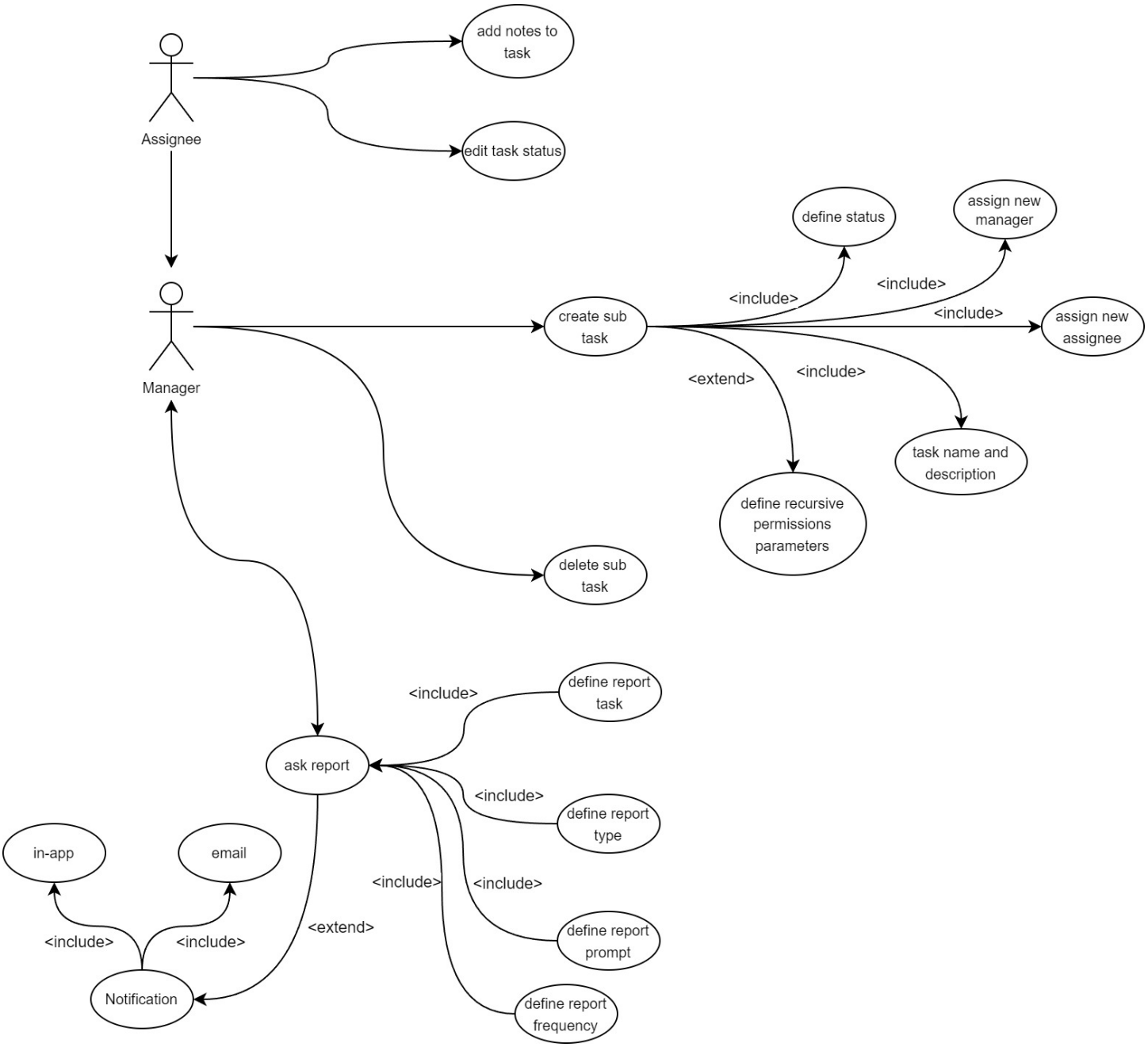
### 5.3 Owner structure




#### Description

The organization owner holds **full system capabilities**. It is up to him to *add* and *remove* users, as well as create the organization itself. They are also tasked with **paying** the subscription for the service, calculated based on the platform usage rate and managed through *external banking APIs*.
More of that will be discussed in later sections.

## 5.4 Assignee and Manager structure




## Description

As an **assignee**, my primary responsibility is to *add notes* and *edit task statuses* based on the daily workload. Additionally, a **manager** also has the permission to create *subtasks*.

This critical aspect involves several components:

- Definition of the **status** (modifiable during the entire lifecycle of the subtask),
- Nomination of the *manager*, as well as the *assignees*,
- Definition of *recursive* permission parameters (they define how far down on the tasktree permissions will be inherited),
- *Naming and description* of the task.

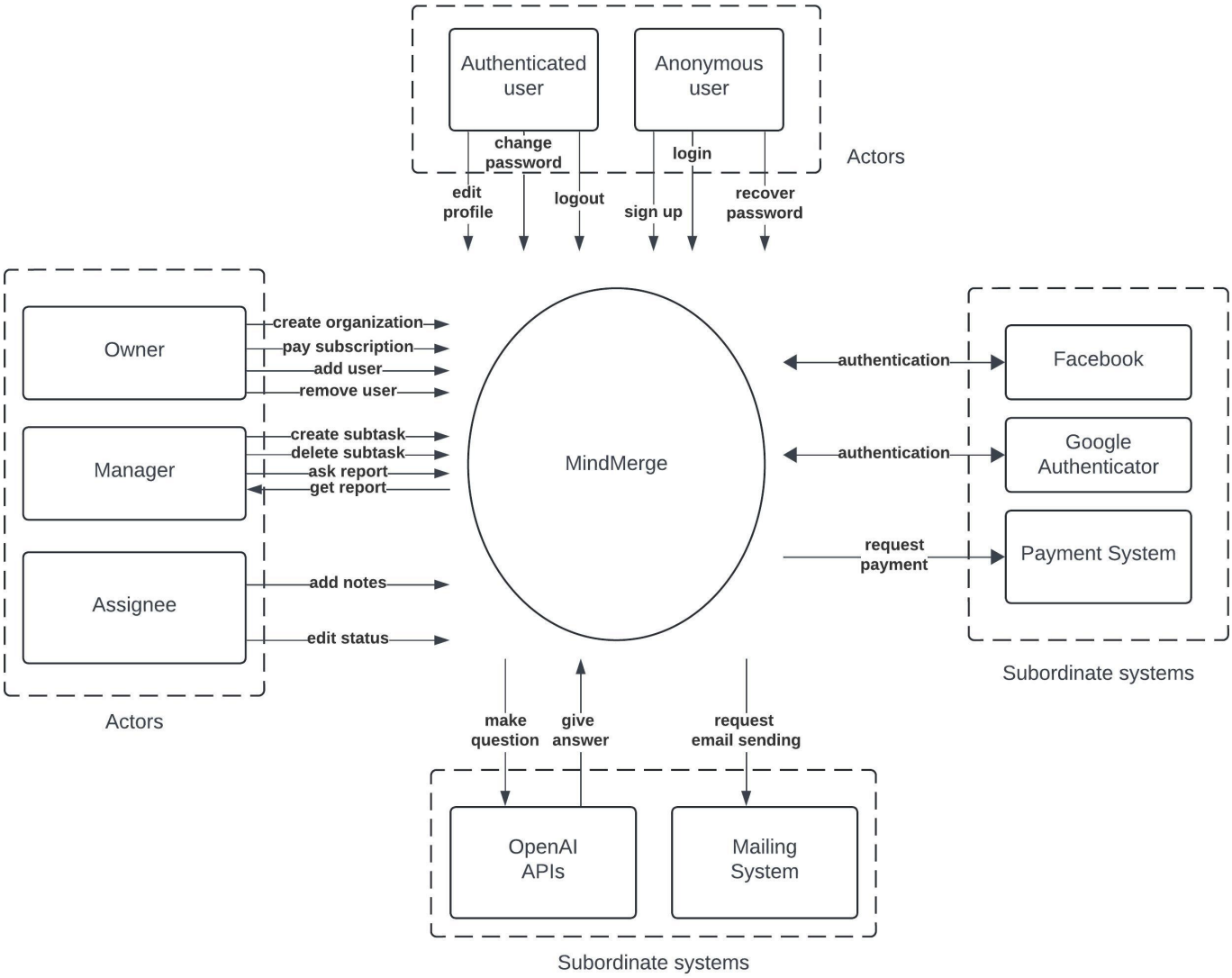
Managers also have the permission to **delete** subtasks, whether *completed or not*.

Furthermore, managers gain the capability to **request a report**. This request should include details such as the inquiring task, its *type* (AI-managed or not), its *prompt, and its frequency*. The *notification system* will inform the targeted users via email or in-app notification.

# 6 Context Diagram

This section illustrates the context diagram of the system and describes the relations between MindMerge and two different types of external entities.

## 6.1 Diagram




## 6.2 Subordinate systems

The external softwares that are used by MindMerge to provide services to the users are:

- **Google and Facebook Authenticator:** they are used to give to users the possibility to log in using their Google or Facebook account. Allowing *third party authentication*, the system doesn't force user to create a new MindMerge account.
- **Payment System:** It is used to allow Organization's owner to pay for the system's subscription.
- **OpenAI APIs:** They are used, on manager request, to generate an *automatic report* of job done by assignees. MindMerge requests a report to the APIs and is given back the result.
- **Mailing system**: It is used as part of the notification system to *notify* creation of new tasks or updates to the existing ones via mail.

### 6.3 Actors

Users that interact with the system and are able to use some of its functionalities.

- **Anonymous user:** A user that **has not** yet authenticated into the system. He is able to *access* login functionalities, also by using subordinate systems provided for that, to *sign up* in case he doesn't have an existing account or to *recover* password in case he isn't able to remember it.
- **Authenticated user:** A user that has completed authentication procedure and **is logged** into the system. He has the possibility to logout in every moment, becoming an anonymous user, but also to edit his profile or change his password.
- **Owner:** A user that is the **head** of a specific organization. He has the possibility to *create* an organization, *pay* the subscription using the external payment system and *add to and/or remove from* his organization. He is himself a manager and an assignee, so he also has all the possibilities related to these roles.
- **Manager:** A user that **manages** a specific task. He has the possibility to *create* subtasks and do some operations related to that (define task is status, assign new manager or new assignes to it, defining it's name, description and recursive permission parameters). He is also able to *delete* the subtasks he has created or to *ask and receive* reports, that can be generated automatically by using OpenAI APIs or manually by an assignee. He is an assignee himself, so he also has all the possibilities related to that role.
- **Assignee:** A user that is required to complete a specific task by a manager. He has the possibility to *edit* the task's status, to *notify* the manager that he has done something he was asked to do, or to *add* notes, in order to communicate something related to the work done.

## 7 Components Diagram

This section illustrate the component diagram of the system. To improve readability we split the components into several sub components. Each sub component has a color assigned to it, and the interfaces that are provided by that specific component are painted with the same color. This helps to quickly identify which component is responsible for a specific interface.

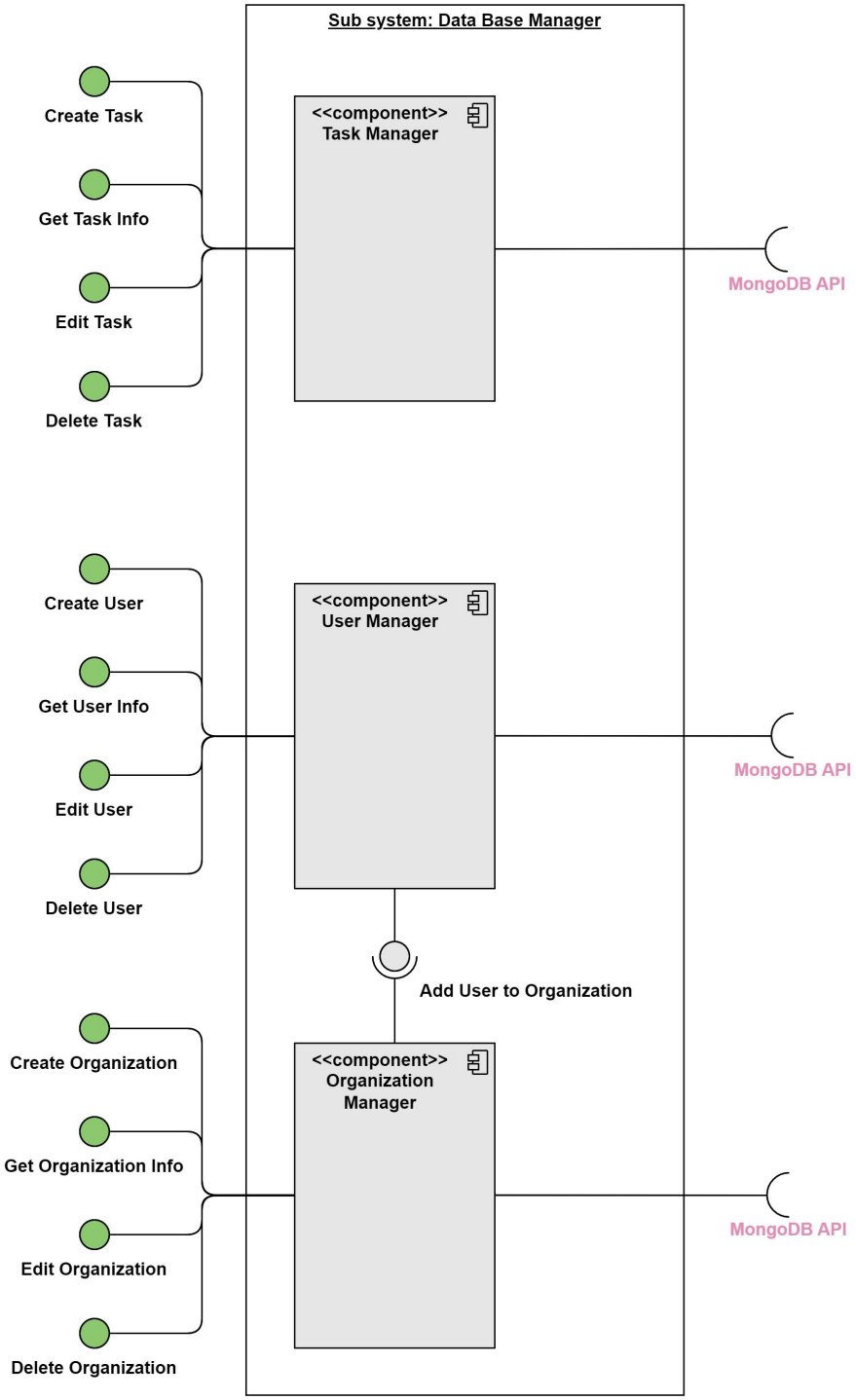
The list of sub components is the following:

- **Data Base Manager:** This component is responsible for managing the **data base**, it is an interface between the data base and the rest of the subsystems. The color assigned to this component is green.
- **Notification Manager:** This component is responsible for managing the **notifications**; This means sending notification, and serve the data to the front end to visualize the pending notification of a user. The color assigned to this component is yellow.
- **Account Manager:** This component manage **accounts**, in particular it handles sign up, log in, account changes and account deletion. The color assigned to this component is red.
- **LLM Prompter:** This component is a simple library that provide an **interface** to prompt different **LLMs**. It is objective is to make the implementation of the rest of the system agnostic from the specific LLM used. The color assigned to this component is aqua green.
- **Task Tree Navigator:** This component is a simple library with some algorithm to navigate a tree data structure. In our specific case the tree represent the tasks and subtasks of an organization. Since this algorithms are used many times throughout the system they have been put in a specific component to make them reusable. The color assigned to this component is light blue.
- **Report Manager:** This component manage the **reports**, in particular it is responsible for generating report *automatically* (using an LLM) or *manually* (reminding users with a notification that they have to deliver a report). The component also allow users to set report schedule, so that the reports generation can be triggered automatically with a customizable frequency. The color assigned to this component is orange.
- **Organization Manager:** This component allow an user to create an **organization**, or perform some action on his organizations, like adding/removing a user, or paying the subscription to use the software. The color assigned to this component is purple.
- **Task Manager:** This component allow users to interact with the **tasks**. In particular it is responsible to visualize, create, delete and edit tasks inside one organization. The color assigned to this component is blue.

- **Front End:** This component is responsible for the visualization of the data, and the interaction with the user. This component dose not have a specific color assigned to it, since it dose not provide any interface to other components.
- **External APIs:** All interfaces provided by external APIs in the system (like authentication, payment, etc.) are painted with the color pink.

## 7.1 Data Base Manager

### 7.1.1 Diagram

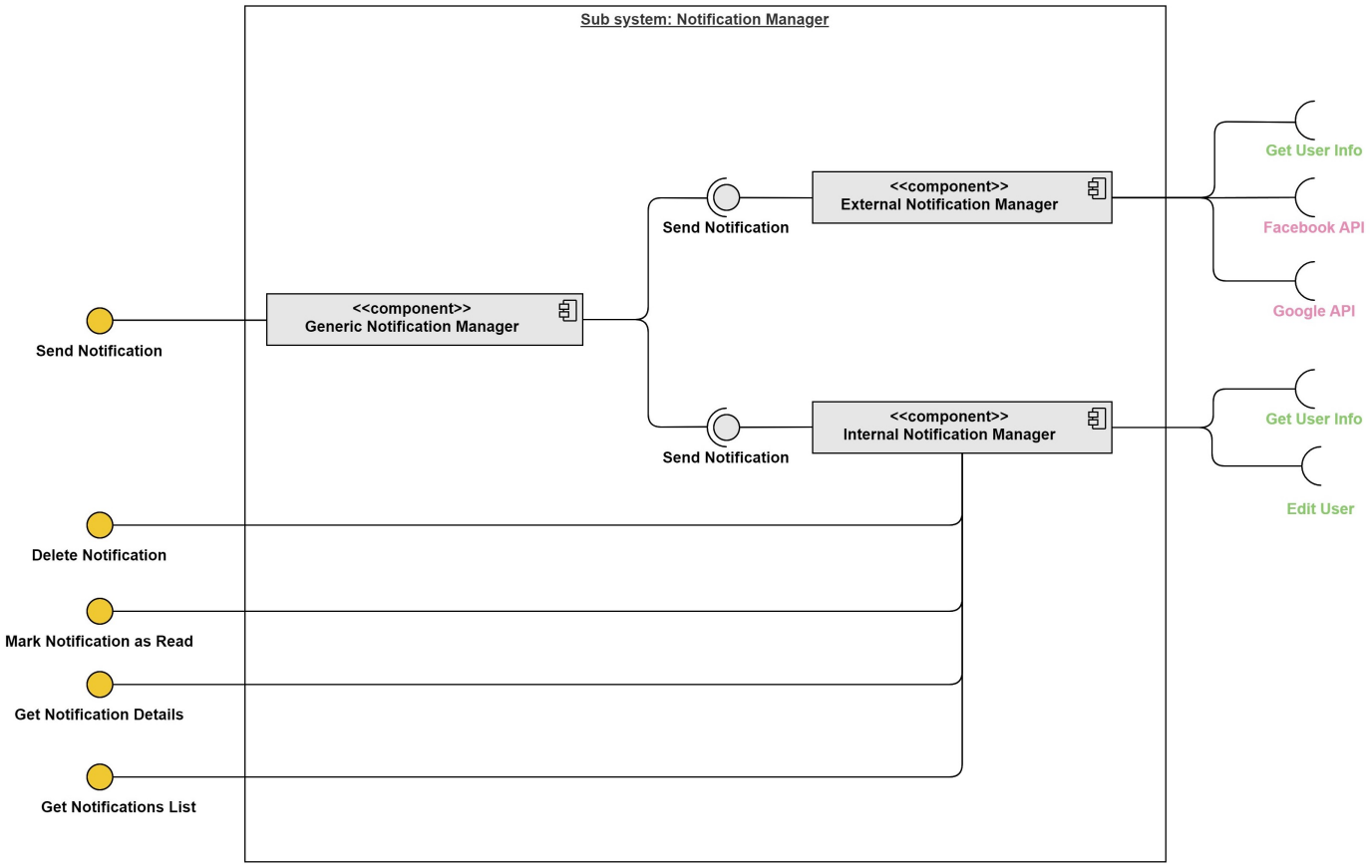


### 7.1.2 Descriptions

We have modeled the system with 3 particular objects (**tasks**, **users** and **organizations**). The database manager has a specific component to manage each one of them. The interfaces provided are divided for the kind of action that needs to be performed (**creation**, **deletion**, **modification** or **read operations**).

## 7.2 Notification Manager

### 7.2.1 Diagram




### 7.2.2 Descriptions

The notification manager system has 2 specific tasks:

- Allowing other components to **send** notification to users.

  This is done by the *"send notification"* interface

- Allow a user to **visualize** pending notification in the **front end**

  This id done by the other 4 interfaces

The system is composed of 3 different component

- **Generic Notification Manager:**

  This component is the one that provided the "Send Notification" interface, and it only act as a **switch**. The system allow for notification to be sent either via **mail**, or with an **in-app** notification. When a notification comes in to be send this component will decide where to send it.

- **External Notification Manager:**

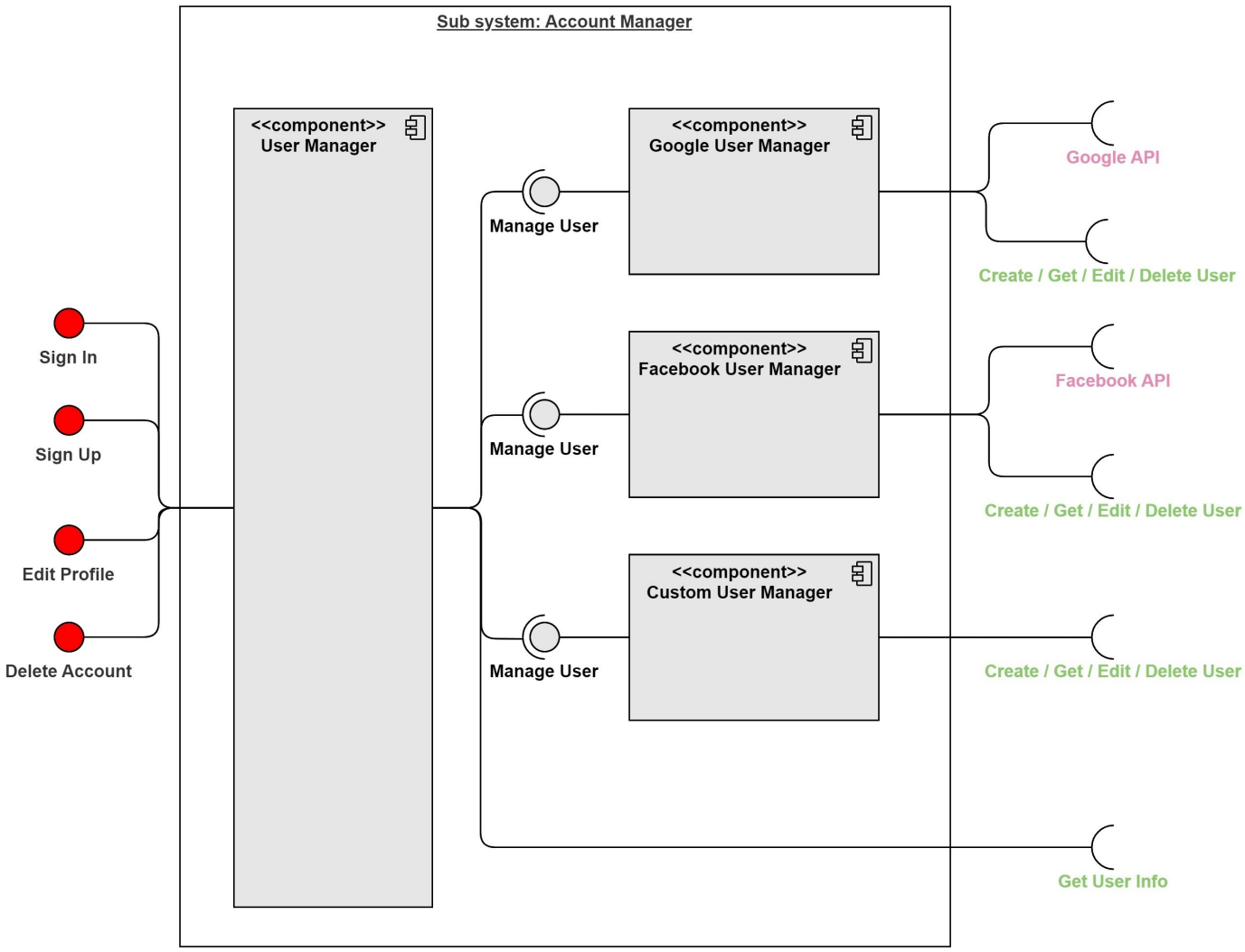
  This component will send a **mail** with a notification to a specific user. This component has various external APIs connected to it, as how the notification is sent will depend on how the user signed In.

- **Internal Notification Manager:**

  This component manage the **in-app** notifications. It can either be asked to send a notification (in this case it writes it in the database) or it can be asked to get a notification by the front end, so that it can be visualized (in this case it reads the necessary data from the database)

## 7.3 Account Manager

### 7.3.1 Diagram




### 7.3.2 Descriptions

The account manager system allow users to **register**, log in, **delete** account and **change account information** (like password, name, etc.).

The specific implementation depends on how the user signed in. For this reason the system has a **switch** component ("User Manager") that will decide which specific implementation to use for each specific user. Once this is decided the user can be handled by one of the 3 components:

- **Google User Manager:**

  This component will handle all the user that signed in with a **Google** account.

- **Facebook User Manager:**

  This component will handle all the user that signed in with a **Facebook** account.

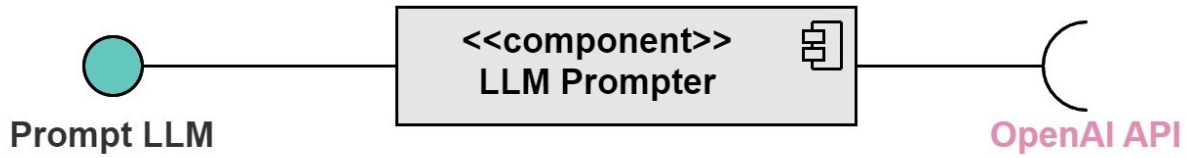
- **Custom User Manager:**

  This component will handle all the user that **register with an email and password**.

Note that we have bundled the database interfaces for managing users into one single interface to enhance readability.

## 7.4 LLM Prompter

### 7.4.1 Diagram

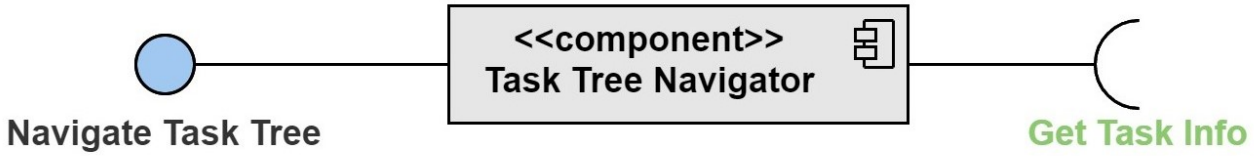



### 7.4.2 Descriptions

This is a simple component that **abstract** the detail about the specific **LLM** that is been used from the rest of the system. This is a good practice that can save some time in the future, if we decide to change the LLM that we are using. In this example we are using Open AI APIs.

## 7.5 Task Tree Navigator

### 7.5.1 Diagram

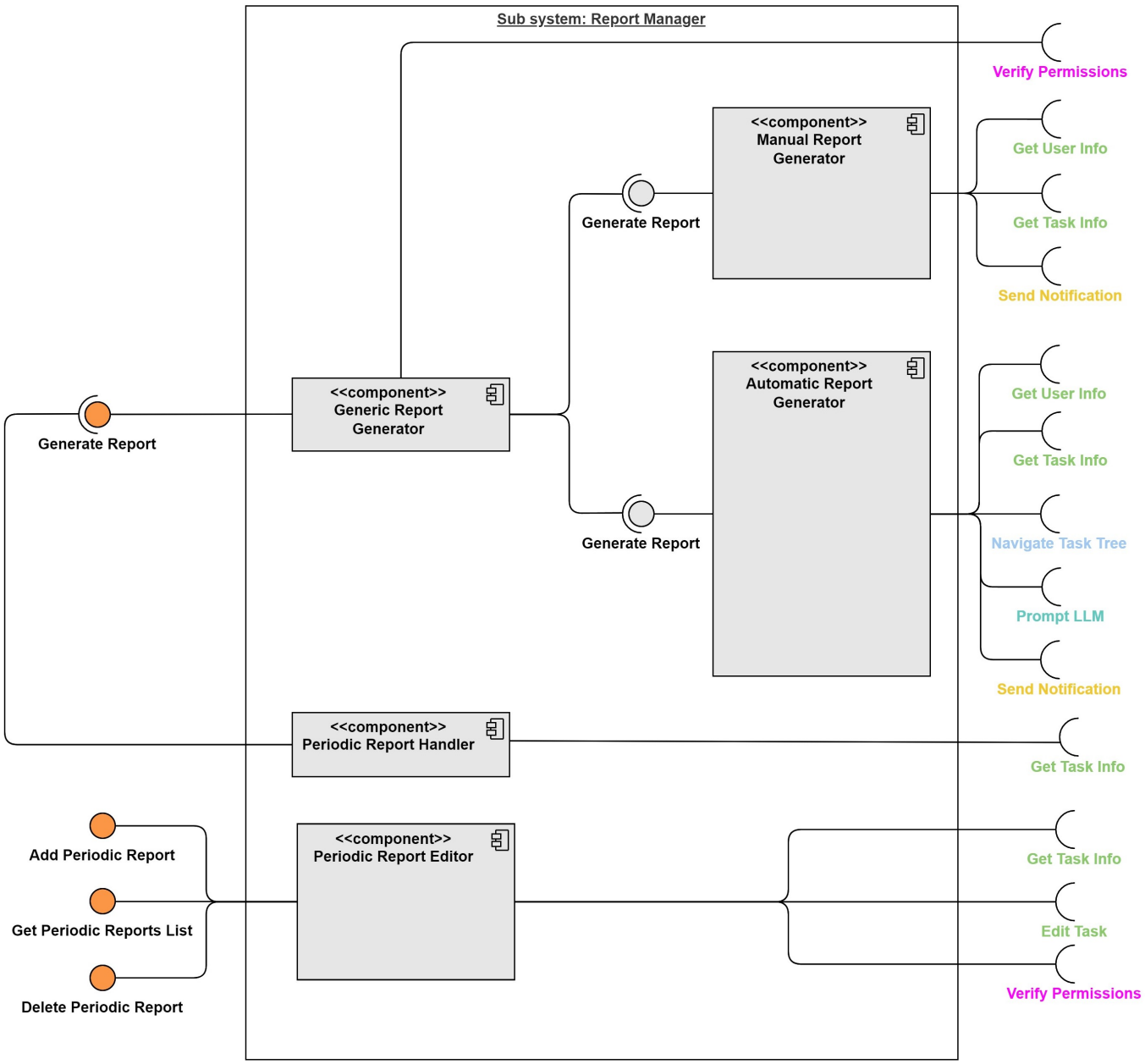



### 7.5.2 Descriptions

This component provide some library useful to navigate the task tree. For example it can be used to find all the children of a task, up to a certain depth.

## 7.6 Report Manager

### 7.6.1 Diagram




### 7.6.2 Descriptions

The report manager got two main tasks:

- Generating reports (both manually and automatically)
- Managing the report schedule

The system is composed of 5 different components:

- **Manual Report Generator:**

  This component is responsible for generating **manual reports**, generated by a user and **not** by the system. So in practice this component will just recall the report deadline.

- **Automatic Report Generator:**

This component is responsible for generating automatic reports. An **automatic report** is created by the system. This component will use some LLM APIs, as well as the data that it could find in the database to generate the report, and then it will deliver it to the user who requested it, through the notification system.

- **Generic Report Generator:**

  This component is the one that provided the "Generate Report" interface, and it only act as a **switch**, That will forward the request to the correct component (manual or automatic report generator). The component also verify that the user has the permissions to ask for the report

- **Periodic Report Editor:**

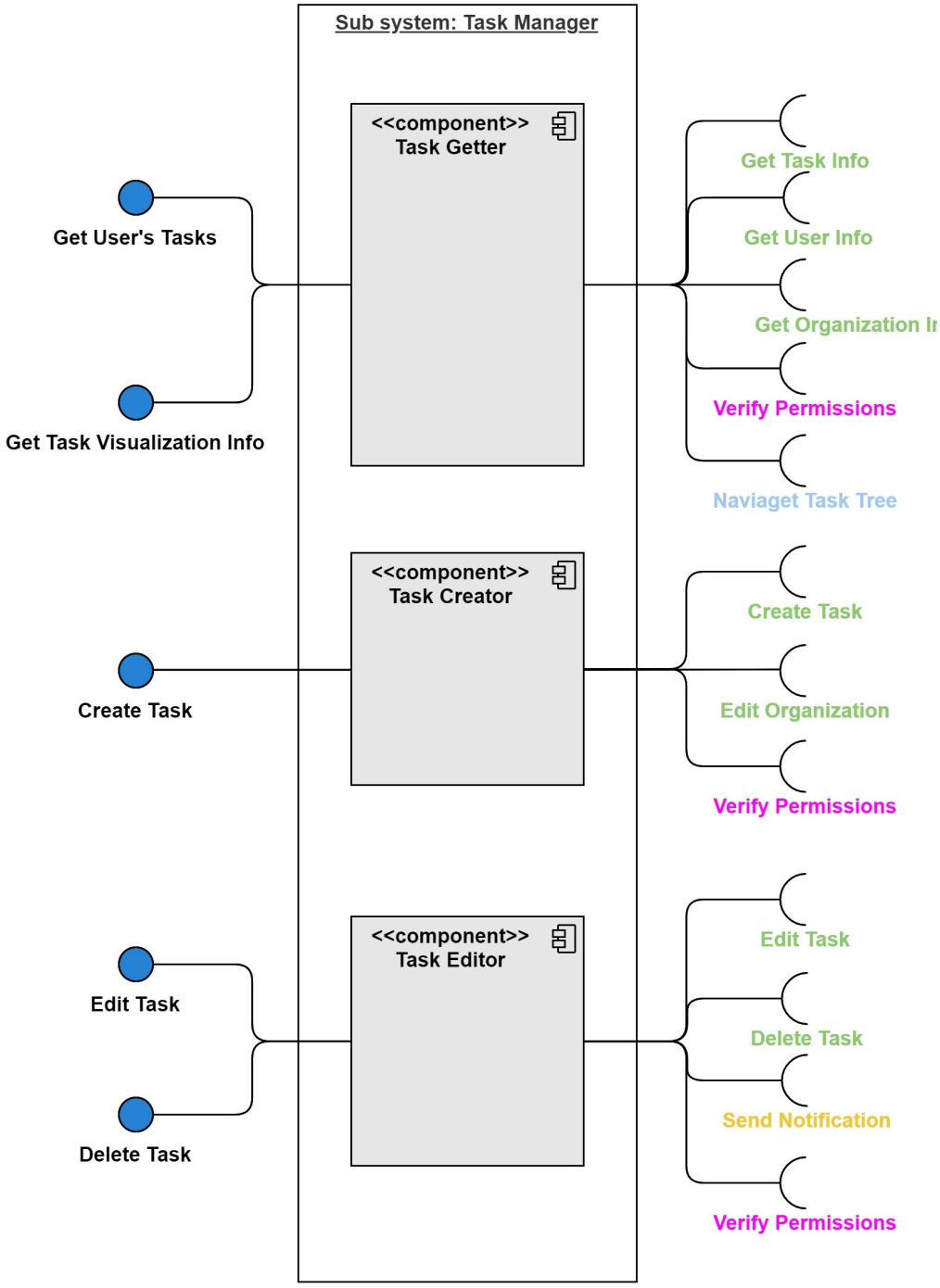
  This component is responsible for managing the report schedule. It will allow users to **set** the **frequency** of the scheduled reports, as well as to **edit** or **delete** them. Every time an operation is requested this component will verify the user's permission before executing the request.

- **Periodic Report Handler:**

  This component is responsible for handling the scheduled reports. It will read the **schedule** from the database, and then it will **trigger** a report generation event when the time comes.

## 7.7 Task Manager

### 7.7.1 Diagram




### 7.7.2 Descriptions

The task manager is the component that allow users to interact with the **tasks**. The interactions are divided into 3 main categories:

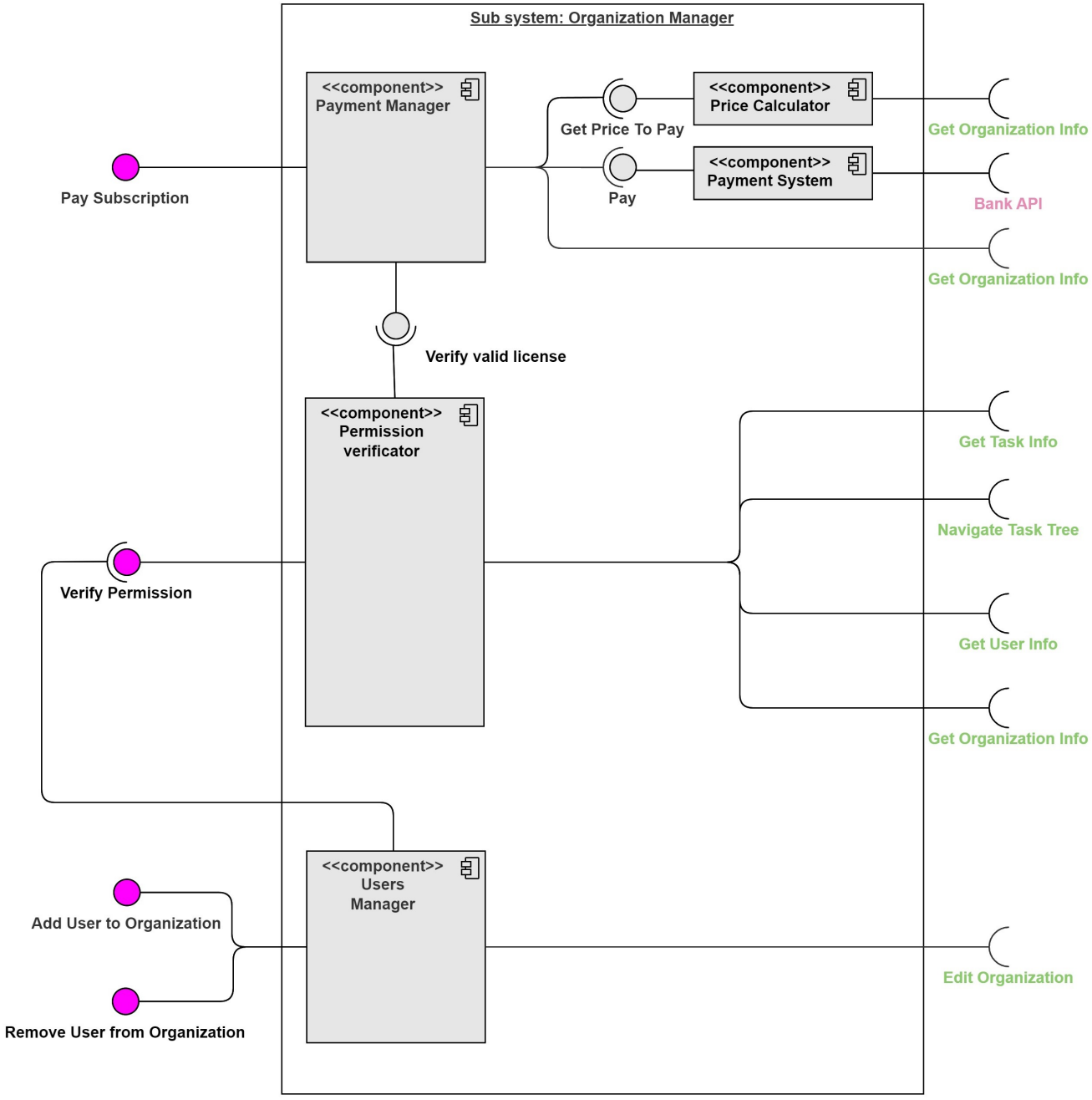
- **Task Creation:** This is the process of **creating a new task**. That include operations like setting the task name, the father task, the assignees, the managers etc. This operations are handled by the "Task Creator" component.
- **Task Modification:** This is the process of **modifying** an existing task. That includes operations like adding a description or some notes, changing state, deadline assignee etc; or even deleting the task. This operations are handled

by the “Task Editor” component; This component will also send a notification to the manager of the task if the notification are enabled for the particular task.

- **Task Reading Operation:** This process is responsible for **reading** the task data from the database, and then serve it to the front end. These operations are handled by the “Task Getter” component.

## 7.8 Organization Manager

### 7.8.1 Diagram




### 7.8.2 Descriptions

The organization manager has 3 main tasks:

- **Allowing the manager to pay a subscription**

  The manager shall be able to **pay** the **subscription** of our software, this is done by the component “Payment Manager”. The component has 2 subordinate components:

- **Price Calculator:** This component **calculates** the **price** the owner should pay, based on the **software usage** amount
- **Payment System:** This component is used to connect with the necessary **banking** system to handle payments

- **Permission Verificator**

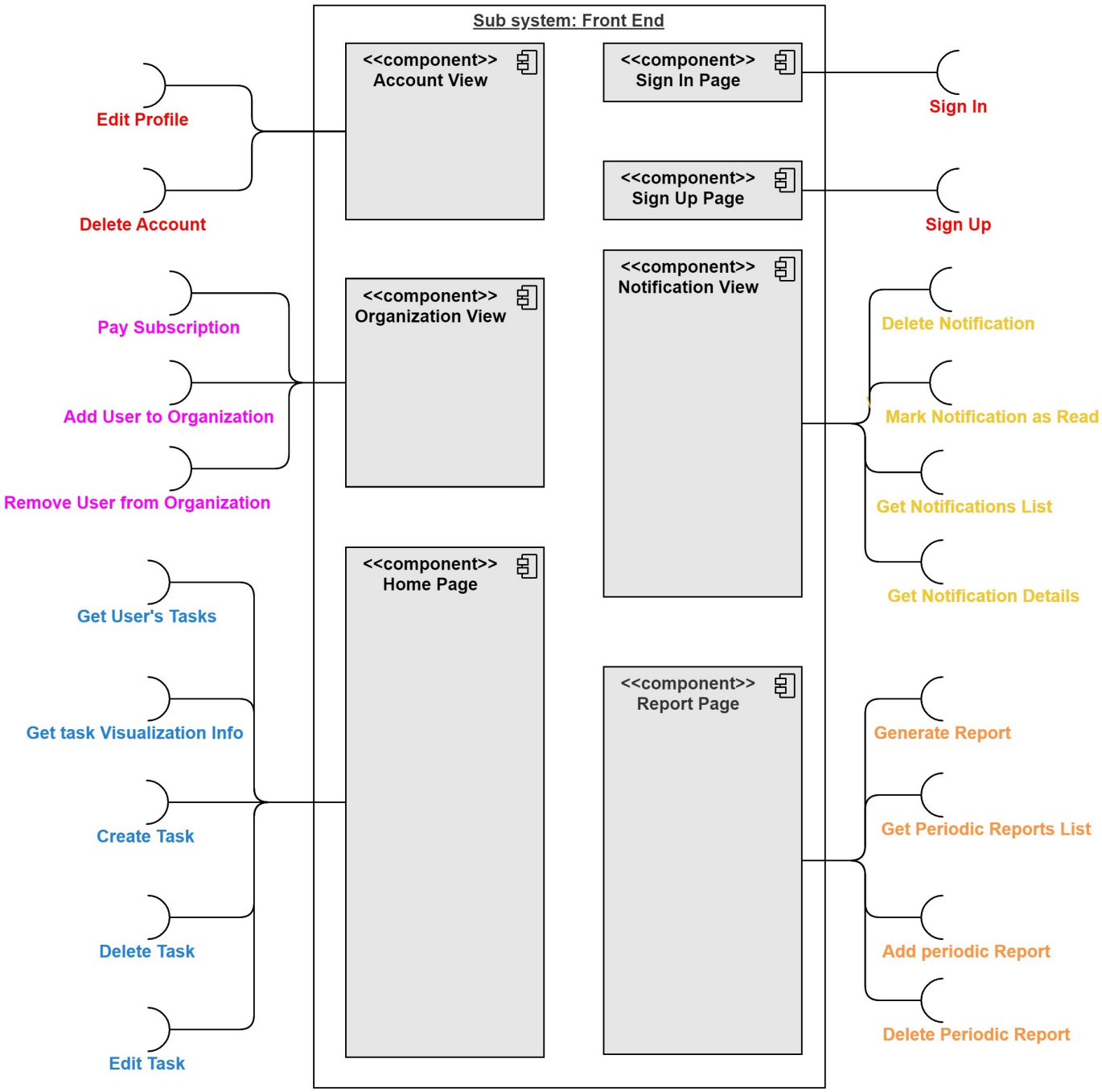
  This component is responsible for **allowing** or **denying** users **actions**. It will consider the permissions set in the task by connecting to the database, and it will also check the license validity, so we can limit certain actions if the owner hasn't renewed the subscription.

- **User Manager**

  This component is the one that will allow an owner to add or remove users form his organization.

## 7.9 Front End

### 7.9.1 Diagram



### 7.9.2 Descriptions

In the front end diagram we decided to create one component for every page of the application, for a total of 7 components:

- **Sign In Page:** In this page the users will be able to sign in to the application.
- **Sign Up Page:** In this page the users will be able to sign up for the application
- **Home Page:** This is the main page of the application, where the user will be able to see/edit/create all the tasks of the organization.
- **Report Page:** This page will allow users to ask for a report, or set up a Periodic Report.
- **Notification View:** This page will allow users to see all the pending notifications.
- **Organization View:** This page will allow the organization owner to manage the organization, including things like adding/removing users, or paying the subscription.
- **Account View:** This page will allow the user to manage his account, including things like changing the password, or deleting the account.

# 8 Class Diagram

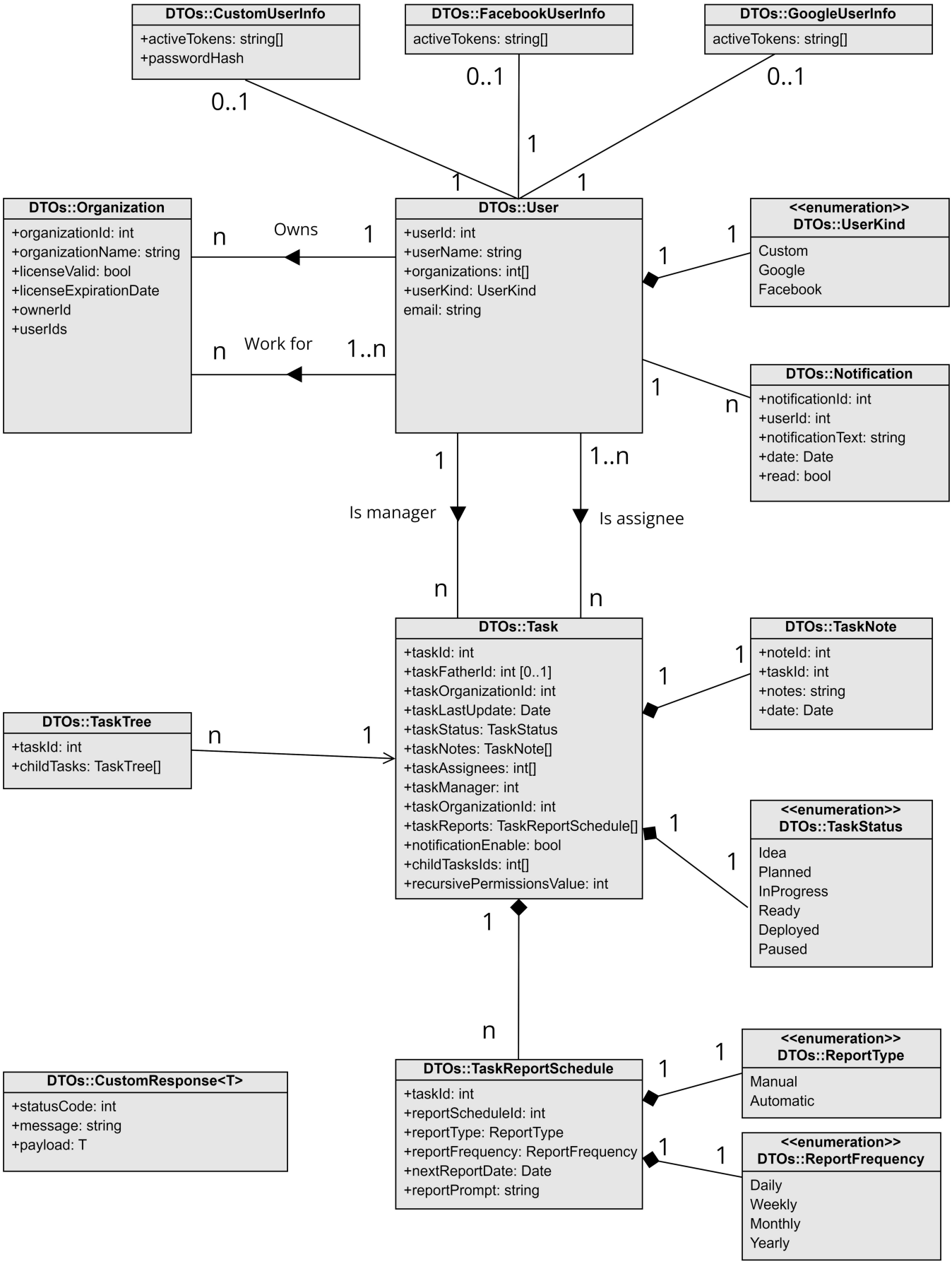
This section will illustrate the class diagram of the system. We have divided the class diagram in different modules, each of them in a separate diagram.

All the class names are preceded by the name of the module they belong to, to avoid confusion.

In some cases a class can inherits from a class that is defined in another module, in this case we have represented the foreign class, with the name and tre dots in the place where attributes and methods should be.

## 8.1 Data Transfer Objects

### 8.1.1 Diagram



### 8.1.2 Description

The data transfer objects (DTOs) are used to represent all the objects of the system;
In some classes there is a "Date" type used, this is an external dependency from the JavaScript standard library.
A list explaining all the classes follows:

- **ReportType:**

  This is an enum that represents the two type of reports that our system can handle: **Manual** and **Automatic**.

- **ReportFrequency:**

  This is an enum that represents the various **frequency** a periodic report can have.

- **TaskReportSchedule:**

  This is a class that represents **one report scheduled** for a specific day, and with a specific frequency. This will be used by the report system to memorize and execute all the scheduled reports.

- **TaskNote:**

  This class represents a **note** that a **user** add to a specific task. A task can have more that one note, and this is represented in the class diagram.

- **TaskStatus:**

  This is an enum that represents the various **status** a task can have.

- **Task:**

  This is the class that represents a **Task**.

  A task has a double **associations** with users, represented by the two arrows named "Is manager" and "Is assignee". As shown in the graph, a task must have exactly one manager and at least one assignee.

  A task is composed of various primitive attributes, like title, task id etc. It also got some complex attributes, like a list of `TaskNotes`, a list of `TaskReportSchedule` and a `TaskStatus`. This composition is represented in the class diagram.

- **TaskTree:**

  A Task Tree is a **recursive** data structure that represents a tree of tasks, starting from a root. The tree has a root task, and a series of TaskTree children.

  A TaskTree is associated to one task (the root task). This association is only navigable from the TaskTree to the Task - and **not** the other way around - as one task may be associated with more than one TaskTree. This is because the visibility of the tasks depends on the user that is looking at the tree.

- **UserKind:**

  This is an enum that represents the various **kinds of user** that may be in the system. This derives form the way a user signs up.

- **Notification:**

  This class represents a **notification** issued to a particular user in a specific date.

- **Custom/Facebook/GoogleUserInfo:**

  Depending on the method in which an user could have signed up, we may require some additional information that has to be stored in the database. This 3 classes allows us to extend these informations for specific users.

- **User:**

  This class represents a **single user** of the system. Each user could either be linked to tasks (as a manager or an assignee), and organizations. The link to an organization is represented by the association named "Owns" or "Works for".

  An user can also be associated with an unlimited number of notifications and (depending on the type of sign up) with one of the UserInfo between `CustomUserInfo`, `FacebookUserInfo` and `GoogleUserInfo`,

- **Organization:**

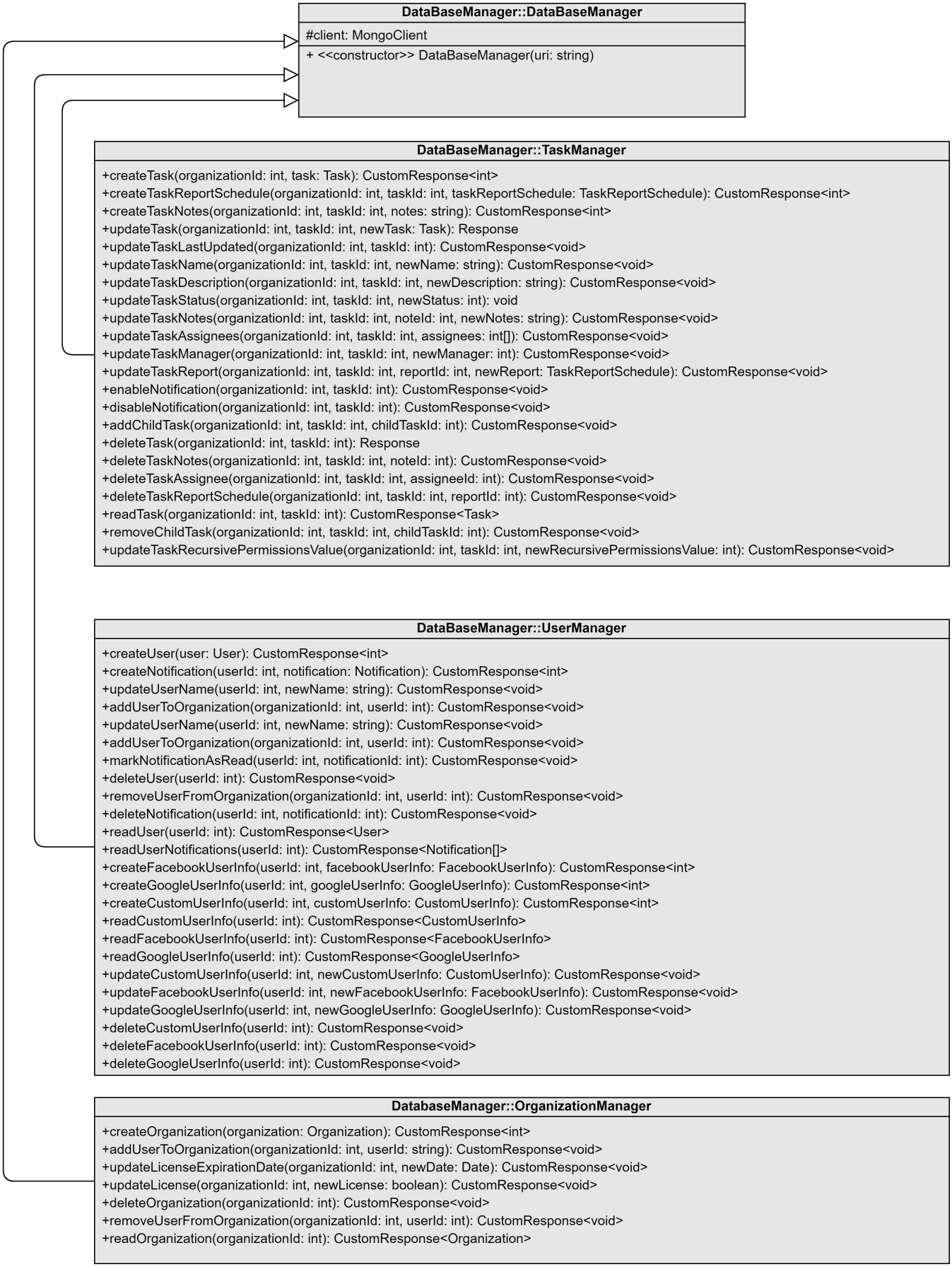
  This class represents an **organization**. It has a double association with users, one that represent the owner and one for the assignees.

- **CustomResponse**

  This is a custom class, that is going to be used as a **response** to all our APIs. It contains a status **code**, a **message** and a **payload**. The payload is a **generic** object represented with the type T

## 8.2 Data Base Manager

### 8.2.1 Diagram



### 8.2.2 Description

The Data Base Manager is used to store, retrieve and manage every info and object regarding tasks, users and organizations, such as the user's notes files and the tasks attributes. This includes:

- **DataBaseManager:**

  A minimalist class containing a constructor only used to instantiate the Mongo driver, required to intercommunicate with MongoDB Atlas. This class is not part of our system, and is an external dependency.

- **TaskManager:**

  A class containing all the methods required for the task system to work. As an instance, That includes methods for the creation, the attributes modification (e.g. report schedule, name) and the subtask management.

- **UserManager:**

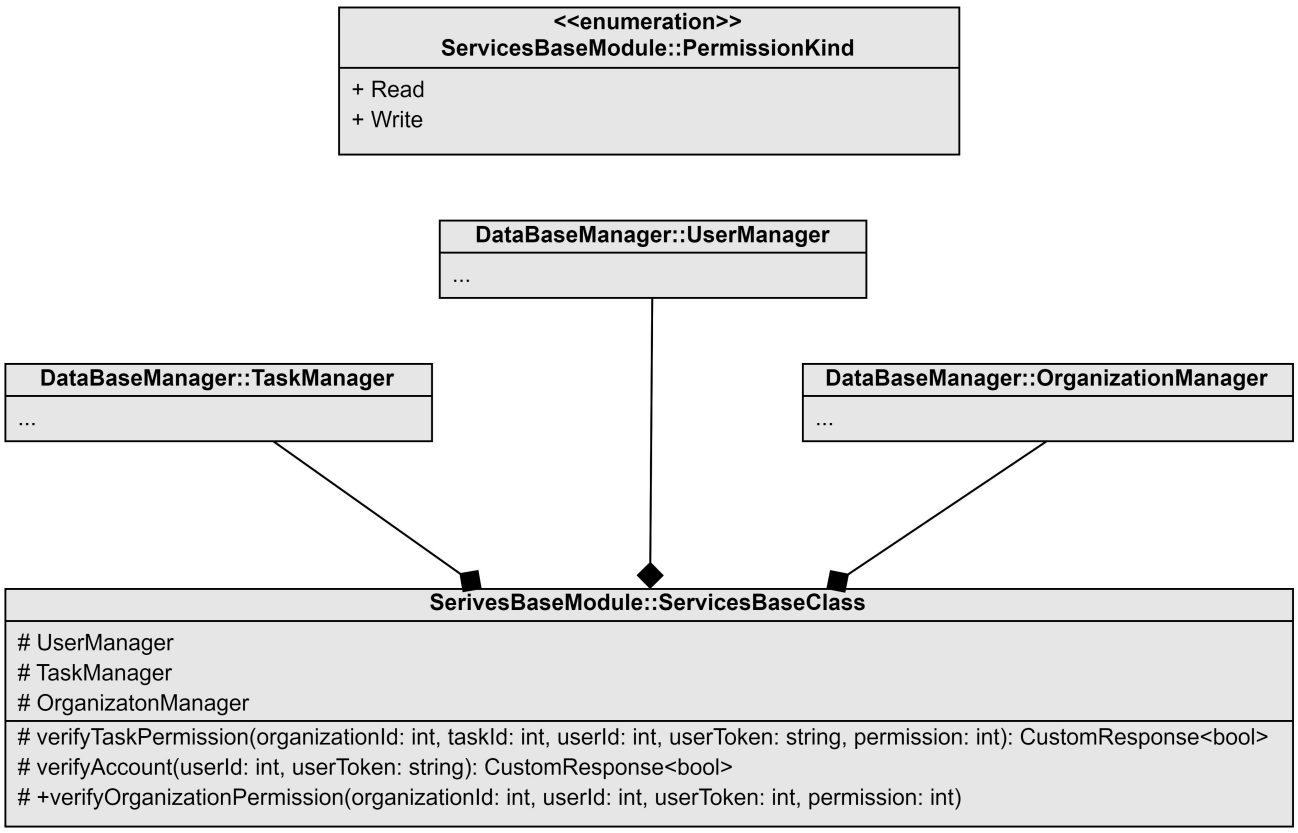
  A class containing all the methods regarding the user management system. That includes methods to create and update user info.

- **OrganizationManager:**

  A class containing all the methods related to the organization itself, managing its creation, deletion and inner user management.

## 8.3 Services Base Module

### 8.3.1 Diagram




### 8.3.2 Description

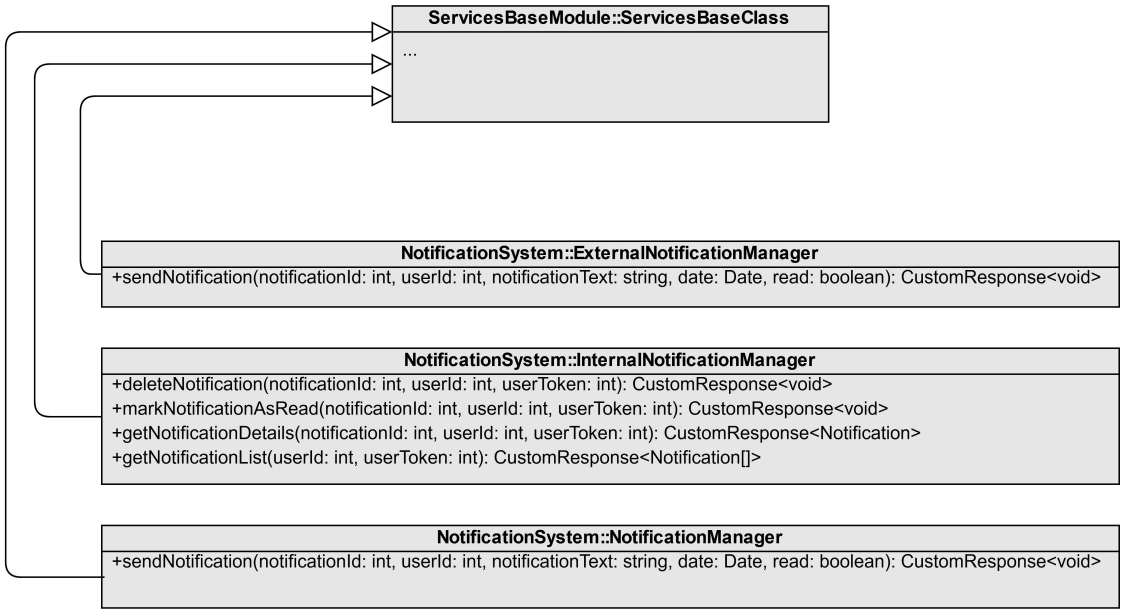
This module is primarily used to define the `ServicesBaseClass`. This class has 3 attributes (that are used to interact with the database), and 3 methods (that are used to verify users or organizations permissions).

This class is meant to provide a base1 for all the services of the system.

The enum `PermissionKind` is necessary to represents the Read and Write permission attributes.

## 8.4 Notification System

### 8.4.1 Diagram




### 8.4.2 Description

The notification system is used to *send and receive* notifications by the users and to *view and handle* existing notifications, by using an internal and an external manager. Every notification is associated with a specific user, a date and a flag to see if it has been read. The system is composed by:

- **External Notification Manager**

  This class refers to the external notification system. It implements the possibility to send notifications via emails to a specific user.

- **Internal Notification Manager**

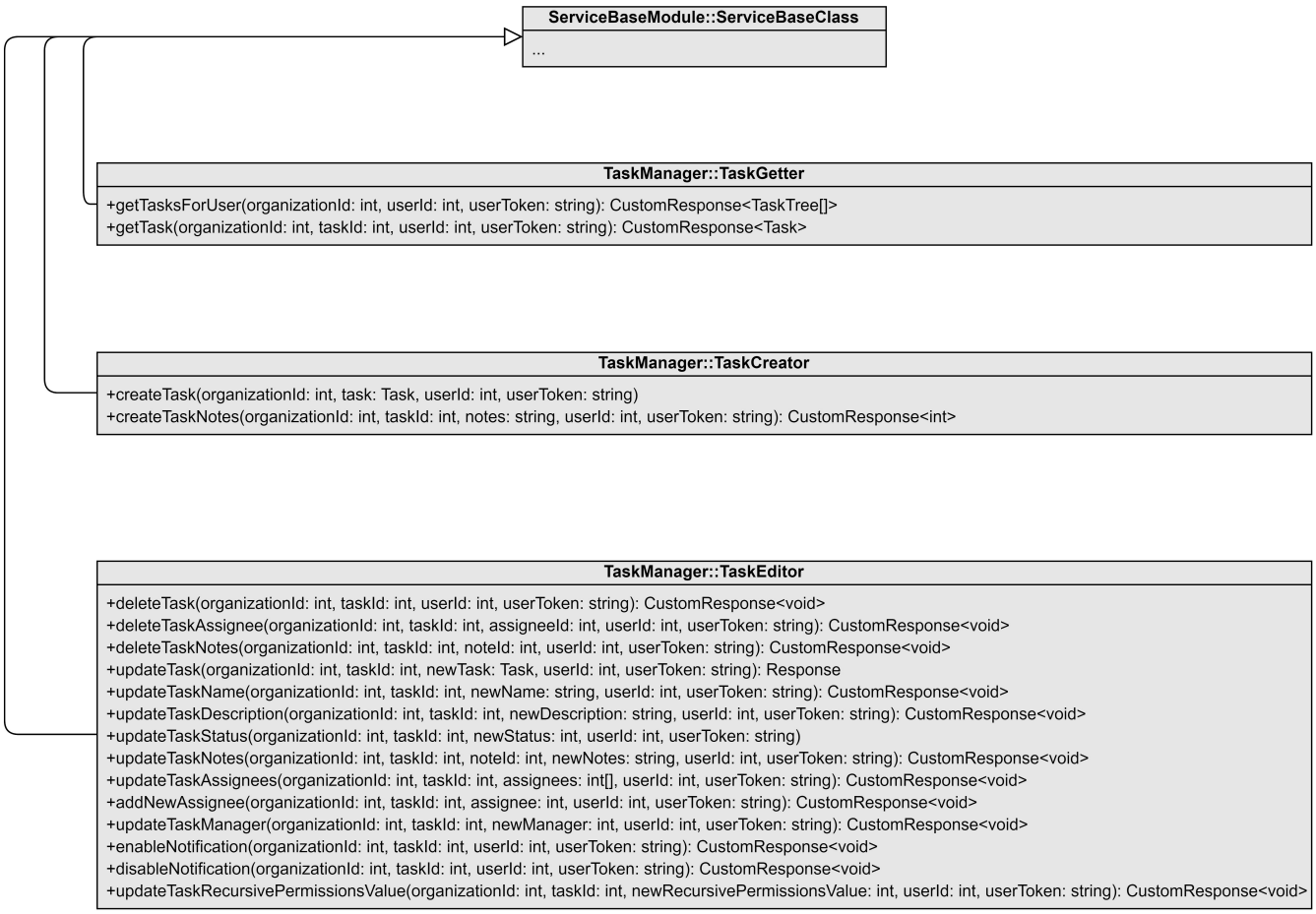
  This class refers to the internal notification system, that works in-app. It is possible to delete sent notifications, to mark them as read, to get details about one of them and to get the list of notifications associated to a specific user.

- **Notification Manager**

  This class refers to the **generic** notifications handler. This component acts as a switch, so it decides whether a notification should be sent via external or internal system.

## 8.5 Task Manager

### 8.5.1 Diagram



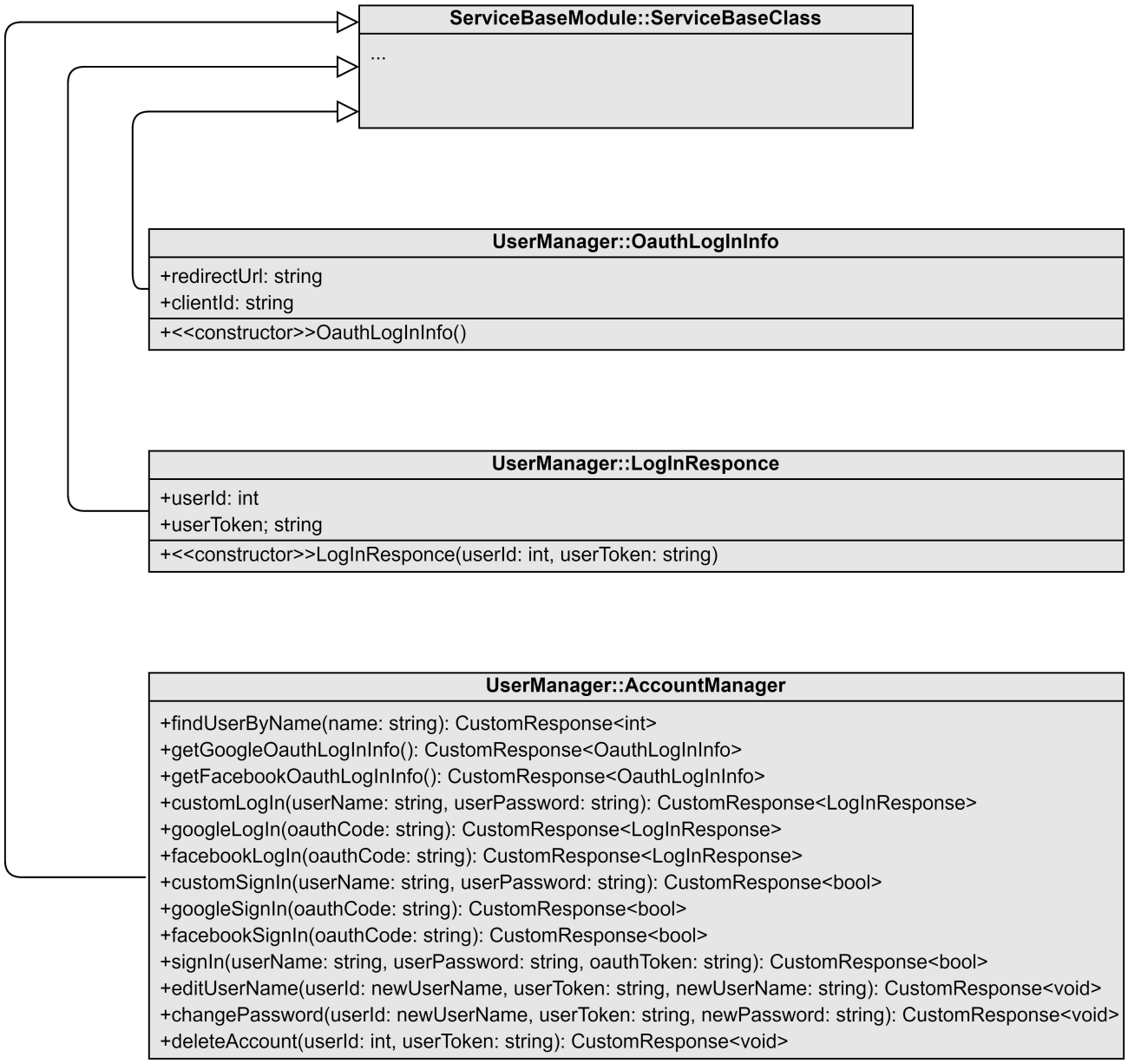

### 8.5.2 Description

The *Task Manager* module allows users to interact with the tasks. Here is a list containing each class of this module:

- **TaskGetter** This class *returns* any task **requested** by the user for his specific organization. It can either respond with the entire task tree or a specific one.
- **TaskCreator** This class allows users to create **new** tasks, or even add notes to already existing ones.
- **TaskEditor** This class include various methods for task **modification**. Some of the functionalities included are:
  - Addition and removal of assignees in a specified task
  - Update task name
  - Edit or delete task notes
  - Edit or delete specified Task
  - Enable or disable notification for specified task

## 8.6 User Manager

### 8.6.1 Diagram




### 8.6.2 Description

The User manager system is used to provide functionalities for the user's account management.
Once connected to the ServiceBaseClass discussed above, it provides the following classes:

- **OauthLogInInfo:**

  A class which contains the link for the redirection and the clientId as a string.

- **LogInResponse:**

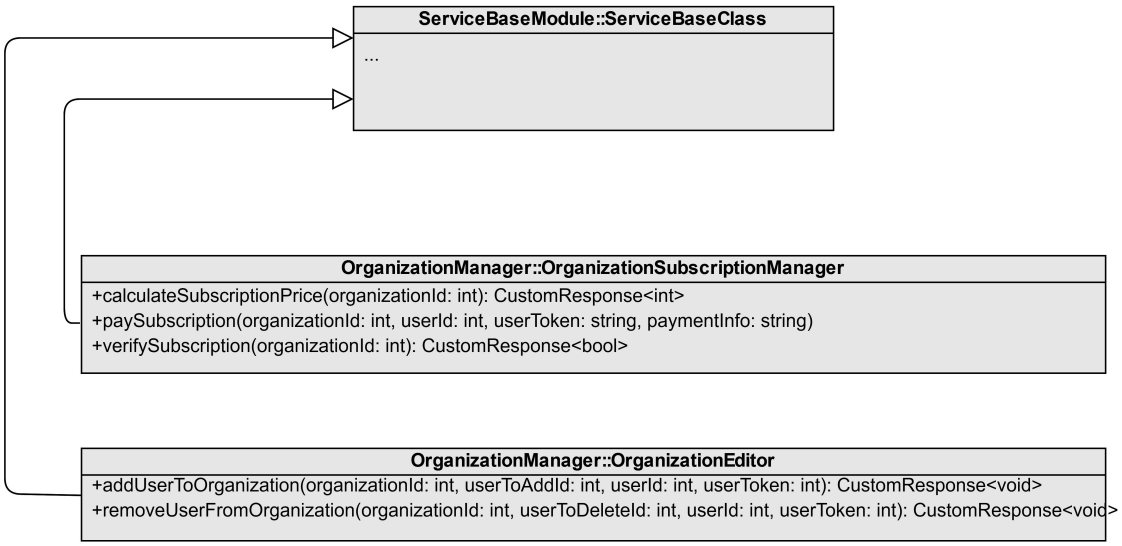
  A class containing the ID of the user and the string based on the response.

- **AccountManager:**

  A class which contains all the methods used to manage the accounts, such as the logins management via external APIs or the change of the password.

## 8.7 Organization Manager

### 8.7.1 Diagram



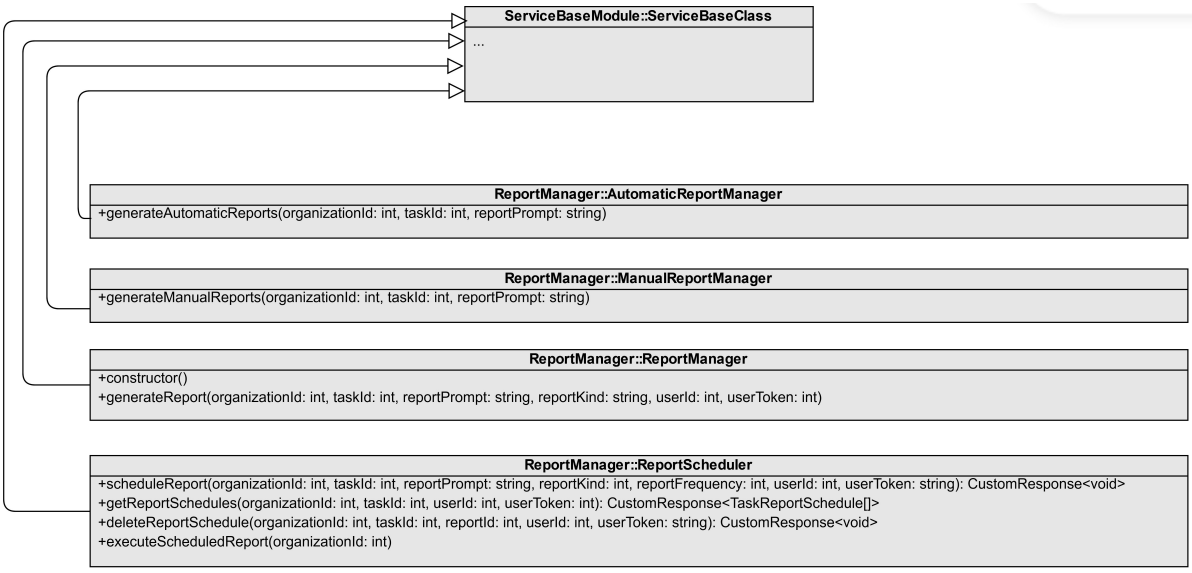

### 8.7.2 Description

The Organization Manager is a module that provide some actions used by the Owner of an organization. The functionalities provided varies from the **payment** of the subscription to the **addition or removal** of users from the organization. This module is divided into the following classes:

- **OrganizationSubscriptionManagement** This class handles the subscription side of the Organization. It contains methods for:
  - subscription verification and payment
  - calculation of total subscription fee
- **OrganizationEditor** This class allows Owners to add and remove new members in their Organization

## 8.8 Report Manager

### 8.8.1 Diagram



### 8.8.2 Description

The report manager is responsible for generating reports and managing report schedule. It is composed by:

- **Automatic Report Manager** This class refers to the **automatic report generation**, implementing the possibility to generate the report. It is connected to external APIs that will use information available in the database to complete the process.
- **Manual Report Manager** This class refers to the **manual report generation**, so report that are realised manually by users on request.
- **Report Manager**

  This class refers to the **generic** report manager, that will decide if a report should be generated automatically or manually.
- **Report Scheduler**

  This class referes to the **report scheduler management**. It implements the possibility to schedule the sending of a report, get a list of the scheduled reports for a specific user, delete report requests scheluled for a specific user and execute report request that have been scheduled.