

Management Tool Application

Microservices and micro frontends

Overview

The management app is a comprehensive **web-based application** designed to streamline various administrative tasks and improve organizational efficiency. The app offers a user-friendly interface and robust functionality to meet the diverse needs of managers, administrators, and employees across different departments.

The app includes modules for task and project management, allowing users to create, assign, and track tasks across various projects. Users receive timely notifications and alerts regarding important events, task assignments, deadlines, and updates within the app. Customizable notification settings enable users to stay informed and respond promptly to critical information.

Architecture

The architecture follows a microservices-based approach, leveraging technologies for service discovery and an API gateway to manage external client requests. The frontend is composed of micro frontends, which allows for modular development and independent deployment of frontend components. Fig.1

Users interact with the frontend through the browser or other client applications, accessing various features and functionalities seamlessly integrated into the user interface.

Frontend to API Gateway: micro frontends communicate with the API Gateway through HTTP protocols.

API Gateway to Microservices: API Gateway forwards requests to the appropriate microservices based on routing rules. Communication between the API Gateway and microservices also occurs via HTTP.

Microservices to Eureka: Microservices register themselves with Eureka upon startup using HTTP/HTTPS.

Microservices to Microservices (via Pub/Sub systems): Microservices communicate asynchronously via Pub/Sub systems. They can publish messages to topics/queues and subscribe to topics/queues for message consumption.

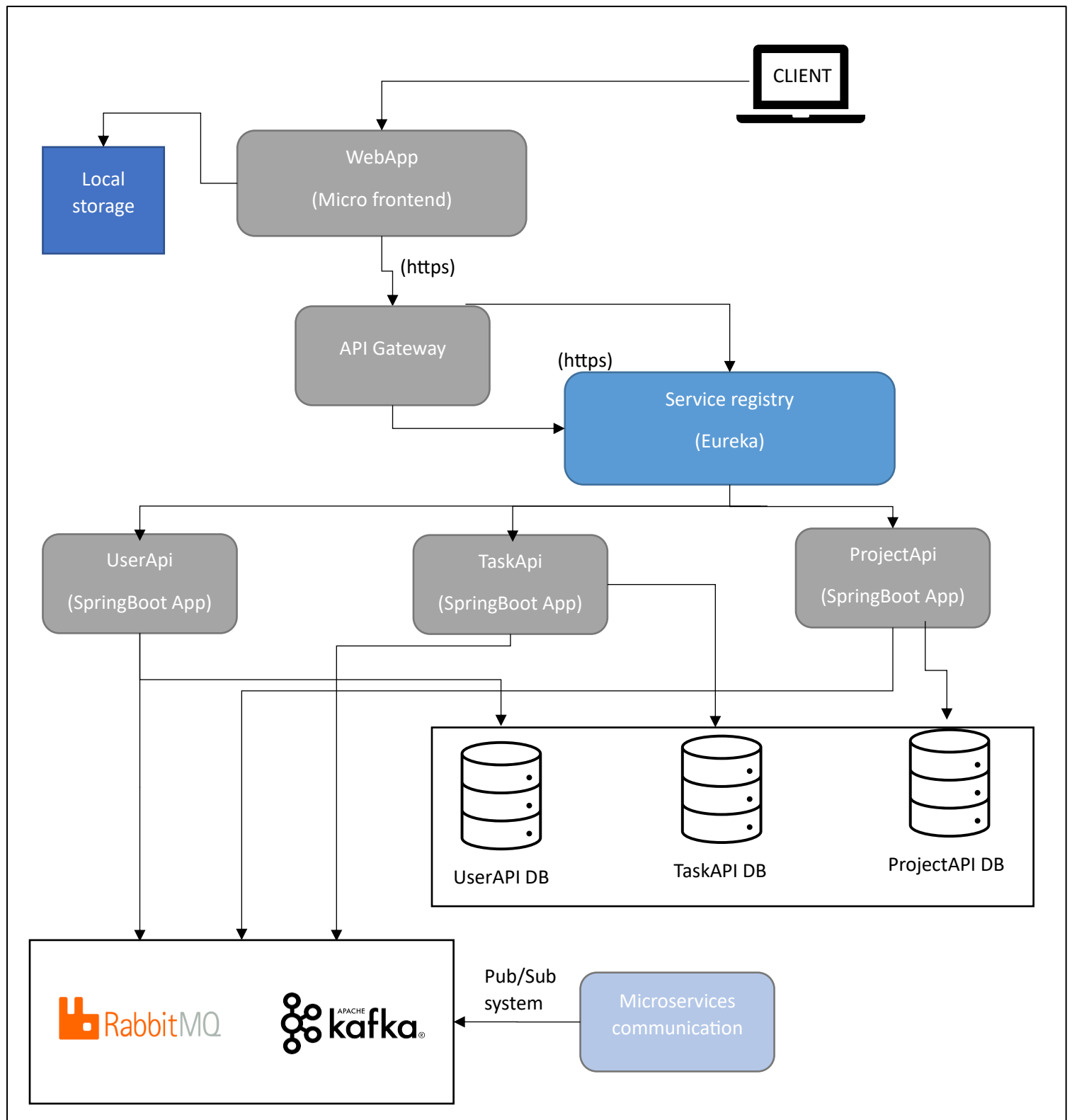


Fig. 1 - High-level architecture

Frontend

The architectural approach is micro frontends where a frontend application is composed of multiple smaller, independently deployable applications. Each micro frontend is responsible for a specific feature or functionality.

Module Federation allows for dynamic loading of modules at runtime. The application integrates Module Federation to compose micro frontends into a cohesive system. This approach enables seamless communication and sharing of components between micro frontends while maintaining independence.

Overall, the frontend architecture leverages React, Webpack with Module Federation, and other libraries to build modular, scalable, and maintainable micro frontends.

In this architecture, **React** was chosen as the primary framework for developing both micro frontends due to its popularity, robust ecosystem, and flexibility in building reusable UI components. In this setup, **Webpack** played a crucial role in enabling Module Federation, a feature that facilitates dynamic module loading and integration of micro frontends into a single application. It allows for seamless integration and communication between micro frontends while maintaining their independence.

The two micro frontends communicate with each other via **Module Federation**. The first micro frontend serves as a global state store, responsible for managing and storing the application's global state. The second micro frontend, representing the actual user interface, consumes the global state provided by the first micro frontend and updates its UI accordingly. It may subscribe to changes in the global state and trigger re-renders of relevant components when the state changes. The app is mainly composed of the following pages: Login, Registration, Dashboard and Task.

Backend

The backend architecture comprises three microservices, a service registry, and an API gateway to ensure robustness, scalability, and efficient communication between components.

As the main framework the app is built using **Spring Boot**, providing a lightweight and efficient runtime environment. **Hibernate** is used to facilitate database operations and simplify data persistence. Also, some important libraries and tools used are: Lombok that helps reduce boilerplate code, **MapStruct** a tool used for mapping between different Java bean types.

As **Pub/Sub systems**, **Kafka** and **RabbitMQ** are used for building scalable and distributed messaging systems. In this architecture, Kafka and RabbitMQ are employed as message brokers to facilitate communication and event streaming between microservices, enabling asynchronous message processing and decoupling of components.

RESTful APIs are implemented to expose functionalities and enable communication between the frontend and backend components.

Microservices register themselves with **Eureka** when they start up. During registration, each microservice provides metadata such as its name, network location (host and port), and health

status. This information allows Eureka to maintain an up-to-date registry of available services within the system. Eureka seamlessly integrates with Spring Cloud.

Structure of the project

Microservices: ServiceRegistryApplication, ProjectServiceApplication, TaskServiceApplication, UserServiceApplication, ApiGatewayApplication and another one for common classes and its main functionality is as a library.

Each service uses Controller-Service-Repository pattern.

The Controller exposes the REST API for each service and serves as the entry point for external interactions with the application.

The Service contains the business logic and application-specific functionality of the system. It encapsulates the core operations.

The repository layer is responsible for interacting with the underlying data storage mechanisms and provides a unified interface for performing CRUD operations on data entities.

Pub/Sub systems – Configuration

Kafka – App Architecture Fig. 2

When executing CRUD operations on a task the service Task Service sends an event to notify the change. The consumers Project Service and User Service consume the message from the topic to update the related data such as notifications.

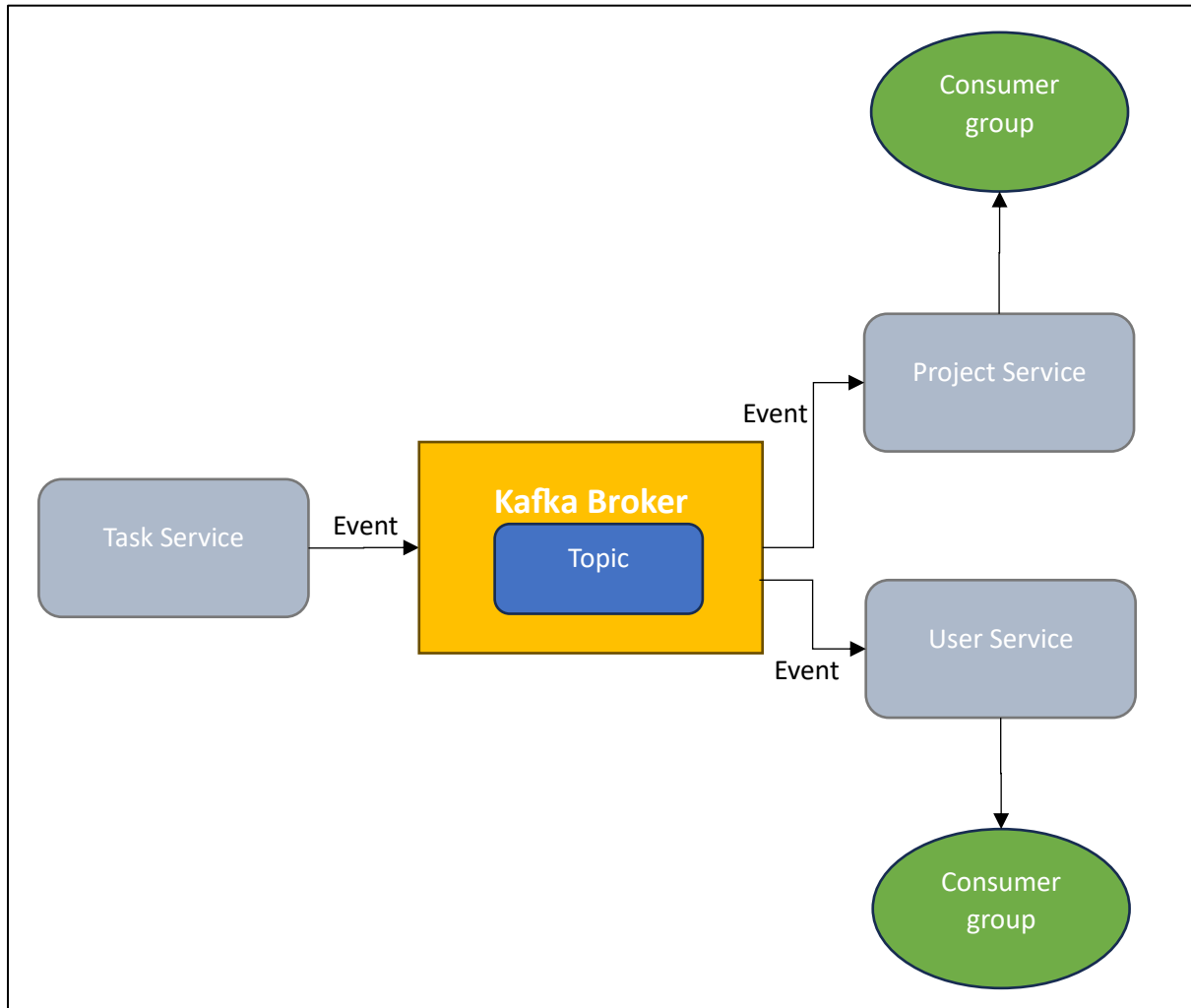


Fig. 2 – Kafka architecture in the management tool app

RabbitMQ – App Architecture Fig. 3

When executing operations in Project Service, it sends an event on the queue to notify the change. The consumers User Service consumes the message from the queue to update the related data in the corresponding table such as assigned projects.

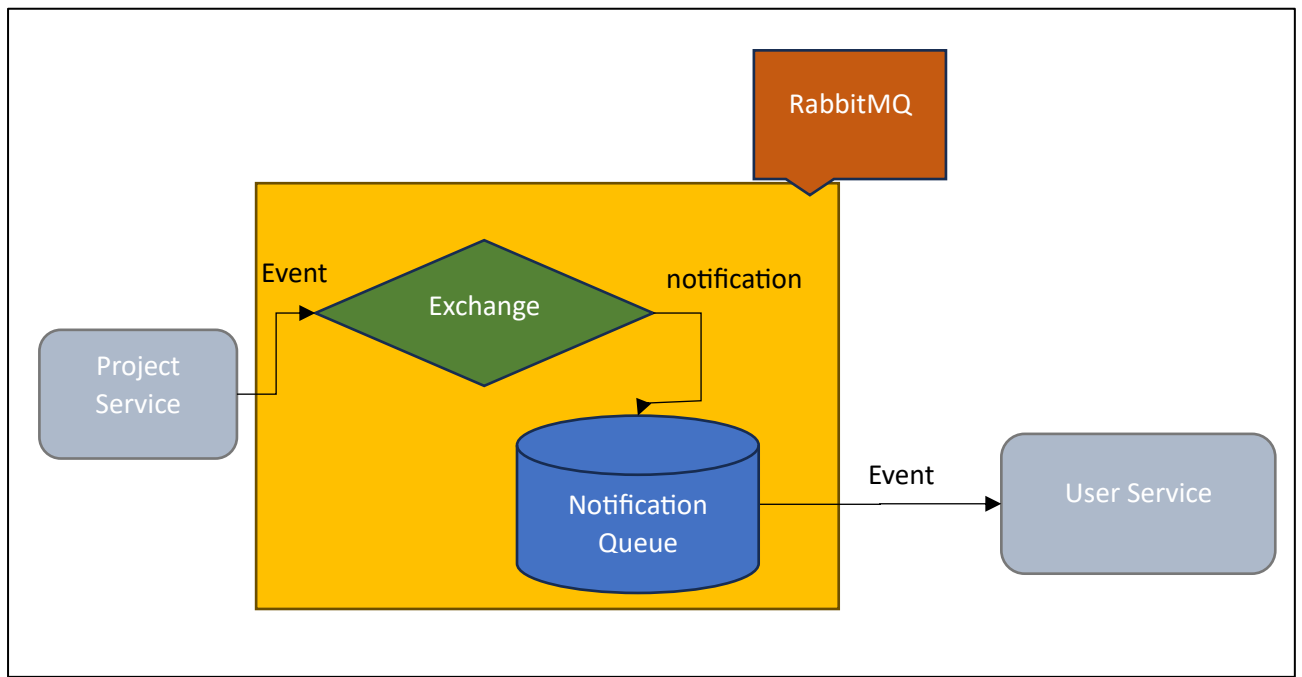
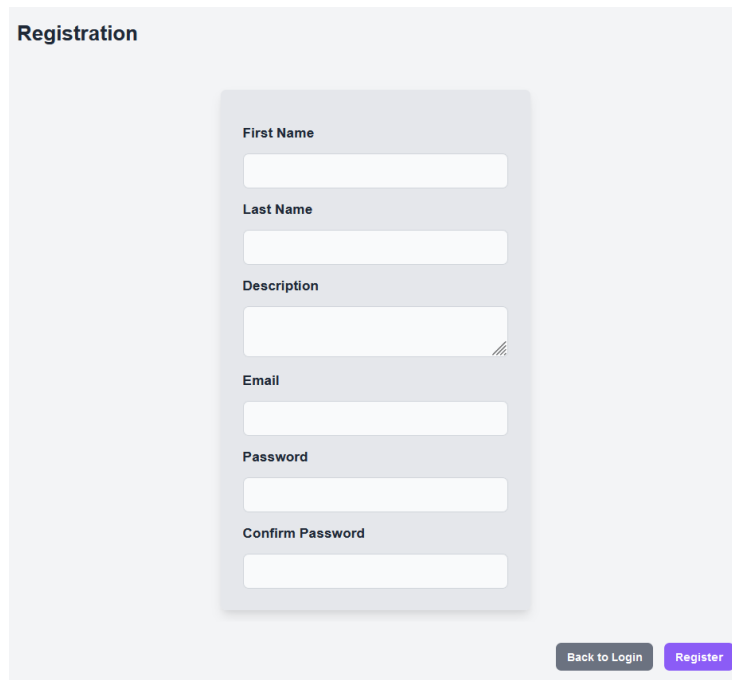


Fig. 3 – RabbitMQ architecture in the management tool app

Application – Main flow

When opening the web app, the first page is the Login page which gives access to the whole management tool. Before Logging in, the user must create an account by registering and providing some unique information. Fig. 4



The registration form is titled "Registration" and is contained within a light gray box. It features six input fields: "First Name", "Last Name", "Description" (with a small icon on the right), "Email", "Password", and "Confirm Password". At the bottom right of the form box are two buttons: "Back to Login" (gray) and "Register" (purple).

Registration

First Name

Last Name

Description

Email

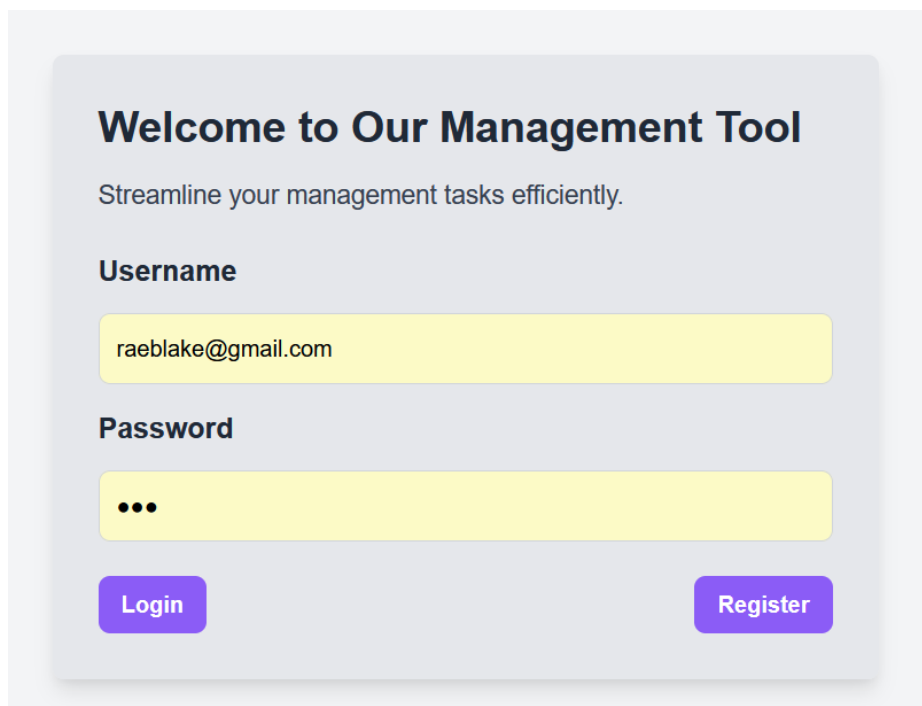
Password

Confirm Password

Back to Login Register

Fig. 4 – Registration page

After registering successfully, the user is redirected to the Login page to access his account.
Fig. 5



The login form is titled "Welcome to Our Management Tool" and is contained within a light gray box. It features two input fields: "Username" (with the text "raeblake@gmail.com") and "Password" (with three dots). At the bottom are two buttons: "Login" (purple) and "Register" (purple).

Welcome to Our Management Tool

Streamline your management tasks efficiently.

Username

raeblake@gmail.com

Password

...

Login Register

Fig. 5 – Login page

The management app opens the dashboard. Here the tasks are the focus of the user because they help him stay organised and keep track of his work. So, all the functionalities from this page are meant to be easy to access, visible, clear and comprehensive.

On the left part of the page the menu provides all the projects the user is assigned to. The main part shows cards with tasks by project and by assignee. Fig. 6

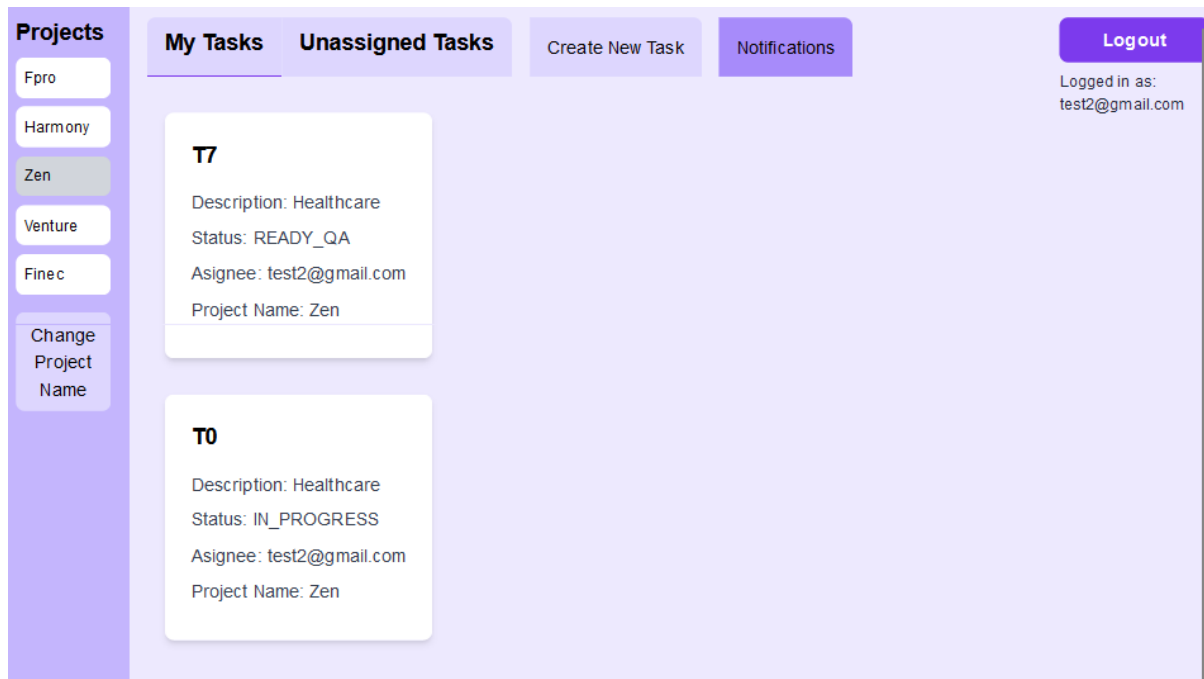


Fig. 6 – Dashboard page

For example, the user can choose to see only the tasks he is working on or other tasks that don't belong to anyone. Fig. 7

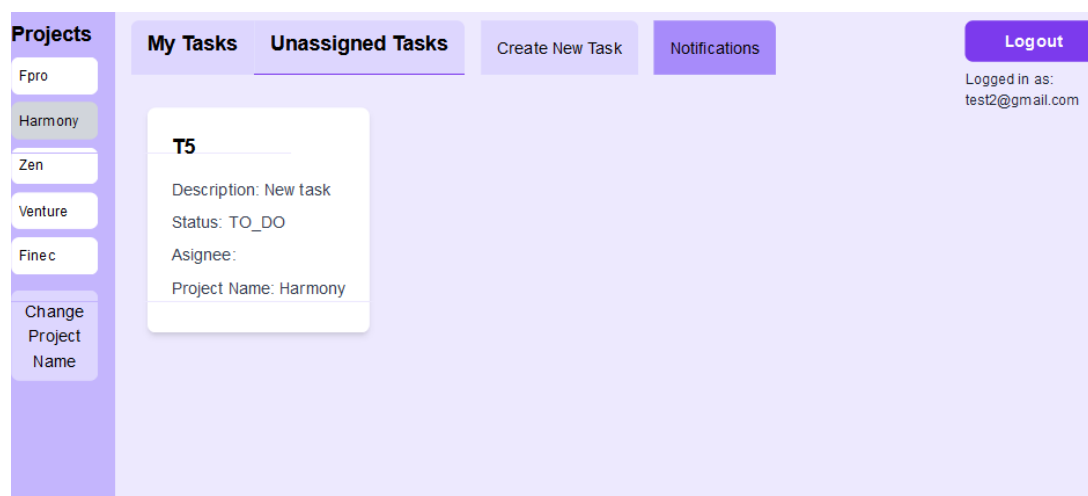
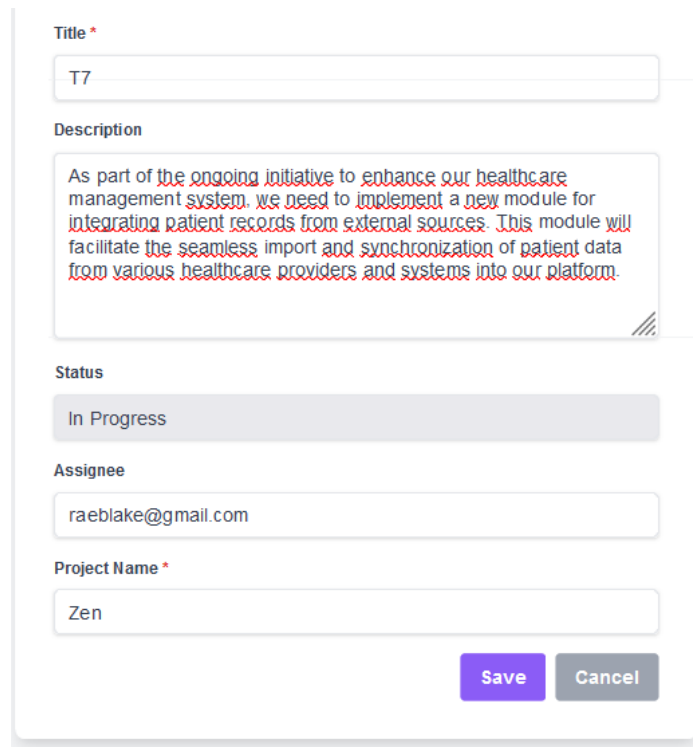


Fig. 7 – Unassigned tasks page

A task is composed of project title, description of the task, status, assignee and project name.

Each Task can be edited when pressed or just viewed in a larger window. Fig. 8

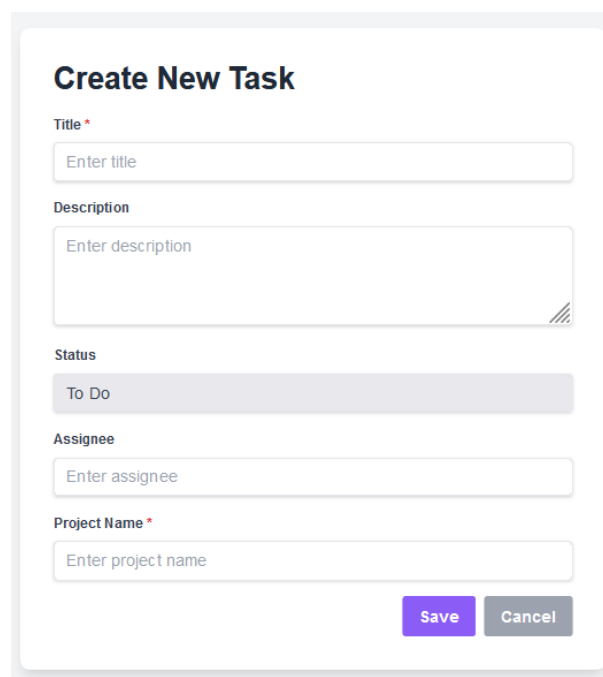


The form for updating a task is titled "Update task page". It contains the following fields and controls:

- Title ***: A text input field containing the value "T7".
- Description**: A text area containing the text: "As part of the ongoing initiative to enhance our healthcare management system, we need to implement a new module for integrating patient records from external sources. This module will facilitate the seamless import and synchronization of patient data from various healthcare providers and systems into our platform." The text is redacted with a pattern of red 'x' marks.
- Status**: A dropdown menu with the selected value "In Progress".
- Assignee**: A text input field containing the email address "raeblake@gmail.com".
- Project Name ***: A text input field containing the value "Zen".
- Buttons**: Two buttons at the bottom right, "Save" (purple) and "Cancel" (grey).

Fig. 8 – Update task page

The user can create his own tasks by pressing the “Create New Task” button. Then a new window pop up to create a new task. Fig. 9



The form for creating a new task is titled "Create New Task". It contains the following fields and controls:

- Title ***: A text input field with the placeholder text "Enter title".
- Description**: A text area with the placeholder text "Enter description".
- Status**: A dropdown menu with the selected value "To Do".
- Assignee**: A text input field with the placeholder text "Enter assignee".
- Project Name ***: A text input field with the placeholder text "Enter project name".
- Buttons**: Two buttons at the bottom right, "Save" (purple) and "Cancel" (grey).

Fig. 9 – Create new task page

Also, another main feature of the app is being notified (Fig. 10) when a ticket you are assigned to changes. For this, the notification feature helps the user to keep track of the new events such as: changing the description of a task or any other component, being assigned to a task etc.

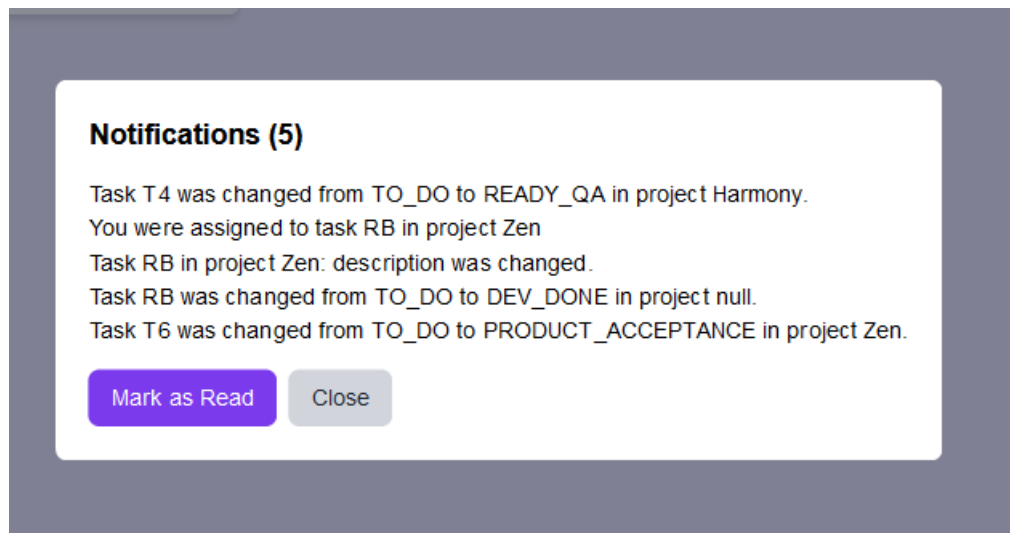


Fig. 10 – Notification pop-up

After opening the notification window, the user can mark them as read so he can keep track of the new events that change.