Notion Project: Development of a Digital Content Management Application

Santiago Alejandro Guada Bohorquez

Jhomar Armando Bojaca Landinez

Software Modeling I

Carlos Andrés Sierra

Universidad Distrital Francisco José de Caldas

October 3 , 2024

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

1. Introduction

This report details the development process of an application aimed at managing digital workspaces, inspired by the functionalities and features of Notion. The choice to replicate this application is based on its popularity and versatility, which allow users to organize information, collaborate on projects and manage tasks efficiently.

Throughout the project, surveys and interviews have been conducted with potential users to identify their needs and preferences. Based on this information, user stories have been formulated that will guide the development of the application, ensuring that the expectations and requirements of the target audience are addressed.

The main objective is to create an intuitive and functional tool that facilitates personal and professional organization, integrating features such as page creation, databases, permission management and real-time collaboration. This report will also include details on the minimum functionalities required, relevant UML diagrams and an analysis of the object-oriented programming principles applied in the development of the code. This text provides a clear context about the project and sets the foundation for what will be discussed in the rest of the report. Be sure to adjust any specific details you want to highlight or modify to suit your needs.

2. User Stories

In this section, the user stories that will guide the development of the replicated Notion application are presented. The user stories are structured into different categories, depending on the type of user that will interact with the application. Each story follows the format "As a <role>, I want <action>, so what <impact>", which allows for clear identification of user needs and expectations.

User Story Structure

-General User: Includes stories that reflect the basic needs of end users who will use the application to manage their information.
-Administrator: Focuses on the functionality required by administrators who manage permissions and application usage.
-Developer: Addresses the specific needs of developers who will work on integrating and improving the system.
-Specific Functionality: Details additional features that will improve the user experience.

The following are the collected user stories:

General User:

1. As a user, I want to register for the application, so that I can access my workspaces and personalized pages.
2. As a user, I want to log in with my credentials, so that I can securely access my information.
3. As a user, I want to create a new page within a workspace, so that I can document my ideas and projects.
4. As a user, I want to edit the content of an existing page, so that I can keep my information up to date.
5. As a user, I want to search for information on my pages, so that I can quickly find what I need.

Administrator:

6. As an administrator, I want to manage user permissions in the workspace, so that I can ensure only authorized people have access to sensitive information.
7. As an administrator, I want to view usage statistics of the application, so that I can identify areas for improvement and optimize performance.

Developer:

8. As a developer, I want to access the application's API, so that I can integrate additional features that enhance the user experience.
9. As a developer, I want to receive error reports automatically, so that I can quickly resolve any issues.

Specific Features:

10. As a user, I want to create a database to store relevant information, so that I can efficiently access and manage data.
11. As a user, I want to upload files to my workspace, so that I have access to important documents from anywhere.
12. As a user, I want to filter and sort data in the database, so that I can quickly find the information I need.

3. CRC Cards

This section presents CRC cards, which are fundamental tools in the object-oriented design process. Each card represents a class within the system, describing its responsibilities and the collaborations it has with other classes. This approach allows to clearly identify how the different components of the system interact and facilitates the understanding of the overall design of the application.

CRC Card Structure

Class: Name of the class that represents a component of the system.

Responsibility: Main functions that the class must fulfill.

Collaborations: Other classes with which this class interacts to carry out its responsibilities.

Below are examples of CRC cards for the most relevant classes of the system:

Class: User	
Responsibilities: Register a new user (register()). Log in (login()). Update the user's profile (updateProfile()).	**Collaborators:** Collaborates with the Workspace class, as a user can have multiple workspaces.

Class: Workspace	
Responsibilities: Create, modify, or delete a workspace (updateWorkspace()). Add members to the workspace (addMember()). Remove members from the workspace (removeMember()).	**Collaborators:** Collaborates with the User class, as a user can own or be a member of several workspaces. Collaborates with the Page class, as a workspace can contain multiple pages . Collaborates with the CustomDatabase class, as a workspace can have custom databases. Collaborates with the File class to manage files within the workspace.

Class: Page	
Responsibilities: Create a page (createPage()). Update a page (updatePage()).	**Collaborators:** Collaborates with the Workspace class, as a page belongs to a workspace.

Delete a page (deletePage()).	

Class: CustomDatabase	
Responsibilities: Create a new entry in the custom database (createEntry()). Update an entry (updateEntry()). Delete an entry (deleteEntry()).	**Collaborators:** Collaborates with the Workspace class, as a custom database belongs to a workspace.

Class: File	
Responsibilities: Upload files to the workspace (uploadFile()). Delete files from the workspace (deleteFile()).	**Collaborators:** Collaborates with the Workspace class, as files are associated with a workspace

4. Minimum Functionalities

The application should include the following minimum functionalities to ensure a basic and effective experience for users:

- User Registration: Allow users to create an account in the application to access their workspaces and personalized pages.
- Login: Facilitate secure access to the application using user credentials.
- Page Creation: Enable users to create new pages within a workspace, where they can document their ideas and projects.
- Page Editing: Allow users to modify the content of existing pages to keep their information up to date.
- Information Search: Provide functionality that allows users to search for information within their pages to quickly locate the desired content.
- Permission Management: Allow administrators to manage users' access permissions to workspaces, ensuring that only authorized individuals can access sensitive information.

- Statistics Viewing: Give administrators the ability to view statistics on application usage, which will help identify areas for improvement.

These functionalities are essential to establish a solid foundation in the application, allowing users to interact with the system effectively and meeting the initial expectations of the project.

5. Class Diagrams

The class diagram is a fundamental visual representation in object-oriented design, as it illustrates the system's classes, their attributes, methods, and the relationships between them. This diagram provides a clear view of the system's structure and helps to understand how different components interact.

Class Diagram Structure

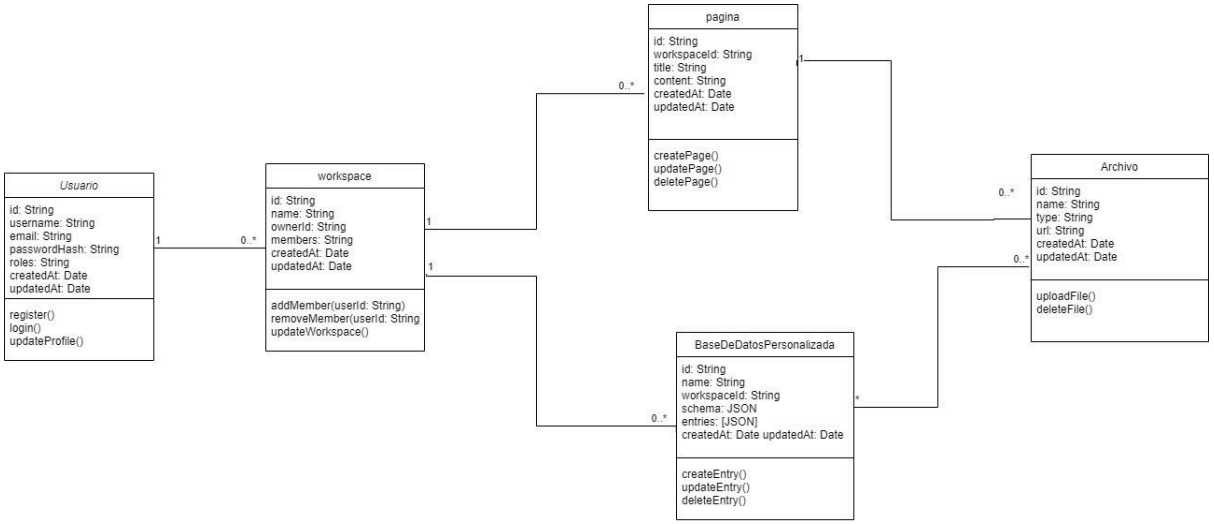
Classes: Each class represents a concept or entity within the system, such as User, Page, Workspace, and Comment.

Attributes: Attributes that define the properties of each class are listed.

Methods: Methods that describe the actions that instances of each class can perform are listed.

Relationships: Relationships between classes are shown, such as association, inheritance, or composition, allowing you to understand how they connect and collaborate with each other.

Below is the class diagram for the Notion replicated system:



This diagram is essential for guiding code development and ensuring that all required functionalities are correctly implemented. It also facilitates communication between team members by providing a clear visual model of the system.

6. UML Diagrams

Unified Modeling Language (UML) diagrams are essential visual tools in software development that allow different aspects of the system to be represented in a clear and structured way. In this project, several types of UML diagrams have been used to model the functionalities and behavior of the replicated Notion application.

Types of UML Diagrams Included

- Sequence Diagrams: These diagrams show how objects interact in a particular scenario over time. Each diagram illustrates the sequence of messages exchanged between objects to perform a specific functionality, such as user registration or page creation.
- Activity Diagram: Represents the workflows within the system, showing the activities that take place and the decisions that are made at each step of the process. This diagram is useful for visualizing how operations are executed within the application.
- Deployment Diagram: This diagram shows the physical architecture of the system, including the nodes where the software components will be deployed and how they communicate with each other.

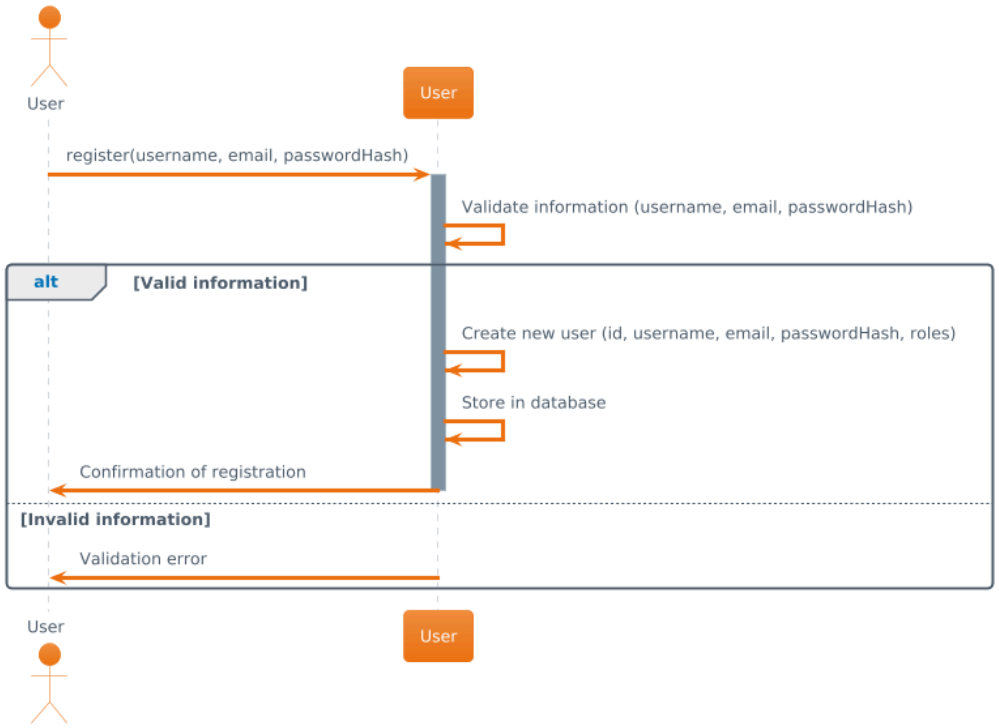
Importance of UML Diagrams

UML diagrams are crucial for:

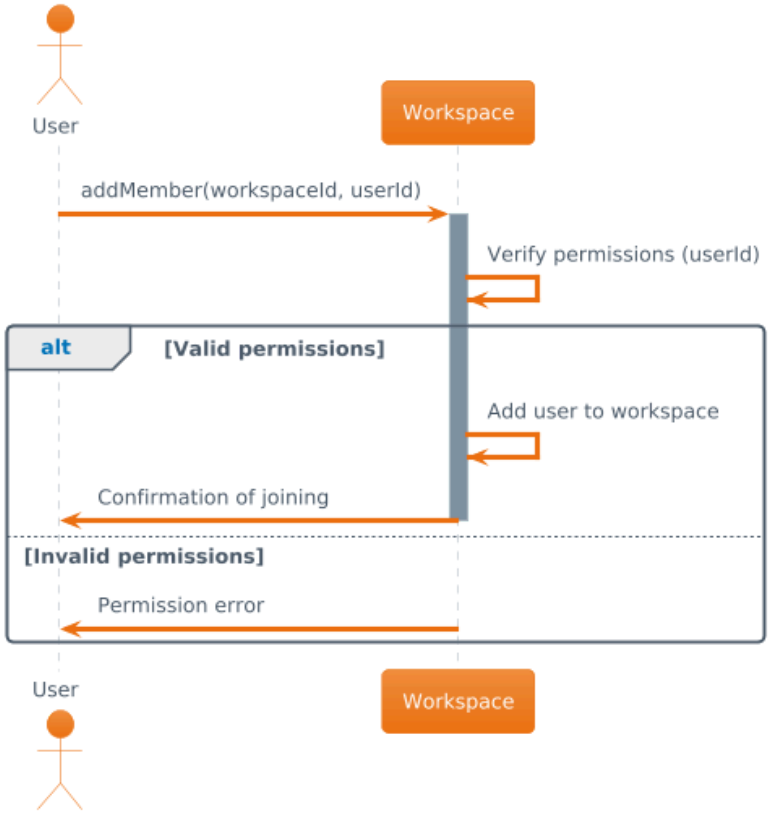
- Facilitating communication between team members.
- Providing a clear and understandable view of the system.
- Assisting in identifying requirements and designing the software.
- Serving as documentation for future reference and system maintenance.

Sequence diagrams:

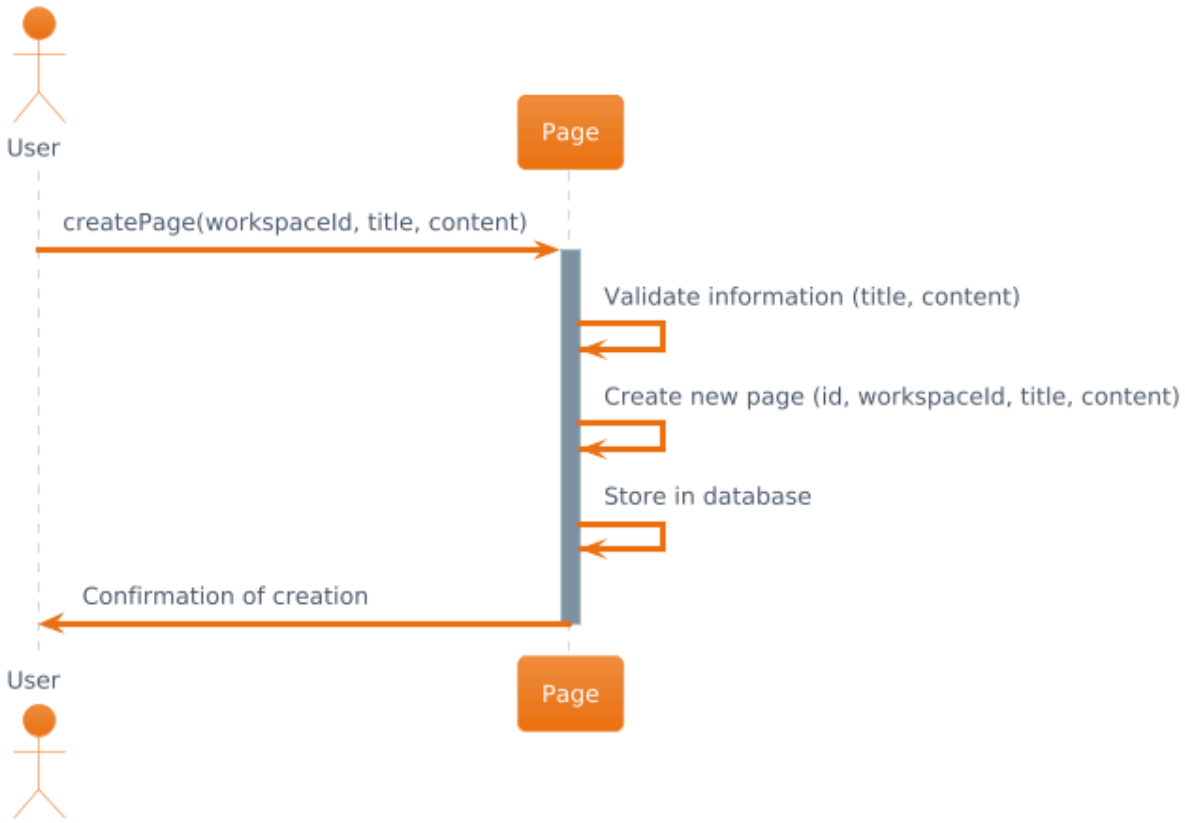
- **Register user**



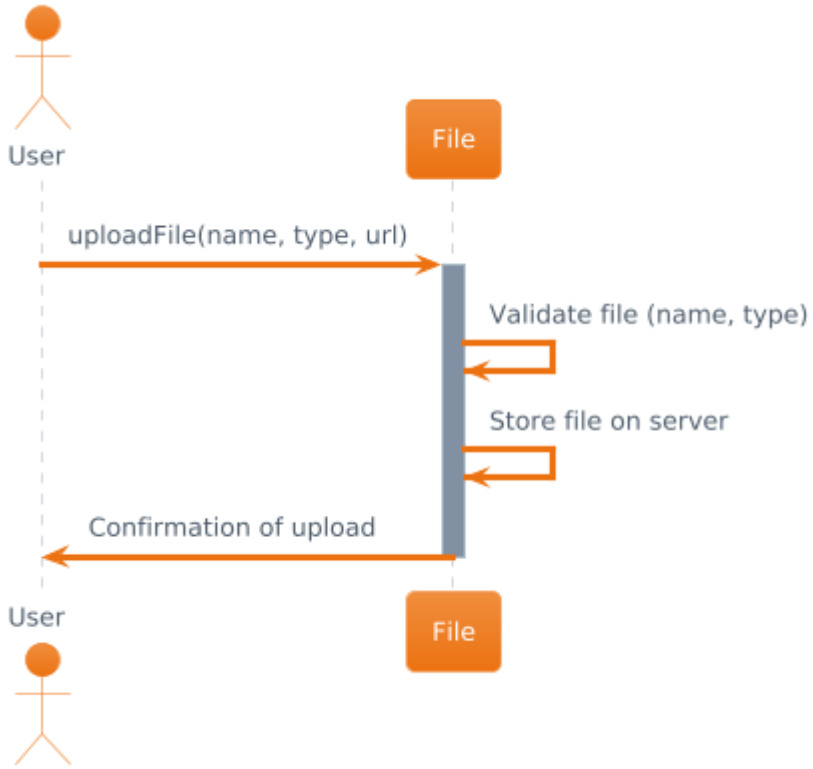
- **Join workspace**



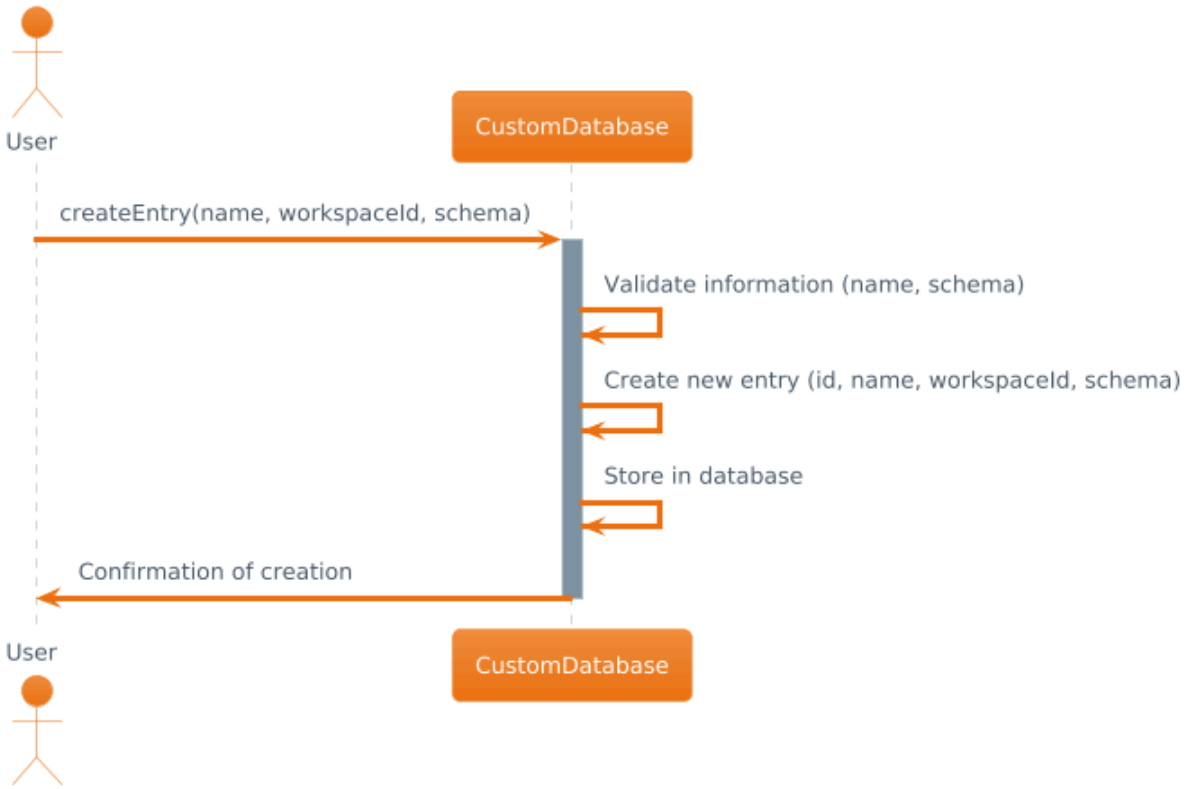
- **Create a page**



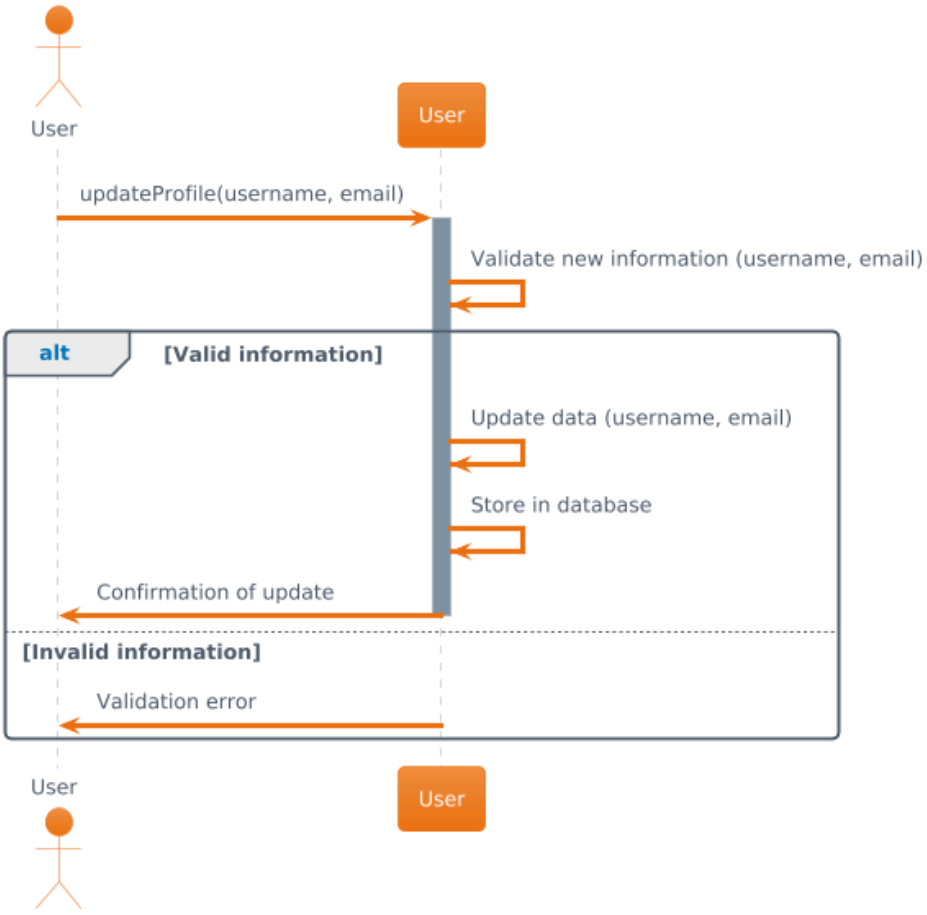
- **Upload a file**



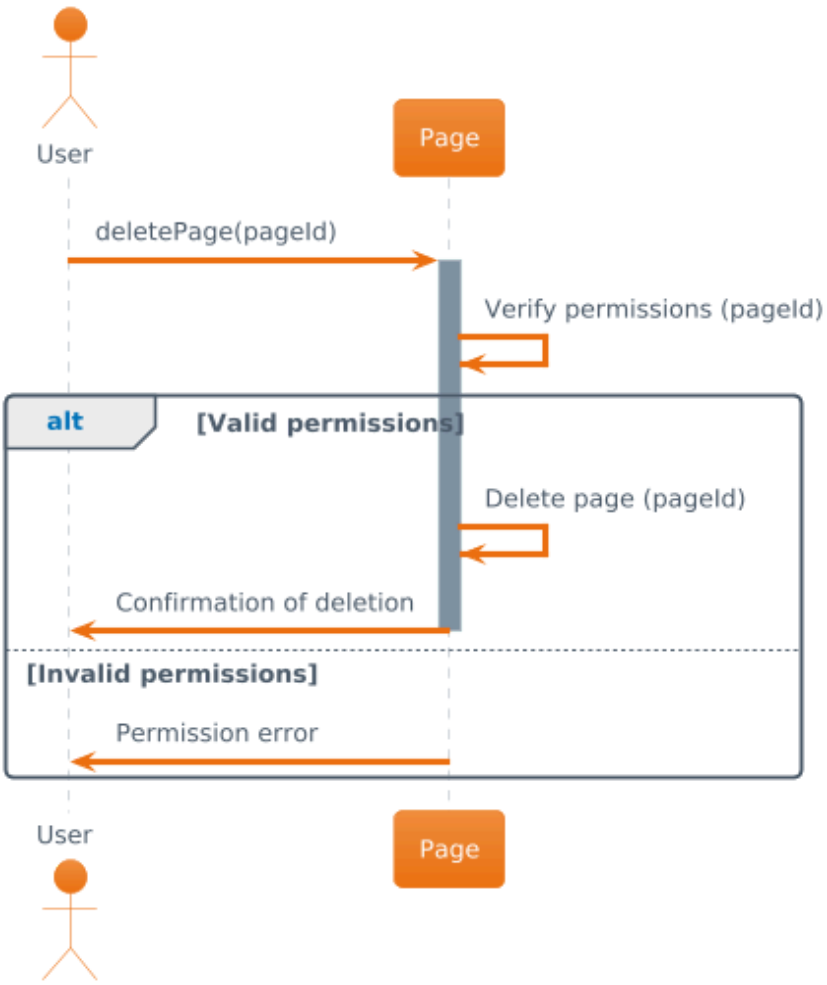
- **Create Custom Database**



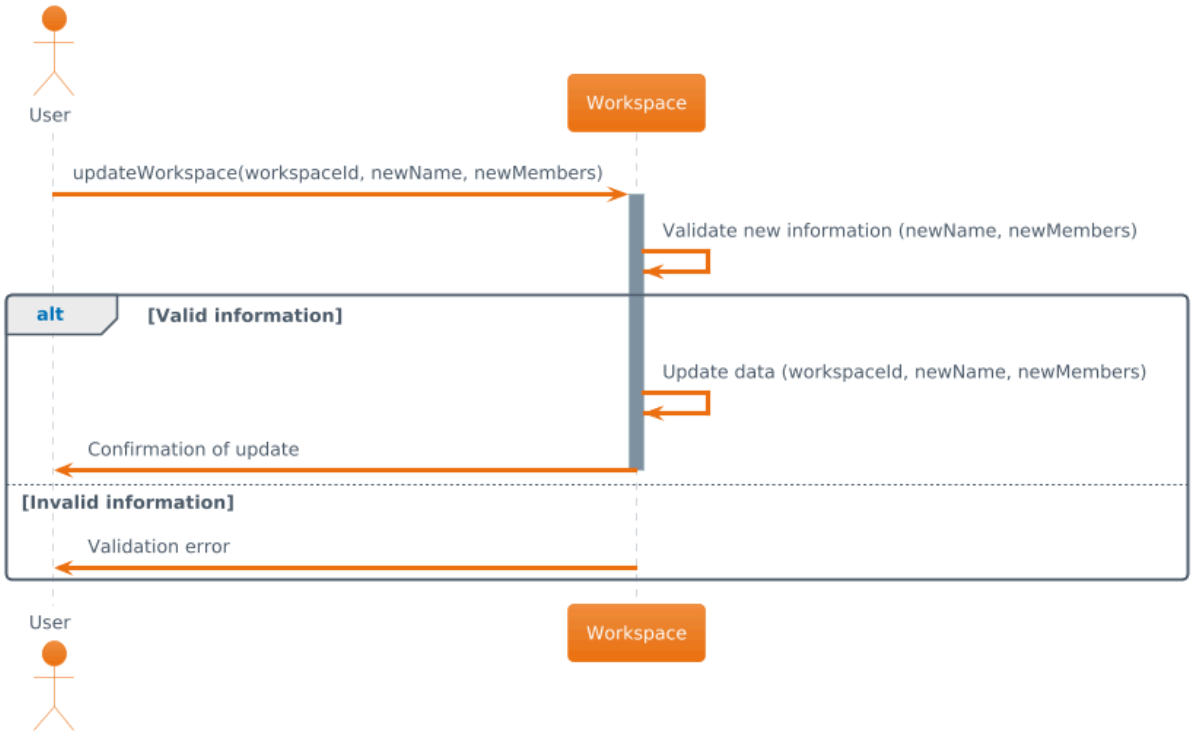
- **Update User Profile**



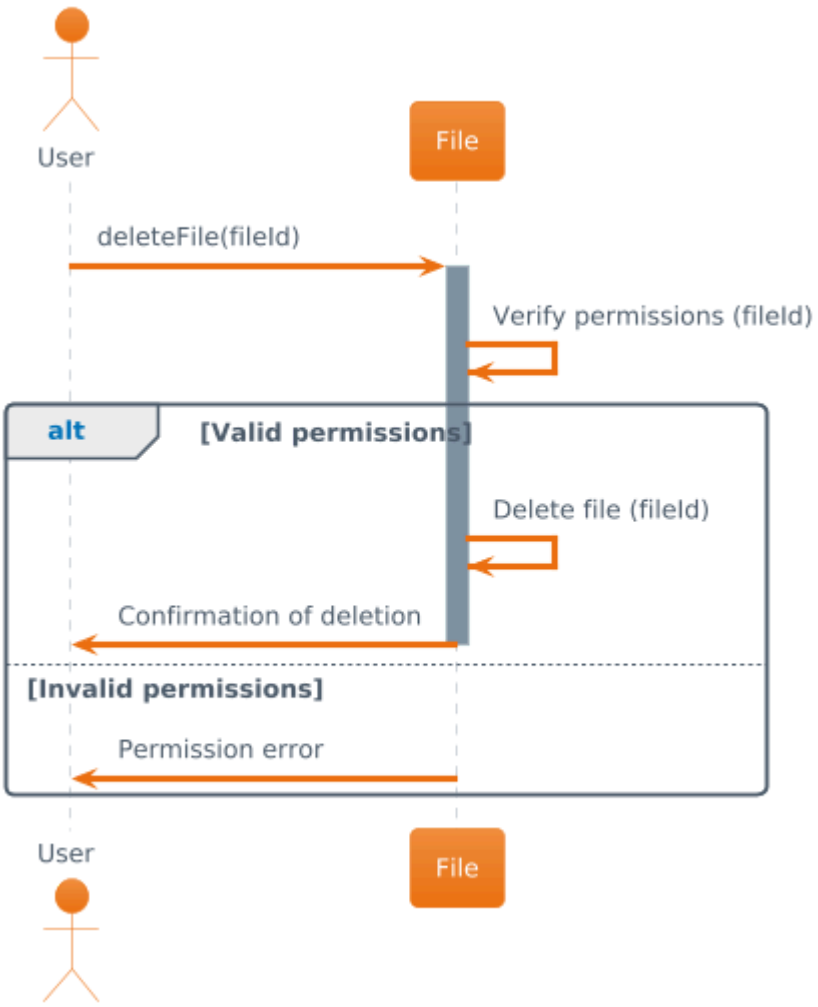
- **Delete a page**



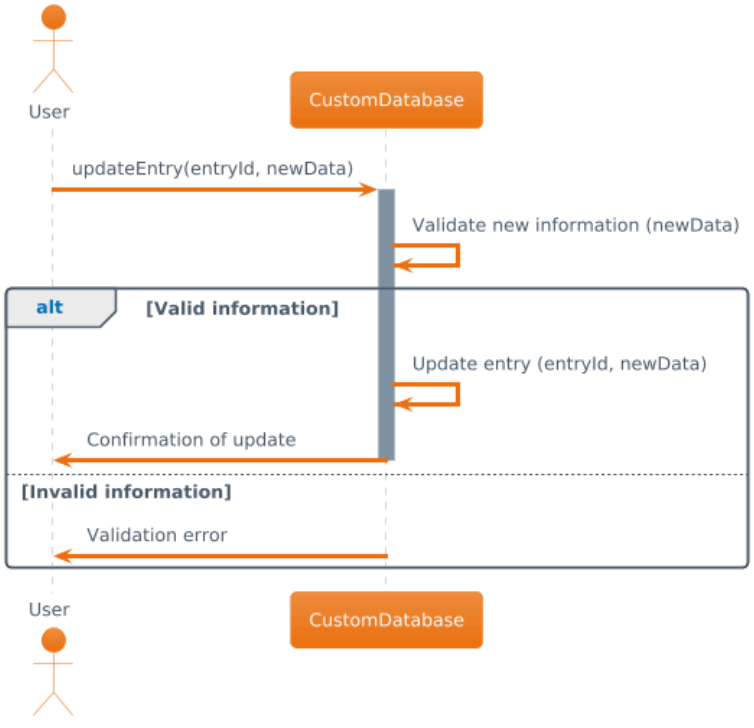
- **Update workspace**



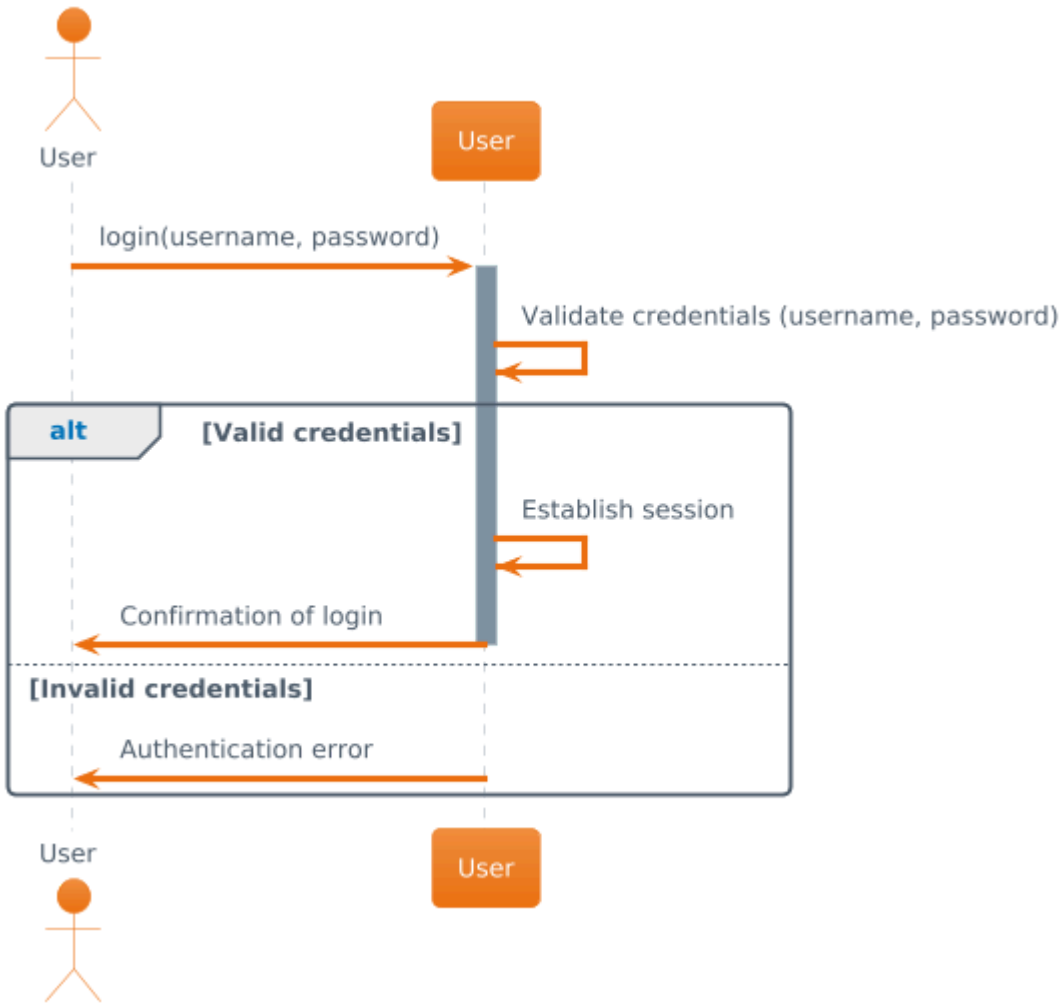
- **Delete file**



- **Updating Entry in Custom Database**



- **Login**



7. Activity Diagrams

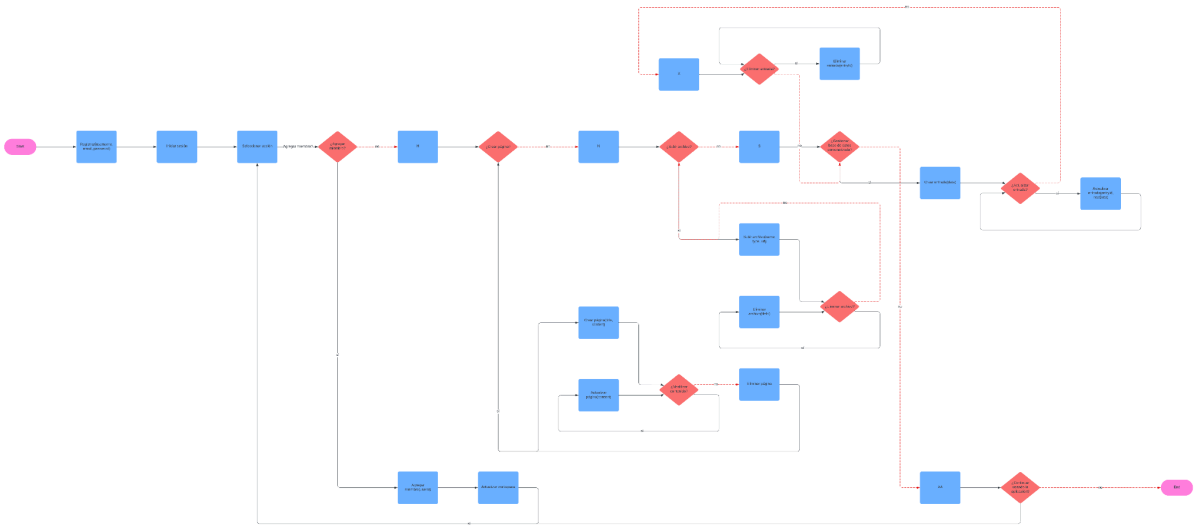


image link:

https://drive.google.com/file/d/1RDFH7iPFyf-dFbyIoEX_6wYgmTGCSHC_/view?usp=sharing

8.Deployment diagrams

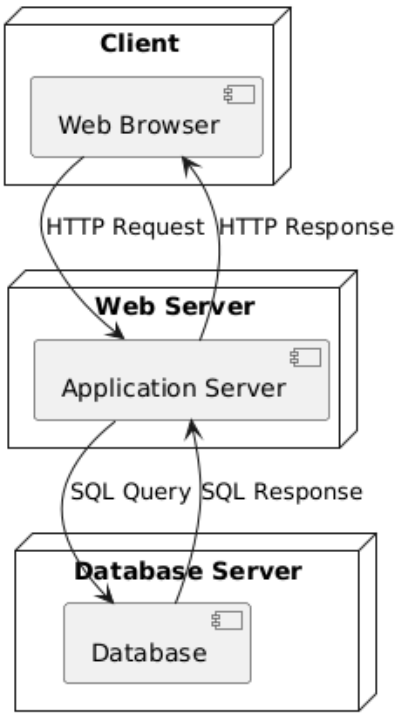
The deployment diagram is a visual representation that illustrates the physical architecture of the system, showing how software components are distributed on the hardware and how they interact with each other. This diagram is crucial to understanding how the replicated Notion application will be deployed in a real environment.

Deployment Diagram Structure

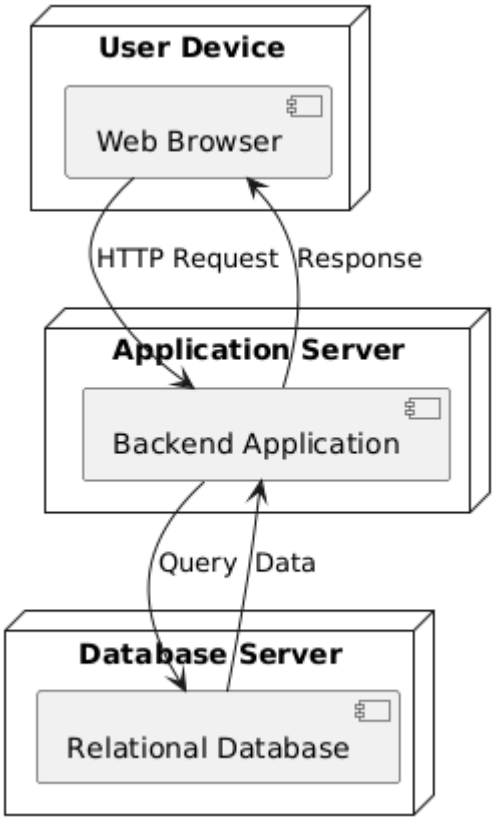
- Nodes: Represent the physical or virtual devices where the application components will run, such as servers, databases, and clients.
- Components: Indicate the parts of the software that will be deployed on each node, such as the web application, database, and any additional services.
- Connections: Show the relationships and communications between nodes, including network protocols and ports used for communication.

Below is the deployment diagram for the replicated Notion system:

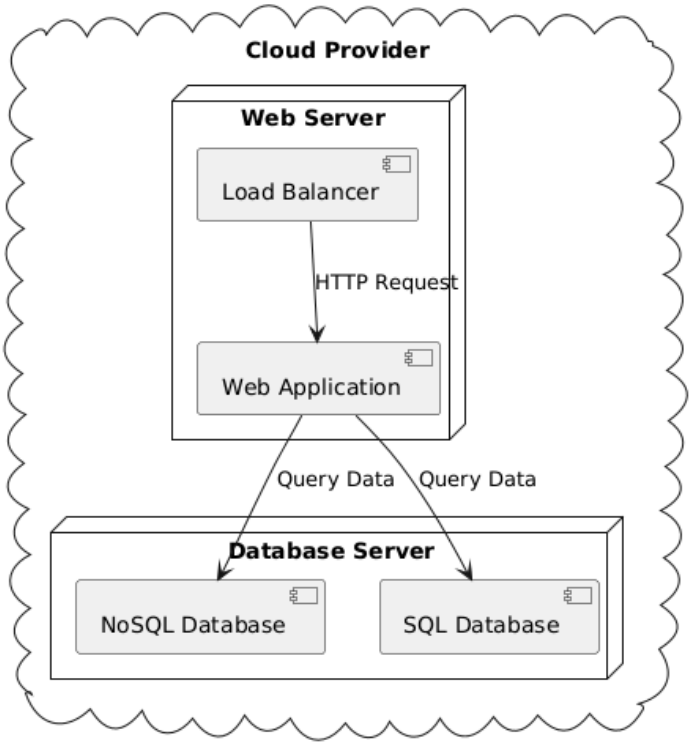
Basic Deployment Diagram
This diagram shows a client, a web server, and a database.



Deployment Diagram for Web Applications
This diagram illustrates how the components of a web application are deployed.

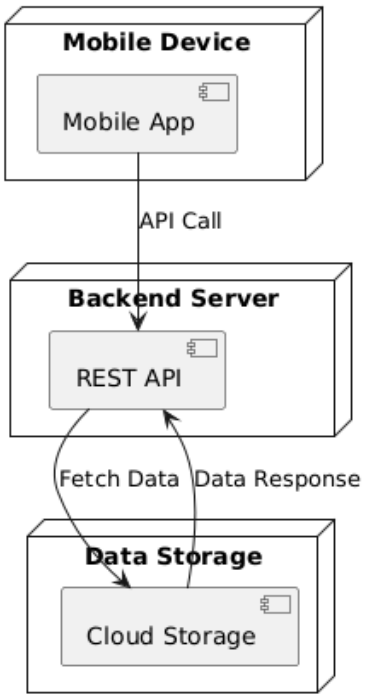


Deployment Diagram for Cloud Infrastructure
This diagram shows how resources are distributed in the cloud.



Mobile Application Deployment Diagram

This diagram illustrates the architecture behind a mobile application.



9. GitHub Repository

The GitHub repository is a fundamental part of the project, as it houses all the source code, documentation, and resources necessary for the development of the replicated Notion application. This repository not only facilitates collaboration between team members, but also allows for effective tracking of changes made to the code over time.

The link to the GitHub repository is as follows:
https://github.com/jhomarABL/project-software-modeling
Through this link, interested parties will be able to access the source code and all the documentation associated with the project.