

# **WORKSHOP No. 1 – OBJECT-ORIENTED DESIGN AND REQUIREMENTS**

Juan Sebastián Herrera Rodríguez

20242020032

Isabela Chica Becerra

20242020035

Faculty of Engineering, Francisco Jose de Caldas District University

Object-Oriented Programing

Carlos Andres Sierra

Bogotá D.C

2025

## PROJECT DEFINITION

**PinBerry** is a simplified and personalized version of Pinterest, where users can view, upload, delete and save photos. The platform aims to be simple and easy to use, allowing anyone to organize their visual content in an intuitive way.

## PROJECT OBJECTIVES

- *General objectives*
  - To create a web application that allows users to view, upload, delete and save images in an organized and accessible manner.
  
- *Specific objectives*
  - Develop a user-friendly interface that facilitates navigation and image management.
  - Implement a class and object-based structure to manage images and folders.
  - Allow users to upload and delete images according to their preferences.
  - Provide the option to save images to folders within the platform or download them.

# 1. REQUIREMENTS DOCUMENTATION

## *Functional requirements:*

- User Account Management

- The user will be able to register an account;

By providing a username, a password, and uploading a profile picture.

- The user will be able to edit their account information;

By updating their password and changing their profile picture.

- The user will be able to log in;

By entering their registered username and password to access their account.

- Photo Upload and Download

- The user will be able to upload photos;

By selecting images from their device's gallery or taking new photos with the device's camera.

- The user will be able to remove photos that they have upload before

- The user will be able to download photos;

By saving images from their fyp or folders to their device.

- Photo Organization

- The user will be able to create folders;

To organize their photos into different categories.

- The user will be able to delete photos;

Removing selected images from their folders.

- Photo Browsing

- The user will be able to scroll through their photo collection;

With the system dynamically loading more photos as the user scrolls, enabling smooth navigation through large image libraries.

### ***Non-functional requirements:***

- Usability and Design

- The system should have a clean and attractive interface;

It must be visually appealing and user-friendly.

- The system should be easy to use;

Users should understand and navigate it without difficulty.

- Reusability

- The system should use reusable components;

Common functions (like upload or validation) must be modular and shared.

- Flexibility and Maintainability

- The system should be flexible;

It must allow new features to be added with minimal changes.

- The system should be maintainable;

Code should be clean, organized, and follow good practices.

## 2. USER STORIES

<b>Title:</b> View Photos	<b>Priority:</b> High	<b>Estimate:</b> 16-20 hours
<b>User Story:</b>  As a User,  I want to view the available images on the platform,  So that in can explore interesting visual content.		
<b>Acceptance criteria:</b>  <b>Given</b> that I am on the homepage  <b>When</b> I navigate through the gallery,  <b>Then</b> I should be able to see all the available images.		

<b>Title:</b> Upload Photos	<b>Priority:</b> High	<b>Estimate:</b> 20-24 hours
<b>User Story:</b>  As a User,  I want to upload images to the platform,  So that I can organize and store them.		
<b>Acceptance criteria:</b>  <b>Given</b> that I am on the upload page,  <b>When</b> I select an image and confirm the upload,  <b>Then</b> the image should be stored on the platform and appear in my gallery.		

<b>Title:</b> Delete Photos	<b>Priority:</b> Medium	<b>Estimate:</b> 16-20 hours
<b>User Story:</b>  As a User,  I want to delete images that I have uploaded,  So that I can remove content I no longer need to.		
<b>Acceptance criteria:</b>  <b>Given</b> that I am on my gallery page,  <b>When</b> I select an image and press the delete button,  <b>Then</b> the image should be permanently removed from my gallery.		

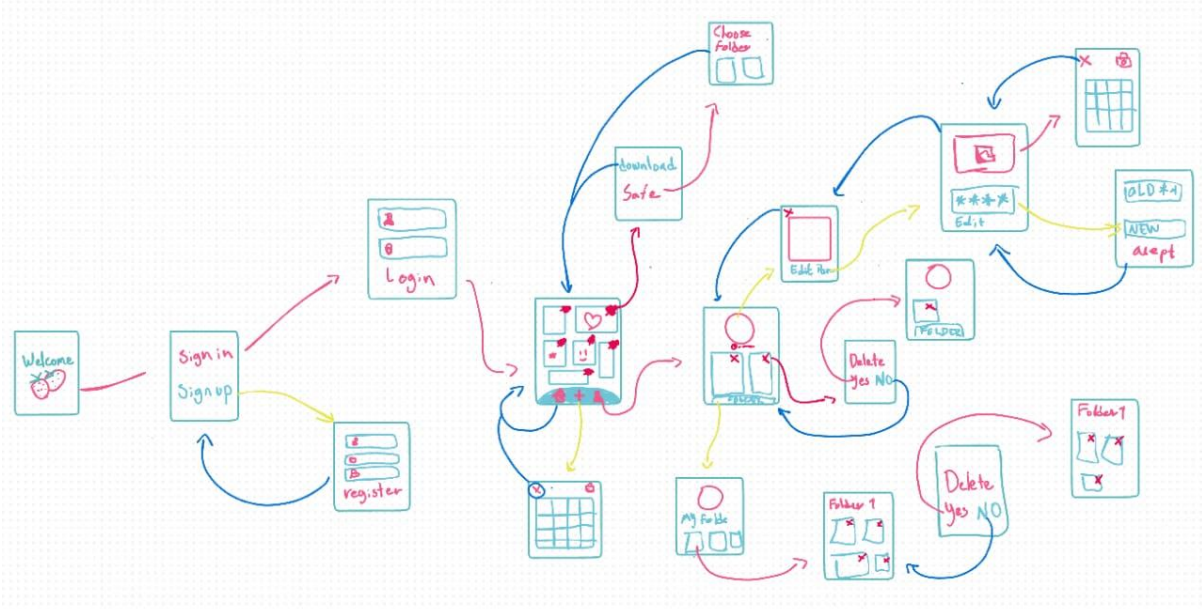
<b>Title:</b> Save Photos	<b>Priority:</b> Medium	<b>Estimate:</b> 16-20 hours
<b>User Story:</b>  As a User,  I want to save images by downloading them or adding them to folders,  <b>So that I</b> can organize my content better.		
<b>Acceptance criteria:</b>  <b>Given</b> that I am viewing an image,  <b>When</b> I choose to download or add it to a folder,		

**Then** the image should be saved according to my selection.



### 3. MOCKUPS

## Mockups drawn diagram



[https://www.canva.com/design/DAGkTfEUicw/4lp\\_0\\_doZkH3n4L4n3Y25A/edit?utm\\_content=DAGkTfEUicw&utm\\_campaign=designshare&utm\\_medium=link2&utm\\_source=sharebutton](https://www.canva.com/design/DAGkTfEUicw/4lp_0_doZkH3n4L4n3Y25A/edit?utm_content=DAGkTfEUicw&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)

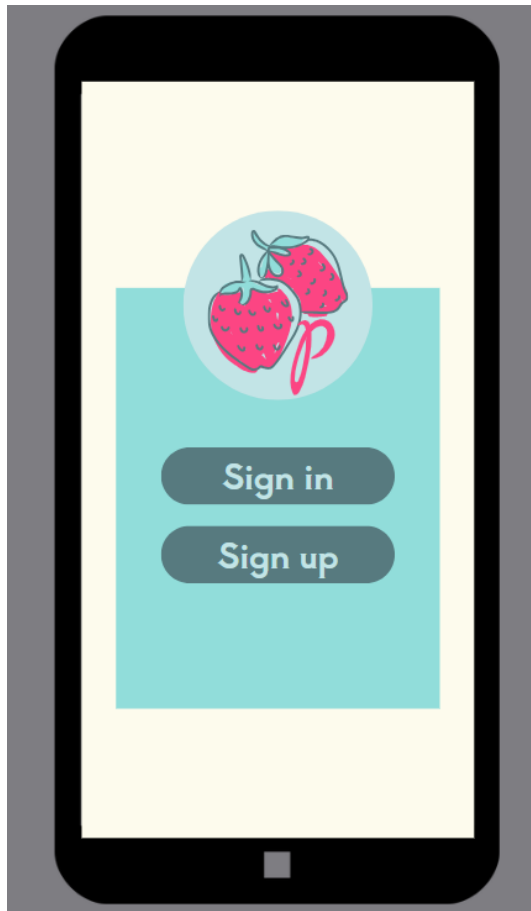
## Cell Phone Interface

- Application startup

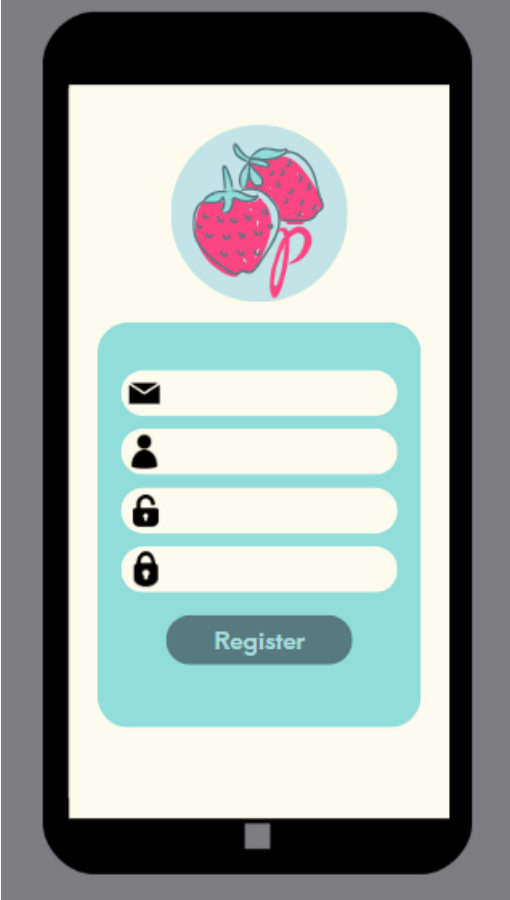
At the moment, this is only the beginning of



When the user enters the application, has the option to register or login if they already have an account

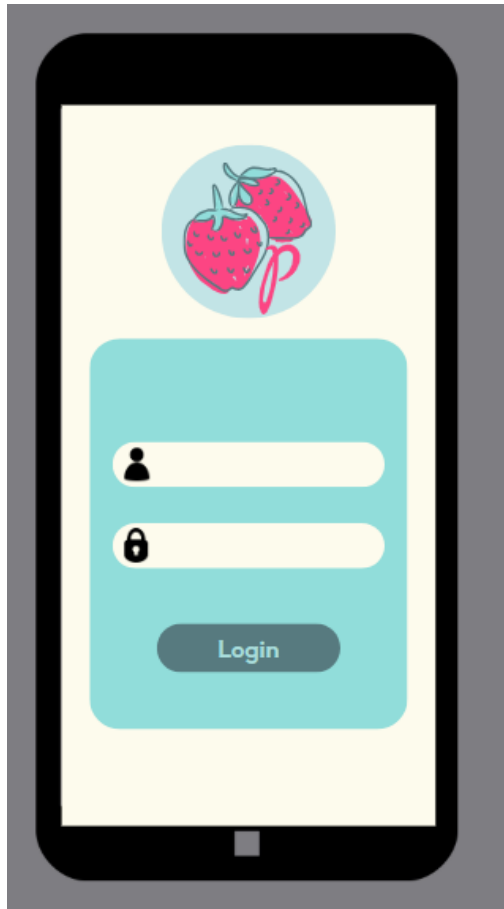


If the user decides to register, will be directed to the following interface:

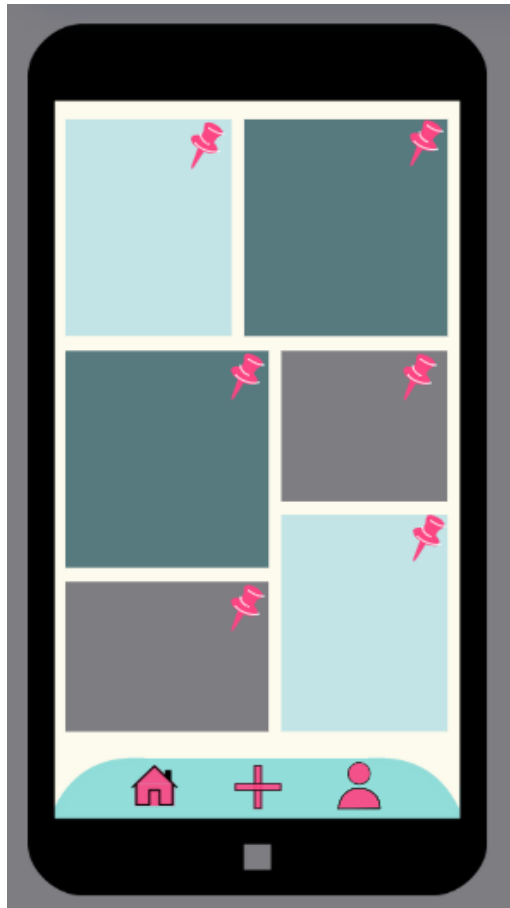


A mobile application registration screen. At the top, there is a circular logo featuring two red strawberries with green leaves. Below the logo, there is a light blue rounded rectangle containing four white input fields. The first field has an envelope icon, the second has a person icon, the third has a padlock icon, and the fourth has a padlock icon. Below the input fields is a dark blue rounded rectangle with the word "Register" in white text.

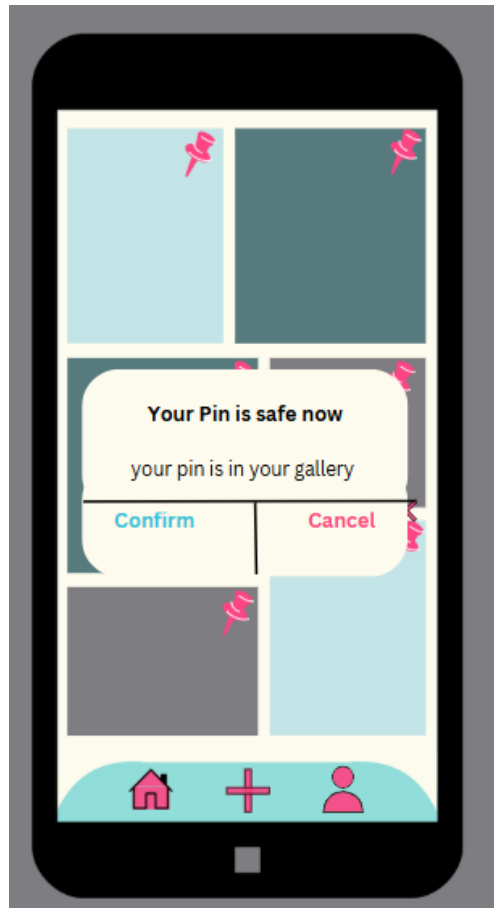
If, instead, decide to log in their account or they have already registered and are about to log in, they will be redirected to the following interface:



Once the user is logged in, they will be able to enter completely to the application, and it will be presented with the main interface.



In this main interface, the user will be able to see the various photos that have been uploaded by other users and if he/she wishes, will be able to save the photo of his/her choice, if they click the pink pin. Then it is going to appear a warning box showing that the pin has been saved.

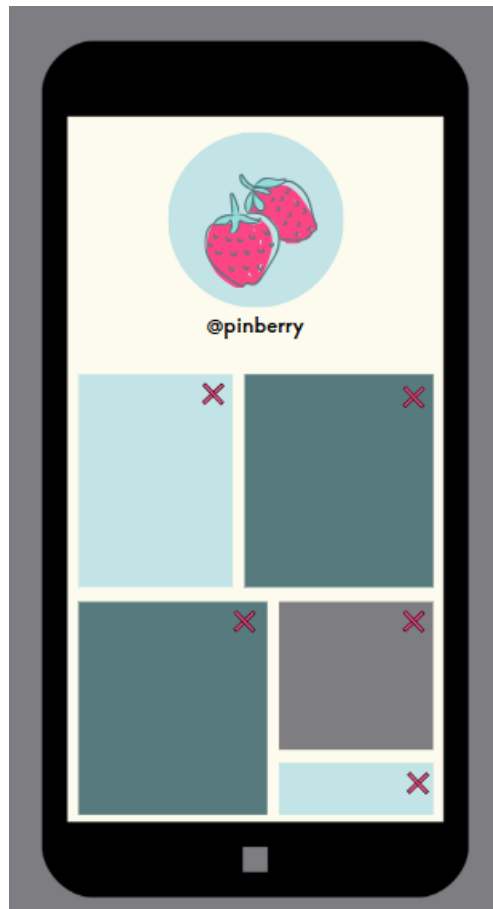


Also, apart from being able to save the photos they want, the user can also upload pictures to the application from their gallery. They just need to click on the plus in the middle of the low screen.

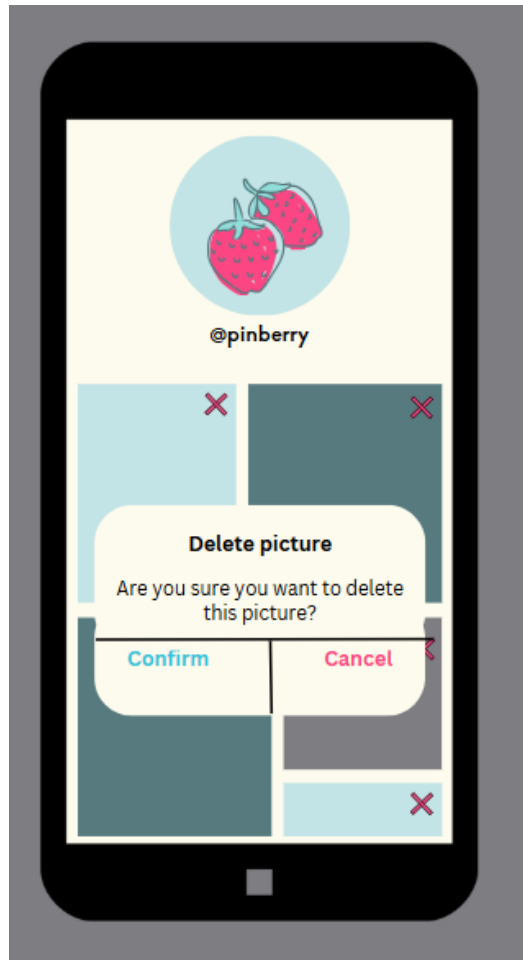




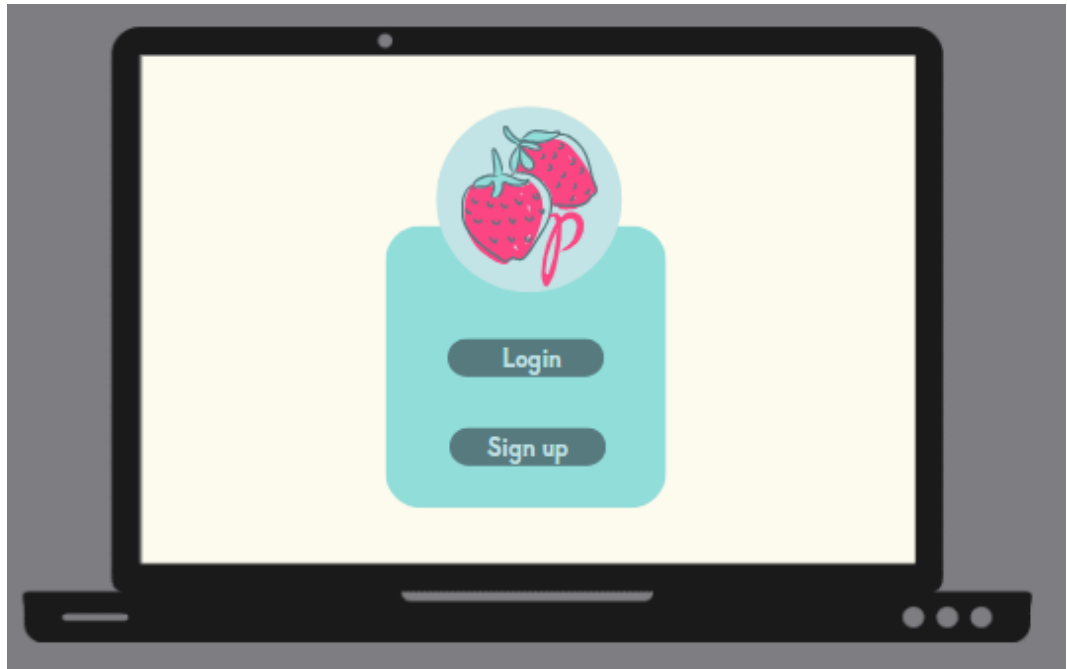
In addition, if the user clicks on the person's icon at the bottom right, they can also review their own profile to review the photos that has been uploaded



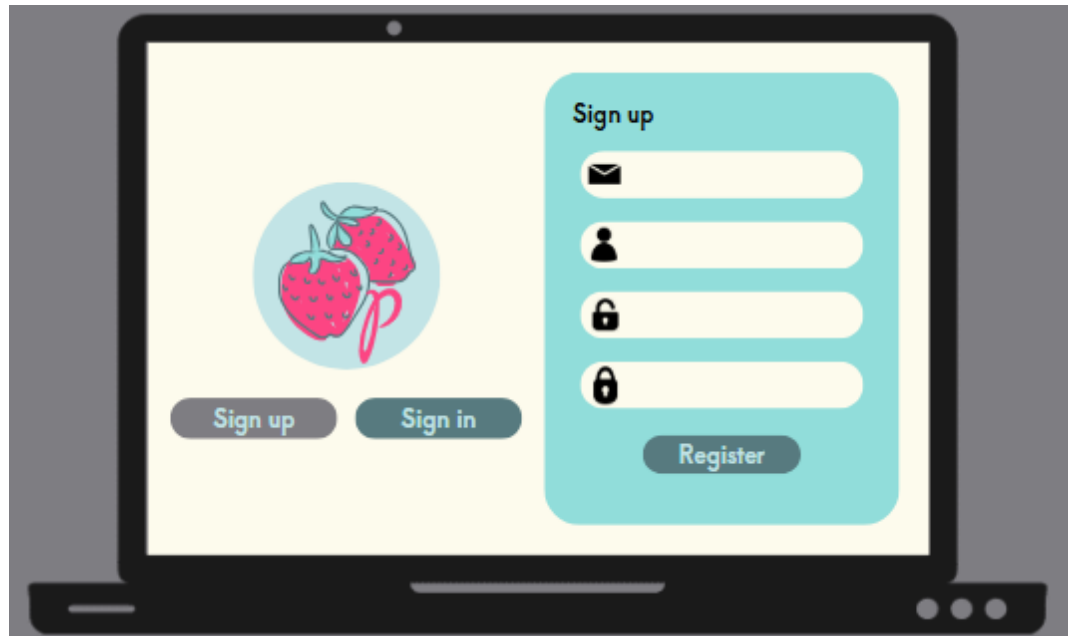
Once they are in their own profile, they also have the option to delete any of the pictures they have previously uploaded, just click on the "X" found at the top right corner of the photo that they want to remove. Once they click on the "X" a warning box will appear to confirm if they really want to delete the photo



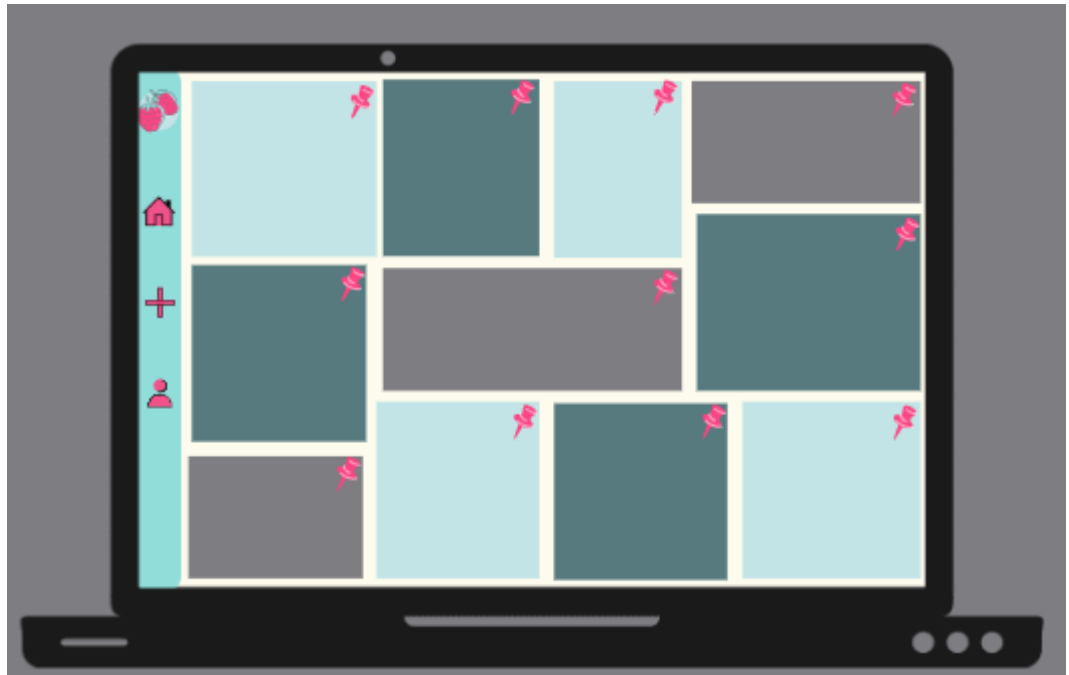
It is expected that the application can also works on computer, so here is a presentation of how we want to create the interfaces.



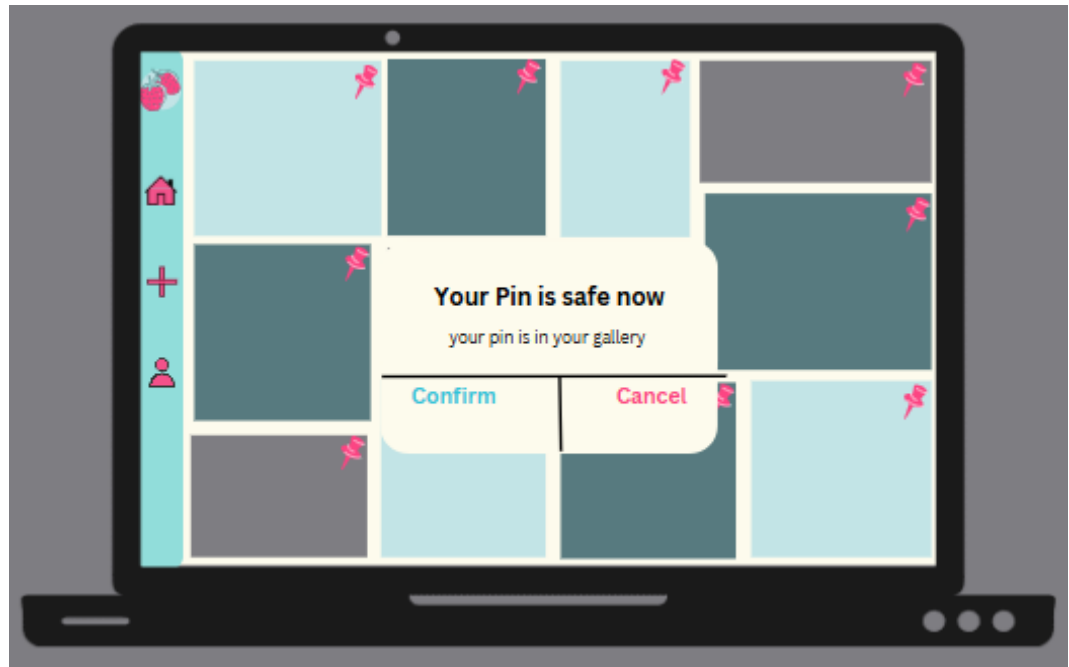
On the computer, it changes the interface, but not the process that the user must do, in this case if the click on the Sign up button, the corresponding space will open for the user to enter the necessary data



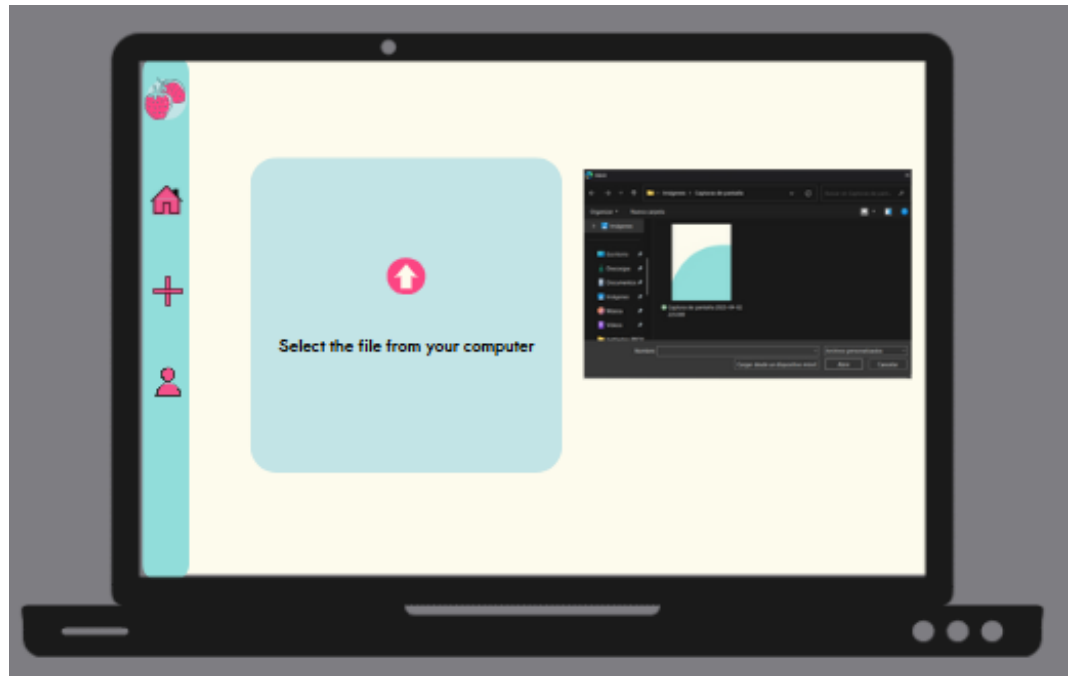
Once you have registered or logged in, you will be redirected to the main page of the application.



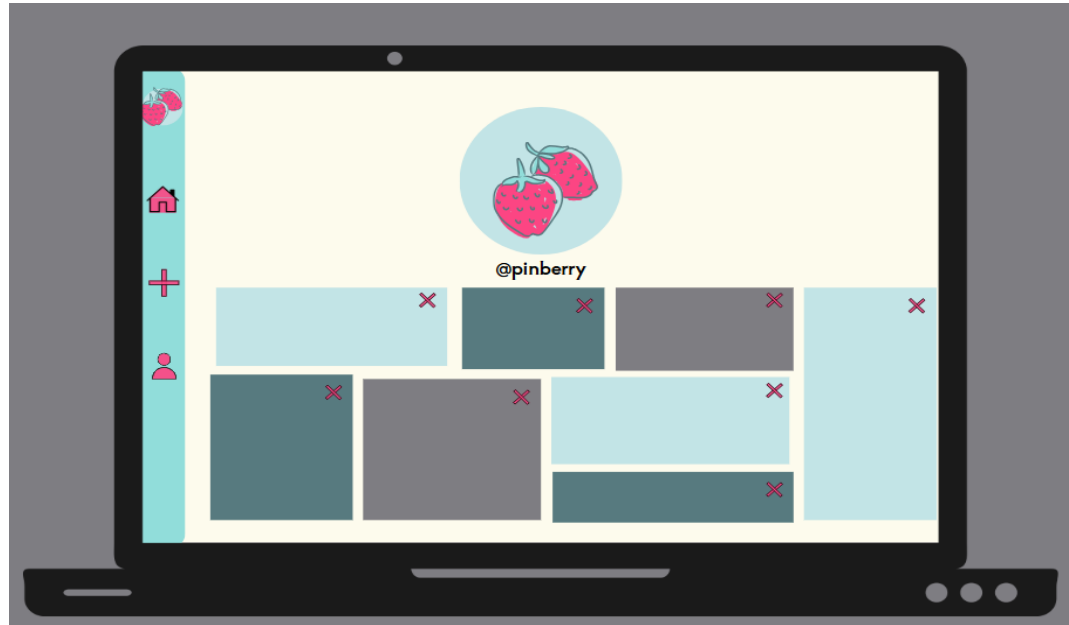
Here they can also save the photos that they like the most or that are of their preference. It is practically the same, just click on the pink pins and it will be safe in your gallery.



In addition, they can also upload photos from their computer, but in this case the file explorer will open to choose the photos they want to upload. Just like the above, the user must click on the plus, but this time it is at the left of the screen.

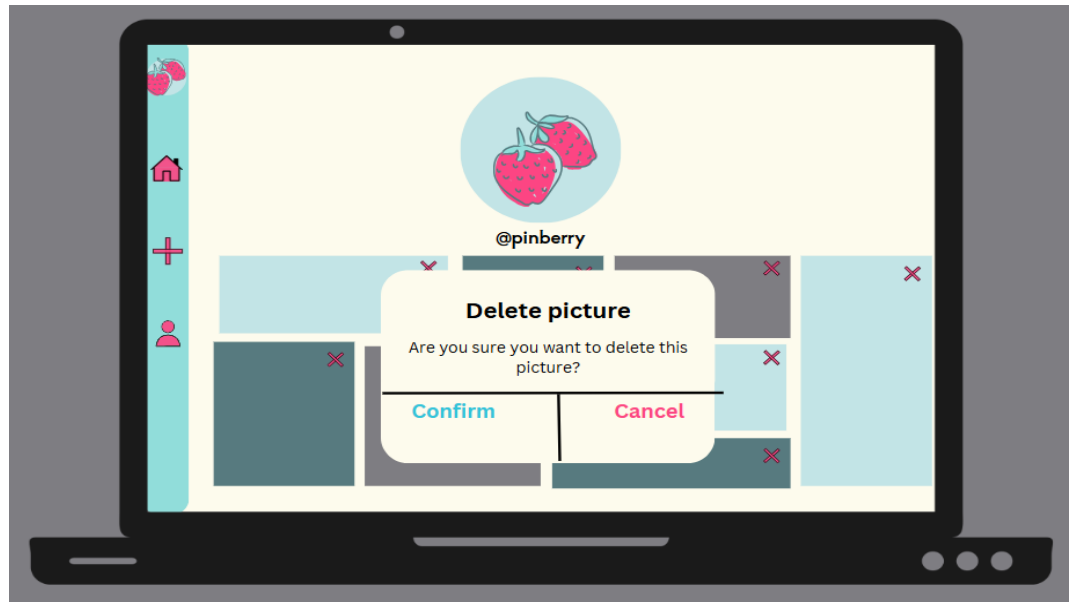


Again, the user can check their profile from their computer to view the photos they have uploaded. The user should click on the person's icon on the screen's left panel.





And finally, they can also delete the pictures they want, and they have uploaded in their profile, it is the same as in mobile, just click the "X" in the upper right corner and they will get the confirmation box to delete



#### 4. CRC Cards:

Class: User	
<b>Responsabilities:</b> <ul style="list-style-type: none"><li>– Register with username, password, and profile picture</li><li>– Log in and log out</li><li>– Edit profile (password, picture)</li></ul>	<b>Collaborator:</b> <ul style="list-style-type: none"><li>• Photo</li><li>• Folder</li><li>• AuthenticationService</li></ul>

Class: Photo	
<b>Responsabilities:</b> <ul style="list-style-type: none"> <li>– Store image data and metadata (upload date, source, etc.)</li> <li>– Handle upload (from gallery or camera)</li> <li>– Handle download</li> <li>– Delete photo</li> </ul>	<b>Collaborator:</b> <ul style="list-style-type: none"> <li>• User</li> <li>• Folder</li> <li>• StorageService</li> </ul>

Class: Folder	
<b>Responsabilities:</b> <ul style="list-style-type: none"> <li>– Create and delete folders</li> <li>– Edit the name</li> <li>– Add and remove photos</li> <li>– List photos within the folder</li> </ul>	<b>Collaborator:</b> <ul style="list-style-type: none"> <li>• Photo</li> <li>• User</li> </ul>

Class: Gallery
----------------

<b>Responsabilities:</b> <ul style="list-style-type: none"> <li>– Display photos with scrolling</li> <li>– Load more photos dynamically</li> </ul>		<b>Collaborator:</b> <ul style="list-style-type: none"> <li>• Photo</li> <li>• User</li> </ul>
--	--	--

<b>Class: AuthenticationService</b>		
<b>Responsabilities:</b> <ul style="list-style-type: none"> <li>– Verify login credentials</li> <li>– Manage session tokens</li> </ul>		<b>Collaborator:</b> <ul style="list-style-type: none"> <li>• User</li> <li>• DataBase</li> </ul>

<b>Class: StorageService</b>		
<b>Responsabilities:</b> <ul style="list-style-type: none"> <li>– Save and retrieve photos</li> <li>– Manage file system or cloud storage</li> </ul>		<b>Collaborator:</b> <ul style="list-style-type: none"> <li>• Photo</li> <li>• Folder</li> </ul>

## 1. ACTUALIZATIONS

*Changes from the 2 second workshop:*

- We add to the "folder" class a new attribute (Name)
- We add to the class "folder" a new method (be able to modify the name)

## 2. Technical Design (UML Diagrams)

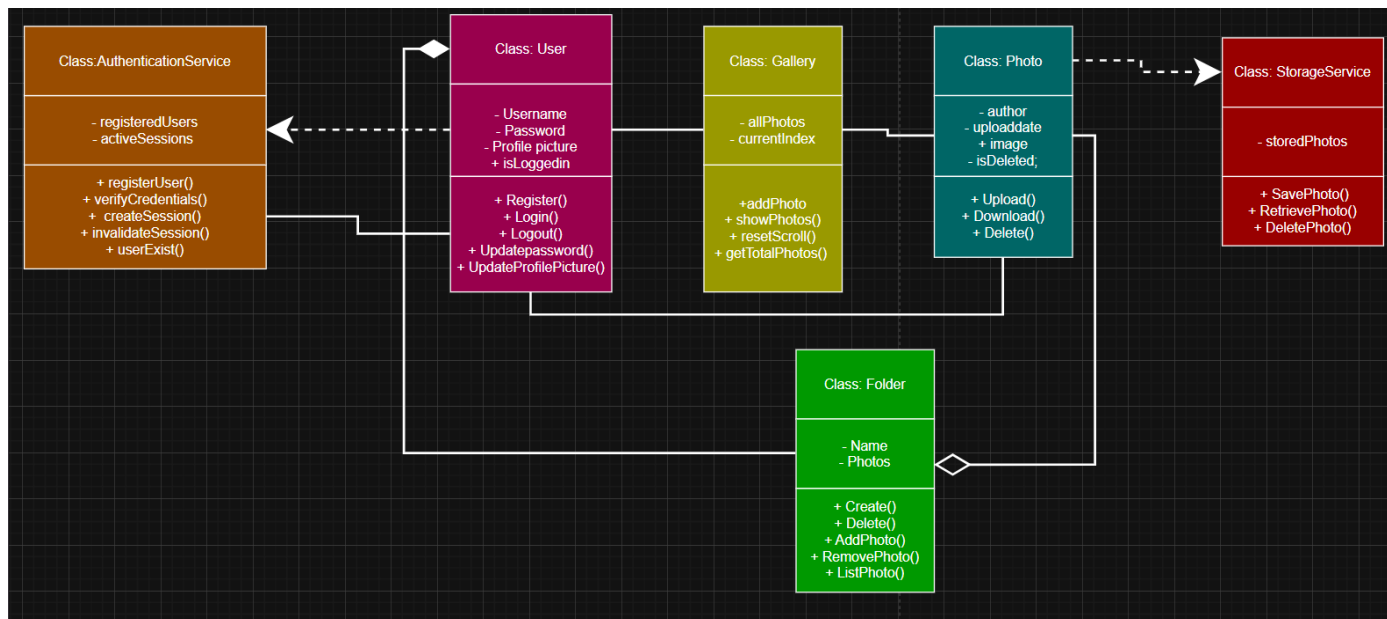
In the context of the PinBerry application, applying inheritance is neither necessary nor beneficial given the system's current structure and intended functionality. Inheritance is best suited for scenarios involving a hierarchical relationship between classes where multiple subclasses can meaningfully share and override behavior from a common superclass.

However, in PinBerry, each class (**User**, **Photo**, **Folder**, **Gallery**, **AuthenticationService**, and **StorageService**) encapsulates a distinct set of responsibilities and behaviors that do not overlap in a way that would justify abstraction through inheritance.

There are no polymorphic behaviors required, no current or projected subclasses (e.g., no different types of users, media items, or folder types), and no shared logic between services that would benefit from inheritance. Introducing base classes such as **MediaItem**, **Container**, or **BaseService** would create unnecessary architectural overhead, violating the principle of **KISS** (Keep It Simple, Stupid) and reducing the system's clarity and maintainability.

Instead, using **composition over inheritance** ensures that each class remains focused and independently testable, with explicit relationships (e.g., a **User** owns **Photos**, **Folders**; **Folder** contains **Photos**) that reflect the domain model clearly. This decision aligns with best practices in object-oriented design, which recommend favoring inheritance only when it clearly improves code reuse, extensibility, and reduces duplication — none of which apply in the current design. Therefore, avoiding inheritance is not only justified but optimal for the simplicity, maintainability, and clarity of the PinBerry system.

## 2.1 Class Diagram



## 2.2 methods implementation

### 2.2.1 User class:

Responsibilities: register, login, edit profile.

Implemented methods:

- `register(username, password, profilePicture)`
- `login(username, password)`
- `editProfile(newPassword, newPicture)`
- `logout()`
- `Updatepassword()`
- `UpdateProfilePicture()`

OOP application:

- Encapsulates credentials and profile picture as private attributes.
- Could delegate authentication to AuthenticationService (composition).
- Complies with SRP (Single Responsibility Principle): only handles user data.

### 2.2.2 Photo Class

Responsibilities: upload, download, delete photo.

Implemented methods:

- upload(source)
- download()
- delete()

OOP implementation:

- Contains private attributes: imageData, uploadDate, metadata.
- Composition with StorageService: Photo delegates file operations.
- Could inherit from an abstract Media class (if videos or gifs are added later).

### **2.2.3 Folder class**

Responsibilities: create folder, add/remove photos, list content.

Implemented methods:

- create(name)
- delete()
- addPhoto(photo)
- removePhoto(photo)
- listPhotos()

OOP implementation:

- Has an internal Photo list as a collection.
- Composition: contains and manipulates photos, but does not implement them.
- Complies with aggregation principle: Folder does not physically “own” the photo (that is done by StorageService).

### **2.2.4 Gallery class**

Responsibilities: display photos with scroll.

Implemented methods:

- loadPhotos(user)
- scrollDown()

OOP application:

- May contain Photo and User as dependencies.
- Coupling design principle: only communicates with Photo to get data.

### **2.2.5 AuthenticationService class**

Responsibilities: validate login, manage sessions.

Implemented methods:

- `verifyCredentials(username, password)`
- `CreateSession()`
- `InvalidateSession()`
- `logout(token)`

OOP implementation:

- Encapsulated logic, separate from User model (decoupling).
- Composition: handles communication with the database.

### 2.2.6 StorageService class

Responsibilities: save, retrieve and delete files.

Implemented methods:

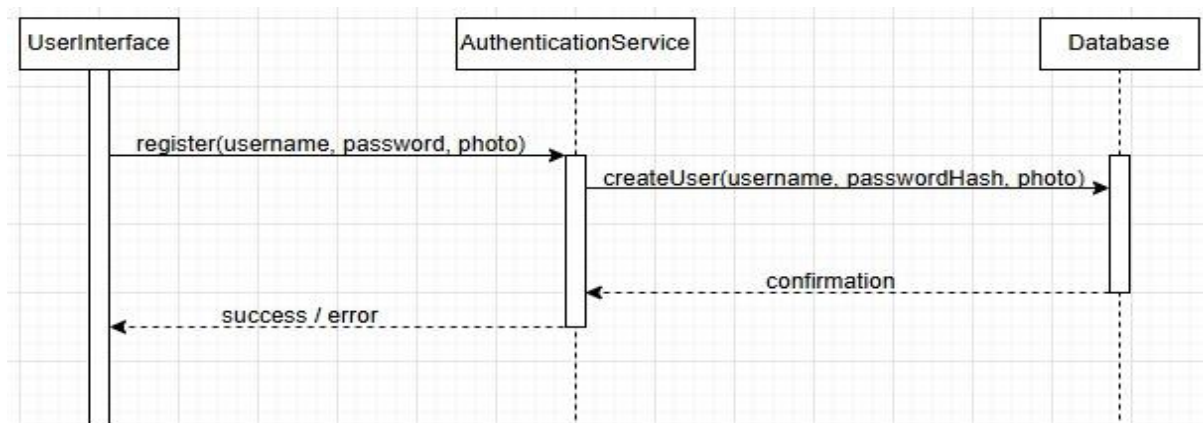
- `SavePhoto(photo)`
- `fetch(photoId)`
- `delete(photoId)`

OOP implementation:

- Complies with the dependency inversion principle (other classes depend on its interface, not its implementation).
- Fully reusable if someday moving from local storage to cloud storage.

## 2.3 Sequence Diagram

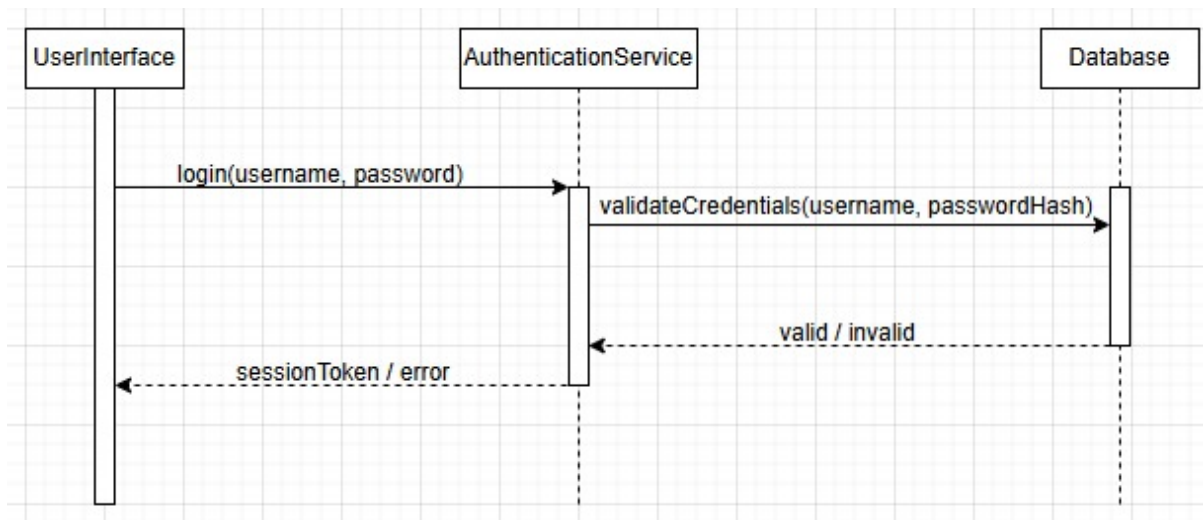
### 2.3.1 Register User



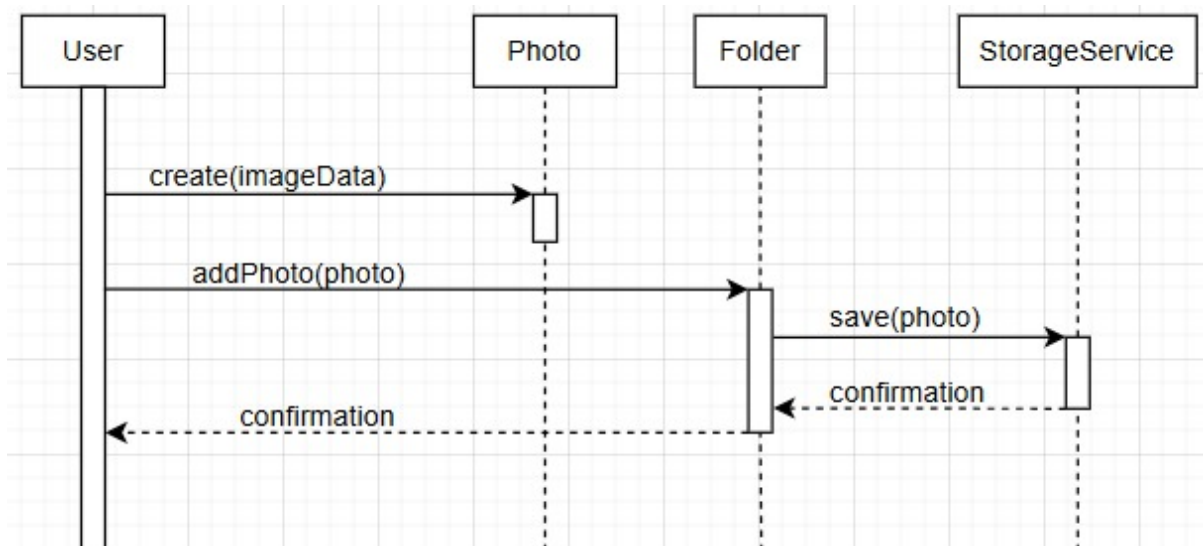




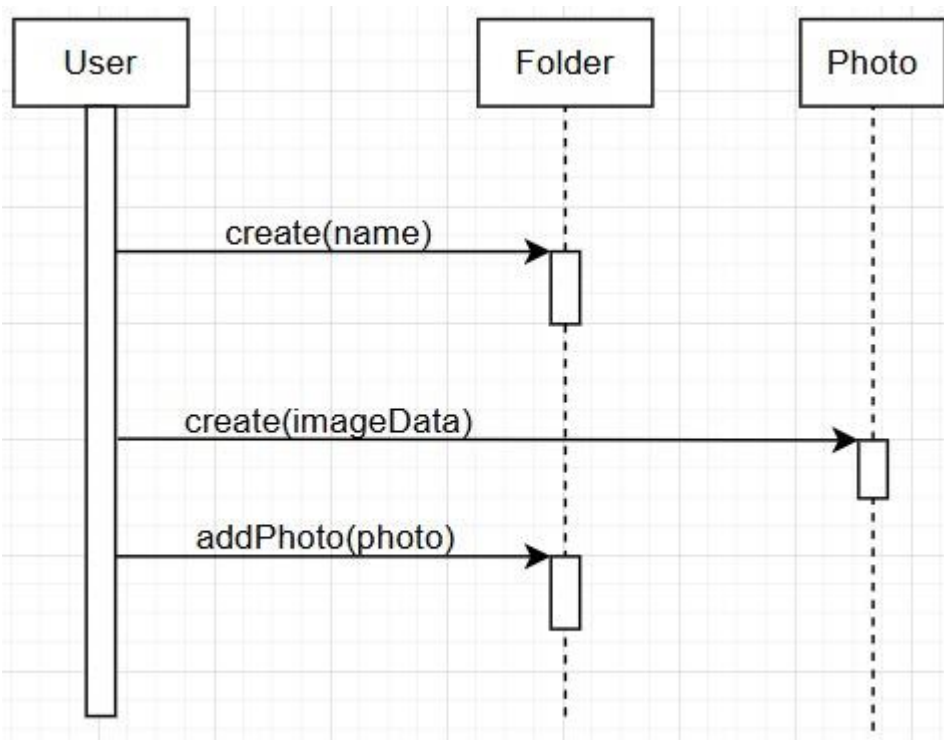
### 2.3.2 Log in



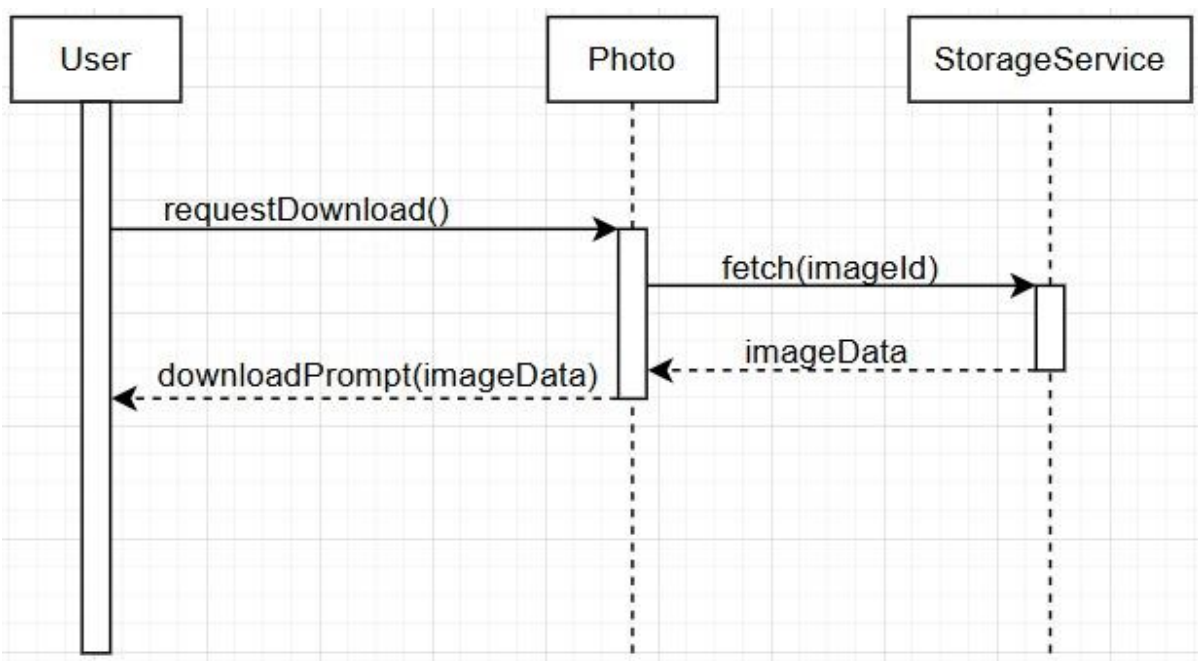
### 2.3.3 Upload photo



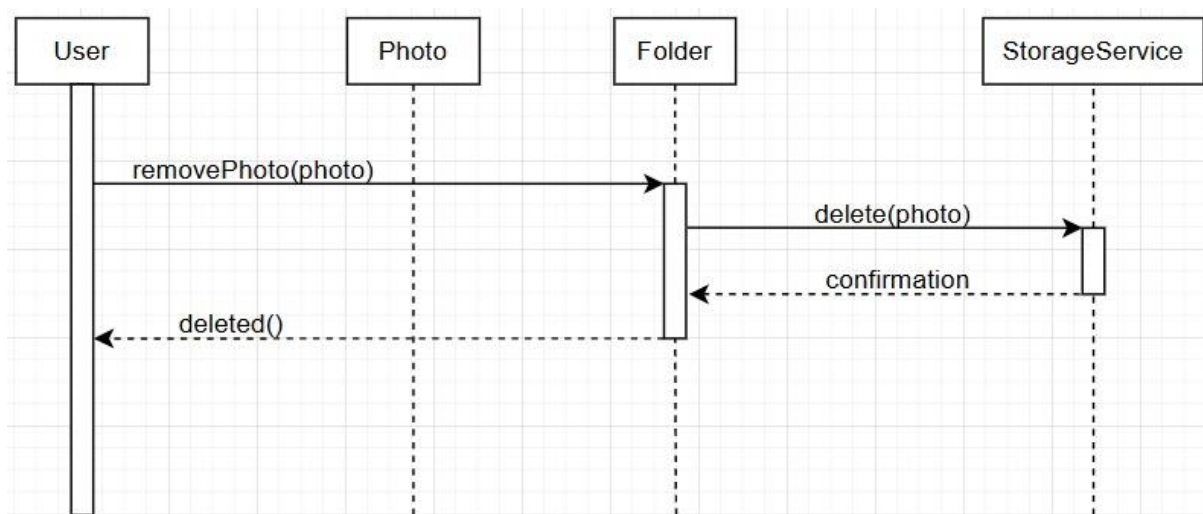
### 2.3.4 Create Folder and Add Photo



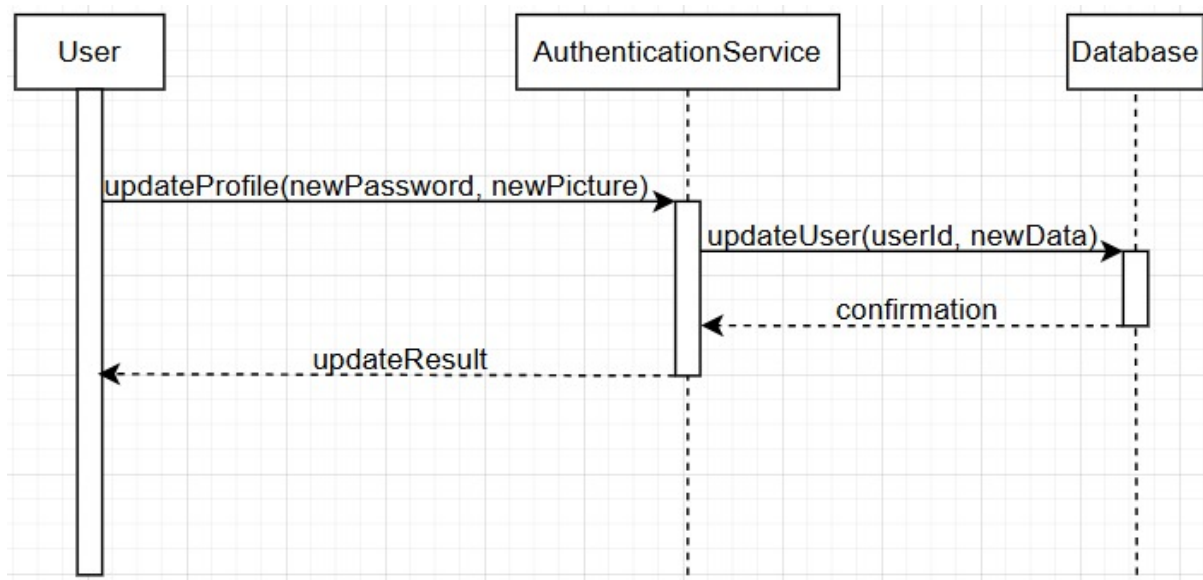
### 2.3.5 Download Photo



### 2.3.6 Delete Photo



### 2.3.7 Edit Profile



### 3. Implementation Plan for OOP Concepts

#### 3.1

In this application, encapsulation is applied rigorously to protect both sensitive data and the internal logic of each class. We are not simply declaring attributes as private and exposing them with standard getters and setters, because that kind of superficial encapsulation does not truly safeguard the system. Instead, our goal is to ensure that the only way to access or modify internal state is through carefully controlled methods, which enforce specific business rules—such as verifying user authentication, checking token validity, or ensuring data ownership.

For instance, the `User` class will contain fields like `username`, `password`, and `profilePicture`, but these attributes will not be exposed with public accessors like `getPassword()` or `setPassword()`. Instead, methods such as `updatePassword(String oldPassword, String newPassword)` will handle password changes internally, verifying the old password before updating. This ensures that no external component can directly retrieve or manipulate sensitive user information without proper authorization.

Similarly, the `Photo` class encapsulates its `imageData` and `metadata`, and these are never accessible directly. If another object wants to download a photo, it must call a method like `downloadPhoto(User user)` which first checks whether the requesting user has the right permissions. This approach forces all interactions with the internal state to go through secure, validated logic, rather than raw access to attributes.

Even system-level services like `AuthenticationService` and `StorageService`, which deal with session tokens or file storage, are encapsulated. Their internal state—such as a `Map<User, Token>` for sessions or file references for storage—is private and never exposed. If another class needs to verify a token, it must call something like `isTokenValid(User user,`

String token) instead of accessing the token store directly. This protects the system from unexpected changes and ensures consistency.

In summary, this system's encapsulation serves two essential purposes:

- To protect internal and sensitive data from unauthorized access, and
- To enforce that all interactions go through safe, validated, and rule-driven logic.

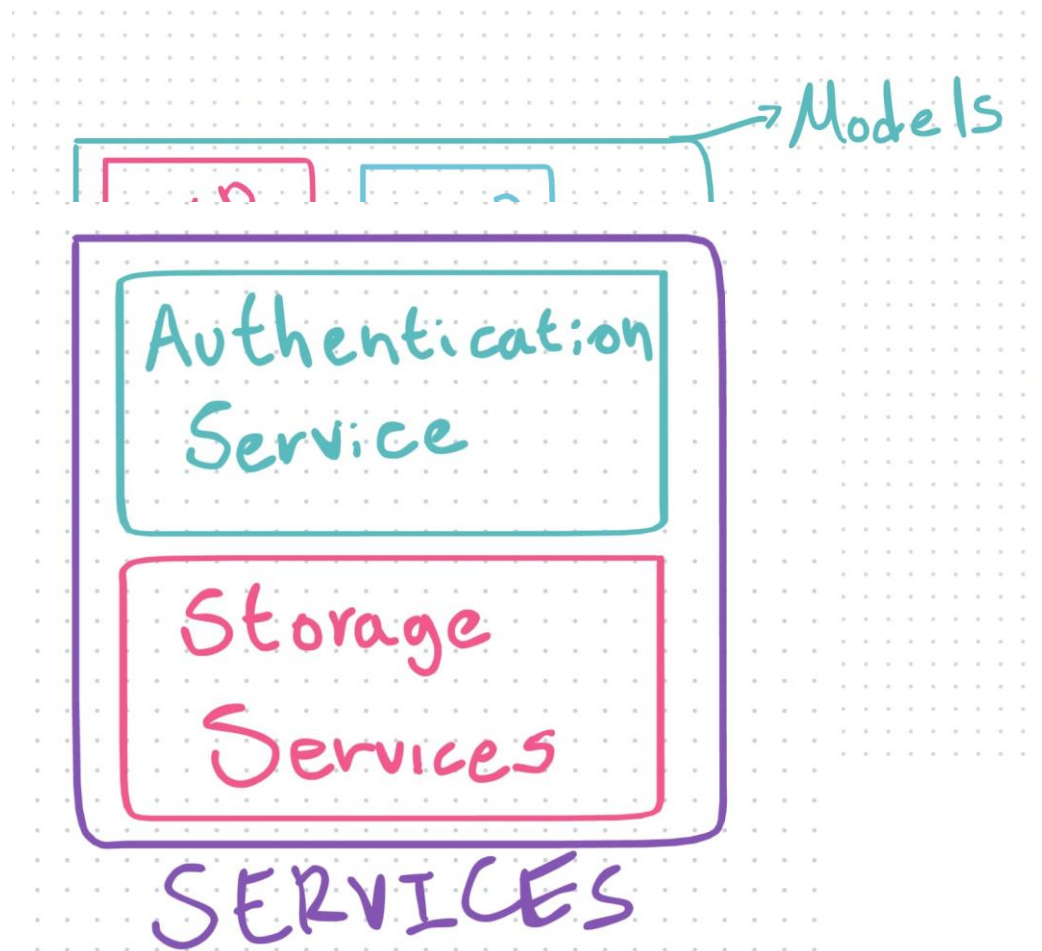
This strong, business-driven encapsulation makes the system secure, reliable, and easy to maintain. It avoids the common mistake of exposing internal state through public accessors, and instead promotes a design where objects only interact in ways that the developer explicitly allows.

On the other Hand, polymorphism is not currently applied within the domain classes (e.g., User, Photo, Folder, Gallery) because each class has distinct and non-overlapping responsibilities. There is no shared behavior that would justify abstracting a common interface or superclass among them. Introducing polymorphism in this context would lead to unnecessary complexity and violate the principle of simplicity.

However, the design remains open to polymorphism where it is meaningful—for example, in the StorageService, where different implementations (such as local or cloud-based storage) could follow a common interface like IStorageStrategy. Similarly, if the gallery supported multiple rendering modes (e.g., grid, list, collage), polymorphism could be used to switch between display strategies. These would be valid use cases of polymorphism via composition or interfaces, ensuring code flexibility without compromising clarity or cohesion.

Overall, the decision to not apply polymorphism in the current version is intentional and justified, based on the specific and self-contained nature of the classes. It avoids abstraction for its own sake while preserving extensibility if needed in future iterations.

### 3.2



Database



## 4. Work in Progress Code

### 4.2

#### 4.2.1 User class

Class name: User

Private attributes:

- username: String
- password: String
- profilepicture : String

Public attributes:

- isLoggedIn : Boolean

Public builder

- User (username: String, password: String, profilepicture: String)

Public methods:

- login (inputPassword: String): Boolean
  - Validates the password and changes isLoggedIn to true if it is correct.
- logout (): void
  - Logs the user out (isLoggedIn = false).
- setupdatePassword(currentPassword: String, newPassword: String) : Boolean
  - Changes the password if the user provides the current password correctly.
- setProfilePicture(newPicture: String) : boolean
  - Updates the profile picture if the string is not empty.
- getUsername() : String
  - Returns the username.
- isLoggedIn() : boolean
  - Returns the user's logged in status.

Relationships with other classes:

- Use AuthenticationService (For Authentication)
- Has → Folder
- Has → Photo

#### 4.2.2 Photo class

Class name: Photo

Private attributes:

- author : String
- uploaddate : String
- image : String
- isDeleted : boolean

Public builder:

- Photo(author: String, uploaddate: String, image: String)

Public Methods:

- upload() : void
  - Simulates the upload of an image. Only prints message (does not save the file).
- delete() : boolean
  - Marks the picture as deleted if it was not already deleted. Returns true if deleted, false if already deleted.
- download() : boolean
  - Allows downloading if the photo has not been deleted. Returns true if successful, false otherwise.
- getUploaddate() : String
  - Returns the date the photo was uploaded.
- getAuthor() : String
  - Returns the name of the author of the photo.
- getImage() : String
  - Returns the content or reference of the image.
- isActive() : boolean
  - Returns true if the photo has not been deleted (based on isDeleted).

Relationships with other classes:

- Has a relationship to User → because the author is a user (represented by the author attribute).
- It may be associated with Folder → if it is organized into folders.
- Uses StorageService → although it is not in the actual code, it is assumed that it would delegate the actual storage there.

#### **4.2.3 Folder class:**

Class name: Folder

Private attributes:

- name : String
- photos : ArrayList<Photo>

Public builder:

- Folder(name: String)

Public Methods:

- static create(folderName: String) : Folder
  - Validates and creates a folder if the name is valid. Factory method.
- addPhoto(photo: Photo) : boolean
  - Adds an active photo to the folder if it is not null.
- removePhoto(photo: Photo) : boolean
  - Removes a photo from the folder if it exists in the list.
- listPhotos() : void
  - Displays in console the names (image strings) of all photos inside the folder.
- renameFolder(newName: String) : boolean
  - Renames the folder if the new name is not empty.
- getName() : String
  - Returns the current folder name.

Relationships with other classes:

- Contains → Photo
  - Represented by the photos attribute : ArrayList<Photo>
  - → This is an aggregation relationship: Folder has a collection of photos, but does not “own” them in terms of physical storage.
- Associated with User
  - In the general design, Folder is associated with a User (although it is not directly visible here). It is the user who creates and manages folders.

#### 4.2.4 Gallery Class

Class name: Gallery

Private attributes:

- allPhotos : ArrayList<Photo>
  - List of all photos visible to the user.
- currentIndex : int
  - Controls the current scroll (position in the gallery).

Public builder:

- Gallery()

- Initializes the photo list and index.

Public Methods:

- addPhoto(photo: Photo) : boolean
  - Adds an active photo to the gallery if it is not null.
- showPhotos(quantity: int) : void
  - Prints in console the next active “quantity” photos, from the current position (simulates infinite scroll).
- resetScroll() : void
  - Resets the index to the beginning (allows to show the gallery again from zero).
- getTotalPhotos() : int
  - Returns the total number of photos in the gallery (active or not).

Relationships with other classes:

- Contains → Photo
  - Represented by the allPhotos attribute: ArrayList<Photo> → Lightweight composition: the gallery manages references to photos, it does not create or delete them.

#### 4.2.5 AuthenticationService class

Class name: AuthenticationService

Private attributes:

- registeredUsers : ArrayList<User>
  - List of all registered users in the system.
- activeSessions : ArrayList<User>
  - Users currently logged in.

Public builder:

- AuthenticationService()

Public methods:

- registerUser(user: User) : boolean
  - Adds a new user if it does not exist yet.
- verifyCredentials(username: String, password: String) : boolean
  - Searches for the user and calls its login() method to validate access.

- `createSession(username: String, password: String) : boolean`
  - Validates credentials and logs in if there is no active session for that user.
- `invalidateSession(username: String) : boolean`
  - Logs out the specified user and removes him from active sessions.

Private method:

- `userExists(username: String) : boolean`
  - Checks if the username is already registered (internal use).

Relationships with other classes:

- Use  $\rightarrow$  User
  - Collaborates directly with the User class by invoking `login()`, `logout()` and `getUsername()`.

#### 4.2.6 StorageService class

Class name: StorageService

Private attributes:

- `storedPhotos : ArrayList<Photo>`
  - List of stored photos. It does not physically delete the photos, it only manages them.

Public builder:

- `StorageService()`
  - Initializes the empty list of stored photos.

Public methods:

- `savePhoto(photo: Photo) : boolean`
  - Adds the photo if it is active and has not been stored before.
- `retrievePhoto(index: int) : Photo`
  - Returns a photo if the index is valid, or null if it does not exist.
- `deletePhoto(photo: Photo) : boolean`
  - Calls Photo's `delete()` method. Does not physically delete it from `storedPhotos`.
- `listStoredPhotos() : void`
  - Prints in console the stored photos with author and date.

Relationships with other classes:

- Use → Photo
  - Invokes `getAuthor()`, `getUploaddate()` and `delete()` methods directly.

## BIBLIOGRAFÍA

- ElJacobo. (2024, mayo 30). *CÓMO Documentar con Markdown USANDO VSCode | Windows 10* [Video]. YouTube. <https://youtu.be/GW-PGABYMP8?si=T3L3Ah31MDNWnU3f>
- MoureDev by Brais Moure. (2023, febrero 16). *Curso de GIT y GITHUB desde CERO para PRINCIPIANTES* [Video]. YouTube. <https://youtu.be/3GymExBkKjE?si=yXa4Eypzq5998qH1>
- ChatGPT. (2025). *Asistencia sobre la vinculación entre Git y GitHub* [Respuesta generada por inteligencia artificial]. OpenAI. <https://chat.openai.com/>
- Lucid Software Español. (2019, febrero 4). Tutorial - Diagrama de Clases UML [Video]. YouTube. <https://youtu.be/Z0yLerU0g-Q?si=XAJgsszGMItbSM0N>