# **STYLLE**

**Software Architecture Document**

| Subject | Software Architecture Document |
| --- | --- |
| Version / Status | 1.00 |
| Release Date | 01/06/2024 |
| Filename | styLLe-SAD-v1.00.doc |
| Document Reference | styLLe-SAD |

# **Table Of Contents**

[**STYLLE 1**](#_heading=h.gjdgxs)

[**Table Of Contents 1**](#_heading=h.1fob9te)

[**1. Introduction 5**](#_heading=h.14hx32g)

[1.1. Purpose of the Document 5](#_heading=h.2et92p0)

[1.2. Scope of the Document 5](#_heading=h.3dy6vkm)

[1.3. Intended Audience 5](#_heading=h.1t3h5sf)

[**2. Reference and applicable documents 6**](#_heading=h.17dp8vu)

[**3. Acronyms and abbreviations 7**](#_heading=h.35nkun2)

[**4. Architectural representation 8**](#_heading=h.44sinio)

[**5. Process view 9**](#_heading=h.z337ya)

[5.1. Use Case Diagram 9](#_heading=h.3j2qqm3)

[5.2. User Case Description 10](#_heading=h.1y810tw)

[UC1: Sign up 10](#_heading=h.4i7ojhp)

[UC2: Sign in 11](#_heading=h.3as4poj)

[UC3: Change password 13](#_heading=h.23ckvvd)

[UC4: Edit user 14](#_heading=h.vx1227)

[UC5: Edit user profile picture 16](#_heading=h.3tbugp1)

[UC6: View own image 17](#_heading=h.2lwamvv)

[UC7: View liked image 19](#_heading=h.3ygebqi)

[UC8: Find image 20](#_heading=h.3q5sasy)

[UC9: View image detail 22](#_heading=h.43ky6rz)

[UC10: Upload image 24](#_heading=h.2afmg28)

[UC11: Edit image 26](#_heading=h.haapch)

[UC12: Delete image 28](#_heading=h.2szc72q)

[UC13: Like image 30](#_heading=h.45jfvxd)

[UC14: Copy / Share image 32](#_heading=h.3x8tuzt)

[UC15: Download image 34](#_heading=h.3oy7u29)

[UC16: Add tag 35](#_heading=h.3gnlt4p)

[UC17: Edit tag 37](#_heading=h.38czs75)

[UC18: Delete tag 39](#_heading=h.302dr9l)

[**6. Logical view 42**](#_heading=h.4cmhg48)

[6.1. Purpose 42](#_heading=h.2rrrqc1)

[6.2. Key elements 42](#_heading=h.16x20ju)

[6.2.1. User management component 42](#_heading=h.261ztfg)

[6.2.2. Image management component 43](#_heading=h.356xmb2)

[6.2.3. Metadata management component 44](#_heading=h.1kc7wiv)

[6.2.4. Viewing and interaction component 45](#_heading=h.2jh5peh)

[6.2.5. Search and recommendation component 46](#_heading=h.3im3ia3)

[6.2.6. Security component 46](#_heading=h.1xrdshw)

[**7. Implementation view 48**](#_heading=h.3w19e94)

[7.1. Code Organization and Structure 48](#_heading=h.qbtyoq)

[7.2. User Management Module 49](#_heading=h.320vgez)

[7.2.1. UserController 49](#_heading=h.1h65qms)

[7.2.2. AdminController 49](#_heading=h.415t9al)

[7.3. Authentication Module 49](#_heading=h.2gb3jie)

[7.3.1. AuthService 49](#_heading=h.vgdtq7)

[7.3.2. SecurityService 50](#_heading=h.3fg1ce0)

[7.4. Image Management Module 50](#_heading=h.1ulbmlt)

[7.4.1. ImageController 50](#_heading=h.4ekz59m)

[7.4.2. ImageService 50](#_heading=h.2tq9fhf)

[7.5. Metadata Management Module 50](#_heading=h.18vjpp8)

[7.5.1. MetadataController 51](#_heading=h.3sv78d1)

[7.5.2. TagController 51](#_heading=h.280hiku)

[7.5.3. MetadataService 51](#_heading=h.n5rssn)

[7.6. Viewing and Interaction Module 51](#_heading=h.375fbgg)

[7.6.1. GalleryController 51](#_heading=h.1maplo9)

[7.6.2. InteractionController 51](#_heading=h.46ad4c2)

[7.7. Metadata Management Module 52](#_heading=h.2lfnejv)

[7.7.1. SearchController 52](#_heading=h.10kxoro)

[7.7.2. RecommendationController 52](#_heading=h.3kkl7fh)

[7.8. Core Services Module 52](#_heading=h.1zpvhna)

[7.8.1. UserService 52](#_heading=h.4jpj0b3)

[7.8.2. RecommendationService 52](#_heading=h.2yutaiw)

[7.9. Application Dependencies 53](#_heading=h.1e03kqp)

[7.10. Data Backup Procedure 53](#_heading=h.3xzr3ei)

[7.10.1. Types of Backups 53](#_heading=h.2d51dmb)

[7.10.2. Backup Procedure 53](#_heading=h.sabnu4)

[Incremental Backups: 53](#_heading=h.488uthg)

[Full Backups: 54](#_heading=h.488uthg)

[On-Demand Backups: 54](#_heading=h.488uthg)

[7.10.3. Restore Procedure 54](#_heading=h.1rf9gpq)

[Identify Backup: 54](#_heading=h.4bewzdj)

[7.10.4. Security Measures 55](#_heading=h.2qk79lc)

[7.10.5. Automation and Monitoring 55](#_heading=h.3pp52gy)

[**8. Data view 56**](#_heading=h.33zd5kd)

[8.1. Firebase Realtime Database Structure 56](#_heading=h.1j4nfs6)

[8.1.1. Users Collection 56](#_heading=h.434ayfz)

[8.1.2. Images Collection 56](#_heading=h.xevivl)

[8.1.3. Tags Collection 57](#_heading=h.1wjtbr7)

[8.1.4. Interaction Logs 57](#_heading=h.2vor4mt)

[8.2. Firebase Storage Structure 58](#_heading=h.3utoxif)

[8.2.1. Images Directory 58](#_heading=h.29yz7q8)

[8.2.2. User Profile Images Directory 58](#_heading=h.393x0lu)

[8.3. Relationships Between Data Entities 58](#_heading=h.12jfdx2)

[8.3.1. User-Image Relationship 58](#_heading=h.3mj2wkv)

[8.3.2. Interaction Logs and Image/User Relationship 59](#_heading=h.21od6so)

[8.3.3. Tag-Image Relationship 59](#_heading=h.gtnh0h)

[8.3.4. Image-Tag Relationship (Inverse) 59](#_heading=h.30tazoa)

[**9. Deployment view 60**](#_heading=h.3zy8sjw)

[9.1. Google Play Store Deployment 60](#_heading=h.2f3j2rp)

[9.2. Apple AppStore Deployment 60](#_heading=h.u8tczi)

[9.3. Continuous Deployment 61](#_heading=h.3e8gvnb)

[9.4. Monitoring and Maintenance 61](#_heading=h.1tdr5v4)

[**10. System definition 62**](#_heading=h.2sioyqq)

[10.1. Functional Requirements 62](#_heading=h.17nz8yj)

[10.2. Design Constraints 64](#_heading=h.ly7c1y)

[10.3. Quality Attribute Requirements 65](#_heading=h.35xuupr)

[10.3.1. Performance 65](#_heading=h.1l354xk)

[10.3.2. Security 67](#_heading=h.zdd80z)

[10.3.3. Scalability 70](#_heading=h.1csj400)

[10.3.4. Usability 71](#_heading=h.2bxgwvm)

[10.3.5. Maintainability 73](#_heading=h.1q7ozz1)

# **Table Of Images**

[Image 1: Use case List 10](#_heading=h.3ohklq9)

[Image 2: Sign up sequence diagram 11](#_heading=h.3whwml4)

[Image 3: Sign in sequence diagram 13](#_heading=h.49x2ik5)

[Image 4: Change password sequence diagram 15](#_heading=h.32hioqz)

[Image 5: Edit user sequence diagram 16](#_heading=h.2u6wntf)

[Image 6: Edit user profile picture sequence diagram 18](#_heading=h.nmf14n)

[Image 7: View own image sequence diagram 20](#_heading=h.4k668n3)

[Image 8: View liked image sequence diagram 21](#_heading=h.1rvwp1q)

[Image 9: Find image sequence diagram 23](#_heading=h.kgcv8k)

[Image 10: View image detail sequence diagram 25](#_heading=h.1x0gk37)

[Image 11: Upload image sequence diagram 27](#_heading=h.48pi1tg)

[Image 12: Edit image sequence diagram 29](#_heading=h.2fk6b3p)

[Image 13: Delete image sequence diagram 31](#_heading=h.meukdy)

[Image 14: Like image sequence diagram 33](#_heading=h.1yyy98l)

[Image 15: Copy / Share image sequence diagram 35](#_heading=h.1qoc8b1)

[Image 16: Download image sequence diagram 37](#_heading=h.1idq7dh)

[Image 17: Add tag sequence diagram 38](#_heading=h.1a346fx)

[Image 18: Edit tag sequence diagram 41](#_heading=h.11si5id)

[Image 19: Delete tag sequence diagram 43](#_heading=h.thw4kt)

[Image 20: Logical view 45](#_heading=h.3qwpj7n)

[Image 21: Functional Overview 65](#_heading=h.3rnmrmc)

# **Table Of Tables**

[Table 1 Reference table 8](#_heading=h.lnxbz9)

[Table 2: Abbreviations and Acronyms table 9](#_heading=h.1ksv4uv)

[Table 3: Sign up description 12](#_heading=h.2xcytpi)

[Table 4: Sign up business rule 13](#_heading=h.qsh70q)

[Table 5: Sign in description 13](#_heading=h.1pxezwc)

[Table 6: Sign in sequence diagram 15](#_heading=h.3o7alnk)

[Table 7: Change password description 15](#_heading=h.ihv636)

[Table 8: Change password business rule 16](#_heading=h.2grqrue)

[Table 9: Edit user description 17](#_heading=h.3fwokq0)

[Table 10: Edit user business rule 18](#_heading=h.3tbugp1)

[Table 11: Edit user profile picture description 18](#_heading=h.28h4qwu)

[Table 12: Edit user profile picture business rule 20](#_heading=h.1mrcu09)

[Table 13: View own image description 20](#_heading=h.111kx3o)

[Table 14: View own image business rule 22](#_heading=h.1egqt2p)

[Table 15: View liked image description 22](#_heading=h.2dlolyb)

[Table 16: View liked image business rule 23](#_heading=h.2r0uhxc)

[Table 17: Find image description 23](#_heading=h.25b2l0r)

[Table 18: Find image business rule 25](#_heading=h.1jlao46)

[Table 19: View image detail description 25](#_heading=h.2iq8gzs)

[Table 20: View image detail business rule 27](#_heading=h.1baon6m)

[Table 21: Upload image description 27](#_heading=h.pkwqa1)

[Table 22: Upload image business rule 30](#_heading=h.3mzq4wv)

[Table 23: Edit image description 30](#_heading=h.319y80a)

[Table 24: Edit image business rule 31](#_heading=h.1tuee74)

[Table 25: Delete image description 32](#_heading=h.184mhaj)

[Table 26: Delete image business rule 33](#_heading=h.1ljsd9k)

[Table 27: Like image description 34](#_heading=h.2koq656)

[Table 28: Like image business rule 35](#_heading=h.2y3w247)

[Table 29: Copy / Share image 35](#_heading=h.2ce457m)

[Table 30: Copy / Share image business rule 37](#_heading=h.2pta16n)

[Table 31: Download image description 37](#_heading=h.243i4a2)

[Table 32: Download imgae business rule 38](#_heading=h.2hio093)

[Table 33: Add tag description 39](#_heading=h.1vsw3ci)

[Table 34: Add tag business rule 41](#_heading=h.2981zbj)

[Table 35: Edit tag description 41](#_heading=h.1nia2ey)

[Table 36: Edit tag business rule 43](#_heading=h.20xfydz)

[Table 37: Delete tag description 43](#_heading=h.1f7o1he)

[Table 38: Delete tag business rule 45](#_heading=h.1smtxgf)

[Table 39: Code organiztion and Structure 53](#_heading=h.i17xr6)

[Table 40: Users collection 60](#_heading=h.2i9l8ns)

[Table 41: Images collection 61](#_heading=h.3hej1je)

[Table 42: Tags collection 61](#_heading=h.4gjguf0)

[Table 43: Interaction logs 62](#_heading=h.1au1eum)

[Table 44: Images directory 62](#_heading=h.p49hy1)

[Table 45: User profile images directory 62](#_heading=h.1o97atn)

[Table 46: Image Loading Time Scenario 70](#_heading=h.452snld)

[Table 47: Memory Management Scenario 71](#_heading=h.2k82xt6)

[Table 48: Registing Attacks Scenario 72](#_heading=h.3jd0qos)

[Table 49: Detecting Attacks Scenario 73](#_heading=h.1yib0wl)

[Table 50: Recovering from Attacks Scenario 74](#_heading=h.4ihyjke)

[Table 51: Database Scalibity Scenario 75](#_heading=h.3ws6mnt)

[Table 52: Search Algorithm Scenario 76](#_heading=h.r2r73f)

[Table 53: Personalized Recommendations Scenario 77](#_heading=h.3b2epr8)

[Table 54: Modular Code Structure 79](#_heading=h.4a7cimu)

# **1. Introduction**

## **1.1. Purpose of the Document**

The aim of this document is to provide a comprehensive architectural overview of the styLLE system. This document describes how functional analysis and use cases are translated and structured in the architecture by the development team.

## **1.2. Scope of the Document**

This document presents the technical architecture of the styLLe system. In this document, we focus on the choices made for the styLLe system. Hereafter, the readers will find information about the frameworks, tools and technologies used by the styLLe system.

## **1.3. Intended Audience**

The present document is intended to be read by the following people:

* Publishing operation team;
* Publications Office Project Team;
* Developments Project Team.

# 

# **2. Reference and applicable documents**

This section contains the lists of all references and applicable document. When referring to any of the documents below, the bracketed reference will be used in the text, such as [[R01](https://docs.google.com/document/d/1qZB3foMcHtBu3jBBwYElo_4tUslK-C4M/edit#bookmark=id.2hio093)].

| **Reference Documents** | | | | |
| --- | --- | --- | --- | --- |
| **Ref.** | **Title** | **Reference** | **Version** | **Date** |
| R01 | styLLe-SRS-Software Requirement Specification | styLLe-SRS | 1.00 | 07/09/2009 |
| R02 | styLLe-ADD-ArchitectureDesign Document | styLLE-Add | 1.00 | 28/04/2010 |

*Table 1 Reference table*

# **3. Acronyms and abbreviations**

| **Abbreviations and Acronyms** | |
| --- | --- |
| **Abbreviation** | **Meaning** |
| CRUD | Create, read, update, delete |
| FB | Firebase |
| FDB | Firebase Database |
| FS | Firebase Storage |

*Table 2: Abbreviations and Acronyms table*

# **4. Architectural representation**

This document is the result of the design phase.

This document presents the necessary views to represent the software architecture:

* The Process View: presents the dynamic aspects of the system, focusing on use case interactions and detailed specifications of each use case.
* The Logical View: presents the decomposition of the software architecture into subsystems and packages;
* The Implementation View describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems;
* The Data View describes the persistent data storage perspective of the system;
* The Deployment View describes the physical infrastructure on which the styLLe software is deployed and run. It specifies the physical nodes and network configuration that executes the software, and also maps the processes defined in the Process View onto physical nodes.

# 

# **5. Process view**

## **5.1. Use Case Diagram**

*Image 1: Use case list*

## **5.2. User Case Description**

### UC1: Sign up

| **Name** | **Sign up** |
| --- | --- |
| **Description** | This use case allows users to sign up for their own account in StyLLe application . They can choose between two options to sign up, sign up with email and password or with Google |
| **Actor** | User |
| **Trigger** | * A user want to sign up for a new account |
| **Pre-condition** | * The staff has accessed the StyLLe interface. |
| **Post-condition** | * POST-1. The user can log in with their newly created account. |

*Table 3: Sign up description*

Sequence Diagram

### 

*Image 2: Sign up sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR1*** | **Loading Screen Rules:**   * The system loads the “Sign up” screen. |
| ***(2)*** | ***BR2*** | * The system signs up a new account for user with “signUp” method.   **[signUp] method**   * There are 2 options to sign up, "emailPassword" or "Google"   + If [signupOption] = "emailPassword", the user has to enter his [email] and [password] as account credentials.     - If [email] and [password] is "valid", then show a success message for the user.     - If [email] and [password] is "inValid", then show error message for user and require re-enter his credential   + If [signupOption]= "Google", then their google accounts be checked by GoogleAuthService     - If the checked account is "valid" then show a success message for the user.     - If the checked account is "inValid" then show an error message for the user. * On [Error], throw [GenericException] |

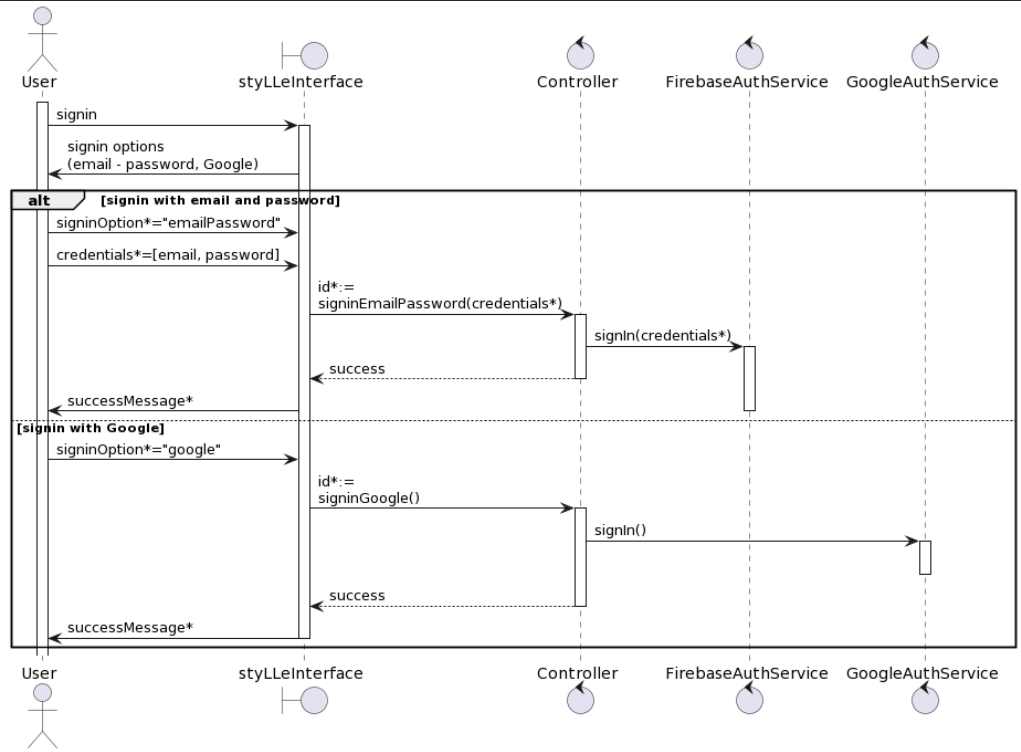
*Table 4: Sign up business rule*

### UC2: Sign in

| **Name** | **Sign in** |
| --- | --- |
| **Description** | When the users want to access the StyLLe application or do some actions like viewing photos, uploading photos… they have to sign in to the application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to access the application. |
| **Pre-condition** | * PRE-1. User already had his own account. |
| **Post-condition** | * POST-1. User is logged in to the application and can do actions in the application. |

*Table 5: Sign in description*

Sequence Diagram



*Image 3: Sign in sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR3*** | **Loading Screen Rules:**   * The system loads the “Sign in” screen. |
| ***(2)*** | ***BR4*** | * The system signs the user in with “signIn” method   **[signIn] method**   * There are 2 options to sign in, "emailPassword" or "Google"   + If [signupOption] = "emailPassword", the user has to enter his [email] and [password], then the system will check his account information in database     - If [email] and [password] is "exist", then show a success message for the user, leading the user to the Home screen of the app.     - If [email] and [password] is "notExist", then show an error message for the user and require re-enter his credential.   + If [signupOption]= "Google", then their google accounts be checked by GoogleAuthService     - If the checked account "exist" then shows a success message for the user, leading the user to the Home screen of the app.     - If the checked account is "notExist" then show an error message for the user. * On [Error], throw [GenericException] |

*Table 6: Sign in sequence diagram*

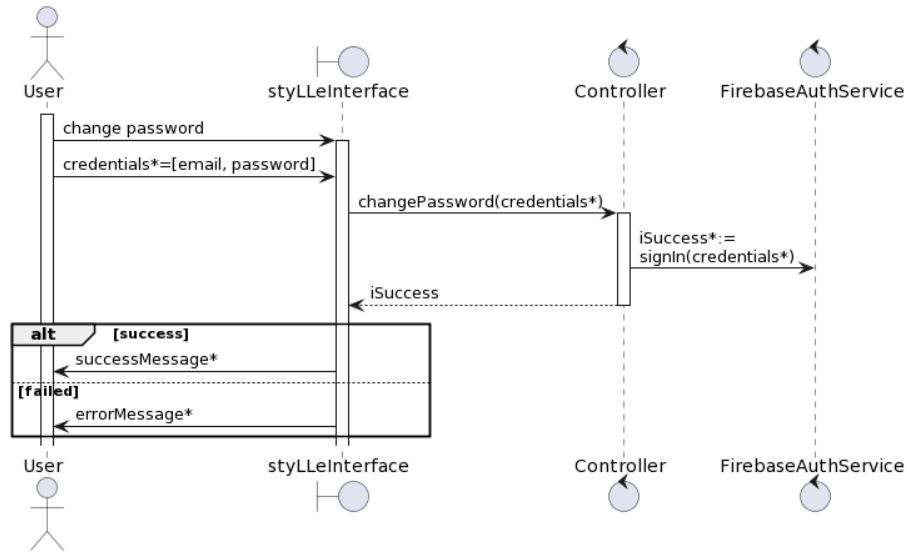
### 

### UC3: Change password

| **Name** | **Modify Image** |
| --- | --- |
| **Description** | The user can change their old password to a new password for more convenience or insecure security. |
| **Actor** | User, Admin |
| **Trigger** | * A user indicates that he needs to change his account password. |
| **Pre-condition** | * The user has accessed the StyLLe application. |
| **Post-condition** | * The user changed his account password successfully. |

*Table 7: Change password description*

Sequence Diagram



*Image 4: Change password sequence diagram*

### 

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR5*** | **Loading Screen Rules:**   * The system loads the “Change Password” screen. |
| ***(3)*** | ***BR6*** | * The user has to enter his new password, then the system will change the account password with the “changePassword” method.   **[changePassword] method**   * This method takes a “newPassword” as arguments   + If “newPassword” is "valid" then edit the user password with “newPassword”, leading the user to the Sign in page of the app.   + If the new information is "inValid" then throw [NotValidDataException]. * On [Error], throw [GenericException] |

*Table 8: Change password business rule*

### UC4: Edit user

| **Name** | **Edit user** |
| --- | --- |
| **Description** | Once the user indicates that he wants to edit his account information such as name, email, phone number, address… he can change his account information on his own. |
| **Actor** | User |
| **Trigger** | * User wants to edit his account information |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | The user can change his account information. |

*Table 9: Edit user description*

Sequence Diagram

### 

*Image 5: Edit user sequence diagram*

Business Rules

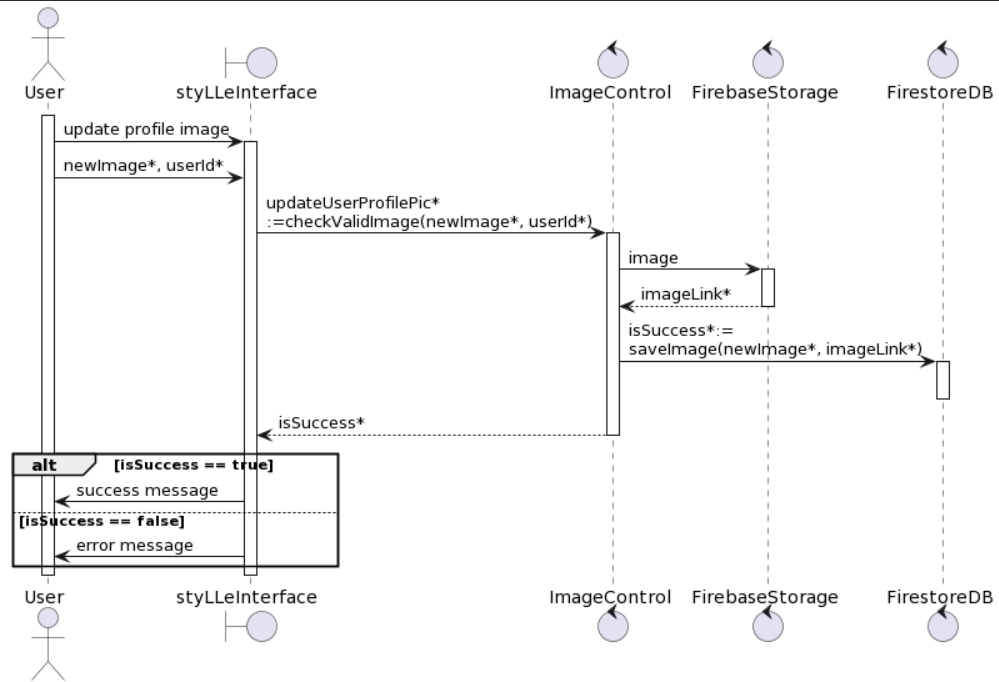
| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR7*** | **Loading Screen Rules:**   * The system loads the “Profile” screen. |
| ***(2)*** | ***BR8*** | * When the user chooses the "Edit" button "Profile" screen, the system loads the “EditProfile” screen and then enters the new information (name, address, phoneNumber…) of profile. * When User press “Save” button, the system will prompt a confirmation message (Refer to MSG 3) * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Update the newly modified item by calling the [editUser] method, while catching any errors that may occur     - If [Error], the system shows MSG 5     - Else,       * The system shows MSG 7       * Exit the “EditProfile” screen   **[editUser] method**   * This method takes a “userData” as arguments   + If the new information is "valid" and then edit new user information with “userData”   + If the new information is "inValid" then throw [NotValidDataException]. * On [Error], throw [GenericException] |

*Table 10: Edit user business rule*

### UC5: Edit user profile picture

| **Name** | **Edit user profile picture** |
| --- | --- |
| **Description** | User indicates that he wants to upload his new profile picture |
| **Actor** | User |
| **Trigger** | * The user needs to change his new profile picture. |
| **Pre-condition** | Staff is logged into StyLLe. |
| **Post-condition** | POST-1: User changed his profile picture successfully. |

*Table 11: Edit user profile picture description*

Sequence Diagram

*Image 6: Edit user profile picture sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR9*** | **Loading Screen Rules:**   * The system loads the “Edit profile picture” screen. |
| ***(2)*** | ***BR10*** | * When user selects an photo from gallery and upload to the system, the system will check validation of the photo   + If [checkValidImage(photo)] = "true", then the system will edit the user profile picture with [editProfilePicture] method.   + If checkValidImage(photo) = "false", then throw [NotValidDataException].   [**editProfilePicture**] **method**   * The system gets the [imageLink] from FirebaseStorage with the image the user uploaded * Saves the new image with [imageLink] to the FirestoreDatabase   + If [saveImage] = "success" then shows MSG6 to the user.   + Else, throw [ErrorWhileDoingActionException]. * On [Error], throw [GenericException] |

*Table 12: Edit user profile picture business rule*

### 

### UC6: View own image

| **Name** | **View own image** |
| --- | --- |
| **Description** | User indicates that he wants to see his own images on his profile which he has uploaded to the community. |
| **Actor** | User |
| **Trigger** | * The user needs to view his own image on his profile. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User can view his own uploaded images. |

*Table 13: View own image description*

Sequence Diagram

### 

*Image 7: View own image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR11*** | **Loading Screen Rules:**   * The system loads the “Profile” screen. |
| ***(2)*** | ***BR12*** | * When the user presses the button "View own image" on the "Profile screen", the system will retrieve all "Image" items with [viewOwnImage] method.   [**viewOwnImage**] **method**   * This method takes a “userId” as arguments * The system retrieves the “Image” item where [userId] = [this.userId].   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "OwnImageList". * On [Error], throw [GenericException] |

*Table 14: View own image business rule*

### 

### UC7: View liked image

| **Name** | **View liked image** |
| --- | --- |
| **Description** | User indicates that he wants to see his liked images on his profile. |
| **Actor** | User |
| **Trigger** | * The user needs to see his liked images on his profile. |
| **Pre-condition** | Staff is logged into ESMS. |
| **Post-condition** | POST-1: User can view his liked images. |

*Table 15: View liked image description*

Sequence Diagram

### 

*Image 8: View liked image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR13*** | **Loading Screen Rules:**   * The system loads the “Profile” screen. |
| ***(2)*** | ***BR14*** | * When the user presses the button "View liked image" on the "Profile screen", the system will retrieve all "Image" items with [**viewLikedImage**] method.   [**viewLikedImage**] **method**   * This method takes a “userId” as arguments * The system retrieves the “Image” item where [likedBy] = [this.userId].   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "LikedImageList". * On [Error], throw [GenericException] |

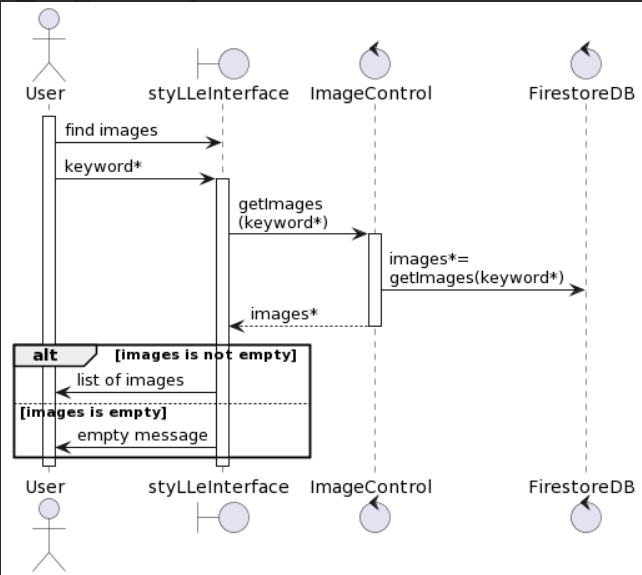
*Table 16: View liked image business rule*

### 

### UC8: Find image

| **Name** | **Find image** |
| --- | --- |
| **Description** | This use case allows users to search for images in the StyLLe app on the images name or tag. |
| **Actor** | User |
| **Trigger** | * The user wants to search for a particular image or image related to tag |
| **Pre-condition** | User is logged into the app. |
| **Post-condition** | POST-1. The user can view the list of found images and the general information of each image.  POST-2. The user can select an image to view detailed information or can perform other actions on the image (like, download, copy, share…). |

*Table 17: Find image description*

Sequence Diagram

*Image 9: Find image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR15*** | **Loading Screen Rules:**   * The system loads the “Search” screen. |
| ***(2)*** | ***BR16*** | * The system retrieves “Popular Image” items where arranged by the hearts given by other users and newly updated by the community.   + If there is no retrieved value, the system shows an empty list.   + Else, display all the retrieved items in the [Popular Image List]. * And the system retrieves some collections of images group by tag for user easily to explore |
| ***(3)*** | ***BR17*** | * When User press “Search” button after type a [Keyword] in the search bar, the system retrieves “Image” items with [findImage] method   [**findImage**] **method**   * This method takes a “Image” item as arguments * The system retrieves “Image” item where [Item.name] contains [Keyword] or [Item.tag] contains [Keyword]   + If there is retrieved value, the system shows MSG 2   + Else, display all the retrieved items in the [SearchImageList]. * On [Error], throw [GenericException] |

*Table 18: Find image business rule*

### 

### UC9: View image detail

| **Name** | **View image detail** |
| --- | --- |
| **Description** | User wants to see detailed information of a specific image |
| **Actor** | User, Admin |
| **Trigger** | * The user indicates that he wants to see more information about a specific image. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: Image information (Information of poster, description, tags… ) is shown on the screen |

*Table 19: View image detail description*

Sequence Diagram

### 

*Image 10: View image detail sequence diagram*

#### 

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR18*** | **Loading Screen Rules:**   * The system loads the “Image” list on the screen. |
| ***(2)*** | ***BR19*** | * When user selects an photo from “Image” list on the screen, the system will get information of that item with [getImageDetail] method   + If getImageDetail(imageName)" = "Success", then show the image information on the screen   + Else, show MSG15   **[getImageDetail] method**   * This method takes a “Image” item as arguments * The system retrieves “Image” item where [name] = [Image.name] and assign it to [itemToShow]   + If [itemToShow.length] == 0, throw [ImageNotFoundException].   + Else, return "Success". * On [Error], throw [GenericException] |

*Table 20: View image detail business rule*

#### 

### UC10: Upload image

| **Name** | **Upload image** |
| --- | --- |
| **Description** | User wants to upload images from the gallery to the StyLLe application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to upload images from the gallery to the StyLLe application. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User uploads his image from gallery successfully. |

*Table 21: Upload image description*

Sequence Diagram

### 

*Image 11: Upload image sequence diagram*

#### 

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR20*** | **Loading Screen Rules:**   * The system loads the “Edit profile picture” screen. |
| ***(2)*** | ***BR21*** | * When user selects an photo from gallery and upload to the system, the system will check validation of the photo   + If checkValidImage(photo) = "true", then the system will upload the image with the [uploadImage] method.   + If checkValidImage(photo) = "false", then throw [NotValidDataException] * Then the system loads the “UploadLoad” screen. * User have to enter the information in form: * **Image information form**   + **Description**: text   + **Tag**: text (No special characters),   **[uploadImage] method**   * The system gets the [imageLink] from FirebaseStorage with the image the user uploaded * Saves the new image with [imageLink] to the FirestoreDatabase   + If [saveImage] = "success" then shows MSG6 to the user.   + Else, throw [ErrorWhileDoingActionException]. |
|  |  | * When User press Submit form, perform validation on the fields:   + All fields must be filled   + **Description**: 5-100 chars, not null.   + **Tag**: 5-50 chars. * The form can only be submitted once all the fields are validated. * If the form is valid, save “Image ”to the database |
| ***(3)*** | ***BR22*** | * When User press “Add” button, the system will prompt a confirmation message (Refer to MSG 3) * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Store the newly added item calling the [addImage] method, while catching any errors that may occur, then exit the “AddImage” screen.   **[addImage] method**   * This method takes a “Image” item as arguments * Call [checkValidImage], The system will retrieve “Image” items where [name] = [newImage.name] and assign them to [ImageList] * Store [newImage] into the database * On [Error], throw [GenericException] |

*Table 22: Upload image business rule*

### UC11: Edit image

| **Name** | **Edit image** |
| --- | --- |
| **Description** | The User is able to edit some information of a image e.g., modifying the description, tag, … and saving the modified image. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to edit some information of image |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User uploads can edit an image"s information successfully. |

*Table 23: Edit image description*

Sequence Diagram

### 

*Image 12: Edit image sequence diagram*

#### 

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR23*** | **Loading Screen Rules:**   * The system loads the “Profile” screen, then the user chooses the "View own images" button to show images he uploaded on the screen, the system will retrieve all "Image" items where [userId] = [this.userId].   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "OwnImageList". |
| ***(2)*** | ***BR24*** | * When the user chooses the "Edit" button of an photo from "OwnImageList", the system loads the "EditingImage" screen and the enter the new information (description, tag…) of the image that he wants to edit |
| ***(3)*** | ***BR25*** | * When User press “Save” button, the system will prompt a confirmation message (Refer to MSG 3) * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Update the newly modified item by calling the [editImage] method, while catching any errors that may occur     - If [Error], the system shows MSG 5     - Else,       * The system shows MSG 7       * Exit the “Edit Image” screen   **[editImage] method**   * This method takes a “Image” item as arguments * The system retrieves “Image” item where [name] = [Image.name]   + If [updateImage(updateImageData)] = "false" then throw [ErrorWhileDoingActionException].   + Else, edit the image with the new information. * On [Error], throw [GenericException] |

*Table 24: Edit image business rule*

### UC12: Delete image

| **Name** | **Delete image** |
| --- | --- |
| **Description** | User wants to delete images that he has uploaded to the application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to delete images that he has uploaded. |
| **Pre-condition** | Useris logged into ESMS. |
| **Post-condition** | POST-1: User can delete his uploaded images. |

*Table 25: Delete image description*

Sequence Diagram

### 

*Image 13: Delete image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR26*** | **Loading Screen Rules:**   * The system loads the “Profile” screen, then the user chooses the "View own images" button to show images he uploaded on the screen, the system will retrieve all "Image" items where [userId] = [this.userId].   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "OwnImageList". |
| ***(2)*** | ***BR27*** | When the user chooses the "Delete" button of an photo from "OwnImageList", then the user press “Delete” button, the system will prompt a confirmation message (Refer to MSG 8)   * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Delete the selected item calling the [deleteImage] method, while catching any errors that may occur     - If [Error],       * If [Error] = [ImageNotExistsException], the system shows MSG2       * Else, the system shows MSG 9     - Else,       * The system shows MSG 10   **[deleteImage] method**   * This method takes a “Image” item as arguments * The system retrieves “Image” item where [name] = [Image.name] and assign it to [itemToDelete]   + If [itemToDelete.length] == 0, throw [ImageNotFoundException].   + Else, delete itemToDelete from the database. * On [Error], throw [GenericException] |

*Table 26: Delete image business rule*

### 

### UC13: Like image

| **Name** | **Upload image** |
| --- | --- |
| **Description** | User wants to like his favorite images or he wants to save it for later to see. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to like images and add them to his liked images list. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User can like his favorite images. |

*Table 27: Like image description*

Sequence Diagram

### 

*Image 14: Like image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR28*** | **Loading Screen Rules:**   * The system loads the “ImageList” on the screen. |
| ***(2)*** | ***BR29*** | * When a user selects a photo from “ImageList '', the system will add the liked image to “LikedImageList” of the user by [likeImage] method.   [**likeImage**] **method**   * This method takes a “Image” as arguments * Set “Image.likedby” = “this.userId” * On [Error], throw [GenericException] |

*Table 28: Like image business rule*

### UC14: Copy / Share image

| **Name** | **Copy / Share image** |
| --- | --- |
| **Description** | User can copy an image or share an image link to other friends in StyLLe application or other platforms. |
| **Actor** | User, Admin |
| **Trigger** | * The user indicates that he wants to copy or share an image link. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: Users can copy or share images to others. |

*Table 29: Copy / Share image*

Sequence Diagram

### 

*Image 15: Copy / Share image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR30*** | **Loading Screen Rules:**   * The system loads the “ImageList” on the screen. |
| ***(2)*** | ***BR31*** | * When a user selects a photo from “ImageList'', the system will copy the image to clipboard or share the image by [copyOrShareImage] method.   **[copyOrShareImage]** **method**   * This method takes a “Image” as arguments   + If the user chooses the “Copy” button, the system will copy the “Image” to clipboard and the user can paste it anywhere.   + If the user chooses the “Share” button, the system will get the [imageLink] from FirebaseStorage and show a bottomsheet in which the user can choose friends to share with or other platforms to share images. * On [Error], throw [GenericException] |

*Table 30: Copy / Share image business rule*

### UC15: Download image

| **Name** | **Download image** |
| --- | --- |
| **Description** | Users can download images to his device to be able to see it directly from their photo albums. |
| **Actor** | User, Admin |
| **Trigger** | * The user indicates that he downloads images to his device. |
| **Precondition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User downloads his image to his device successfully. |

*Table 31: Download image description*

Sequence Diagram

### 

*Image 16: Download image sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR32*** | **Loading Screen Rules:**   * The system loads the “ImageList” on the screen. |
| ***(2)*** | ***BR33*** | * When a user selects a photo from “ImageList'', the system will download the image by [downloadImage] method.   **[downloadImage]** **method**   * This method takes a “Image” as arguments * The system will get the [imageLink] from FirebaseStorage and download image to device by [imageLink] * On [Error], throw [GenericException] |

*Table 32: Download imgae business rule*

### UC16: Add tag

| **Name** | **Add tag** |
| --- | --- |
| **Description** | User wants to upload images from the gallery to the StyLLe application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to upload images from the gallery to the StyLLe application. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User uploads his image from gallery successfully. |

*Table 33: Add tag description*

Sequence Diagram

### 

*Image 17: Add tag sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR34*** | **Loading Screen Rules:**   * The system loads the “Tag Management” screen. * The system retrieves “Tag” items   + If there is no retrieved value, the system shows MSG 1   + Else, display all the retrieved items in the ”TagList”. |
| ***(2)*** | ***BR35*** | * When the user clicks the “AddTag” button, the system loads the “AddTag” screen. * User have to enter the information in form: * **Tag information form**   + Name: text (no special characters) |
|  |  | * When User press Submit form, perform validation on the fields:   + All fields must be filled   + **Name**: 5-50 chars (no special characters), not null * The form can only be submitted once all the fields are validated. * If the form is valid, save “Tag ”to the database |
| ***(3)*** | ***BR36*** | * When User press “Add” button, the system will prompt a confirmation message (Refer to MSG 3) * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Store the newly added item calling the [addTag] method, while catching any errors that may occur     - If [Error],       * If [Error] = [TagAlreadyExistsException], the system shows MSG4       * Else, the system shows MSG 5     - Else,       * the system shows MSG 6       * Exit the “AddTag” screen   **[addTag] method**   * This method takes a “Tag” item as arguments * Call [checkValidTag], The system will retrieve “Tag” items where [name] = [newTag.name] and assign them to [tagList] * If [tagList.length] > 0, throw [TagtAlreadyExistsException]. * Else, store [newTag] into the database * On [Error], throw [GenericException] |

*Table 34: Add tag business rule*

### UC17: Edit tag

| **Name** | **Edit tag** |
| --- | --- |
| **Description** | User wants to upload images from the gallery to the StyLLe application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to upload images from the gallery to the StyLLe application. |
| **Pre-condition** | User is logged into ESMS. |
| **Post-condition** | POST-1: User uploads his image from gallery successfully. |

*Table 35: Edit tag description*

Sequence Diagram

### 

*Image 18: Edit tag sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR37*** | **Loading Screen Rules:**   * The system loads the “Tag” screen   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "TagList". |
| ***(2)*** | ***BR38*** | * When the user chooses the "Edit" button of a tag from "TagList", the system loads the "EditTag" screen and the enter the new information (name…) of the tag that he wants to edit |
| ***(3)*** | ***BR39*** | * When User press “Save” button, the system will prompt a confirmation message (Refer to MSG 3) * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Update the newly modified item by calling the [editTag] method, while catching any errors that may occur     - If [Error], the system shows MSG 5     - Else,       * The system shows MSG 7       * Exit the “EditTag” screen   **[editTag] method**   * This method takes a “Tag” item as arguments * The system retrieves “Tag” item where [name] = [Tag.name]   + If [updateTag(updateTagData)] = "false" then throw [ErrorWhileDoingActionException].   + Else, edit the tag with the new information. * On [Error], throw [GenericException] |

*Table 36: Edit tag business rule*

### UC18: Delete tag

| **Name** | **Delete tag** |
| --- | --- |
| **Description** | User wants to upload images from the gallery to the StyLLe application. |
| **Actor** | User |
| **Trigger** | * The user indicates that he wants to upload images from the gallery to the StyLLe application. |
| **Pre-condition** | Useris logged into ESMS. |
| **Post-condition** | POST-1: User uploads his image from gallery successfully. |

*Table 37: Delete tag description*

Sequence Diagram

### 

*Image 19: Delete tag sequence diagram*

Business Rules

| **Activity** | **BR Code** | **Description** |
| --- | --- | --- |
| ***(1)*** | ***BR1*** | **Loading Screen Rules:**   * The system loads the “Tag” screen   + If there is no retrieved item, the system return MSG1   + Else, display all the retrieved items from "TagList". |
| ***(2)*** | ***BR2*** | When the user chooses the "Delete" button of an tag from "TagList", the system will prompt a confirmation message (Refer to MSG 8)   * If [Confirmation] = “Cancel”,   + Close the confirmation modal. * Else,   + Close the confirmation modal   + Delete the selected item calling the [deleteTag] method, while catching any errors that may occur     - If [Error],       * If [Error] = [TagNotExistsException], the system shows MSG2       * Else, the system shows MSG 9     - Else,       * The system shows MSG 10   **[deleteTag] method**   * This method takes a “Tag” item as arguments * The system retrieves “Tag” item where [name] = [Tag.name] and assign it to [itemToDelete]   + If [itemToDelete.length] == 0, throw [TagNotFoundException].   + Else, delete itemToDelete from the database. * On [Error], throw [GenericException] |

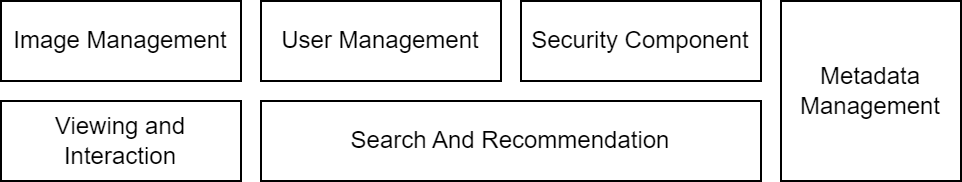
*Table 38: Delete tag business rule*

# **6. Logical view**

## **6.1. Purpose**

The Logical View provides a detailed representation of the system's functionality and its structure. It defines the key functional elements, their responsibilities, relationships, and how they interact to fulfill the requirements.

## **6.2. Key elements**



*Image 20: Logical view*

### **6.2.1. User management component**

The UserController is responsible for managing user-related operations. It handles user registration, login, logout, and profile management. When a new user signs up, the UserController collects user details and interacts with the AuthService to create a new account in the Firebase Authentication system. For logging in, it validates the user credentials against Firebase Auth. It also manages user sessions, ensuring secure login and logout processes. When users want to update their profiles, the UserController handles the request and interacts with the UserService to update the user information in the Firebase Realtime Database.

Responsibilities:

* Handles user registration (signUp())
* Manages user login (logIn())
* Allows users to log out (logOut())
* Manages profile updates (updateProfile())

Interactions:

* Interacts with AuthService for authentication and authorization.
* Communicates with UserService for user-related operations.

**AdminController:**

The AdminController is dedicated to administrative tasks. It provides functionality for creating, editing, and deleting user accounts, ensuring only authorized administrators can perform these operations. When an admin creates a new user, the AdminController validates the input, interacts with the AuthService to create the account, and then uses the UserService to store additional user details. Editing and deleting users follow similar interactions, ensuring that changes are properly reflected in the Firebase Realtime Database.

Responsibilities:

* Allows admin to create new tags(createTag())
* Enables admin to edit tags information (editTag())
* Provides functionality for deleting tags (deleteTag())
* Provides functionality for deleting users (deleteUser())

Interactions:

* Interacts with AuthService to ensure only authorized admins can perform operations.
* Communicates with UserService for managing user accounts.

### **6.2.2. Image management component**

The ImageController handles image uploads and editing requests from users and delegates the processing and storage tasks to the ImageService. The ImageService processes the image files, performs necessary transformations, and stores them in Firebase Storage.

**ImageController:**

The ImageController manages all image-related operations, including uploading, editing, and deleting images. When a user uploads an image, the ImageController handles the file upload process, interacting with the ImageService to process and store the image in Firebase Storage. It also ensures that the associated metadata, such as descriptions and tags, are stored in the Firebase Realtime Database through the MetadataController. For image editing, the ImageController provides functionality to crop and rotate images, utilizing the ImageService for these operations. Deleting an image involves removing the image file from Firebase Storage and its metadata from the database.

Responsibilities:

* Manages image uploads (uploadImage())
* Provides image editing functionality (editImage())
* Handles image deletion (deleteImage())

Interactions:

* Communicates with ImageService for image processing and storage.
* Interacts with MetadataController for updating image metadata.

**ImageService:**

The ImageService is responsible for processing images and managing their storage. It handles tasks such as image compression, resizing, cropping, and rotating to optimize performance and ensure a smooth user experience. The service interacts with Firebase Storage to store and retrieve images, ensuring efficient and secure management of image files. Additionally, the ImageService works closely with the MetadataService to ensure that each image is correctly linked with its metadata, such as descriptions and tags.

Responsibilities:

* Processes images (e.g., cropping, rotating) (processImage())
* Stores images in Firebase Storage (storeImage())
* Retrieves images from storage (retrieveImage())

Interactions:

* Interfaces with Firebase Storage API for storing and retrieving images.
* Collaborates with MetadataService to link images with their metadata.

### **6.2.3. Metadata management component**

The MetadataController manages image metadata, such as descriptions and tags, by interacting with the MetadataService. The MetadataService updates the metadata in the Firebase Realtime Database, ensuring that each image's metadata is accurately stored and retrievable.

**MetadataController:**

The MetadataController manages the metadata associated with images, including descriptions and tags. When a user adds or edits a description for an image, the MetadataController handles the request and updates the relevant metadata in the Firebase Realtime Database through the MetadataService. For tags, the MetadataController provides functionality to add, edit, and manage tags associated with images, ensuring that each image is correctly categorized for search and recommendation purposes.

Responsibilities:

* Allows users to add descriptions to images (addDescription())
* Manages the addition of tags to images (addTags())
* Provides functionality to edit metadata (editMetadata())

Interactions:

* Communicates with MetadataService for metadata operations.
* Interacts with ImageController to ensure metadata is linked to images.

**TagController:**

The TagController is specifically designed for managing tags. It allows administrators to create new tags, edit existing ones, and delete tags that are no longer needed. This ensures that the tagging system remains clean and relevant. The TagController interacts with the MetadataService to update the tags in the Firebase Realtime Database, maintaining the integrity and accuracy of the metadata associated with images.

Responsibilities:

* Enables admin to create new tags (createTag())
* Allows admin to edit existing tags (editTag())
* Manages tag deletion (deleteTag())

Interactions:

* Works with MetadataService for tag operations.
* Communicates with AdminController to ensure tag management is restricted to admins.
* Viewing and Interaction Component

### **6.2.4. Viewing and interaction component**

**GalleryController:**

The GalleryController provides the functionality for displaying images in a gallery format. It retrieves images from the ImageService and their associated metadata from the MetadataService, presenting them in a visually appealing and user-friendly manner. The controller supports various viewing modes, such as grid and list views, and allows users to view detailed information about individual images, including descriptions, tags, and user interactions like likes and shares.

Responsibilities:

* Displays images in a gallery format (viewGallery())
* Provides detailed views of individual images (viewImageDetails())

Interactions:

* Communicates with ImageService to retrieve images.
* Interacts with MetadataService to display metadata alongside images.

**InteractionController:**

The InteractionController manages user interactions with images, including liking, sharing, and downloading images. When a user likes an image, the InteractionController records the like in the Firebase Realtime Database through the UserService, ensuring that the user's interactions are tracked and can be used for recommendations. Sharing functionality involves generating shareable links and tracking shares, while downloading functionality ensures that users can securely download images to their devices.

Responsibilities:

* Allows users to like images (likeImage())
* Enables image sharing functionality (shareImage())
* Provides image download capability (downloadImage())

Interactions:

* Communicates with ImageService for retrieving images.
* Interacts with UserService to track user interactions (likes, shares).

### **6.2.5. Search and recommendation component**

**SearchController:**

The SearchController handles the search functionality within the app. It allows users to search for images based on keywords and tags. The controller interacts with the MetadataService to perform searches and retrieve relevant images, ensuring that the search results are accurate and quickly retrievable. The search functionality is optimized to handle large datasets efficiently, providing a seamless user experience.

Responsibilities:

* Manages search functionality to find images based on keywords and tags (searchImages())

Interactions:

* Communicates with MetadataService to perform searches based on tags and descriptions.
* Interacts with GalleryController to display search results.

**RecommendationController:**

The RecommendationController provides personalized image recommendations to users. It leverages machine learning techniques, such as collaborative filtering and content-based filtering, to generate recommendations based on user preferences, interaction history, and metadata. The controller interacts with the UserService to retrieve user data and with the RecommendationService to generate and provide recommendations, enhancing the overall user experience by suggesting relevant and interesting content.

Responsibilities:

* Provides personalized image recommendations (getRecommendations())

Interactions:

* Communicates with UserService to retrieve user preferences and interaction history.
* Works with RecommendationService to generate and provide recommendations.

### **6.2.6. Security component**

The AuthService ensures secure user authentication by encrypting and decrypting sensitive data using the SecurityService. This collaboration ensures that user credentials and other sensitive information are protected during transmission and storage.

**AuthService:**

The AuthService is responsible for managing user authentication and authorization. It handles user login and signup processes, interacting with Firebase Authentication to validate credentials and manage user sessions. The AuthService ensures that only authenticated users can access certain functionalities and that sensitive data is securely transmitted and stored.

Responsibilities:

* Manages user authentication (authenticate())
* Handles user authorization (authorize())

Interactions:

* Interfaces with Firebase Authentication for user login and signup processes.
* Communicates with UserController and AdminController for authentication purposes.

**SecurityService:**

The SecurityService provides additional security functionalities, such as data encryption and decryption. It ensures that sensitive user data, including images and metadata, is securely stored and transmitted, protecting user privacy and preventing unauthorized access. The SecurityService works in conjunction with the AuthService to provide a secure environment for all user interactions.

Responsibilities:

* Encrypts sensitive data (encryptData())
* Decrypts data for secure communication and storage (decryptData())

Interactions:

* Used by AuthService for securing user credentials.
* Utilized by ImageService and MetadataService for securing image and metadata data.

# 

# **7. Implementation view**

The Implementation View provides a detailed blueprint of how the system's functionality is implemented in code. It describes the organization of the source code, the structure of the main modules, layers, and packages, and outlines key implementation considerations.

## **7.1. Code Organization and Structure**

| /controllers  user\_controller.dart  admin\_controller.dart  image\_controller.dart  metadata\_controller.dart  tag\_controller.dart  gallery\_controller.dart  interaction\_controller.dart  search\_controller.dart  recommendation\_controller.dart    /services  auth\_service.dart  security\_service.dart  image\_service.dart  metadata\_service.dart  user\_service.dart  recommendation\_service.dart    /models  user.dart  image.dart  metadata.dart  tag.dart  interaction.dart  recommendation.dart    /repositories  user\_repository.dart  image\_repository.dart  metadata\_repository.dart  tag\_repository.dart | /views  login\_view.dart  signup\_view.dart  profile\_view.dart  gallery\_view.dart  image\_view.dart  search\_view.dart  recommendation\_view.dart    /widgets  custom\_widgets.dart    /config  firebase\_config.dart  security\_config.dart    main.dart |
| --- | --- |

*Table 39: Code organiztion and Structure*

## **7.2. User Management Module**

This module is responsible for handling all user-related operations, including user registration, authentication, profile management, and administrative tasks. It ensures that user data is securely managed and that only authorised users can access certain functionalities.

### **7.2.1. UserController**

The UserController is a central component responsible for managing user interactions with the app. It handles operations such as user registration, login, logout, and profile updates.

When a user attempts to sign up, the UserController collects the user's details, validates the input, and communicates with the AuthService to create a new user account in Firebase Authentication. Similarly, for login operations, it validates credentials and initiates a session using Firebase Authentication.

### **7.2.2. AdminController**

The AdminController handles administrative tasks that are exclusive to users with admin privileges. This includes creating, editing, and deleting user accounts.

The controller ensures that only authorized admin users can perform these operations by checking their permissions with the AuthService.

The AdminController also interfaces with the UserService to carry out the necessary database operations for managing user accounts.

## **7.3. Authentication Module**

This module ensures secure user authentication and authorization processes. It handles user sign-up, login, session management, and authorization checks.

### **7.3.1. AuthService**

The AuthService is responsible for managing authentication processes. It provides methods for signing up new users, logging in existing users, and logging out users. The service interacts directly with Firebase Authentication to handle these operations.

Additionally, it includes methods to authorize user actions by checking their roles and permissions. The AuthService also leverages the SecurityService to encrypt and decrypt sensitive data during authentication processes.

### **7.3.2. SecurityService**

The SecurityService ensures that all sensitive data, such as user credentials, are securely stored and transmitted. It provides encryption and decryption functionalities to protect data at rest and in transit.

This service works in conjunction with the AuthService to secure user authentication processes and ensure that user data remains confidential.

## **7.4. Image Management Module**

This module handles all operations related to image uploading, processing, storing, and retrieving. It ensures that images are efficiently managed and presented to users.

### **7.4.1. ImageController**

The ImageController manages user requests related to images, such as uploading new images, editing existing images, and deleting images.

When a user uploads an image, the controller handles the file upload, processes the image using the ImageService, and ensures that the image metadata is correctly updated through the MetadataController.

The ImageController also manages image editing operations, allowing users to crop or rotate images before finalizing the upload.

### **7.4.2. ImageService**

The ImageService is tasked with processing images and managing their storage. It includes functionalities for image compression, resizing, cropping, and storing images in Firebase Storage. The service ensures that images are efficiently processed to enhance the user experience, even on slower network connections.

It also retrieves images from Firebase Storage when requested by other components of the app.

## **7.5. Metadata Management Module**

This module is responsible for managing image metadata, including descriptions, tags, and other relevant information. It ensures that metadata is accurately associated with images and can be efficiently retrieved.

### **7.5.1. MetadataController**

The MetadataController handles operations related to image metadata, such as adding descriptions and tags to images. It provides methods for adding, editing, and retrieving metadata.

The controller interacts with the MetadataService to perform these operations and ensures that metadata is correctly associated with the corresponding images in the database.

### **7.5.2. TagController**

The TagController manages the creation, editing, and deletion of tags used for categorizing images. This controller is essential for maintaining an organized and searchable image library. It ensures that only authorized admin users can manage tags, verifying permissions with the AuthService.

The TagController interacts with the MetadataService to update tags in the Firebase Realtime Database.

### **7.5.3. MetadataService**

The MetadataService implements the business logic for managing image metadata. It provides functionalities for storing, updating, and retrieving metadata from Firebase Realtime Database. The service ensures that metadata operations are performed efficiently and that the data is accurately linked to the corresponding images.

## **7.6. Viewing and Interaction Module**

This module handles the display of images and user interactions with those images, such as liking, sharing, and downloading.

### **7.6.1. GalleryController**

The GalleryController manages the presentation of images in a gallery format. It retrieves images from the ImageService and their associated metadata from the MetadataService, presenting them in a user-friendly manner.

The controller supports various viewing modes, such as grid and list views, and provides detailed views of individual images. It ensures that images are displayed quickly and efficiently to enhance the user experience.

### **7.6.2. InteractionController**

The InteractionController handles user interactions with images, including liking, sharing, and downloading images. It tracks user interactions by communicating with the UserService, ensuring that user activity is logged and can be used for generating personalized recommendations.

The controller also ensures that user interactions are reflected in real-time, providing immediate feedback to users.

## **7.7. Search and Recommendation Module**

This module provides functionalities for searching images and generating personalized recommendations based on user preferences and behavior.

### **7.7.1. SearchController**

The SearchController manages the search functionality of the app. It allows users to search for images based on keywords and tags.

The controller interacts with the MetadataService to perform searches and retrieve relevant images. It ensures that search results are accurate and quickly retrievable, enhancing the user experience.

### **7.7.2. RecommendationController**

The RecommendationController generates personalized recommendations for users based on their preferences and interaction history. It uses machine learning techniques to analyze user data and generate tailored recommendations.

The controller communicates with the UserService to retrieve user data and with the RecommendationService to apply recommendation algorithms. This module enhances the user experience by suggesting relevant content, keeping users engaged with the app.

## **7.8. Core Services Module**

This module includes core services that support the overall functionality of the app, such as user management and recommendation generation.

### **7.8.1. UserService**

The UserService implements the core business logic for managing users. It provides methods for creating and updating user profiles, tracking user interactions, and retrieving user data from Firebase Realtime Database. The service ensures that user-related operations are performed efficiently and that user data is securely managed.

### **7.8.2. RecommendationService**

The RecommendationService implements algorithms for generating personalized recommendations. It leverages techniques like collaborative filtering and content-based filtering to analyze user data and interaction history.

The service interacts with the Firebase Realtime Database to fetch user data and generate recommendations, ensuring that users receive relevant and engaging content.

## **7.9. Application Dependencies**

In the Flutter application, Firebase services are integrated using the firebase\_core package for initializing Firebase, along with specific packages such as firebase\_auth for authentication, cloud\_firestore for database operations, and firebase\_storage for image storage.

## **7.10. Data Backup Procedure**

### **7.10.1. Types of Backups**

Incremental Backups: capture only the changes made since the last backup. These are performed frequently (e.g., every hour) to ensure minimal data loss and quick recovery.

Full Backups: backups capture all data and are performed less frequently (e.g., weekly or monthly). They provide a comprehensive restore point that can be used to fully restore the system.

On-Demand Backups: can be triggered manually at any time, typically before major updates or changes to the system, to ensure a safe rollback point.

### **7.10.2. Backup Procedure**

#### **Incremental Backups:**

Frequency: Every hour

Data Included: Changes in user information, new or updated images, metadata, and interactions

Procedure:

* Firebase Realtime Database:
  + Utilize Firebase’s built-in export functionality to create an export of only the changed data since the last backup.
  + Store these exports in a secure Cloud Storage bucket.
* Firebase Storage:
  + Track changes to images and create snapshots of newly uploaded or modified images.
  + Store the snapshots in a dedicated backup directory within Firebase Storage.

#### **Full Backups:**

Frequency: Weekly or monthly

Data Included: Entire database and all stored images

Procedure:

* Firebase Database:
  + Perform a full export of the Firebase Realtime Database.
  + Store the export files in a secure Cloud Storage bucket with appropriate naming conventions (full\_backup\_YYYYMMDD).
* Firebase Storage:
  + Copy all image files to a dedicated backup directory within Firebase Storage.
  + Ensure the backup directory is organized and named appropriately ( full\_backup\_images\_YYYYMMDD).

#### **On-Demand Backups:**

Frequency: As needed

Data Included: Entire database and all stored images at the time of the backup

Procedure:

* Firebase Realtime Database:
  + Trigger a manual export of the database using Firebase’s export functionality.
  + Store the export files in a secure Cloud Storage bucket with a time stamped directory name.
* Firebase Storage:
  + Create a snapshot of all image files and store them in a dedicated, timestamped backup directory within Firebase Storage.

### **7.10.3. Restore Procedure**

#### **Identify Backup:**

Determine the most recent and relevant backup (incremental or full) based on the data loss incident.

Retrieve the backup files from the secure Cloud Storage bucket.

Restore Firebase Realtime Database:

* Use Firebase’s import functionality to restore the database from the backup files.
* Ensure the import process does not overwrite any new changes made since the backup.

Restore Firebase Storage:

* Identify the images that need to be restored based on the backup type (incremental or full).
* Use Firebase Storage’s restore functionality to copy the backup images back to their original locations.

Validation:

* Verify the integrity and completeness of the restored data.
* Ensure all user information, images, and metadata are correctly restored and accessible.

Post-Restoration Testing:

* Perform comprehensive testing of the application to ensure all functionalities are working correctly post-restoration.
* Address any discrepancies or issues identified during testing.

### **7.10.4. Security Measures**

Encryption: Ensure all backup files are encrypted both in transit and at rest to protect sensitive data.

Access Control: Restrict access to backup files to authorized personnel only. Use IAM policies to enforce access controls.

Audit Logs: Maintain audit logs of all backup and restore operations to track activities and ensure accountability.

### **7.10.5. Automation and Monitoring**

Automated Scripts: Implement automated scripts to perform regular incremental and full backups as per the defined schedule.

Monitoring: Set up monitoring and alerts to notify administrators of backup successes or failures. Use Firebase’s monitoring tools and integrate with third-party monitoring services if necessary.

# 

# **8. Data view**

The Data View provides a detailed understanding of how data is structured and organized within Firebase Realtime Database and Firebase Storage for your Flutter-based image-sharing app. By delineating the database schema and relationships between data entities, developers can efficiently interact with and manage the app's data, ensuring smooth functionality and optimal performance.

## **8.1. Firebase Realtime Database Structure**

The Firebase Realtime Database is used to store structured data, including user information, image metadata, and interaction logs. Here's an overview of the database structure:

### **8.1.1. Users Collection**

Each user is represented by a unique user ID (uid).

User data includes:

* Email
* Display name
* Role (admin or regular user)
* Profile picture URL
* Timestamp of account creation

| /users  /{uid}  - email  - displayName  - role  - profilePictureUrl  - createdAt |
| --- |

*Table 40: Users collection*

### **8.1.2. Images Collection**

Each uploaded image is represented by a unique image ID.

Image data includes:

* URL of the image stored in Firebase Storage
* Description
* Tags (stored as an array)
* Uploader user ID
* Timestamp of upload

| /images  /{imageId}  - imageUrl  - description  - tags: ["tag1", "tag2", ...]  - uploaderId  - uploadTimestamp |
| --- |

*Table 41: Images collection*

### **8.1.3. Tags Collection**

Each tag is represented by a unique image ID.

Image data includes:

* Name
* Description
* Illustration picture URL
* The number of images using this tag
* Timestamp of tag creation

| /tags  /{tagId}  - name  - description  - illustration  - imageCount  - createdAt |
| --- |

*Table 42: Tags collection*

### **8.1.4. Interaction Logs**

Logs interactions such as likes, shares, and downloads.

Each interaction is stored as a separate entry with a unique ID.

Interaction data includes:

* Type of interaction (like, share, download)
* ID of the user performing the interaction
* ID of the image being interacted with
* Timestamp of the interaction

| /interactions  /{interactionId}  - type  - userId  - imageId  - timestamp |
| --- |

*Table 43: Interaction logs*

## **8.2. Firebase Storage Structure**

Firebase Storage is used to store the actual image files uploaded by users. The structure is relatively straightforward:

### **8.2.1. Images Directory**

All uploaded images are stored in a dedicated directory within Firebase Storage.

| /images  /{imageFileName} |
| --- |

*Table 44: Images directory*

### **8.2.2. User Profile Images Directory**

All uploaded images are stored in a dedicated directory within Firebase Storage.

| /userProfiles  /{imageFileName} |
| --- |

*Table 45: User profile images directory*

## **8.3. Relationships Between Data Entities**

### **8.3.1. User-Image Relationship**

Each image in the /images collection is associated with the user who uploaded it through the uploaderId field.

This relationship allows for retrieving all images uploaded by a specific user.

### **8.3.2. Interaction Logs and Image/User Relationship**

Interaction logs in the /interactions collection maintain relationships with both users and images.

The userId field identifies the user performing the interaction, while the imageId field identifies the image being interacted with.

This relationship enables tracking user interactions with specific images and analyzing user engagement patterns.

### **8.3.3. Tag-Image Relationship**

Tags are associated with images through the tags array field in each image document.

This relationship allows for efficient retrieval of images tagged with a specific tag.

### **8.3.4. Image-Tag Relationship (Inverse)**

Conversely, each tag document maintains a count of how many images are tagged with it.

This relationship enables tracking the popularity and usage frequency of each tag.

# 

# **9. Deployment view**

## **9.1. Google Play Store Deployment**

Build Preparation

* Generate a release build of the Flutter app using the flutter build command with appropriate configurations for Android.
* Ensure compatibility with various Android devices and screen sizes.

Google Play Console:

* Create or log in to a developer account on the Google Play Console.
* Navigate to the "Create app" section and fill in essential metadata for the app listing.

App Signing:

* Opt-in to Google Play App Signing to ensure secure distribution.
* Upload the app's release signing key or allow Google Play to generate one.

App Release:

* Upload the release build bundle (.aab file) to the Google Play Console.
* Configure release tracks (e.g., production, beta, alpha) and rollout percentages for staged rollouts.

App Review:

* Submit the app for review by Google Play's team, ensuring compliance with policies and guidelines.
* Address any issues or feedback provided by the review team promptly.

App Publication:

* Once approved, publish the app to the Google Play Store for public availability.
* Monitor user feedback and reviews to address any issues post-launch.

## **9.2. Apple AppStore Deployment**

Build Preparation:

* Generate a release build of the Flutter app using the flutter build command with appropriate configurations for iOS.
* Ensure compatibility with various iOS devices and screen resolutions.

Apple Developer Account:

* Create or log in to a developer account on the Apple Developer website.

App Identifier and Provisioning Profiles:

* Generate an App ID and provisioning profiles for the app in the Apple Developer portal.
* Configure app capabilities and entitlements as needed.

Code Signing:

* Set up code signing identities and provisioning profiles in Xcode for code verification and app distribution.

App Store Connect:

* Log in to App Store Connect and create a new app listing for the Flutter app.
* Fill in metadata such as the app name, description, screenshots, and promotional images.

App Review:

* Submit the app for review by Apple's App Store Review team, ensuring compliance with App Store guidelines and policies.
* Address any issues or feedback provided by the review team promptly.

App Publication:

* Once approved, release the app to the Apple App Store for public availability.
* Monitor user feedback and reviews to address any issues post-launch.

## **9.3. Continuous Deployment**

Automation Tools:

* Set up automation tools such as Fastlane or App Store Connect API to automate the deployment process.
* Automate tasks like building, testing, and releasing app updates to both app stores.

CI/CD Integration:

* Integrate the deployment pipeline with continuous integration and continuous deployment (CI/CD) systems to trigger builds and releases automatically.

## **9.4. Monitoring and Maintenance**

App Monitoring:

* Implement monitoring solutions to track app performance, stability, and user engagement across both platforms.
* Use tools like Firebase Performance Monitoring, Google Analytics, or Apple App Analytics to gather insights and metrics.

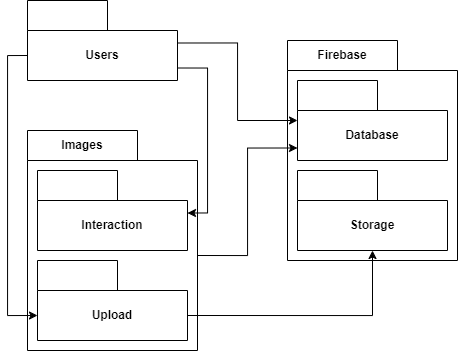
Bug Fixes and Updates:

* Regularly monitor user feedback and app analytics to identify and prioritize bug fixes and feature enhancements.
* Release updates to the app as needed, addressing user-reported issues and adding new features.

# 

# **10. System definition**

## **10.1. Functional Requirements**



*Image 21: Functional Overview*

**User**: Allows users to sign up, log in, and manage their accounts securely.

* **Sign Up**: Enables new users to create accounts.
  + - Attribute*:* 
      * *Memory Management***:** Optimize memory usage to handle new user data and profile images.
      * *Resisting Attacks***:** Ensure secure sign-up process to prevent automated bot sign-ups and other malicious activities.
* **Log In**: Allows existing users to authenticate and access the app.
  + - Attribute:
      * *Memory Management:* Optimize memory usage for handling login data and sessions.
      * *Resisting Attacks*: Implement login options using multiple methods such as Gmail, Facebook, and Google to resist unauthorized access.
      * *Detecting Attacks*: Logging all the login sessions for detecting force attacks or unusual login attempts.
      * *Recovering from Attacks*: Provide account recovery options and support users to get their accounts back.
* **Account Management**: Provides options to update profile information and manage account settings.
  + - Attribute:
      * *Image Loading Time*: Ensure quick loading of profile images and other media within account settings.
      * *Memory Management:* Efficiently handle updates to user data without excessive memory consumption.
      * *Resisting Attacks:* Secure authorization to prevent unauthorized changes to account information.
      * *Recovering from Attacks:* Have measures to revert any unauthorized changes and notify users of suspicious activity.

**Images Upload**: Allows users to upload images to the app.

* **Upload Interface**: Provides a user-friendly interface for selecting and uploading images.
  + - Attribute:
      * *Image Loading Time*: Ensure the interface loads quickly and allows users to preview images without delay.
      * *Memory Management*: Optimize memory usage to handle large image files efficiently during the selection and upload process.
      * *Resisting Attacks*: Admin manage of images uploaded to the app to prevent malicious files from being uploaded.
      * *Recovering from Attacks*: Have measures to remove malicious uploads and restore the system to a secure state.
* **Image Processing**: Performs image processing tasks such as cropping and rotating before uploading.
  + - Attribute:
      * *Image Loading Time*: Quickly process and preview changes like cropping or rotating images before uploading.
      * *Memory Management:* Efficiently handle image processing tasks without consuming excessive memory.
* **Metadata Input**: Allows users to provide image descriptions, tags, and other relevant information.
  + - Attribute:
      * *Memory Management:* Handle large amounts of metadata efficiently without degrading performance.
      * *Resisting Attacks*: Admin manage of tags created in the app to prevent sensitive contents from being created.
      * *Recovering from Attacks*: Have measures to remove sensitive contents and restore the system to a secure state.

**Images Interaction**: Enables users to interact with images by liking, sharing, and downloading.

* **Like**: Allows users to express appreciation for images by liking them.
  + - Attribute:
      * *Image Loading Time***:** Ensure that liking an image updates quickly without lag, providing immediate feedback to the user.
      * *Memory Management:* Efficiently handle the data of likes to ensure the app performs well even with many likes.
* **Share**: Facilitates sharing images with other users or external platforms.
  + - Attribute:
      * *Image Loading Time***:** Ensure the sharing interface loads quickly, and images are promptly prepared for sharing.
      * *Memory Management***:** Optimize memory usage when preparing images for sharing, especially for large files.
      * *Resisting Attacks***:** Secure the sharing feature to prevent unauthorized sharing of private images.
* **Download**: Enables users to download images to their devices for offline viewing.
  + - Attribute:
      * *Image Loading Time***:** Ensure images are downloaded quickly and efficiently for offline viewing.
      * *Memory Management:* Efficiently handle the data transfer and storage for downloaded images to avoid excessive memory usage.

## 

## **10.2. Design Constraints**

Three design constraints are required:

1. **Platform compatibility**: The app must be compatible with multiple mobile platforms, such as iOS, Android and website, requiring the use of cross-platform development frameworks or separate codebases for each platform.
2. **Device compatibility:** Optimizing the app's UI layout, font sizes, and image resolutions to accommodate various screen sizes, aspect ratios, and pixel densities across different mobile devices, ensuring a consistent and visually appealing user experience on devices ranging from smartphones to tablets.
3. **Scalability requirements**: The app must be able to handle a potentially large user base and increasing amounts of image data, requiring scalable architecture and cloud-based infrastructure to support growth and accommodate spikes in traffic.

## **10.3. Quality Attribute Requirements**

### **10.3.1. Performance**

**Table 1: Quality Attribute Scenario 1: Image Loading Time**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | User opens the image sharing mobile app and navigates to view images |
| Stimulus Source | User interaction with the app triggers the need to load images |
| Environment | * Network Connection: Speed of the network connection (Wi-Fi, 4G/5G, slower networks). * Device Processing Power: Processing power of the device (high-end, mid-range, older smartphones). * App State: Whether the app is running in the foreground or background, number of other apps running simultaneously. * Image Source: Whether the images are loaded from the cache or the server |
| Artifact | Image loading and rendering mechanism within the mobile app |
| Response | 1. The app starts loading images immediately after the user navigates to the image viewing page. 2. Utilize caching mechanisms to preload images if possible. 3. Display thumbnail images before loading full-resolution images. 4. Apply optimizations such as image compression and progressive loading (load low-resolution images first, then gradually increase resolution if needed). 5. Indicate the loading status to the user using loading icons or placeholder images if necessary. |
| Response measure | * Objective: Load images within a specified time threshold (e.g., 1 second for thumbnail, 3 seconds for full-size images). * Subjective: Users perceive images as loading almost instantly, providing a seamless browsing experience. |

*Table 46: Image Loading Time Scenario*

**Table 2: Quality Attribute Scenario 2: Memory Management**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | Continuous usage of the app leads to the allocation and deallocation of memory resources. |
| Stimulus Source | App's internal processes and user interactions trigger memory allocation and deallocation. |
| Environment | * Available RAM: The amount of available RAM on different devices (high-end, mid-range, and older devices). * App State: Whether the app is running in the foreground or background, and the number of other apps running simultaneously. |
| Artifact | Memory allocation and deallocation mechanisms within the app's codebase. |
| Response | 1. The app continuously monitors and efficiently manages memory allocation and deallocation. 2. Implement automatic garbage collection to free up unused memory. 3. Optimize the codebase to minimize memory leaks. 4. Use techniques such as caching and data compression to manage memory more effectively. 5. Monitor memory status and handle memory-related issues promptly to prevent performance degradation. |
| Response measure | * Objective: Keep memory usage within predefined limits (e.g., X MB of RAM). * Subjective: Users do not experience any bad performance or crashes due to memory issues during normal usage. |

*Table 47: Memory Management Scenario*

### **10.3.2. Security**

**Table 3: Quality Attribute Scenario 3: Resisting Attacks**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | User interacts with the app, which involves the exchange of sensitive data such as images and personal information. |
| Stimulus Source | External entities attempting to gain unauthorized access to user data or compromise system integrity. |
| Environment | * Network Connectivity: Varies from secure Wi-Fi to potentially insecure public networks. * Threat Landscape: Includes various potential threats such as hackers, malicious software, and other security vulnerabilities. |
| Artifact | * Authentication: Ensuring only authorized users can access the app. * Authorization: Controlling what authenticated users can access and modify. * Secure Communication Protocols: Ensuring data integrity and confidentiality during exchanges. |
| Response | 1. Implement login options using multiple methods such as Gmail, Facebook, and Google to resist unauthorized access. 2. Enforce strict authorization rules to control access to different parts of the app. 3. Regularly update and patch security vulnerabilities. 4. Educate users on best practices for maintaining account security. 5. Admin manage of tags created in the app to prevent sensitive contents from being created. 6. Admin manage of images uploaded to the app to prevent malicious files from being uploaded. 7. Ensure secure sign-up process to prevent automated bot sign-ups and other malicious activities. 8. Secure authorization to prevent unauthorized changes to account information. 9. Secure the sharing feature to prevent unauthorized sharing of private images. |
| Response measure | * Objective: No successful unauthorized access or data breaches occur. * Subjective: Users perceive the app as secure and trustworthy, leading to increased confidence in sharing personal information and images. |

*Table 48: Registering Attacks Scenario*

**Table 4: Quality Attribute Scenario 4: Detecting Attacks**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | 1. An external attacker attempts to log into the app. 2. An external attacker attempts to attack network traffic between clients and Firebase servers. |
| Stimulus Source | Potential attackers attempting to exploit vulnerabilities in the system. |
| Environment | * Network Environment**:** Active network environment with potential threats such as intrusion attempts or abnormal traffic patterns. * Operational Conditions: Varying network load, different user activities, and diverse device types. |
| Artifact | 1. Logging mechanisms for all login attempts. 2. Network Intrusion Detection System (NIDS) integrated into the app's infrastructure |
| Response | 1. Logging: Record detailed logs of all login attempts, including successful and failed attempts. 2. Alert Mechanisms: Generate alerts for unusual login patterns or multiple failed attempts. 3. Analysis: Analyze logs to identify potential unauthorized access attempts. 4. Real-Time Monitoring: The NIDS continuously monitors network traffic between clients and Firebase servers. |
| Response measure | * Objective:   + NIDS successfully identifies and alerts about potential attacks or unusual network behavior.   + Ensure all login attempts are logged and analyzed for potential threats. * Subjective: Users are unaware of potential threats due to effective behind-the-scenes detection mechanisms. |

*Table 49: Detecting Attacks Scenario*

**Table 5: Quality Attribute Scenario 5: Recovering from Attacks**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | Detection of a successful attack or compromise of system integrity. |
| Stimulus Source | Previous attack events or system vulnerabilities exploited by attackers. |
| Environment | Post-attack scenario requiring restoration of system integrity and data. |
| Artifact | Backup and recovery mechanisms implemented within the app's infrastructure. |
| Response | 1. Detection: Identify the nature and extent of the attack or compromise. 2. Isolation: Quarantine affected systems or components to prevent further damage. 3. Data Recovery: Restore data from backup sources to a pre-attack state. 4. System Restoration: Rebuild or restore affected systems to their original configuration. 5. Security Updates: Patch any vulnerabilities exploited by the attackers to prevent future attacks. 6. Have measures to remove sensitive contents, malicious uploads and restore the system to a secure state. 7. Have measures to revert any unauthorized changes and notify users of suspicious activity. |
| Response measure | * Objective: Data recovery and system restoration processes are completed within an acceptable time frame. * Subjective: Users experience minimal disruption and perceive the app as resilient to attacks, with minimal data loss or downtime. |

*Table 50: Recovering from Attacks Scenario*

### 

### **10.3.3. Scalability**

**Table 6: Quality Attribute Scenario 6: Database Scalability**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | Increase in the number of users, images, and interactions within the image sharing mobile app. |
| Stimulus Source | Growth in user base and content uploaded to the platform. |
| Environment | Varies based on user activity and demand, with potential spikes in usage during peak hours or viral content sharing. |
| Artifact | Database infrastructure, including storage, querying, and data retrieval mechanisms. |
| Response | 1. Automatic Scaling: Using Firestore and Realtime Database, which are designed to automatically scale to handle growing data and user loads. 2. Load Balancing: Distribute incoming database queries evenly across multiple servers to prevent overload on any single server. 3. Database Optimization: Optimize database schema, indexing, and query performance to improve efficiency. 4. Caching: Implement caching mechanisms to reduce the need for repeated database queries. 5. Data Partitioning: Partition large datasets across multiple servers to improve performance and scalability. |
| Response measure | * Objective: Database operations such as reads and writes remain within acceptable latency thresholds, even under high load conditions. * Subjective: Users experience consistent app performance and responsiveness, regardless of the number of concurrent users or volume of uploaded images. |

*Table 51: Database Scalability Scenario*

### **10.3.4. Usability**

**Table 7: Quality Attribute Scenario 7: Search Algorithms**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | User initiates a search query within the image sharing mobile app. |
| Stimulus Source | User interaction triggers the need for retrieving relevant search results. |
| Environment | Varies based on the size of the image database and complexity of search queries. |
| Artifact | Search functionality and algorithms implemented within the app. |
| Response | 1. Efficient Indexing: Maintain optimized indexes of image metadata to facilitate fast search operations. 2. Query Optimization: Implement efficient search algorithms to quickly retrieve relevant results. 3. Real-Time Updates: Update search indexes in real-time as new images are added or existing images are modified. 4. Relevance Ranking: Rank search results based on relevance to the user's query, considering factors such as image metadata, user preferences. 5. User Feedback Integration: Incorporate user feedback to continuously improve search relevance and performance. |
| Response measure | * Objective: Search queries are executed within a specified time frame (e.g., milliseconds) and return relevant results. * Subjective: Users find search results satisfactory and perceive the search feature as efficient and user-friendly. |

*Table 52: Search Algorithm Scenario*

**Table 8: Quality Attribute Scenario 8: Personalized Recommendations**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | Users interact with the app, browsing images and engaging with content, drive the need for personalized content recommendations. |
| Stimulus Source | User behavior and preferences. |
| Environment | Dynamic user interactions and content updates influence recommendation generation. |
| Artifact | Recommendation engine and algorithms integrated into the app's functionality. |
| Response | App provides personalized recommendations tailored to each user's interests and preferences by:   1. User Profiling: Collect and analyze user behavior data (e.g., image views, likes, shares) to create user profiles. 2. Content Analysis: Analyze image metadata and content characteristics to identify patterns and similarities. 3. Recommendation Generation: Utilize machine learning algorithms to generate personalized recommendations. 4. Feedback Loop: Incorporate user feedback to refine recommendation algorithms and improve accuracy over time. |
| Response measure | * Objective: Recommendation algorithms accurately predict user preferences and suggest relevant content. * Subjective: Users perceive recommendations as helpful and valuable, leading to increased engagement and satisfaction. |

*Table 53: Personalized Recommendations Scenario*

### **10.3.5. Maintainability**

**Table 9: Quality Attribute Scenario 9: Modular Code Structure**

| **Element** | **Statement** |
| --- | --- |
| Stimulus | Development team works on enhancing, maintaining, or extending the image sharing mobile app. |
| Stimulus Source | Development activities, including coding, debugging, testing, and code reviews. |
| Environment | Development environment where codebase evolution and maintenance tasks occur. |
| Artifact | Codebase structure and organization, including modules, classes, and components. |
| Response | Codebase is organized into cohesive modules with clear boundaries, facilitating easier maintenance, scalability, and extensibility by:   1. Modularization: Organize codebase into cohesive modules with clear boundaries, each responsible for specific functionality or feature set. 2. Decoupling: Minimize dependencies between modules to improve code maintainability, scalability, and extensibility. 3. Abstraction: Abstract common functionality into reusable components or libraries to avoid code duplication and promote code reuse. 4. Encapsulation: Encapsulate module internals to hide implementation details and reduce the risk of unintended side effects. 5. Naming Conventions: Follow consistent naming conventions and directory structures to facilitate easy navigation and understanding of the codebase. 6. Documentation: Document module interfaces, dependencies, and usage guidelines to assist developers in utilizing and extending existing code. |
| Response measure | * Objective: Codebase adheres to modular design principles, with components logically grouped and decoupled. * Subjective: Development team finds it easy to navigate and understand the codebase, leading to faster development cycles and reduced time for bug fixes or feature enhancements. |

*Table 54: Modular Code Structure*

## 