Design Document for Image Processing System

Web Application

Version 1.0 approved

Prepared by

Group 2

Vũ Trọng Dũng

Bùi Minh Sơn

Nguyễn Việt Hoàng

Trần Quang Huy

Hồ Vũ Tuấn Minh

Lecturer

Mr. Trần Bình Dương

Process Impact

February 6 , 2025

2. Revision History

Name	Date	Reason For Changes	Version
Vũ Trọng Dũng	25/02/2025	4.3 (UC-10 ; UC-11 ; UC-12 ; UC-13) 4.5 4.6 4.7 5.2.2.1 ; 5.2.2.2 ; 5.2.2.3 ; 5.2.2.4 6.4.6 ; 6.4.7 ; 6.4.8 ; 6.4.9 ; 6.4.10 ; 6.4.11 6.5	1.0
Nguyễn Việt Hoàng	29/02/2025	4.3 (UC-20) 5.2.1.7 6.1 6.4.1	1.0
Trần Quang Huy	05/03/2025	4.3 (UC-8 ; UC-9 ; UC-21 ; UC-22) 5.1.2 5.2.1.8 ; 5.2.1.9 ; 5.2.1.10 ; 5.2.1.11 6.2.3 6.3	1.0
Hồ Vũ Tuấn Minh	06/03/2025	4.2 4.3 (UC-14 ; UC-15 ; UC-16 ; UC-17 ; UC-18 ; UC-19) 5.2.1.1 ; 5.2.1.2 ; 5.2.1.3 ; 5.2.1.4 ; 5.2.1.5 ; 5.2.1.6 6.2.1 6.2.2 6.3	1.0
Bùi Minh Sơn	08/03/2025	4.1 4.3 (UC-1 ; UC-2 ; UC-3 ; UC-4 ; UC-5; UC-6 ; UC-7) 5.1.1 ; 5.1.3 ; 5.1.4 5.2.1 6.4	1.0

3. Table of Contents, Table of Figures , Table of Tables

2. Revision History 2
3. Table of Contents, Table of Figures , Table of Tables 3
4. Requirements Modeling 7
4.1. Business workflows 7
Figure 4.1.1. Business workflow of the Image format conversion process 7
Table 4.1.1. Description of the Image format conversion workflow 8
Figure 4.1.2. Business workflow of the Image splitting process 8
Table 4.1.2. Description of the Image splitting workflow 9
Figure 4.1.3. Business workflow of the Image merging process 9
Table 4.1.3. Description of the Image merging workflow 11
4.2. Use-case diagrams 11
Figure 4.2.1. Use-case diagram 11
4.3. Use-case description for each use-cases 12
4.3.1. Use Case List 12
Table 4.3.1. Use Case List 13
4.3.2. Use Case Description 13
UC-1: Convert image format 13
UC-2: Split an image 14
UC-3: Merge images 16
UC-4: View progress 17
UC-5: Upload image(s) 18
UC-6: Download image(s) 19
UC-7: Preview processed image(s) 20
UC-8: View profile 21
UC-9: Edit profile 23
UC-10: Sign in 25
UC-11: Sign up 27
UC-12: Sign out 28
UC-13: Reset password 29
UC-14: View all users 31
UC-15: View user details 33
UC-16: Create user 34
UC-17: Update user 35
UC-18: Disable user 37
UC-19: Delete user 39
UC-20: View logs 40
UC-21: View online file storage 42
UC-22: Delete image files 43

4.4. Activity Diagram 44
4.5. Non-functional Requirements 44
4.5.1. Usability 44
4.5.2. Security 45
4.5.3. Maintainability 45
4.5.4. Availability & Reliability 45
4.5.5. Data Storage & File Handling 45
4.5.6. Logging & Monitoring 45
4.6. Business Rules Table 46
Table 4.6.1. Business Rules Table 47
4.7. Concurrent Tasks Description 47
5. Analysis Modeling 48
5.1. Static Modeling Diagrams and Explanation 48
5.1.1. Contextual External Class Diagram 48
Figure 5.1.1.1. Contextual external class diagram 48
5.1.2. Business Processing Classes Hierarchical Structure 49
Figure 5.1.2.1. Business Processing Classes Hierarchical Structure 49
5.1.3. Entity Class Diagram 51
Figure 5.1.3.1. Entity class diagram 51
5.1.4. Classes which perform Concurrent Tasks responsibility 53
5.2. Dynamic Modeling Diagrams and Explanation 53
5.2.1. Communication Diagrams 53
5.2.1.1. Create User 53
Figure 5.2.1.1. Create User Communication diagram 54
5.2.1.2. Update User 55
Figure 5.2.1.2. Update User Communication diagram 55
5.2.1.3. Disable User 56
Figure 5.2.1.3. Disable User Communication diagram 56
5.2.1.4. Delete User 58
Figure 5.2.1.4. Delete User Communication diagram 58
5.2.1.5. View User Details 59
Figure 5.2.1.5. View User Communication diagram 59
5.2.1.6. View All Users 60
Figure 5.2.1.6. View All User Communication diagram 60
5.2.1.7. View Logs 61
Figure 5.2.1.7. View Logs Communication diagram 61
5.2.1.8. View Profile 62
Figure 5.2.1.8. View Profile Communication diagram 62
5.2.1.9. Edit Profile 63
Figure 5.2.1.9. Edit Profile Communication diagram 63
5.2.1.10. View online file storage 65

Figure 5.2.1.10. View Online File Storage Communication diagram....65
5.2.1.11. Delete Image File....66
Figure 5.2.1.11. Delete Image File Communication diagram....66
5.2.2. Integrated Communication Diagrams....67
5.2.2.1. Sign up....67
Figure 5.2.2.1. Sign Up Communication diagram....67
5.2.2.2. Reset password....69
Figure 5.2.2.2. Reset Password Communication diagram....69
5.2.2.3: Sign out....70
Figure 5.2.2.3. Sign Out Communication diagram....70
5.2.2.4: Sign In....71
Figure 5.2.2.4. Sign In Communication diagram....71
5.2.3. Concurrent Communication Diagrams....72
5.2.3.1. Convert image format....72
Figure 5.2.3.1. Communication diagram for use-case “Convert image format”....72
5.2.3.2. Split an image....73
Figure 5.2.3.2. Communication diagram for use-case “Split an image”....73
5.2.3.3. Merge images....75
Figure 5.2.3.3. Communication diagram for use-case “Merge images”....75
6. Design Modeling....76
6.1. List of Architecture Patterns which be selected....76
6.2. Architecture in three views....77
6.2.1. Architecture in Static View....77
Figure 6.2.1.1. Architecture in Static View....77
6.2.1.1. External Users & Systems....78
6.2.1.2. Image Processing System Components....78
6.2.1.3. Interaction & Communication....78
6.2.1.4. Internal Layered Architecture of the Software System....79
Figure 6.2.1.2. Internal layered architecture of the software system....79
6.2.2. Architecture in Dynamic View....81
Figure 6.2.2.1. Architecture in dynamic view....81
Figure 6.2.2.2. Dynamic view of the internal layered architecture....83
6.2.3. Architecture in Deployment View....84
Figure 6.2.3.1. Architecture in Deployment View....84
Figure 6.2.3.2. IMS Database Design....85
6.4. Detailed Specifications for each component and class....87
6.4.1. LogView....87
6.4.2. UserController....88
6.4.3. UserDao....89
6.4.4. UserListView....90
6.4.5. EmailService....90

6.4.6. AuthenticationView....................90
6.4.7. AuthenticationController....................92
6.4.8. AuthenticationService....................94
6.4.9. User....................95
6.4.10. OTPVerifyView....................97
6.4.11. RecoverPasswordView....................99
6.4.12. Image....................100
6.4.13. LogController....................101
6.4.14. Log....................102
6.4.15. ImageListView....................104
6.4.16. UserProfileView....................104
6.4. Concurrent Tasks/Classes Specifications....................104
6.4.1. ImageConverter....................104
6.4.2. ImageSplitter....................105
6.4.3. ImageMerger....................106
6.4.4. ProcessUpdatingSocket....................107
6.4.5. ImageProcessingController....................108
6.4.6. ImageListController....................109
6.4.7. LogDao....................110
6.4.8. ImageDao....................111
6.4.9. ImageProcessingView....................111
6.4.10. ImageDownloadingView....................112
6.5. List of Quality Attributes and Evaluation Results....................112
6.5.1. Usability....................113
6.5.2. Security....................113
6.5.3. Maintainability....................114
6.5.4. Availability & Reliability....................114
6.5.5. Data Storage & File Handling....................114
6.5.6. Logging & Monitoring....................115

4. Requirements Modeling

4.1. Business workflows

This Image Processing System has three main workflows: **Image Format Conversion**, **Image Splitting** and **Image Merging.**

Image Format Conversion

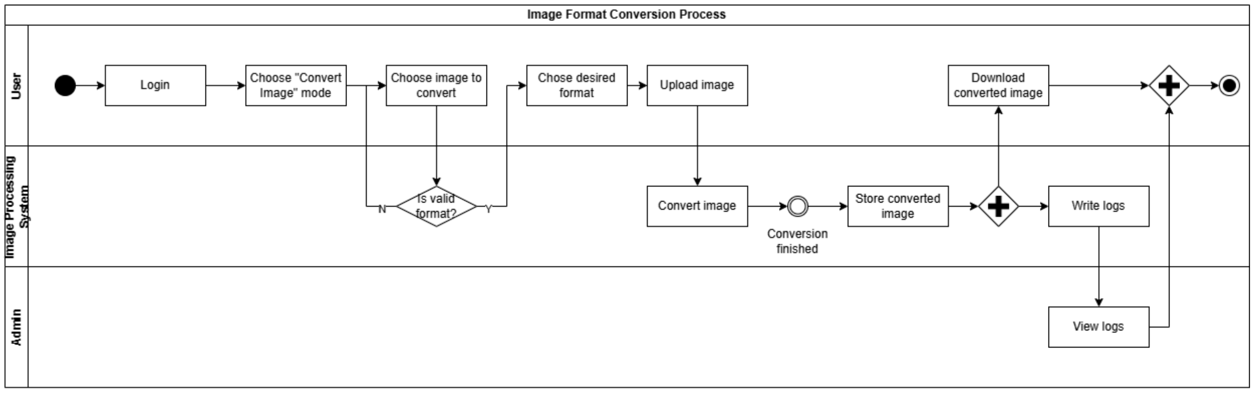


Figure 4.1.1. Business workflow of the Image format conversion process

Step	Step Name	Description
Step 1	Login	User logs into the system.
Step 2	Choose “Convert Image” mode	User chooses “Convert Image” mode to convert image format.
Step 3	Choose image to convert	User chooses an image to convert from their computer’s file system. If the image’s format is invalid, return to step 2, else, proceed.
Step 4	Choose desired format	User chooses the format which they want to convert their image to. The desired format must not be the same as the chosen image’s format.
Step 5	Upload image	User uploads the image to the server.
Step 6	Convert image	The system converts the image format to the desired format.

Step 7	Store converted image	Upon finishing conversion, the system stores the converted image on the server's file system.
Step 8.1	Download converted image	The user downloads the converted image to their computer.
Step 8.2.1	Write logs	The system writes logs about the conversion process.
Step 8.2.2	View logs	The admin views the log

Table 4.1.1. Description of the Image format conversion workflow

Image Splitting

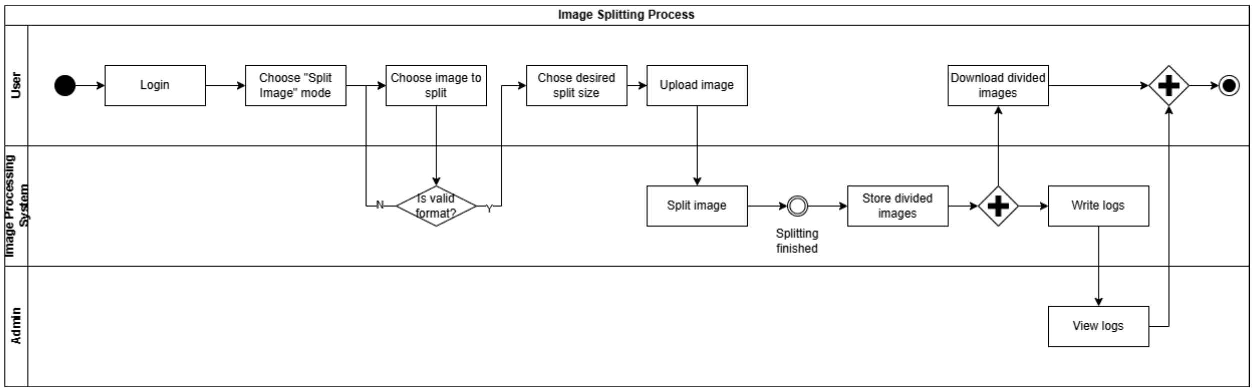


Figure 4.1.2. Business workflow of the Image splitting process

Step	Step Name	Description
Step 1	Login	User logs into the system.
Step 2	Choose "Split Image" mode	User chooses "Split Image" mode to split an image.
Step 3	Choose image to split	User chooses an image to convert from their computer's file system. If the image's format is invalid, return to step 2, else, proceed.
Step 4	Choose desired split size	User chooses the split size they want. The split size is the size of the square tiles which are split from the original

		image.
Step 5	Upload image	User uploads the image to the server.
Step 6	Split image	The system split the image into square tiles with the desired split size.
Step 7	Store divided images	Upon finishing splitting, the system stores the image tiles on the server's file system.
Step 8.1	Download divided images	The user downloads the divided image (image tiles) to their computer.
Step 8.2.1	Write logs	The system writes logs about the splitting process.
Step 8.2.2	View logs	The admin views the log

Table 4.1.2. Description of the Image splitting workflow

Image Merging

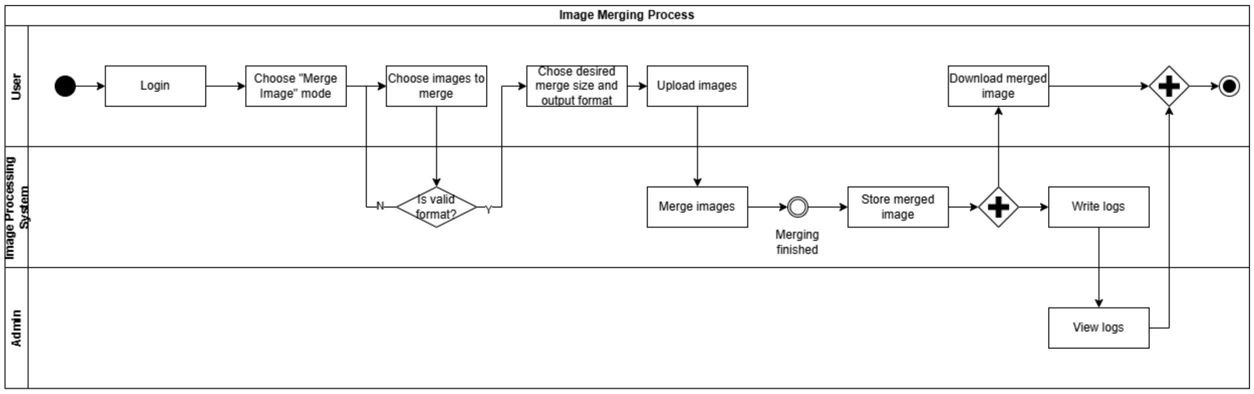


Figure 4.1.3. Business workflow of the Image merging process

Step	Step Name	Description
Step 1	Login	User logs into the system.
Step 2	Choose "Merge Image" mode	User chooses "Merge Image" mode to merge images into a larger one.

Step 3	Choose images to merge	User chooses images to merge from their computer's file system. If the format of any image is invalid, return to step 2, else, proceed.
Step 4	Choose desired merge size and output format	User chooses the desired size (width and height) and the format of the merged image.
Step 5	Upload image	User uploads the images to the server.
Step 6	Merge images	The system merges the images into a large image with the desired size.
Step 7	Store merged image	Upon finishing merging, the system stores the merged image on the server's file system.
Step 8.1	Download merged image	The user downloads the merged image to their computer.
Step 8.2.1	Write logs	The system writes logs about the splitting process.
Step 8.2.2	View logs	The admin views the log

Table 4.1.3. Description of the Image merging workflow

4.2. Use-case diagrams

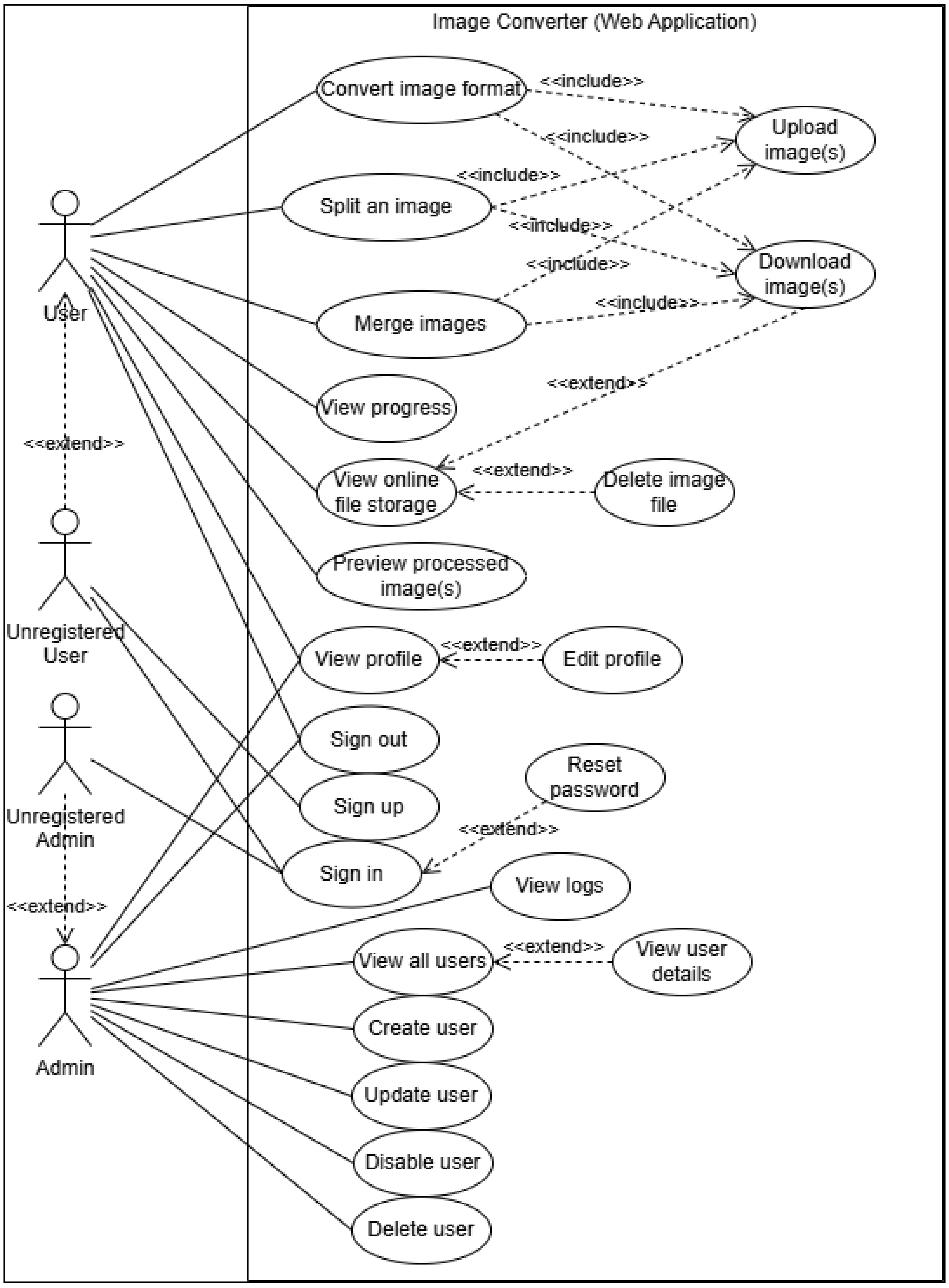


Figure 4.2.1. Use-case diagram

4.3. Use-case description for each use-cases

4.3.1. Use Case List

Primary Actor	Secondary actor	Use Case name	Note
User	None	UC-1: Convert image format	
User	None	UC-2: Split an image	
User	None	UC-3: Merge images	
User	None	UC-4: View progress	
User	None	UC-5: Upload image(s)	
User	None	UC-6: Download image(s)	
User	None	UC-7: Preview processed image(s)	
User	None	UC-8: View profile	
User	None	UC-9: Edit profile	
Unregistered User, Unregistered Admin	None	UC-10: Sign in	
Unregistered User	None	UC-11: Sign up	
User, Admin	None	UC-12: Sign out	
Unregistered User, Unregistered Admin	None	UC-13: Reset password	
Admin	None	UC-14: View all users	
Admin	None	UC-15: View user details	
Admin	None	UC-16: Create user	
Admin	None	UC-17: Update user	

Admin	None	UC-18: Disable user	
Admin	None	UC-19: Delete user	
Admin	None	UC-20: View logs	
User	None	UC-21: View online file storage	
User	None	UC-22: Delete image files	

Table 4.3.1. Use Case List

4.3.2. Use Case Description

UC-1: Convert image format

UC ID and Name:	UC-1: Convert image format		
Created By:	Bui Minh Son	**Date Created:**	2/10/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	The user wants to convert their images to another format.		
Description:	As a user, I want to convert my images to another format.		
Preconditions:	PRE-1: The user must log into the system.		
Post-conditions:	POST-1: The converted image is downloaded to the user's local machine.		
Normal Flow:	1: The user visits the page "Convert Image". 2: The user chooses the desired format, then chooses the image for conversion. 3: The user then uploads the image to the server, and waits until the conversion is done. 4: The user can upload more images, and the conversion for those images are processed concurrently. 5: When the conversion is done, the user can download the converted image(s).		
Alternative Flows:	None		

Exceptions:	1: Unable to connect to the server. 1.1: Display error message “Unable to connect to server” on the screen. 2: Server error while processing. 2.1: Display an error message on the screen to notify the user that there is an error on the server, and ask the user to try again later. 3: Invalid image format. 3.1: After choosing the image, a message will be displayed on the screen, notifying that the image format is not accepted. 3.2: The user then chooses another image.
Priority:	High
Frequency of Use:	Frequent
Business Rules:	BR-1
Other Information:	None
Assumptions:	None

UC-2: Split an image

UC ID and Name:	UC-2: Split an image		
Created By:	Bui Minh Son	**Date Created:**	2/6/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	The user wants to split images into smaller square tiles.		
Description:	As a user, I want to split my images into smaller square tiles.		
Preconditions:	PRE-1: The user must log into the system.		
Post-conditions:	POST-1: The tiles split from the images are downloaded to the user’s local machine.		

	POST-2: If not downloaded right away, the tiles split from the images are stored on the server's file system for 30 days, and are accessible by the user.
Normal Flow:	1: The user visits the page "Split Image". 2: The user chooses the desired tile size, then chooses the image for splitting. 3: The user then uploads the image to the server, and waits until the splitting is done. 4: The user can upload more images, and the splitting for those images are processed concurrently. 5: When the splitting is done, the user can download the image tiles.
Alternative Flows:	None
Exceptions:	1: Unable to connect to the server. 1.1: Display error message "Unable to connect to server" on the screen. 2: Server error while processing. 2.1: Display an error message on the screen to notify the user that there is an error on the server, and ask the user to try again later. 3: Invalid image format. 3.1: After choosing the image, a message will be displayed on the screen, notifying that the image format is not accepted. 3.2: The user then chooses another image.
Priority:	High
Frequency of Use:	Frequent
Business Rules:	BR-1
Other Information:	None
Assumptions:	None

UC-3: Merge images

UC ID and Name:	UC-3: Merge small images		
Created By:	Bui Minh Son	**Date Created:**	2/6/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	The user wants to merge small images into a larger image.		
Description:	As a user, I want to merge my images into a larger image.		
Preconditions:	PRE-1: The user must log into the system.		
Post-conditions:	POST-1: The merged image is downloaded to the user's local machine. POST-1: If not downloaded right away, the merged image is stored on the server's file system for 30 days, and is accessible by the user.		
Normal Flow:	1: The user visits the page "Merge Image". 2: The user chooses the desired image size and output format, then chooses the images for merging. 3: The user then uploads the images to the server, and waits until the merging is done. 4: If the format of any uploaded images do not match the output format, the system will convert them before merging. 5: When the merging is done, the user can download the merged image, or preview.		
Alternative Flows:	None		
Exceptions:	1: Unable to connect to the server. 1.1: Display error message "Unable to connect to server" on the screen. 2: Server error while processing. 2.1: Display an error message on the screen to notify the user that there is an error on the server, and ask the user to try again later. 3: Invalid image format. 3.1: After choosing the image, a message will be displayed on the screen, notifying that the image format is not accepted. 3.2: The user then chooses another image.		

Priority:	High
Frequency of Use:	Not frequent
Business Rules:	BR-1
Other Information:	None
Assumptions:	None

UC-4: View progress

UC ID and Name:	UC-4: View progress		
Created By:	Bui Minh Son	**Date Created:**	2/10/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	This function triggers when the server is processing image(s) for the user.		
Description:	As a user, I want to view the progress of image processing.		
Preconditions:	PRE-1: The user must log into the system. PRE-2: The system is processing the user's uploaded image.		
Post-conditions:	POST-1: Upon finishing processing, the progress bar disappears.		
Normal Flow:	1: When an image is uploaded to the server, a web socket is established between the client and the server. 2: The server updates the progress to the client upon finishing each step in the process.		
Alternative Flows:	None		
Exceptions:	1: Unable to connect to the server. 1.1: Display error message “Unable to connect to server” on the screen.		

	2: Server error while processing. 2.1: Display an error message on the screen to notify the user that there is an error on the server, and ask the user to try again later.
Priority:	Normal
Frequency of Use:	Frequent
Business Rules:	None
Other Information:	None
Assumptions:	None

UC-5: Upload image(s)

UC ID and Name:	UC-5: Upload image(s)		
Created By:	Bui Minh Son	**Date Created:**	2/8/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	This function triggers when the user clicks the "Upload image" button.		
Description:	As a user, I want to upload my image(s) to the server, so that it can process my image(s).		
Preconditions:	PRE-1: The user must log into the system.		
Post-conditions:	POST-1: All user-uploaded images are on the server's temporary folder, ready for processing.		
Normal Flow:	1: The user chooses the processing mode (convert, merge, split). 2: The user clicks on the "Upload image" button, and chooses the image to be processed. 3: The user chooses the desired output format for the image. 4: After that, click "Upload".		

	5: Image is uploaded and is stored in a temporary folder on the server, and a task created and is queued for processing. 6: A progress bar for the uploaded image appeared on the screen. Meanwhile, the user can still choose more images to upload.
Alternative Flows:	None
Exceptions:	1: Invalid image format. 1.1: After step 4, a message will be displayed on the screen, notifying that the image format is not accepted. 1.2: The user then chooses another image. 2: Error during uploading image. 1.1: After step 4, a message will be displayed on the screen, notifying the user to try again later.
Priority:	Normal
Frequency of Use:	Frequent
Business Rules:	BR-1
Other Information:	None
Assumptions:	None

UC-6: Download image(s)

UC ID and Name:	UC-6: Download image(s)		
Created By:	Bui Minh Son	**Date Created:**	2/8/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	This function triggers every time when the server finishes processing a user’s uploaded image and the user clicks on the “Download” button.		

Description:	As a user, I want to download the processed image(s) to my local machine.
Preconditions:	PRE-1: The user has images being processed on the server.
Post-conditions:	POST-1: The processed image is saved to the user's local machine.
Normal Flow:	1: Upon finishing processing, the "Download" button appears. 2: The user clicks "Download" to download the image to their device. 3: When downloading is finished, the file is deleted from the server's storage.
Alternative Flows:	None
Exceptions:	1: Error during downloading image. 1.1: After step 2 of the normal flow, a message appears on the screen to notify the user to try again later.
Priority:	Normal
Frequency of Use:	Frequent
Business Rules:	None
Other Information:	None
Assumptions:	None

UC-7: Preview processed image(s)

UC ID and Name:	UC-7: Preview processed image(s)		
Created By:	Bui Minh Son	**Date Created:**	2/10/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	This function is triggered when image processing is completed		

Description:	As a user, I want to preview the processed image, so I can decide whether or not to download it.
Preconditions:	PRE-1: The system has finished processing the user's provided image(s).
Post-conditions:	None
Normal Flow:	1: After processing finished, a button “Preview” appears. 2: The user clicks on “Preview”. 3: The processed image will appear on the screen for the user to preview.
Alternative Flows:	None
Exceptions:	None
Priority:	Normal
Frequency of Use:	Not frequent
Business Rules:	None
Other Information:	None
Assumptions:	None

UC-8: View profile

UC ID and Name:	UC-8: View profile		
Created By:	Tran Quang Huy	**Date Created:**	02/11/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	User-Initiated Triggers by **clicks on "View User Profile"** in the settings menu.		

Description:	This use case describes how a user updates their profile information within an **Image Processing System**.
Preconditions:	The user must be logged into the system. The system must be online and accessible. The user must have permission to update their profile.
Post-conditions:	The user successfully views their profile. The system logs profile access for audit and security purposes.
Normal Flow:	NF-1: **User Login (if applicable):** The user logs into the system using valid credentials. NF-2: **Navigate to Profile Section:** The user accesses the **profile** section from the dashboard or the account settings. NF-3: **Profile Display:** ● The system loads and displays the user's profile, including: ○ Personal details (name, email, profile picture). ○ Account preferences (image format settings, default processing options). ○ Image history (previously uploaded and processed images). ○ Activity log (recent activities related to image processing). NF-4: **View Profile Details:** The user can see the complete details of their account and processing activities. NF-5: **Navigation Options:** The user can: ● Edit their profile (update personal information, change settings). ● View and manage uploaded images. ● Navigate back to the dashboard or other sections.
Alternative Flows:	N/A
Exceptions:	E-1: **Slow Network or Timeout** ● If network issues or server delays prevent loading the profile: ○ Return error: **"Unable to load your profile due to a network issue. Please try again later."** ○ Allow users to retry or contact support.

	E-2: **Profile Not Found (Error Flow)** ● If the user's profile data cannot be found or is corrupted, the system will: ○ Return error: **"Error: Profile data not found."** ○ Suggest re-login or contact support.
Priority:	Medium
Frequency of Use:	Not very frequently
Business Rules:	BR-3
Other Information:	N/A
Assumptions:	User must sign-in first

UC-9: Edit profile

UC ID and Name:	UC-9: Edit profile		
Created By:	Tran Quang Huy	**Date Created:**	02/11/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	User-Initiated Triggers by **clicks on "Edit Profile"** in the settings menu.		
Description:	This use case describes how a user updates their profile information within an **Image Processing System**.		
Preconditions:	The user must be logged into the system. The system must be online and accessible. The user must have permission to update their profile.		

Post-conditions:	The profile is successfully updated and stored in the system. System logs profile updates for security and audit purposes. Users can see the updated profile in future sessions.
Normal Flow:	**NF-1: User Login:** The user logs into the image processing system with valid credentials. **NF-2: Navigate to Profile Settings:** The user accesses the profile section from the dashboard. **NF-3: Edit Profile Information:** The user updates details such as: ● Name ● Email ● Profile picture ● Password **NF-4: Save Changes:** The user submits the updated profile. **NF-5: Validation Check:** The system validates the entered data (e.g., correct email format, password security). **NF-6: Update Confirmation:** The system saves changes and confirms: *"Profile updated successfully."*
Alternative Flows:	AL-1: **Unauthorized Access** ● If a user without proper permissions tries to edit their profile, the system denies access: *"Access denied. You do not have permission to edit this profile."* AL-2: **Profile Picture Upload Failure** ● If the user uploads an unsupported image format, the system rejects it: *"Invalid file format. Please upload a JPEG or PNG file."*
Exceptions:	E-1: **Invalid Input (Error Handling Flow)** ● If the user enters invalid data (e.g., incorrect email format, weak password), the system displays an error message: *"Invalid input. Please enter a valid email address."* E-2: Profile Picture Upload Exception E-3: Network Timeout Exception

	● If the system is down or experiencing slow response times, it displays: *"Profile update failed. Please try again later."* E-4: Session Expired Exception ● The user's session times out before saving changes. ● Display error: **"Session expired. Please log in again to continue."** E-5: Database Update Exception ● **Cause:** Failure in updating the database after profile changes. ● Display error: **"Profile update failed due to a system error. Please try again later."**
Priority:	Medium
Frequency of Use:	Not very frequently
Business Rules:	BR-2, BR-5
Other Information:	N/A
Assumptions:	User must sign-in first

UC-10: Sign in

UC ID and Name:	UC-10: Sign in		
Created By:	Vu Trong Dung	**Date Created:**	2/10/2025
Primary Actor:	Unregistered User, Unregistered Admin	**Secondary Actors:**	None
Trigger:	The user wants to access their account and initiates the sign-in process.		
Description:	This use case describes the process of a user logging into the system using their email and password.		

Preconditions:	The user must have an existing, registered account.
Post-conditions:	PC-1: If successful → The user is authenticated and redirected to their dashboard. PC-2: If unsuccessful → The user remains on the login page and sees an appropriate error message.
Normal Flow:	NF-1: The user enters their email and password on the login page. NF-2: The system validates the credentials. ● If valid, the system logs the user in and redirects them to their dashboard.
Alternative Flows:	AF-1: Incorrect Email or Password ● If the user enters incorrect credentials, the system displays: "Invalid email or password."
Exceptions:	E1: Account Not Found → If the email is not registered, the system notifies the user. E2: System Failure → If there is a server error, the system displays: "Unable to sign in. Please try again later."
Priority:	High
Frequency of Use:	Multiple times daily per user
Business Rules:	BR-2, BR-5, BR-6
Other Information:	N/A
Assumptions:	The user remembers their credentials or has a way to reset their password if needed.

UC-11: Sign up

UC ID and Name:	UC-11: Sign up		
Created By:	Vu Trong Dung	**Date Created:**	2/10/2025
Primary Actor:	Unregistered User	**Secondary Actors:**	None
Trigger:	The user wants to create a new account.		
Description:	This use case describes the process of a new user registering for an account.		
Preconditions:	The user must have a valid email address.		
Post-conditions:	PC-1: If successful → The user receives a confirmation email and can log in. PC-2: If unsuccessful → The system displays an error message, and the user remains on the registration page.		
Normal Flow:	NF-1: The user navigates to the Sign Up page. NF-2: The user enters their full name, email, password, and confirms the password. NF-3: The system validates the input. ● If valid, the system generates a 6-digit OTP and sends it to the user's email. NF-4: The user enters the OTP. NF-5: The system verifies the OTP. ● If correct, the system creates the account and redirects the user to the login page.		
Alternative Flows:	AF-1: Email Already Exists ● If the email is already registered, the system displays: "This email is already in use. Please log in."		
Exceptions:	E1: Weak Password → The system rejects passwords that do not meet security criteria.		

	E2: Invalid OTP → If the user enters an incorrect or expired OTP, they must request a new one. E3: System Failure → If a server error occurs, the system displays: "Registration unavailable. Please try again later."
Priority:	High
Frequency of Use:	Occasionally
Business Rules:	BR-2, BR-5 , BR-6
Other Information:	N/A
Assumptions:	The user has access to their email to verify the OTP.

UC-12: Sign out

UC ID and Name:	UC-12: Sign out		
Created By:	Vu Trong Dung	**Date Created:**	2/10/2025
Primary Actor:	User, Admin	**Secondary Actors:**	None
Trigger:	The user wants to log out.		
Description:	This use case describes the process of a user logging out from the system.		
Preconditions:	The user must be logged in.		
Post-conditions:	PC-1: If successful → The user session is terminated, and they are redirected to the login page.		

	PC-2: If unsuccessful → The user remains logged in due to a system failure.
Normal Flow:	NF-1: The user clicks Sign Out. NF-2: The system terminates the session. NF-3: The system redirects the user to the login page.
Alternative Flows:	AF-1: Auto Logout Due to Inactivity ● If the user is inactive for too long, the system logs them out automatically.
Exceptions:	E1: Session Already Expired → If the session expired, the user is simply redirected to the login page. E2: System Failure → If the system fails to terminate the session, the user sees an error.
Priority:	High
Frequency of Use:	Multiple times per day
Business Rules:	BR-7
Other Information:	N/A
Assumptions:	N/A

UC-13: Reset password

UC ID and Name:	UC-13: Reset password		
Created By:	Vu Trong Dung	**Date Created:**	2/10/2025
Primary Actor:	Unregistered User, Unregistered Admin	**Secondary Actors:**	None
Trigger:	The user forgets their password and requests a reset.		

Description:	This use case describes how a user resets their password using link reset password sent to their email.
Preconditions:	The user must have a registered email.
Post-conditions:	PC-1: If successful → The user sets a new password and can log in. PC-2: If unsuccessful → The password remains unchanged.
Normal Flow:	NF-1: The user clicks "Forgot Password". NF-2: The system asks for the registered email. NF-3: The system sends a link reset password to the email. NF-4: The user clicks on the link reset password. ● If valid, the user sets a new password. NF-5: The system updates the new password and redirects to the login page.
Alternative Flows:	AF-1: Email Not Found - If the email entered by the user is not registered, the system displays an error message: "This email is not associated with any account." AF-2: Link reset password Expired - If the reset password expires (after 5 minutes), the user must request a new link reset password.
Exceptions:	E1: Email Delivery Failure → If the link reset password email fails to send, the system notifies the user: "Failed to send the link reset password. Please check your email or try again later." E2: Password Does Not Meet Security Requirements → If the new password does not meet security criteria, the system displays an error: "Passwords must be at least 8 characters long and include an uppercase letter, a number, and a special character."
Priority:	High
Frequency of Use:	Occasionally
Business Rules:	BR-5

Other Information:	N/A
Assumptions:	1. The user has access to their registered email inbox. 2. The email service used by the system is working properly and can send link reset password messages without delay. 3. The user is aware that password resets should not be shared with others for security reasons.

UC-14: View all users

UC ID and Name:	UC-14: View all users		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/10/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to view the list of all users in the system		
Description:	This use case allows an administrator to view a list of all registered users in the system, including basic details such as name, email, role, and activity status		
Preconditions:	PRE-1: The admin must log into the system. PRE-2: The admin must have permissions to view user data		
Post-conditions:	POST-1: The system displays a table listing all users with relevant details		
Normal Flow:	1.The admin logs into the system. 2.The admin navigates to the "User Management" page. 3.The admin clicks the "View All Users" button. 4.The system retrieves user data from the database. 5.The system displays the full list of users		
Alternative Flows:	AF-1: If the admin wants to search for a specific user, they can use the search function. AF-2: If the admin wants to filter users, they can apply filters such as role, account status, or registration date		

Exceptions:	E1: If the system fails to retrieve user data, display an error message: "Unable to load user list." E2: If there are no users in the system, display the message: "No users found”
Priority:	High
Frequency of Use:	Frequently
Business Rules:	BR-3, BR-4
Other Information:	None
Assumptions:	None

UC-15: View user details

UC ID and Name:	UC-15: View user details		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/10/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to view detailed information about a specific user		
Description:	This use case allows an administrator to view detailed information of a selected user, including their name, email, role, status, and activity history		
Preconditions:	PRE-1: The admin must be logged into the system. PRE-2: The admin must have permission to view user details. PRE-3: The user whose details are being viewed must exist in the system		
Post-conditions:	POST-1: The system displays the detailed profile of the selected user		
Normal Flow:	1.The admin logs into the system.		

	2.The admin navigates to the "User Management" page. 3.The admin searches for or selects a user from the list. 4.The admin clicks on the "View Details" button. 5.The system retrieves the user's information from the database. 6.The system displays the user's detailed information on a new page or modal
Alternative Flows:	None
Exceptions:	E1: If the selected user does not exist, display an error message: "User not found." E2: If there is an issue retrieving user details, display an error: "Unable to load user details."
Priority:	High
Frequency of Use:	Frequently
Business Rules:	BR-3 BR-4
Other Information:	None
Assumptions:	None

UC-16: Create user

UC ID and Name:	UC-16: Create user		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/10/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to create a new user account in the system		

Description:	This use case allows an administrator to create a new user account by entering user details. After successful creation, the system sends a notification email to the new user
Preconditions:	PRE-1: The admin must be logged into the system. PRE-2: The admin must have permission to create new users. PRE-3: The user’s email must be unique and valid
Post-conditions:	POST-1: A new user account is created in the system. POST-2: The system sends an email notification to the newly created user
Normal Flow:	1.The admin logs into the system. 2.The admin navigates to the "User Management" page. 3.The admin clicks on the "Create User" button. 4.The system displays a form for entering user details 5.The admin fills in the required fields and submits the form. 6.The system validates the input data. 7.The system creates the new user in the database. 8.The system generates a welcome email and sends it to the user's registered email. 9.The system displays a confirmation message
Alternative Flows:	AF-1: If the admin wants to set a temporary password, they can enable the "Force Password Change on First Login" option
Exceptions:	E1: If the email is already in use, display an error: "This email is already registered." E2: If required fields are missing, prompt the admin to complete the form. E3: If the email notification fails to send, display a warning: "User created, but the notification email could not be sent."
Priority:	Normal
Frequency of Use:	When new users need to be added
Business Rules:	BR-2, BR-3, BR-5
Other Information:	None

Assumptions:	None

UC-17: Update user

UC ID and Name:	UC-17: Update user		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/11/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to update a user’s information in the system		
Description:	This use case allows an administrator to modify user details such as name, email, role, status		
Preconditions:	PRE-1: The admin must be logged into the system. PRE-2: The admin must have permission to update user details. PRE-3: The user being updated must exist in the system		
Post-conditions:	POST-1: The user’s information is successfully updated in the system. POST-2: If critical information (e.g., email, role) is changed, the system may notify the user		
Normal Flow:	1.The admin logs into the system. 2.The admin navigates to the "User Management" page. 3.The admin searches for the user they want to update. 4.The admin selects the user and clicks on the "Edit" button. 5.The system displays a form with the user’s current details. 6.The admin updates the necessary fields and submits the form. 7.The system validates the input data. 8.The system updates the user’s information in the database. 9.The system displays a success message: "User updated successfully."		
Alternative Flows:	AF-1: If the admin wants to reset the user’s password, they can use the "Reset Password" option.		

	AF-2: If the admin wants to deactivate the user, they can change the account status to "Inactive."
Exceptions:	E1: If the user does not exist, display an error message: "User not found." E2: If the updated email already exists in the system, show an error: "This email is already in use." E3: If required fields are missing, prompt the admin to complete the form. E4: If the system fails to save changes, display an error: "Failed to update user information."
Priority:	High
Frequency of Use:	When user details need to be modified
Business Rules:	BR-3, BR-5
Other Information:	None
Assumptions:	None

UC-18: Disable user

UC ID and Name:	UC-18: Disable user		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/11/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to deactivate a user account in the system		
Description:	This use case allows an administrator to disable a user account, preventing the user from logging in. After the account is disabled, the system sends a notification email to inform the user		

Preconditions:	PRE-1: The admin must be logged into the system. PRE-2: The admin must have permission to disable user accounts. PRE-3: The user being disabled must exist in the system
Post-conditions:	POST-1: The user account is marked as "Inactive" in the system. POST-2: The system sends an email notification to inform the user about the account deactivation
Normal Flow:	1.The admin logs into the system. 2.The admin navigates to the "User Management" page. 3.The admin searches for the user to be disabled. 4.The admin selects the user and clicks on the "Disable" button. 5.The system displays a confirmation prompt: "Are you sure you want to disable this user?" 6.The admin confirms the action. 7.The system updates the user’s status to "Disabled" in the database. 8.The system generates and sends an email notification to the user. 9.The system displays a success message: "User has been disabled, and a notification email has been sent."
Alternative Flows:	AF-1: If the admin wants to provide a reason for disabling the user, they can enter a note before confirming. AF-2: If the admin wants to re-enable the user later, they can update the user’s status back to "Active"
Exceptions:	E1: If the user does not exist, display an error message: "User not found." E2: If the system fails to disable the user, show an error: "Failed to disable user. Please try again." E3: If the email notification fails to send, display a warning: "User disabled, but the notification email could not be sent."
Priority:	High
Frequency of Use:	When user accounts need to be deactivated
Business Rules:	BR-3, BR-6

Other Information:	None
Assumptions:	None

UC-19: Delete user

UC ID and Name:	UC-19: Delete user		
Created By:	Ho Vu Tuan Minh	**Date Created:**	2/11/2025
Primary Actor:	Admin	**Secondary Actors:**	None
Trigger:	The admin wants to permanently delete a user from the system		
Description:	This use case allows an administrator to delete a user account. Before deletion, the system removes all files and associated data linked to the user. Once deletion is completed, the system sends a notification email to inform the user		
Preconditions:	PRE-1: The admin must be logged into the system. PRE-2: The admin must have permission to delete user accounts. PRE-3: The user being deleted must exist in the system		
Post-conditions:	POST-1: The user account is permanently removed from the system. POST-2: All files and associated data of the user are deleted. POST-3: A notification email is sent to inform the user about the deletion		
Normal Flow:	1.The admin logs into the system. 2.The admin navigates to the "User Management" page. 3.The admin searches for the user to be deleted. 4.The admin selects the user and clicks the "Delete" button. 5.The system displays a confirmation prompt:"Are you sure you want to delete this user?" 6.The admin confirms the deletion.		

	7.The system removes all files and associated data of the user,deletes the user from the database, and sends a notification email to the user. 8.The system displays a success message
Alternative Flows:	AF-1: If the admin wants to retain user files but delete only the account, they can uncheck the "Delete all files" option. AF-2: If the admin wants to archive the user instead of deleting them, they can select "Deactivate" instead
Exceptions:	E1: If the user does not exist, display an error message: "User not found." E2: If the system fails to delete user data, display an error: "Failed to delete user data. Please try again." E3: If the email notification fails to send, display a warning: "User deleted, but the notification email could not be sent."
Priority:	High
Frequency of Use:	When user accounts need to be permanently removed
Business Rules:	BR-3, BR-7, BR-10
Other Information:	None
Assumptions:	None

UC-20: View logs

UC ID and Name:	UC-20: View logs		
Created By:	Nguyen Viet Hoang	**Date Created:**	
Primary Actor:	Admin	**Secondary Actors:**	None

Trigger:	The admin selects the option to view system logs from the admin panel or settings section.
Description:	This use case allows the admin to access and view detailed logs of system activities, including user actions, file uploads, conversions, and other system events. The logs are intended for monitoring system usage and identifying any potential issues.
Preconditions:	PRE-1:The admin is logged in with the appropriate admin credentials. PRE-2:The admin has permission to access logs within the system.
Post-conditions:	POST-1: The admin successfully views the logs. POST-2: If any errors or unusual activities are identified, the admin can take appropriate action fixing a system issue.
Normal Flow:	1.The admin logs into the admin panel. 2.The admin selects the "View logs" option from the dashboard. 3.The system retrieves and displays the logs, including time , activity details, and the responsible users. 4.The admin reviews the logs. 5.The admin may filter logs by date, user, or event type, if necessary. 6.The admin can download or export the logs for further analysis.
Alternative Flows:	1.If no logs are available, the system will display a message: "No logs found for the selected time period." 2.If the admin does not have sufficient permissions to view logs, the system will display an error message: "You do not have permission to view logs."
Exceptions:	If the system fails to retrieve logs due to a server issue, the system displays an error message: *"Unable to retrieve logs at this time. Please try again later."*
Priority:	Medium
Frequency of Use:	Occasional
Business Rules:	BR-8
Other Information:	N/A

Assumptions:	N/A

UC-21: View online file storage

UC ID and Name:	UC-21: View online file storage		
Created By:	Tran Quang Huy	**Date Created:**	
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	User chooses the Image Storage tab to show all the uploaded images on to the system.		
Description:	This use case describes how a user accesses and views their stored files in an online file storage system.		
Preconditions:	The user must be authenticated (logged in). The file storage system must be online and accessible. The user must have the necessary permissions to view files.		
Post-conditions:	User is able to view all uploaded images in file storage		
Normal Flow:	**NF-1: User Login:** The user logs into the file storage system using valid credentials. NF-2: **Dashboard Access:** The system displays the user's dashboard with a list of tabs and functions. NF-3: **Navigation:** The user can navigate the file storage tabs. NF-4: System will show all the uploaded images of that user..		
Alternative Flows:	AL-1: Slow Network or Server Timeout ● If the system experiences slow response times, it displays: *"Loading... Please wait."* or *"Network error. Retry later."*		
Exceptions:	The user successfully views the file or receives an appropriate error message.		
Priority:	High		

Frequency of Use:	Very frequently
Business Rules:	BR-9
Other Information:	N/A
Assumptions:	User must sign-in first

UC-22: Delete image files

UC ID and Name:	UC-22: Delete image file		
Created By:	Tran Quang Huy	**Date Created:**	02/11/2025
Primary Actor:	User	**Secondary Actors:**	None
Trigger:	User clicks on the icon [:] on the top-right corner of the image then chooses the delete option. User clicks on the Select Image button on the top-right corner in Image Storage, choose images you want to delete then choose the delete option.		
Description:	This use case describes the process of a user deleting their image(s) using system functions.		
Preconditions:	Users must sign-in and go to the Image storage section. The user's account must have at least 1 image in the Image Storage.		
Post-conditions:	Image(s) deleted successfully.		
Normal Flow:	NF-1: The user initiates an image deletion request. NF-2: The system validates the input ● If invalid (the image has been locked), message "The image has been locked, unlock the image first to delete this image!" NF-3: The system verifies if the user has the required **permissions** to delete the image. ● If the user is unauthorized, access is denied. Message "You're not authorized to delete the image!" NF-4: A pop-up screen will appear to confirm that the user wants to delete the image. ● If "Yes" Delete the Image Files		

	● If "No" end the process and return to the Image Storage main screen
Alternative Flows:	N/A
Exceptions:	E1: Image Locked/In Use Exception E2: Database Update Exception E3: Network Timeout Exception E4: Compliance/Retention Policy Exception
Priority:	High
Frequency of Use:	Very frequently
Business Rules:	BR-9, BR-10
Other Information:	N/A
Assumptions:	User must sign-in first

4.4. Activity Diagram

None

4.5. Non-functional Requirements

4.5.1. Usability

- **User-Friendly Interface**: The interface should be intuitive, easy to use.
- **Cross-Browser Support**: The system must work properly on popular browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge.
- **Responsive Design**: The application should optimize the interface for mobile devices, tablets, and desktop users.

4.5.2. Security

- **Access Control:** The system must enforce role-based access, allowing only authorized users and admins to access permitted functions. Admin access is restricted to the internal network.
- **Data Ownership:** Users can only access their own data. Cross-user access is not allowed.
- **Data Protection:** All data must be transmitted securely (e.g., HTTPS). Sensitive information like password must be hashed before storage.
- **Threat Prevention:** The system must protect against common attacks such as SQL Injection, XSS, and CSRF.

4.5.3. Maintainability

- **Maintainable Code**: The system should follow modular architecture to facilitate easier upgrades and maintenance. (**6.2.1.4. Internal Layered Architecture of the Software System**)

4.5.4. Reliability

- **Uptime**: The system should guarantee a minimum uptime of 96% to ensure users can access it anytime.

4.5.5. Availability

- System must be available 24/7 (except on scheduled maintenance).

4.6. Business Rules Table

ID	Rule Definition	Type of Rule	Static or Dynamic
BR-1	Accepted image formats include .jpg, .bmp, .png and .webm	Validation Rule	Static
BR-2	Each email must be unique to each user.	Validation Rule	Static
BR-3	Only administrators can access the full user list and manage user detail information.	Access Control Rule	Static
BR-4	Sensitive user information such as passwords must not be displayed.	Security & Privacy Rule	Static

BR-5	Users must receive a notification email upon account creation or when information of the user changed.	Notification Email	Dynamic
BR-6	Disabled users cannot log into the system until reactivated.	Access Control Rule	Dynamic
BR-7	The system must delete all files and associated data before removing the user.	Data Management Rule	Static
BR-8	Logs should be secure and accessible only to authorized users (admin).	Security & Privacy Rule	Static
BR-9	Only authenticated users (already signed-in) can access images in Image Storage, and users can only access images that they own	Access Control Rule	Dynamic
BR-10	System may require two-step verification for irreversible deletions.	Data Management Rule	Dynamic

Table 4.6.1. Business Rules Table

4.7. Concurrent Tasks Description

There are 5 tasks that require concurrent processing, which are:

- Converting image format
- Splitting an image
- Merging images
- Updating progress to the user
- Write logs

5. Analysis Modeling

5.1. Static Modeling Diagrams and Explanation

5.1.1. Contextual External Class Diagram.

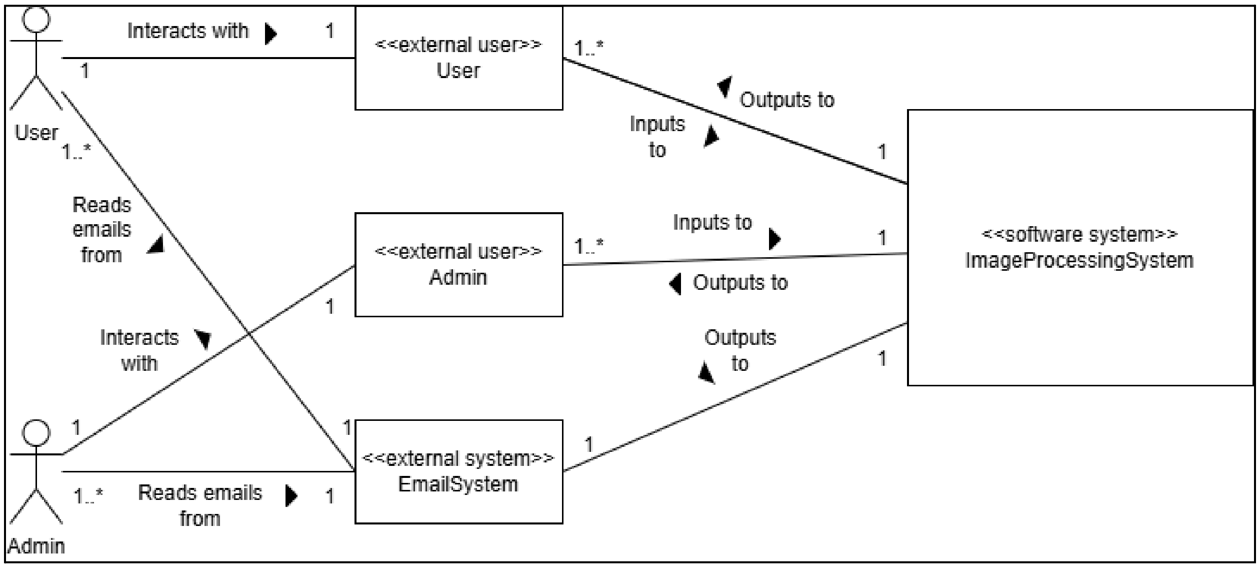


Figure 5.1.1.1. Contextual external class diagram

The **Image Processing System** interacts with five external classes that facilitate user access and system communication:

- **User and Admin:** Human users and administrators interacting with the system through a browser-based UI.
- **User and Admin (external user):** The <<external user>> classes represent the standard I/O devices such as screen, keyboard or mouse, through which the human Users and Admins interact with the system. They enable data input, image uploads, and system responses display, and is the primary access point for human users.
- **Email System:** The Email System is responsible for sending system-generated emails to Users and Admins. Functions include:
    - Sending notifications (e.g., system alerts).
    - Delivering OTP codes for authentication.
    - Providing password reset links for account recovery.

5.1.2. Business Processing Classes Hierarchical Structure

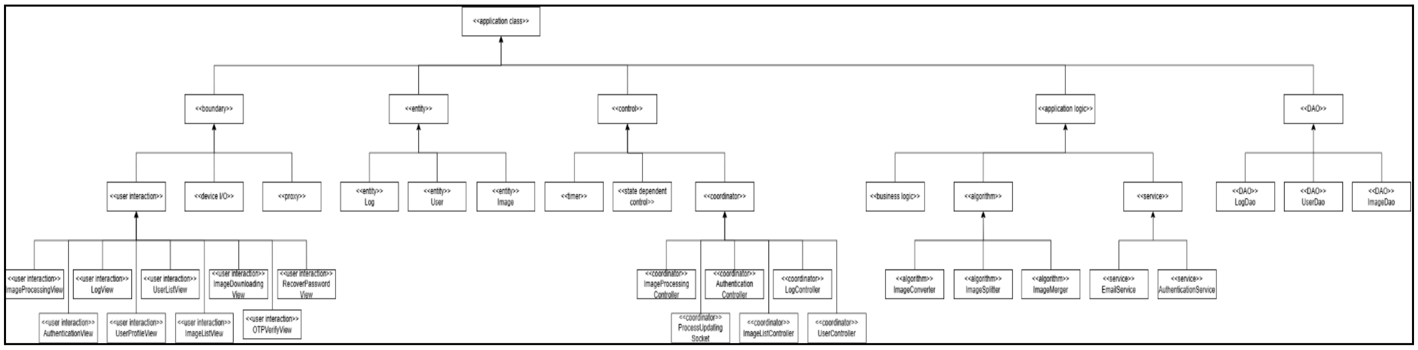

<<application class>>
<<boundary>>
<<entity>>
<<control>>
<<application logic>>
<<DAO>>
<<user interaction>>
<<device I/O>>
<<proxy>>
<<entity>> Log
<<entity>> User
<<entity>> Image
<<timer>>
<<state dependent control>>
<<coordinator>>
<<business logic>>
<<algorithm>>
<<service>>
<<DAO>> LogDao
<<DAO>> UserDao
<<DAO>> ImageDao
<<user interaction>> ImageProcessingView
<<user interaction>> LogView
<<user interaction>> UserListView
<<user interaction>> ImageDownloading View
<<user interaction>> RecoverPassword View
<<user interaction>> AuthenticationView
<<user interaction>> UserProfileView
<<user interaction>> ImageListView
<<user interaction>> OTPVerifyView
<<coordinator>> ImageProcessing Controller
<<coordinator>> Authentication Controller
<<coordinator>> LogController
<<coordinator>> ProcessUpdating Socket
<<coordinator>> ImageListController
<<coordinator>> UserController
<<algorithm>> ImageConverter
<<algorithm>> ImageSplitter
<<algorithm>> ImageMerger
<<service>> EmailService
<<service>> AuthenticationService

Figure 5.1.2.1. Business Processing Classes Hierarchical Structure

The diagram classifies application classes by stereotypes, organizing them into hierarchical layers. At the top level, the system has the **<<application class>>** stereotype, which represents all application-related classes. Below it, classes are categorized into more specific stereotypes based on their roles in the system:

<<user interaction>>

This stereotype represents UI-related classes that users interact with. This stereotype is in the **<<boundary>>** super-stereotype. Classes under this stereotype are:

- ***ImageProcessingView***: The view where user uploads images for processing and views progress
- ***LogView***: Display log list and log details
- ***UserListView***: Display user list and user profiles
- ***UserProfileView***: Display user's profile
- ***ImageDownloadingView***: The view where user can download or preview images
- ***AuthenticationView***: The view where user perform authentication actions (signIn, signUp, signOut)
- ***OTPVerifyView***: The view where user inputs OTP code
- ***RecoverPasswordView***: The view where user inputs new password to sign in
- ***ImageListView***: Display image list

<<entity>>

This stereotype represents core data classes that store and manage persistent system information. These classes usually model real-world objects within the system. Classes under this stereotype are:

- ***Log:*** Contains user actions logs

- ***User:*** Contains system's user data
- ***Image:*** Represents image-related objects in the system

<<coordinator>>

This stereotype represents controller-like classes responsible for managing interactions between various components. This stereotype is in the **<<control>>** super-stereotype. Classes under this stereotype are:

- ***ImageProcessingController***: Handles requests for processing images
- ***AuthenticationController***: Handles authentication requests
- ***LogController***: Handles requests for system's logs
- ***ImageListController***: Handles requests for images stored on the server
- ***UserController***: Handles requests for system's users information
- ***ProcessUpdatingSocket***: Updating image-processing progress from the server to client

<<application logic>>

This stereotype represents the core functional components responsible for executing the primary business processes of the Image Processing System. This stereotype contains:

- **<<algorithm>>:** Represents classes or components that implement computational logic, data transformation, or mathematical operations for processing data. Classes under this stereotype are:
    - ***ImageConverter***: Handles format conversion of uploaded images.
    - ***ImageSplitter***: Splits images into smaller parts.
    - ***ImageMerger***: Merges multiple images into a single output file.
- **<<services>>:** Contains supporting services that enhance system functionality but do not handle core business logic directly. Classes under this stereotype are:
    - ***EmailService***: Manages sending email notifications, password reset links, and verification OTP**.**
    - ***AuthenticationService***: Ensures user authentication and access control before allowing operations.

<<DAO>> (Data Access Object)

This stereotype represents database access classes, which handle CRUD operations (Create, Read, Update, Delete) for persistent storage. These classes act as the interface between the application and the database. Classes under this stereotype are:

- ***LogDao***: Manages log records in the database
- ***UserDao***: Manages user data in the database
- ***ImageDao***: Manages image-related database records

5.1.3. Entity Class Diagram

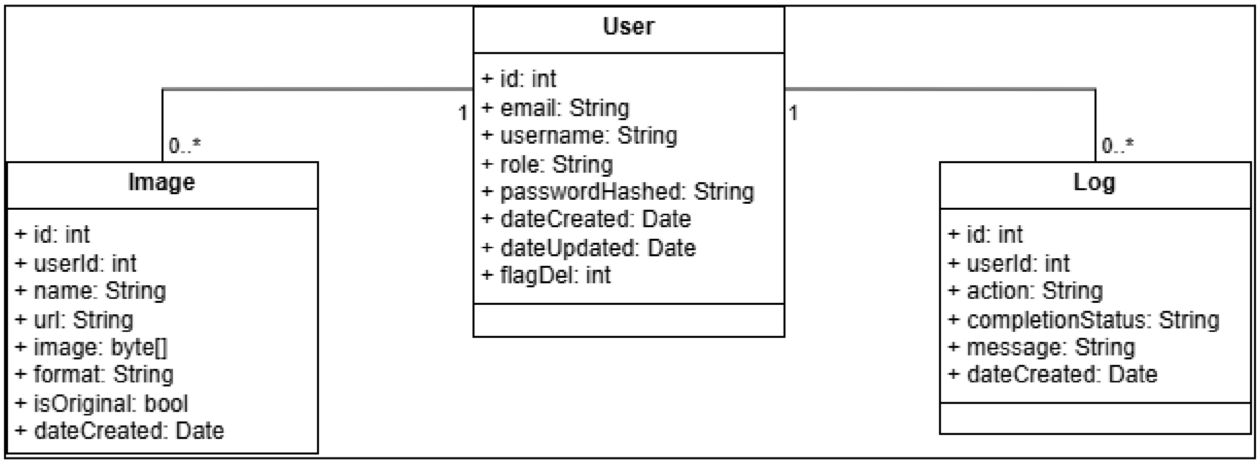


Figure 5.1.3.1. Entity class diagram

The entity class diagram represents the structure of the Image Processing System, showing the relationships between its key entities. It consists of three primary entities: **User**, **Log**, **Image**.

Entity name	Description	Attributes	Data Type	Note
User	Represents user related data	id	int	*N/A*
		email	String	*N/A*
		username	String	*N/A*
		role	String	*N/A*
		passwordHashed	String	*N/A*

		dateCreated	Date	*N/A*
		dateUpdated	Date	*N/A*
		flagDel	int	0: User active 1: User inactive
Image	Represents image data	id	int	*N/A*
		userId	int	*N/A*
		url	String	*N/A*
		name	String	Name of the image
		image	byte[]	The image is in binary format, stored in a byte array
		format	String	*N/A*
		isOriginal	bool	Determine whether the image is original (user-uploaded) or is processed and stored on database
		dateCreated	Date	*N/A*
Log	Represents data about user activities	id	int	*N/A*
		userId	int	*N/A*
		action	String	Specify the action taken by the user
		completionStatus	String	Specify whether the action is completed or not
		message	String	Specify completion or error message
		dateCreated	Date	*N/A*

Relationships Between Entities

- **User ↔ Image**: A user can have none or multiple images.
- **User ↔ Log**: A user can be associated with none or multiple logs.

5.1.4. Classes which perform Concurrent Tasks responsibility

There are 10 classes that perform concurrent tasks:

- ImageProcessingView
- ImageDownloadingView
- ImageProcessingController
- ImageListController
- ImageConverter
- ImageSplitter
- ImageMerger
- LogDao
- ImageDao
- ProcessUpdatingSocket

5.2. Dynamic Modeling Diagrams and Explanation

5.2.1. Communication Diagrams

5.2.1.1. Create User

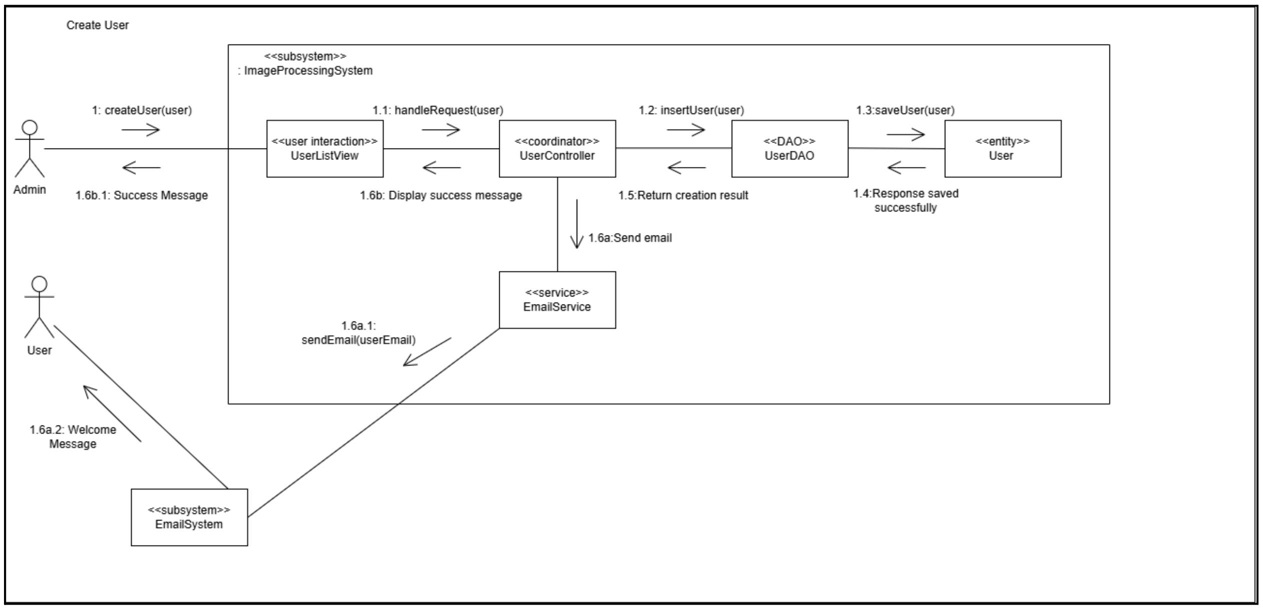


Figure 5.2.1.1. Create User Communication diagram

Description:

1.createUser(user): The Admin initiates the process of creating a new user by sending a request to the User Management System.

1.1. handleRequest(user): The request is handled by the User List View, which forwards the user details to the User Controller for processing.

1.2. insertUser(user): The User Controller processes the request and forwards the user details to the User DAO for database interaction.

1.3. saveUser(user): The User DAO interacts with the database and saves the new user's details.

1.4. Response saved successfully: The database confirms that the user's details have been successfully saved.

1.5. Return creation result: The User DAO returns the creation result to the User Controller.

1.6a. **Send email**: The User Controller instructs the Email Service to send a welcome email to the newly created user.

1.6a.1. sendEmail(userEmail): The Email Service processes the request and sends an email to the user.

1.6a.2. Welcome Message: The user receives a welcome message via email.

1.6b. Display success message: The User List View displays a success message indicating that the user was successfully created.

1.6b.1. Success Message: The Admin receives a success message confirming that the user has been created.

5.2.1.2. Update User

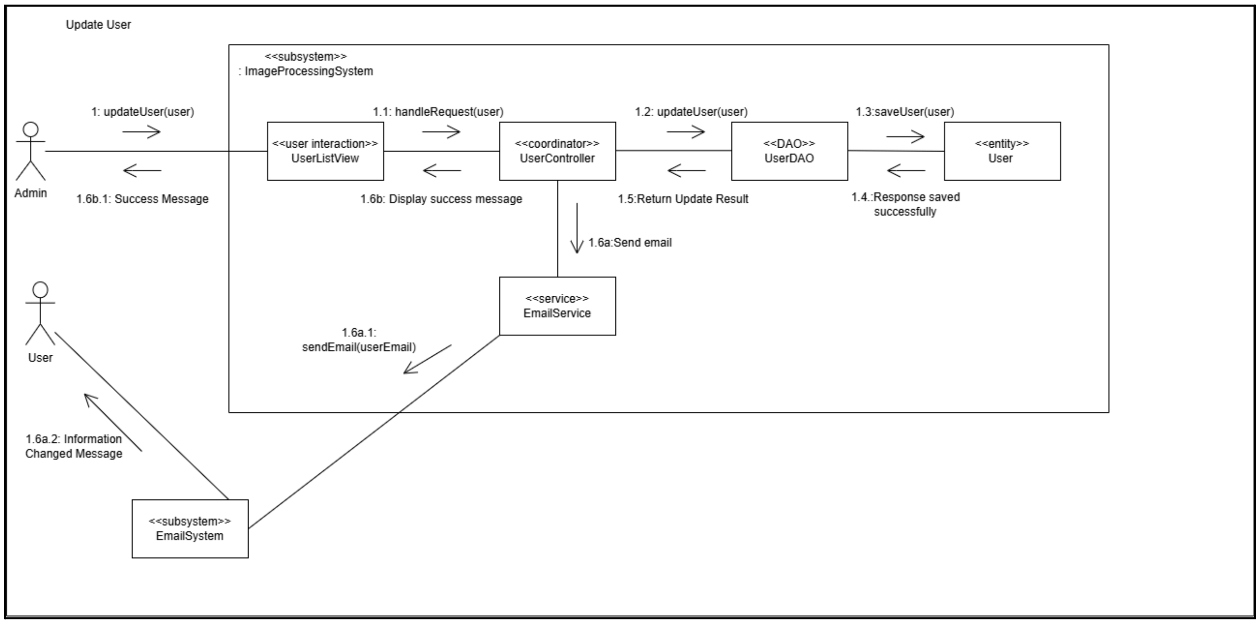


Figure 5.2.1.2. Update User Communication diagram

Description:

1.updateUser(user): The Admin initiates the process to update an existing user's information in the User Management System.

1.1. handleRequest(user): The request is processed by the User List View, which forwards the user details to the User Controller.

1.2. updateUser(user): The User Controller processes the update request and sends the user details to the User DAO for database operations.

1.3. saveUser(user): The User DAO interacts with the database and updates the user's details.

1.4. Response saved successfully: The database confirms that the user's updated information has been successfully saved.

1.5. Return Update Result: The User DAO returns the update result to the User Controller.

1.6a. Send email: The User Controller instructs the Email Service to notify the user about the information update.

1.6a.1. sendEmail(userEmail): The Email Service processes the request and sends an email notification to the user.

1.6a.2. Information Changed Message: The user receives an email informing them that their details have been updated.

1.6b. Display success message: The User List View displays a success message indicating that the update was successful.

1.6b. Success Message: The Admin receives a confirmation message stating that the user's details have been updated successfully.

5.2.1.3. Disable User

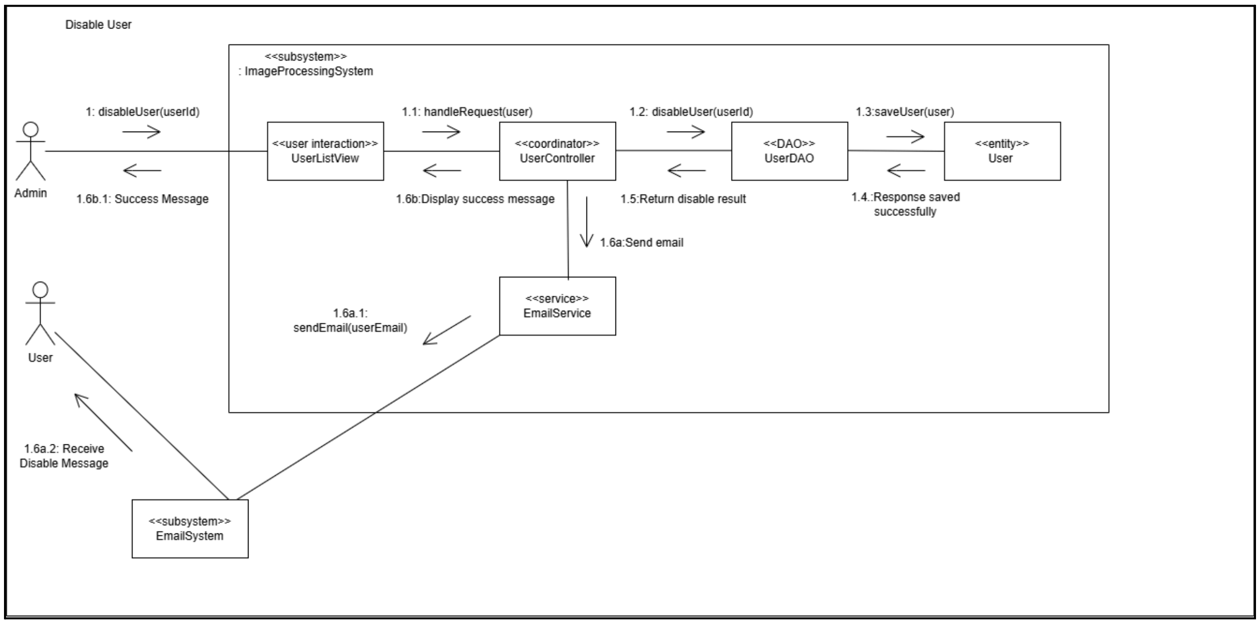


Figure 5.2.1.3. Disable User Communication diagram

Description:

1.disableUser(userId): The Admin initiates the process of disabling a user by sending a request to the User Management System.

1.1. handleRequest(user): The request is processed by the User List View, which forwards the request to the User Controller.

1.2. disableUser(userId): The User Controller processes the request and sends the user details to the User DAO for database operations.

1.3. saveUser(user): The User DAO interacts with the database and updates the user's status to disabled.

1.4. Response saved successfully: The database confirms that the user's status has been successfully updated.

1.5. Return disable result: The User DAO returns the disable operation result to the User Controller.

1.6a. Send email: The User Controller instructs the Email Service to notify the user about their account being disabled.

1.6a.1. sendEmail(userEmail): The Email Service processes the request and sends a disable notification email to the user.

1.6a.2. Receive Disable Message: The user receives an email informing them that their account has been disabled.

1.6b. Display success message: The User List View displays a success message indicating that the user has been disabled.

1.6b.1. Success Message: The Admin receives a confirmation message stating that the user has been successfully disabled.

5.2.1.4. Delete User

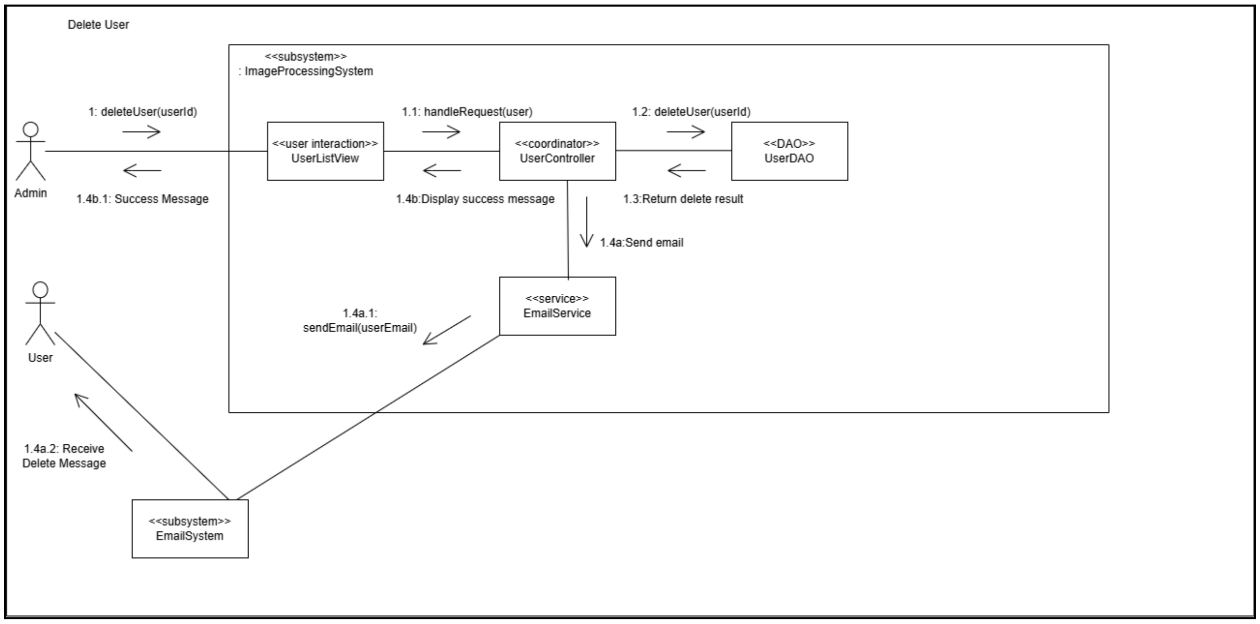


Figure 5.2.1.4. Delete User Communication diagram

Description:

1.deleteUser(userId): The Admin initiates the deletion process by sending a request to delete a user from the system.

1.1.handleRequest(user): The request is received and processed by the User List View, which forwards it to the User Controller.

1.2. deleteUser(userId): The User Controller processes the request and sends it to the User DAO for database operations.

1.3. Return delete result: The User DAO performs the deletion and returns the result to the User Controller.

1.4a. Send email: The User Controller triggers the Email Service to send a notification to the user regarding their account deletion.

1.4a.1. sendEmail(userEmail): The Email Service processes the request and sends a deletion notification to the user's registered email.

1.4a.2. Receive Delete Message: The user receives an email informing them that their account has been deleted.

1.4b. Display success message: The User List View updates the UI to show a success message confirming the deletion.

1.4b.1. Success Message: The Admin receives a confirmation message stating that the user has been successfully deleted.

5.2.1.5. View User Details

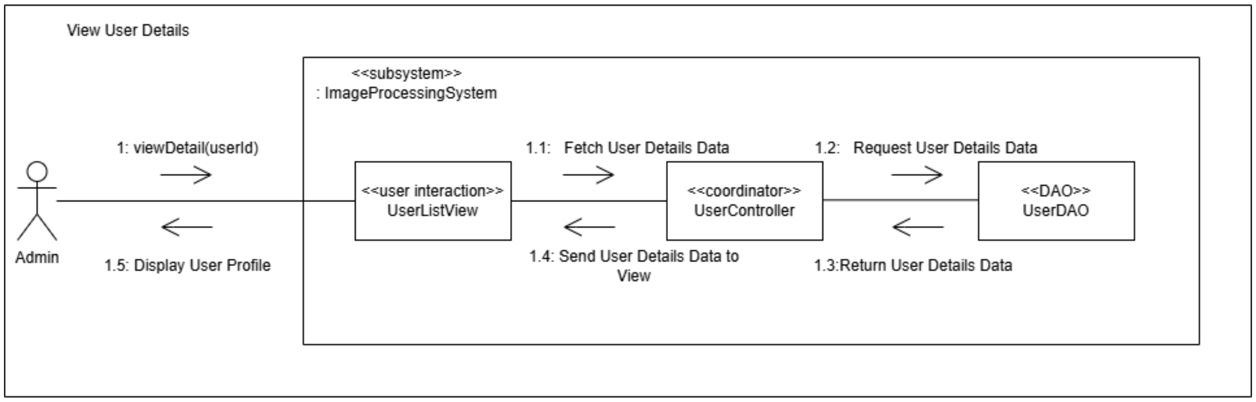

View User Details
<<subsystem>>
: ImageProcessingSystem
1: viewDetail(userId)
1.1: Fetch User Details Data
1.2: Request User Details Data
Admin
<<user interaction>> UserListView
<<coordinator>> UserController
<<DAO>> UserDAO
1.5: Display User Profile
1.4: Send User Details Data to View
1.3:Return User Details Data

Figure 5.2.1.5. View User Communication diagram

Description:

1.viewDetail(userId): The Admin initiates the request to view user details by providing a userId.

1.1.Fetch User Details Data: The request is sent to the User List View to start the process.

1.2.Request User Details Data: The User Controller receives the request and forwards it to the User DAO.

1.3.Return User Details Data: The User DAO retrieves the requested user details from the database and returns the data to the User Controller.

1.4.Send User Details Data to View: The User Controller forwards the retrieved user details to the User List View.

1.5.Display User Profile: The User List View displays the user details to the Admin.

5.2.1.6. View All Users

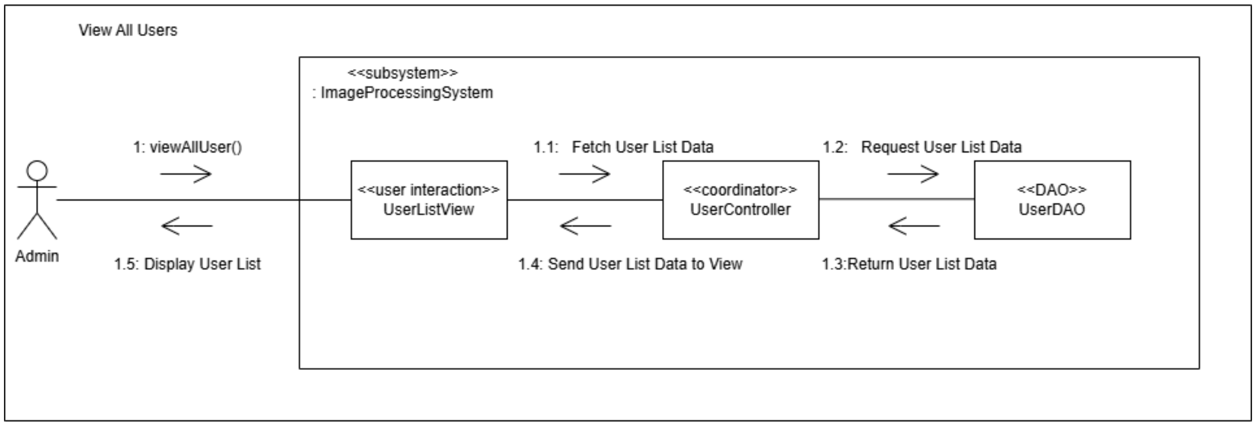


Figure 5.2.1.6. View All User Communication diagram

Description:

1.viewAllUser(): The Admin initiates the request to view all users.

1.1.Fetch User List Data: The request is sent to the User List View, which starts the process.

1.2.Request User List Data: The User Controller receives the request and forwards it to the User DAO.

1.3.Return User List Data: The User DAO retrieves the list of all users from the database and returns it to the User Controller.

1.4.Send User List Data to View: The User Controller forwards the retrieved user list data to the User List View.

1.5.Display User List: The User List View presents the data to the Admin.

5.2.1.7. View Logs

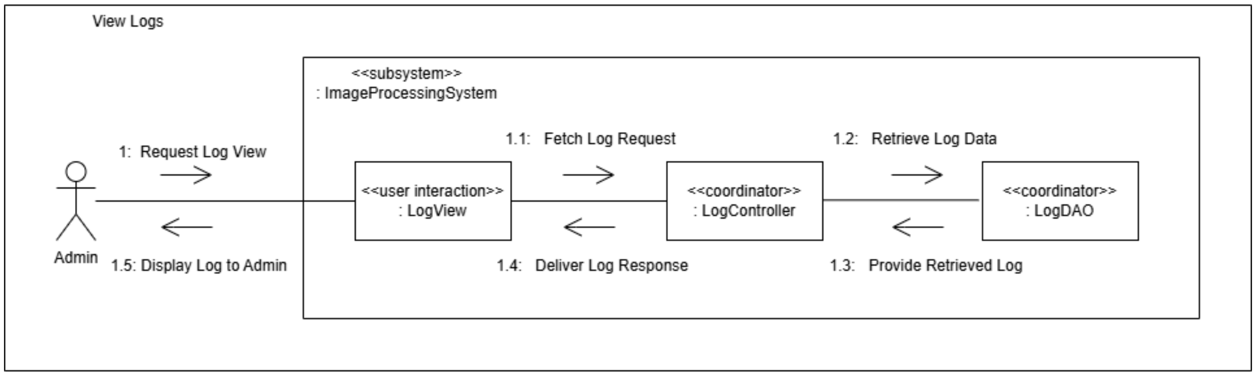


Figure 5.2.1.7. View Logs Communication diagram

Description:

1. Request Log View: The process begins when an admin requests to view the logs from the system.

1.1. Fetch Log Request: The Log View subsystem sends a fetch log request to the Log Controller, which acts as the coordinator for this operation.

1.2. Retrieve Log Data: The Log Controller retrieves the required log data from the system.

1.3 Provide Retrieved Log: After retrieving the logs, the Log Controller sends the data to the Log DAO (Data Access Object), which ensures proper data handling and transfer.

1.4. Deliver Log Response: Finally, the Log DAO delivers the retrieved log back to the Log Controller, which sends the log data to the user interface.

1.5. Display Log to Admin: The logs are displayed to the admin on the user interface for viewing and analysis.

5.2.1.8. View Profile

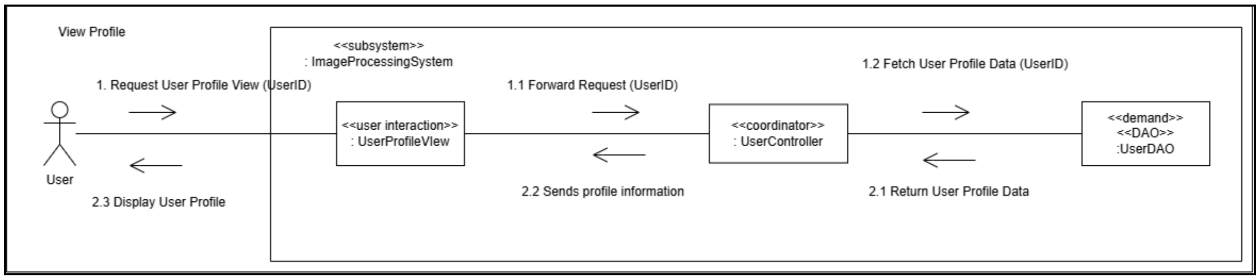


Figure 5.2.1.8. View Profile Communication diagram

Description:

1.1 User Requests User Profile View (UserID)

- The process starts when the user initiates a request to view their profile by providing their **UserID**.
- This request is sent to the **UserProfileView** component.

1.2 UserProfileView Forwards the Request to UserController

- The **UserProfileView** component, which is responsible for handling user interactions, forwards the request along with the **UserID** to the **UserController**.
- This step ensures that the request is properly processed by the backend system.

1.3 UserController Fetches User Profile Data from UserDAO

- The **UserController**, which acts as a coordinator, receives the forwarded request.
- It then sends a request to the **UserDAO** (Data Access Object) to fetch the user's profile data using the provided **UserID**.

1.4 UserDAO Returns User Profile Data

- The **UserDAO**, which is responsible for interacting with the database, retrieves the profile information for the requested **UserID**.
- The profile data is then sent back to the **UserController**.

1.5 UserController Sends Profile Information to UserProfileView

- Once the **UserController** receives the profile data from the **UserDAO**, it forwards the information to the **UserProfileView** component.
- This ensures that the retrieved data is sent back to the user interface.

1.6 UserProfileView Displays User Profile to the User

- The **UserProfileView** component processes the received data and presents it to the user.
- The profile information is displayed on the user interface for the user to view.

5.2.1.9. Edit Profile

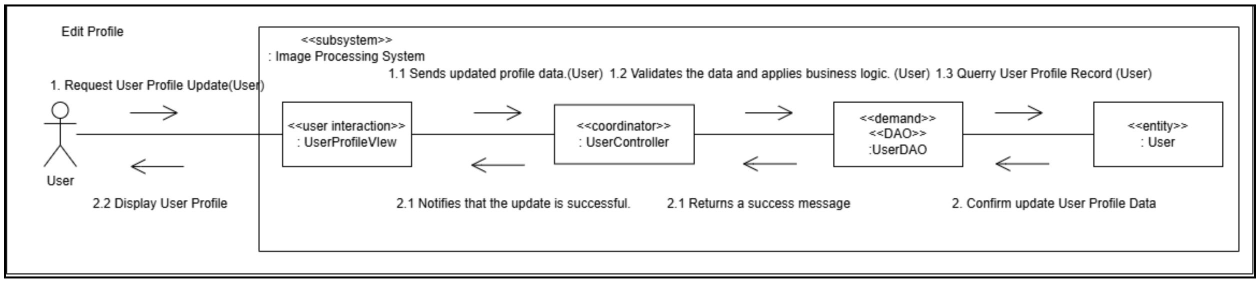

Edit Profile
<<subsystem>>
: Image Processing System
1. Request User Profile Update(User)
1.1 Sends updated profile data.(User)
1.2 Validates the data and applies business logic. (User)
1.3 Querry User Profile Record (User)
User
<<user interaction>> : UserProfileView
<<coordinator>> : UserController
<<demand>> <<DAO>> :UserDAO
<<entity>> : User
2.2 Display User Profile
2.1 Notifies that the update is successful.
2.1 Returns a success message
2. Confirm update User Profile Data

Figure 5.2.1.9. Edit Profile Communication diagram

Description:

1.1 User Requests User Profile Update

- The user initiates a request to update their profile by providing the modified profile data.
- This request is sent to the **UserProfileView**, which is responsible for handling user interactions.

1.2 UserProfileView Sends Updated Profile Data to UserController

- The **UserProfileView** forwards the request to the **UserController**.
- This step ensures that the updated user data is sent for further validation and processing.

1.3 UserController Processes the Update Request

- The **UserController**, acting as a coordinator, performs the following actions:
  - **1.1 Sends Updated Profile Data (User):** The controller receives the updated profile data from the UI.
  - **1.2 Validates the Data and Applies Business Logic (User):** The system checks if the provided data meets validation requirements and applies necessary business rules.

  - **1.3 Queries the User Profile Record (User):** The **UserController** requests the existing profile record from the **UserDAO** for comparison and update.

1.4 UserDAO Queries and Updates the User Profile

- The **UserDAO** component is responsible for interacting with the database. It performs the following actions:
  - **Queries the existing user profile data** to fetch the current record.
  - **Updates the user profile** with the new information provided.
- The updated user data is then saved in the database.

1.5 UserDAO Confirms the Profile Update

- After successfully updating the profile, the **UserDAO** returns a confirmation message to the **UserController**.

1.6 UserController Returns a Success Message

- Once the profile update is confirmed, the **UserController** sends a success message back to the **UserProfileView**

1.7 UserProfileView Notifies the User of the Successful Update

- The **UserProfileView** component notifies the user that their profile update was successful.

1.8 UserProfileView Displays the Updated Profile

- Finally, the updated user profile information is displayed to the user on the interface.

5.2.1.10. View online file storage.

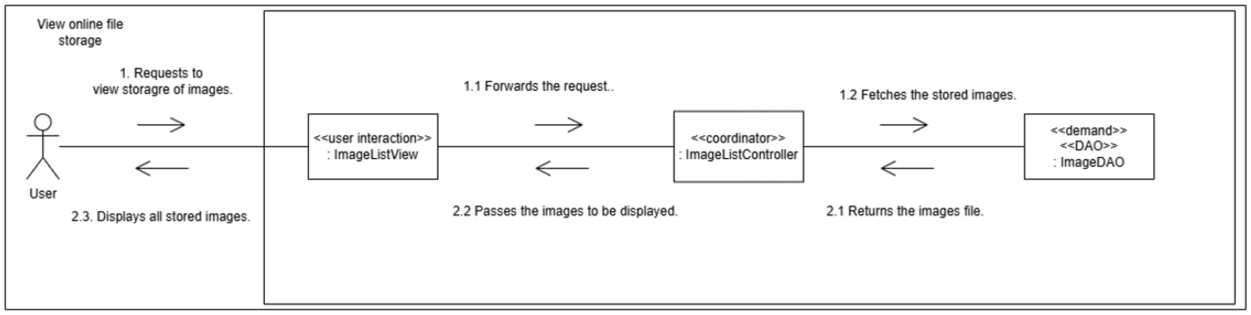

View online file storage
1. Requests to view storagre of images.
User
2.3. Displays all stored images.
<<user interaction>> : ImageListView
1.1 Forwards the request..
2.2 Passes the images to be displayed.
<<coordinator>> : ImageListController
1.2 Fetches the stored images.
2.1 Returns the images file.
<<demand>> <<DAO>> : ImageDAO

Figure 5.2.1.10. View Online File Storage Communication diagram

Description:

1.1 User Requests to View Stored Images

- The user initiates a request to view stored images in online file storag**e.**
- This request is sent to the **ImageListView**, which handles user interactions**.**

1.2 ImageListView Forwards the Request

- The **ImageListView** component forwards the user's request to the **ImageListController.**
- This ensures that the request is processed before accessing stored images.

1.3 ImageListController Fetches the Stored Images

- The **ImageListController**, acting as a coordinator, interacts with the **ImageDAO** to retrieve the stored images.
- The following steps occur:
  + **1.2 Fetches the stored images:** The **ImageDAO** queries the database or storage system to retrieve the requested images.
  + **2.1 Returns the image file:** The stored images are sent back to the **ImageDAO**, which then returns them to the **ImageListController**.

1.4 ImageListController Passes the Images to ImageListView

- Once the **ImageListController** receives the images from **ImageDAO**, it forwards them to **ImageListView** for display.

1.5 Images are Displayed to the User

- The **ImageListView** displays all retrieved images to the user.

5.2.1.11. Delete Image File.

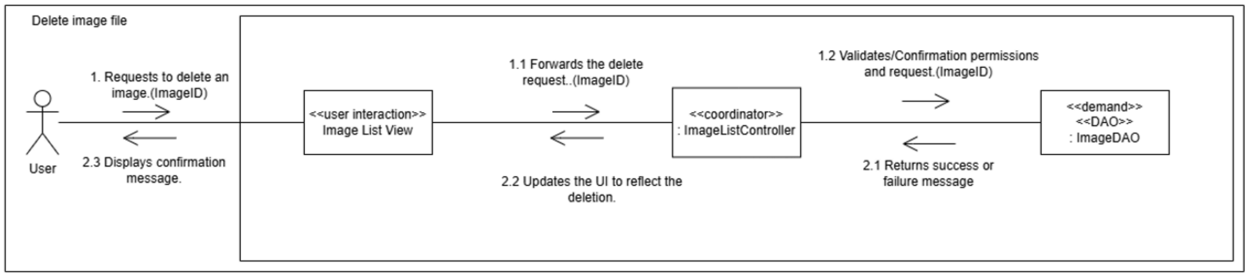

Delete image file
1. Requests to delete an image.(ImageID)
2.3 Displays confirmation message.
User
<<user interaction>> Image List View
1.1 Forwards the delete request..(ImageID)
2.2 Updates the UI to reflect the deletion.
<<coordinator>> : ImageListController
1.2 Validates/Confirmation permissions and request.(ImageID)
2.1 Returns success or failure message
<<demand>> <<DAO>> : ImageDAO

Figure 5.2.1.11. Delete Image File Communication diagram

Description:

1.1 User Requests to Delete an Image

- The user initiates a request to delete an image, specifying an **ImageID**.
- This request is sent to the **ImageListView**, which handles user interactions.

1.2 ImageListView Forwards the Delete Request

- The **ImageListView** component forwards the delete request along with the **ImageID** to the **ImageListController**.
- This ensures that the request is validated and processed before deletion.

1.3 ImageListController Validates and Processes the Request

- The **ImageListController**, acting as a coordinator, performs the following tasks:
  - **1.2 Validates permissions and the delete request:** Ensures that the user has the necessary permissions to delete the image.
  - **Forwarded the request to the ImageDAO** for deletion.

2.1 ImageDAO Deletes the Image and Returns a Response

- The **ImageDAO** (Data Access Object) handles the actual deletion process:
    - **2.1 Deletes the image from storage.**
    - **Returns a success or failure message** to the **ImageListController** based on the operation's outcome.

2.2 ImageListController Updates the UI

- Once the **ImageListController** receives the response from **ImageDAO**, it:
    - **2.2 Updates the UI** to reflect that the image has been deleted.
    - **Send the confirmation message** to **ImageListView**.

2.3 ImageListView Displays the Confirmation Message

- Finally, the **ImageListView** displays a confirmation message to inform the user that the image has been successfully deleted (or if an error occurred).

5.2.2. Integrated Communication Diagrams

5.2.2.1. Sign up

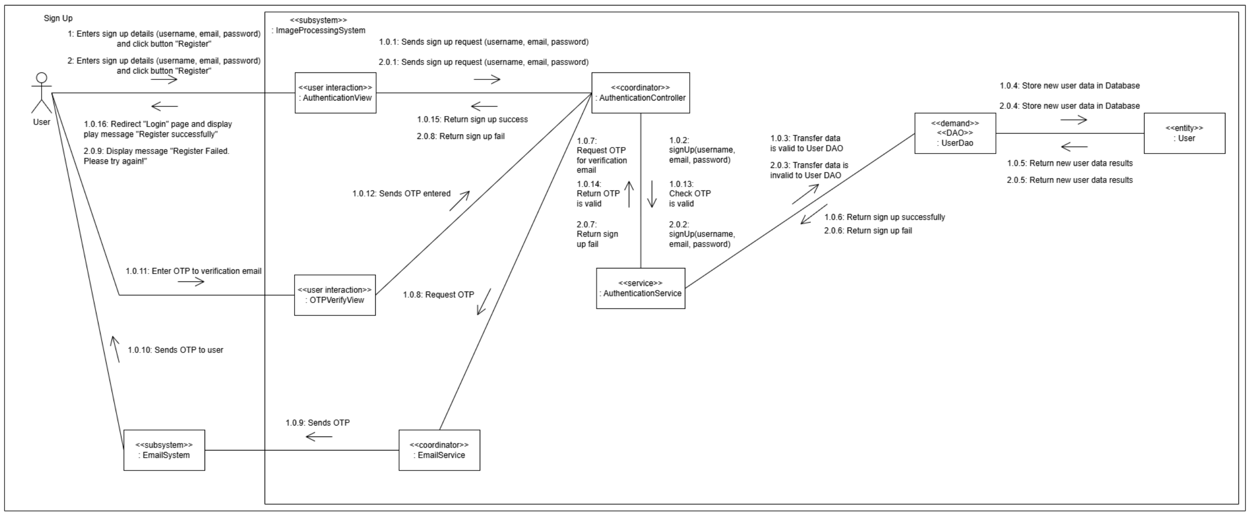

Sign Up
1: Enters sign up details (username, email, password) and click button "Register"
2: Enters sign up details (username, email, password) and click button "Register"
User
1.0.16: Redirect "Login" page and display play message "Register successfully"
2.0.9: Display message "Register Failed. Please try again!"
<<subsystem>> : ImageProcessingSystem
1.0.1: Sends sign up request (username, email, password)
2.0.1: Sends sign up request (username, email, password)
<<user interaction>> : AuthenticationView
1.0.15: Return sign up success
2.0.8: Return sign up fail
<<coordinator>> : AuthenticationController
1.0.7: Request OTP for verification email
1.0.14: Return OTP is valid
2.0.7: Return sign up fail
1.0.2: signUp(username, email, password)
1.0.13: Check OTP is valid
2.0.2: signUp(username, email, password)
<<service>> : AuthenticationService
1.0.3: Transfer data is valid to User DAO
2.0.3: Transfer data is invalid to User DAO
1.0.6: Return sign up successfully
2.0.6: Return sign up fail
<<demand>> <<DAO>> : UserDao
1.0.4: Store new user data in Database
2.0.4: Store new user data in Database
1.0.5: Return new user data results
2.0.5: Return new user data results
<<entity>> : User
1.0.12: Sends OTP entered
1.0.11: Enter OTP to verification email
<<user interaction>> : OTPVerifyView
1.0.8: Request OTP
1.0.10: Sends OTP to user
1.0.9: Sends OTP
<<subsystem>> : EmailSystem
<<coordinator>> : EmailService

Figure 5.2.2.1. Sign Up Communication diagram

Description:

-True thread:

1: Enters sign up details (username, email, password) and click button "Register"

1.0.1: Sends sign up request (username, email, password)

1.0.2: signUp(username, email, password)

1.0.3: Transfer data is valid to User DAO

1.0.4: Store new user data in Database

1.0.5: Return new user data results

1.0.6: Return sign up successfully

1.0.7: Request OTP for verification email

1.0.8: Request OTP

1.0.9: Sends OTP

1.0.10: Sends OTP to user

1.0.11: Enter OTP to verification email

1.0.12: Sends OTP entered

1.0.13: Check OTP is valid

1.0.14: Return OTP is valid

1.0.15: Return sign up success

1.0.16: Redirect "Login" page and display play message "Register successfully"

- False thread:

2: Enters sign up details (username, email, password) and click button "Register"

2.0.1: Sends sign up request (username, email, password)

2.0.2: signUp(username, email, password)

2.0.3: Transfer data is invalid to User DAO

2.0.4: Store new user data in Database

2.0.5: Return new user data results

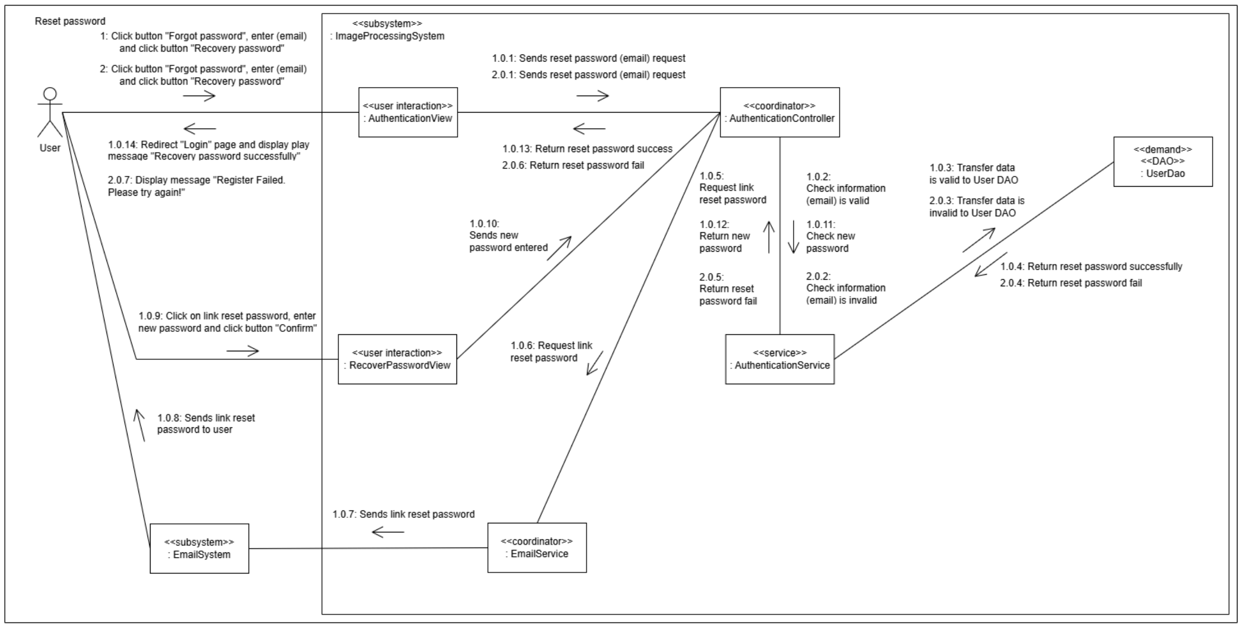
2.0.6: Return sign up fail

2.0.7: Return sign up fail

2.0.8: Return sign up fail

2.0.9: Display message "Register Failed. Please try again!"

5.2.2.2. Reset password



Reset password
1: Click button "Forgot password", enter (email) and click button "Recovery password"
2: Click button "Forgot password", enter (email) and click button "Recovery password"
User
<<subsystem>> : ImageProcessingSystem
<<user interaction>> : AuthenticationView
<<coordinator>> : AuthenticationController
<<demand>> <<DAO>> : UserDao
<<user interaction>> : RecoverPasswordView
<<service>> : AuthenticationService
<<subsystem>> : EmailSystem
<<coordinator>> : EmailService
1.0.1: Sends reset password (email) request
2.0.1: Sends reset password (email) request
1.0.13: Return reset password success
2.0.6: Return reset password fail
1.0.14: Redirect "Login" page and display play message "Recovery password successfully"
2.0.7: Display message "Register Failed. Please try again!"
1.0.5: Request link reset password
1.0.12: Return new password
2.0.5: Return reset password fail
1.0.2: Check information (email) is valid
1.0.11: Check new password
2.0.2: Check information (email) is invalid
1.0.3: Transfer data is valid to User DAO
2.0.3: Transfer data is invalid to User DAO
1.0.4: Return reset password successfully
2.0.4: Return reset password fail
1.0.10: Sends new password entered
1.0.9: Click on link reset password, enter new password and click button "Confirm"
1.0.6: Request link reset password
1.0.8: Sends link reset password to user
1.0.7: Sends link reset password

Figure 5.2.2.2. Reset Password Communication diagram

Description:

- True thread:

1: Click button "Forgot password", enter (email) and click button "Recovery password"

1.0.1: Sends reset password (email) request

1.0.2: Check information (email) is valid

1.0.3: Transfer data is valid to User DAO

1.0.4: Return reset password successfully

1.0.5: Request link reset password

1.0.6: Request link reset password

1.0.7: Sends link reset password

1.0.8: Sends link reset password to user

1.0.9: Click on link reset password, enter new password and click button "Confirm"

1.0.10: Sends new password entered

1.0.11: Check new password

1.0.12: Return new password

1.0.13: Return reset password success

1.0.14: Redirect "Login" page and display play message "Recovery password successfully"

-False thread:

2: Click button "Forgot password", enter (email) and click button "Recovery password"

2.0.1: Sends reset password (email) request

2.0.2: Check information (email) is invalid

2.0.3: Transfer data is invalid to User DAO

2.0.4: Return reset password fail

2.0.5: Return reset password fail

2.0.6: Return reset password fail

2.0.7: Display message "Register Failed. Please try again!"

5.2.2.3: Sign out

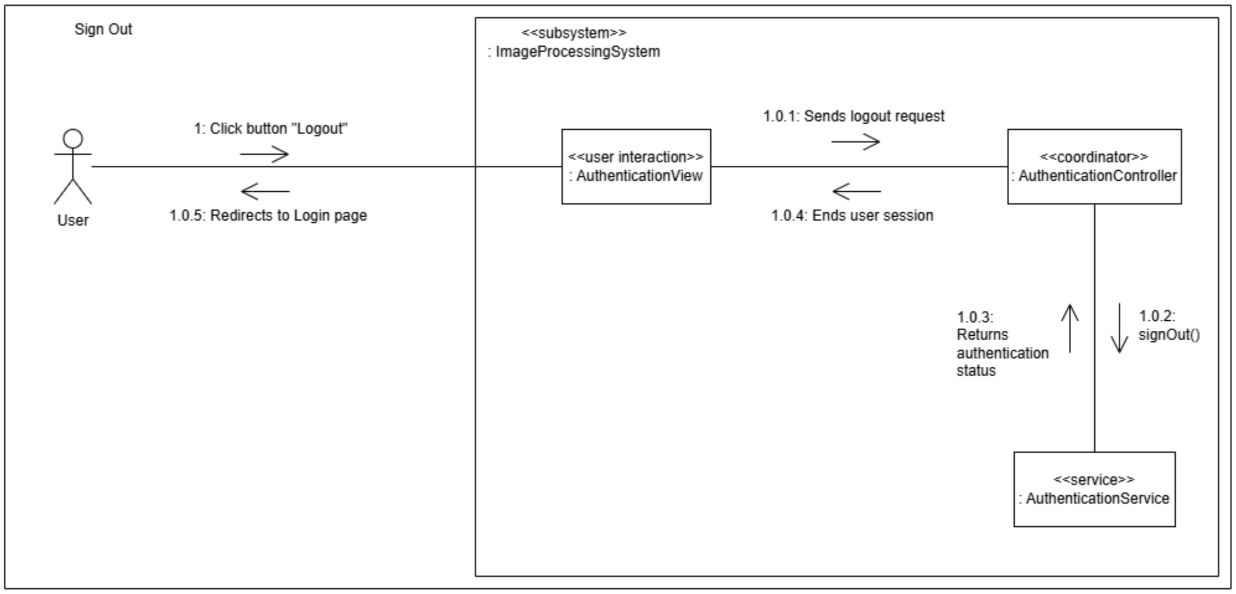


Figure 5.2.2.3. Sign Out Communication diagram

Description:

1: Click button "Logout"

1.0.1: Sends logout request

1.0.2: Logout

1.0.3: Returns authentication status

1.0.4: Ends user session

1.0.5: Redirects to Login page

5.2.2.4: Sign In

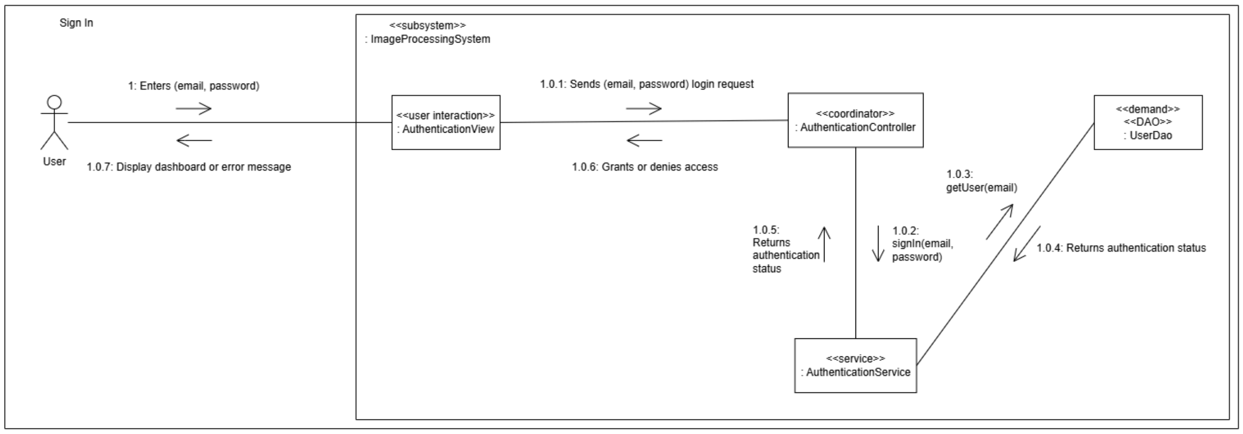


Figure 5.2.2.4. Sign In Communication diagram

Description:

1: Enters (email, password)

1.0.1: Sends (email, password) login request

1.0.2: signIn(email, password)

1.0.3: getUser(email)

1.0.4: Returns authentication status

1.0.5: Returns authentication status

1.0.6: Grants or denies access

1.0.7: Display dashboard or error message

5.2.3. Concurrent Communication Diagrams

5.2.3.1. Convert image format

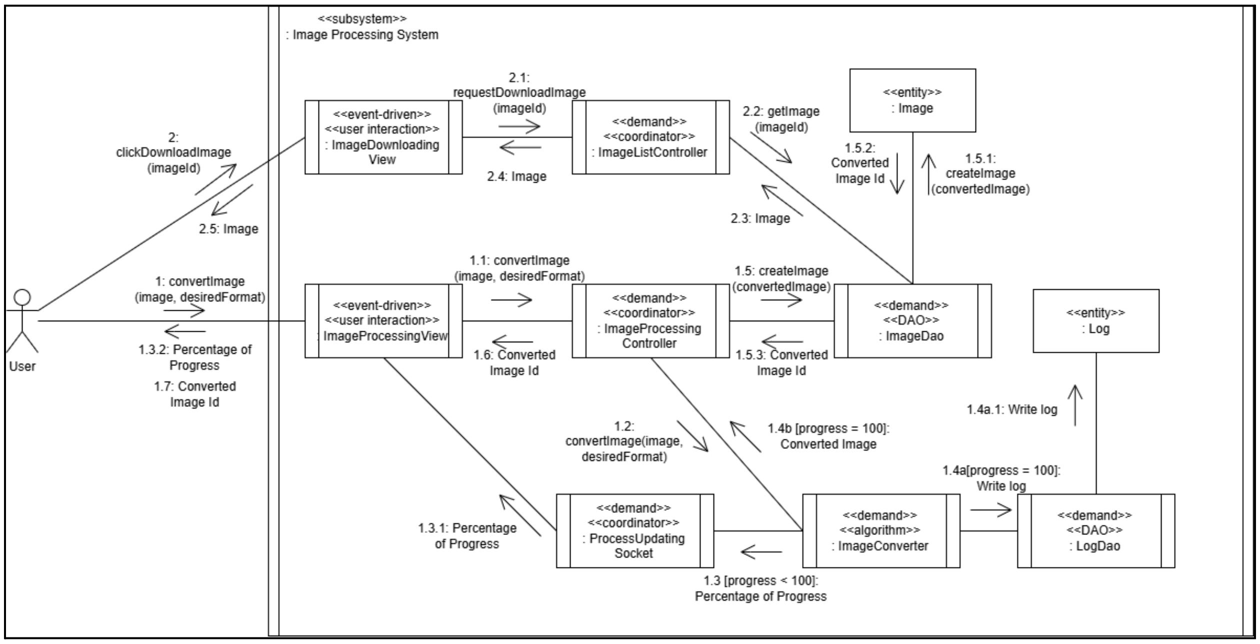

User
<<subsystem>>
: Image Processing System
<<event-driven>>
<<user interaction>>
: ImageDownloading View
<<demand>>
<<coordinator>>
: ImageListController
<<entity>>
: Image
<<event-driven>>
<<user interaction>>
: ImageProcessingView
<<demand>>
<<coordinator>>
: ImageProcessing Controller
<<demand>>
<<DAO>>
: ImageDao
<<entity>>
: Log
<<demand>>
<<coordinator>>
: ProcessUpdating Socket
<<demand>>
<<algorithm>>
: ImageConverter
<<demand>>
<<DAO>>
: LogDao
2: clickDownloadImage (imageId)
2.5: Image
2.1: requestDownloadImage (imageId)
2.4: Image
2.2: getImage (imageId)
2.3: Image
1.5.2: Converted Image Id
1.5.1: createImage (convertedImage)
1: convertImage (image, desiredFormat)
1.3.2: Percentage of Progress
1.7: Converted Image Id
1.1: convertImage (image, desiredFormat)
1.6: Converted Image Id
1.5: createImage (convertedImage)
1.5.3: Converted Image Id
1.4a.1: Write log
1.2: convertImage(image, desiredFormat)
1.4b [progress = 100]: Converted Image
1.4a[progress = 100]: Write log
1.3.1: Percentage of Progress
1.3 [progress < 100]: Percentage of Progress

Figure 5.2.3.1. Communication diagram for use-case "Convert image format"

Description:

1: The user uploads an image and initiates an image conversion request with the desired format.

1.1: The request is sent to the *ImageProcessingController*.

1.2: The controller forwards the request to the *ImageConverter*.

1.3: The *ImageConverter* continuously reports the progress of the conversion via the *ProcessUpdatingSocket*.

1.3.1 and 1.3.2: Progress percentage is sent back to the *ImageProcessingView*, where the user can see.

1.4a and 1.4a.1: When the conversion is complete, the *ImageConverter* writes logs to the database via the *LogDao.*

1.4b: The *ImageConverter* also returns the converted image to the *ImageProcessingController*.

1.5, 1.5.1, 1.5.2 and 1.5.3: The *ImageProcessingController* then saves the image to the database via the *ImageDao.* The *ImageDao* then returns the image Id to the *ImageProcessingController.*

1.6 and 1.7: The image Id is then returned to the *ImageProcessingView*, where the user can perform further interactions.

2: The user requests to download the converted image using its Id.

2.1: The Image *DownloadingView* sends the request to the *ImageListController*.

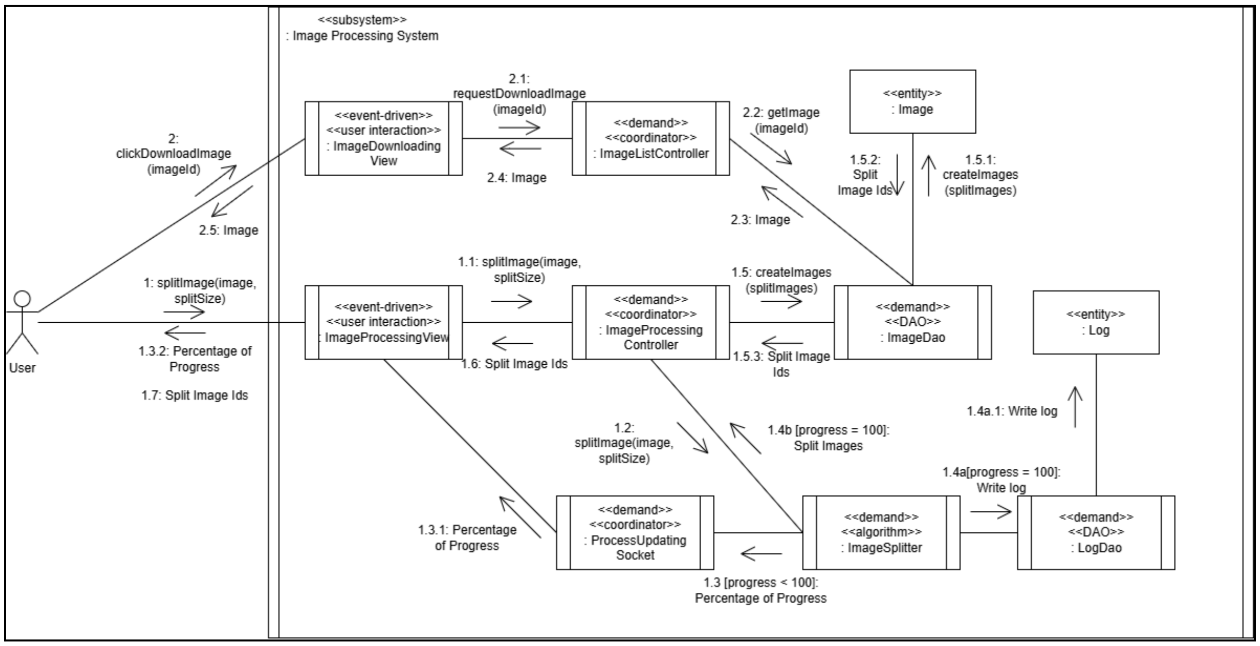
2.2: The *ImageListController* retrieves the converted image from the *ImageDao*.

2.3: The image is returned to the *ImageListController*.

2.4: The image is returned to the *DownloadingView*.

2.5: The user can now preview or save the converted image.

5.2.3.2. Split an image



<<subsystem>>
: Image Processing System
User
2: clickDownloadImage (imageId)
2.5: Image
1: splitImage(image, splitSize)
1.3.2: Percentage of Progress
1.7: Split Image Ids
<<event-driven>> <<user interaction>> : ImageDownloading View
2.1: requestDownloadImage (imageId)
2.4: Image
<<demand>> <<coordinator>> : ImageListController
2.2: getImage (imageId)
2.3: Image
<<entity>> : Image
1.5.2: Split Image Ids
1.5.1: createImages (splitImages)
<<event-driven>> <<user interaction>> : ImageProcessingView
1.1: splitImage(image, splitSize)
1.6: Split Image Ids
<<demand>> <<coordinator>> : ImageProcessing Controller
1.5: createImages (splitImages)
1.5.3: Split Image Ids
<<demand>> <<DAO>> : ImageDao
<<entity>> : Log
1.4a.1: Write log
1.2: splitImage(image, splitSize)
1.4b [progress = 100]: Split Images
1.4a[progress = 100]: Write log
1.3.1: Percentage of Progress
<<demand>> <<coordinator>> : ProcessUpdating Socket
<<demand>> <<algorithm>> : ImageSplitter
<<demand>> <<DAO>> : LogDao
1.3 [progress < 100]: Percentage of Progress

Figure 5.2.3.2. Communication diagram for use-case "Split an image"

Description:

1: The user uploads an image and initiates an image split request with the desired split size.

1.1: The request is sent to the *ImageProcessingController*.

1.2: The controller forwards the request to the *ImageSplitter*.

1.3: The *ImageSplitter* continuously reports the progress of the conversion via the *ProcessUpdatingSocket*.

1.3.1 and 1.3.2: Progress percentage is sent back to the *ImageProcessingView*, where the user can see.

1.4a and 1.4a.1: When the splitting is complete, the *ImageSplitter* writes logs to the database via the *LogDao.*

1.4b: The *ImageSplitter* also returns the split images to the *ImageProcessingController*.

1.5, 1.5.1, 1.5.2 and 1.5.3: The *ImageProcessingController* then saves the images to the database via the *ImageDao.* The *ImageDao* then returns the image Ids to the *ImageProcessingController.*

2: The user requests to download the split images using their Ids.

2.1: The *ImageDownloadingView* sends the request to the *ImageListController*.

2.2: The *ImageListController* retrieves the images from the *ImageDao*.

2.3: The image is returned to the *ImageListController*.

2.4: The image is returned to the *DownloadingView*.

2.5: The user can now preview or save the split images.

5.2.3.3. Merge images

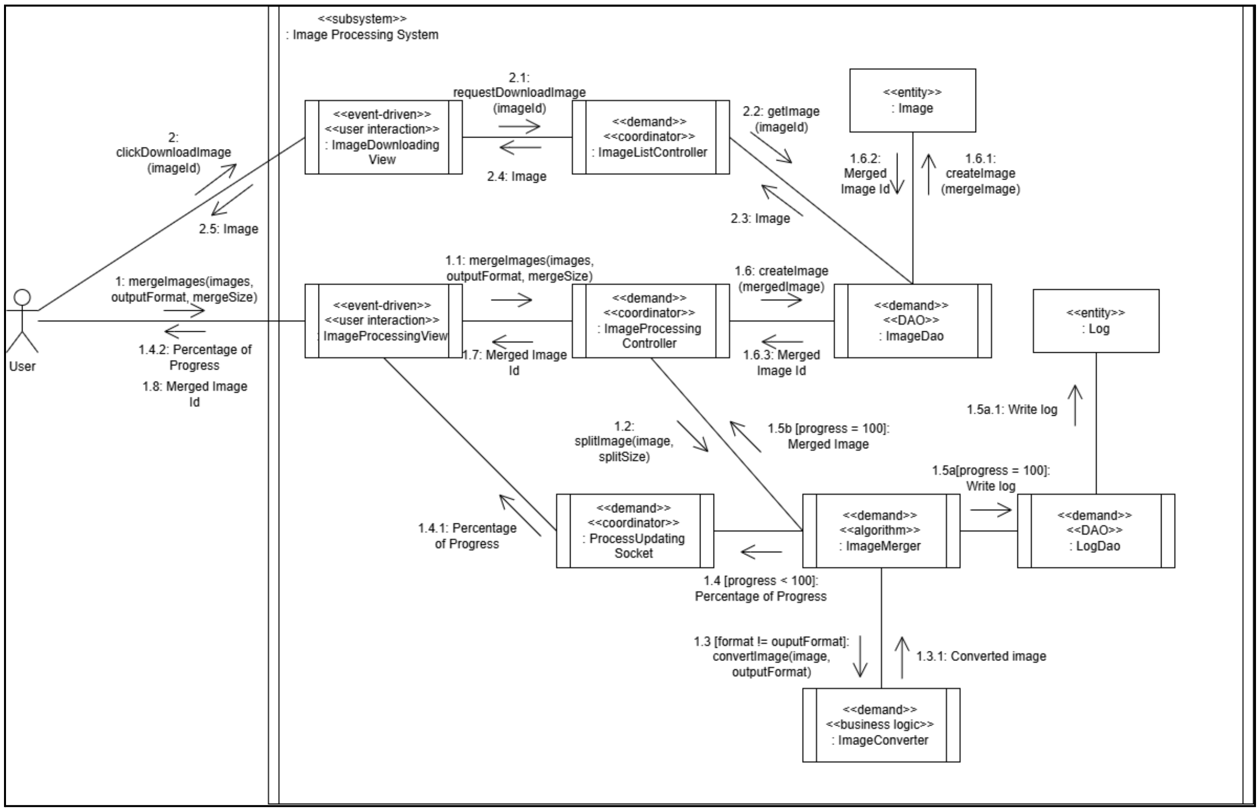

<<subsystem>>
: Image Processing System
User
2: clickDownloadImage (imageId)
2.5: Image
2.1: requestDownloadImage (imageId)
2.4: Image
<<event-driven>> <<user interaction>> : ImageDownloading View
<<demand>> <<coordinator>> : ImageListController
2.2: getImage (imageId)
2.3: Image
<<entity>> : Image
1.6.2: Merged Image Id
1.6.1: createImage (mergeImage)
1: mergeImages(images, outputFormat, mergeSize)
1.4.2: Percentage of Progress
1.8: Merged Image Id
<<event-driven>> <<user interaction>> : ImageProcessingView
1.1: mergeImages(images, outputFormat, mergeSize)
1.7: Merged Image Id
<<demand>> <<coordinator>> : ImageProcessing Controller
1.6: createImage (mergedImage)
1.6.3: Merged Image Id
<<demand>> <<DAO>> : ImageDao
<<entity>> : Log
1.5a.1: Write log
1.2: splitImage(image, splitSize)
1.5b [progress = 100]: Merged Image
1.5a[progress = 100]: Write log
1.4.1: Percentage of Progress
<<demand>> <<coordinator>> : ProcessUpdating Socket
<<demand>> <<algorithm>> : ImageMerger
<<demand>> <<DAO>> : LogDao
1.4 [progress < 100]: Percentage of Progress
1.3 [format != ouputFormat]: convertImage(image, outputFormat)
1.3.1: Converted image
<<demand>> <<business logic>> : ImageConverter

Figure 5.2.3.3. Communication diagram for use-case "Merge images"

Description:

1: The user requests to merge multiple images into a single image with the specified format and size.

1.1: The *ImageProcessingView* sends the request to the *ImageProcessingController*.

1.2: The controller forwards the request to the *ImageMerger*.

1.3 and 1.3.1: If the images' format differs from the output format, they are first converted using the *ImageConverter*, then the image is returned to the *ImageMerger*.

1.4: The *ImageMerger* continuously reports the progress of the conversion via the *ProcessUpdatingSocket*.

1.4.1 and 1.4.2: Progress percentage is sent back to the *ImageProcessingView*, where the user can see.

1.5a and 15.a.1: Once merging is complete, the *ImageMerger* writes logs to the database via the *LogDao.*

1.5b: The *ImageMerger* also returns the merged image to the *ImageProcessingController*.

1.6, 1.6.1, 1.6.2, 1.6.3: The *ImageProcessingController* then saves the images to the database via the *ImageDao.* The *ImageDao* then returns the image Id to the *ImageProcessingController.*

2: The user requests to download the merged image using its Id.

2.1: The *ImageDownloadingView* sends the request to the *ImageListController*.

2.2: The *ImageListController* retrieves the images from the *ImageDao*.

2.3: The image is returned to the *ImageListController*.

2.4: The image is returned to the *DownloadingView*.

2.5: The user can now preview or save the merged image.

6. Design Modeling

6.1. List of Architecture Patterns which be selected

The Image Processing System's architecture follows a combination of two primary design patterns:

Client-Server Pattern

- The system is structured using the client-server architecture, where the clients (Web Clients) interact with the server-side services to process images, manage user authentication, and handle other business logic.
- The client layer consists of Web UI components that allow users (Admin & End Users) to interact with the system.S
- The server layer is responsible for handling business logic, processing images, managing user authentication, and interacting with external services like the Email System.
- Benefits:
    - Enhances scalability by allowing multiple clients to connect to a centralized server.
    - Improves maintainability by separating concerns between client and server.
    - Supports multi-platform access with a single server-side implementation.
    - Increases security by centralizing sensitive data and authentication logic.
    - Enables seamless updates without requiring client-side modifications.

Layered Architecture Pattern

- The system is organized into multiple layers, each with a distinct responsibility, ensuring modularity, separation of concerns, and maintainability between layers.
- The layers of the system will be described in the figure *6.2.1.2* below.

6.2. Architecture in three views

6.2.1. Architecture in Static View

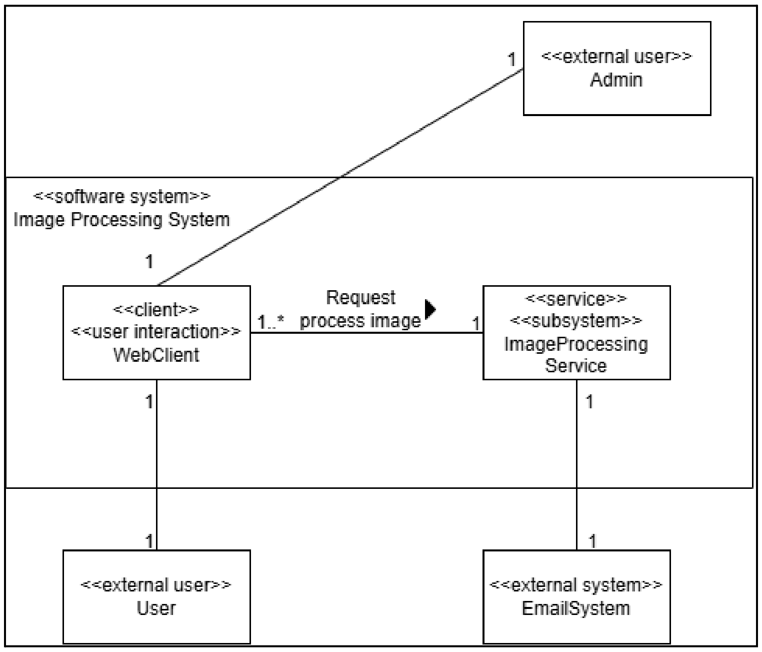


Figure 6.2.1.1. Architecture in Static View

The diagram represents the static view of the Image Processing System, highlighting its key components, interactions, and external dependencies.

6.2.1.1. External Users & Systems

- **Admin**: An external user who can manage and oversee the image processing system.
- **User**: A general external user who interacts with the system to process images, including converting formats, splitting and merging.
- **EmailSystem**: An external system integrated with the Image Processing System for notifications and email-based authentication methods.

6.2.1.2. Image Processing System Components

- **WebClient (Client Component)**:
    - Acts as the user interface for both Admins and Users.
    - Facilitates interaction with the Image Processing Service by sending requests.
    - All of the software UI views operate on WebClient
- **ImageProcessing Service (Service Component)**:
    - Handles image processing requests received from the WebClient.
    - Processes and manages images before storing or forwarding them to the necessary systems.

6.2.1.3. Interaction & Communication

- **Users (Admin & General Users) interact with the WebClient**, which serves as the primary entry point into the system.
- **WebClient sends requests to process images to the ImageProcessing Service**. The service can handle multiple clients.
- **The ImageProcessing Service may interact with the Email System** to send confirmation emails or notifications.

6.2.1.4. Internal Layered Architecture of the Software System

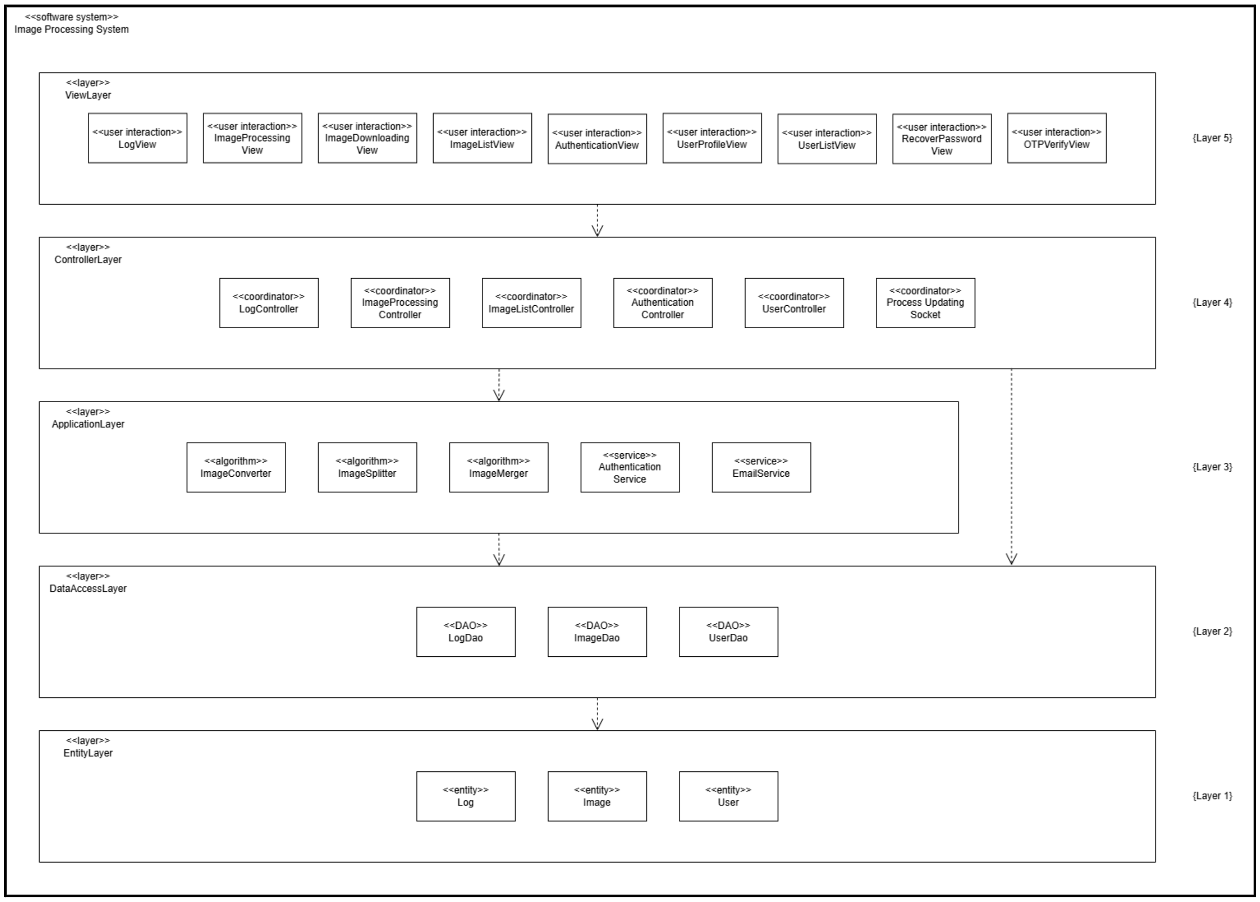

<<software system>> Image Processing System
<<layer>> ViewLayer
<<user interaction>> LogView
<<user interaction>> ImageProcessing View
<<user interaction>> ImageDownloading View
<<user interaction>> ImageListView
<<user interaction>> AuthenticationView
<<user interaction>> UserProfileView
<<user interaction>> UserListView
<<user interaction>> RecoverPassword View
<<user interaction>> OTPVerifyView
{Layer 5}
<<layer>> ControllerLayer
<<coordinator>> LogController
<<coordinator>> ImageProcessing Controller
<<coordinator>> ImageListController
<<coordinator>> Authentication Controller
<<coordinator>> UserController
<<coordinator>> Process Updating Socket
{Layer 4}
<<layer>> ApplicationLayer
<<algorithm>> ImageConverter
<<algorithm>> ImageSplitter
<<algorithm>> ImageMerger
<<service>> Authentication Service
<<service>> EmailService
{Layer 3}
<<layer>> DataAccessLayer
<<DAO>> LogDao
<<DAO>> ImageDao
<<DAO>> UserDao
{Layer 2}
<<layer>> EntityLayer
<<entity>> Log
<<entity>> Image
<<entity>> User
{Layer 1}

Figure 6.2.1.2. Internal layered architecture of the software system

The Image Processing System is structured into five layers. Each layer has a specific role in handling user interactions, business logic, data access, and entity management.

Layer 1: Entity Layer

- The foundation of the system, consisting of core data entities:
    - **Log**: Represents system logs.
    - **Image**: Represents image-related data.
    - **User**: Represents user-related data.

Layer 2: Data Access Layer

- Handles database operations, providing an interface for data storage and retrieval.

- Contains DAO (Data Access Object) components:
  - **Log DAO**: Manages log data.
  - **Image DAO**: Manages image-related database operations.
  - **User DAO**: Manages user data.

Layer 3: Application Layer

- Implements core functionalities of the system.
- Includes algorithm components:
  - **Image Converter**: Converts images into different formats.
  - **Image Splitter**: Splits images into multiple parts.
  - **Image Merger**: Merges multiple images into one.
- Includes service components:
  - **Authentication Service**: Handles user authentication and security.
  - **Email Service**: Manages email-related operations, such as notifications.

Layer 4: Controller Layer

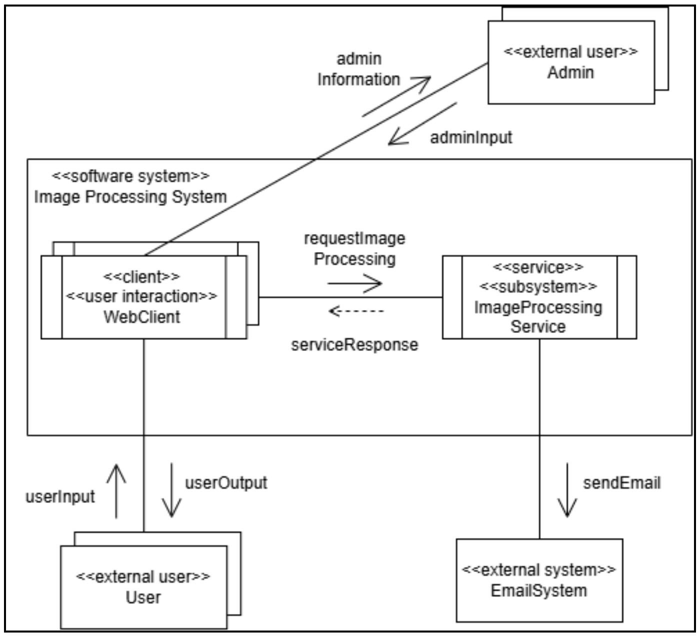
- Acts as an intermediary between the View and Business Logic layers, and View and DAO layers.
- Contains controller components (coordinators):
  - **Log Controller**: Handles requests for log-related data.
  - **Image Processing Controller**: Handles image processing requests.
  - **Image List Controller**: Handles requests for listing and downloading images.
  - **Authentication Controller**: Handles authentication requests.
  - **User Controller**: Handles requests for user-related data.
  - **Process Updating Socket**: Handles real-time updates for ongoing processes.

Layer 5: View Layer

- Provides user interfaces for system interaction.
- Contains various views:
  - **Log View**: Displays system logs.
  - **Image Processing View**: Allows users to process images.
  - **Image Downloading View**: Enables users to download processed images.
  - **Image List View**: Shows lists of images.
  - **Authentication View**: Manages user authentication.

  - **User Profile View**: Displays user profile details.
  - **User List View**: Shows a list of users.
  - **RecoverPasswordView**: Allows users to reset their authentication password.
  - **OTPVerifyView**: Allows users to input OTP and verify
- The views in the view layer operates on the WebClient, described in figure 6.2.1.1

6.2.2. Architecture in Dynamic View



<<external user>> Admin
admin Information
adminInput
<<software system>> Image Processing System
<<client>> <<user interaction>> WebClient
requestImage Processing
serviceResponse
<<service>> <<subsystem>> ImageProcessing Service
userInput
userOutput
sendEmail
<<external user>> User
<<external system>> EmailSystem

Figure 6.2.2.1. Architecture in dynamic view

This diagram illustrates the interaction between users, admins, web clients, an image processing service, and an external email system. It represents a high-level **component view** of how different entities communicate within the system.

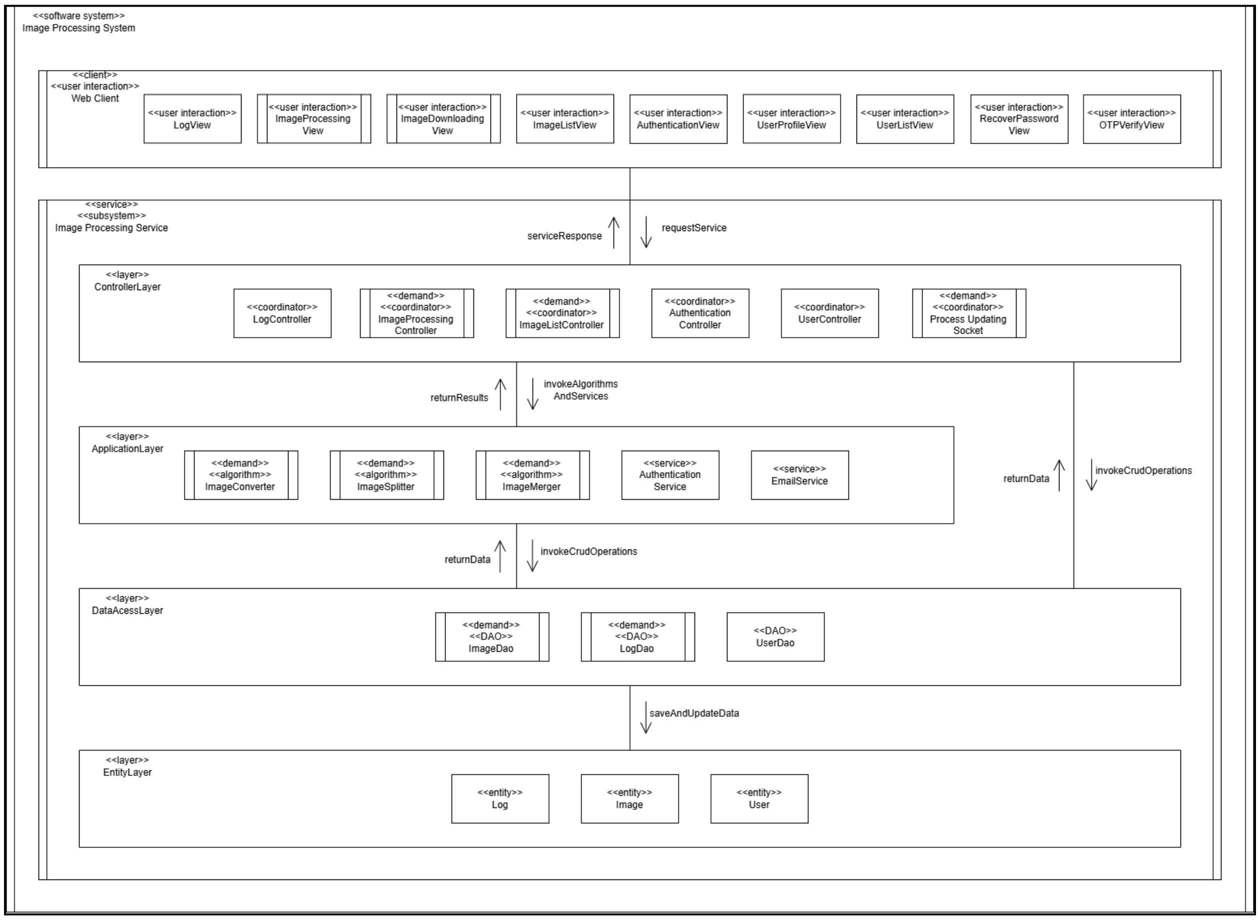
Key Components & Their Roles:

1. **External Users**
   - **User**: Regular user accessing the system via the web client.
   - **Admin**: Has elevated privileges to manage users or system settings.
2. **Web Client**
   - Front-end web interface allowing users to interact with and display data to them.
   - Interacts with the **Image Processing Service** for backend operations.

3. **Image Processing Service**
    - The core processing unit that handles image-related operations: conversion, splitting, merging, and storing processed images.
    - Also handles user data and system data management.
    - Sends notifications to users through the **Email System**.
4. **External System**
    - **Email System**: A third-party email service responsible for sending automated emails (e.g., notifications, status updates, authentication emails).

System Flow:

1. Users (User/Admin) interact with the Web Client.
2. The Web Client sends image processing requests to the Image Processing Service.
3. The Image Processing Service executes image operations and may trigger email notifications.
4. The Email System sends relevant notifications to users.



<<software system>> Image Processing System
<<client>> <<user interaction>> Web Client
<<user interaction>> LogView
<<user interaction>> ImageProcessing View
<<user interaction>> ImageDownloading View
<<user interaction>> ImageListView
<<user interaction>> AuthenticationView
<<user interaction>> UserProfileView
<<user interaction>> UserListView
<<user interaction>> RecoverPassword View
<<user interaction>> OTPVerifyView
<<service>> <<subsystem>> Image Processing Service
serviceResponse
requestService
<<layer>> ControllerLayer
<<coordinator>> LogController
<<demand>> <<coordinator>> ImageProcessing Controller
<<demand>> <<coordinator>> ImageListController
<<coordinator>> Authentication Controller
<<coordinator>> UserController
<<demand>> <<coordinator>> Process Updating Socket
returnResults
invokeAlgorithms AndServices
<<layer>> ApplicationLayer
<<demand>> <<algorithm>> ImageConverter
<<demand>> <<algorithm>> ImageSplitter
<<demand>> <<algorithm>> ImageMerger
<<service>> Authentication Service
<<service>> EmailService
returnData
invokeCrudOperations
returnData
invokeCrudOperations
<<layer>> DataAcessLayer
<<demand>> <<DAO>> ImageDao
<<demand>> <<DAO>> LogDao
<<DAO>> UserDao
saveAndUpdateData
<<layer>> EntityLayer
<<entity>> Log
<<entity>> Image
<<entity>> User

Figure 6.2.2.2. Dynamic view of the internal layered architecture

This layered architecture diagram illustrates the message flow between components in the Image Processing System. Below is an explanation of how data flows through different layers:

- The WebClient, where the Views operate on, interacts with the service by sending requests (e.g. logging in, processing images, viewing logs, etc.) to the corresponding controllers in the controller layer.
- The controllers act as coordinators, invoking necessary operations and returning the results to the client. They can interact with the classes in both the application layer (for invoking algorithms and services) and the DAO layer (for invoking CRUD operations).
- The classes in the application layer can also interact with the DAO layer to request for data and data-related operations.
- The DAO classes can save and update data to the database using the entity classes.

6.2.3. Architecture in Deployment View

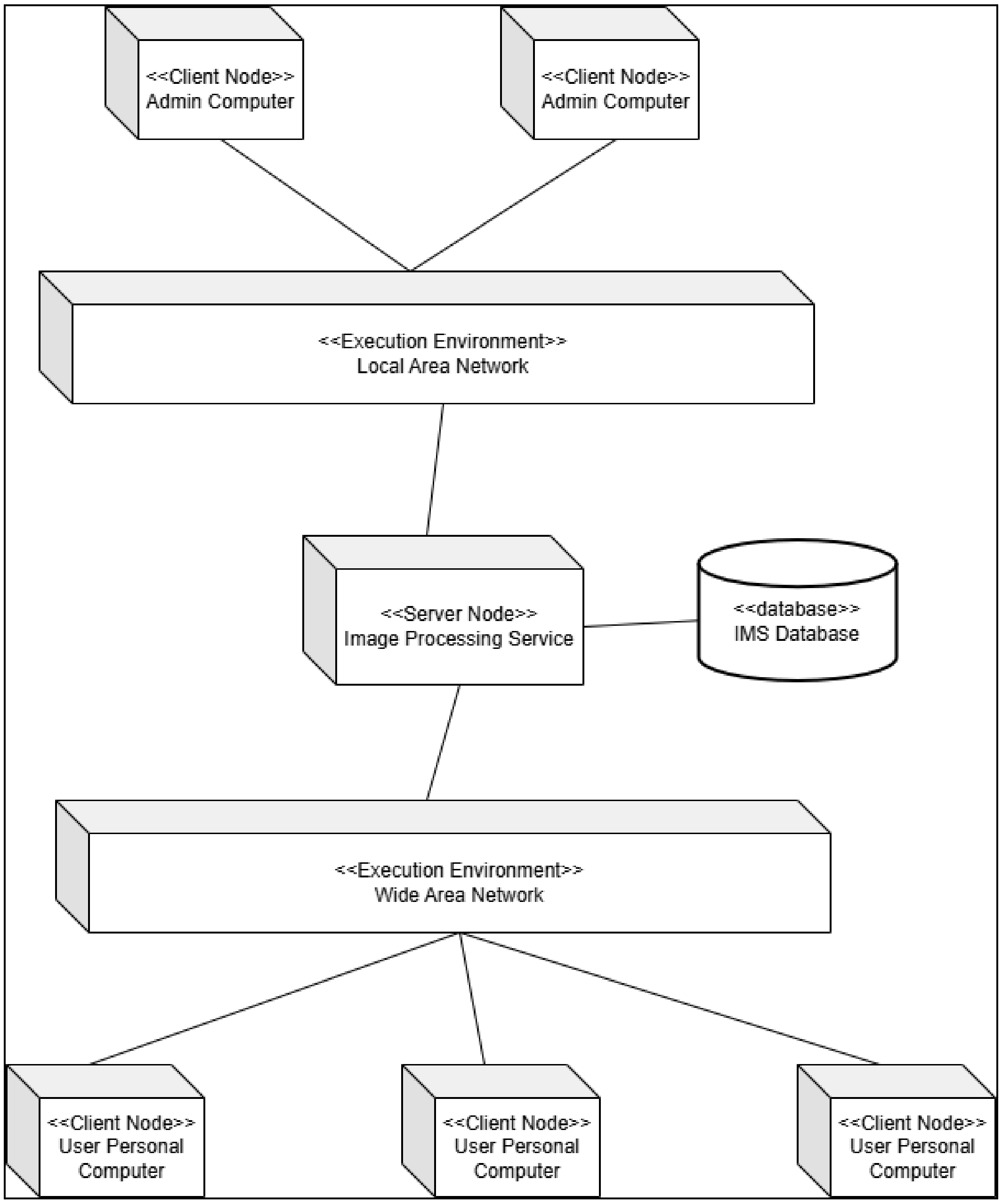


Figure 6.2.3.1. Architecture in Deployment View

The deployment diagram illustrates the deployment view for the architecture of the Image Processing System. The system consists of client nodes, execution environments, a central server, and a database.

Client Nodes: The system supports two types of users:

- **Admin Computers**: Connected through a Local Area Network (LAN), allowing administrators to manage and monitor the system.
- **User Personal Computers**: Connected through a Wide Area Network (WAN), enabling users to access the service remotely.

Execution Environments:

- **Local Area Network (LAN)**: Provides connectivity for admin users to the server.
- **Wide Area Network (WAN)**: Enables external users to access the image processing service over a broader network.

Server Node: The Image Processing Service (IMS) is deployed on a dedicated server node, which is responsible for handling image-related operations such as processing, conversion, and storage. This server is accessible by both admin and user computers through their respective networks.

Database: The IMS Database stores image data, user-related information, and logs of processing activities.

6.3. Database Specifications

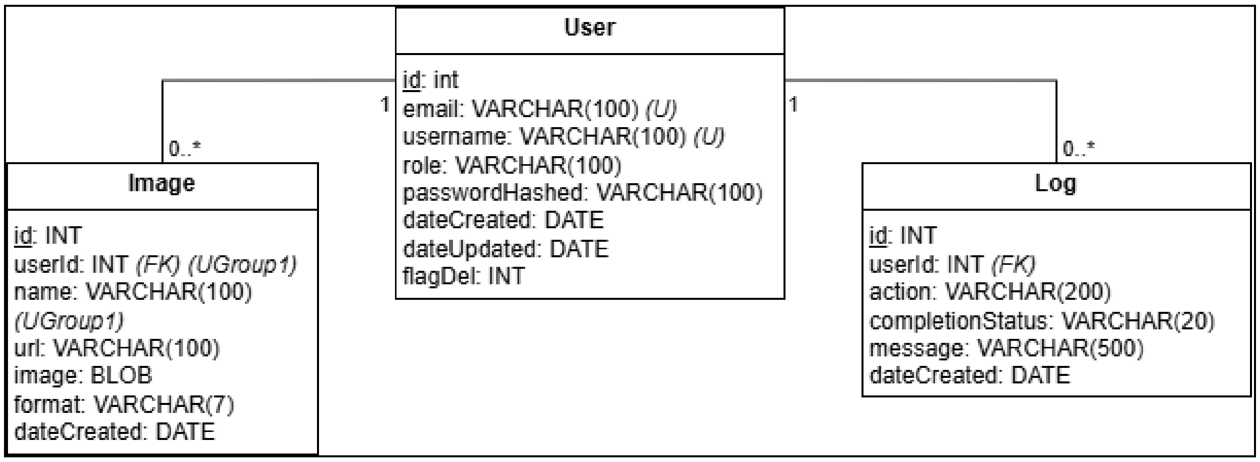


Figure 6.3.1. IMS Database Design

Relation name	**Description**	**Attributes**	**Data Type**	**Constraints**	**Note**
User	Storing user related data	id	INT	Primary key	*N/A*
		email	VARCHAR(100)	Unique, Not Null	*N/A*
		username	VARCHAR(100)	Unique, Not Null	*N/A*
		role	VARCHAR(100)	Not Null	*N/A*
		password Hashed	VARCHAR(100)	Not Null	*N/A*
		dateCreated	DATE	Not Null	*N/A*
		dateUpdated	DATE	*N/A*	*N/A*
		flagDel	INT	Not Null	0: User active 1: User inactive
Image	Storing image data	id	INT	Primary key	*N/A*
		userId	INT	Foreign key, Not Null, Unique Group 1	*N/A*
		name	VARCHAR(100)	Unique Group 1, Not Null	Each user's images must have unique names
		url	VARCHAR(100)	*N/A*	*N/A*
		image	BLOB	*N/A*	BLOB: Binary Large Object

		format	VARCHAR(7)	*N/A*	*N/A*
		dateCreated	DATE	Not Null	*N/A*
Log	Storing data about user activities	id	INT	Primary key	*N/A*
		userId	INT	Foreign key	*N/A*
		action	VARCHAR(200)	Not Null	Specify the action taken by the user
		completion Status	VARCHAR(20)	Not Null	Specify whether the action is completed or not
		message	VARCHAR(500)	*N/A*	Specify completion or error message
		dateCreated	DATE	Not Null	*N/A*

6.4. Detailed Specifications for each component and class

6.4.1. LogView

Stereotype: <<user interaction>>

Purpose: Display the logs of system activities to the user (admin) for monitoring and tracking purposes.

Interactions

Interaction	Parameters	Description
displayLogs	*N/A*	Display all the available logs to the admin.
searchLogs	(searchString: String)	Search logs that match the search string entered by the admin.

refreshLogs	*N/A*	Refresh and reload the log list to show the latest logs.
viewLogDetail	(logId: String)	View detailed information of a specific log entry.

6.4.2. UserController

Stereotype: <<coordinator>>

Purpose: Manage user-related operations in the system, including adding, updating, disabling, and deleting users.

Attributes

Attribute name	Access modifier	Type	Description
userDao	private	IUserDao	Handle database operations for user data
userEmail	private	IEmailService	Handle sending emails to users

Constructor

Parameter	Type	Description
userDao	any implementation of interface **IUserDao**	Inject an implementation of interface **IUserDao** into the class
userEmail	any implementation of interface **IEmailService**	Inject an implementation of interface **IEmailService** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
getAllUsers	public	*N/A*	List<User>	Handle requests to get all users

getUser	public	(userId: int)	User	Handle requests to get a particular user
insertUser	public	(user: User)	void	Handle requests to add a new user to the system
updateUser	public	(user: User)	void	Handle requests to update existing user information
disableUser	public	(userId: int)	void	Handle requests to disable a user based on their ID
deleteUser	public	(userId: int)	boolean	Handle requests to delete a user and returns success status

6.4.3. UserDao

Stereotype: <<DAO>>

Purpose: Handle CRUD operations on users on the database, implements interface **IUserDao**.

Methods

Method name	Access modifier	Parameters	Return type	Description
getAllUsers	public	*N/A*	List<User>	Get data of all users
getUser	public	(userId: int)	User	Get data of a particular user
saveUser	public	(user:User)	void	Save the new user's details to the database
insertUser	public	(user: User)	void	Add a new user to the system
updateUser	public	(user: User)	void	Update existing user information
disableUser	public	(userId: int)	void	Disable a user based on their ID
deleteUser	public	(userId: int)	boolean	Delete a user and returns success status

6.4.4. UserListView

Stereotype: <<user interaction>>

Purpose: Provide a user interface for displaying, managing, and interacting with the list of users.

Interactions

Interaction	Parameters	Description
viewUser	(userId: int)	Send requests to view details of a particular user
viewAllUsers	*N/A*	Send requests to view all users
insertUser	(user: User)	Send requests to insert users
updateUser	(user: User)	Send requests to update users
disableUser	(userId: int)	Send requests to disable users
deleteUser	(userId: int)	Send requests to delete users

6.4.5. EmailService

Stereotype: <<service>>

Purpose: Handle the sending of emails to users for notifications or verification purposes, implements interface **IEmailService**.

Methods

Method name	Access modifier	Parameters	Return type	Description
sendEmail	public	(userEmail: String, subject: String, body: String)	void	Send an email to the specified user email address

6.4.6. AuthenticationView

Stereotype: <<user interaction>>

Purpose: The AuthenticationView class is responsible for providing the graphical user interface (GUI) for user authentication. This includes login screens, registration forms, and options for password recovery. It handles user input, such as entering an email, password, and selecting authentication-related actions.

Interactions

Interaction	Parameters	Description
displayLoginForm()	None	Displays the login screen with fields for email and password.
getLoginInput()	None	Captures the user's email and password input from the login form.
displayLoginError(String errorMsg)	(errorMsg: String)	Displays an error message if the login attempt fails (e.g., "Invalid credentials").
displayRegistrationForm()	None	Displays the registration screen with fields for username, email, and password.
getRegistrationInput()	None	Captures user input (username, email, password) from the registration form.
displayRegistrationError(String errorMsg)	(errorMsg: String)	Displays an error message if registration fails (e.g., "Email already in use").
displayPasswordRecoveryOptions()	None	Displays options for password recovery, such as "Forgot Password".

getPasswordRecoveryInput()	None	Captures user input (email) for password recovery.
displayPasswordResetSuccess()	None	Displays a message confirming that password recovery instructions were sent.
displayAuthenticationSuccess(String username)	(username: String)	Displays a welcome message upon successful login (e.g., "Welcome, [username]!")
redirectToHomePage()	None	Redirects the user to the homepage after successful authentication.
displayLogoutConfirmation()	None	Displays a confirmation message when the user logs out.

6.4.7. AuthenticationController

Stereotype: <<coordinator>>

Purpose: The AuthenticationController acts as an intermediary between the AuthenticationView and AuthenticationService. It coordinates user actions login, register, logout and reset password, retrieves the necessary data from the backend, processes it, and updates the view accordingly. It handles input validation, error handling, and redirection logic.

Attributes

Attribute name	Access modifier	Type	Description
authenticationService	private	IAuthenticationService	Service that handles core authentication logic.

Constructor

Parameter	Type	Description

authenticationService	any implementation of interface **IAuthenticationService**	Inject an implementation of interface **IAuthenticationService** into the class

Method

Method name	Access modifier	Parameters	Return type	Description
handleLogin()	public	None	boolean	Coordinates the login process by getting input, validating, and calling the service.
validateInput()	private	(email: String, password: String)	boolean	Validates email format and password strength.
redirectToHome()	private	None	void	Redirects to the homepage on successful login.
handleError()	private	(errorMessage: String)	void	Updates the view with an error message if login fails.
signUp()	public	(email: String, username: String, password: String)	void	Handles sign-up
signOut()	public	None	void	Handles user logout by clearing the session and redirecting to the login page.
resetPassword()	public	(email: String) (newPassword: String)	void	Handles password reset.
signIn()	public	(email: String) (password: String)	boolean	Handles user login by validating email and password, and checking

				user credentials through the service.
sendOtp()	public	(email: String)	void	Generates and sends an OTP to the user's registered email.
resendOtp()	public	(email: String)	void	Resends a new OTP to the user’s email if the original one has expired or wasn’t received.
verifyOtp()	public	(email: String) (otp: int)	boolean	Verifies the OTP entered by the user against the generated one.

6.4.8. AuthenticationService

Stereotype: <<service>>

Purpose: The AuthenticationService handles core user authentication processes such as signing in, signing out, signing up with OTP verification, and resetting passwords with OTP. It interacts with the UserDAO for database-related operations and manages OTP generation, validation, and account locking for signUp and resetPassword functionalities only. Implements interface **IAuthenticationService**.

Attributes

Attribute name	Access modifier	Type	Description
userDAO	private	IUserDAO	DAO for interacting with the database to perform user-related operations.

Constructor

Parameter	Type	Description
userDAO	any implementation of interface	Inject an implementation of interface

	IUserDAO	**IUserDao** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
signIn()	public	(email: String, password: String)	boolean	Validates user credentials against the database using UserDAO. Returns true if login is successful.
signOut()	public	(sessionId: int)	void	Handles user logout by clearing the user session.
signUp()	public	(email: String, username: String, password: String)	boolean	Creates a new user in the database on successful verification.
resetPassword()	public	(email: String, newPassword: String)	boolean	Updates the password in the database on successful verification.
validateOtp()	public	(email: String, otp: int)	boolean	Verifies the OTP entered by the user. Returns true if the OTP is valid.

6.4.9. User

Stereotype: <<entity>>

Purpose: The User entity represents the user in the system. It stores user-related data, including username, password, email, role, id, photo URL, date created, date updated, flag delete. This class may also include methods for managing user profile details, such as updating passwords or personal information.

Attributes

Attribute name	Access modifier	Type	Description
email	private	String	Stores the user's unique email address for login.
passwordHashed	private	String	Stores the user's encrypted password.
username	private	String	Stores the user's display name.
role	private	String	Stores the user's role (e.g., admin, customer).
id	private	int	Unique identifier for the user.
dateCreated	private	Date	Stores the date when the user was created.
dateUpdated	private	Date	Stores the date when the user profile was last updated.
flagDel	private	int	Flag to indicate if the user is marked as deleted.

Methods

Method name	Access modifier	Parameters	Return type	Description
getEmail()	public	None	String	Returns the user's email.
getPassword()	public	None	String	Returns the user's encrypted password.
getUsername()	public	None	String	Returns the user's display name.
getRole()	public	None	String	Returns the user's role.
getId()	public	None	int	Returns the user's unique identifier.
getDateCreated()	public	None	Date	Returns the date the user was created.
getDateUpdated()	public	None	Date	Returns the date the user profile was last updated.

getFlagDel()	public	None	int	Returns the deletion flag.
setPassword()	public	(password: String)	void	Set user password
setUsername()	public	(username: String)	void	Set user username
setDateUpdated()	public	(dateUpdated: Date)	void	Set date updated
setFlagDel()	public	(flagDel: int)	void	set deletion flag

6.4.10. OTPVerifyView

Stereotype: <<user interaction>>

Purpose: The OTPVerifyView provides the graphical interface for users to enter and verify a one-time password (OTP) sent via email. This view is used during multi-factor authentication (MFA) or password recovery processes. It includes input fields for OTP entry, error messages for invalid OTPs, and a timer if the OTP has an expiration period. If the user enters an incorrect OTP more than 3 times, the system will lock the associated email account for 48 hours, preventing further OTP verification attempts during the lockout period.

Interactions

Interaction	Parameters	Description
displayOtpInput()	None	Displays input fields for the user to enter the OTP received via email.
showErrorMessage()	(errorMessage: String)	Displays an error message (e.g., for incorrect or expired OTP)
startUpTimer()	(durationInSeconds: int)	Starts a countdown timer for OTP expiration and shows the remaining time to the user.

resendOtpButton()	None	Provides an interactive "Resend OTP" button that allows the user to request a new OTP.
submitOtp()	None	Triggers OTP verification logic when the user submits the entered OTP.
resetOtpInput()	None	Clears the OTP input fields, typically used after a failed attempt.
showSuccessMessage()	(message: String)	Displays a success message (e.g., "OTP verified successfully!") after a valid OTP is entered.
disableResendButton()	None	Temporarily disables the "Resend OTP" button to prevent spamming OTP requests.
enableResendButton()	None	Re-enables the "Resend OTP" button after a cooldown period or timer completion.
updateTimerDisplay()	(remainingSeconds: int)	Updates the OTP timer display to show the remaining time before expiration.
redirectAfterSuccess()	(targetUrl: String)	Redirects the user to the next login page or reset password page after OTP verification.
trackOtpAttempts()	(email: String)	Tracks the number of incorrect OTP attempts. After 3 failed attempts, it locks the associated

		email account.
showLockoutMessage()	None	Displays a message indicating that the account is locked due to too many failed OTP attempts.
lockAccountFor48h()	(email: String)	Locks the associated email account for 48 hours after 3 incorrect OTP attempts. Prevents further verification.
checkAccountStatus()	(email: String)	Checks the email account's lockout status and restricts access if within the 48-hour lockout period.

6.4.11. RecoverPasswordView

Stereotype: <<user interaction>>

Purpose: The RecoverPasswordView provides the user interface for completing the password recovery process after the user clicks a reset password link sent to their registered email. Users will set a new password in a single input field. This view includes password validation to ensure it meets security criteria. Upon successfully resetting the password, the user is redirected to the login page.

Interactions

Interaction	Parameters	Description
displayPasswordField()	None	Displays an input field for the user to enter a new password.
validatePassword()	(password: String)	Validates the new password based on predefined rules 8 length, at least 1 special characters, 1 uppercase letters, and 1 number.

showErrorMessage()	(errorMessage: String)	Displays an error message if password validation fails.
submitNewPassword()	(password: String)	Triggers the password reset process and submits the new password to the backend.
redirectToLogin()	None	Redirects the user to the login page after a successful password reset.
resetPasswordField()	None	Clears the password input field after a failed validation.
showSuccessMessage()	(message: String)	Displays a success message after the new password is successfully set "Password reset successfully!".
disableSubmitButton()	None	Temporarily disables the "Submit" button to prevent multiple submissions during processing.
enableSubmitButton()	None	Re-enables the "Submit" button after validation or a backend response.
checkPasswordRequirements()	(password: String)	Checks and displays live feedback on password strength ("Strong," "Medium," "Weak") based on predefined criteria.

6.4.12. Image

Stereotype: <<entity>>

Purpose: Represents image-related data

Attributes

Attribute name	Access modifier	Type	Description
id	private	int	Id of the image on the database
userId	private	int	The Id of the user who owns the image
url	private	String	The URL of the image
image	private	byte[]	The image in binary data, stored in a byte array
format	private	String	The format of the image
isOriginal	private	bool	Determine whether the image is original (uploaded by the user) or processed
dateCreated	private	Date	The date when the image first exists on the system

Methods

Method name	Access modifier	Parameters	Return type	Description
getId	public	*N/A*	int	Get the Id of the image
getUserId	public	*N/A*	int	Get the userId of the image
getUrl	public	*N/A*	String	Get the url of the image
getImage	public	*N/A*	byte[]	Get the image in binary data
getFormat	public	*N/A*	String	Get the format of the image
getIsOriginal	public	*N/A*	bool	Get the original status of the image
getDateCreated	public	*N/A*	Date	Get the date when the image first

				exists on the system

6.4.13. LogController

Stereotype: <<coordinator>>

Purpose: Manage log-related operations in the system, including fetching logs, searching, refreshing log data, and viewing detailed log information.

Attributes

Attribute name	Access modifier	Type	Description
logDao	private	ILogDao	Handles database operations for log data

Constructor

Parameter	Type	Description
logDao	any implementation of interface **ILogDao**	Inject an implementation of interface **ILogDao** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
fetchLogs	public	*N/A*	List<Log>	Handle requests to get all logs
searchLogs	public	(searchString: String)	List<Log>	Handle requests to search logs by search strings
getLogDetail	public	(logId: String)	Log	Handle requests to get detailed information about a specific log entry

6.4.14. Log

Stereotype: <<entity>>

Purpose: Represents log records to track user activities.

Attributes

Attribute name	Access modifier	Type	Description
id	private	int	Id of the log record on the database
userId	private	int	The Id of the user who is associated with the log record
action	private	String	The action taken by the user
completionStatus	private	String	Determine whether the action is completed or not
message	private	String	Success message or error message
dateCreated	private	String	The date when the log record first exists on the system

Methods

Method name	Access modifier	Parameters	Return type	Description
getId	public	*N/A*	int	Get the id of the log record
getUserId	public	*N/A*	int	Get the id of the user who is associated with the record
getAction	public	*N/A*	String	Get the action taken
getCompletionStatus	public	*N/A*	String	Get the completion status of the action
getMessage	public	*N/A*	String	Get the message of the log record

getDateCreated	public	*N/A*	String	Get the date when the log record first exists on the system

6.4.15. ImageListView

Stereotype: <<user interaction>>

Purpose: View the processed images owned by the user.

Interactions

Interaction	Parameters	Description
viewImageList	*N/A*	View list of images
viewImage	(imageId: int)	View a particular image
searchImages	(searchString: String)	Search for particular images by name
deleteImage	(imageId: int)	Delete an image

6.4.16. UserProfileView

Stereotype: <<user interaction>>

Purpose: Provide a user interface where the user can view, manage, and interact with their profile.

Interactions

Interaction	Parameters	Description
viewProfile	(userId: int)	View user profile
editProfile	(newData: User)	Edit user profile

6.5. Concurrent Tasks/Classes Specifications

6.5.1. ImageConverter

Stereotype: <<algorithm>>

Purpose: Handle converting image format. Implements interface **IImageConverter**.

Attributes

Attribute name	Access modifier	Type	Description
logDao	private	ILogDao	Handle writing logs to the database
processUpdatingSocket	private	IProcessUpdatingSocket	Handle updating the progress to the client

Constructor:

Parameter	Type	Description
logDao	any implementation of interface ILogDao	Inject an implementation of interface **ILogDao** into the class
processUpdatingSocket	any implementation of interface IProcessUpdatingSocket	Inject an implementation of interface **IProcessUpdatingSocket** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
convertImage	public	(image: Image, desiredFormat: String)	Image	Convert user-uploaded image to the format desired by the user

6.5.2. ImageSplitter

Stereotype: <<algorithm>>

Purpose: Handle splitting an image into smaller tiles. Implements interface **IImageSplitter**.

Attributes

Attribute name	Access	Type	Description

	modifier		
logDao	private	ILogDao	Handle writing logs to the database
processUpdatingSocket	private	IProcessUpdatingSocket	Handle updating the progress to the client

Constructor

Parameter	**Type**	**Description**
logDao	any implementation of interface **ILogDao**	Inject an implementation of interface **ILogDao** into the class
processUpdatingSockets	any implementation of interface **IProcessUpdatingSocket**	Inject an implementation of interface **IProcessUpdatingSocket** into the class

Methods

Method name	**Access modifier**	**Parameters**	**Return type**	**Description**
splitImage	public	(image: Image, splitSize: int)	Image	Split an user-uploaded image into image tiles with size desired by the user

6.5.3. ImageMerger

Stereotype: <<algorithm>>

Purpose: Handle merging multiple images into a larger image. Implements interface **IImageMerger**.

Attributes

Attribute name	**Access modifier**	**Type**	**Description**
logDao	private	ILogDao	Handle writing logs to the database

imageConverter	private	IImageConverter	Handle converting images to desired output format
processUpdatingSocket	private	IProcessUpdatingSocket	Handle updating the progress to the client

Constructor

Parameter	Type	Description
logDao	any implementation of interface **ILogDao**	Inject an implementation of interface **ILogDao** into the class
imageConverter	any implementation of interface **IImageConverter**	Inject an implementation of interface **IImageConverter** into the class
processUpdatingSocket	any implementation of interface **IProcessUpdatingSocket**	Inject an implementation of interface **IProcessUpdatingSocket** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
mergeImages	public	(images: List<Image>, outputFormat: String mergeSize: double)	Image	Merge user-uploaded images into a larger image with size and format desired by the user.

6.5.4. ProcessUpdatingSocket

Stereotype: <<coordinator>>

Purpose: Handle updating progress of image-processing to user. Implements interface **IProcessUpdatingSocket**.

Attributes

Attribute name	Access modifier	Type	Description
webSocketSession	private	Map<String, Session>	Store web socket sessions

Constructor

None

Methods

Method name	Access modifier	Parameters	Return type	Description
onOpen	public	(session: Session)	void	Perform any action upon opening a web socket connection
onClose	public	(session: Session)	void	Perform any action upon closing a web socket connection
sendProgressUpdate	public	(userId: int, imageName: String, progress: int)	void	Send progress updates from the service to client

6.5.5. ImageProcessingController

Stereotype: <<coordinator>>

Purpose: Handle requests to process images (conversion, splitting, merging) from clients.

Attributes

Attribute name	Access modifier	Type	Description
imageDao	private	IImageDao	Handle saving processed images to database

Constructor

Parameter	Type	Description
imageDao	any implementation of interface **IImageDao**	Inject an implementation of interface **IImageDao** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
convertImage	public	(image: Image, desiredFormat: String)	Image	Handle image-conversion requests from client
splitImage	public	(image: Image, splitSize: int)	Image	Handle image-splitting requests from client
mergeImages	public	(images: List<Image>, outputFormat: String, mergeSize: double)	Image	Handle image-merging requests from client

6.5.6. ImageListController

Stereotype: <<coordinator>>

Purpose: Handle requests involving viewing, downloading and deleting images saved on database.

Attributes

Attribute name	Access modifier	Type	Description
imageDao	private	ImageDao	Handle CRUD operations on images saved on database

Constructor

Parameter	Type	Description

imageDao	any implementation of interface **IImageDao**	Inject an implementation of interface **IImageDao** into the class

Methods

Method name	Access modifier	Parameters	Return type	Description
getImagesByUserId	public	(userId: int)	List<Image>	Handle requests to get all images owned by a particular user
getImage	public	(imageId: int)	Image	Handle requests to get a particular image by its Id
searchImage	public	(searchString: String)	List<Image>	Handle requests to search for some particular images
downloadImage	public	(imageId: int)	Image	Handle requests to get a particular image by its Id to download the on user's computer
deleteImage	public	(imageId: int)	boolean	Handle requests to delete a particular image

6.5.7. LogDao

Stereotype: <<DAO>>

Purpose: Handle fetching logs and creating logs on database. Implements interface **ILogDao**.

Methods

Method name	Access modifier	Parameters	Return type	Description
getLog	public	(logId: int)	Log	Get a particular log record from the database

getAllLogs	public	*N/A*	List<Log>	Get all log records from the database
searchLog	public	(searchString: String)	List<Log>	Search for particular logs using a search string
writeLog	public	(log: Log)	Log	Create a log record on the database

6.5.8. ImageDao

Stereotype: <<DAO>>

Purpose: Handle CRUD operations on images on database. Implements interface **IImageDao**

Methods

Method name	Access modifier	Parameters	Return type	Description
getImage	public	(imageId: int)	Image	Get a particular image from the database
getImagesByUserId	public	(userId: int)	List<Image>	Get all images owned by a user
searchImage	public	(searchString: String)	List<Image>	Get all images whose names contain the search string
createImage	public	(image: Image)	Image	Create an image record on the database
deleteImage	public	(imageId: int)	boolean	Delete an image record on the database

6.5.9. ImageProcessingView

Stereotype: <<user interaction>>

Purpose: The view where users can upload images for processing (converting, splitting, merging) and get updates about the progress.

Interactions

Interaction	Parameters	Description
convertImage	(image: Image, desiredFormat: String)	Send user requests to convert image format
splitImage	(image: Image, splitSize: int)	Send user requests to split an image
mergeImages	(images: List<Image>, outputFormat: String, mergeSize: double)	Send user requests to merge images
viewPercentage OfProgress	*N/A*	View the progress made to process an image

6.5.10. ImageDownloadingView

Stereotype: <<user interaction>>

Purpose: The view where user can preview and download processed images.

Interactions

Interaction	Parameters	Description
previewImage	(imageId: int)	Preview an image before downloading
downloadImage	(imageId: int)	Save an image to the user's local device

6.6. List of Quality Attributes and Evaluation Results

This section evaluates how the system achieves the specified non-functional requirements.

6.6.1. Usability

The image processing system web application achieves usability goals through the following means:

- **User-Friendly Interface**:
  - The interface has been designed with simplicity and intuitiveness in mind. Important functions like image uploads, processing options, and download buttons are clearly labeled and easily accessible.
  - Dropdown menus, tooltips, and error feedback enhance the user experience by making navigation intuitive and minimizing user mistakes.
- **Cross-Browser Support**:
  - The system's front-end is developed using **HTML5**, **CSS3**, and modern JavaScript frameworks, ensuring that it functions correctly on popular browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge.
- **Responsive Design**:
  - The system uses **responsive web design principles** such as flexible grids, media queries, and scalable images to ensure a seamless experience on mobile phones, tablets, and desktops. UI components automatically adjust based on screen size, providing an optimized layout for different devices.

6.6.2. Security

Security is a critical component, and the system incorporates the following measures to protect users' data and prevent common web vulnerabilities:

- **Personal Data Protection**:
  - All user data transmissions are encrypted using **HTTPS**. This prevents data from being intercepted by third parties during transmission.
  - User passwords are hashed before being stored in the database using strong cryptographic hashing algorithms, protecting sensitive information even in the event of a data breach.
- **Data Ownership:**
  - Each image is marked with their owner id. And before returning the requested image(s), the user session will be checked to determine whether the user is the valid owner. This ensures only the user who owns the image(s) can have access.
- **Access Control**:
  - The system implements **role-based access control (RBAC)** to ensure that different user roles (e.g., Admin and regular User) have appropriate access to resources and functions. Admins can manage user accounts, view logs, and update settings, while regular users can only perform basic tasks like image uploads and processing.

- **Security Measures Against Common Attacks**:
    - **SQL Injection Mitigation**: The system uses **prepared statements** with parameterized queries, along with wrapper classes to prevent SQL injection attacks.
    - **Cross-Site Scripting (XSS)** Mitigation: User inputs are sanitized and validated to prevent malicious scripts from being executed on the client side.
    - **Cross-Site Request Forgery (CSRF)** Mitigation: **CSRF tokens** are implemented for sensitive operations to verify that requests originate from legitimate users.

6.6.3. Maintainability

The system is designed to be maintainable and scalable through the following practices:

- **Maintainable Code**:
    - The system follows a **modular architecture** with well-defined layers, each handle a specific task (see *Internal Layered Architecture of the Software System*)
    - The system is designed so that components are loosely coupled, allowing swapping different implementations of features without significant modifications of the existing codebase.

6.6.4. Reliability

The system ensures high availability and reliability as follows:

- **Uptime Guarantee**:
    - The deployment environment is configured to maintain 96% uptime with redundancy and failover mechanisms. This ensures users can access the application at any time with minimal downtime.

6.6.5. Availability

- The system maintains functional 24/7, except on scheduled maintenance
- The maintenance schedule will be notified to users via email