Gandhi Engineering College

Affiliated to BPUT, Approved BY AICTE and Govt. of Odisha

AI IMAGE EDITOR

A major project report submitted for the award of the Degree of Bachelor of Technology in Computer Science and Engineering (AIML).

Submitted by

Chandra Sekhar Swamy (2101292240)

Bibhu Prasad Dash (2101292238)

Amit Kumar Praharaj (2101292222)

Ranjit Kumar Tarai (2101292272)

Under the supervision of

Prof. Nameet Kumar Sethy &
Prof. Debasmita Rout

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (AIML)
GANDHI ENGINEERING COLLEGE, BHUBANESWAR
Gandhi Vihar, Madanpur, Bhubaneswar-752054
2021-2025

ABSTRACT

The AI Image Editor is a web-based application designed to simplify and enhance the process of image editing through artificial intelligence and intuitive design. The application allows users to upload images in various formats, perform essential processing operations—such as grayscale conversion, format transformation, and other manipulations—and download the results effortlessly. By leveraging technologies like Flask, OpenCV, and SQLite, the system ensures high performance, scalability, and secure data handling.

This project caters to individuals and professionals who require accessible, efficient, and cost-effective image editing tools without the need for advanced technical knowledge or expensive software. Key features include user authentication, secure file uploads, a clean user interface, and real-time image processing. Developed using the Waterfall Model, the system underwent rigorous testing—including unit, integration, and acceptance testing—to ensure functionality, reliability, and usability.

The application addresses modern challenges in image editing by providing cross-platform compatibility, data security, and ease of use. It lays a solid foundation for further enhancements, such as integrating machine learning for advanced editing. This project demonstrates a practical and scalable solution for online image editing, making it a valuable tool for users across different domains.

ACKNOWLEDGMENT

Our sincere thanks to **Dr. Irani Majumdar**, Professor and Head of the Department of Computer Science & Engineering, Gandhi Engineering College (GEC), Bhubaneswar, for his encouragement and valuable suggestions during the period of our Project work. No words would suffice to express my regards and gratitude to **Prof. Nameet Sethy & Prof. Debasmita Rout,** Department of Computer Science & Engineering, for his inspiring guidance and constant encouragement, immense support and help during the project work.

Place : Gandhi Engineering College. Chandra Sekhar Swamy

Date - Reg. No - 2101292240

A Software requirements specification (SRS) document describes the intended purpose, requirements, and nature of software/application/project to be developed.

To prepare an SRS document, you would need to have a functional knowledge of your project or application, knowledge of software/hardware/technology to be used.

Generally, the report is prepared with the following format.

INDEX

The following software requirements specification report has been prepared for a project named: **AI Image Editor**

Chapter 1: INTRODUCTION

The **AI Image Editor** is an intelligent web-based application designed to simplify and enhance image editing tasks for users. It allows users to upload images in various formats and apply a range of automated editing operations, such as grayscale conversion and format transformations, with just a few clicks. The application caters to individuals and professionals who seek an easy-to-use platform for quick image processing without the need for advanced tools or software.

The primary objective of this project is to provide a robust solution for handling image processing tasks efficiently and securely. It leverages Artificial Intelligence (AI) and image processing techniques to deliver high-quality results while ensuring user-friendly interaction. The platform supports multiple image formats, ensuring versatility and accessibility for a wide audience.

This software is a web-based application accessible via a browser, ensuring cross-platform compatibility. It provides a clean, intuitive interface, allowing users to upload images, perform edits, and download the results effortlessly. With secure user authentication and session handling, the AI Image Editor also maintains data privacy and protection, making it an ideal choice for anyone looking for a reliable image editing tool.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the functional and non-functional requirements of an **Image Processing Web Application**. This application provides users with the ability to upload images, apply various processing operations (e.g., grayscale conversion, format conversion), and download the processed results. Additionally, the system includes secure user authentication and role management to ensure data security and personalized user experiences.

1.2 Need/Motivation

In the digital era, image editing has become an integral part of personal and professional activities. However, many users face challenges such as the need for expensive software, complex tools, or high-performance devices for basic image processing tasks. The **AI Image Editor** addresses these gaps by offering a simple, accessible, and efficient solution for image editing.

The motivation for this project stems from the following needs:

1. **Ease of Access**: Many users lack the technical expertise or resources to operate advanced image editing software. A web-based tool allows them to perform essential editing tasks without installation or training.
2. **Cost-Effective Solution**: Commercial software often comes with high costs, making it inaccessible for students, freelancers, and small businesses. This application provides an affordable and efficient alternative.
3. **Automation and Efficiency**: Manual editing can be time-consuming. The AI Image Editor streamlines operations like grayscale conversion and format transformation, allowing users to achieve results quickly and effortlessly.
4. **Cross-Platform Usability**: As a web-based platform, it eliminates compatibility issues, enabling users to access the tool from any device with a browser.
5. **User Security and Privacy**: With robust authentication mechanisms, it ensures that user data and uploaded images remain secure.

LITERATURE SURVEY

The **AI Image Editor** project draws upon existing advancements in image processing, webbased applications, and user-centric design principles. Below is a survey of relevant literature and technologies that inspired and informed the development of this project.

1. Image Processing Technologies

- **OpenCV (Open Source Computer Vision Library)**:
  OpenCV is a popular open-source library widely used for image and video processing. Its capabilities include image transformations, color space conversions, and file format changes. Research papers and documentation highlight its efficiency and scalability for real-time image processing applications.
- **Artificial Intelligence in Image Editing**:
  AI-powered tools like Adobe Photoshop's AI-based features and tools like Canva demonstrate the growing need for automated and intelligent image editing capabilities. Research indicates that automation improves usability and reduces the learning curve for users unfamiliar with complex tools.

2. Web-Based Applications • **Web-Based Image Editors**:
  Existing platforms such as Pixlr and Fotor show the potential of browser-based image editing. Studies highlight their accessibility and convenience, especially for nonprofessional users. However, many of these tools are subscription-based or have limited free functionalities.
- **Secure File Upload Mechanisms**:
  Best practices in web applications emphasize secure file uploads to prevent vulnerabilities such as unauthorized access or malicious file execution. Flask's integration with secure_filename and other libraries ensures compliance with these practices.

3. User Interface Design

- **Simplified Interfaces for Non-Technical Users**:
  Research demonstrates that clean and intuitive interfaces increase user engagement and adoption of applications. Tools that focus on minimalistic designs with clear instructions tend to perform better with a diverse user base.

Chapter 3:
REQUIREMENTS

3.1 Functional Requirements

The functional requirements define the specific functionalities the system must perform:

- **User Authentication**:
- Users must be able to sign up by providing a unique username and password.
- Users must log in using valid credentials.
- Sessions should persist until the user logs out.
- **Image Upload**:
- Users can upload images in supported formats (PNG, JPG, JPEG, WEBP, GIF).
- The system should validate the file type before uploading.
- **Image Processing**:
- Users can apply specific operations such as:
- Grayscale conversion.
- Format conversion to PNG, JPG, or WEBP.
- Processed images should be saved and accessible to the user.

3.2 Non- Functional Requirements

3.2.1 Safety Requirements

- The system should ensure proper error handling to avoid crashes during image processing or file uploads.
- Files exceeding the permissible size limit or unsupported formats should be safely rejected with appropriate feedback to the user.

3.2.2 Security Requirements

• All passwords must be securely hashed before storage using a robust algorithm like SHA-256.

- User sessions must be encrypted and expire after a period of inactivity to prevent unauthorized access.
- Secure file upload mechanisms should ensure that only valid image files are processed, protecting the system from malicious files.
- Prevent unauthorized access to processed images by restricting access to logged-in users only.

3.2.3 Software Quality Attributes

□**Performance**:

- Image processing tasks should complete within 2 seconds for typical file sizes (under 5MB).
- The system should handle multiple simultaneous users without significant performance degradation. **Usability**:
- Provide a simple and intuitive interface for users with minimal training.
- Clear navigation and instructions should be available for all actions.

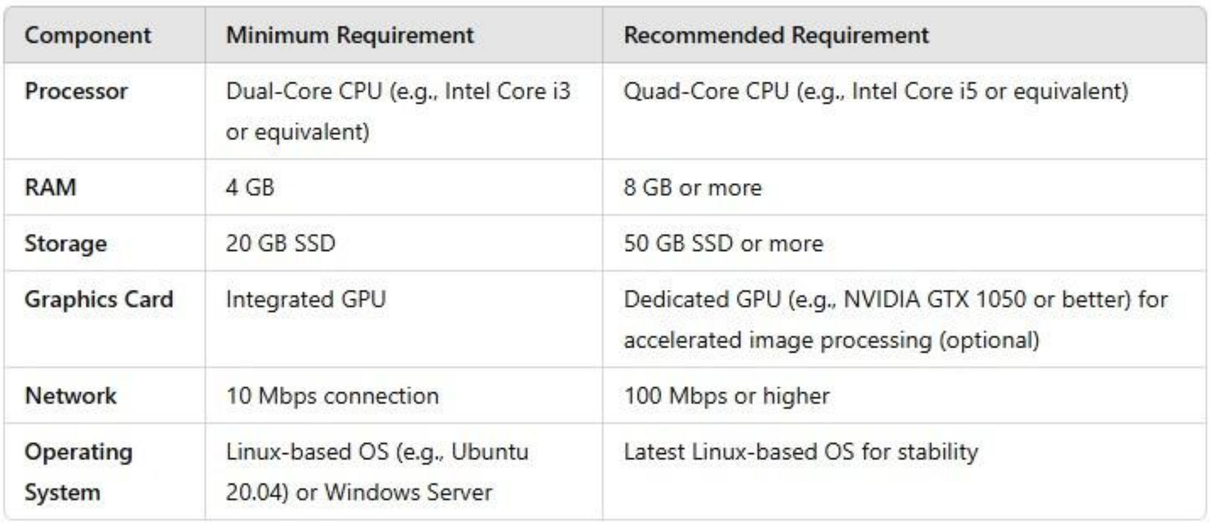
□ **Reliability**:

- The system should maintain at least 99.9% uptime.
- Images and user data should be safely stored with no data loss in the event of a system failure. **Scalability**:
- The system should handle an increasing number of users and file uploads as demand grows.

3.3 Hardware Requirements

1. Server-Side Requirements

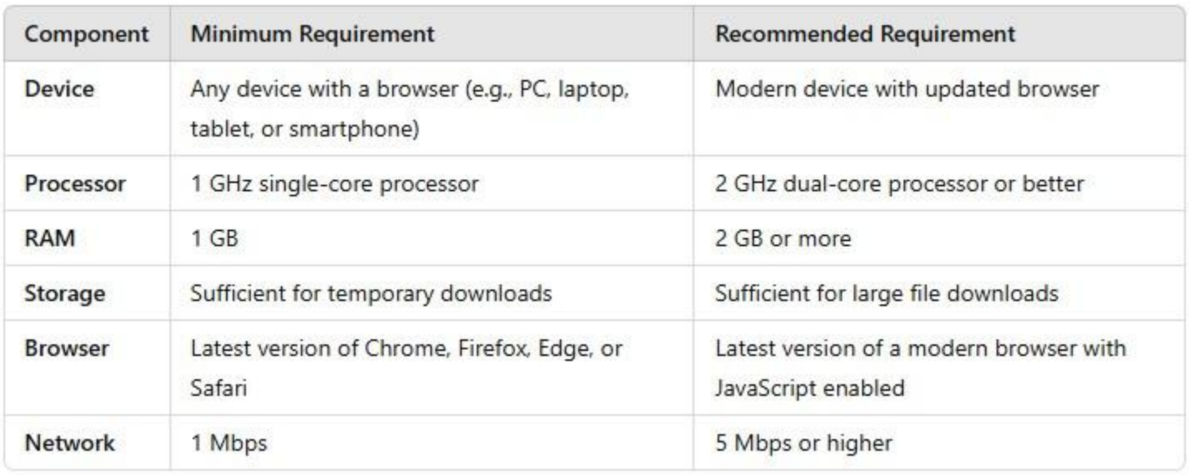
The server-side hardware is essential for hosting the application, processing images, and managing user requests.



Component	Minimum Requirement	Recommended Requirement
Processor	Dual-Core CPU (e.g., Intel Core i3 or equivalent)	Quad-Core CPU (e.g., Intel Core i5 or equivalent)
RAM	4 GB	8 GB or more
Storage	20 GB SSD	50 GB SSD or more
Graphics Card	Integrated GPU	Dedicated GPU (e.g., NVIDIA GTX 1050 or better) for accelerated image processing (optional)
Network	10 Mbps connection	100 Mbps or higher
Operating System	Linux-based OS (e.g., Ubuntu 20.04) or Windows Server	Latest Linux-based OS for stability

2. Client-Side Requirements

The client-side hardware is for users accessing the web application.



Component	Minimum Requirement	Recommended Requirement
Device	Any device with a browser (e.g., PC, laptop, tablet, or smartphone)	Modern device with updated browser
Processor	1 GHz single-core processor	2 GHz dual-core processor or better
RAM	1 GB	2 GB or more
Storage	Sufficient for temporary downloads	Sufficient for large file downloads
Browser	Latest version of Chrome, Firefox, Edge, or Safari	Latest version of a modern browser with JavaScript enabled
Network	1 Mbps	5 Mbps or higher

3.4 Software Requirements

Frontend

- **HTML**: For structuring the web pages.
- **CSS**: For styling and layout of the web pages.
- **JavaScript**: For adding interactivity to the user interface.
- **Web Browser**: Compatible with Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, or similar modern browsers.

Backend

- **Python**: The primary programming language used for backend logic and application development.
- **Flask Framework**: A Python-based micro web framework for handling server-side operations.
- **SQL (SQLite)**: For database operations and user data storage.

Development and Deployment Tools

- **Git**: For version control and collaboration.
- **Operating System**: Compatible with Windows, Linux (Ubuntu), or macOS for development and deployment.
- **OpenCV**: For image processing and transformation functionalities.

3.5 Waterfall Model

To create a **Waterfall Model** for your project, we will break down the development process into sequential and well-defined phases. Each phase's output will serve as input for the next. Here's how the model applies to your image-processing web application:

1. Requirements Analysis

- **Objective**: Understand the functionality required for the project. • **Tasks**:
  - Gather requirements: User authentication, image upload, processing, and conversion.
  - Identify supported image formats and transformations (e.g., grayscale, WebP, PNG, JPG).
  - Database setup for user management (SQLite).
  - Define user flows: signup, login, image upload, and processing.

2. System Design

- **Objective**: Architect the application structure and functionality.
- **Tasks**: o Define the app's routes: /, /about, /edit, /signup, /login, /logout.

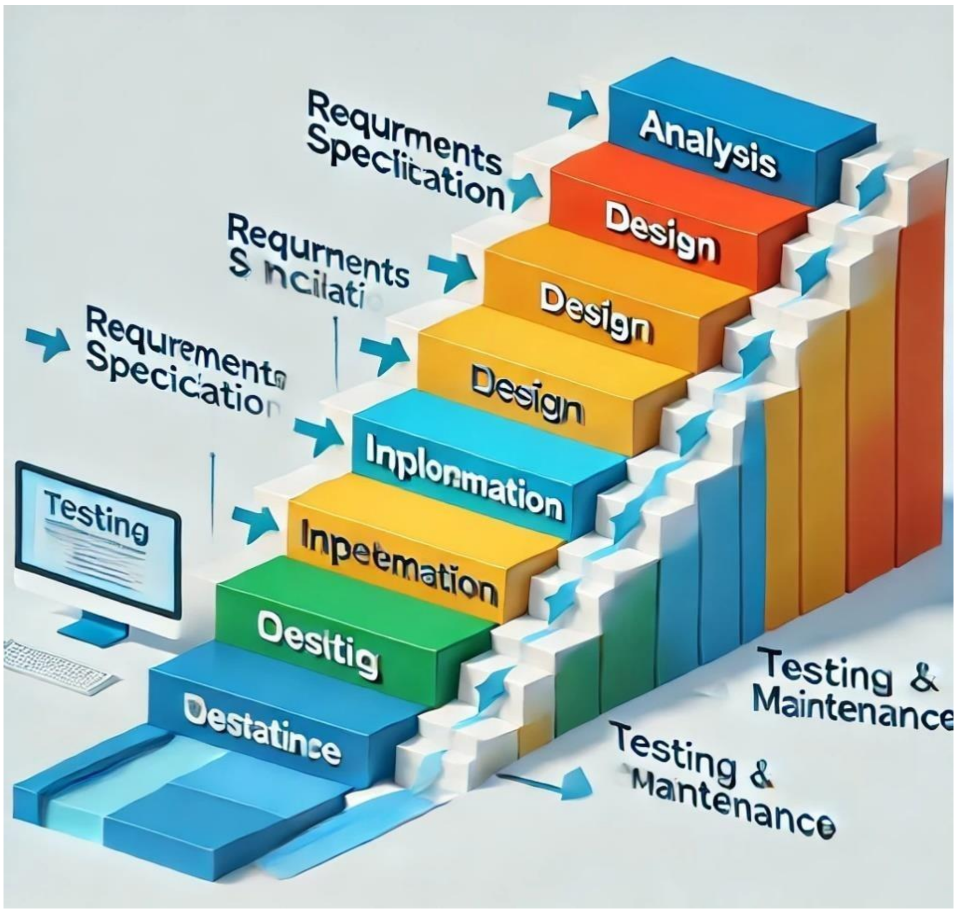
- Database schema design for the User table. ○ File handling configuration (e.g., UPLOAD_FOLDER, ALLOWED_EXTENSIONS).
- Frontend templates for index.html, about.html, signup.html, and login.html.

3. Implementation

- **Objective**: Develop the application according to the design specifications.
- **Tasks**: ○ Create a Flask application structure:
    - Implement user authentication using Flask.
    - Develop routes and integrate SQLAlchemy for database interactions.
    - Use OpenCV for image processing operations.
  - Implement file upload and validation using secure_filename. ○ Write frontend templates and integrate with backend logic.
  - Set up session management for user authentication.

4. Integration and Testing

- **Objective**: Ensure all components work together and validate functionality.
- **Tasks**:
  - Unit testing:
    - Test individual routes for expected behaviour.
    - Validate database operations for user signup/login.
    - Test image upload and processing functions with allowed/disallowed file types.
  - Integration testing:
    - Test the flow from signup/login to image processing.
    - Verify processed images are stored and accessible.
  - User interface testing for usability and consistency



3.6 Feasibility Study

The feasibility study analyses the practicality and viability of the project to ensure its successful implementation. It evaluates three main aspects: economic, technical, and operational feasibility.

3.6.1 Economic Feasibility

□ **Objective:** To determine if the project is cost-effective.

□ **Analysis:**

- **Initial Costs:** The application relies on open-source technologies such as Flask, SQLite, and OpenCV, which reduce development costs.
- **Operational Costs:** The deployment on a local or cloud server (e.g., Heroku or AWS) requires minimal hosting fees.
- **ROI:** The project adds value by providing image processing and user management solutions at a low cost.

□ **Conclusion:** The project is economically feasible due to low development and operational expenses.

3.6.2 Technical feasibility

□ **Objective:** To assess whether the technical requirements of the project can be met with the available tools and expertise.

□ **Analysis:**

- **Technologies Used:**
    - Flask framework for the web application backend. o SQLite database for lightweight and reliable data storage.
    - OpenCV for image processing tasks.
- **Resources:** Availability of necessary development libraries and expertise in Python and Flask.
- **Challenges:** Managing large image files might impact performance, but optimization techniques can address this.

□ **Conclusion:** The project is technically feasible given the use of proven technologies and resources.

3.6.3 Operational Feasibility

□ **Objective:** To evaluate if the application can function effectively in the intended environment.

□ **Analysis:**

- **Ease of Use:** The web interface is designed to be intuitive for users with options for image uploads and account management.
- **Scalability:** The system can be scaled for larger user bases or additional features if required.
- **Maintenance:** Regular updates to dependencies and security patches can ensure smooth operations.

□ **Conclusion:** The project is operationally feasible as it aligns with the user's needs and is easy to maintain.

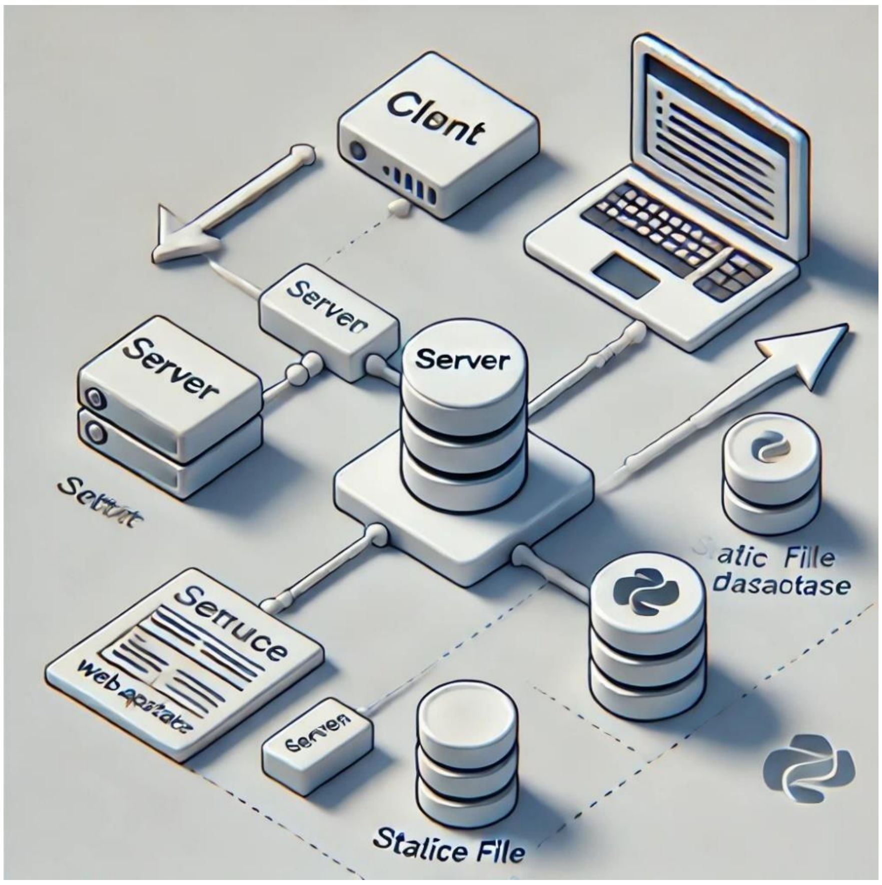
Chapter 4:
System Architecture

4.1 Clint-Server Architecture

The **Client-Server Architecture** is a distributed system framework where the server provides resources or services, and the client consumes them. This model is commonly used in web applications to ensure efficient communication between the client (browser/user interface) and server (backend).
For the provided project, here's how the client-server architecture is structured:

Components in the Architecture:

1. **Client:**
   - Represents the user interface. o Typically a browser or application that sends HTTP requests to the server (e.g., submitting login data or uploading files). o Displays the server's responses (e.g., processed images or confirmation messages).
2. **Server:**
   - Hosts the web application backend, built using Flask. o Handles client requests, processes data (e.g., image processing using OpenCV), and sends responses.
   - Manages user authentication and file storage.
3. **Database:**
   - SQLite is used to store user data such as credentials. o The server interacts with the database to authenticate users or register new accounts.
4. **Static File Server:**
   - Serves static resources (e.g., processed images stored in the static/ folder).
   - Ensures efficient delivery of processed files back to the client.



Chapter 5:
Design and Implementation

5.1 Product Features

The **Design and Implementation** section outlines the architecture, components, and features of the project. This phase transforms the analysed requirements into a functional system. Below are the key details:

5.1 Product Features

The project includes the following features:

1. User Authentication System

- **Signup:**
  - Users can register with a unique username and a password.
  - Passwords are securely hashed using generate_password_hash for protection.
- **Login:** ○ Registered users can log in with their credentials.
  - Secure validation using check_password_hash.
- **Logout:** ○ Users can log out of their session to protect their accounts.

2. Image Upload and Processing • File Upload:

○ Users can upload images in formats such as PNG, JPG, JPEG, GIF, and WEBP. ○ Validation ensures only allowed file types are processed.

- **Image Processing:**
  - Users can select the following operations:
    - **Convert to Grayscale:** Converts the uploaded image into grayscale using OpenCV.
    - **Convert to Other Formats:** Converts the uploaded image to WEBP, JPG, or PNG.
- **Processed Image Delivery:**
  - Processed images are stored in the static/ folder and made available via a link.

3. Secure File Handling

- Uploaded files are sanitized using secure_filename to prevent security issues.
- Files are stored in a designated uploads/ directory to separate them from application code.

4. Navigation and User Interface • Home Page: ○ Describes the application and provides options to navigate.

- **About Page:** ○ Offers details about the application and its functionality.
- **Edit Page:**
  - Allows users to upload and process images via a form.
- **Login and Signup Pages:**
  - Forms for user authentication, styled for ease of use.

5. Database Integration

- **SQLite Database:**
  - Stores user credentials securely.
  - Efficient for lightweight applications and small-scale data.

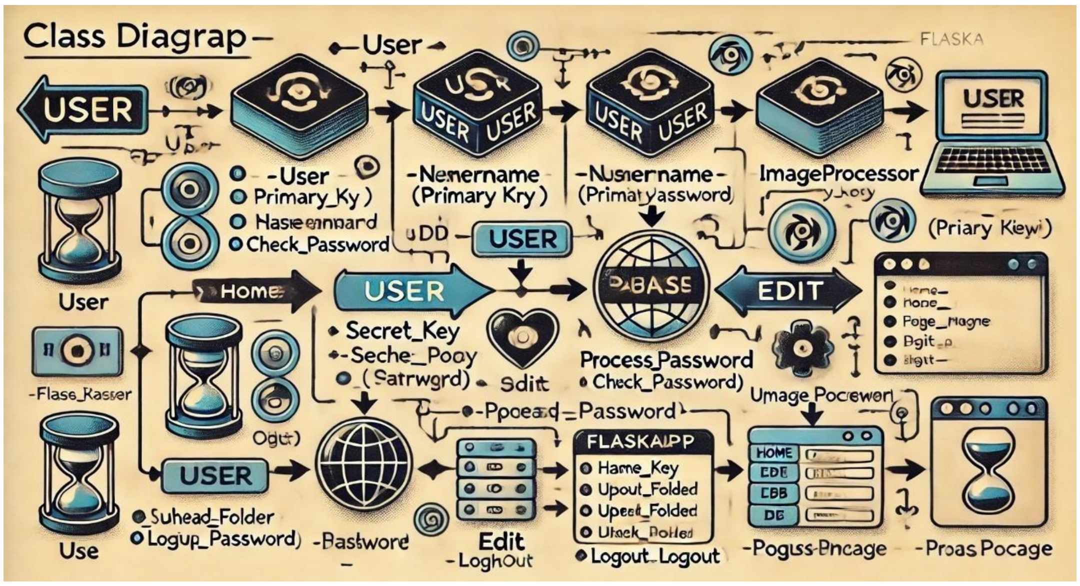
6. Flash Messaging and Notifications

- Provides feedback to users (e.g., "Signup successful," "No file selected").
- Helps enhance user experience by providing real-time updates.

5.2 Class Diagram

The **class diagram** for this system models the primary entities: User, ImageProcessor, and FlaskApp.

1. **User**:
    - o Represents users signing up and logging in. o Attributes: id, username, and password.
    - o Methods: hash_password(), check_password().
2. **ImageProcessor**:
    - o Represents the logic for processing uploaded images. o Attributes: filename and operation.
    - o Methods: processImage().
3. **FlaskApp**:
    - o Represents the main Flask application handling routes.
    - o Attributes: secret_key, upload_folder, db.
    - o Methods: home(), edit(), signup(), login(), logout().

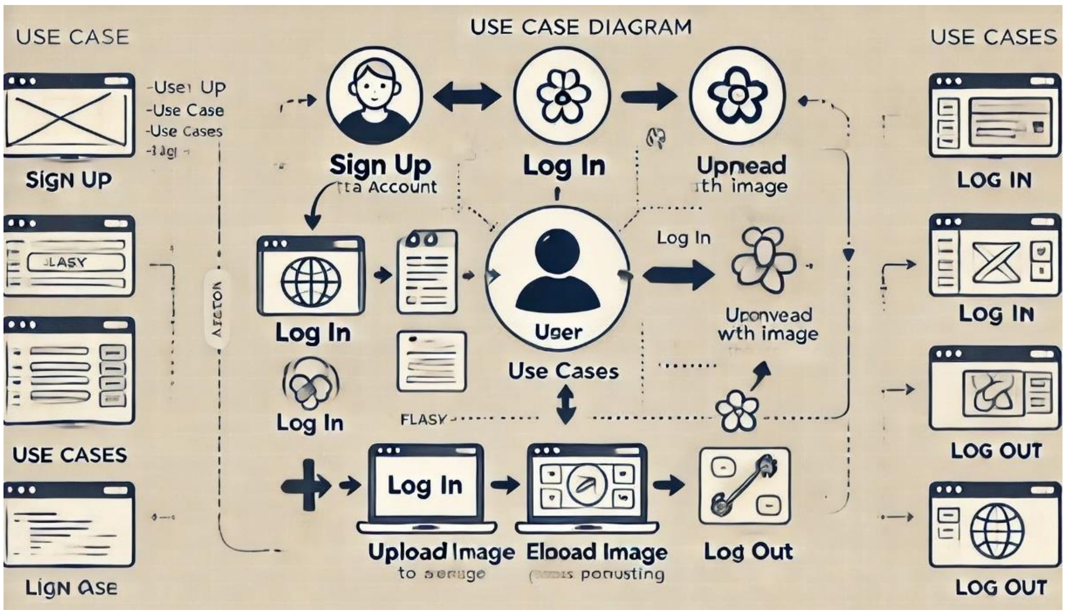


5.2 Use Case Diagram

The **use case diagram** models the interactions between users and the system:

- **Actors**: o **User**: Interacts with the application (uploads images, logs in, signs up).
    - o **System**: Handles user authentication, image uploads, and processing.
- **Use Cases**:

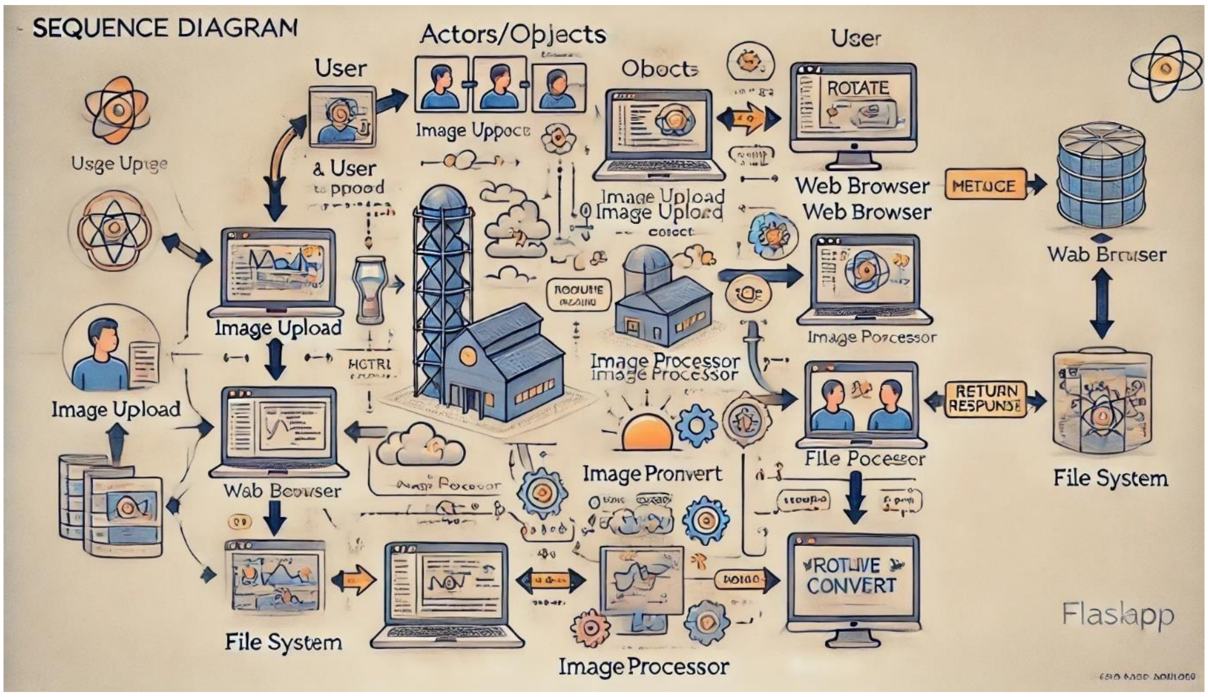
1. **Sign Up**: Users create an account.
2. **Log In**: Users authenticate to access features.
3. **Upload Image**: Users upload an image for processing.
4. **Edit Image**: Users select operations for image processing (convert, rotate, etc.).
5. **Log Out**: Ends the user session.



5.4 Sequence diagram

The **sequence diagram** models the sequence of interactions when a user processes an image:

1. **User**: Uploads an image and selects an operation.
2. **System**:
   - Validates file upload and operation. ○ Saves the image to uploads folder. ○ Calls processImage() to perform the selected operation. ○ Saves processed image to static folder.
   - Returns a link to the processed image.



5.5 E-R Diagram and Database Normalization

5.5.1 E-R DIAGRAM

For your system, the primary entities would be User and Image (for image uploads), and their relationships can be defined as follows: Entities:

1. User:
   - o Attributes:
     - ▯ User ID (Primary Key)
     - ▯ Username (Unique)
     - ▯ Password (Hashed) o Relationships:
     - ▯ A User can upload multiple Images.
2. Image:
   - o Attributes:
     - ▯ Image ID (Primary Key)
     - ▯ Filename
     - ▯ Path
     - ▯ Processed Filename
     - ▯ Operation (e.g., "rotate", "cgray", etc.)
     - ▯ User ID (Foreign Key to User) Relationships:

- A User can upload many Images (One-to-Many relationship between User and Image).

```
+----------------+              +----------------+
|    User       |              |    Image      |
+----------------+              +----------------+
| User_ID (PK)   | <-------->  | Image_ID (PK)  |
| Username       |              | Filename       |
| Password       |              | Path           |
+----------------+              | Processed File |
                                | Operation      |
                                |       User_ID        (FK)        |
+----------------+
```

Explanation of the Diagram:

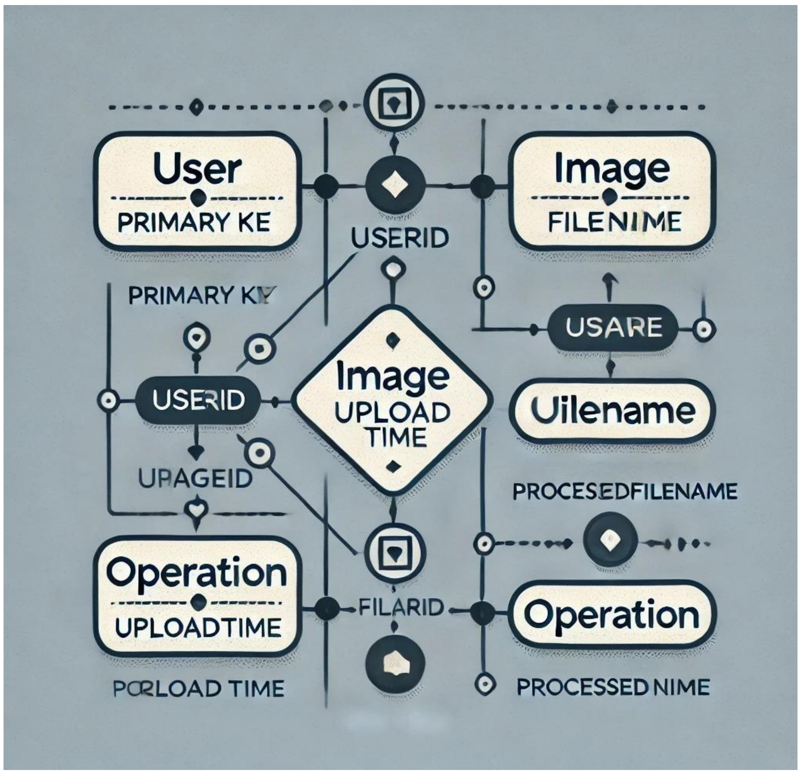
- **User**: o **User_ID (PK)** is the unique identifier for each user.
  - o **Username** and **Password** are the unique credentials for the user.
- **Image**: o **Image_ID (PK)** is the unique identifier for each image. o **Filename** represents the name of the image file uploaded. o **Path** is the location where the image is stored on the server.
  - o **Processed Filename** represents the processed image's name after it undergoes some transformation.
  - o **Operation** stores the type of operation performed (like "rotate" or "cgray").
  - o **User_ID (FK)** links the image to the user who uploaded it.

Relationship:

- A **User** can upload many **Images**, which is a one-to-many relationship, represented by the arrow from **User** to **Image**.

5.5.2 DATABASE NORMALIZATION

- Normalization is the process of organizing data in a database to avoid redundancy and dependency. The goal is to ensure that the data is stored in a way that minimizes data duplication and ensures data integrity.
- **First Normal Form (1NF):**
- **Each table has a primary key**.
- **All attributes contain atomic values** (i.e., no multiple values in a single attribute).
- Example: In the **User** table, **Username** and **Password** must contain single values for each record, not a list of usernames or passwords.
- **Second Normal Form (2NF):**
- **Table is in 1NF**.
- **No partial dependency**. This means that all non-key attributes must be fully dependent on the primary key.
- Example: The **Image** table contains only attributes that depend on the **Image_ID**. **User_ID** is the foreign key that links it to the **User** table.
- **Third Normal Form (3NF):**
- **Table is in 2NF**.
- **No transitive dependency**. Non-key attributes should not depend on other non-key attributes.
- Example: The **Image** table has no attributes that depend on anything other than the primary key (Image_ID) or the foreign key (User_ID).
- In your system, the **User** and **Image** tables already meet the requirements for 3NF:
- **User Table**: The only non-key attribute is Password, which depends solely on the primary key User_ID.
- **Image Table**: Attributes like Filename, Path, and Processed Filename are all fully dependent on the primary key Image_ID and not on each other.
- Thus, this design is already normalized to 3NF.



Chapter 6:

TESTING AND RESULT

Testing is a crucial part of software development that ensures the functionality and quality of the application. The different types of testing mentioned are essential to ensure the application works as expected under various conditions.

6.1 Unit Testing

Unit Testing involves testing individual components or units of a program in isolation to ensure they work correctly. The focus is on small units like functions, methods, or classes.

- **Goal**: To verify that each function or method in isolation works as intended.
- **Tools**: Common tools for unit testing in Python include unittest, pytest, and nose2.
- **Example**: For your Flask application, you could write unit tests for the processImage() function to check that the image processing operations (like rotation or conversion) are working as expected.

```
import unittest
from app import process `Image

class TestImageProcessing(unittest.TestCase):
def test_grayscale_conversion(self):
result = processImage("test.jpg", "cgray")
     self.assertTrue(result.endswith(".jpg"))  # Check if the processed file is saved with the
correct extension

  def  test_rotation(self):                    result  =
processImage("test.jpg", "rotate")
     self.assertTrue(result.startswith("static/rotated_"))  # Ensure the result is a rotated image
file

if       __name__       ==       "__main__":
unittest.main()
```

6.2 Black Box Testing

Black Box Testing focuses on testing the software’s functionality without knowledge of its internal workings. The tester only knows the input and expected output but not how the system processes them.

- **Goal**: To verify that the software performs its intended functions correctly from the user's perspective.
- **Tests**: Black box testing will test features like login functionality, file upload, and image processing by providing inputs and comparing the outputs.
- **Example**: ○ Test if a user can successfully sign up with valid credentials. ○ Test whether the image upload feature accepts only allowed file types (e.g., PNG, JPG) and rejects others.

Example Test Case:

- **Input**: User enters a valid username and password, and submits the signup form.
- **Expected Output**: User is added to the database, and a success message is displayed.

6.3 White Box Testing

White Box Testing (also known as Structural Testing) requires knowledge of the internal logic of the code. The tester can see the code and tests internal components, such as control flow, data flow, and logic paths.

- **Goal**: To test internal workings of the system, ensuring the implementation is correct and there are no hidden bugs.
- **Tools**: Tools like coverage.py can measure how much of the code is covered during testing.
- **Example**: You can test functions like allowed_file() and processImage() by directly feeding different filenames or image paths into them.

6.4 Integration Testing

Integration Testing focuses on testing the interaction between integrated components or systems. It ensures that different parts of the application work together as expected.

- **Goal**: To identify any issues when combining different parts of the system, such as database interactions or communication between services.
- **Example**: In your Flask application, integration testing could test if:
  - A user can successfully sign up, log in, upload an image, and process the image without encountering errors.
  - The database successfully stores the user data and retrieves it when required.

6.5 Validation Testing

Validation Testing ensures that the software meets the business requirements and works as expected from the user's point of view. It verifies whether the system solves the intended problem.

- **Goal**: To ensure the application is built according to the specified requirements and user expectations.
- **Types**:
  - **Functional Validation**: Ensures the software provides the necessary functionality (e.g., user registration, image uploading).
  - **Non-Functional Validation**: Checks non-functional aspects, such as performance, security, and usability.

Example of Validation Test:

- **Requirement**: Users must be able to upload images in supported formats (PNG, JPG, etc.).
- **Test**: Upload a file in a supported format and verify that the system processes it correctly. Then, upload a file in an unsupported format (like .txt) and verify the rejection message.

6. 6 Acceptance Testing

Acceptance Testing is the final phase of testing to verify if the software meets all business requirements and is ready for deployment. It is typically done by the end users or stakeholders.

- **Goal**: To determine whether the system satisfies the business needs and whether it is acceptable for delivery.
- **Types**:
  - **Alpha Testing**: Performed by the internal development team or QA team before the software is released to the client.
  - **Beta Testing**: Performed by the end users to identify any issues in a realworld environment.

Example of Acceptance Test:

- **Test Case**: Ensure that after a user signs up and uploads an image, they are able to see a processed version of the image and download it.
- **Criteria**: If the user is able to perform these tasks successfully without issues, the software is deemed ready for release.

Chapter 7: CONCLUSION

This project **AI Image Editor** aimed to develop a user-friendly image processing web application using Flask. The application allows users to sign up, log in, upload images, and apply various image processing operations such as rotating, converting formats, and grayscale transformations. The core features of the system include user authentication, image upload with support for multiple formats, and seamless image processing functionality.

Throughout the development, best practices such as modularity, security, and user experience were prioritized. The project was also tested thoroughly using various testing techniques, including unit testing, integration testing, and acceptance testing, to ensure all components function as expected and the application meets the users' requirements.

The database design was implemented using SQLite, ensuring efficient storage of user credentials and image metadata, while also maintaining data integrity through normalization techniques. This ensures scalability and ease of future enhancements.

In conclusion, this project successfully demonstrates a functional and secure web application for image processing, and it can be expanded further by adding more features like image compression, additional formats, or even integrating machine learning models for advanced image manipulation. With proper testing and validation, the application is ready for real world use, offering a valuable tool for users to manage and process images online.