## **Requirements**

### **Functional Requirements**

1. Image Upload: Users can upload images in various formats (JPEG, PNG, GIF).
2. Image Compression: The system compresses images using configurable settings.
3. Download Optimized Images: Users can download processed images.

### **Non-Functional Requirements**

1. Performance: Image optimization must occur in less than 2 seconds for each image.
2. Scalability: The system should handle up to 1000 concurrent users.
3. Security: All data must be encrypted during transmission.

### **User Stories**

1. As a user, I want to upload images so that I can optimize them for my website.
2. As a user, I want to download optimized images to use in my projects.
3. As a user, I want to view the original and optimized file sizes for comparison.

### **Compliance Standards**

* GDPR for data protection
* Web Content Accessibility Guidelines (WCAG)

## **System Design**

### **High-Level Architecture**

**System Architecture Diagram**

### **Components**

1. Frontend: User interface for image upload and management.
2. Backend: Handles image processing using Flask and OpenCV.

### **Interaction Flow**

* User uploads an image via the frontend.
* The backend processes the image and returns the optimized image for download.

## **Technologies Used**

* Frontend: React.js
  + *Reason*: High performance, reusable components, and a large ecosystem.
* Backend: Flask
  + *Reason*: Lightweight framework suitable for building RESTful APIs.
* Image Processing Library: OpenCV
  + *Reason*: Powerful library for image processing tasks.

## **API Specifications**

### **Endpoints**

* POST /api/images/upload
  + *Request*: Multipart form data with image file.
  + *Response*: JSON with optimized image URL.
* GET /api/images/:id
  + *Response*: JSON with image metadata and download link.

### **Error Codes**

* 400: Bad Request
* 404: Not Found
* 500: Internal Server Error

### **Security Measures**

* HTTPS for secure data transmission.

## **Data Models and Storage**

### **Data Flow**

* The system does not require persistent storage for images; all processing occurs in-memory.

## **Implementation Details**

### **Algorithms and Design Patterns**

* Image Compression Algorithm: Implemented using OpenCV functions for efficient image compression.
* Design Pattern: MVC (Model-View-Controller) for separating concerns in the application structure.

## **Deployment Plan**

* Deployment Pipeline: CI/CD processes will be set up using GitHub Actions or similar tools.
* Hosting Environment: The application will be hosted on a cloud platform (e.g., Heroku, AWS EC2).

### **Monitoring and Logging Strategies**

* Use logging libraries to capture application logs for monitoring and debugging.

## **Testing Plan**

### **Testing Methodologies**

* Unit Testing: Validate individual components using frameworks like pytest.
* Integration Testing: Ensure that different parts of the application work together seamlessly.