CSE 611 Project Development: Photo Editor

Documentation

Pavana Lakhsmi Venugopal
pavanala@buffalo.edu

Saad Ahmed
saadahm2@buffalo.edu

Taraka Rohit Adusumilli
tarakaro@buffalo.edu

Ayesha Humaera
ayeshahu@buffalo.edu

This project was conducted under the guidance of
Professor Jinjun Xiong.

Contents

1 Introduction and Motivation

1.1 Overview

The Photo Editor Web Application is a versatile and user-friendly tool designed for enhancing and manipulating digital images. It empowers users with a range of features for photo editing, including cropping, resizing, adjusting brightness, contrast, creating collages, mosaics, merging images into a PDF, background change, noise removal, image compression, and video compression with various options.

Gone are the days of struggling with complex software or settling for basic photo editing tools. The Photo Editor Web Application emerges as a game-changer, empowering individuals of all skill levels to transform their digital images with ease and precision. Whether you're a seasoned photographer seeking advanced editing power, a hobbyist exploring your creative side, or simply looking to enhance everyday snapshots, Photo Editor welcomes you with a comprehensive set of features and an intuitive interface designed to unleash your inner photo maestro.

Imagine generating professional-grade passport photos with just a click. Photo Editor's unique passport creation feature allows you to customize your background and seamlessly adjust cropping based on specific country requirements. No more trips to the photo studio – the world of compliant passport photos is now at your fingertips.

Crop unwanted elements, resize images to perfection, banish noise for a crisp finish, or stitch multiple pictures into captivating collages. For a touch of artistic flair, Photo Editor lets you apply mesmerizing mosaic effects and play with brightness and contrast to create moods and atmospheres that resonate with your vision.

Worried about compatibility? Photo Editor embraces the most popular image formats, including JPEG and PNG, ensuring seamless integration with your existing workflow. And for those seeking to optimize storage or share their creations effortlessly, the built-in compression features offer a valuable ally.

In a landscape of limited photo editing options, the Photo Editor project emerges as a beacon of accessibility and empowerment. Aiming to bridge the gap between basic tools and complex software, it offers a user-friendly suite of features to individuals of all skill levels. From casual users and hobbyists to professionals and businesses, Photo Editor democratizes image manipulation, inviting everyone to unleash their creative potential.

Its development journey, however, wasn't without its hurdles. Balancing the desire for advanced features with user-friendliness, the team meticulously crafted an intuitive interface. They wrestled with the intricacies of perfect balance between photo quality and processing speed. Through these challenges, Photo Editor was forged, ready to unleash a world of creative possibilities for everyone.

1.2 Problem Statement

The digital photo editing landscape is fragmented, leaving users frustrated with limited functionalities and scattered feature sets. Popular tools like Canva, Ribbet, BeFunky, Photoshop Express, and PicsArt offer basic editing capabilities but lack crucial features for specific needs. Notably, there is an absence of a comprehensive solution for passport photo creation that includes face and pose detection, spectacles recognition, and background alteration to comply with diverse country requirements. Furthermore, access to advanced functionalities like mosaic effects, image format conversion, and image/video compression is often limited: available options are either paid, ad-infested, or restrict user freedom. This lack of a unified, user-friendly, and secure platform necessitates a reliable web application that eliminates these limitations and empowers users with a comprehensive, free, and secure photo editing experience.

1.3 Scope of the Product

The photo editor project is a web application designed for use by UB Faculty/students on any computer. The application features a user-friendly interface, enabling users to effortlessly improve and edit their digital photos. The project encompasses functionalities like cropping, resizing, adjusting brightness and contrast, creating collages, applying mosaic effects, generating PDFs, changing backgrounds, eliminating noise, and performing both image and video compression.

Key Features:

- **Advanced Passport Creation:** A unique feature allows users to enable/disable the apply white background' option, offering the choice between using high-end tools or minimalistic features for generating passport images. It offers a wide range of country selection and cropping based on specific country requirements.
- **Editing Tools:** Users can enjoy tools such as cropping, resizing, noise removal, collage creation, mosaic application, and brightness/contrast adjustment.
- **Image Format Support:** The application supports various image formats, including JPEG and PNG, ensuring compatibility with commonly used file types.
- **Compression Features:** Users can compress both images and videos, optimizing file sizes for efficient storage and sharing.
- **User-Friendly Interface:** The software boasts a clean and intuitive interface, providing a seamless experience for users to navigate and apply editing tools with ease.
- **Output Options:** Users have the flexibility to save edited images in different file formats.

This scope encapsulates the fundamental functionality of the Photo Editor project. Additional features and enhancements may be incorporated over time in response to user feedback and market demand.

2 Technical Details

2.1 System Architecture and Designs

2.1.1 Frontend

The frontend of the "CSE 611 Photo Editing" application is meticulously crafted using React, a powerful JavaScript library for building user interfaces. React's component-based architecture allows for each editing feature to exist as an independent module, ensuring a clean separation of concerns and facilitating ease of maintenance and scalability.

The application is structured to provide users with an intuitive flow, beginning with a homepage that showcases the diverse editing features available. Upon selecting a feature, the user is directed to the respective feature page, which is dynamically loaded to present the specific functionality offered. The flowchart provided in the documentation illustrates this user journey, outlining the decision points and subsequent actions leading to a complete editing process.

Each feature component, such as "Passport Photo Creation," "Mosaic," and "Noise Removal," is designed with a focus on user experience. Instructions are provided to guide users through the editing process, which includes uploading images or videos, selecting editing parameters, and initiating the editing process.

The frontend interfaces with the backend through well-defined API calls, triggering the necessary processing. In the event of an error during this interaction, the application is designed to capture these exceptions and provide the user with clear error messages, allowing them to understand and rectify the issue, or retry the operation.

The following features highlight the capabilities of the frontend architecture:

- **Passport Photo Creation:** Users can generate passport-compliant photos with tools to adjust size and background as per global standards.
- **Mosaic:** A feature that lets users blend multiple images into a mosaic layout, providing an artistic assemblage of pictures.
- **Format Conversion:** Offers versatility in image handling by allowing conversions between various file formats.
- **Crop & Resize:** Basic yet essential tools that give users control over image dimensions and scaling.
- **Background Change:** Empowers users to manipulate the image background, aiding in creating professional-looking photos.
- **Image & Video Compression:** These tools are crucial for reducing file sizes, which is particularly beneficial for storage and web use.
- **Collage:** Enables the creation of a photographic collage, combining memories into a single image.
- **PDF Conversion:** Facilitates the conversion of images to PDF format and vice versa, adding to the application's utility.
- **Brightness and Contrast Adjustment:** Interactive sliders allow users to fine-tune the visual aspects of their images.
- **Noise Removal:** A sophisticated tool designed to enhance image quality by reducing unwanted noise.

In conclusion, the frontend of the "CSE 611 Photo Editing" application offers a robust, user-friendly interface backed by the solid, component-driven architecture of React. It is designed to facilitate a seamless editing experience while providing high-quality results that users can download and utilize. The modular design approach allows for future enhancements and the addition of new features, ensuring the application remains cutting-edge and user-centric.

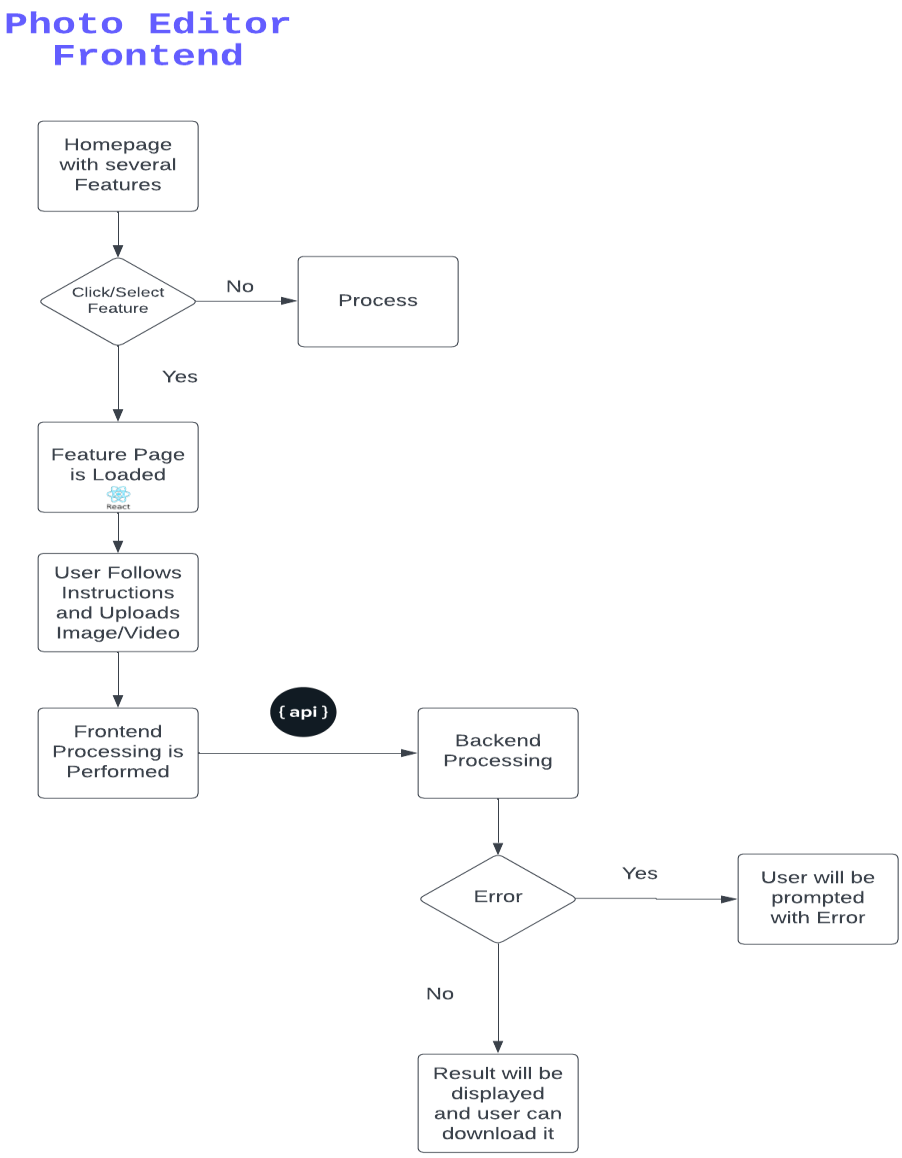


Figure 1: Frontend Architecture

2.1.2 Backend

The backend architecture of the "CSE 611 Photo Editing" application is a sophisticated orchestration of microservices, each dedicated to a specific feature of the photo editing platform. This design facilitates a distributed system that is both scalable and resilient, capable of handling specific tasks in isolation and ensuring uninterrupted service delivery.

Utilizing the Django REST framework, a powerful and flexible toolkit for building Web APIs, each microservice is crafted to serve a distinct editing function such as Passport Photo Creation, Mosaic Generation, Format Conversion, and others. This choice of framework allows for the creation of clean, RESTful APIs that provide efficient communication between the client-side and server-side components.

The backend services are deployed within the University at Buffalo's Kubernetes Cluster, which offers a container orchestration platform that automates the deployment, scaling, and operation of application containers. Kubernetes provides a high degree of fault tolerance, self-hehealing, and zero-downtime deployments, which are essential for maintaining a robust backend for an application that may experience variable load patterns.

Each microservice is encapsulated as a modular service, allowing for independent updates and maintenance without affecting the entire system. This modularity also aids in the development process, as teams can work on separate services simultaneously without overlap, increasing development speed and efficiency.

The Backend Architecture flow begins with the client side, where the user interacts with the application through a web browser. The user's requests are then sent to the API Gateway, which is the entry point for all backend requests. The API Gateway, built with the Django REST Framework, handles these requests and performs initial validations, such as checking for correct input parameters, before routing them to the appropriate microservice.

The microservices cover a wide array of functionalities:

- **Passport Photo Microservice:** Manages the creation and formatting of passport-sized photos according to international standards.
- **Mosaic Maker Microservice:** Facilitates the combination of multiple images into a single mosaic image.
- **Format Conversion Microservice:** Provides the capability to convert images from one file format to another.
- **Image Compression Microservice:** Reduces the file size of images while aiming to maintain quality.
- **Crop and Resize Microservice:** Offers tools to alter the dimensions and scale of an image.
- **Video Compression Microservice:** Similar to image compression, it decreases video file sizes for easier sharing and storage.
- **Background Change Microservice:** Enables users to alter or remove the background of an image.
- **Collage Microservice:** Allows the combination of several images into a collage layout.
- **PDF, Brightness and Contrast, Noise Removal Microservice:** A versatile service that handles document conversion to PDF, adjusts image brightness and contrast, and removes noise from images.

After processing, the results are returned through the API Gateway. If an error occurs during processing, it is communicated back to the user with informative error messages, allowing for troubleshooting or alternative actions. The final processed results are then made available to the user, typically with options to review and download the edited content.

Kubernetes' orchestration ensures that each microservice can be scaled according to demand, and system resources are optimized across the cluster. This backend architecture not only provides a robust, maintainable, and scalable system but also ensures a seamless and responsive user experience on the frontend.

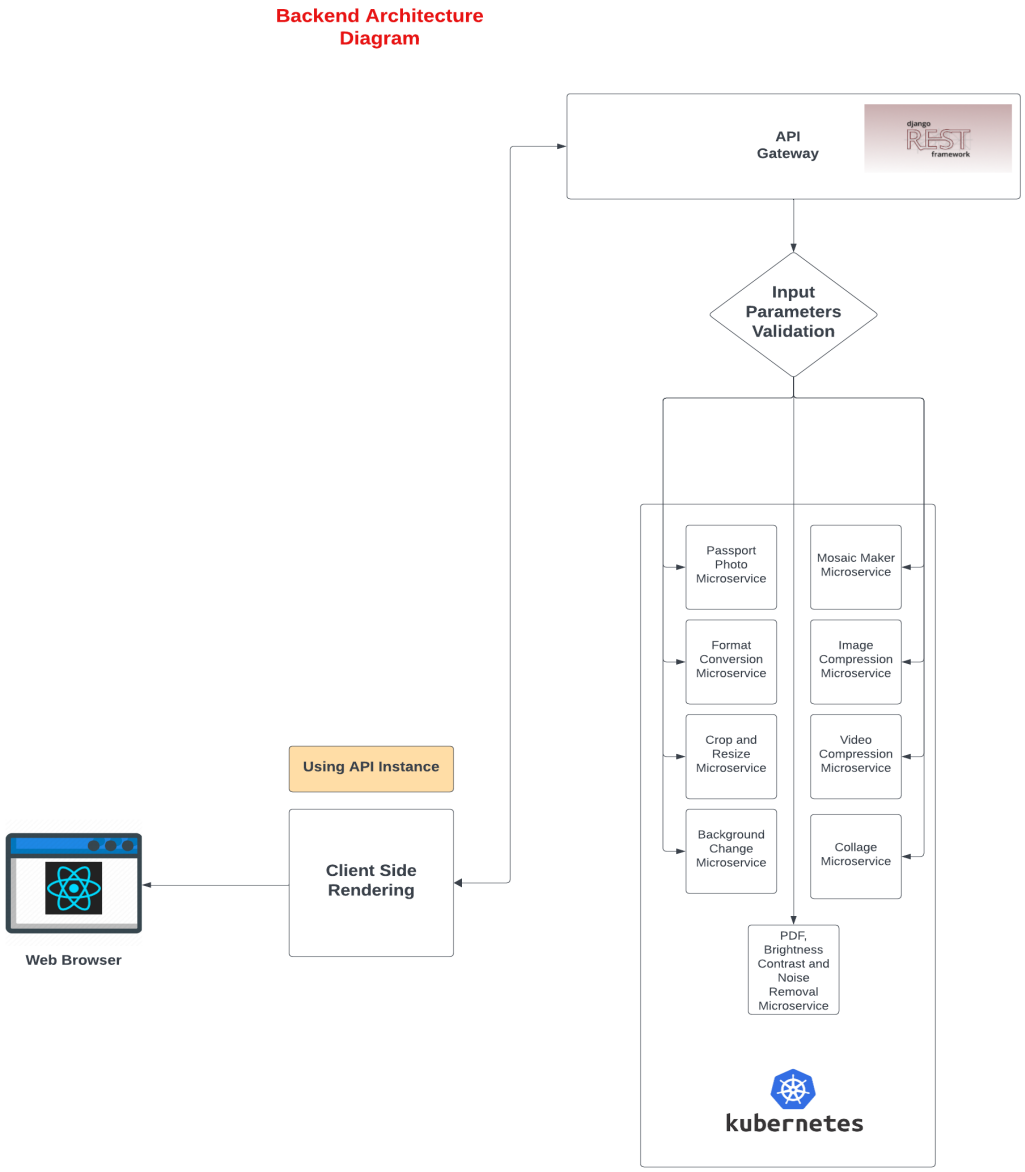


Figure 2: Backend Architecture

2.2 Business Case for the Product

The Photo Editor product addresses the growing need for high-quality images, catering to both personal and professional uses. This product presents several key benefits:

- **Increased Productivity:** The intuitive design and efficient processing capabilities streamline image editing tasks, significantly enhancing productivity for users.
- **Improved Image Quality:** Advanced features and tools within the product ensure a higher standard of image quality, meeting the expectations of users requiring professional-grade results.
- **Increased Reach:** By catering to a broad spectrum of needs, from personal photo enhancements to professional branding, the product extends its reach to a diverse user base.
- **Increased Revenue:** The product's versatility and quality make it an attractive option for commercial use, potentially leading to increased revenue streams.
- **Competitive Advantage:** With its comprehensive suite of features, the product stands out in the market, offering users a competitive edge in image editing and creation.

Notably, the product is particularly beneficial for students requiring passport-sized photos for job applications, presentations, and other academic or professional purposes. This specific functionality highlights the product's ability to meet crucial, real-world needs in a convenient and user-friendly manner.

3 Implementation

3.1 Tech Stack

Front End:
The project employs a robust tech stack for a seamless user experience. React.js is utilized to construct the frontend UI, offering the advantages of reusable components and efficient DOM updates. HTML and CSS handle the visual styling, ensuring a polished layout for the web pages. JavaScript orchestrates operations on the frontend, facilitating communication with APIs and enhancing interactivity. Bootstrap plays a pivotal role in making the website responsive, providing predefined UI designs for components like buttons, forms, and navigation bars.

Back End
The backend of this project is structured around the Django framework, a high-level Python web framework that adheres to the model-view-controller architectural pattern. For image processing, the backend relies on OpenCV, an open-source computer vision library implemented in C++. This integration empowers the system with capabilities for intricate image recognition, manipulation, and analysis. The backend's image rendering and manipulation capabilities are facilitated by Pillow, a powerful library supporting various image file formats and providing essential functionality for image processing tasks.

Deployment
The deployment strategy involves utilizing a Kubernetes (k8s) cluster to manage and orchestrate the application's containers. This infrastructure choice reflects a commitment to streamlined deployment processes, scalability, and resilience. The k8s cluster ensures that the application is deployed, scaled, and managed efficiently, contributing to a reliable and optimized user experience.

3.2 Front End Implementation

Building upon the existing foundation of React, HTML, and CSS, we have continued to refine and expand the visual layout of our web pages. This iterative process enables us to maintain a polished and aesthetically pleasing user interface (UI), aligning with the evolving design principles and user experience goals of the project. JavaScript plays a crucial role in extending the frontend's functionality, allowing us to implement new features, enhance existing ones, and ensure smooth user interactions.

The project structure for the Front End is as follows:

- **Public Directory:** This directory serves as the root for static assets exempt from the build process, encompassing essential elements such as images, fonts, and external libraries. Housing these assets directly avoids unnecessary processing, ensuring swift and direct inclusion in the application.
- **Src Directory:** This directory contains essential files for the project. It includes `home.js`, a pivotal component that contributes to the construction of the landing and home pages. This file plays a central role in defining the initial user interface and setting the stage for a seamless user experience. The directory also contains Components and MainComponents folders.
- **MainComponents Directory:** Dedicated to feature components, this directory embodies the frontend code for each feature microservice. While the backend is structured as microservices, the frontend maintains cohesion as a singular entity. These components are designed for reusability and modularity, encapsulating specific features or sections of the user interface, thus enhancing maintainability and scalability.
- **Dockerfile:** The presence of a Dockerfile highlights our commitment to containerized deployment. This file serves as a blueprint for building a Docker image, detailing the base image, dependencies, environment variables, and the necessary commands to set up and execute the application within a container.

3.3 Backend Microservice Implementation

3.3.1 Passport Microservice

Port: 8010
Libraries Used:

- cv2 (OpenCV): Used for image processing and computer vision.
- numpy (NumPy): Used for numerical operations on arrays.
- dlib: A toolkit for machine learning and computer vision, used here for face detection and facial landmark prediction.

Functions:

- **passport_photo_face:** This function is called by `passport_photo_size`. It loads an image, detects faces using a frontal face detector from dlib, extracts facial landmarks, draws the landmarks on the image, and saves the modified image to an output URL. The function also returns a string indicating whether a face was detected, the full face was not detected, or no face was detected.
- **passport_photo_size(request):** It starts by performing initial checks on the input, extracting relevant information, and verifying the country specified in the request. The face detection step utilizes the `passport_photo_face` function to identify faces. Following face detection, the function proceeds with pose detection using a `pose_detector` function. It then performs image processing based on the specified country's data, resizing the image, changing the background to white, and optionally adding a border.

Output: The output URL is prepared, and the processed image is converted to base64 format for display.

3.3.2 Resize and Crop Microservice

Port: 8011
Libraries: cv2
Function:

- **image_resize:** This function utilizes the OpenCV library to resize an input image to a specified width and height. It first reads the input image and checks if either width or height is specified. If only one dimension is provided, the other is calculated to maintain the aspect ratio. The function then performs the resizing using the specified interpolation method, defaulting to `cv2.INTER_AREA`, and saves the resized image at the specified output location. The function returns the resized image, allowing for further processing or analysis if needed.

Output: The output URL is prepared, and the processed image is converted to base64 format for display.

3.3.3 Collage Microservice

Port: 8012
Function:

- **photo_collage:** The photo_collage function handles the web request. It starts by initializing a dictionary to store the response. The code attempts to verify the function passed in the request and processes a POST request if applicable. If successful, it returns a response with a success message, error status set to False, and the updated statistics.

Output: The function returns a response containing a dictionary with information about the success or failure of the photo_collage update operation, along with relevant status codes and messages.

3.3.4 PDF, Noise and Brightness Microservice

Port: 8013
Libraries:

- PIL (Python Imaging Library)
- OpenCV (Open Source Computer Vision Library)

Functions:

- **convert_images_to_pdf:** Utilizes the Python Imaging Library (PIL), specifically the Image module, to convert a collection of images stored in a designated folder into a consolidated PDF file. It begins by initializing an empty list, images, to store the processed images. The function iterates through each file in the specified folder, selecting only those with valid image file extensions such as ".jpg," ".png," or ".jpeg." For each eligible file, the function opens the image using PIL's Image.open method and checks if it is in RGBA mode; if so, it converts it to RGB mode. The processed images are then appended to the images list. Finally, the function saves the first image in the list as a PDF, incorporating all subsequent images in their original order.
- **Noiseremoval:** Employs the OpenCV library (cv2) to perform noise reduction on an input image. It takes two parameters. The function begins by reading the input image using cv2.imread and stores it in the variable img. It then applies a bilateral filter using cv2.bilateralFilter. The bilateral filter is a non-linear filter that preserves edges while reducing noise. In this case, it uses a kernel size of 15, and the sigma values for color and space are both set to 75. The denoised image is obtained through this filtering process.

3.3.5 Format Conversion Microservice

Port: 8014
Libraries:

- Python Imaging Library (PIL)
- pillow_heif
- cv2 (OpenCV)

Function:

- **format_change:** This function begins by performing initial checks and extracting relevant information from the request. It then updates statistics related to format changes, retrieves the desired output format, and dynamically converts the image using either Pillow for GIF, HEIF, and HEIC formats, or OpenCV for other formats. The output URL is constructed, and the function logs information about the conversion process.

Output: The function returns a dictionary (return_dict) containing information about the processing status, output URL, and the image in base64 format. The output URL is constructed based on the input image's name and the desired format specified in the request.

3.3.6 Background Change Microservice

Port: 8015

Libraries:

- cv2: OpenCV library for image processing tasks.
- rembg: A library for removing image backgrounds.
- PIL (Python Imaging Library): Used for opening, manipulating, and saving images.

Functions:

- **color_bg_and_add_border:** This function takes an input image path, an output image path, background color (bg_color), border color (border_color), and optional parameters for border thickness (border_thickness). It uses the rembg library to remove the background of the input image, then adds a border to the image using OpenCV (cv2). The resulting image is saved at the specified output path.
- **background_change:** This function takes a request object, processes the input image, changes its background color, adds a border, and returns a response dictionary. It calls the color_bg_and_add_border function to perform the background change.

Output: The output URL is prepared, and the processed image is converted to base64 format for display.

3.3.7 Mosaic Microservice

Port: 8016

Libraries:

- PIL (Python Imaging Library): Used for image processing.
- numpy: Used for handling arrays and numerical operations.
- random: Used for generating random choices.

Functions:

- **load_image():** Loads and returns a resized and converted RGB numpy array of the main image.
- **mosaicmaker():** The mosaic_maker function is designed to transform a given main image into a visually appealing mosaic by incorporating tiles from a specified list of images. The main image, denoted as face_im_arr, is initially loaded and resized to a fixed dimension of 1024x1024 pixels. The function then iterates over the main image in a grid pattern based on a user-defined pixel_size, randomly selecting tiles from the provided list (images_list). These tiles are resized accordingly and pasted onto the mosaic image, which is created with the same dimensions as the main image. The final step involves blending the original main image with the generated mosaic using the Image.blend method, resulting in a harmonious composition of the tiles and the underlying image.

Output: The resulting mosaic is saved at the specified output path.

3.3.8 Image Compression Microservice

Port: 8017

Libraries:

- cv2: OpenCV library for image processing tasks.

Functions:

- **compress_image():** This function attempts to compress the input image using OpenCV's `cv2.imencode` function with the specified output format and compression parameters. The compressed image is then converted to base64 encoding for easy transmission.
- **image_compression():** The image_compression function serves as an endpoint for handling POST requests related to image compression. It validates the presence of an uploaded file and checks for functionality verification. Using OpenCV, it decodes the image and updates statistical information. Depending on the conditions, it either applies standard compression based on the specified output format or performs custom compression to achieve a target size. The function orchestrates the image compression process by calling the compress_image method.

Output: The output of the image_compression function is encapsulated in a response dictionary, which contains details such as input and output image sizes, both expressed in kilobytes, the chosen output format, and an error status indicator.

3.3.9 Video Compression Microservice

Port: 8018

Libraries:

- os
- subprocess
- shlex
- ffmpeg

Functions:

- **compress_video:** This function is designed to compress a video file using the FFmpeg library. It utilizes FFprobe to determine the duration of the input video. Based on the specified target size, it calculates the total bitrate required and divides it into video and audio bitrates. The function then constructs an FFmpeg command using these calculated values and executes it to compress the video. If successful, the function returns 1; otherwise, it returns 0, indicating an error.
- **compress_video_by_resolution:** The compress_video_by_resolution function focuses on compressing a video based on a specified resolution and quality using FFmpeg. It calculates the Constant Rate Factor (CRF) based on the provided quality percentage. The function then constructs a command to scale the video to the desired resolution and compress it with the calculated CRF. Upon successful execution, it returns 1; otherwise, it returns 0, indicating an error in the compression process.
- **video_compression:** The video_compression function serves as an endpoint for video compression, designed to handle HTTP requests. Upon receiving a request, the function logs its initiation and proceeds to verify the presence of required parameters and functionality. Depending on the provided parameters, either the compress_video or compress_video_by_resolution function is invoked for video compression.

Output: If compression fails, an exception is raised; otherwise, the function retrieves information about input and output video sizes, encodes the compressed video in base64, and constructs a data URL. The final response, encapsulated in a dictionary, includes the video URL, file sizes, success or error status, and an appropriate HTTP status code.

3.4 Local Environment Setup

3.4.1 Front End

1. Clone the repository using the command:
   `git clone https://github.com/xlab-classes/cse611-fall-2023-team-photoediting.git`.
2. Navigate to the frontend directory using: `cd cse611-spring2023-team-photo-editing\frontend`.
3. Install node modules using: `npm install` or `npm install -force`.
4. Start your application using: `npm start`. You can view the application running on port 3000.

3.4.2 Back End

1. Clone the repository using the command:
   `git clone https://github.com/xlab-classes/cse611-spring2023-team-photo-editing.git`.
2. Navigate to the backend directory (`cse611-spring2023-team-photo-editing\backend\photo_editing_api`). This directory contains the backend code as a monolithic application.
3. To run each individual microservice, perform the following steps for each microservice:
   - Enter the microservice directory, e.g., for the passport service directory, run: `cd passport-service`.
   - Create a Python environment variable using: `python -m venv venv`.
   - Activate the virtual environment using: `.\venv\Scripts\Activate.ps1`.
   - Install all the requirements using: `pip install -r requirements.txt`.
   - After successfully installing all required packages, run: `python manage.py runserver {portnumber}`, for example, `python manage.py runserver 8010`.
   - For video compression service do the following:
     - Windows:
       * Download the release version of ffmpeg via the link - https://www.gyan.dev/ffmpeg/builds/ffmpeg-release-essentials.7z
       * Extract, copy to any location of your choice and copy the path to the bin folder (Eg - C:\ffmpeg\bin)
       * In Start menu, search for Environment Variables (User / System) and Edit PATH variable to include the above ffmpeg bin path
       * Restart the Terminal and verify ffmpeg command runs successfully.
     - Linux:
       * sudo apt install ffmpeg
   - Note: Make sure to change the respective URLs in the frontend microservice based on the port number of the service. For example, change the URL in `passportPhoto.jsx` file to `http://127.0.0.1:8010`.
4. Alternatively, to run the backend as a whole component and not as individual microservices, you can perform the following steps:
   - Enter the `photo_editing_api` directory by using: `cd photo_editing_api`.
   - Create a Python environment variable using: `python -m venv venv`.
   - Activate the virtual environment using: `.\venv\Scripts\Activate.ps1`.
   - Install all the requirements using: `pip install -r requirements.txt`.
   - After successfully installing all required packages, run: `python manage.py runserver`. By default, it will run on port 3000.
   - Note: Make sure to change the URLs in each frontend component; the following URL will remain constant for all features: `http://127.0.0.1:3000`.

4 Deployment

4.1 Dockerfile Configuration

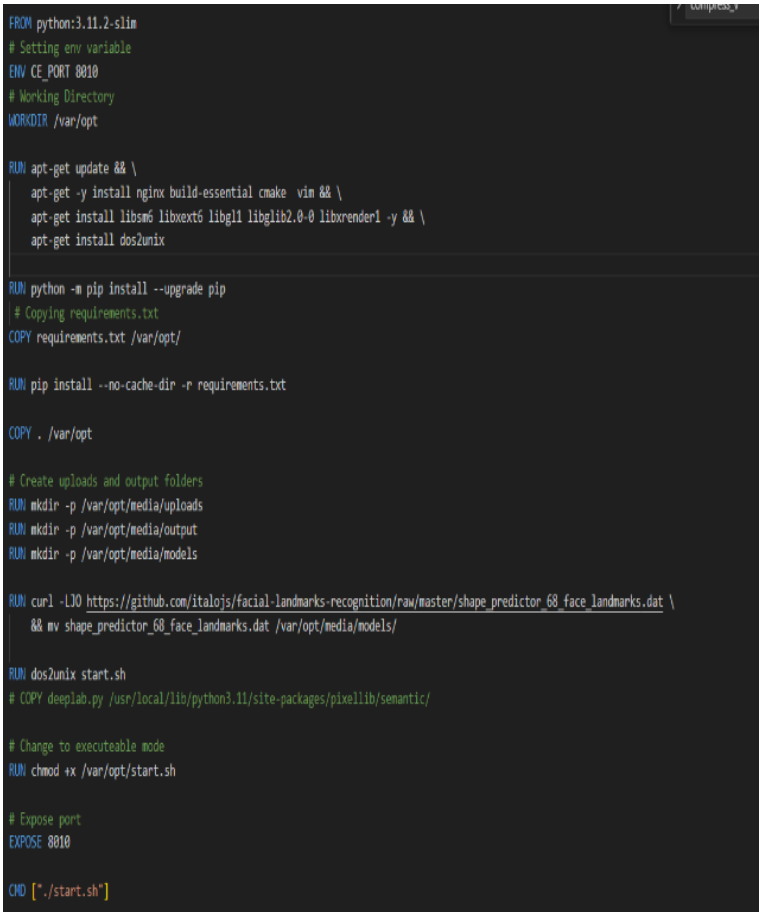
A Dockerfile serves as a fundamental component in the containerization process, offering a structured and reproducible way to encapsulate the configuration and dependencies required for an application's runtime environment. It is an essential tool within the Docker ecosystem, addressing the need for consistency, portability, and efficiency across different deployment environments. The Dockerfile used for building the frontend image is shown below:

```
1  # Fetching the latest node image on alpine linux
2  FROM node:latest AS development
3
4  # Declaring env
5  ENV NODE_ENV development
6
7  # Setting up the work directory
8  WORKDIR /cse611-spring2023-team-photo-editing
9
10 # Installing dependencies
11 COPY ./package.json /cse611-spring2023-team-photo-editing/
12 RUN npm install --force
13
14 # Copying all the files in our project
15 COPY . .
16
17 # Starting our application
18 CMD npm start
```

Figure 3: Frontend Docker Image

The Dockerfile begins by selecting the latest Node.js image based on Alpine Linux as the foundational environment for containerization. This choice ensures a lightweight and efficient base for building and running the application. To facilitate dependency management, the Dockerfile copies the local package.json file into the container's working directory. Following this, the npm install command is executed with the –force flag, ensuring a robust and clean installation of project dependencies. Concluding the Dockerfile, the CMD instruction specifies the default command to initiate the application—npm start. This command serves as the entry point for the container, defining the action to be executed when the container is launched.

Below is the Dockerfile for Passport Microservice :



```
FROM python:3.11.2-slim
# Setting env variable
ENV CE_PORT 8010
# Working Directory
WORKDIR /var/opt

RUN apt-get update && \
    apt-get -y install nginx build-essential cmake  vim && \
    apt-get install libsm6 libxext6 libgl1 libglib2.0-0 libxrender1 -y && \
    apt-get install dos2unix

RUN python -m pip install --upgrade pip
 # Copying requirements.txt
COPY requirements.txt /var/opt/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /var/opt

# Create uploads and output folders
RUN mkdir -p /var/opt/media/uploads
RUN mkdir -p /var/opt/media/output
RUN mkdir -p /var/opt/media/models

RUN curl -LJO https://github.com/italojs/facial-landmarks-recognition/raw/master/shape_predictor_68_face_landmarks.dat \
    && mv shape_predictor_68_face_landmarks.dat /var/opt/media/models/

RUN dos2unix start.sh
# COPY deeplab.py /usr/local/lib/python3.11/site-packages/pixellib/semantic/

# Change to executeable mode
RUN chmod +x /var/opt/start.sh

# Expose port
EXPOSE 8010

CMD ["./start.sh"]
```

Figure 4: Passport Microservice Docker Image

This Dockerfile outlines the process of preparing a container for a Python application, making it easier to deploy consistently across various environments. To make the application easily accessible, the Dockerfile designates a specific port (8015) and establishes a designated working directory (/var/opt). It then installs necessary tools and libraries, ensuring a smooth environment for the Python application to run. For seamless execution, a script called start.sh is prepared, which kick-starts the application when the container launches. The Dockerfile simplifies the process further by exposing the application through port 8015.

4.2 Building and Storing Images

The deployment architecture of the project involves the utilization of microservices, with the frontend of the entire application and backend components of each feature deployed as independent microservices on the xlab Kubernetes cluster. To streamline the deployment workflow, the xlab-class GitHub registry serves as the central repository for storing Docker images.

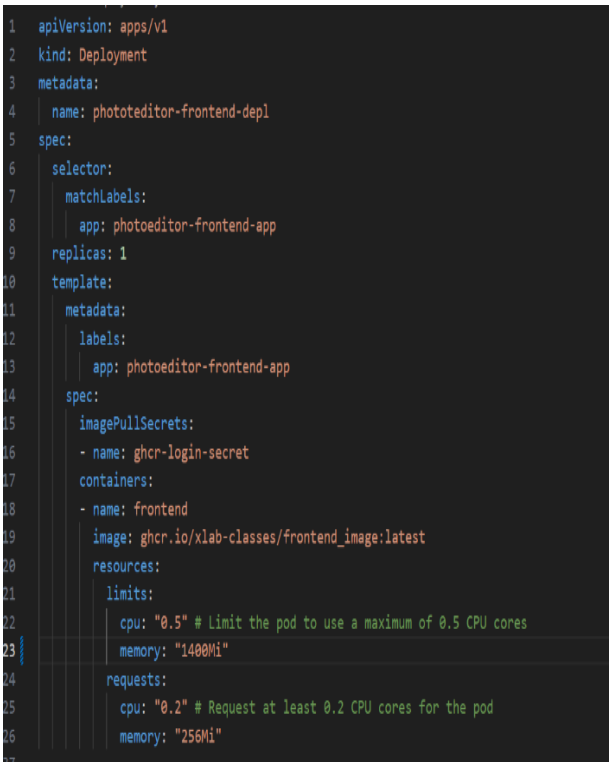
The deployment process is orchestrated through GitHub Actions workflows, specifically tailored for each service branch on the main repository. These workflows are configured to initiate the deployment pipeline upon changes to the specified paths for each microservice. This ensures a modular and efficient deployment approach.

Within each GitHub Actions workflow file, a series of defined steps orchestrate the deployment process. The workflow is triggered to initiate the build process for the Docker image associated with the microservice, facilitated by a Dockerfile residing within each specified path.

The automated nature of the GitHub Actions workflows ensures that any update or push to the designated paths triggers the entire deployment pipeline. This results in the seamless building and updating of Docker images on the ghcr registry, enabling a responsive and efficient deployment mechanism for the microservices on the xlab Kubernetes cluster.

4.3 Deployment Objects and Services Configuration

In order to deploy the microservices of the photo editor application, a Kubernetes Deployment was utilized. The deployment configuration for the frontend is defined in the YAML file below:



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phototeditor-frontend-depl
spec:
  selector:
    matchLabels:
      app: photoeditor-frontend-app
  replicas: 1
  template:
    metadata:
      labels:
        app: photoeditor-frontend-app
    spec:
      imagePullSecrets:
      - name: ghcr-login-secret
      containers:
      - name: frontend
        image: ghcr.io/xlab-classes/frontend_image:latest
        resources:
          limits:
            cpu: "0.5" # Limit the pod to use a maximum of 0.5 CPU cores
            memory: "1400Mi"
          requests:
            cpu: "0.2" # Request at least 0.2 CPU cores for the pod
            memory: "256Mi"
```

Figure 5: Frontend YAML

This configuration specifies the deployment's metadata, such as the name phototeditor-frontend-depl, selector labels- app: photoeditor-frontend-app, and desired number of replicas is set to 1. The pod template is defined to use a container named frontend with the specified Docker image ghcr.io/xlab-classes/frontend_image:latest. Resource constraints are set for CPU and memory, ensuring efficient utilization of resources. Additionally, an image pull secret named ghcr-login-secret is specified to authenticate with the container registry. To expose the deployed component of the photo editor application, a Kubernetes Service was implemented. The service configuration is outlined in the following YAML file:

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: phototeditor-frontend-servc
spec:
  type: NodePort
  selector:
    app: photoeditor-frontend-app
  ports:
  - name: my-app
    port: 3000
    nodePort: 31110
    targetPort: 3000
```

Figure 6: Server Configuration YAML

This Service, named `phototeditor-frontend-servc`, serves as the entry point for accessing the frontend application. The key components of the Service configuration include:

- **Service Type:** The service is of type NodePort, allowing the application to be accessed externally via a static port on each Node in the cluster.
- **Selector:** The service selects pods with the label `app: photoeditor-frontend-app`, directing traffic to the corresponding pods.
- **Ports Configuration:** The service exposes port 3000 internally, and it is mapped to the node port 31110 externally. The `targetPort` is set to 3000, ensuring that the incoming traffic is directed to the correct port on the selected pods.

The deployment components of other microservices were exposed in a similar manner but with different NodePorts. Below is the list of NodePorts and TargetPort for each microservice:

Service	**NodePort**	**TargetPort**
Frontend	31110	3000
Passport	30010	8010
Resize	30011	8011
Collage	30012	8012
PDF, Noise and Brightness	30013	8013
Format Conversion	30014	8014
Background-Change	30015	8015
Mosaic	30016	8016
Image-Compression	30017	8017
Video-Compression	30018	8018

5 Testing

5.1 Functional Testing - Manual

5.1.1 Passport Photo Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
User uploads an image, selects a country, uses crop frame and zoom, and clicks preview.	A preview image loads beside the original, downloadable by the user.	Working as Expected
User uploads an image, selects France', checks apply white background', uses crop frame and zoom, and clicks preview.	A preview with a white background loads beside the original, downloadable by the user.	Working as Expected
User selects a country from the dropdown in the sidebar.	A link to the official website with passport photo requirements for that country appears.	Working as Expected

5.1.2 Background Change Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
The user uploads an image, selects a color, and clicks preview.	The image's background changes to the selected color.	Working as Expected

5.1.3 Noise Removal Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
User uploads an image and clicks preview/download.	The image is cleaned of distortions or noise.	Working as Expected

5.1.4 Brightness and Contrast

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
User uploads an image and selects brightness and contrast using sliders in the sidebar.	A preview image with adjusted brightness and contrast loads dynamically beside the original image based on user dynamic adjustments.	Working as Expected

5.1.5 Resize Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload an image and provide width and height, the output image should be resized to needed specifications.	Modifies the width and height of an image as per user input, and generates a resized image that matches the specified width and height parameters.	Working as Expected

5.1.6 PDF Maker

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload multiple images in the sidebar and click generate PDF button on homepage.	A PDF with combined images is generated and downloaded.	Working as Expected
Users upload multiple images in the sidebar and clicks remove buttons on some images.	A PDF with only present combined images is generated and downloaded.	Working as Expected

5.1.7 Mosaic Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload a template in the left sidebar and select a pixel size value in the slider and upload more than 6 images in the right sidebar and click the preview image button.	The tool should generate a mosaic based on the options selected.	Working as Expected
After the preview image is already available, the user selects a different pixel size value and clicks the preview button.	The tool should modify the previously generated mosaic image and generate a new preview image.	Working as Expected

5.1.8 Crop Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload an image and choose one of the 4 ratio options available.	The tool should generate a cropped image based on the selected aspect ratio.	Working as Expected
Users upload an image, choose one of the 4 ratio options, and click the download button.	The tool should generate a cropped image based on the selected aspect ratio, and the cropped image is downloaded.	Working as Expected
Users upload an image and enter custom aspect ratio in width and height fields, then click the download button.	As the user enters custom ratios, the crop frame dynamically updates, and the user can see the preview and the image is downloaded.	Working as Expected
Users click switch to circle crop, upload an image, crop the image in the interaction area, and click the download button.	The image cropped in circular format is downloaded.	Working as Expected
Users click switch to circle crop, enter custom width and height ratio, upload an image in the interaction area, and click the download button.	The image cropped in circular format is downloaded.	Working as Expected

5.1.9 Collage Feature

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users select a template in the left sidebar, upload images, and click download collage.	The tool should generate and download a collage image.	Working as Expected
Users select a template in the left sidebar and click split in one of the cells.	The cell in the template splits up into two cells.	Working as Expected
Users select a template in the left sidebar and click merge rows when cells are in a split state in one of the cells.	The cell rows in the template are merged back.	Working as Expected

5.1.10 Format Conversion

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload a digital image and choose an image format in the left sidebar.	The tool must accept any image format and change the file format of the image to the selected format while preserving the visual content of the image.	Working as Expected

5.1.11 Image Compression

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload an image and select custom compression in the left sidebar, enter an expected file size value in kb, and select the resultant image file type option.	The tool compresses the image, and the user can view input file size and output file size, along with a download option.	Working as Expected
Users upload an image and select a compression rate in the left sidebar, choose a value on the slider, and select the resultant image file type option.	The tool compresses the image, and the user can view input file size and output file size, along with a download option.	Working as Expected

5.1.12 Video Compression

Input (Functions/Scenarios)	Output (Expected Behavior)	Result
Users upload a video, select custom compression in the left sidebar, enter an approximate value in MB, and click the compress/preview button.	The tool compresses the video, and users can view input file size and output file size, along with a download option.	Working as Expected
Users upload a video, select video resolution in the left sidebar, choose resolution quality on the slider, and select video resolution in MB, then click the compress/preview button.	The tool compresses the video, and users can view input file size and output file size, along with a download option.	Working as Expected

5.2 Backend Microservices Testing - Custom Script

Documentation: Backend API Testing with Custom Backend Script

5.2.1 Overview

This guide explains how to set up and use a custom Python script (`test_apis.py`) along with a pytest configuration (`conftest.py`) for testing backend APIs. The primary goal of these scripts is to automate the process of sending requests to various API endpoints and validating the responses, ensuring the backend behaves as expected. Additionally, the setup provides a custom summary of the test results, including the number of tests passed, failed, and the percentage of success.

5.2.2 Prerequisites

- Python installed on your system.
- pytest and requests libraries installed. You can install them using pip:

```
pip install pytest requests
```

5.2.3 Setup

Test Script (`test_apis.py`): This script contains individual test functions for different API endpoints. Each function sends a request to an endpoint and asserts the expected response.

pytest Configuration (`conftest.py`): This script is used by pytest for custom test execution behavior. It includes a hook that runs after all tests have been executed to print a summary of the test results.

5.2.4 Test Script Structure (`test_apis.py`)

- Function to Get Hosted URLs: `get_service_url(service_name)` returns the base URL for a given service.
- Test Functions: Each test function (e.g., `test_passport_photo_size`) makes an API request and asserts the expected outcome.
- Adding More Test Functions: To test additional endpoints, define new functions following the existing pattern:

```
def test_my_new_endpoint():
    url = f"{get_service_url(my_new_service')}/endpoint/"
    response = requests.post(url, data={...})
    assert response.status_code == 200
    assert Expected Response' in response.json()[message']
```

- List of Test Functions: The `test_functions` list at the end of the script is a collection of all test functions.

5.2.5 pytest Configuration (`conftest.py`)

- Custom Test Summary: The `pytest_sessionfinish` hook in `conftest.py` calculates and prints a summary of test results after all tests have been executed.

5.2.6 Running the Tests

Navigate to the directory containing `test_apis.py` and `conftest.py`. Run the tests using the pytest command:

```
pytest
```

pytest will automatically find and execute the test cases, and the custom summary will be printed at the end.

5.2.7 Custom Summary

The summary includes the total number of tests, the number of passed and failed tests, and the percentage of tests passed. This information is useful for quickly assessing the overall health of the backend API.

5.2.8 Expanding Test Coverage

- To add more tests, simply define new test functions in `test_apis.py`.
- Make sure each test function follows the pattern of sending a request and asserting the expected response.
- Add the new test function to the `test_functions` list.
- pytest will automatically include these new tests in the next execution.

5.2.9 Conclusion

This setup provides a robust framework for automated backend API testing. By adding new test functions as needed and leveraging the power of pytest, you can efficiently validate the behavior of your backend services and quickly get a summary of your test suite's health.

5.3 API Documentation

This API documentation is designed to provide users with information on how to interact with the "CSE 611 Photo Editing" application backend services. The services are accessed via RESTful endpoints, and actions are performed using the HTTP POST method.

Base URLs

Each service has its own unique base URL:

- Passport Photo Size: `http://xlabk8s3.cse.buffalo.edu:30010`
- Format Change: `http://xlabk8s3.cse.buffalo.edu:30014`
- Crop: `http://xlabk8s3.cse.buffalo.edu:30011`
- Noise Removal: `http://xlabk8s3.cse.buffalo.edu:30013`
- Background Change: `http://xlabk8s3.cse.buffalo.edu:30015`
- Brightness and Contrast: `http://xlabk8s3.cse.buffalo.edu:30013`
- Resize: `http://xlabk8s3.cse.buffalo.edu:30011`
- Mosaic Maker: `http://xlabk8s3.cse.buffalo.edu:30016`
- PDF Maker: `http://xlabk8s3.cse.buffalo.edu:30013`
- Video Compression: `http://xlabk8s3.cse.buffalo.edu:30018`
- Image Compression: `http://xlabk8s3.cse.buffalo.edu:30017`

Endpoints

Each endpoint corresponds to a service that the API provides. Below are the endpoints and their functions:

Passport Photo Size Service

- **Endpoint:** `/passport_photo_size/`
- **Method:** POST
- **Description:** This service manages the creation and formatting of passport-sized photos to meet international standards.
- **Data Params:**
  - `function: "passport_photo_size"`
  - `country`: The country for which the passport photo is being formatted.
  - `background_req`: Specifies if a background adjustment is required ("yes" or "no").

Format Change Service

- **Endpoint:** `/format_change/`
- **Method:** POST
- **Description:** Provides the capability to convert images from one file format to another.
- **Data Params:**
  - `function: "format_change"`
  - `format_change`: The desired image format (e.g., "jpg", "gif", "bmp").

Crop Service

- **Endpoint:** `/crop/`
- **Method:** POST
- **Description:** Offers tools to alter the dimensions and scale of an image.
- **Data Params:**
  - `function: "crop"`
  - `crop_params`: Parameters defining the crop area.

Noise Removal Service

- **Endpoint:** `/noise_removal/`
- **Method:** POST
- **Description:** Reduces visual noise from images.
- **Data Params:**
  - `function: "noise_removal"`

Background Change Service

- **Endpoint:** `/background_change/`
- **Method:** POST
- **Description:** Enables users to alter or remove the background of an image.

- **Data Params:**
  - `function: "background_change"`
  - `background_change:` The desired background color or setting.

Brightness and Contrast Service

- **Endpoint:** `/brightness_contrast/`
- **Method:** POST
- **Description:** Adjusts the brightness and contrast of images.
- **Data Params:**
  - `function: "brightness_contrast"`
  - `brightness:` The brightness value.
  - `contrast:` The contrast value.

Resize Service

- **Endpoint:** `/resize/`
- **Method:** POST
- **Description:** Changes the size of the image.
- **Data Params:**
  - `function: "resize"`
  - `resize:` The new dimensions of the image (e.g., "180,250").

Example Python Function to Interact with API

```
import requests

def test_passport_photo_size(country):
    url = f"{get_service_url(passport_photo_size')}/passport_photo_size/"
    response = requests.post(url, files={myfile': open(path_to_image', rb')},
                             data={function': 'passport_photo_size',
                                   'country': country, 'background_req': 'yes'})
    return response.json()

# Example usage:
result = test_passport_photo_size(France')
print(result)
```

Replace path_to_image' with the actual file path to the image you want to process, and France' with the country for which you want to format the passport photo.

Notes

- Ensure that the file paths used in the functions are updated to the location where the image/video files are stored on your system.
- The provided assertion statements are for testing purposes and should be adjusted according to the actual response structure of the API.
- When using the test functions, handle exceptions and errors gracefully to ensure a robust application.

6 Handbook

The displayed image represents the initial landing page of our application, visible upon entering the website URL in any web browser (see Fig. 6.1).

Figure 6.1: Landing Page (Fig. 6.1)

A few pointers on what has changed on the landing page (see Fig. 6.2):

- The user can now find all the services of the Photo Editor tool under the dropdown menu 'Features'.
- A privacy notice has been added to ensure that the user knows that their data is not stored by us.
- The display of all features on the landing page has been given a more compact card arrangement (see Fig. 6.3).

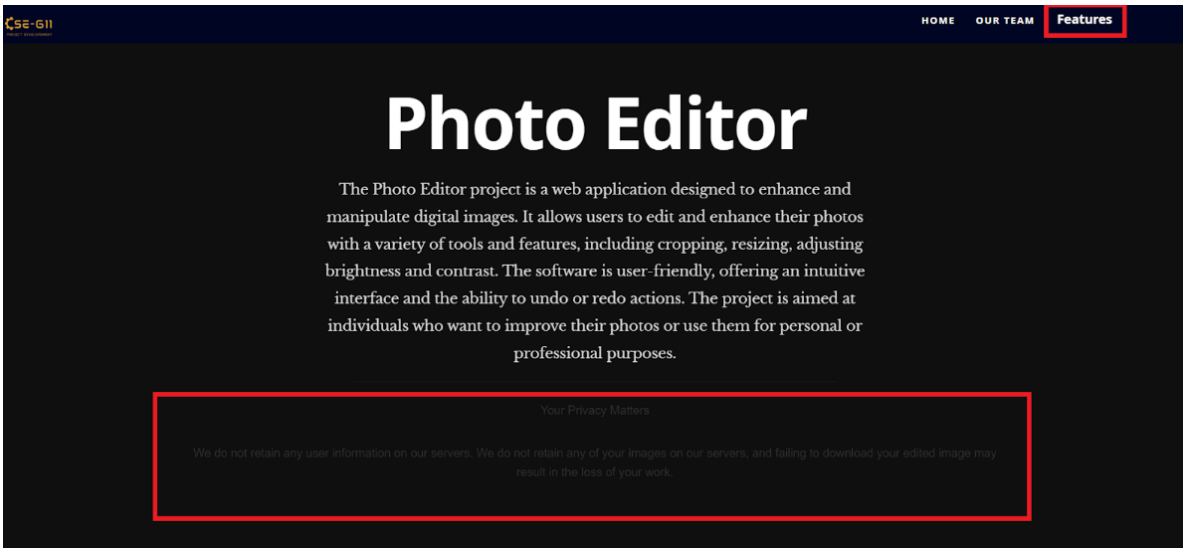


Figure 6.2: Updated Landing Page

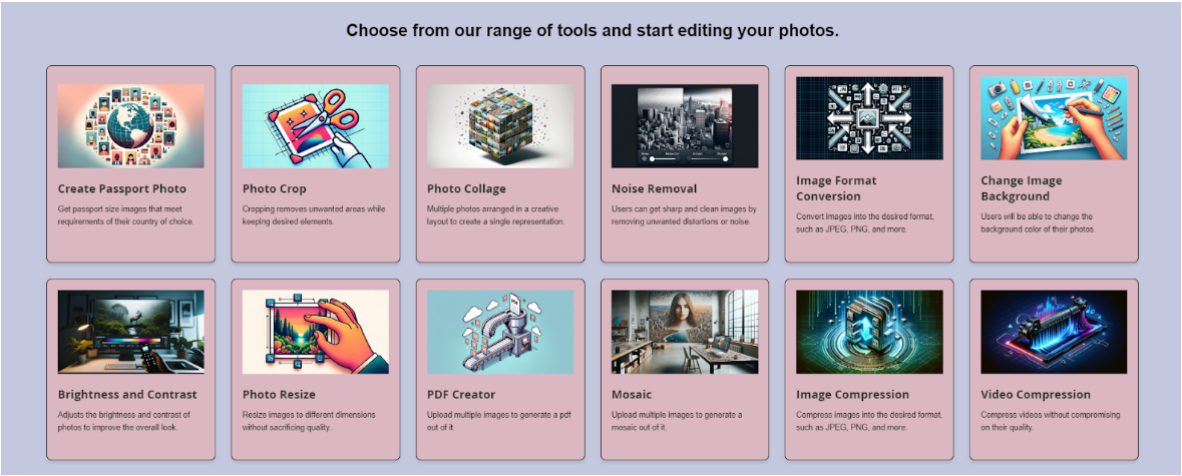


Figure 6.3: Compact Card Arrangement

We will now explore how to operate and use each of these features.

6.0.1 Create Passport Photo

1. **A. Input Image:** This section allows users to upload the image they wish to modify for the passport photo. The user can browse their device or use the drag and drop feature (see Fig. 6.4).
2. **B. Choose Country:** The user can choose the specific country for which they are creating the passport photo by using the dropdown menu or they can search by country name (see Fig. 6.4). Each country has its own set of specifications and requirements for passport photos. By clicking on the button 'Visit Website' that appears after selecting a country, the user will be redirected to the official website containing the passport photo requirements of the chosen country.
3. **C. Apply White Background:** This checkbox enables the user to apply a white background and add a black border to the passport photo (see Fig. 6.5).
4. **D. Preview Photo:** Click this button to apply changes and preview the passport photo before downloading. When all changes are applied, the user will see a pop-up that says 'Successfully processed' (see Fig. 6.5).
5. **E. Download Photo:** Click this button to download the image onto your device (see Fig. 6.5).
6. **F. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.5).

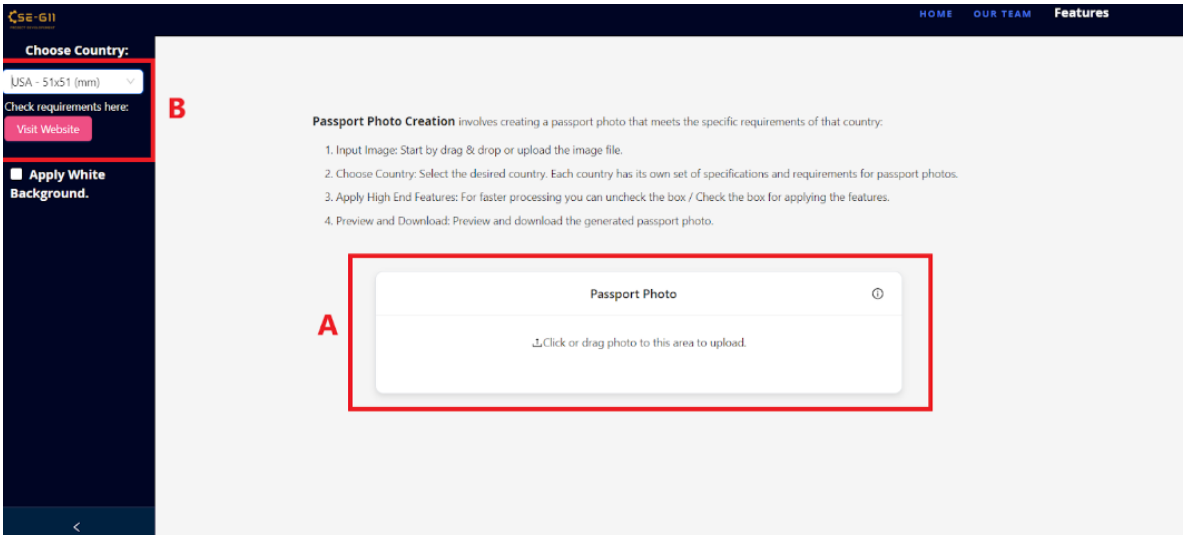


Figure 6.4: Passport Photo Feature

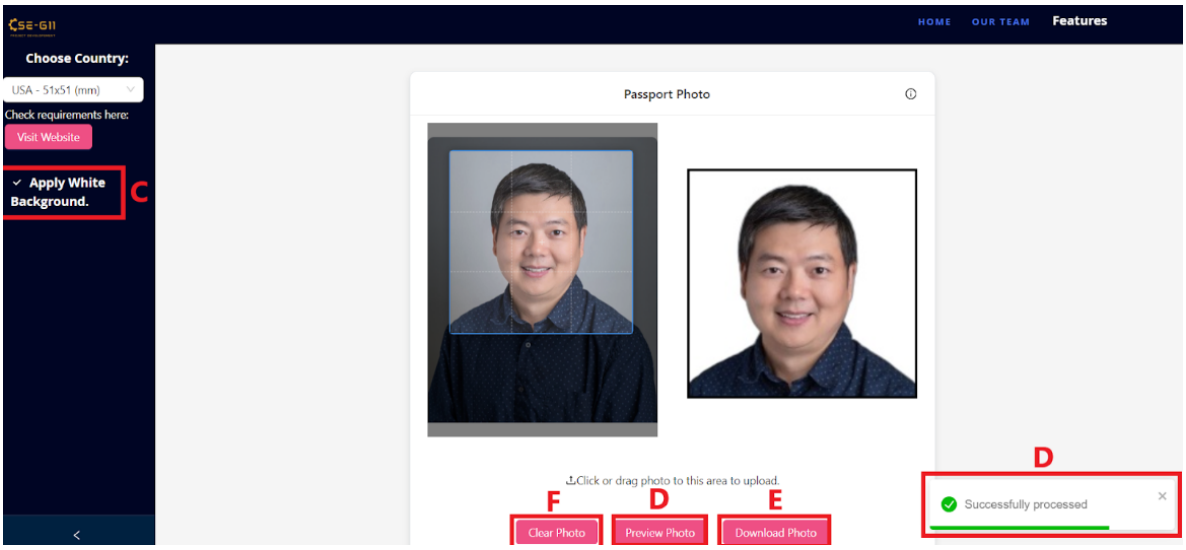


Figure 6.5: Additional Features

6.0.2 Photo Crop

1. **A. Input Image:** This section allows users to upload the image they wish to modify. The user can browse their device or use the drag and drop feature (see Fig. 6.6).
2. **B. Choose Ratio & Custom Ratio:** In the left panel, choose the desired aspect ratio to customize the photo according to your preferences or input specific values to set a custom aspect ratio tailored to your needs (see Fig. 6.6).
3. **C. Switch to Circle Crop:** Click this button when you want to shift to a circle crop (see Fig. 6.6).
4. **D. Switch to Normal Crop:** Click this button when you want to shift to a rectangular crop (see Fig. 6.7).
5. **E. Download Photo:** Click this button to download the image onto your device (see Fig. 6.7).
6. **F. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.7).

When the image is uploaded, the user can enter a size on the left panel or choose one of the options from the custom ratio, or manually adjust the size using their cursor.

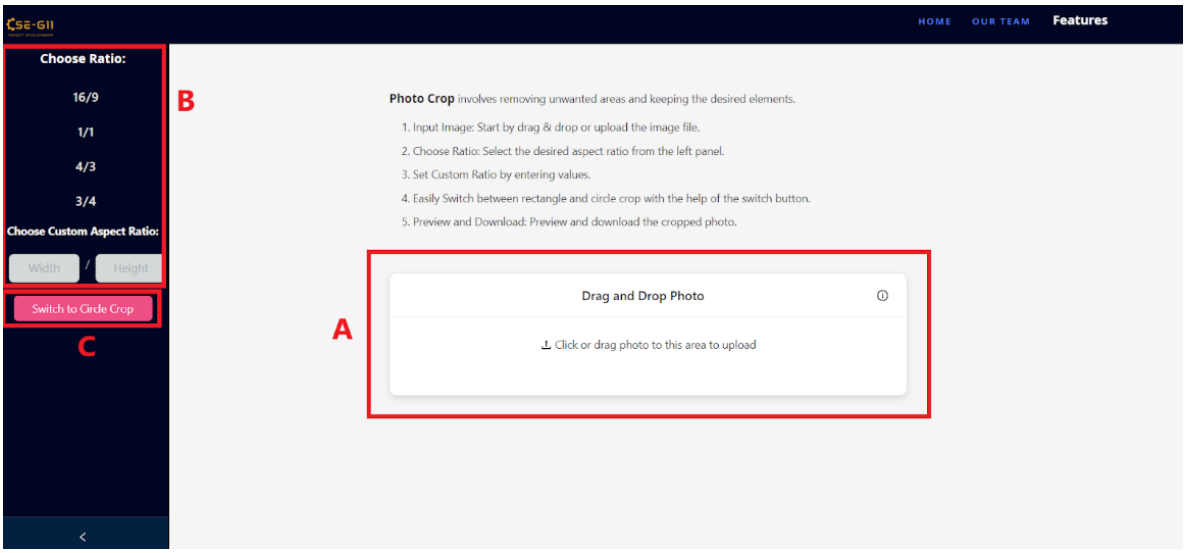


Figure 6.6: Photo Crop and Aspect Ratio Features

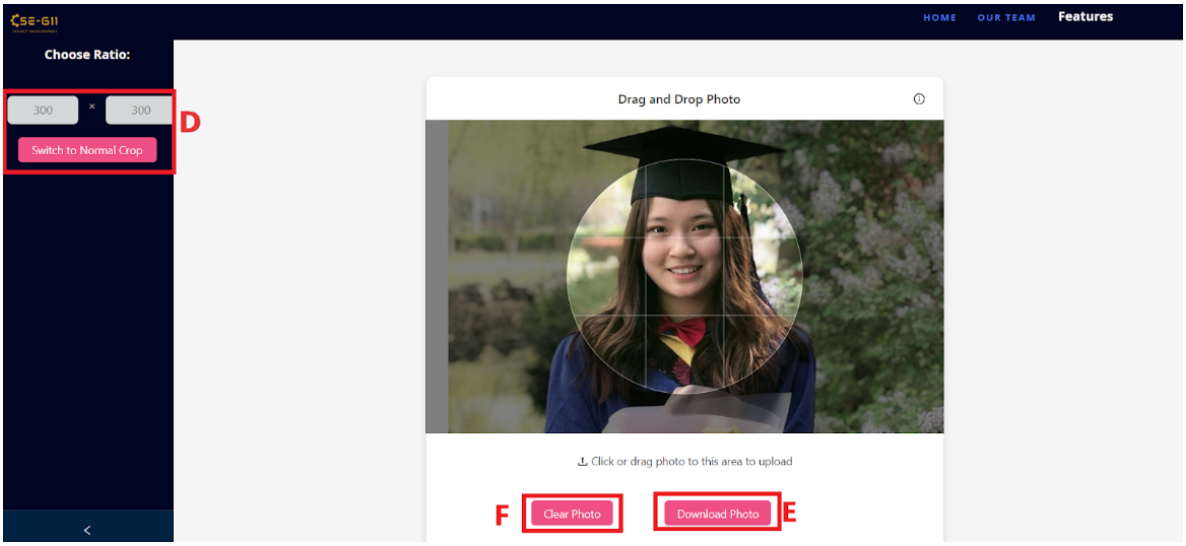


Figure 6.7: Crop and Download Options

6.0.3 Collage Maker

1. **A. Choose Template:** In the left panel, choose the desired template for your photo collage (see Fig. 6.8).

2. **B. Split:** If you wish to split the chosen template further, click on the 'Split' button on the top right corner of the individual template section (see Fig. 6.8).

3. **C. Input Image:** After finalizing a template, you can upload the images you want by clicking on each individual section of the template (see Fig. 6.8).

4. **D. Zoom in & Zoom out:** Click the '+' or '-' button on the top left corner of the individual template section to zoom in or zoom out, respectively (see Fig. 6.8).

5. **E. Download Collage:** Click this button to download the photo collage onto your device (see Fig. 6.8).
6. **F. Remove:** Click this button to remove the uploaded photo (see Fig. 6.9).

For ease of operation, steps are added on the right panel.

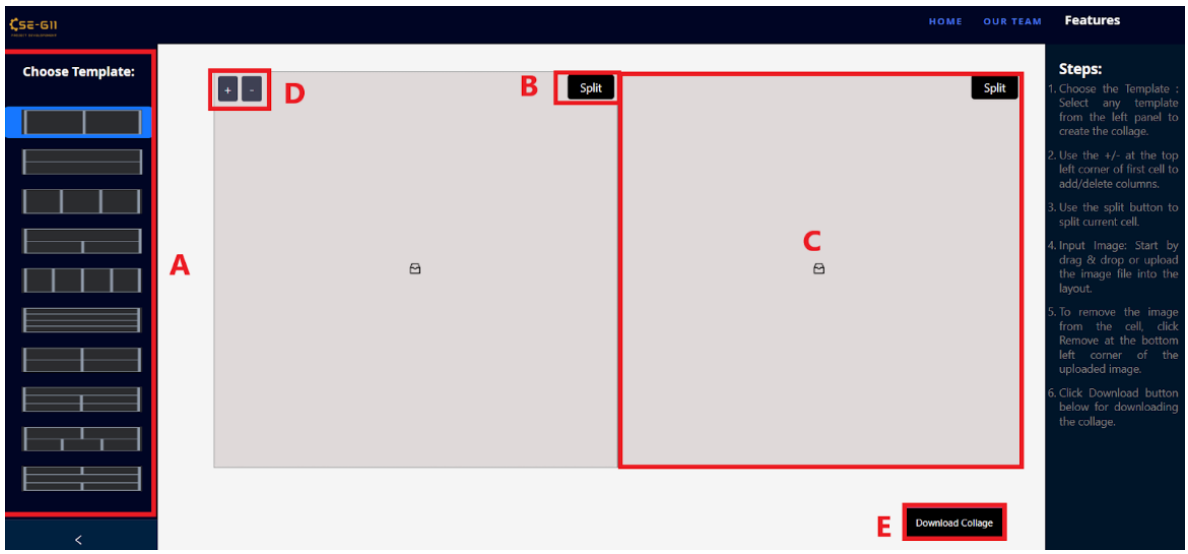


Figure 6.8: Collage Maker Interface and Features (Fig. 6.8)

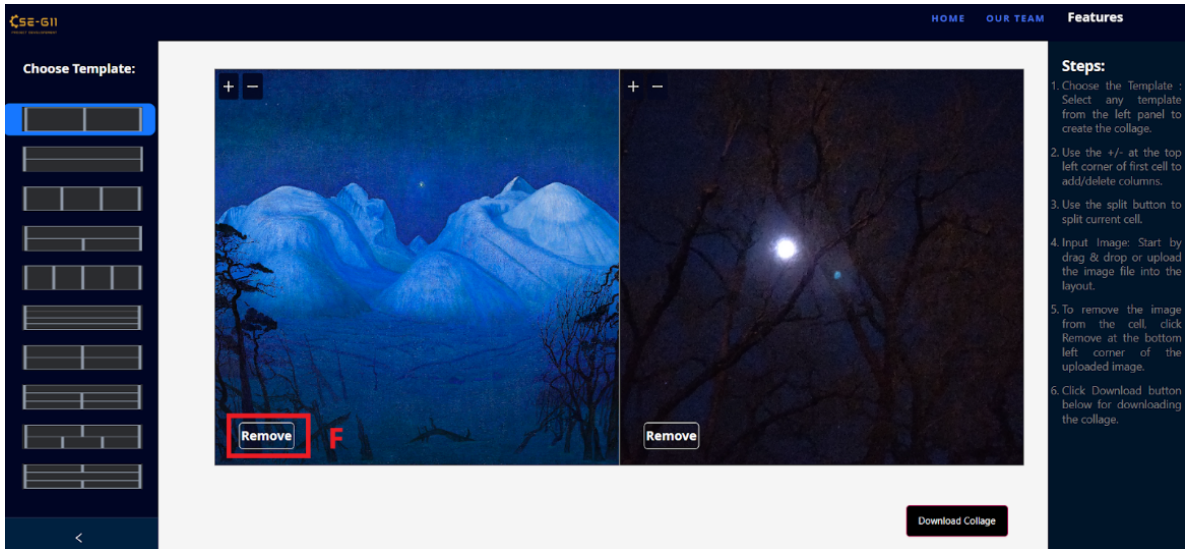


Figure 6.9: Remove Image Feature in Collage Maker (Fig. 6.9)

6.0.4 Noise Removal

1. **A. Input Image:** This section allows users to upload the image they wish to modify (see Fig. 6.10).
2. **B. Preview Photo:** Click this button to apply the noise removal changes and preview the modified image to the right of the original image (see Fig. 6.11).
3. **C. Download Photo:** Click this button to download the image onto your device (see Fig. 6.11).
4. **D. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.11).

Once the user has uploaded an image, the 'Preview' button and the 'Clear' button become visible. The user needs to click on the 'Preview' button to apply the noise removal changes.

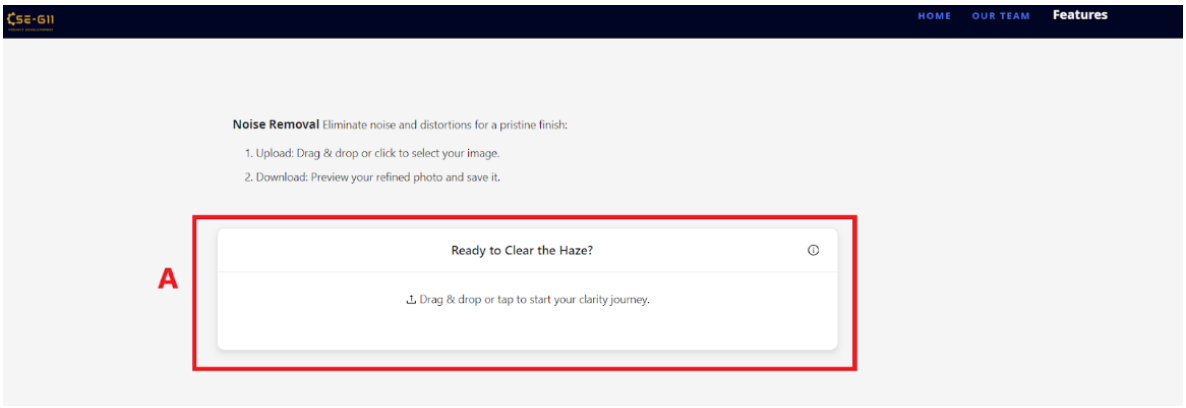


Figure 6.10: Input Image for Noise Removal (Fig. 6.10)

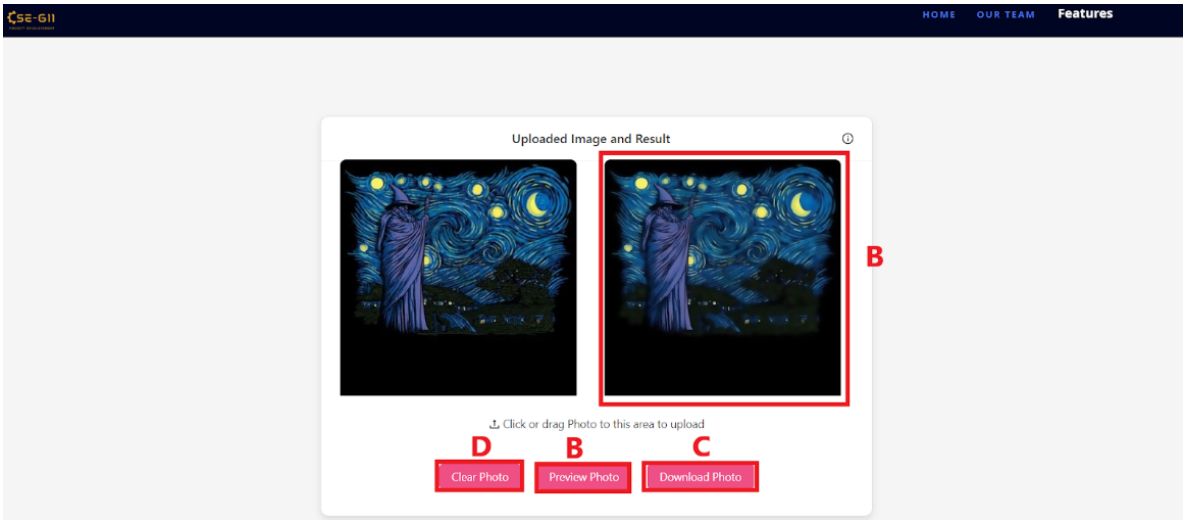


Figure 6.11: Preview and Download Options for Noise Removal (Fig. 6.11)

6.0.5 Format Conversion

1. **A. Input Image:** This section allows users to upload the image they wish to modify (see Fig. 6.12).
2. **B. Convert To:** In the left panel, choose the desired output format of the image (see Fig. 6.12).
3. **C. Download Photo:** After uploading an image and selecting the output format, click this button to convert the uploaded image format to the chosen format and download the image (see Fig. 6.13).
4. **D. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.13).

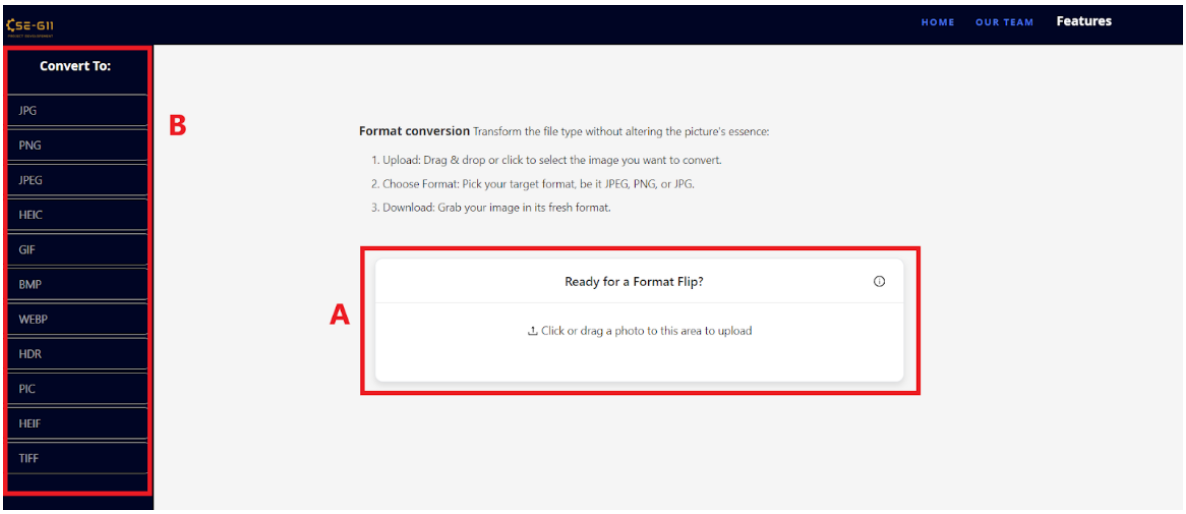


Figure 6.12: Input and Format Conversion Options (Fig. 6.12)

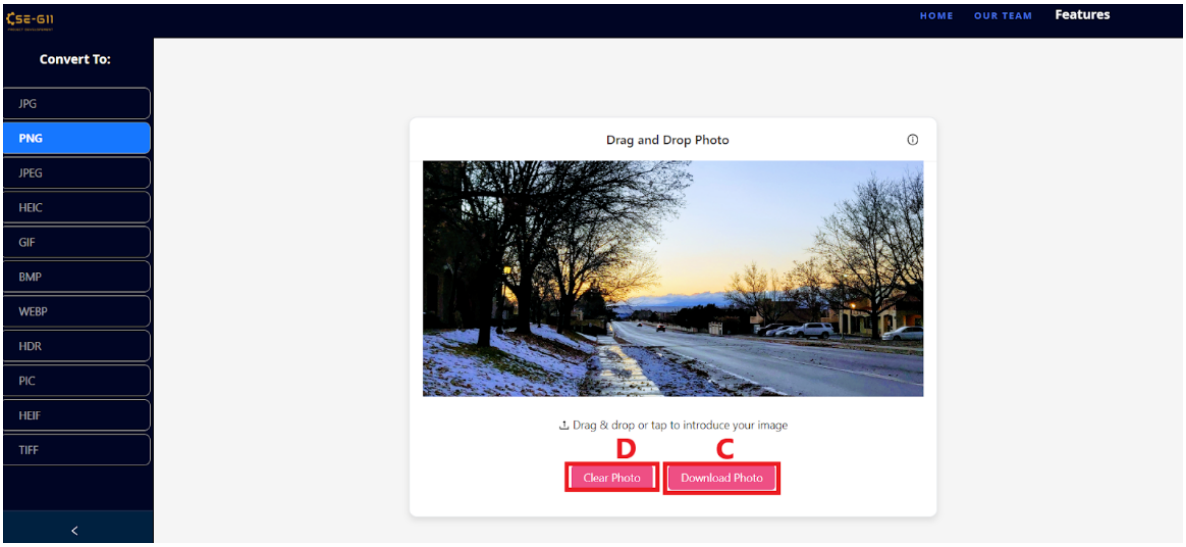


Figure 6.13: Conversion and Download Options (Fig. 6.13)

6.0.6 Background Change

1. **A. Input Image:** This section allows users to upload the image they wish to modify (see Fig. 6.14).
2. **B. Choose Background Color:** In the left panel, choose the desired background color you want to apply to your image. Users can select a color by dragging the cursor across the color spectrum or by entering the HEX number of any color. The spectrum can be adjusted with the sliding knob, and users can also choose the desired gradient for any selected color (see Fig. 6.14).
3. **C. Preview Photo:** Click this button to apply the background color changes and preview the modified image next to the original image (see Fig. 6.15).
4. **D. Download Photo:** Click this button to download the image onto your device (see Fig. 6.15).
5. **E. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.15).

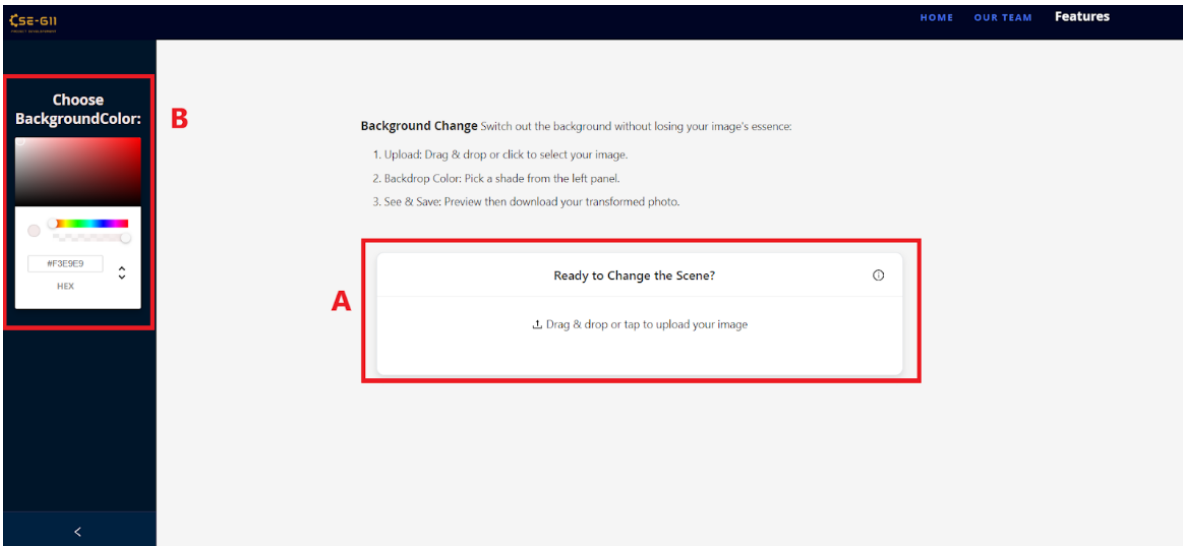


Figure 6.14: Input Image and Background Color Options (Fig. 6.14)

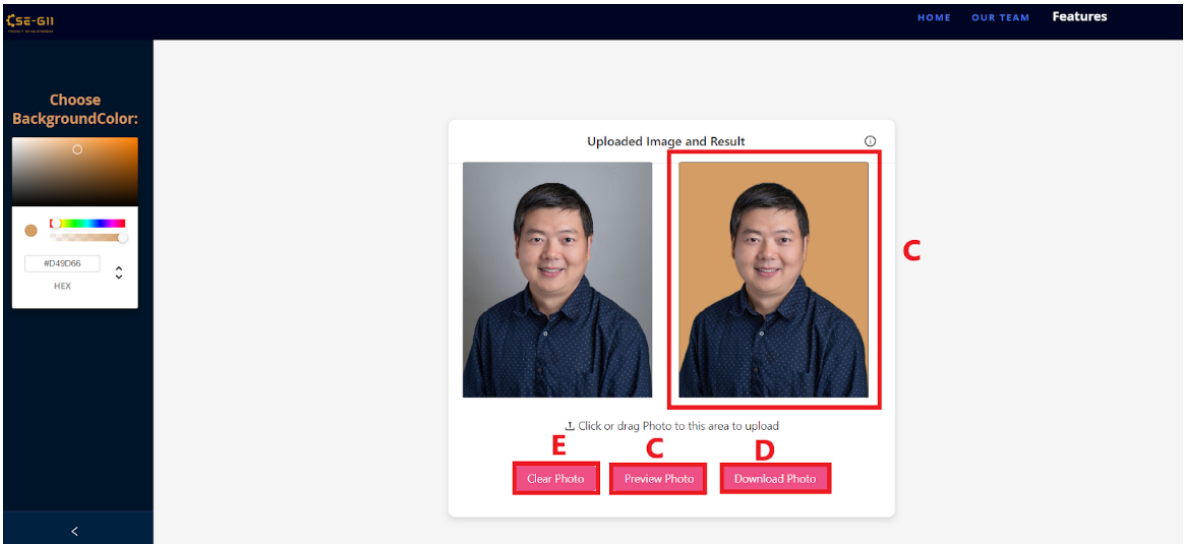


Figure 6.15: Preview and Download Options After Background Change (Fig. 6.15)

6.0.7 Brightness and Contrast

1. **A. Input Image:** This section allows users to upload the image they wish to modify (see Fig. 6.16).
2. **B. Brightness:** In the left panel, adjust the brightness of the image using the sliding knob (see Fig. 6.16).
3. **C. Contrast:** In the left panel, adjust the contrast of the image using the sliding knob (see Fig. 6.16).
4. **D. Download Photo:** Click this button to download the image onto your device (see Fig. 6.17).
5. **E. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.17).

After the user uploads an image, two images will be visible. On the right, the original image, and on the left, the modified image with the changes applied. The image on the right is available for download.

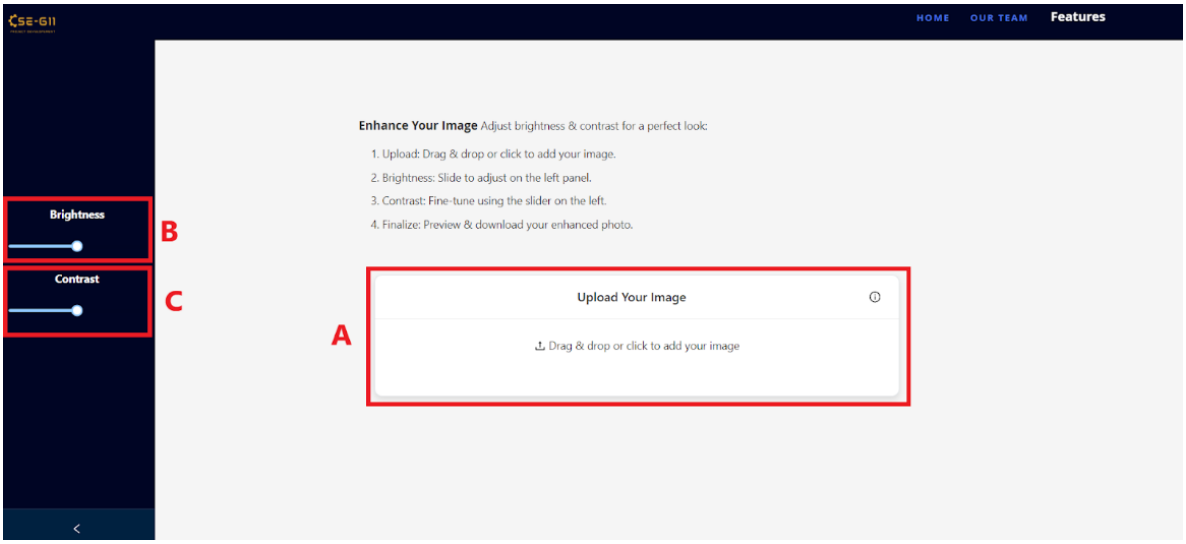


Figure 6.16: Adjusting Brightness and Contrast (Fig. 6.16)

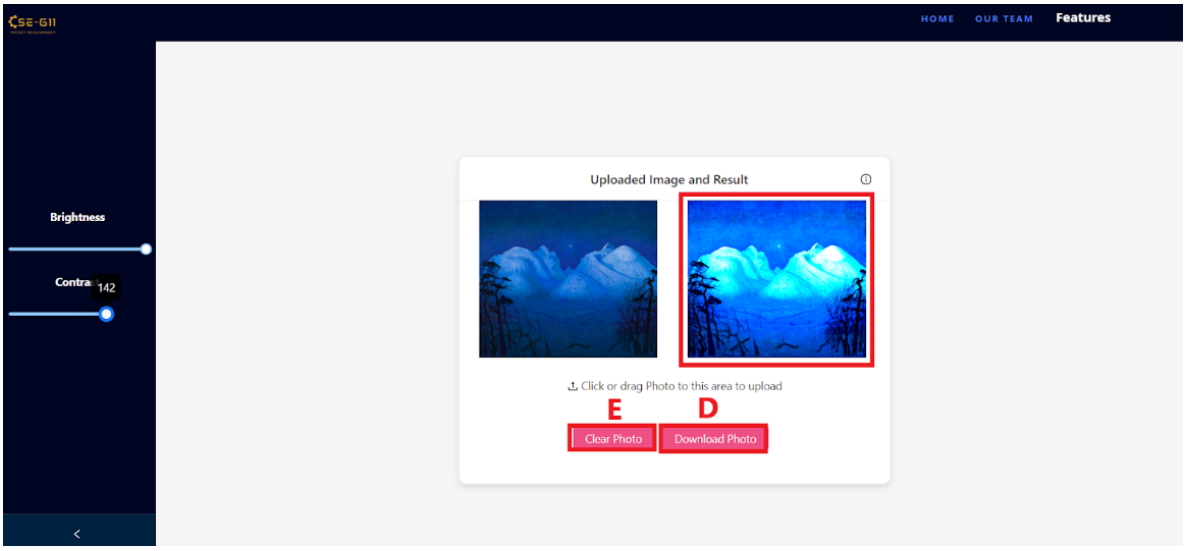


Figure 6.17: Download and Clear Options After Adjustments (Fig. 6.17)

6.0.8 Resize

1. **A. Input Image:** This section allows users to upload the image they wish to modify (see Fig. 6.18).
2. **B. Width:** In the left panel, adjust the width of the image (in pixels) using the sliding knob (see Fig. 6.18).
3. **C. Height:** In the left panel, adjust the height of the image (in pixels) using the sliding knob (see Fig. 6.18).
4. **D. Preview Photo:** Click this button to apply the width and height changes and preview the modified image next to the original image (see Fig. 6.19).
5. **E. Download Photo:** Click this button to download the image onto your device (see Fig. 6.19).
6. **F. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.19).

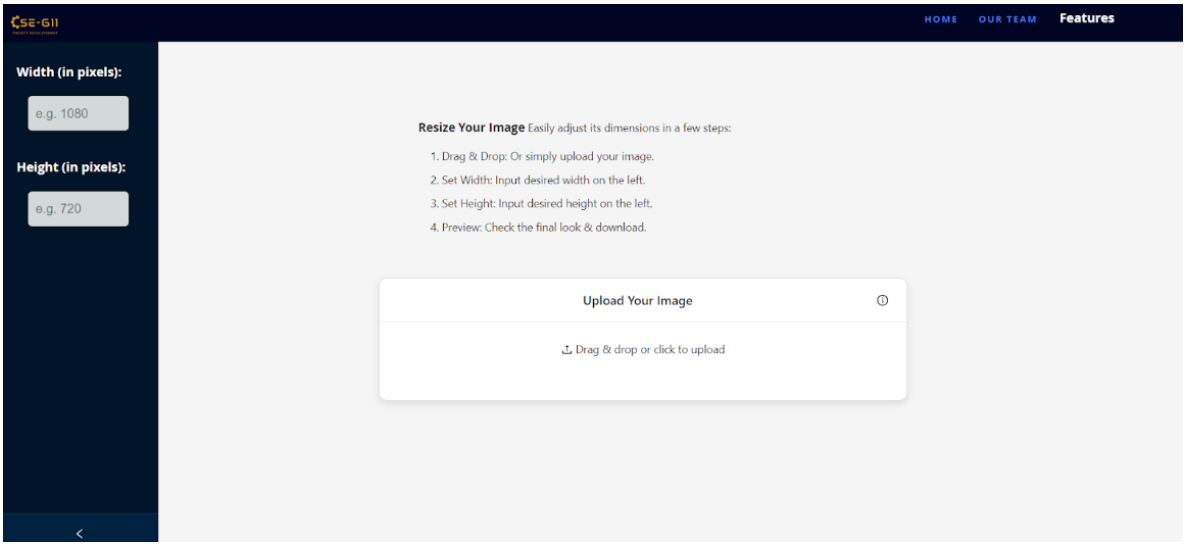


Figure 6.18: Resize Feature: Adjusting Width and Height (Fig. 6.18)

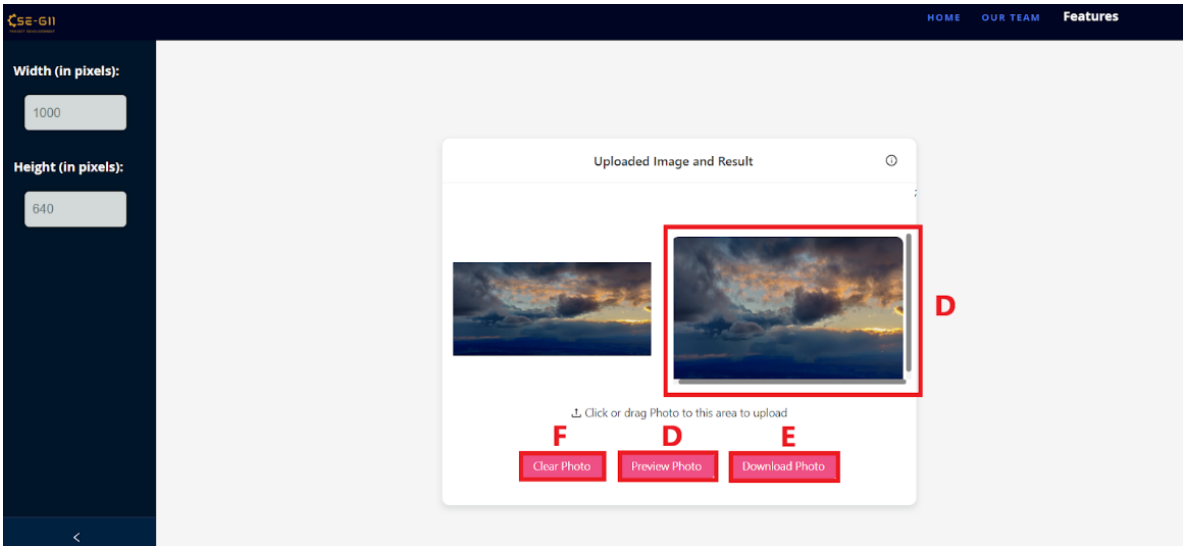


Figure 6.19: Preview and Download Options After Resize (Fig. 6.19)

6.0.9 Mosaic

1. **A. Choose Template:** In the left panel, upload an image by clicking on the '+' button, this image will be used as a template for the Mosaic (see Fig. 6.20).
2. **B. Choose Pixel Size:** The user can increase or decrease the pixel size using this sliding knob (see Fig. 6.20).
3. **C. Input Images:** In the right panel, upload the images by clicking on the '+' button, these images will be used to form the mosaic. A minimum of six images are required to make the mosaic (see Fig. 6.20).
4. **D. Clear All:** By clicking this button all the images uploaded by the user on the right panel will be removed (see Fig. 6.20).
5. **E. Clear:** Click this button to remove the template from the left panel (see Fig. 6.21).
6. **F. Remove:** Click this button to eliminate individual images from the left panel (see Fig. 6.21).
7. **G. Preview Photo:** Click this button to generate the Mosaic with the template and images of your choice (see Fig. 6.21).
8. **H. Download Photo:** Click this button to download the Mosaic onto your device (see Fig. 6.21).

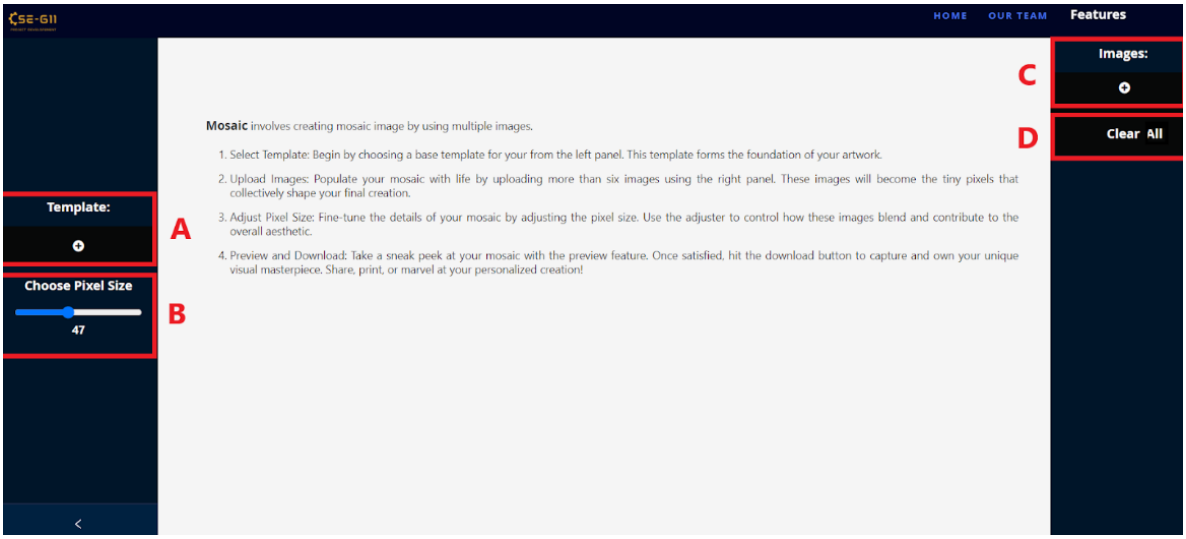


Figure 6.20: Mosaic Feature: Template and Pixel Size Selection (Fig. 6.20)

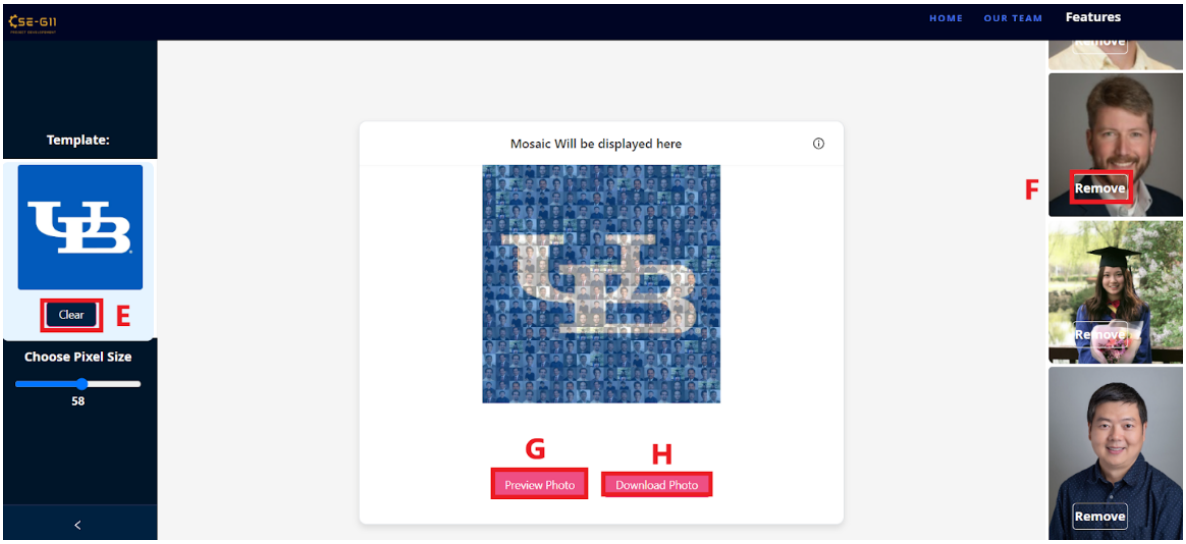


Figure 6.21: Mosaic Options: Clear, Remove, Preview, and Download (Fig. 6.21)

6.0.10 PDF Creator

1. **A. Input Images:** In the right panel, upload the images by clicking on the '+' button, these images will be used to create the PDF (see Fig. 6.22).
2. **B. Clear:** By clicking this button all the images uploaded by the user will be removed (see Fig. 6.22).
3. **C. Create PDF:** After uploading all the desired images, click on this button to create a PDF (see Fig. 6.22).
4. **D. Remove:** Click this button to eliminate individual images from the left panel (see Fig. 6.23).
5. **E. Download, Print and More actions:** Following generation, the PDF will be presented for user review. Subsequently, users can initiate various actions such as downloading, printing, or exploring additional functionalities by clicking on the corresponding buttons situated in the top-right corner of the PDF (see Fig. 6.23).

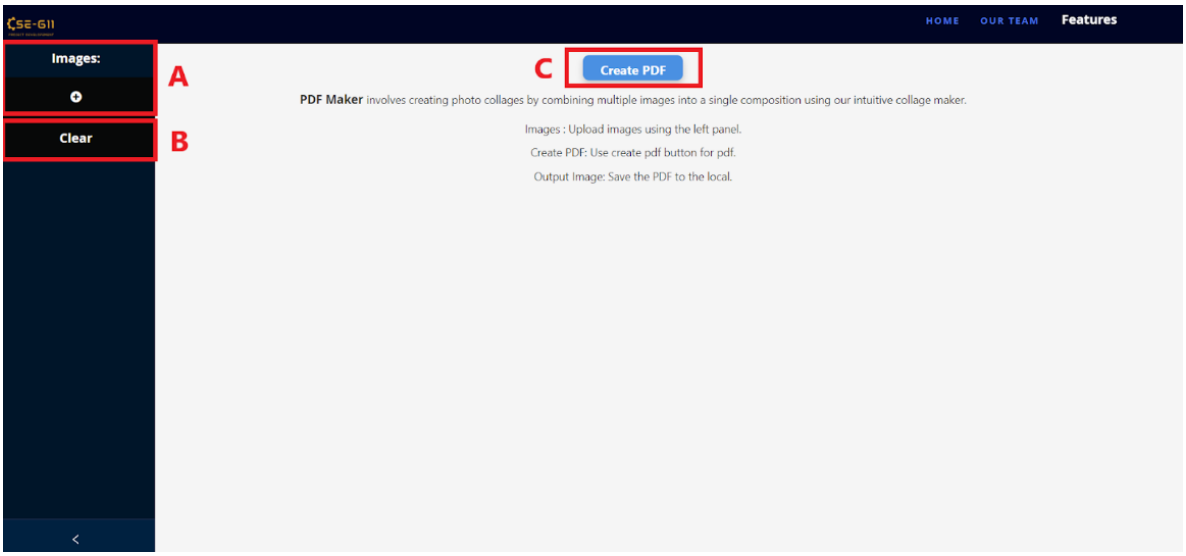


Figure 6.22: PDF Creator: Uploading Images and PDF Creation (Fig. 6.22)

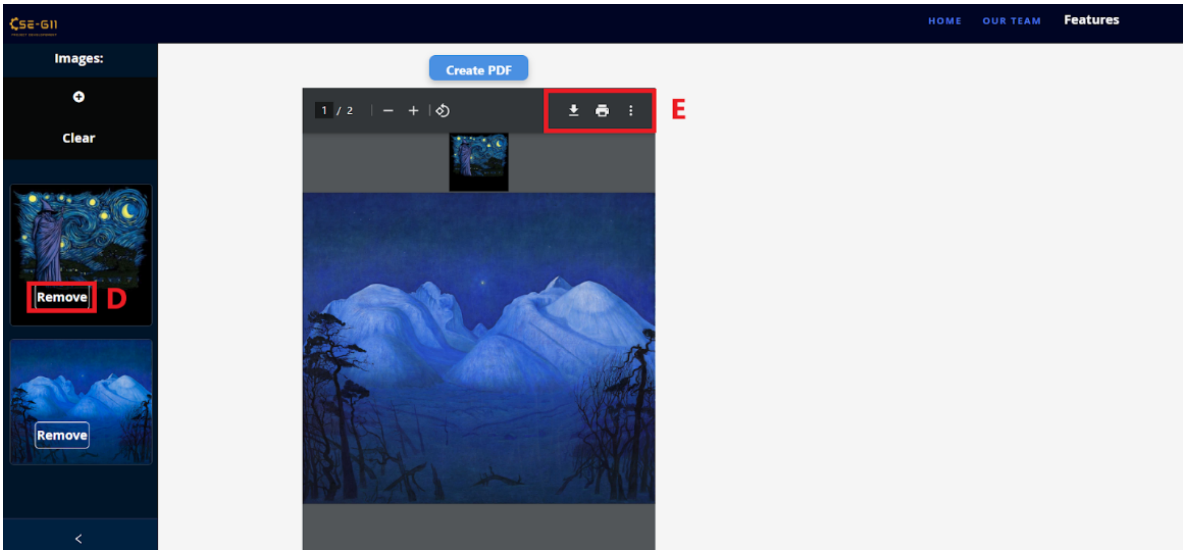


Figure 6.23: PDF Options: Remove, Download, Print, and More (Fig. 6.23)

6.0.11 Image Compression

1. **A. Input Images:** This section allows users to upload the image they wish to modify (see Fig. 6.24).
2. **B. Compression Rate:** Users can manipulate the compression rate using the sliding knob, impacting the image's quality. The set number represents a rough percentage of the original image size, determining the extent of compression (see Fig. 6.25).
3. **C. Custom Compression:** Users can fine-tune the image size by specifying it in the input field. The compressed image may not precisely match the user-entered size but will be compressed to fit within it (see Fig. 6.24).
4. **D. Compress As:** This aspect has three options that specify the output format of the compressed image. Users can choose their required format (see Fig. 6.24).
5. **E. Preview Photo:** Click the button to apply the image compression adjustments. The final image will be displayed for review once the compression is complete (see Fig. 6.25).
6. **F. Download Photo:** Click this button to download the compressed image onto your device (see Fig. 6.25).
7. **G. Clear Photo:** Click this button to remove the uploaded image (see Fig. 6.25).
8. **H. Input & Output file size:** Upon displaying the compressed image, the sizes of both the original and compressed images will be presented underneath for the user's reference (see Fig. 6.25).

Note: When users opt for PNG' as the preferred output format, it's crucial to ensure that the input image is also in PNG' format due to the differences in compression algorithms between JPEG and PNG. A pop-up will prompt the user to upload an image in PNG' format if the condition is not met.

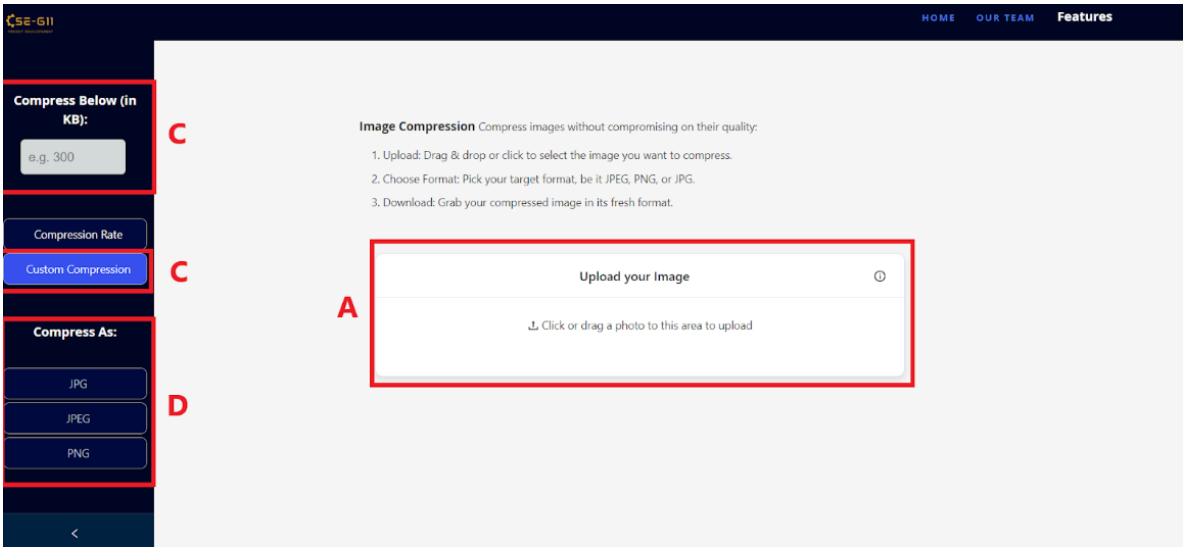


Figure 6.24: Image Compression: Input and Settings (Fig. 6.24)

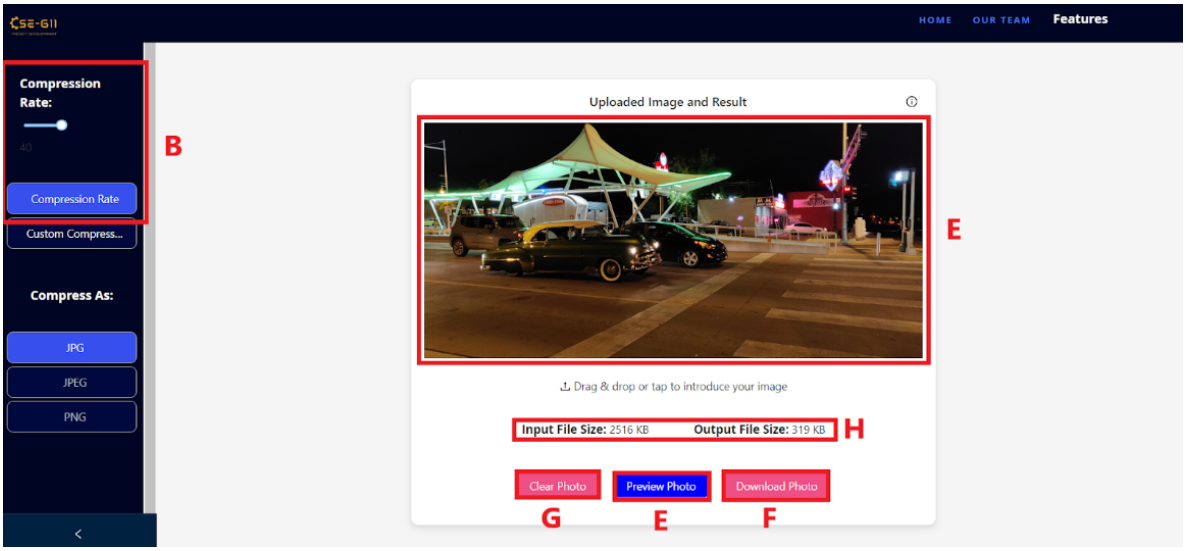


Figure 6.25: Image Compression: Preview, Download, and Size Information (Fig. 6.25)

6.0.12 Video Compression

1. **A. Input Video:** This section allows users to upload the video they wish to modify (see Fig. 6.26).
2. **B. Custom Compression:** Users can fine-tune the video size by specifying it in the input field. The compressed video will approximately match the specified size (see Fig. 6.26).
3. **C. Video Resolution:** Under this aspect, users have two options (see Fig. 6.27):
   (a) *Resolution Quality:* Users can fine-tune the video resolution quality by specifying it in the input field.
   (b) *Video Resolution:* After choosing the desired resolution quality, users are prompted to select the specific resolution for the video they wish to obtain.
4. **D. Compress :** Click this button to initiate the video compression.

5. **Download Video**: Download it onto your device. Once the compression is complete, the final video will be displayed for your review. (see Fig. 6.27).
6. **E. Clear Video:** Click this button to remove the uploaded video (see Fig. 6.27).
7. **F. Input & Output file size:** Upon displaying the compressed video, the sizes of both the original and compressed video will be presented underneath for the user's reference (see Fig. 6.27).
8. **Clear Output Video**:Click this button to clear your Output Video.

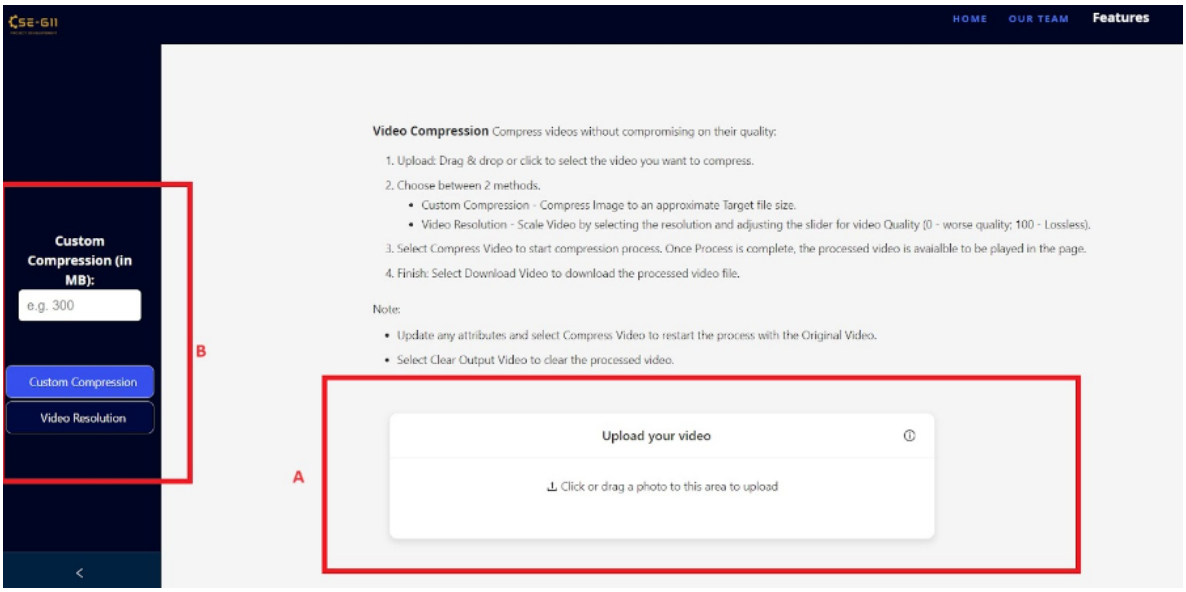


Figure 6.26: Video Compression: Input and Custom Compression (Fig. 6.26)

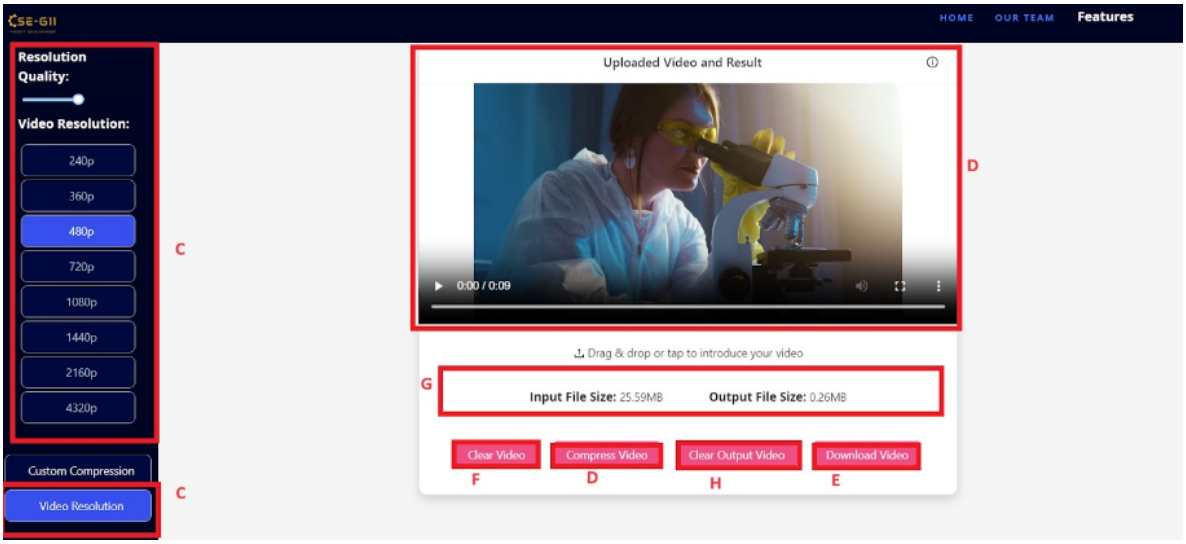


Figure 6.27: Video Compression: Resolution Settings and Download (Fig. 6.27)

7 Future Work

In this section, we will explore potential future enhancements for the Photo Editor, including the following:

- **Integrate support for additional countries in the Create Passport Photo' feature:** This enhancement aims to broaden the application's global usability by accommodating the passport photo requirements of more countries.
- **Incorporate more output formats in the Image Compression' feature:** Expanding the range of output formats, such as TIFF and WEBP, will offer users a wider array of choices for their compressed images, catering to diverse needs and preferences.
- **Ensure website adaptability across various devices:** Enhancing the user experience across different platforms and screen sizes is crucial. This includes optimizing the website for seamless functionality on mobile devices, tablets, and desktops, ensuring a consistent and user-friendly experience regardless of the device used.

These proposed enhancements are aligned with the goal of continuously improving the Photo Editor to meet evolving user needs and technological advancements, ensuring that it remains a valuable and versatile tool for a wide range of users.

8 Contributions

Team Member	Contribution
Saad Ahmed	**Home Page:** • Completely redesigned the website home page. • Implemented grid card layout where each card represents a specific feature. • Introduced interactive functionality enabling users to seamlessly navigate to specific feature pages with a simple click on the corresponding card palette. **Passport Feature:** • Enhanced the passport photo creation feature by implementing dynamic snipping on the passport photo page, tailoring it to meet specific country requirements. • Expanded support for a variety of countries, ensuring compatibility with diverse passport photo standards. • Introduced a dynamic button feature that seamlessly directs users to the official websites of respective countries for detailed passport photo requirements. • Modified Input field to display the size respective to each country. **Crop Feature:** • Revamped the user interface of the Crop Feature, providing a more modern and intuitive design for enhanced user experience. • Introduced a novel circle crop feature, expanding the range of cropping options available to users. • Implemented a user-friendly functionality that allows seamless switching between rectangle and circle crop modes with a simple click of a button. **Collage Feature:** • Updated the entire user interface of the Collage Feature page, ensuring a visually appealing and cohesive design that enhances the overall aesthetic quality. • Introduced capability to modify collages by seamlessly adding or deleting columns • Implemented advanced functionality that allows users to split or merge current cells within the collage. • Introduced new layouts and Streamlined the user experience by introducing drag-and-drop features for effortless rearrangement of elements within the collage, enhancing overall usability • Incorporated intuitive controls to adjust zoom within the collage, providing users with fine-tuned control over the visual composition. **Noise Removal and Brightness and Resize Feature:** • Adjusted the alignment of preview images **Mosaic:** • Redesigned the user interface of the Mosaic Feature page. • Developed the backend infrastructure for the Mosaic Feature. • Developed a user-friendly feature that allows users to chose the pixel size of images,offering precise control over the level of detail in their mosaic compositions. **Statistics:** • Implemented the logic to compute and display statistics highlighting the most frequently used features. • Implemented a dynamic bar graph presentation to visually convey the usage patterns of each feature. **Deployment:** • Developed Docker files for each individual features • Established GitHub workflows to automate the process of building and pushing images to xlab-cluster • Developed deployment and service objects tailored to the xlab cluster, optimizing resource utilization and ensuring robust scalability of the deployed images. • Implemented imagepullsecret to pull the images from ghcr. • Documented entire deployment processes and configurations.

Team Member	Contribution
Ayesha Humaera	**Passport Photo::** • Researched official photo requirements for different countries to integrate into the 'Create Passport Photo' feature. **Brightness and Contrast:** • Fixed bugs in UI and made changes to landing page descriptions. **Background Change:** • Made UI changes to make the UI more appealing. • Changed default color to enable better visuals. **Resize:** • Added new description and updated existing notes to better match the changes made. **Mosaic:** • Implemented a universal 'Delete' button across all pages. Introduced 'Clear All' to remove all user-uploaded images and 'Clear' for individual image deletion. • Implemented a prompt with a confirmation dialog to prevent accidental deletions when the user selects 'Clear All' option. • Fixed UI bugs in Mosaic. **Image Compression:** • Conducted research on image compression algorithms to identify the most effective solution for various image types. • Executed a proof of concept (POC) to validate the functionality of the chosen image compression algorithm. • COmpletely developed the frontend page for the Image Compression feature, initially implementing 'Compression Rate' and later incorporating 'Custom Compression'. • Conducted further research to identify the optimal algorithm for implementing custom compression. • Implemented a universal 'Delete' button across all pages. Introduced 'Clear All' to remove all user-uploaded images and 'Clear' for individual image deletion. • Conducted research on image compression algorithms to identify the most effective solution for various image types. • Executed a proof of concept (POC) to validate the functionality of the chosen image compression algorithm. • Enhanced verification by adding more test cases to cover edge scenarios in Image Compression. Introduced user-friendly pop-ups for notifications. • Implemented an improved approach for Custom Image Compression, addressing and fixing bugs from the previous implementation. • Implemented a visual indicator (color change) for the 'Preview' button upon clicking. Resolved a malfunction where the button didn't revert to its original state after pressing 'Clear'. • Added functionality to choose between 'Compression Rate' and 'Custom Compression' in the Image Compression feature. • Integrated the display of input and output file sizes after generating a compressed image. • Implemented an additional check for PNG images, requiring users to provide a PNG input if the desired output is also PNG. This is crucial due to the inherent differences in compression algorithms between JPEG and PNG. **Miscellaneous:** • Created Documentation and PPTs. • Manual testing of individual features with different inputs (that are possible) to test for errors or bugs in code. • Code refactoring of frontend pages and backend logic. • Removal of extra libraries imported but not used in Python.

Team Member	Contribution
Pavana Lakhsmi Venugopal	**Bug Fixes:** • Updated file paths for all services to ensure the files are stored in uploads and output folder. • Passport Photo Functionality – Fixed 'Apply White Background' functionality as it was not working as expected. • Background Change – Reworked the functionality as it was not working due to incompatible library. • Format Change – Fixed non-working formats. • Corrected the Clear functionality within the Resize feature, ensuring that the image properly clears when the "Clear Photo" option is selected. **Enhancements:** • Format Change – Included 3 new formats. • Updated cron process to clear input and output files more often to avoid memory issues. • Reduced docker image sizes by 25 • Updated python requirements.txt to remove unused libraries. • Refactored code for clarity and performance. • Included message in frontend home page on User Information retention policy. **Video Compression:** • Created a new backend service for video compression functionality. • Created front-end logic for video compression functionality. • Researched on FFMPEG open-source multimedia framework and implemented compression functionality using this framework. • Implemented video compression with customizable target file size. • Added video scaling feature based on video resolution and video quality. • Added ability to view the uploaded video file and processed video file in the webpage. • Included file size comparison between uploaded video file and processed video file. • Added logic in frontend to select among the 2 features, and provide other necessary attributes based on that feature. • Enhanced workflow with dedicated buttons for clearing processed videos and downloading final files. **Microservices:** • Converted unified backend service to individual microservices. • Updated requirements.txt for every service by testing locally and including only required libraries. **Deployments:** • Created workflows for individual microservices. • Updated images to a few of the deployment configurations to pull from GHCR. **Miscellaneous:** • I have also done manual testing for all the flows to make sure features are working as expected. • Created Entire Demo Video. • Created PPT's and documention.