Sri Lanka Institute of Information Technology

KidniFy - A Mobile based Chronic Kidney Disease Patient Care System Using ML and IoT

2023-032

STATUS DOCUMENT - I

Student Name: Marasinghe M.M.K.L.

Student ID: IT20154226

Group Details

Supervisor: Ms. Wishalya Vanshanee Tissera

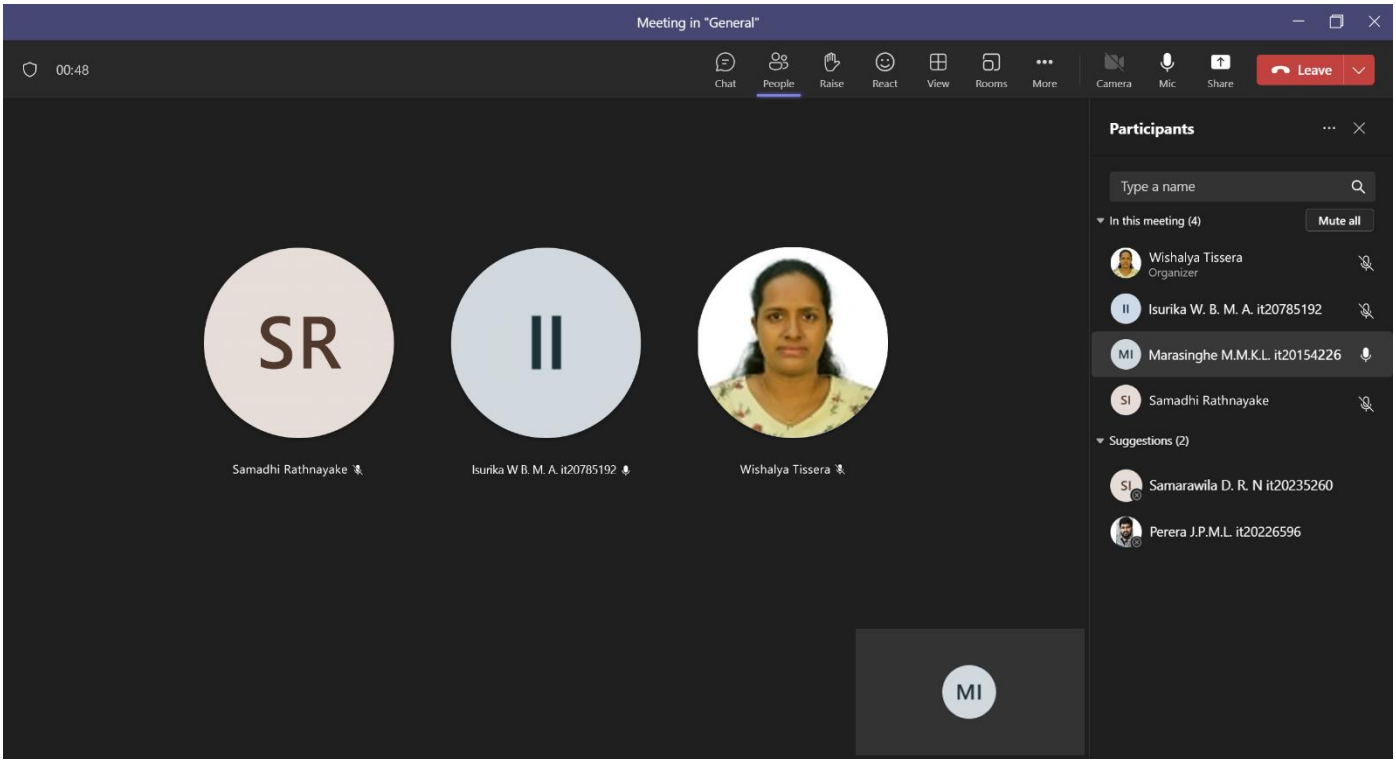
Co-Supervisor: Mr. Samadhi Chathuranga Rathnayake

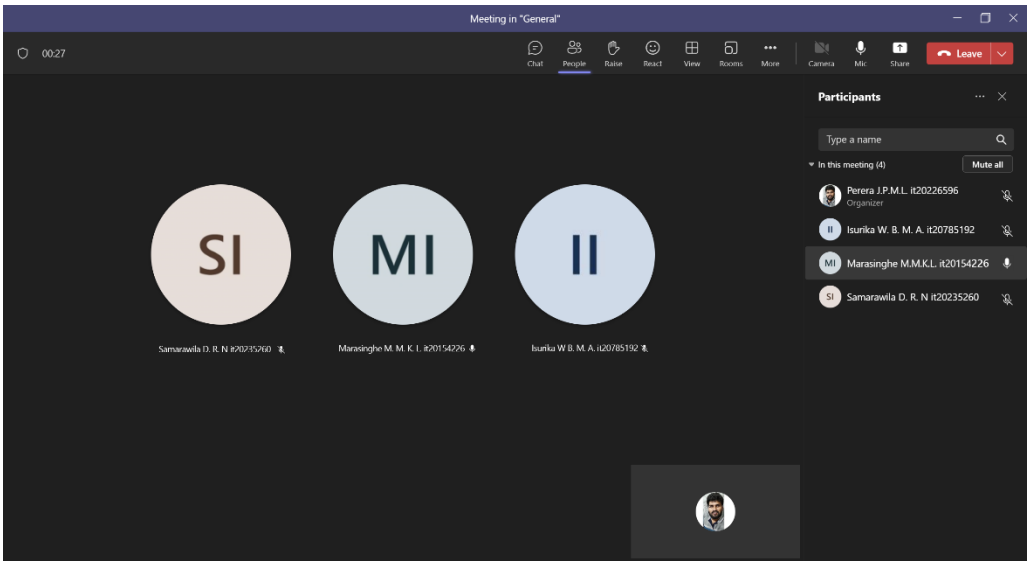
Student Name	Student ID	Contact No	Email Address
Marasinghe M.M.K.L.	IT20154226	0713037712	it20154226@my.sliit.lk
Isurika W.B.M.A.	IT20785192	0701484570	it20785192@my.sliit.lk
Perera J.P.M.L.	IT20226596	0776035479	it20226596@my.sliit.lk
Samarawila D.R.N.	IT20235260	0712421580	it20235260@my.sliit.lk

TABLE OF CONTENTS

1 Teams Meeting

1.1 Screenshots of Meetings & Calls





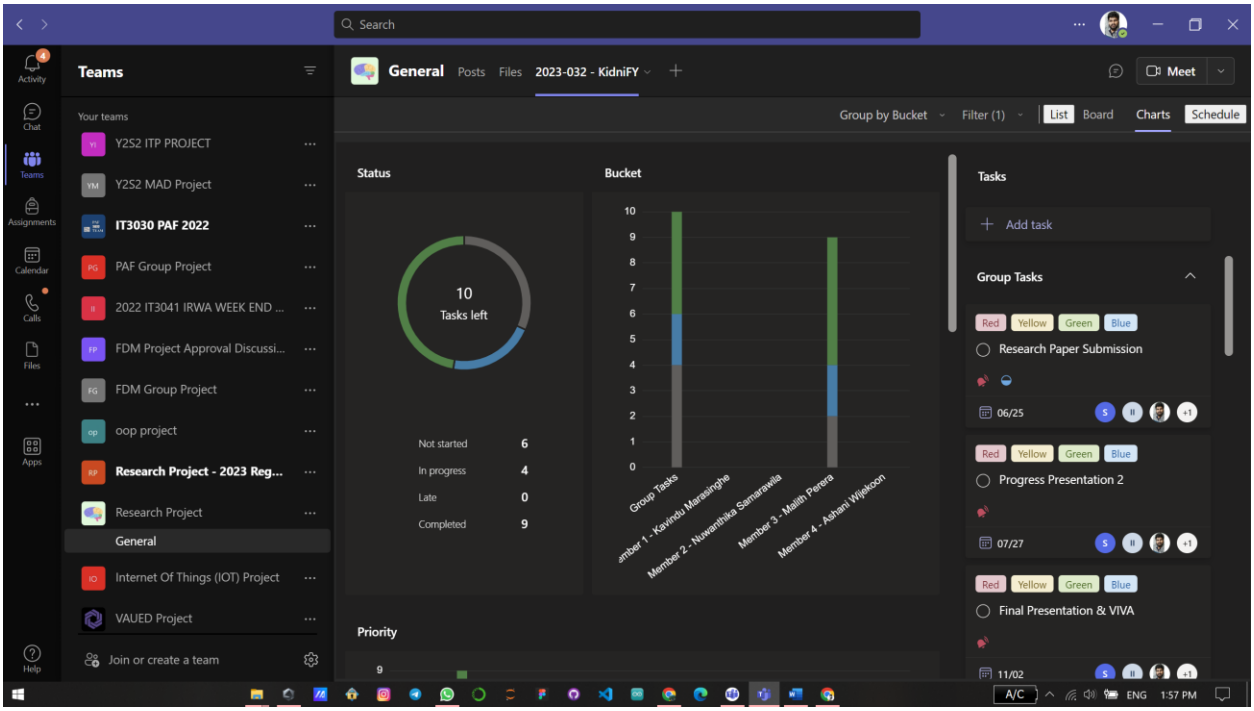
1.2 Meeting with the domain expert

Meeting with Consultant Nephrologist Dr. Pramil Rajakrishnan
At Kurunegala Teaching Hospital - Kidney dialysis Unit | 2023.04.29

Meet the Ward Master of Kidney Dialysis Unit
At Anuradhapura Teaching Hospital | 2023.05.01

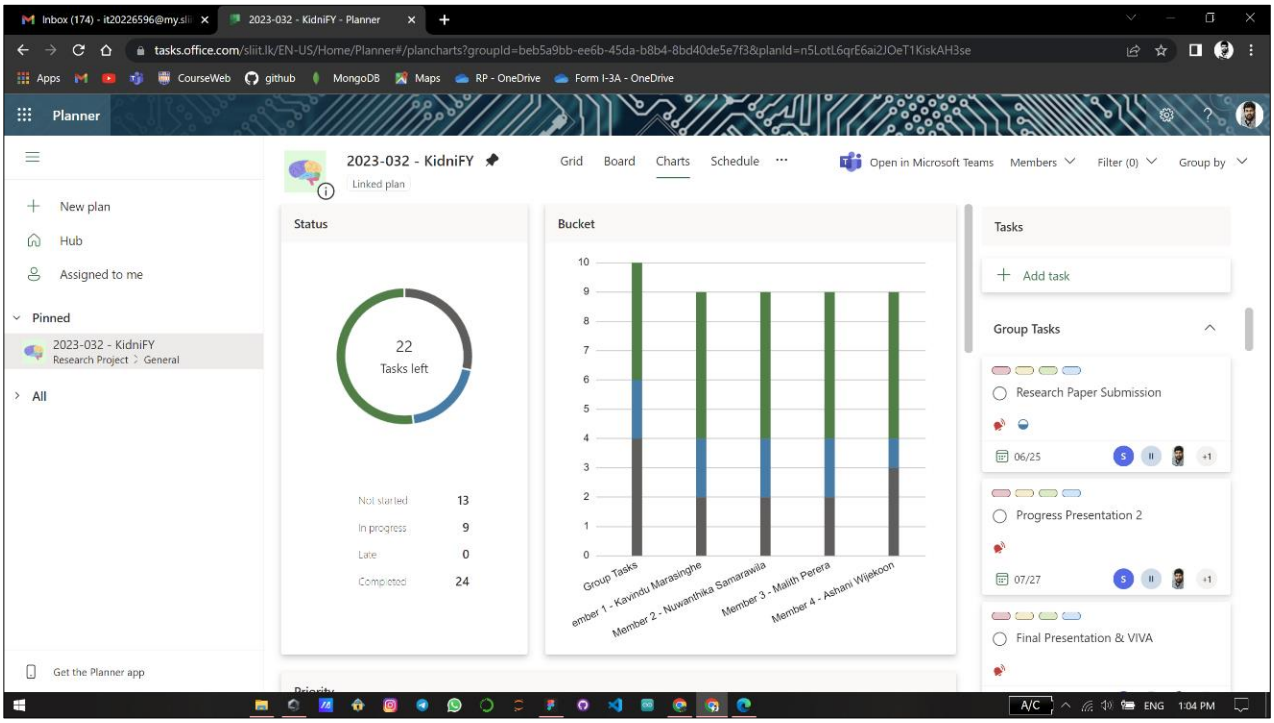
2 Screenshots of the Tasks by Planner

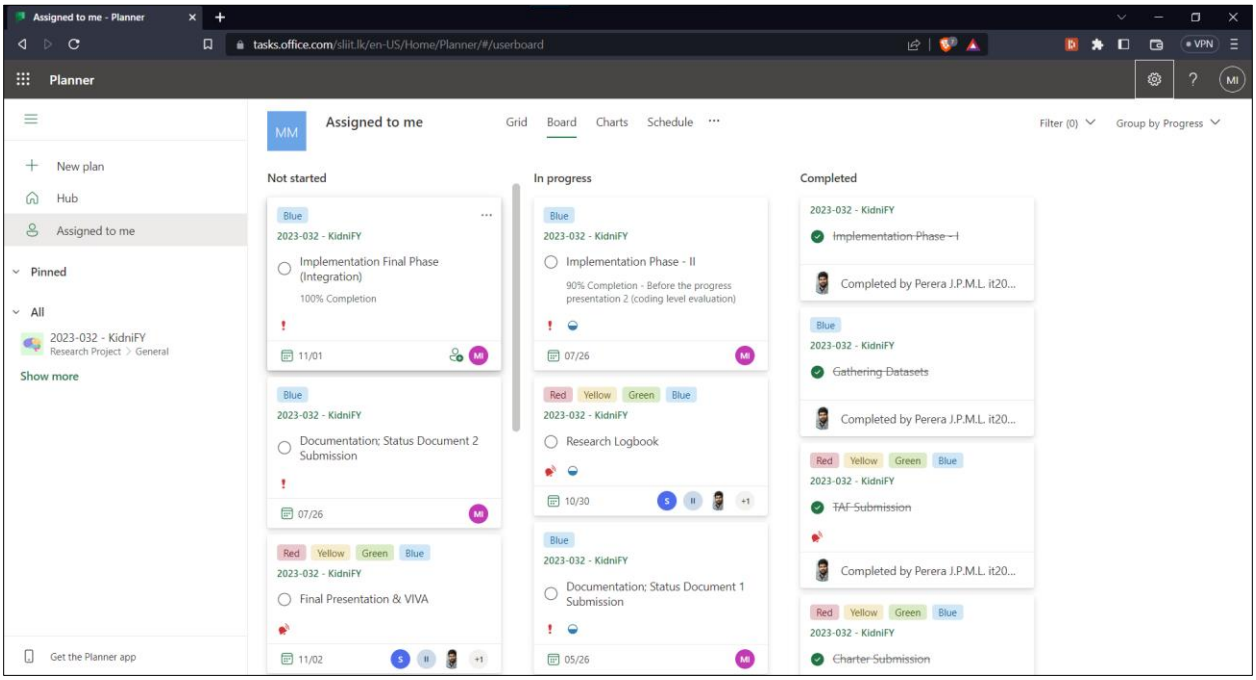
2.1 Chart Overview

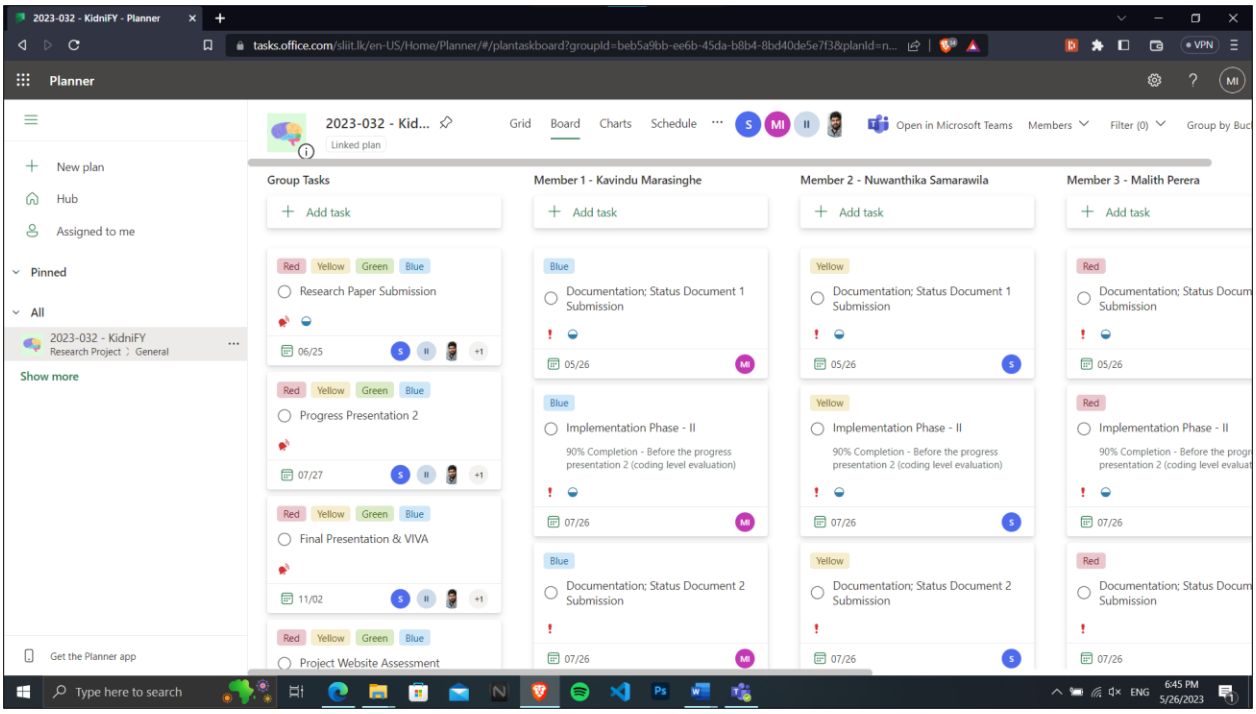


- To-do
- In Progress
- Completed

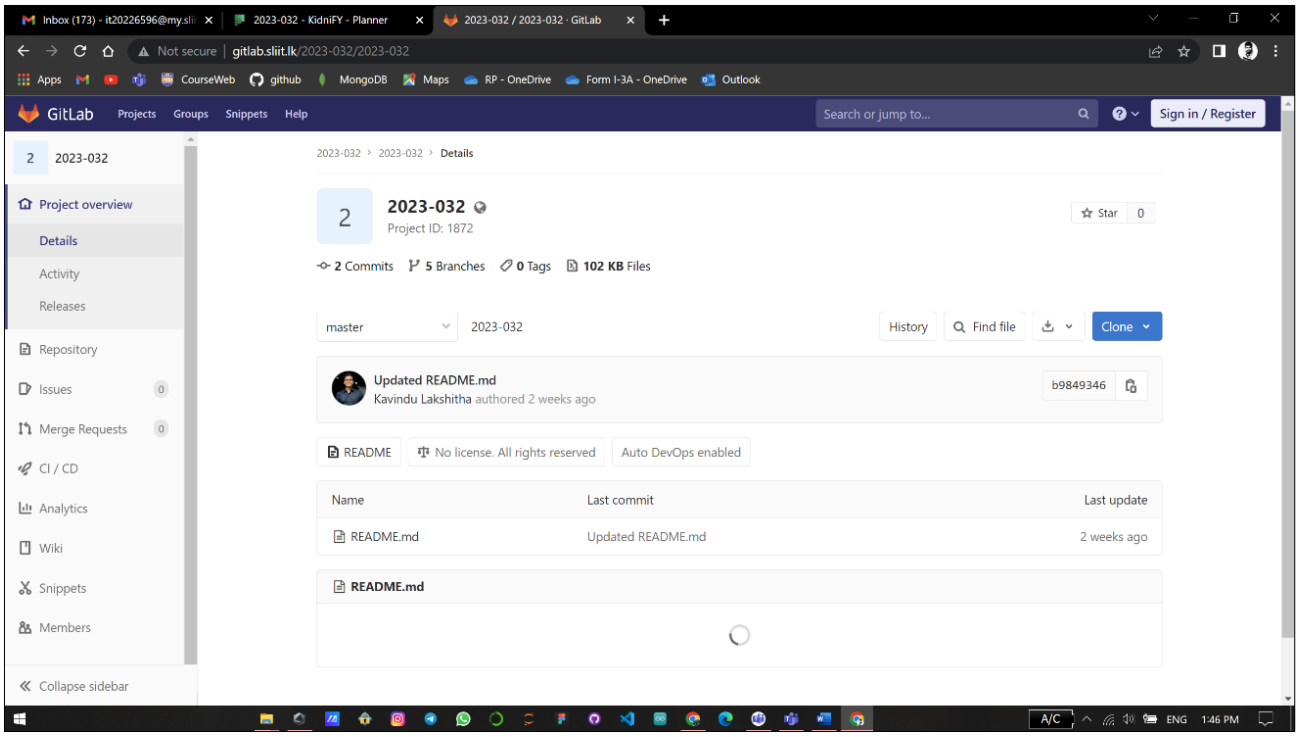
2.2 Bucket List

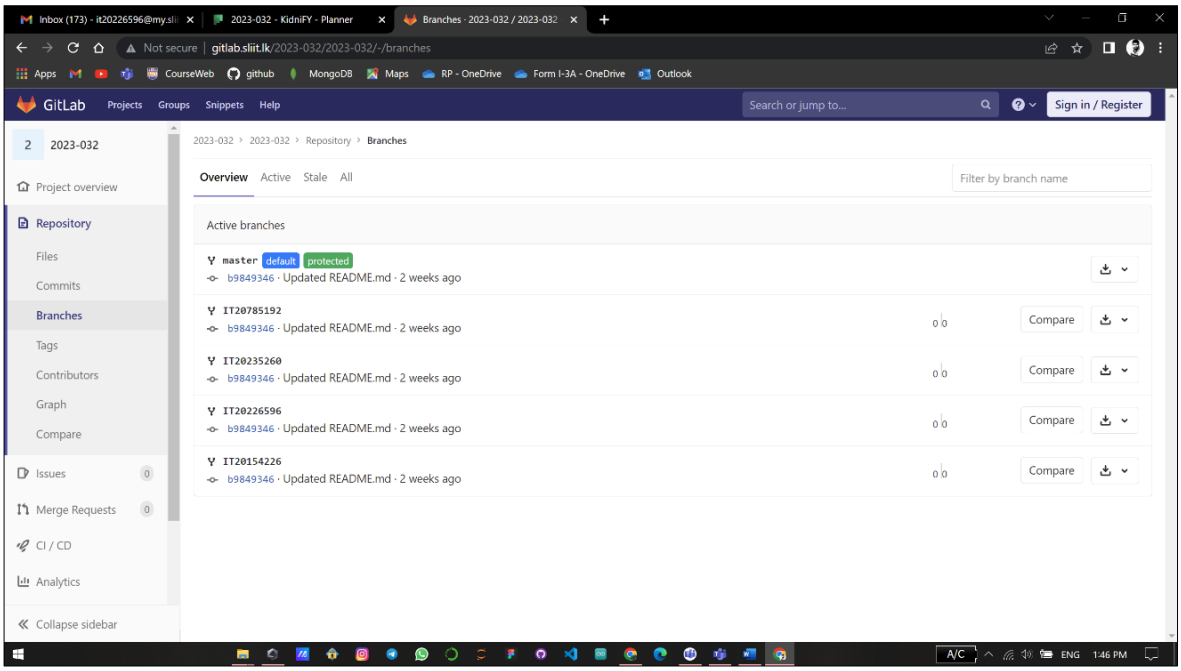






2.3 Screenshots of GitLab





3 Project Implementation

CKD Patient's kidney analysis using image processing

Loading the image dataset

Loading the image dataset

In [1]:
```
import cv2
import glob

# Specifying the directory path containing the images
image_dir = 'D:\kidneyUS_images_14_june_2022\kidneyUS_images_14_june_2022/*.png'  # Update with the appropriate file extension ar

# Use glob to retrieve the file paths of all images in the directory
image_paths = glob.glob(image_dir)

# Create an empty list to store the loaded images
images = []

# Loading each image using OpenCV and handle loading errors
for image_path in image_paths:
    image = cv2.imread(image_path)

    # Checking if the image is loaded successfully
    if image is not None:
        images.append(image)
    else:
        print(f"Error loading image: {image_path}")

# Checking the number of successfully loaded images
num_loaded_images = len(images)
print(f"Successfully loaded {num_loaded_images} images out of {len(image_paths)}")
```

```
Successfully loaded 534 images out of 534
```

Resizing and splitting the image dataset

In [2]:
```
import cv2

# Specify the desired width and height for the resized images
desired_width = 1024
desired_height = 768

# Create an empty list to store the resized images
resized_images = []

# Resize each image using OpenCV
for image in images:
    resized_image = cv2.resize(image, (desired_width, desired_height))
    resized_images.append(resized_image)

# Check if the images are resized successfully
for i, image in enumerate(images):
    original_height, original_width = image.shape[:2]
    resized_height, resized_width = resized_images[i].shape[:2]

    if original_height != resized_height or original_width != resized_width:
        print(f"Image {i+1} was not resized correctly.")
    else:
        print(f"Image {i+1} was resized successfully.")
```

```
Image 1 was resized successfully.
Image 2 was resized successfully.
Image 3 was not resized correctly.
Image 4 was resized successfully.
Image 5 was resized successfully.
Image 6 was resized successfully.
Image 7 was resized successfully.
Image 8 was resized successfully.
Image 9 was resized successfully.
Image 10 was resized successfully.
```

In [3]:
```
# Create an empty list to store the successfully resized images
resized_images_success = []

# Create a new list to store the indices of the successfully resized images
resized_indices_success = []

# Iterate over the original images and their corresponding resized images
for i, (image, resized_image) in enumerate(zip(images, resized_images)):
    if resized_image is not None:
        # If the resized image is not None, it indicates successful resizing
        resized_images_success.append(resized_image)
        resized_indices_success.append(i)
    else:
        print(f"Image {i+1} was not resized successfully and will be dropped.")

# Update the 'images' list with the successfully resized images
images = resized_images_success
```

In [4]:
```
# Check if any images were dropped during resizing
if len(images) == len(resized_images_success):
    print("All images were successfully resized.")
else:
    print(f"{len(images) - len(resized_images_success)} images were dropped during resizing.")
```

```
All images were successfully resized.
```

In [5]:
```
from sklearn.model_selection import train_test_split
import random

# Define the labels
labels = ["1", "2", "3", "3", "4", "5"]

# Assign labels randomly to the resized images
random_labels = random.choices(labels, k=len(resized_images_success))

# Split the data into training, validation, and testing sets
train_images, test_images, train_labels, test_labels = train_test_split(resized_images_success, random_labels, test_size=0.2, rar
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)

# Print the sizes of the resulting sets
print("Training set size:", len(train_images))
print("Validation set size:", len(val_images))
print("Testing set size:", len(test_images))
```

```
Training set size: 341
Validation set size: 86
Testing set size: 107
```

Pre-processing the input data for TensorFlow model Trainning

In [6]:
```
import tensorflow as tf
import numpy as np

# Convert images to NumPy arrays
train_images = np.array(train_images)
val_images = np.array(val_images)
test_images = np.array(test_images)

# Convert images to float32 and normalize
train_images_tensor = tf.convert_to_tensor(train_images.astype(np.float32) / 255.0)
val_images_tensor = tf.convert_to_tensor(val_images.astype(np.float32) / 255.0)
test_images_tensor = tf.convert_to_tensor(test_images.astype(np.float32) / 255.0)

# Convert labels to TensorFlow tensors
train_labels_tensor = tf.convert_to_tensor(train_labels)
val_labels_tensor = tf.convert_to_tensor(val_labels)
test_labels_tensor = tf.convert_to_tensor(test_labels)
```

Defining the CNN Model

In [7]:
```
import tensorflow as tf
from tensorflow.keras import layers

image_height = 1024
image_width = 768
num_channels = 3

# Define the CNN model
model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(768, 1024, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

CNN Model compilation for image classification

```
In [8]: model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

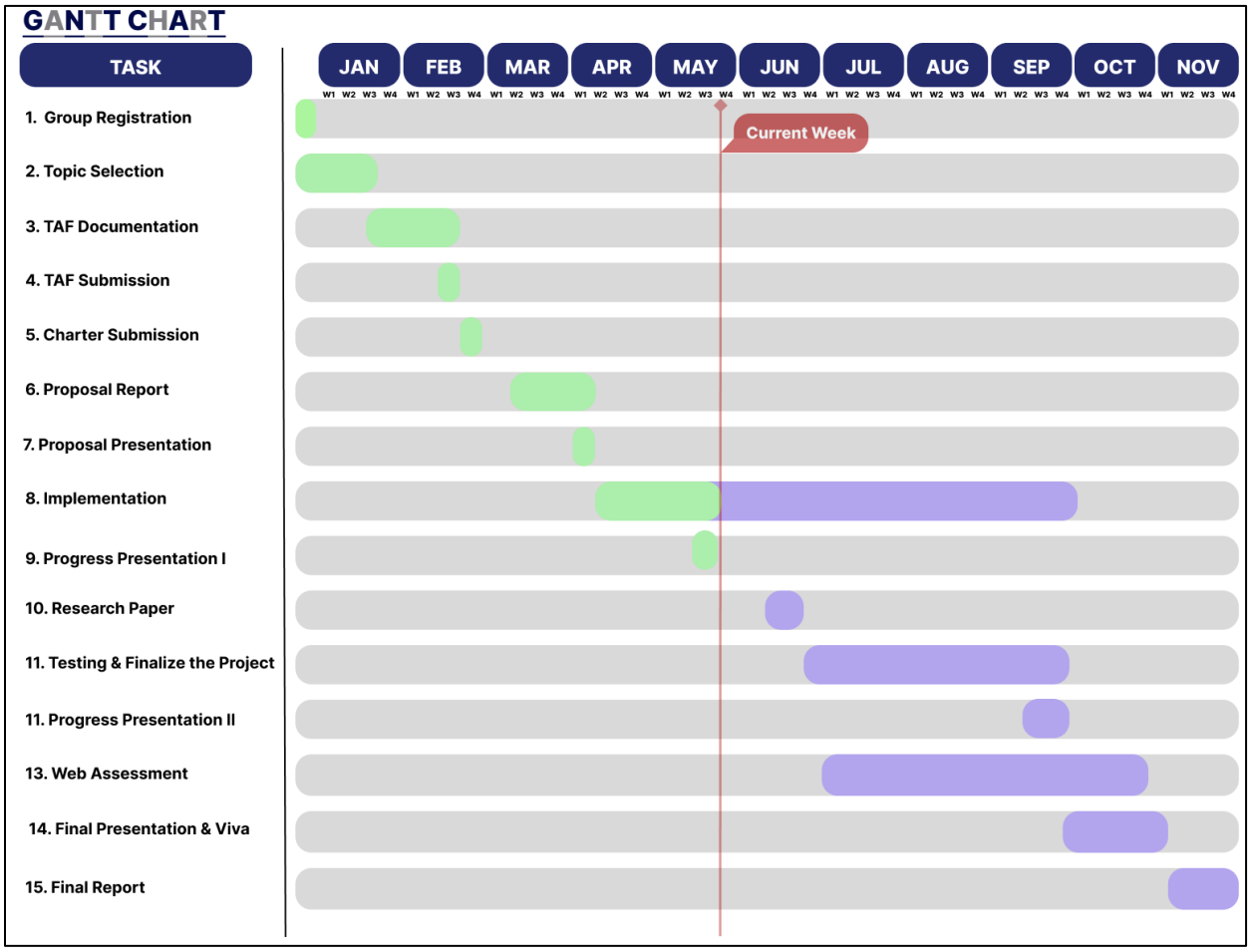
Label encoding and One-Hot Encoding for classification labels

```
In [10]: import numpy as np
         import tensorflow as tf

         actual_labels = ['3', '4', '5', '2', '3', '3', '3', '3', '3', '5', '3', '3', '1', '5', '3', '1', '4', '3', '1', '1', '2', '3', '5

         # Create a dictionary to map the label names to integer values
         label_to_integer = {label: i for i, label in enumerate(set(actual_labels))}

         # Convert the actual labels to numerical values using the mapping
         train_labels = np.array([label_to_integer[label] for label in actual_labels])

         # Convert the numerical labels to one-hot encoded format
         num_classes = len(label_to_integer)
         train_labels_encoded = tf.one_hot(train_labels, num_classes)

In [12]: train_labels_encoded = tf.reshape(train_labels_encoded, (-1, num_classes))

In [13]: import numpy as np

         num_classes = len(np.unique(train_labels))from collections import Counter

         label_counts = Counter(train_labels)
         num_classes = len(label_counts)

In [14]: num_classes = len(set(train_labels))

In [15]: from collections import Counter

         label_counts = Counter(train_labels)
         num_classes = len(label_counts)

In [16]: model.add(layers.Dense(534, activation='softmax'))
```

Training the model

```
In [*]: history = model.fit(train_images_tensor, train_labels_encoded, epochs=10, validation_data=(val_images_tensor, val_labels_tensor))

        Epoch 1/10
```

4 Gantt Chart



- Completed

- In Progress

5 Work Breakdown Structure