Code

Code corresponding for main()

# Main function to control navigation

def main():

    page = st.session\_state.get("page", "login")

    if page == "login":

        if login\_page():

            st.session\_state.page = "main"

    elif page == "main":

        if main\_page():

            st.session\_state.page = "count"

    elif page == "count":

        count\_page()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Code corresponding for login()

# Login page

def login\_page():

    st.title("Login")

    username = st.text\_input("Username")

    password = st.text\_input("Password", type="password")

    if st.button("Login"):

        if login(username, password):

            st.success("Login successful!")

            st.session\_state.username = username

            return True

        else:

            st.error("Invalid username or password. Please try again.")

            return False

Code corresponding for data\_entry()

# Page after successful login

def data\_entry():

    st.title("Royal Hospital")

    username = st.session\_state.username

    st.write(f"Welcome {username}!")

    patient\_name = st.text\_input("Patient name", "")

    st.session\_state.patient = patient\_name

    doctor\_name = st.selectbox("Doctor's Name", ["Select Doctor","Dr. Ambaan", "Dr. Ranga", "Dr. Appi Biju"])

    st.session\_state.doctor = doctor\_name

    if st.button("Proceed"):

        if patient\_name == "" or doctor\_name == "Select Doctor":

            st.error("Invalid Patient/Doctor name!!!")

        else:

            return True

Code corresponding for image upload\_and\_counting()

def count\_page():

    before\_list=[]

    after\_list =[]

    user = st.session\_state.username

    patient = st.session\_state.patient

    doctor = st.session\_state.doctor

    st.title("Royal Hospital")

    st.subheader(f"Staff Name : {user}")

    st.subheader(f"Patient Name : {patient}")

    st.subheader(f"Doctor Name : {doctor}")

    # Create two columns

    col1, col2 = st.columns(2)

    # Function to display uploaded images

    def display\_uploaded\_images(uploaded\_files, col,name):

        dicts={}

        for uploaded\_file in uploaded\_files:

            with col:

                if uploaded\_file is not None:

                    file\_bytes = np.asarray(bytearray(uploaded\_file.read()), dtype=np.uint8)

                    bw\_img,dicts = count\_instruments(file\_bytes)

                    b = ""

                    for i in dicts:

                        b+=str(i)+":"+str(dicts[i])+"\n"

                    st.image(bw\_img, caption=b,width=200)

        return dicts

    # Upload images for the first column

    with col1:

        st.header("Before Surgery")

        uploaded\_files\_col1 = [st.file\_uploader(f"Before Surgery {i}", type=["jpg", "jpeg", "png"]) for i in range(1, 6)]

        before = "Before Image"

        before\_list.append(display\_uploaded\_images(uploaded\_files\_col1, col1,before))

    # Upload images for the second column

    with col2:

        st.header("After Surgery")

        uploaded\_files\_col2 = [st.file\_uploader(f"After Surgery {i}", type=["jpg", "jpeg", "png"]) for i in range(1, 6)]

        after = "After Image"

        after\_list.append(display\_uploaded\_images(uploaded\_files\_col2, col2,after))

    if st.button("compare"):

        diff =check(before\_list,after\_list)

        if diff:

            st.write("Surgery Status : Missing")

            b = ""

            for i in diff:

                 b+=str(i)+":"+str(diff[i])+" "

            st.write("Missing Instruments :" ,b)

        else:

            st.write("Surgery Status : Successful",text\_color="green")

Code corresponding for detection()

class Detection:

    def \_\_init\_\_(self, model\_path, config\_path):

   detection = Detection('./Model/weight/best.onnx', 'data\_yaml')

        self.yolo = YOLO\_Pred(model\_path, config\_path)

    def counting(self, img\_path):

        # img = cv2.imread(img\_path)

        img = cv2.imdecode(img\_path, -1)

        # Predictions

        result = self.yolo.predictions(img)

        if isinstance(result, tuple) and len(result) == 2:

            img\_pred, before = result

        else:

            img\_pred = result

            before = []

        # Making dictionary of before

        b = {}

        for i in before:

            if i not in b:

                b[i] = 1

            else:

                b[i] += 1

        return img\_pred, b

Code corresponding for comparison()

def check(before,after):

    b={}

    for i in before:

        for j in i:

            if j in b:

                b[j] += i[j]

            else:

                b[j] = i[j]

    # print(b)

    a={}

    for i in after:

        for j in i:

            if j in a:

                a[j] += i[j]

            else:

                a[j] = i[j]

    diff = {}

    for i in b:

        if i not in a :

            diff[i] = b[i]

        elif b[i]  != a[i]:

            diff[i] = abs(b[i]-a[i])

    for i in a:

        if i not in b :

            diff[i] = a[i]

    # print(diff,"hasdfghj")

    return diff

Code corresponding for warning()

def speak\_warning\_message(msg):

       # Initialize the Text-to-Speech engine

        engine = pyttsx3.init()

        # Set properties (optional)

        engine.setProperty('rate', 150)  # Speed of speech

        engine.setProperty('volume', 0.9)  # Volume level (0.0 to 1.0)

        # Warning message

        warning\_message = msg

        # Speak the warning message

        engine.say(warning\_message)

        # Wait for speech to finish

        engine.runAndWait()

Code corresponding for model\_loading()

        # Load YAML

        with open('data.yaml',mode = 'r') as f:

            data\_yaml = yaml.load(f, Loader = SafeLoader)

        self.labels = data\_yaml['names']

        self.nc = data\_yaml['nc']

        # Load YOLO model

        self.yolo = cv2.dnn.readNetFromONNX('./Model/weights/best.onnx')

        self.yolo.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)

        self.yolo.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CPU)

Code corresponding for predictions()

    def predictions(self,image):

        row,col,d = image.shape

        l = []

        # get the YOLO prediction from the image

        # step1 convert image into a square image(array)

        max\_rc = max(row,col)

        input\_image = np.zeros((max\_rc,max\_rc,3),dtype = np.uint8)

        input\_image[0:row,0:col] = image

        #step2 get prediction from square array

        INPUT\_WH\_YOLO = 640

        blob = cv2.dnn.blobFromImage(input\_image,1/255,(INPUT\_WH\_YOLO,INPUT\_WH\_YOLO),swapRB = True,  crop = False)

        self.yolo.setInput(blob)

        preds = self.yolo.forward() #detection or prediction rom YOLO

        # Non maximum supression

        #step1 filter detection based on confidence score above 0.4 and probablity score above 0.25

        detections = preds[0]

        boxes = []

        confidences = []

        classes = []

        #width and height of image (input\_image)

        image\_w, image\_h = input\_image.shape[:2]

        x\_factor = image\_w/INPUT\_WH\_YOLO

        y\_factor = image\_h/INPUT\_WH\_YOLO

        for i in range(len(detections)):

            row = detections[i]

            confidence = row[4]

            if confidence > 0.4:

                class\_score = row[5:].max() #maximum probability from 2 objects

                class\_id = row[5:].argmax() #get the index position at which max probablity

                if class\_score > 0.25:

                    cx,cy,w,h = row[0:4]

                    #constructing bounding from 4 values

                    #left top right height

                    left = int((cx - 0.5\*w)\*x\_factor)

                    top = int((cy - 0.5\*h)\*y\_factor)

                    width = int(w\*x\_factor)

                    height = int(h\*y\_factor)

                    box = np.array([left,top,width,height])

                    #append values into the list

                    confidences.append(confidence)

                    boxes.append(box)

                    classes.append(class\_id)

        #clean

        boxes\_np = np.array(boxes).tolist()

        confidences\_np = np.array(confidences).tolist()

        if len(confidences) != 0:

            # Perform non-maximum suppression

            index = cv2.dnn.NMSBoxes(boxes\_np,confidences\_np,0.25,0.45).flatten()

        else:

            index = np.array([], dtype=np.int32)

        # Draw Bounding Box

        for i in index:

            #extract bounding box

            x,y,w,h = boxes\_np[i]

            bb\_conf = int(confidences\_np[i]\*100)

            classes\_id = classes[i]

            class\_name = self.labels[classes\_id]

            l.append(class\_name)

            colors = self.generate\_colors(classes\_id)

            text = f'{class\_name}: {bb\_conf}%'

            cv2.rectangle(image,(x,y),(x+w,y+h),colors,2)

            cv2.rectangle(image,(x,y-30),(x+w,y),colors,-1)

            cv2.putText(image,text,(x,y-10),cv2.FONT\_HERSHEY\_PLAIN,0.7,(0,0,0),1)

        return image,l

Code corresponding for box\_color()

    def generate\_colors(self, ID):

        np.random.seed(10)

        colors = np.random.randint(100, 255, size=(self.nc, 3)).tolist()

        return tuple(colors[ID % len(colors)])