In [1]:
```python
import os
import itertools
import random
import numpy as np
from tensorflow.keras.preprocessing.image import img_to_array , load_img
from tensorflow.keras.mixed_precision import set_global_policy
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense
from tensorflow.keras.optimizers import Adam
import collections
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns
import pandas as pd
```

In [2]:
```python
set_global_policy('mixed_float16')
base_dir = "RIDB_FORMATED"

num_individuals = 20
images_per_person = 5
```

```
INFO:tensorflow:Mixed precision compatibility check (mixed_float16): OK
Your GPU will likely run quickly with dtype policy mixed_float16 as it has compute capabi
lity of at least 7.0. Your GPU: NVIDIA GeForce RTX 3050 Laptop GPU, compute capability 8.
6
```

In [3]:
```python
def generate_image_paths(person_id):
    person_folder = os.path.join(base_dir, f"Person_{person_id}")
    image_paths = []
    for i in range(1, images_per_person + 1):
        image_name = f"IM{str(i).zfill(6)}_{person_id}.JPG"
        full_path = os.path.join(person_folder, image_name)
        image_paths.append(full_path)
    return image_paths
```

In [4]:
```python
# Generate all unique positive pairs from the dataset
all_positive_pairs = []
for person in range(1, num_individuals + 1):
    images = generate_image_paths(person)
    pairs = list(itertools.combinations(images, 2))
    all_positive_pairs.extend(pairs)

# Generate all negative pairs from the dataset
all_negative_pairs_train = []
all_person_images = {person: generate_image_paths(person) for person in range(1, num_ind
for person1, person2 in itertools.combinations(range(1, num_individuals - 4), 2):
    for img1 in all_person_images[person1]:
        for img2 in all_person_images[person2]:
            all_negative_pairs_train.append((img1, img2))


all_negative_pairs_test = []
all_person_images = {person: generate_image_paths(person) for person in range(1, num_ind
for person1, person2 in itertools.combinations(range(16, num_individuals + 1 ), 2):
    for img1 in all_person_images[person1]:
```

```
        for img2 in all_person_images[person2]:
            all_negative_pairs_test.append((img1, img2))
```

In [5]:
```
# Balanced Negative Sampling for Training and Testing
def balanced_negative_sampling(all_negative_pairs, num_samples):
    selected_negatives = random.sample(all_negative_pairs, min(num_samples, len(all_nega
    random.shuffle(selected_negatives)
    return selected_negatives

train_positive = all_positive_pairs[:150]
test_positive = all_positive_pairs[150:200]

train_negative = balanced_negative_sampling(all_negative_pairs_train, 150)
test_negative = balanced_negative_sampling(all_negative_pairs_test, 50)
```

In [6]:
```
final_train_positive = train_positive
final_train_negative = train_negative

final_train_pairs = final_train_positive + final_train_negative
final_train_labels = [1] * len(final_train_positive) + [0] * len(final_train_negative)

final_test_pairs = test_positive + test_negative
final_test_labels = [1] * len(test_positive) + [0] * len(test_negative)

combined_train = list(zip(final_train_pairs, final_train_labels))
random.shuffle(combined_train)
final_train_pairs, final_train_labels = zip(*combined_train)

combined_test = list(zip(final_test_pairs, final_test_labels))
random.shuffle(combined_test)
final_test_pairs, final_test_labels = zip(*combined_test)
```

In [7]:
```
print("Final Training Set:")
print("Total pairs:", len(final_train_pairs))
print("Positive pairs:", final_train_labels.count(1))
print("Negative pairs:", final_train_labels.count(0))
print("\nFinal Test Set:")
print("Total pairs:", len(final_test_pairs))
print("Positive pairs:", final_test_labels.count(1))
print("Negative pairs:", final_test_labels.count(0))
```

```
Final Training Set:
Total pairs: 300
Positive pairs: 150
Negative pairs: 150

Final Test Set:
Total pairs: 100
Positive pairs: 50
Negative pairs: 50
```

In [8]:
```
train_pairs = final_train_pairs
train_labels = final_train_labels
test_pairs = final_test_pairs
test_labels = final_test_labels
```

In [9]:
```
def extract_person_id(filepath):
    dir_name = os.path.basename(os.path.dirname(filepath))
```

```python
    if dir_name.startswith("Person_"):
        return dir_name.split("_")[1]
    else:
        return None

# Count positive pairs (label==1) where the person ID in image1 differs from image2
mismatch_count = 0
for pair, label in zip(test_pairs, test_labels):
    if label == 1:
        person1 = extract_person_id(pair[0])
        person2 = extract_person_id(pair[1])
        if person1 != person2:
            mismatch_count += 1

print("Number of label=1 pairs with different person IDs:", mismatch_count)


# Count negative pairs (label == 0) where the person ID in both images is the same
same_person_negative_count = 0
for pair, label in zip(test_pairs, test_labels):
    if label == 0:
        person1 = extract_person_id(pair[0])
        person2 = extract_person_id(pair[1])
        if person1 == person2:
            same_person_negative_count += 1

print("Number of label=0 pairs with the same person ID:", same_person_negative_count)
```

```
Number of label=1 pairs with different person IDs: 0
Number of label=0 pairs with the same person ID: 0
```

In [10]:

```python
# Count the occurrences of each pair in the training set.
train_pair_counts = collections.Counter(train_pairs)
train_duplicate_pairs = {pair: count for pair, count in train_pair_counts.items() if cou

if train_duplicate_pairs:
    print("Repeated pairs in the training set:")
    for pair, count in train_duplicate_pairs.items():
        print(f"{pair}: {count} times")
else:
    print("No repeated pairs in the training set.")

# Count the occurrences of each pair in the test set.
test_pair_counts = collections.Counter(test_pairs)
test_duplicate_pairs = {pair: count for pair, count in test_pair_counts.items() if count

if test_duplicate_pairs:
    print("Repeated pairs in the test set:")
    for pair, count in test_duplicate_pairs.items():
        print(f"{pair}: {count} times")
else:
    print("No repeated pairs in the test set.")

# Check for pairs that appear in both train and test sets.
common_pairs = set(train_pairs) & set(test_pairs)
if common_pairs:
    print("Pairs present in both train and test sets:")
    for pair in common_pairs:
        print(pair)
```

```python
else:
    print("No pairs are present in both train and test sets.")
```

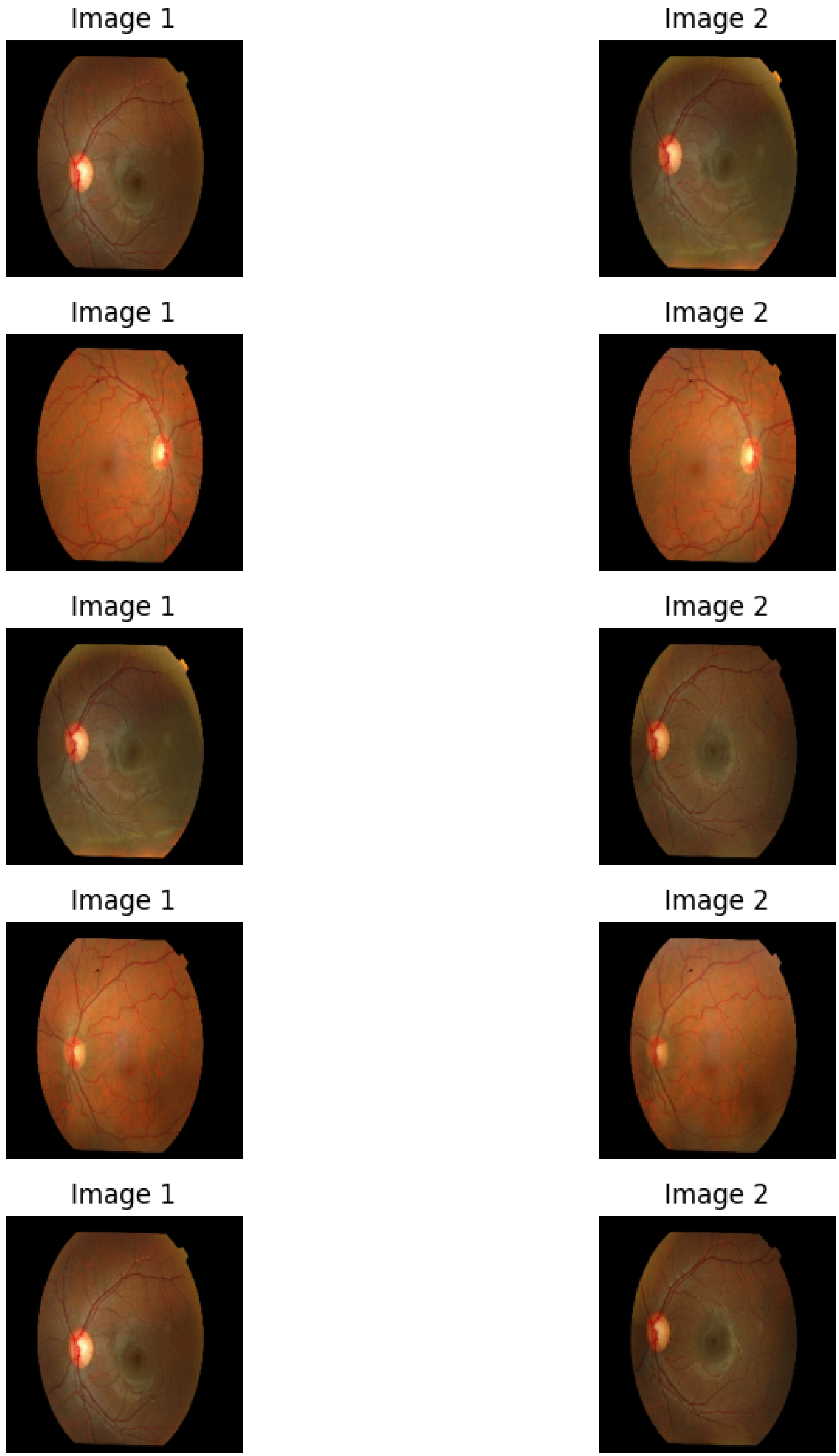
```
No repeated pairs in the training set.
No repeated pairs in the test set.
No pairs are present in both train and test sets.
```

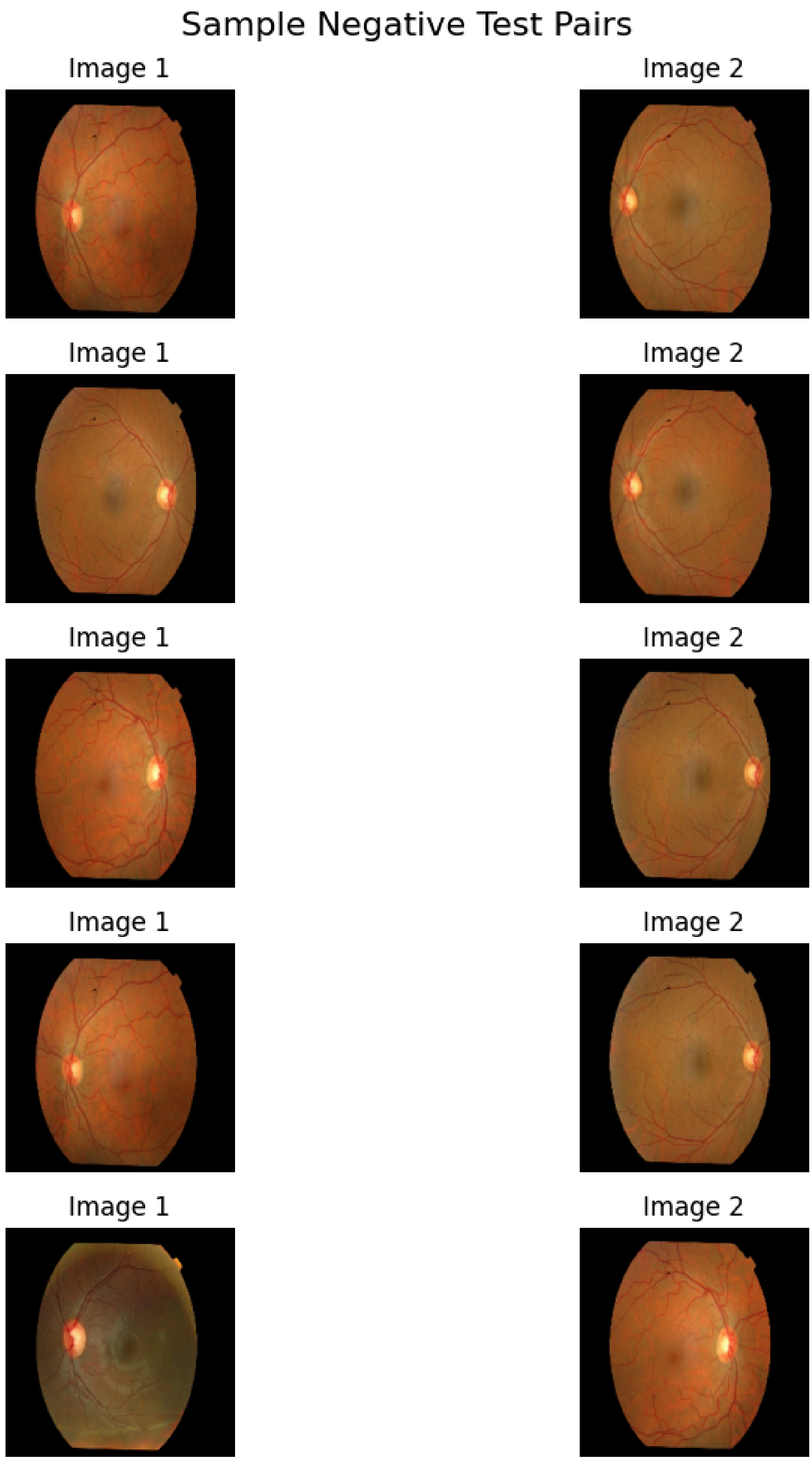
In [11]:

```python
def plot_image_pairs(pairs, num_samples=5, title="Sample Image Pairs"):
    fig, axes = plt.subplots(num_samples, 2, figsize=(10, num_samples * 2))
    fig.suptitle(title, fontsize=16)
    for i in range(num_samples):
        img1_path, img2_path = pairs[i]
        img1 = load_img(img1_path, target_size=(224, 224))
        img2 = load_img(img2_path, target_size=(224, 224))
        axes[i, 0].imshow(img1)
        axes[i, 0].axis("off")
        axes[i, 0].set_title("Image 1")
        axes[i, 1].imshow(img2)
        axes[i, 1].axis("off")
        axes[i, 1].set_title("Image 2")
    plt.tight_layout()
    plt.show()

test_pairs_list = list(test_pairs)
test_labels_list = list(test_labels)

positive_pairs = [pair for pair, label in zip(test_pairs_list, test_labels_list) if labe
negative_pairs = [pair for pair, label in zip(test_pairs_list, test_labels_list) if labe

print("Total positive test pairs:", len(positive_pairs))
print("Total negative test pairs:", len(negative_pairs))

sample_positive = random.sample(positive_pairs, 5)
sample_negative = random.sample(negative_pairs, 5)

plot_image_pairs(sample_positive, num_samples=5, title="Sample Positive Test Pairs")
plot_image_pairs(sample_negative, num_samples=5, title="Sample Negative Test Pairs")
```

```
Total positive test pairs: 50
Total negative test pairs: 50
```

Sample Positive Test Pairs




Sample Negative Test Pairs
Image 1
Image 2
Image 1
Image 2
Image 1
Image 2
Image 1
Image 2
Image 1
Image 2

In [12]:
```
def load_and_preprocess_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    img = img_to_array(img)
    img = img / 255.0
    return img

def load_pair(pair):
    img1_path, img2_path = pair
    img1 = load_and_preprocess_image(img1_path)
    img2 = load_and_preprocess_image(img2_path)
    return img1, img2

def create_dataset(pairs, labels):
    imgs1, imgs2 = [], []
    for pair in pairs:
        image1, image2 = load_pair(pair)
        imgs1.append(image1)
        imgs2.append(image2)
    return np.array(imgs1), np.array(imgs2), np.array(labels)

train_img1, train_img2, train_labels_np = create_dataset(train_pairs, train_labels)
test_img1, test_img2, test_labels_np = create_dataset(test_pairs, test_labels)

print("Training samples:", train_img1.shape, train_img2.shape, train_labels_np.shape)
print("Testing samples:", test_img1.shape, test_img2.shape, test_labels_np.shape)
```

```
Training samples: (300, 224, 224, 3) (300, 224, 224, 3) (300,)
Testing samples: (100, 224, 224, 3) (100, 224, 224, 3) (100,)
```

In [13]:
```
def euclidean_distance(vectors):
    vector1, vector2 = vectors
    sum_square = tf.reduce_sum(tf.square(vector1 - vector2), axis=1, keepdims=True)
    return tf.sqrt(tf.maximum(sum_square, tf.keras.backend.epsilon()))
```

In [14]:
```
def embedding_network(in_shape):
    input_layer = keras.layers.Input(in_shape)
    x = keras.layers.Conv2D(128, (5, 5), activation="relu")(input_layer)
    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = keras.layers.Conv2D(64, (3, 3), activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = keras.layers.Conv2D(32, (3, 3), activation="relu")(x)
    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = keras.layers.Dropout(0.3)(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(128, activation="relu")(x)
    x = keras.layers.Dense(64, activation="relu")(x)
    x = keras.layers.Dense(32, activation="relu")(x)
    return keras.Model(inputs=input_layer, outputs=x, name="base_network")


def SiameseNetwork(in_shape):
    input_1 = Input(shape=in_shape)
    input_2 = Input(shape=in_shape)
    embedding_net_obj = embedding_network(in_shape)
    twin_1 = embedding_net_obj(input_1)
    twin_2 = embedding_net_obj(input_2)
    merge_layer = Lambda(euclidean_distance, output_shape=(1,))([twin_1, twin_2])
    dense_layer = Dense(128, activation="relu")(merge_layer)
```

```
    output_layer = Dense(1, activation="sigmoid", dtype="float32")(dense_layer)

    return keras.Model(inputs=[input_1, input_2], outputs=output_layer)



input_shape = (224, 224, 3)
siamese_net = SiameseNetwork(input_shape)
siamese_net.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=Adam(learning_rate=2e-3),
    metrics=['accuracy']
)
siamese_net.summary()
```

```
Model: "model"
__________________________________________________________________________________________
________
Layer (type)                    Output Shape          Param #     Connected to
==========================================================================================
========
input_1 (InputLayer)            [(None, 224, 224, 3) 0

__________________________________________________________________________________________
________
input_2 (InputLayer)            [(None, 224, 224, 3) 0

__________________________________________________________________________________________
________
base_network (Functional)       (None, 32)           2881344     input_1[0][0]
                                                                 input_2[0][0]

__________________________________________________________________________________________
________
lambda (Lambda)                 (None, 1)            0           base_network[0][0]
                                                                 base_network[1][0]

__________________________________________________________________________________________
________
dense_3 (Dense)                 (None, 128)          256         lambda[0][0]
__________________________________________________________________________________________
________
dense_4 (Dense)                 (None, 1)            129         dense_3[0][0]
==========================================================================================
========
Total params: 2,881,729
Trainable params: 2,881,729
Non-trainable params: 0

__________________________________________________________________________________________
________
```

In [15]:
```
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint_path = "best_siamese_model.h5"

checkpoint_callback = ModelCheckpoint(
    checkpoint_path,
    monitor="val_accuracy",
    save_best_only=True,
    save_weights_only=True,
    mode="max",
    verbose=1
)
```

In [16]:
```
history = siamese_net.fit(
    [train_img1, train_img2],
    train_labels_np,
    validation_data=([test_img1, test_img2], test_labels_np),
    epochs=30,
    batch_size=8,
    callbacks=[checkpoint_callback]
)
```

```
Epoch 1/30
38/38 [==============================] - 9s 147ms/step - loss: 0.6264 - accuracy: 0.6067
- val_loss: 0.5045 - val_accuracy: 0.7600

Epoch 00001: val_accuracy improved from -inf to 0.76000, saving model to best_siamese_mod
el.h5
Epoch 2/30
38/38 [==============================] - 5s 123ms/step - loss: 0.5015 - accuracy: 0.7033
- val_loss: 0.5370 - val_accuracy: 0.8400

Epoch 00002: val_accuracy improved from 0.76000 to 0.84000, saving model to best_siamese_
model.h5
Epoch 3/30
38/38 [==============================] - 5s 123ms/step - loss: 0.4833 - accuracy: 0.7767
- val_loss: 0.4157 - val_accuracy: 0.9100

Epoch 00003: val_accuracy improved from 0.84000 to 0.91000, saving model to best_siamese_
model.h5
Epoch 4/30
38/38 [==============================] - 5s 122ms/step - loss: 0.3883 - accuracy: 0.8433
- val_loss: 0.2876 - val_accuracy: 0.9200

Epoch 00004: val_accuracy improved from 0.91000 to 0.92000, saving model to best_siamese_
model.h5
Epoch 5/30
38/38 [==============================] - 5s 121ms/step - loss: 0.3475 - accuracy: 0.8567
- val_loss: 0.3248 - val_accuracy: 0.9200

Epoch 00005: val_accuracy did not improve from 0.92000
Epoch 6/30
38/38 [==============================] - 5s 121ms/step - loss: 0.3151 - accuracy: 0.8767
- val_loss: 0.2867 - val_accuracy: 0.9200

Epoch 00006: val_accuracy did not improve from 0.92000
Epoch 7/30
38/38 [==============================] - 5s 121ms/step - loss: 0.3090 - accuracy: 0.8667
- val_loss: 0.2597 - val_accuracy: 0.9100

Epoch 00007: val_accuracy did not improve from 0.92000
Epoch 8/30
38/38 [==============================] - 5s 121ms/step - loss: 0.2766 - accuracy: 0.8933
- val_loss: 0.1995 - val_accuracy: 0.9100

Epoch 00008: val_accuracy did not improve from 0.92000
Epoch 9/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2281 - accuracy: 0.9200
- val_loss: 0.2080 - val_accuracy: 0.9300

Epoch 00009: val_accuracy improved from 0.92000 to 0.93000, saving model to best_siamese_
model.h5
Epoch 10/30
38/38 [==============================] - 5s 120ms/step - loss: 0.3473 - accuracy: 0.8533
- val_loss: 0.2670 - val_accuracy: 0.9300

Epoch 00010: val_accuracy did not improve from 0.93000
Epoch 11/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2824 - accuracy: 0.8800
- val_loss: 0.2044 - val_accuracy: 0.9300
```

```
Epoch 00011: val_accuracy did not improve from 0.93000
Epoch 12/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2296 - accuracy: 0.9133
- val_loss: 0.2049 - val_accuracy: 0.9300

Epoch 00012: val_accuracy did not improve from 0.93000
Epoch 13/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2154 - accuracy: 0.9133
- val_loss: 0.1905 - val_accuracy: 0.9300

Epoch 00013: val_accuracy did not improve from 0.93000
Epoch 14/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1607 - accuracy: 0.9633
- val_loss: 0.1991 - val_accuracy: 0.9300

Epoch 00014: val_accuracy did not improve from 0.93000
Epoch 15/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1407 - accuracy: 0.9600
- val_loss: 0.1656 - val_accuracy: 0.9400

Epoch 00015: val_accuracy improved from 0.93000 to 0.94000, saving model to best_siamese_
model.h5
Epoch 16/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2231 - accuracy: 0.9133
- val_loss: 0.2345 - val_accuracy: 0.8800

Epoch 00016: val_accuracy did not improve from 0.94000
Epoch 17/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1521 - accuracy: 0.9533
- val_loss: 0.1857 - val_accuracy: 0.9300

Epoch 00017: val_accuracy did not improve from 0.94000
Epoch 18/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2395 - accuracy: 0.9167
- val_loss: 0.2482 - val_accuracy: 0.8900

Epoch 00018: val_accuracy did not improve from 0.94000
Epoch 19/30
38/38 [==============================] - 5s 119ms/step - loss: 0.3216 - accuracy: 0.8667
- val_loss: 0.2491 - val_accuracy: 0.8700

Epoch 00019: val_accuracy did not improve from 0.94000
Epoch 20/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2302 - accuracy: 0.9033
- val_loss: 0.1588 - val_accuracy: 0.9300

Epoch 00020: val_accuracy did not improve from 0.94000
Epoch 21/30
38/38 [==============================] - 5s 121ms/step - loss: 0.1633 - accuracy: 0.9400
- val_loss: 0.1772 - val_accuracy: 0.9000

Epoch 00021: val_accuracy did not improve from 0.94000
Epoch 22/30
38/38 [==============================] - 4s 118ms/step - loss: 0.1311 - accuracy: 0.9567
- val_loss: 0.1231 - val_accuracy: 0.9500

Epoch 00022: val_accuracy improved from 0.94000 to 0.95000, saving model to best_siamese_
model.h5
Epoch 23/30
```

```
38/38 [==============================] - 5s 120ms/step - loss: 0.1254 - accuracy: 0.9633
- val_loss: 0.1915 - val_accuracy: 0.9100

Epoch 00023: val_accuracy did not improve from 0.95000
Epoch 24/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1204 - accuracy: 0.9567
- val_loss: 0.1978 - val_accuracy: 0.9200

Epoch 00024: val_accuracy did not improve from 0.95000
Epoch 25/30
38/38 [==============================] - 5s 121ms/step - loss: 0.0893 - accuracy: 0.9767
- val_loss: 0.1517 - val_accuracy: 0.9000

Epoch 00025: val_accuracy did not improve from 0.95000
Epoch 26/30
38/38 [==============================] - 5s 120ms/step - loss: 0.0665 - accuracy: 0.9867
- val_loss: 0.2326 - val_accuracy: 0.8900

Epoch 00026: val_accuracy did not improve from 0.95000
Epoch 27/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1066 - accuracy: 0.9600
- val_loss: 0.1122 - val_accuracy: 0.9300

Epoch 00027: val_accuracy did not improve from 0.95000
Epoch 28/30
38/38 [==============================] - 5s 120ms/step - loss: 0.2216 - accuracy: 0.9133
- val_loss: 0.1409 - val_accuracy: 0.9300

Epoch 00028: val_accuracy did not improve from 0.95000
Epoch 29/30
38/38 [==============================] - 5s 120ms/step - loss: 0.1874 - accuracy: 0.9233
- val_loss: 0.1136 - val_accuracy: 0.9700

Epoch 00029: val_accuracy improved from 0.95000 to 0.97000, saving model to best_siamese_
model.h5
Epoch 30/30
38/38 [==============================] - 5s 120ms/step - loss: 0.0826 - accuracy: 0.9667
- val_loss: 0.1341 - val_accuracy: 0.9400

Epoch 00030: val_accuracy did not improve from 0.97000
```

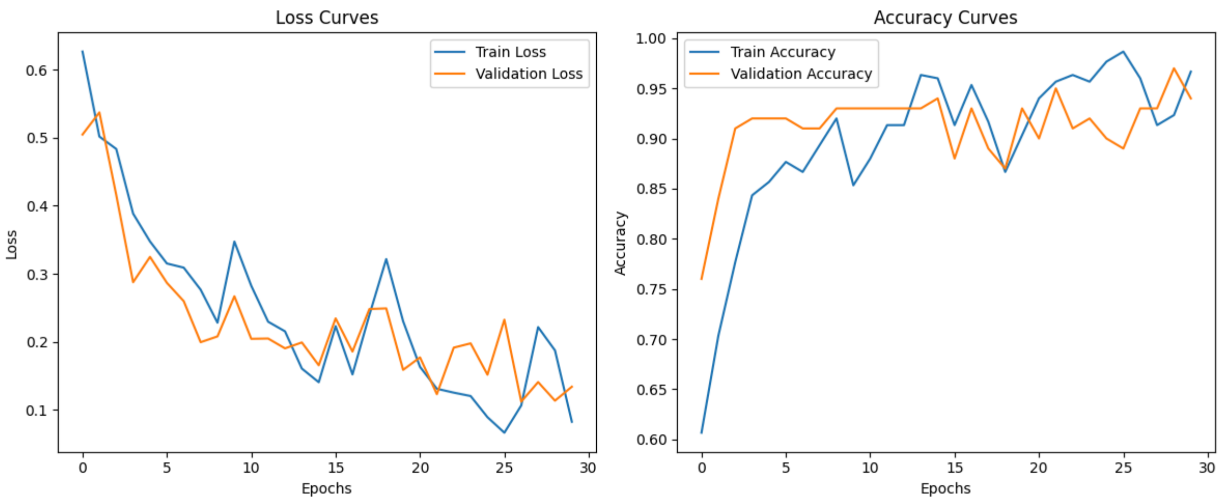
In [17]:

```
siamese_net.load_weights(checkpoint_path)
```

In [18]:

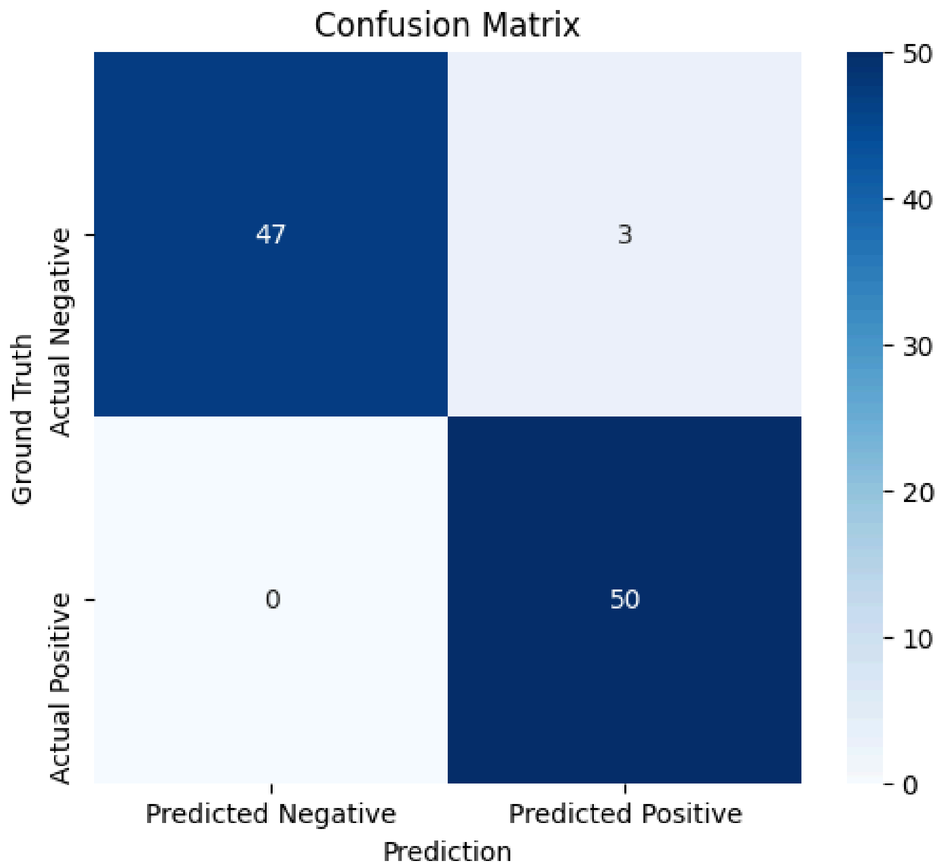
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("Loss Curves")

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy Curves")
```

```
plt.tight_layout()
plt.show()
```

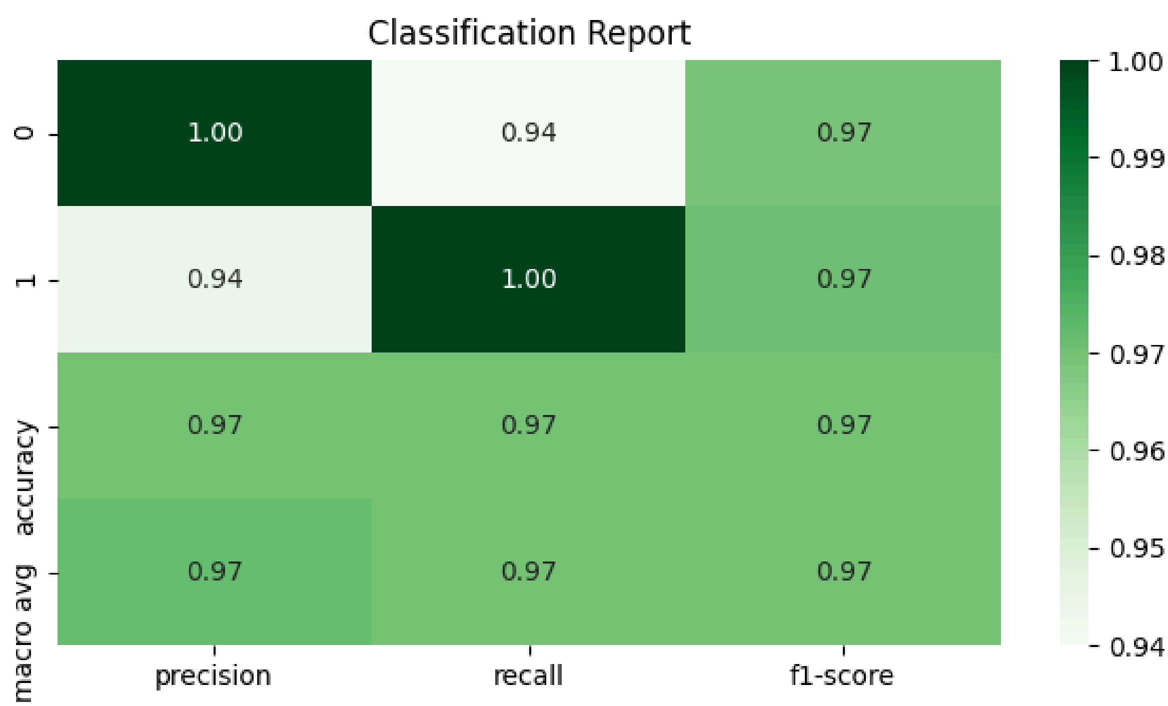


In [19]:
```
test_predictions = siamese_net.predict([test_img1, test_img2])
test_predictions = test_predictions.flatten()

threshold = 0.5
test_pred_labels = (test_predictions > threshold).astype(int)
```

In [20]:
```
cm = confusion_matrix(test_labels_np, test_pred_labels)

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues",
            xticklabels=["Predicted Negative", "Predicted Positive"],
            yticklabels=["Actual Negative", "Actual Positive"])
plt.title("Confusion Matrix")
plt.xlabel("Prediction")
plt.ylabel("Ground Truth")
plt.show()
```



In [21]:
```python
report = classification_report(test_labels_np, test_pred_labels, output_dict=True)
report_df = pd.DataFrame(report).transpose()

plt.figure(figsize=(8, 4))
sns.heatmap(report_df.iloc[:-1, :-1], annot=True, cmap="Greens", fmt=".2f")
plt.title("Classification Report")
plt.show()
```

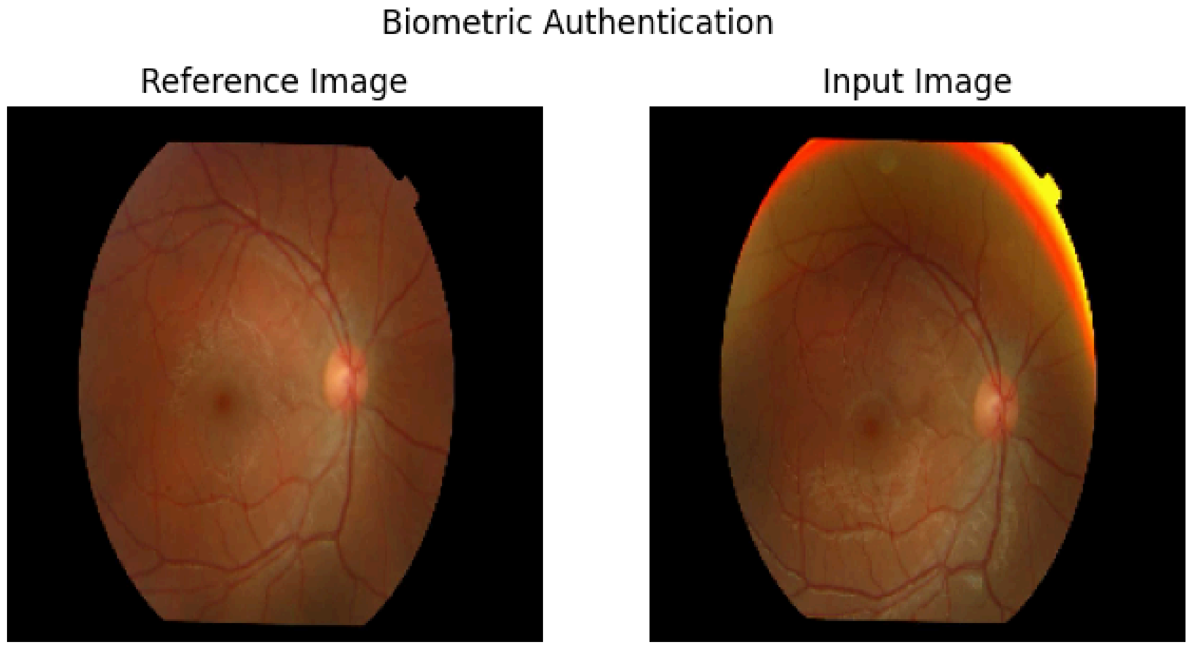


In [22]:
```
def authenticate_user(user_id, input_image_path, threshold=0.5):
    ref_path = generate_image_paths(user_id)[0]

    ref_img = load_and_preprocess_image(ref_path)
    input_img = load_and_preprocess_image(input_image_path)

    ref_img_exp = np.expand_dims(ref_img, axis=0)
    input_img_exp = np.expand_dims(input_img, axis=0)

    similarity = siamese_net.predict([ref_img_exp, input_img_exp])
    similarity_score = similarity[0][0]

    authenticated = similarity_score > threshold

    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    ref_disp = load_img(ref_path, target_size=(224, 224))
    input_disp = load_img(input_image_path, target_size=(224, 224))
    axes[0].imshow(ref_disp)
    axes[0].set_title("Reference Image")
    axes[0].axis("off")
    axes[1].imshow(input_disp)
    axes[1].set_title("Input Image")
    axes[1].axis("off")
    plt.suptitle("Biometric Authentication")
    plt.show()

    print(f"User ID: {user_id}")
    print(f"Reference image: {ref_path}")
    print(f"Input image: {input_image_path}")
    print(f"Similarity score: {similarity_score:.4f}")
    if authenticated:
        print("Authentication Successful: The user is verified.\n")
    else:
        print("Authentication Failed: The user is not verified.\n")

    return similarity_score, authenticated
```

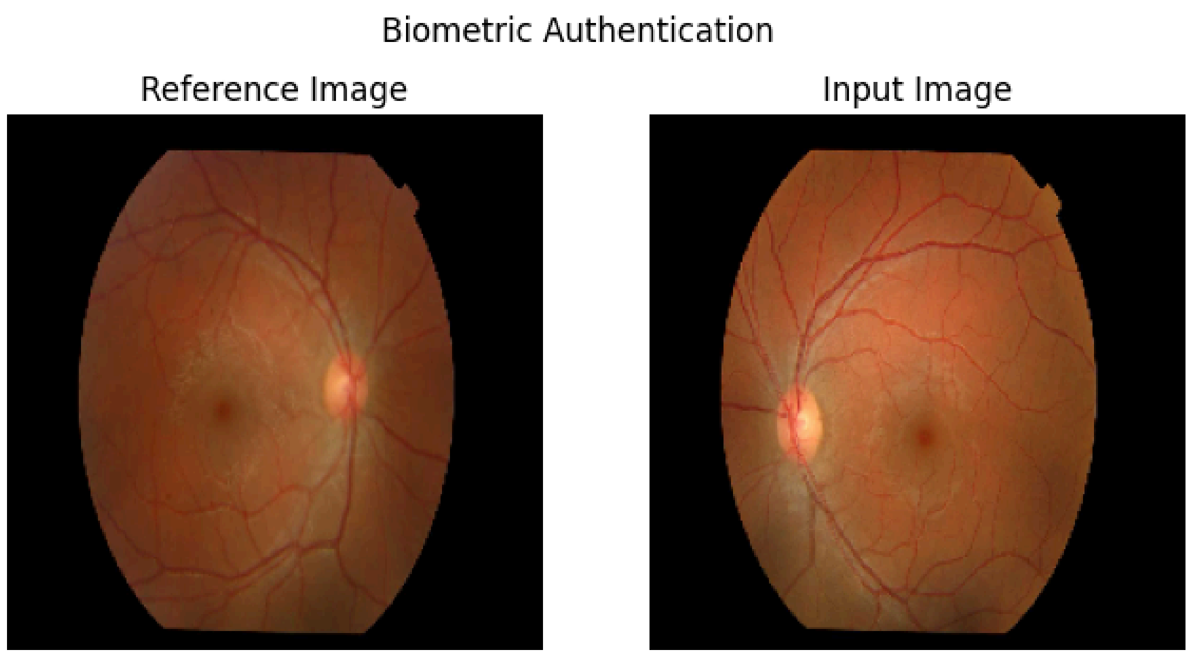
In [23]:
```
# Example 1: Genuine authentication.
user_id = 1
input_image_genuine = os.path.join(base_dir, "Person_1", "IM000002_1.JPG")
print("Genuine Authentication Example:")
score, auth = authenticate_user(user_id, input_image_genuine, threshold=0.5)

# Example 2: Impostor authentication.
user_id = 1
input_image_impostor = os.path.join(base_dir, "Person_2", "IM000001_2.JPG")
print("Impostor Authentication Example:")
score, auth = authenticate_user(user_id, input_image_impostor, threshold=0.5)

# Example 3: Genuine authentication.
user_id = 20
input_image_impostor = os.path.join(base_dir, "Person_20", "IM000005_20.JPG")
print("Impostor Authentication Example:")
score, auth = authenticate_user(user_id, input_image_impostor, threshold=0.5)

# Example 4: Impostor authentication.
user_id = 10
input_image_impostor = os.path.join(base_dir, "Person_20", "IM000005_20.JPG")
```

```
print("Impostor Authentication Example:")
score, auth = authenticate_user(user_id, input_image_impostor, threshold=0.5)
```

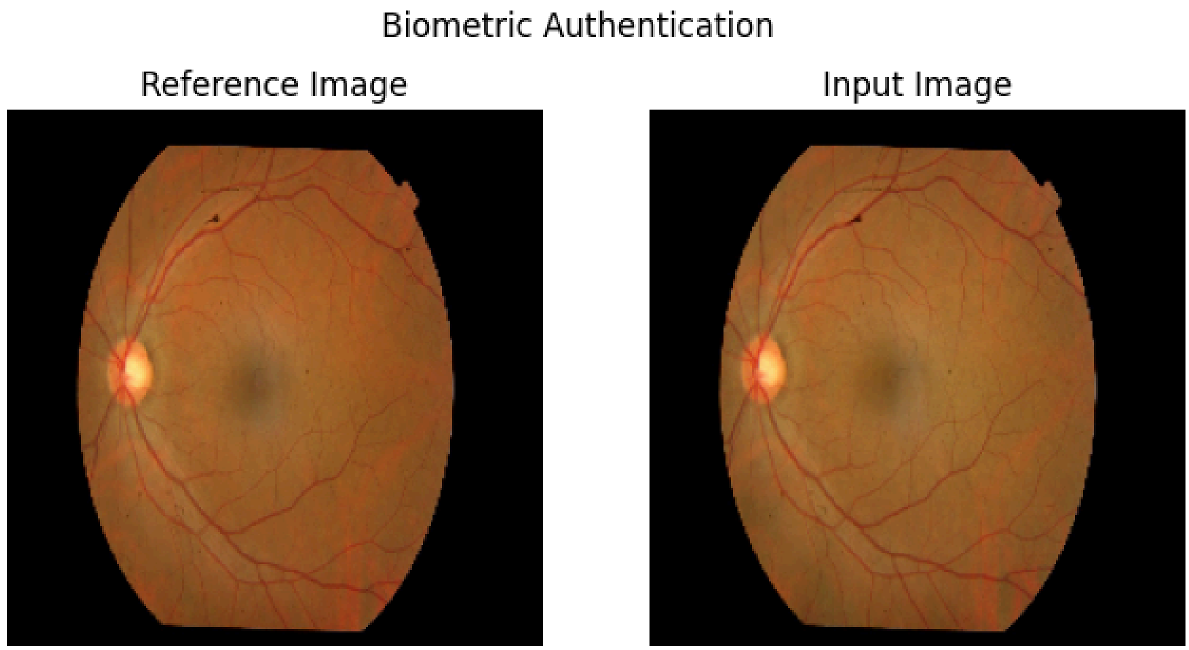
```
Genuine Authentication Example:
```



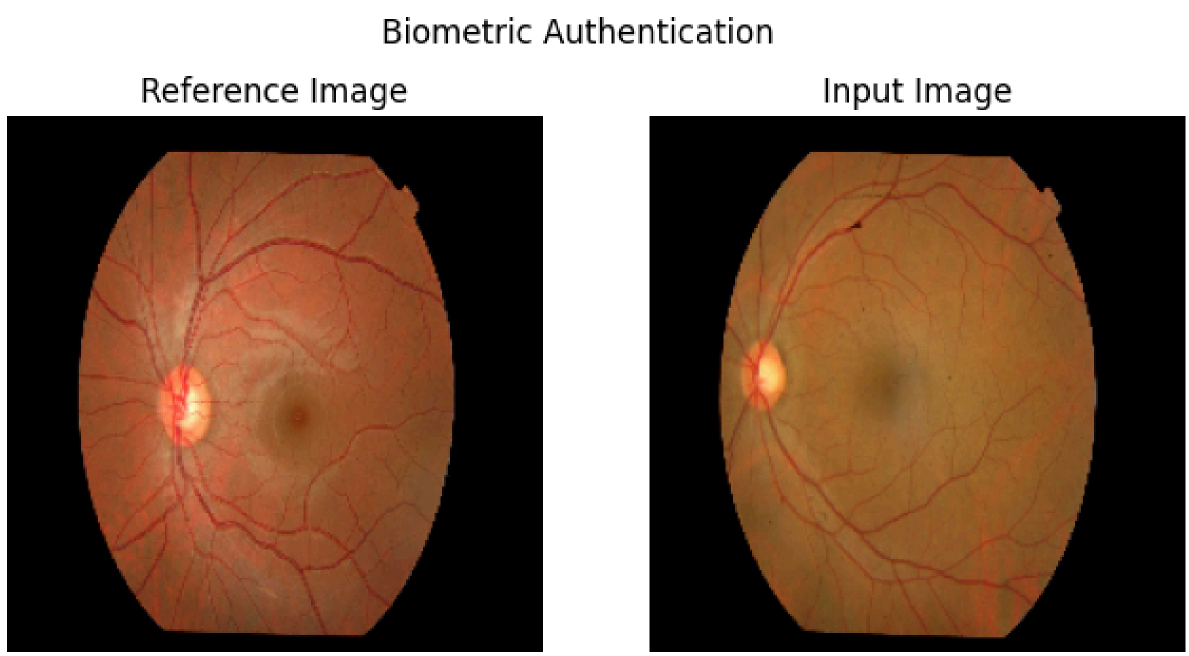

```
User ID: 1
Reference image: RIDB_FORMATED\Person_1\IM000001_1.JPG
Input image: RIDB_FORMATED\Person_1\IM000002_1.JPG
Similarity score: 0.8967
Authentication Successful: The user is verified.

Impostor Authentication Example:
```



```
User ID: 1
Reference image: RIDB_FORMATED\Person_1\IM000001_1.JPG
Input image: RIDB_FORMATED\Person_2\IM000001_2.JPG
Similarity score: 0.3134
Authentication Failed: The user is not verified.

Impostor Authentication Example:
```




```
User ID: 20
Reference image: RIDB_FORMATED\Person_20\IM000001_20.JPG
Input image: RIDB_FORMATED\Person_20\IM000005_20.JPG
Similarity score: 0.9829
Authentication Successful: The user is verified.

Impostor Authentication Example:
```



```
User ID: 10
Reference image: RIDB_FORMATED\Person_10\IM000001_10.JPG
Input image: RIDB_FORMATED\Person_20\IM000005_20.JPG
Similarity score: 0.0120
Authentication Failed: The user is not verified.
```