High level design

[Project]

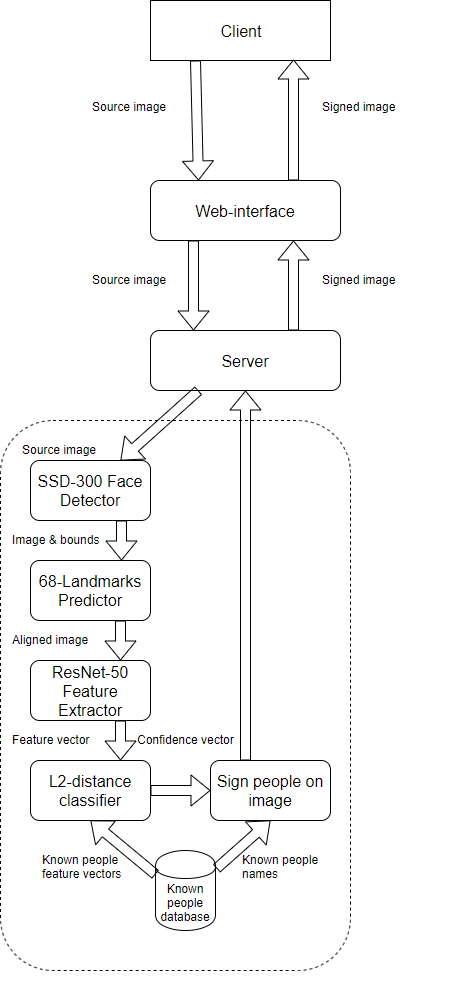
**Technology stack:**

1) Server part - Python (+ TF + Keras libraries)

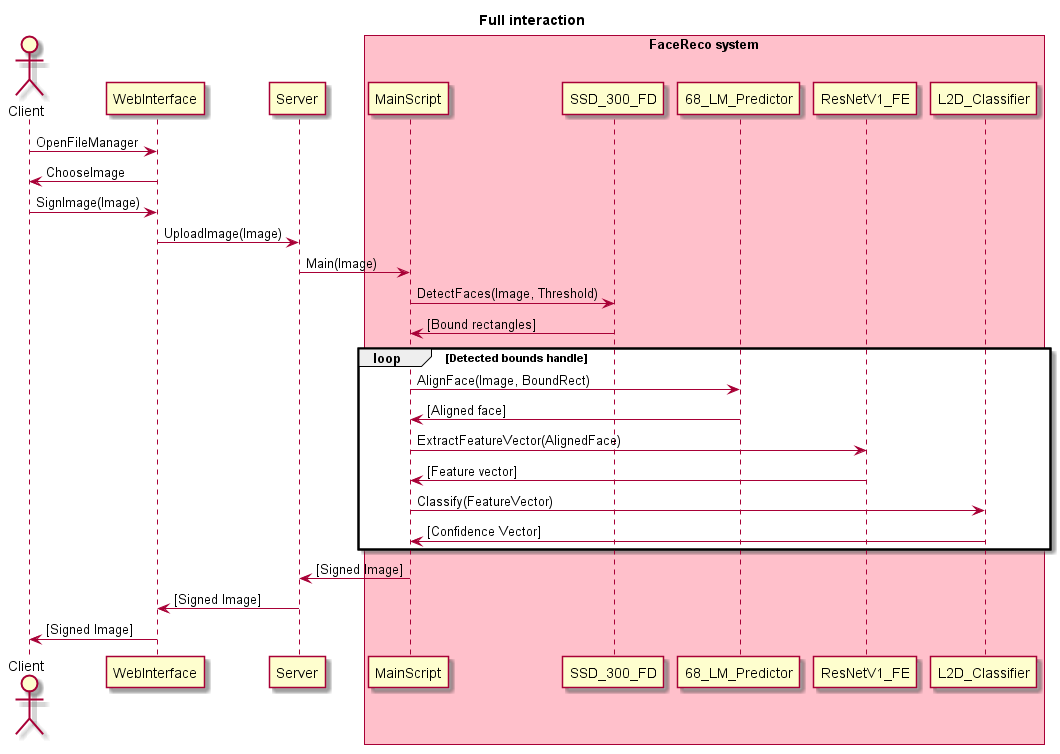
2) Web - Flask Microframework, JavaScript

3) Auxiliary – Github, Trello

**Data flow diagram:**



**Sequence diagram**



**Original code parts**

# Classes initialization and source feature vectors loading

Classes = [`MXG`, `Sanaken`, `Zofinka`, `Toalk`, `Zissxzirsziiss`, `kiasummer`]

Known\_face\_encodes = [

Np.loadtxt(`persons/MXG/fv.txt`),

Np.loadtxt(`persons/Sanaken/fv.txt`),

Np.loadtxt(`persons/ Zofinka /fv.txt`),

Np.loadtxt(`persons/ Toalk /fv.txt`),

Np.loadtxt(`persons/ Zissxzirsziiss /fv.txt`),

Np.loadtxt(`persons/M kiasummer XG/fv.txt`),

]

Fd = FaceDetector()

faceboxes = fd.get\_faceboxes(image)

# Converting received faceboxes matrix into the dlib.rectangle (Adapter)

for I in range(len(faceboxes)):

face\_rect = dlib.rectangle(

faceboxes[i][0], faceboxes[i][1], faceboxes[i][2], faceboxes[i][3])

# Print recognition result

ind = compare\_faces(known\_face\_encodes, feature\_vector)

if (ind != -1):

cv2.putText(image, classes[ind], (faceboxes[i][0], faceboxes[i][1] - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

else:

cv2.putText(image, “unknown”, (faceboxes[i][0], faceboxes[i][1] - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)

cv2.rectangle(image, (faceboxes[i][0], faceboxes[i][1]), (faceboxes[i][2], faceboxes[i][3]), (255, 0, 0))

cv2.imshow(“Preview”, image)

cv2.waitKey(0)

cv2.destroyAllWindows

**Original auxiliary methods**

# L2 Distance implementation

def compare\_2\_faces(known\_face\_encoding, face\_encoding\_to\_check):

return (np.linalg.norm(known\_face\_encoding - face\_encoding\_to\_check))

# Classifier powered by L2-Distance method above

def compare\_faces(known\_faces, face\_to\_check):

tolerance=0.6

ind, length = -1, sys.float\_info.max

for i in range(len(known\_faces)):

for face in known\_faces[i]:

cur = compare\_2\_faces(face, face\_to\_check)

if (cur < length):

length = cur

ind = i

if (length <= tolerance):

return ind

else:

return -1

# Marks drawing

def draw\_marks(image, marks, color=(0, 255, 0)):

"""Draw mark points on image"""

for mark in marks:

cv2.circle(image, (int(mark.x), int(

mark.y)), 1, color, -1, cv2.LINE\_AA)

# Get faceboxes method (Adapter cutting detected bounds with a small confidence)

def get\_faceboxes(self, image, threshold=0.5):

rows, cols, \_ = image.shape

confidences = []

faceboxes = []

self.face\_net.setInput(cv2.dnn.blobFromImage(

image, 1.0, (300, 300), (104.0, 177.0, 123.0), False, False))

detections = self.face\_net.forward()

for result in detections[0, 0, :, :]:

confidence = result[2]

if confidence > threshold:

x\_left\_bottom = int(result[3] \* cols)

y\_left\_bottom = int(result[4] \* rows)

x\_right\_top = int(result[5] \* cols)

y\_right\_top = int(result[6] \* rows)

confidences.append(confidence)

#dlib rectangle for alignment

faceboxes.append(

[x\_left\_bottom, y\_left\_bottom, x\_right\_top, y\_right\_top])

self.detection\_result = [faceboxes, confidences]

return confidences, faceboxes