MUS-Project Documentation

Gerald Spenlingwimmer

May 26, 2019

Contents

1 Idea 2
2 Expectation 2
3 The Microsoft Face API 2
4 Twilio Setup 3
4.1 Setup Freetext Messages 3
4.2 The API Key 6
4.3 The C# Code 6
5 Face API Client 7
5.1 Setup Azure Face 7
5.2 Lifecycle of the Face API 9
5.3 Code 10
5.4 Face Client 10
5.5 Access the Training Status 11
6 WPF Frontend 13
6.1 Step By Step 13
6.2 Code 13
6.3 Speech Output 14
7 Packages 14
8 Results, Findings and Discussion 14

1 Idea

When thinking about smart cities or interesting technologies with a broad field of applications in future technologies facial recognition and especially facial identification is an interesting thing. The basic idea is to simulate a door with facial identification. For this purpose a small app has been built to show the general concepts which could than be applied to many use cases. The Kinect is used as a camera to monitor e.g. a door. Then the doorbell is rang and a photo of the person waiting at the door is taken. This photo is than compared with a network of know people. If the person is a known one a message is sent to the e.g. the tenant. This is something quite interesting for smart cities since Microsoft promises that they do not store the images of the faces sent to their AI it should be just fine with data protection guidelines and especially everywhere where you need to make an appointement beforehand facial identification could be used to unlock doors. Of course the facial identification can be tricked with photos and videos and therefore should only be used when no security problem is caused by using this mechanism for the identification of people.

2 Expectation

A simple application capable of simulating a door with facial identification. A small app that shows the live camera feed of the Kinect. Another app that can be trained with images so that it can distinguish people from one another. A messanger app wich can send Whatsapp messages. Some speech output on the PC telling you the name of the person. The app should then be capable to do all together. As soon as the dorbell is rang an image is taken. The image shall then be analysed and the name of the person should then be returned if the person is a known person. The name is sent as Whatsapp message and additionally spoken by the computer.

3 The Microsoft Face API

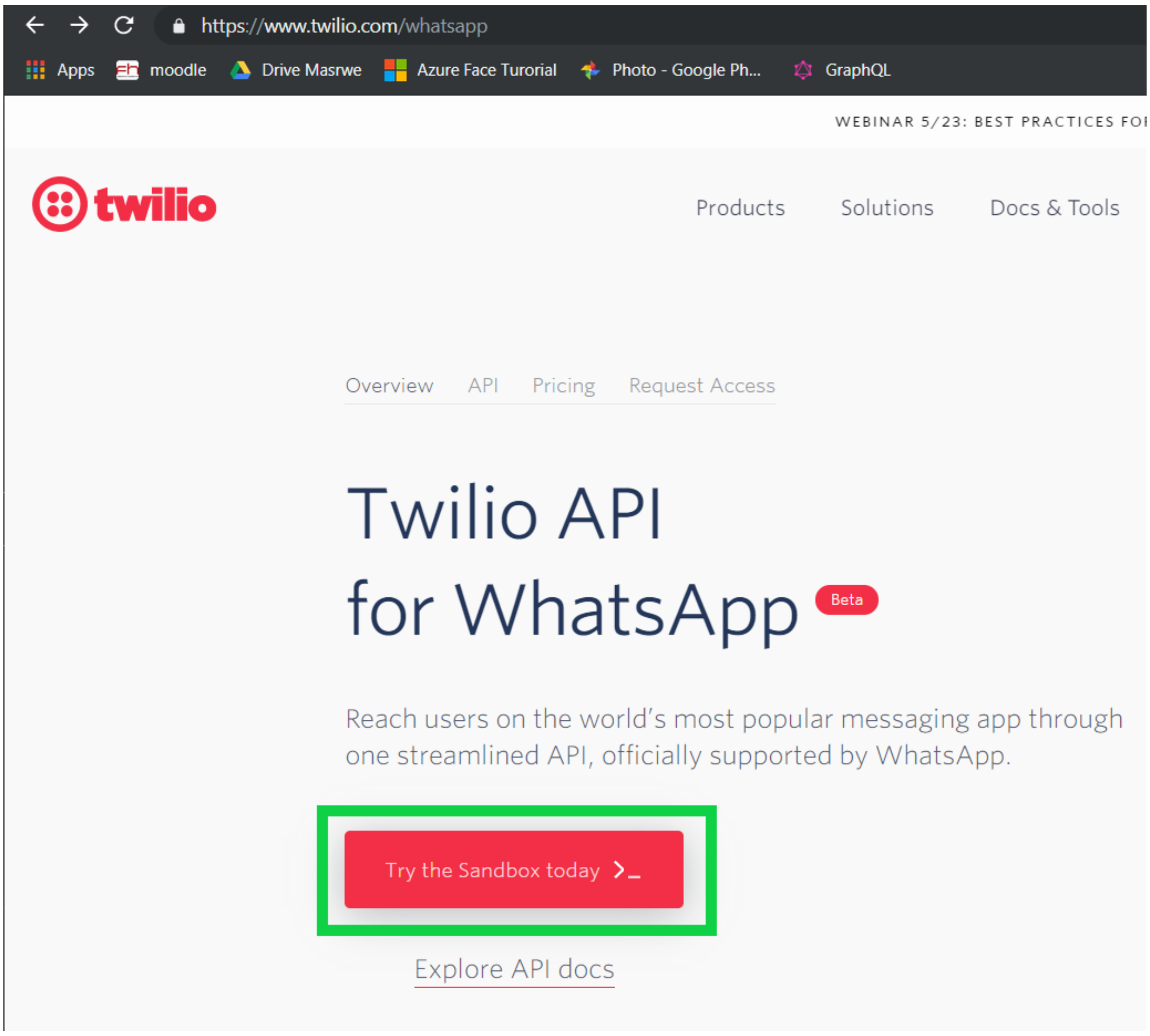
The Faca API provides many interesting features one can play around with. The data returned by the API for a single image looks like in listing 13 located in Appendix B. There is a lot of interesting stuff returned like gender, age, haircolor, makeup, wears glasses, does smile, face landmarks, emotion and much more. This is very nice but does not have an impact on this project since facts like the gender or age are quite irrelevant for the facial identification done in this project. Another thing offered by the Face API is the comparison of two images of faces. This comes closer to the wanted functionality since now the only thing needed would be a database of images of the people that should be recognized and now the image would just need to be compared with the database and the person could be identified. Of course this is still a low level approach and the Face API offers something even better facial identification. The main steps to do are to create a database of images. These images are then put into a model. The model is then trained with these images. Now a simple request can be used to identify a person from a simple image which is exactly what is needed for this project.

4 Twilio Setup

One feature of the app is to send Whatsapp messages to a registered client. Twilio provides a mechanism to send Whatsapp messanges to registered clients for free with some restrictions. Freetext messages are only allowed after a client replied to one of the standard pattern messanges. Then freetext messages can be sent for the next 24 hours.

4.1 Setup Freetext Messages

- Navigate to Twilio for Whatsapp.
- Create an Account (FREE)
- Navigate to the Sandbox
- Do Tutorial
- Retrieve API-Key
- Do C# implementation

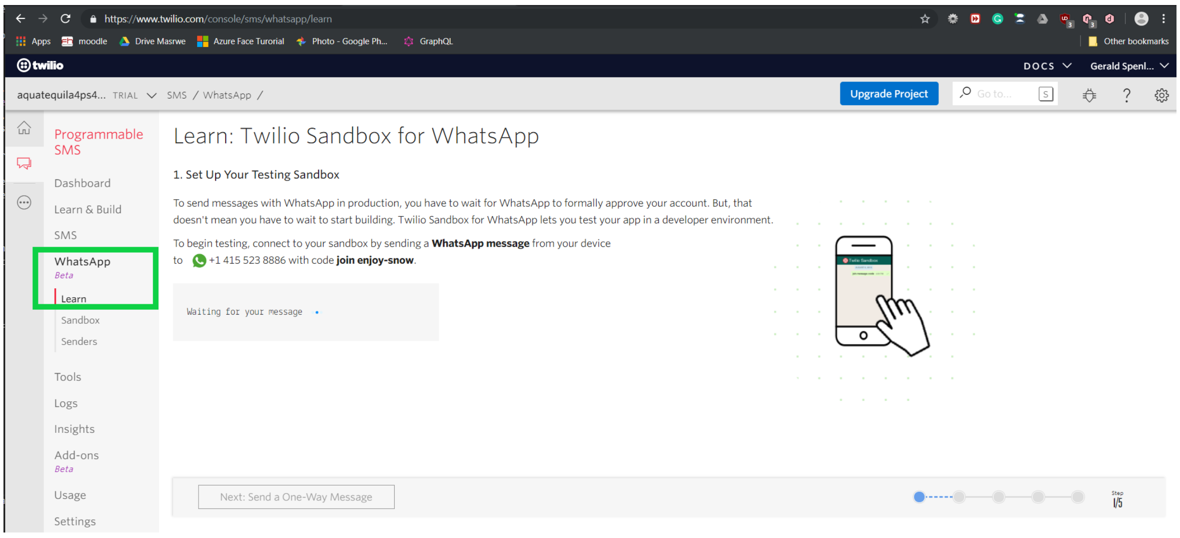


Twilio API
for WhatsApp

Reach users on the world's most popular messaging app through one streamlined API, officially supported by WhatsApp.

Figure 1: Navigate to the Sandbox

Figure 1 shows the button to access the Sandbox marked with a green border. Then make sure that Whatsapp-Learn is selected, marked with a green rectangel in Figure 2.



Learn: Twilio Sandbox for WhatsApp

1. Set Up Your Testing Sandbox

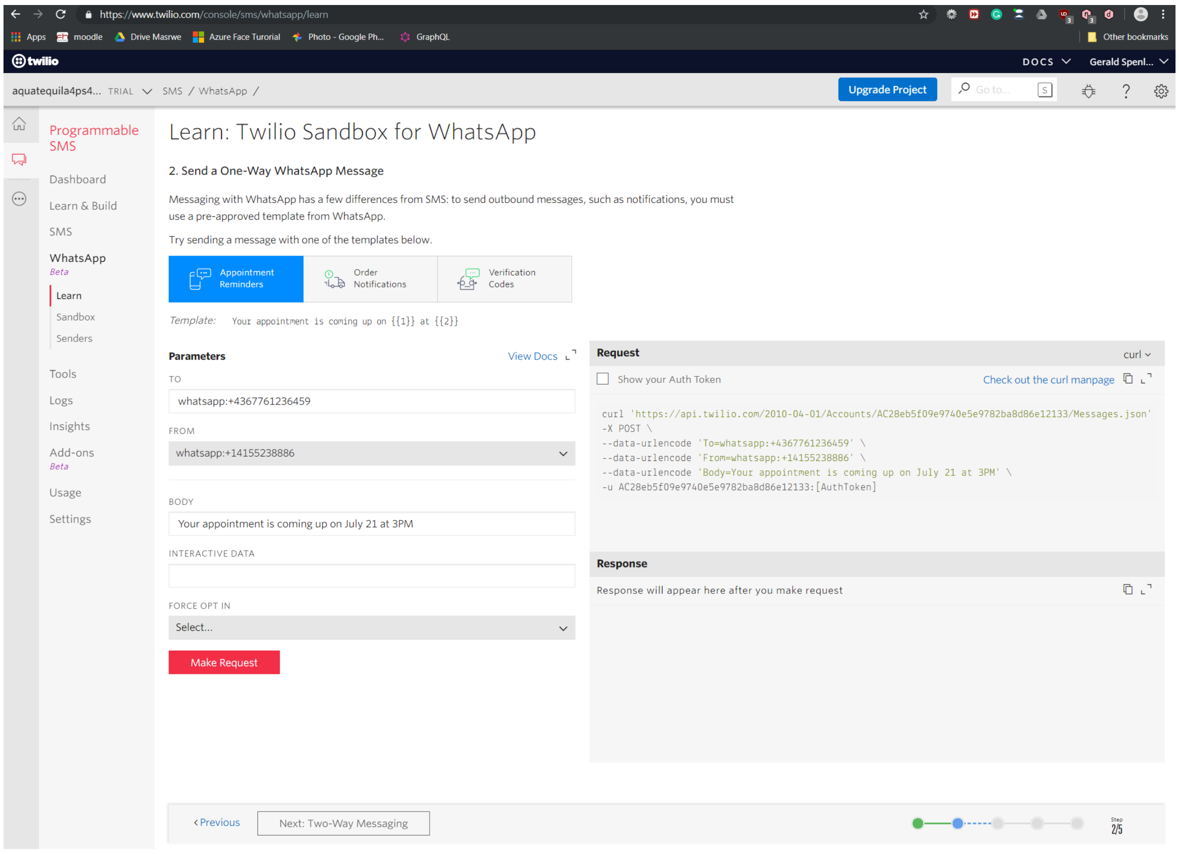
To send messages with WhatsApp in production, you have to wait for WhatsApp to formally approve your account. But, that doesn't mean you have to wait to start building. Twilio Sandbox for WhatsApp lets you test your app in a developer environment.

To begin testing, connect to your sandbox by sending a **WhatsApp message** from your device to +1 415 523 8886 with code **join enjoy-snow**.

```
Waiting for your message
```

Figure 2: Register for Messages

Now follow the instructions to complete the tutorial. First a message from the phone you wnat to receive the messages from the API needs to register by sending the message ”join enjoy-snow” to the number provided by Twilio. Afterwards, select a template and send the message to your phone shown in Figure 3.



Learn: Twilio Sandbox for WhatsApp

2. Send a One-Way WhatsApp Message

Messaging with WhatsApp has a few differences from SMS: to send outbound messages, such as notifications, you must use a pre-approved template from WhatsApp.

Try sending a message with one of the templates below.

Template: `Your appointment is coming up on {{1}} at {{2}}`

Parameters

TO	whatsapp:+4367761236459
FROM	whatsapp:+14155238886
BODY	Your appointment is coming up on July 21 at 3PM
INTERACTIVE DATA	
FORCE OPT IN	Select...

Request

```
curl 'https://api.twilio.com/2010-04-01/Accounts/AC28eb5f09e9740e5e9782ba8d86e12133/Messages.json'
-X POST \
--data-urlencode 'To=whatsapp:+4367761236459' \
--data-urlencode 'From=whatsapp:+14155238886' \
--data-urlencode 'Body=Your appointment is coming up on July 21 at 3PM' \
-u AC28eb5f09e9740e5e9782ba8d86e12133:[AuthToken]
```

Response

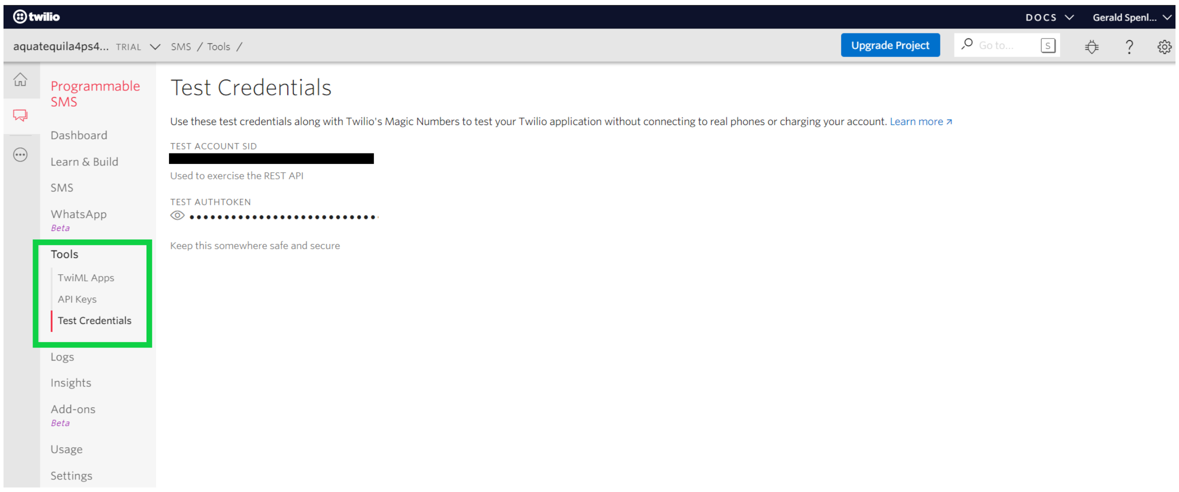
Response will appear here after you make request

Figure 3: Sending a Tempalte Message

Now reply on your phone to the previous message and you are all set for freetext communication for the next 24 hours.

4.2 The API Key

To retreive your key and token navigate to Tools-Test Credentials shown in Figure 4



Test Credentials

Use these test credentials along with Twilio's Magic Numbers to test your Twilio application without connecting to real phones or charging your account. Learn more

TEST ACCOUNT SID
[redacted]
Used to exercise the REST API

TEST AUTHTOKEN
••••••••••••••••••••••••••••

Keep this somewhere safe and secure

Figure 4: Get Your Key and Token

4.3 The C# Code

```
public static class WhatsappSender
    {
        private static string configPath = @"C:/Users/
    GeraldSpenlingwimmer/Desktop/config/twilioconfig.json";

        private class Config
        {
            public string Sid { get; set; }
            public string AuthToken { get; set; }
            public string TwilioPhone { get; set; }
            public string TargetPhone { get; set; }

        }
        private static string twilioPhone = "";
        private static string targetPhone = "";

        static WhatsappSender()
        {
            using (StreamReader r = new StreamReader(configPath))
            {
                string jsonString = r.ReadToEnd();
                Config config = JsonConvert.DeserializeObject<Config>(
    jsonString);
                TwilioClient.Init(config.Sid, config.AuthToken);
                twilioPhone = config.TwilioPhone;
                targetPhone = config.TargetPhone;
            }
        }

        public static void Send(string messageBody)
        {
            Task.Factory.StartNew(() =>
            {
                var message = MessageResource.Create(
                from: new Twilio.Types.PhoneNumber(twilioPhone),
                body: messageBody,
                to: new Twilio.Types.PhoneNumber(targetPhone)
                );
            });
        }
    }
```

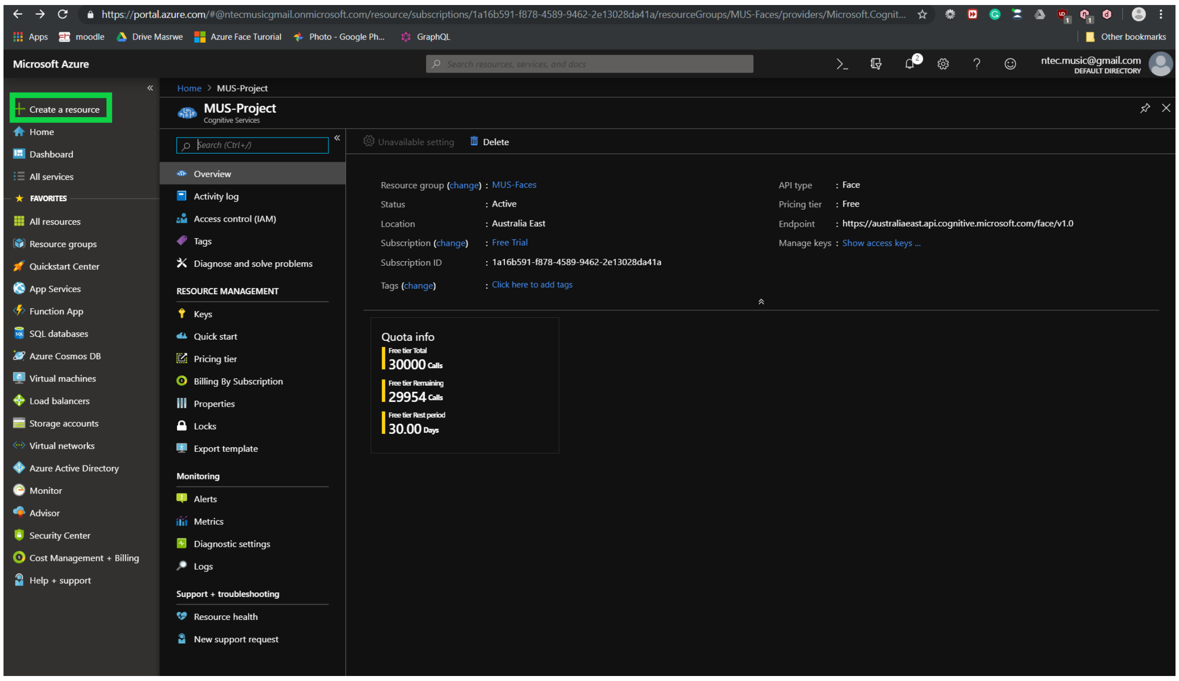
Listing 1: Whatsapp Client

5 Face API Client

- Create Microsoft Account (if you have none)
- Create [Free] Azure Account
- Create Face API Project in Azure
- Get Subscription Key and Endpoint
- Start C# Coding

5.1 Setup Azure Face

Navigate to Azure and select ”create new resource” shown in Figure 5.



https://portal.azure.com/#@ntecmusicgmail.onmicrosoft.com/resource/subscriptions/1a16b591-f878-4589-9462-2e13028da41a/resourceGroups/MUS-Faces/providers/Microsoft.Cognit...

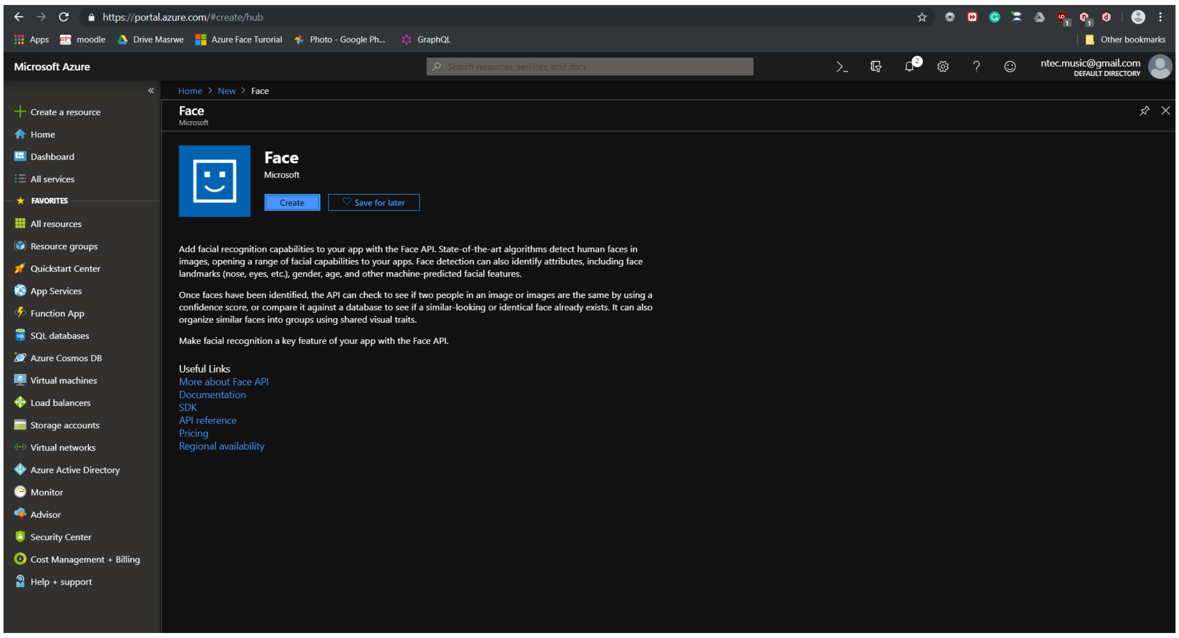
Resource group (change) : MUS-Faces
Status : Active
Location : Australia East
Subscription (change) : Free Trial
Subscription ID : 1a16b591-f878-4589-9462-2e13028da41a
Tags (change) : Click here to add tags

API type : Face
Pricing tier : Free
Endpoint : https://australiaeast.api.cognitive.microsoft.com/face/v1.0
Manage keys : Show access keys ...

Quota info
Free tier Total 30000 Calls
Free tier Remaining 29954 Calls
Free tier Rest period 30.00 Days

Figure 5: Create a Resource

In the search field enter ”Face” and hit enter. Then select ”Create” shown ikn Figure 6.



https://portal.azure.com/#create/hub

Face
Microsoft

Add facial recognition capabilities to your app with the Face API. State-of-the-art algorithms detect human faces in images, opening a range of facial capabilities to your apps. Face detection can also identify attributes, including face landmarks (nose, eyes, etc.), gender, age, and other machine-predicted facial features.

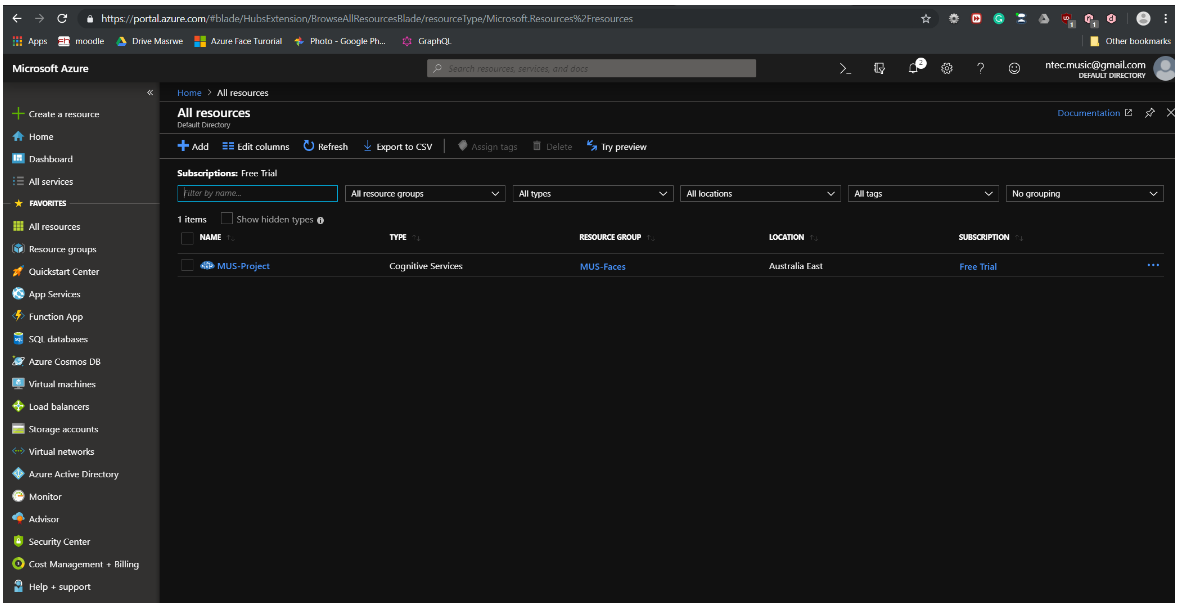
Once faces have been identified, the API can check to see if two people in an image or images are the same by using a confidence score, or compare it against a database to see if a similar-looking or identical face already exists. It can also organize similar faces into groups using shared visual traits.

Make facial recognition a key feature of your app with the Face API.

Useful Links
More about Face API
Documentation
SDK
API reference
Pricing
Regional availability

Figure 6: Create Face Resource

Enter the information about the project and hit create again. Now navigate to ”all resources” like in Figure 7 and click on the name of your project.



Subscriptions: Free Trial

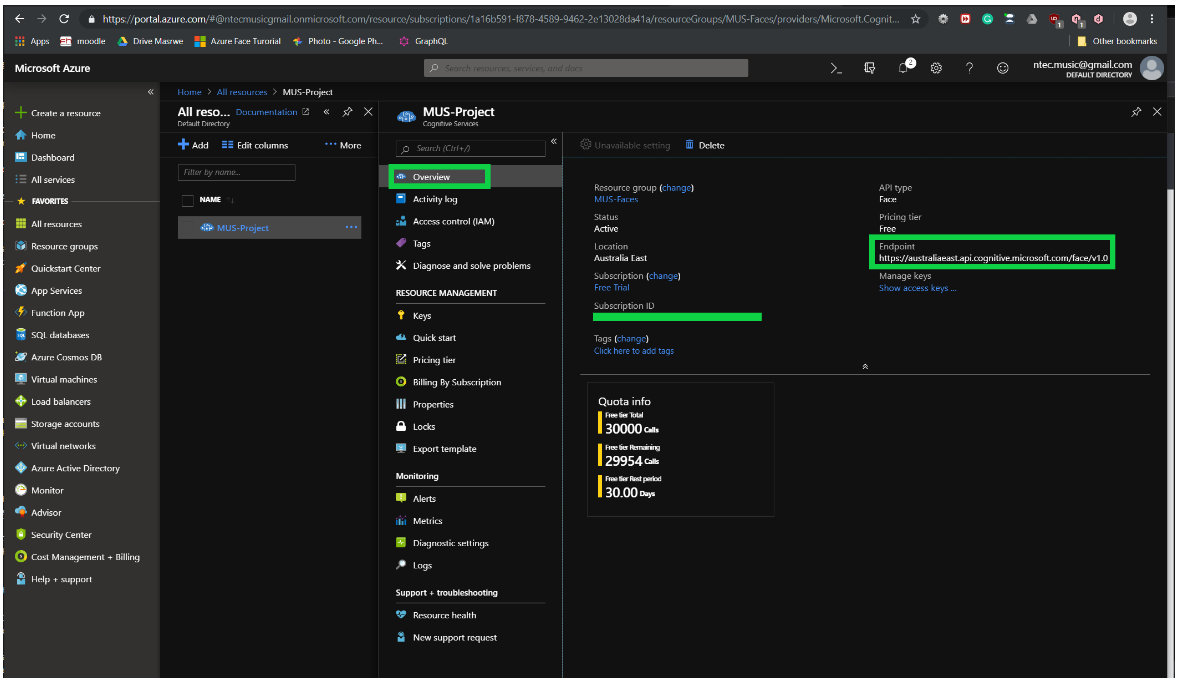
All resource groups | All types | All locations | All tags | No grouping

1 items

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
MUS-Project	Cognitive Services	MUS-Faces	Australia East	Free Trial

Figure 7: Find Your Key and Endpoint

Navigate to overview shown in Figure 8 where you will find your access point and your subscription key.



Resource group (change)
MUS-Faces

Status
Active

Location
Australia East

Subscription (change)
Free Trial

Subscription ID
[redacted]

Tags (change)
Click here to add tags

API type
Face

Pricing tier
Free

Endpoint
https://australiaeast.api.cognitive.microsoft.com/face/v1.0

Manage keys
Show access keys ...

Quota info

Free tier Total
30000 Calls

Free tier Remaining
29954 Calls

Free tier Rest period
30.00 Days

Figure 8: Get Your Key and Endpoint

5.2 Lifecycle of the Face API

5.2.1 Explanation of the Important Properties for this App

- **Person-Group:** A group of persons e.g. ”Family”, ”Friends” or ”Colleagues”

- **Person:** A person in a group to store image data to e.g. Myself in the group ”Family” to store x images of myself for facial authentication.

5.2.2 Initialization

- Create a Person-Group
- Create Persons in the Person-Group
- Add Images to the Persons
- Train the Model

5.2.3 Request

- Identify Person From Image

5.2.4 When the Model Needs to be Changed

- Delete Person-Group
- Re-Initialize With Changes

5.3 Code

5.4 Face Client

```
    public MyFace()
    {
        using (StreamReader r = new StreamReader(configPath))
        {
            string jsonString = r.ReadToEnd();
            Config config = JsonConvert.DeserializeObject<Config>(
jsonString);
            faceClient = new FaceClient(
                new ApiKeyServiceClientCredentials(config.SubscriptionKey
),
                new System.Net.Http.DelegatingHandler[] { });
            faceClient.Endpoint = config.FaceEndpoint;
        }
    }
```

Listing 2: Face Client

5.4.1 Create a Person-Group

```
    await faceClient.PersonGroup.CreateAsync(”myfriends”, ”My Friends”);
```

Listing 3: Create a Person Group

5.4.2 Create a Person in a Group

```
var friend = await faceClient.PersonGroupPerson.CreateAsync("myfriends", "NAME_OF_PERSON");
```

Listing 4: Create a Person in a Group

5.4.3 Add Images to a Person

```
public async void AddImageToPersonInGroup(string group, Guid personId, string directory, string personName)
{
    foreach (string imagePath in Directory.GetFiles(directory, "*.jpg"))
    {
        using (Stream s = File.OpenRead(imagePath))
        {
            // Detect faces in the image and add to personId
            await faceClient.PersonGroupPerson.AddFaceFromStreamAsync(friendGroup, personId, s);
        }
    }
    Console.WriteLine($"Images for {personName} added");
}
```

Listing 5: Add Images to a Person

5.4.4 Train the Model

```
await faceClient.PersonGroup.TrainAsync("myfriends");
```

Listing 6: Train the Model

5.5 Access the Training Status

```
public async void TrainingStatus()
{
    TrainingStatus trainingStatus = null;
    while (true)
    {
        trainingStatus = await faceClient.PersonGroup.GetTrainingStatusAsync(friendGroup);
        if (trainingStatus.Status != TrainingStatusType.Running)
        {
            Console.WriteLine("Train model finished");
            break;
        }

        await Task.Delay(10000);
    }
}
```

Listing 7: Training Status

5.5.1 Identify a Person

```
        public async void Identify(string imagePath)
        {
            try
            {
                using (Stream s = File.OpenRead(imagePath))
                {
                    var faces = await faceClient.Face.DetectWithStreamAsync(s);
                    var faceIds = faces.Where(x => x.FaceId.HasValue).Select(face => face.FaceId.Value).ToArray();

                    var results = await faceClient.Face.IdentifyAsync(faceIds, friendGroup);

                    foreach (var result in results)
                    {
                        if (result.Candidates.ToArray().Length == 0) // no match found
                        {
                            string message = "Hello, An unknown person is waiting at your door";
                            WhatsappSender.Send(message);
                            SpeechProcessor.SpeechProcessorGS.Speak(message);
                        }
                        else // match(es) found
                        {
                            // Get top 1 among all candidates returned
                            var bestMatch = result.Candidates[0];
                            var person = await faceClient.PersonGroupPerson.GetAsync(friendGroup, bestMatch.PersonId);
                            var message = $"{person.Name} is waiting at your door with {(int)(bestMatch.Confidence * 100)}% confidence";
                            WhatsappSender.Send(message);
                            SpeechProcessor.SpeechProcessorGS.Speak(message);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
```

Listing 8: Person Authentication

5.5.2 Reset the Model

```
    await faceClient.PersonGroup.DeleteAsync(friendGroup);
```

Listing 9: Reset the Model

6 WPF Frontend

6.1 Step By Step

- Install Microsoft Kinect SDK 2.0
- Open the Kinect SDK Browser(type "sdk browser" in your windows search bar)
- Download the "ColorBasics-WPF" and open the app
- Adjust the code

The "ColorBasics-WPF" example alredy does all the necessary Kinect stuff like initialization and it alos provides a livestream from the Kinect camera and a button to make a screenshot. All that needs to be done is the linking between the Kincet and the Face API client from Section 5. Therefore, only the *ScreenshotButton_Click* mehthod needs to be changed. As soon as the image has been saved the path is passed to the identify method of the face client and the face client and the frontend are linked.

6.2 Code

```
    private void ScreenshotButton_Click(object sender, RoutedEventArgs e)
    {
        if (this.colorBitmap != null)
        {
            // create a png bitmap encoder which knows how to save a .png file
            BitmapEncoder encoder = new PngBitmapEncoder();

            // create frame from the writable bitmap and add to encoder
            encoder.Frames.Add(BitmapFrame.Create(this.colorBitmap));

            string time = System.DateTime.Now.ToString("hh'-'mm'-'ss", CultureInfo.CurrentUICulture.DateTimeFormat);

            string myPhotos = "C:/Users/GeraldSpenlingwimmer/Desktop/testImages/door";

            string path = Path.Combine(myPhotos, "1.png");

            // write the new file to disk
            try
            {
                // FileStream is IDisposable
                using (FileStream fs = new FileStream(path, FileMode.Create))
                {
                    encoder.Save(fs);
                }

                this.StatusText = string.Format(Properties.Resources.SavedScreenshotStatusTextFormat, path);

                myFace.Identify(path);
            }
            catch (IOException)
            {
                this.StatusText = string.Format(Properties.Resources.
    FailedScreenshotStatusTextFormat, path);
            }
        }
    }
```

Listing 10: Screenshot Button Click

6.3 Speech Output

```
    private static SpeechSynthesizer synth = new SpeechSynthesizer();

    public static void Speak(string message)
    {
        // Configure the audio output.
        synth.SetOutputToDefaultAudioDevice();

        // Speak a string.
        synth.Speak(message);
    }
```

Listing 11: Speech Output

7 Packages

```
    <?xml version="1.0" encoding="utf-8"?>
    <packages>
      <package id="Microsoft.Azure.CognitiveServices.Vision.Face" version
    ="2.4.0-preview" targetFramework="net461" />
      <package id="Microsoft.IdentityModel.Logging" version="1.1.2"
    targetFramework="net461" />
      <package id="Microsoft.IdentityModel.Tokens" version="5.1.2"
    targetFramework="net461" />
      <package id="Microsoft.Rest.ClientRuntime" version="2.3.19"
    targetFramework="net461" />
      <package id="Microsoft.Rest.ClientRuntime.Azure" version="3.3.19"
    targetFramework="net461" />
      <package id="Newtonsoft.Json" version="10.0.3" targetFramework="
    net461" />
      <package id="System.IdentityModel.Tokens.Jwt" version="5.1.2"
    targetFramework="net461" />
      <package id="Twilio" version="5.28.3" targetFramework="net461" />
    </packages>
```

Listing 12: packages.config

8 Results, Findings and Discussion

- A WPF application which simulates a doorbell with live camera picture
- A Face API client to authenticate a person

- Face API mehthods to setup and train a model
- A Whatsapp message sender using Twilio
- Simple speech output on the PC

The general expectations were met since all requirements were fullfilled. The big downside with the current app is the speed since it takes about 30 seconds until the Whatsapp Message is received and thats a quite long for the targeted usecase. If it is e.g. used to open doors for employees they will probably be annoyed if the process takes 30-60 seconds. If the field of application is a door in a flat you probably are at the door in less than 30 seconds and hence it's not really useful since it's too slow. For actual authentication it is not safe enough since a picture or a video could be used to trick the system.
The upside is definitely the development process in my case since I did start from zero and it took far less time than I had guessed to devlop such an application since the Kinect samples are already quite good and perfectly separated for the different camera features you want to use. The second big factor is the Face API, it is really easy to use although the documentation is not exactly great and the tutorials from Microsoft seem to feature a different/older version of the SDK than I used the whole process is really easy to understand. Twilio is also really easy to use but the setup takes some time until everything is running.
The Kinect SDK also provides facial data but then the authentication process is not included as far as I have seen. This means, if the app were to be made with the Kinect SDK some algorithm would be necessary to compare the face data with a database which is probably quite hard and for sure far worse than using Azure Face for the task and hence the Kinect is acutally not necessary for this app. Every input from a camera would be enough e.g. a smartphone camera or a webcam would be enough. The conclusion is that the development of such an app is quite fast and the stuff provided by Microsoft is easy to use but all together the app is slow when it comes to performance.

Appendix A - Resources

- Microsoft Face API
- Microsoft Kinect SDK 2.0
- Microsoft Visual Studio
- Microsoft Azure Faces (Account Needed)
- Twilio (Whatsapp Messages)

Appendix B - Face API Dataset

```
JSON : [
{
    "faceId": "78633857-941c-401e-8f53-1fd66875d29f",
    "faceRectangle": {
    "top": 128,
    "left": 459,
    "width": 224,
    "height": 224
    },
    "faceAttributes": {
    "hair": {
        "bald": 0.07,
        "invisible": false ,
        "hairColor": [
        {
            "color": "brown",
            "confidence": 0.98
        },
        {
            "color": "blond",
            "confidence": 0.53
        },
        {
            "color": "black",
            "confidence": 0.41
        },
        {
            "color": "red",
            "confidence": 0.35
        },
        {
            "color": "gray",
            "confidence": 0.21
        },
        {
            "color": "other",
            "confidence": 0.14
        }
        ]
    },
    "smile": 1.0,
    "headPose": {
        "pitch": -13.2,
        "roll": -11.9,
        "yaw": 5.0
    },
    "gender": "female",
    "age": 24.0,
    "facialHair": {
        "moustache": 0.0,
        "beard": 0.0,
        "sideburns": 0.0
    },
    "glasses": "ReadingGlasses",
    "makeup": {
        "eyeMakeup": true ,
        "lipMakeup": false
    },
    "emotion": {
        "anger": 0.0,
        "contempt": 0.0,
        "disgust": 0.0,
        "fear": 0.0,
        "happiness": 1.0,
        "neutral": 0.0,
        "sadness": 0.0,
        "surprise": 0.0
    },
    "occlusion": {
        "foreheadOccluded": false,
        "eyeOccluded": false,
        "mouthOccluded": false
    },
    "accessories": [],
    "blur": {
        "blurLevel": "low",
        "value": 0.0
    },
    "exposure": {
        "exposureLevel": "goodExposure",
        "value": 0.48
    },
    "noise": {
        "noiseLevel": "low",
        "value": 0.0
    }
    },
    "faceLandmarks": {
    "pupilLeft": {
        "x": 504.8,
        "y": 206.8
    },
    "pupilRight": {
        "x": 602.5,
        "y": 178.4
    },
    "noseTip": {
        "x": 593.5,
        "y": 247.3
    },
    "mouthLeft": {
        "x": 529.8,
        "y": 300.5
    },
    "mouthRight": {
        "x": 626.0,
        "y": 277.3
    },
    "eyebrowLeftOuter": {
        "x": 461.0,
        "y": 186.8
    },
    "eyebrowLeftInner": {
        "x": 541.9,
        "y": 178.9
    },
    "eyeLeftOuter": {
        "x": 490.9,
        "y": 209.0
    },
    "eyeLeftTop": {
        "x": 509.1,
        "y": 199.5
    },
    "eyeLeftBottom": {
        "x": 509.3,
        "y": 213.9
    },
    "eyeLeftInner": {
        "x": 529.0,
        "y": 205.0
    },
    "eyebrowRightInner": {
        "x": 579.2,
        "y": 169.2
    },
    "eyebrowRightOuter": {
        "x": 633.0,
        "y": 136.4
    },
    "eyeRightInner": {
        "x": 590.5,
        "y": 184.5
    },
    "eyeRightTop": {
        "x": 604.2,
        "y": 171.5
    },
    "eyeRightBottom": {
        "x": 608.4,
        "y": 184.0
    },
    "eyeRightOuter": {
        "x": 623.8,
        "y": 173.7
    },
    "noseRootLeft": {
        "x": 549.8,
        "y": 200.3
    },
    "noseRootRight": {
        "x": 580.7,
        "y": 192.3
    },
    "noseLeftAlarTop": {
        "x": 557.2,
        "y": 234.6
    },
    "noseRightAlarTop": {
        "x": 603.2,
        "y": 225.1
    },
    "noseLeftAlarOutTip": {
            "x": 545.4,
            "y": 255.5
        },
        "noseRightAlarOutTip": {
            "x": 615.9,
            "y": 239.5
        },
        "upperLipTop": {
            "x": 591.1,
            "y": 278.4
        },
        "upperLipBottom": {
            "x": 593.2,
            "y": 288.7
        },
        "underLipTop": {
            "x": 597.1,
            "y": 308.0
        },
        "underLipBottom": {
            "x": 600.3,
            "y": 324.8
        }
        }
}
]
```

Listing 13: Face API Face Properties