

Real Time Video Chat Application with Python, JavaScript and Twilio

1- Architecture

After some research regarding make it simple and easier to implemente, I decided to work with Python and Flask for server side and vanilla JavaScript, html and CSS. Although there's a range of technologies for this, the reason of my choice is that Twilio is a service in which is capable of doing the streamming service. In this project, a linux container with all components will be used an its components are listed bellow:

- Python 3.6 or above
- ngrok (it will connect localhost to streamming service).
- Twilio (service for video streamming)
- Chrome web browser

To achieve this, I have checked some cloud services that provides these technologies, and after read about Twilio, I choosed it from <https://github.com/miguelgrinberg/flask-twilio-video>

2 – Documentation

There's a simple way of running this. Requirements are:

- Twilio account to generate a api key in which allows the videochat authenticate and run (<https://www.twilio.com/try-twilio?promo=7fB3Je>).
- Api key which is generated for using twilio service (<https://www.twilio.com/console/project/api-keys>).
- A clone of repository (git clone <https://github.com/miguelgrinberg/flask-twilio-video>).

2.1 – Preparing the environment:

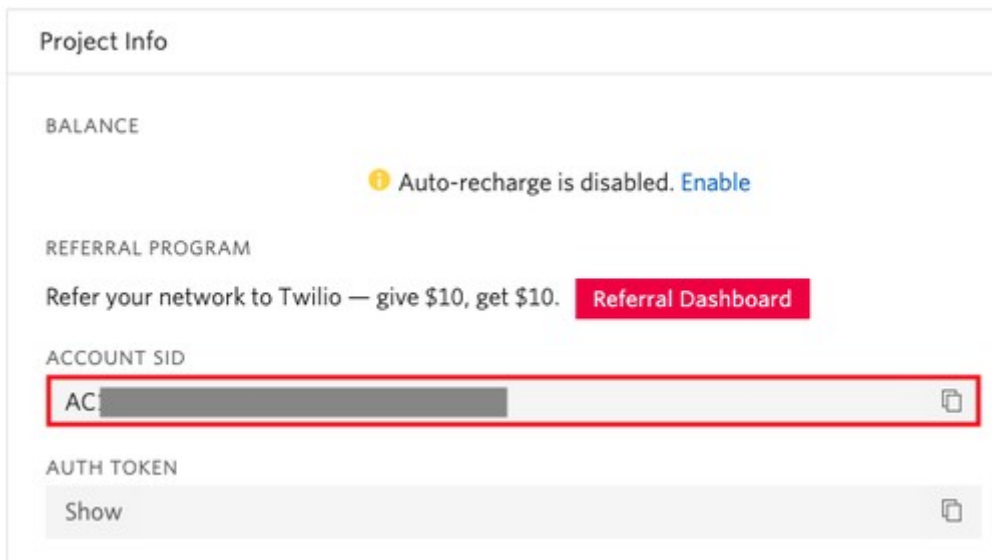
Let's begin by creating the directory where we will store our project files. Open a terminal window, find a suitable parent directory and then enter the following commands:

```
$ mkdir /wrtcvideo  
$ cd /wrtcvideo
```

Now we'll create more two sub-directories called **static** and **templates**:

```
$ mkdir static  
$ mkdir templates
```

2.2 – Getting Twilio account:

A screenshot of the Twilio Project Info page. The page has a light gray background with white sections. The 'Project Info' section is at the top. Below it, the 'BALANCE' section shows a message: 'Auto-recharge is disabled. Enable' with a yellow info icon. The 'REFERRAL PROGRAM' section says 'Refer your network to Twilio — give \$10, get \$10.' and has a red 'Referral Dashboard' button. The 'ACCOUNT SID' section shows a text input field with 'AC' followed by a masked area, and a red border highlights the entire field. A copy icon is on the right. The 'AUTH TOKEN' section has a 'Show' button and a copy icon.

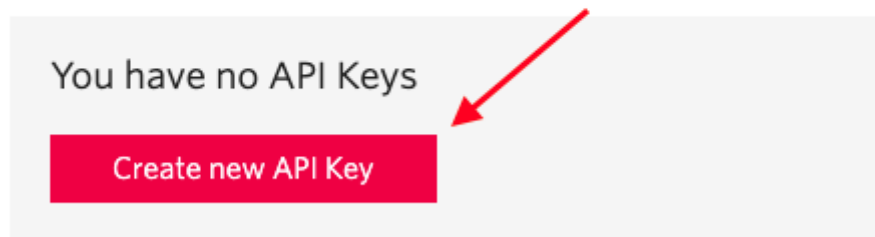
Copy and past Twilio account SID to using later. On project root folder (/wrtcvideo), create a .env file with this content below:

```
TWILIO_ACCOUNT_SID=<your-twilio-account-sid>
```

Also, it's important to mention that you'll need to copy the API Key, once, the programmable video service needs this to authentication. So, in your Twilio account, navigate to API Keys section:

API Keys

API Keys are revokable credentials for the Twilio API. You can use Tokens, which are used by Twilio's Real-Time Communications SD




Enter the key name and with any name you like, leave the key type as standard and then click the "Create API Key" button.


New API Key

Properties

FRIENDLY NAME

KEY TYPE Standard 

Standard Keys cannot manage API Keys, Account Configuration, and Sub Accounts



Create API Key Cancel

Copy all values to future use, but edit the .env file (on root project's directory) again and complete information such as:

```
TWILIO_ACCOUNT_SID=<your-twilio-account-sid>
TWILIO_API_KEY_SID=<your-twilio-api-key-sid>
TWILIO_API_KEY_SECRET=<your-twilio-api-key-secret>
```

2.3 – Create a Python Virtual Environment

Let's install Python dependencies to our project:

```
$python -m venv venv
$source venv/bin/activate
$ (venv) pip install twilio flask python-dotenv
```

2.4 – Creating a Web Server

As mentioned in the Architecture section, we will be using Flask framework to implement our web server. We will use a single file called app.py.

App.py contents:

```
import os
from dotenv import load_dotenv
from flask import Flask, render_template, request, abort
from twilio.jwt.access_token import AccessToken
from twilio.jwt.access_token.grants import VideoGrant

load_dotenv()
twilio_account_sid = os.environ.get('TWILIO_ACCOUNT_SID')
twilio_api_key_sid = os.environ.get('TWILIO_API_KEY_SID')
twilio_api_key_secret = os.environ.get('TWILIO_API_KEY_SECRET')

app = Flask(__name__)

@app.route('/')
def index():
```

```
return render_template('index.html')
```

```
@app.route('/login', methods=['POST'])
def login():
    username = request.get_json(force=True).get('username')
    if not username:
        abort(401)

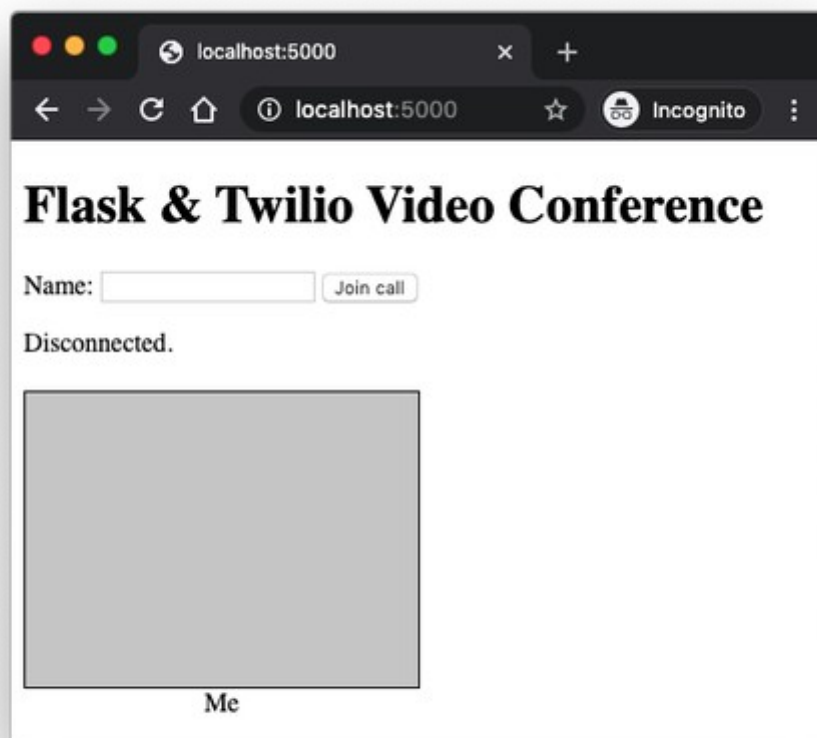
    token = AccessToken(twilio_account_sid, twilio_api_key_sid,
                        twilio_api_key_secret, identity=username)
    token.add_grant(VideoGrant(room='My Room'))

    return {'token': token.to_jwt().decode()}
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

2.5 – Application Page Layout:

Here's is how the page will look like:



This is a combination of html and CSS. Bellow you can see the templates/index.html file.

```
<!doctype html>
<html>
  <head>
```

```

        <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='styles.css') }}">
    </head>
    <body>
        <h1>Flask & Twilio Video Conference</h1>
        <form>
            <label for="username">Name: </label>
            <input type="text" name="username" id="username">
            <button id="join_leave">Join call</button>
            <button id="share_screen" disabled>Share screen</button>
        </form>
        <p id="count">Disconnected.</p>
        <div id="container" class="container">
            <div id="local" class="participant"><div></div><div class="label">Me</div></div>
            <!-- more participants will be added dynamically here -->
        </div>

        <script src="//media.twiliocdn.com/sdk/js/video/releases/2.3.0/twilio-video.min.js"></
script>
        <script src="{{ url_for('static', filename='app.js') }}"></script>
    </body>
</html>

```

The contents of styles.css file:

```

.container {
margin-top: 20px;
width: 100%;
display: flex;
flex-wrap: wrap;
}
.participant {
margin-bottom: 5px;
margin-right: 5px;
}
.participant div {
text-align: center;
}
.participant div video {
background-color: #ccc;
border: 1px solid black;
}
.participant div video:not(.participantZoomed) {
width: 240px;
height: 180px;
}
.participant .label {
background-color: #ddd;
}
.participantZoomed {
position: absolute;
top: 0px;
left: 0px;
width: 100%;
height: 100%;
}
.participantHidden {
display: none;
}

```

2.6 – Displaying video feed

Now we must create the app.js file which will be used for index.html page in order to load video and audio features:

```
const usernameInput = document.getElementById('username');
const button = document.getElementById('join_leave');
const shareScreen = document.getElementById('share_screen');
const container = document.getElementById('container');
const count = document.getElementById('count');
let connected = false;
let room;
let screenTrack;

function addLocalVideo() {
  Twilio.Video.createLocalVideoTrack().then(track => {
    let video = document.getElementById('local').firstChild;
    let trackElement = track.attach();
    trackElement.addEventListener('click', () => { zoomTrack(trackElement); });
    video.appendChild(trackElement);
  });
};

function connectButtonHandler(event) {
  event.preventDefault();
  if (!connected) {
    let username = usernameInput.value;
    if (!username) {
      alert('Enter your name before connecting');
      return;
    }
    button.disabled = true;
    button.innerHTML = 'Connecting...';
    connect(username).then(() => {
      button.innerHTML = 'Leave call';
      button.disabled = false;
      shareScreen.disabled = false;
    }).catch(() => {
      alert('Connection failed. Is the backend running?');
      button.innerHTML = 'Join call';
      button.disabled = false;
    });
  }
  else {
    disconnect();
    button.innerHTML = 'Join call';
    connected = false;
    shareScreen.innerHTML = 'Share screen';
    shareScreen.disabled = true;
  }
};

function connect(username) {
  let promise = new Promise((resolve, reject) => {
    // get a token from the back end
    fetch('/login', {
      method: 'POST',
      body: JSON.stringify({'username': username})
    })
  });
}
```

```

    }).then(res => res.json()).then(data => {
      // join video call
      return Twilio.Video.connect(data.token);
    }).then(_room => {
      room = _room;
      room.participants.forEach(participantConnected);
      room.on('participantConnected', participantConnected);
      room.on('participantDisconnected', participantDisconnected);
      connected = true;
      updateParticipantCount();
      resolve();
    }).catch(() => {
      reject();
    });
  });
  return promise;
};

function updateParticipantCount() {
  if (!connected)
    count.innerHTML = 'Disconnected.';
  else
    count.innerHTML = (room.participants.size + 1) + ' participants online.';
};

function participantConnected(participant) {
  let participantDiv = document.createElement('div');
  participantDiv.setAttribute('id', participant.sid);
  participantDiv.setAttribute('class', 'participant');

  let tracksDiv = document.createElement('div');
  participantDiv.appendChild(tracksDiv);

  let labelDiv = document.createElement('div');
  labelDiv.setAttribute('class', 'label');
  labelDiv.innerHTML = participant.identity;
  participantDiv.appendChild(labelDiv);

  container.appendChild(participantDiv);

  participant.tracks.forEach(publication => {
    if (publication.isSubscribed)
      trackSubscribed(tracksDiv, publication.track);
  });
  participant.on('trackSubscribed', track => trackSubscribed(tracksDiv, track));
  participant.on('trackUnsubscribed', trackUnsubscribed);

  updateParticipantCount();
};

function participantDisconnected(participant) {
  document.getElementById(participant.sid).remove();
  updateParticipantCount();
};

function trackSubscribed(div, track) {
  let trackElement = track.attach();
  trackElement.addEventListener('click', () => { zoomTrack(trackElement); });
  div.appendChild(trackElement);
}

```

```

};

function trackUnsubscribed(track) {
  track.detach().forEach(element => {
    if (element.classList.contains('participantZoomed')) {
      zoomTrack(element);
    }
    element.remove()
  });
};

function disconnect() {
  room.disconnect();
  while (container.lastChild.id !== 'local')
    container.removeChild(container.lastChild);
  button.innerHTML = 'Join call';
  connected = false;
  updateParticipantCount();
};

function shareScreenHandler() {
  event.preventDefault();
  if (!screenTrack) {
    navigator.mediaDevices.getDisplayMedia().then(stream => {
      screenTrack = new Twilio.Video.LocalVideoTrack(stream.getTracks()[0]);
      room.localParticipant.publishTrack(screenTrack);
      screenTrack.mediaStreamTrack.onended = () => { shareScreenHandler() };
      console.log(screenTrack);
      shareScreen.innerHTML = 'Stop sharing';
    }).catch(() => {
      alert('Could not share the screen.')
    });
  }
  else {
    room.localParticipant.unpublishTrack(screenTrack);
    screenTrack.stop();
    screenTrack = null;
    shareScreen.innerHTML = 'Share screen';
  }
};

function zoomTrack(trackElement) {
  if (!trackElement.classList.contains('participantZoomed')) {
    // zoom in
    container.childNodes.forEach(participant => {
      if (participant.className === 'participant') {
        participant.childNodes[0].childNodes.forEach(track => {
          if (track === trackElement) {
            track.classList.add('participantZoomed')
          }
          else {
            track.classList.add('participantHidden')
          }
        });
        participant.childNodes[1].classList.add('participantHidden');
      }
    });
  }
  else {

```



```

// zoom out
container.childNodes.forEach(participant => {
  if (participant.className == 'participant') {
    participant.childNodes[0].childNodes.forEach(track => {
      if (track === trackElement) {
        track.classList.remove('participantZoomed');
      }
      else {
        track.classList.remove('participantHidden');
      }
    });
    participant.childNodes[1].classList.remove('participantHidden');
  }
});
};

addLocalVideo();
button.addEventListener('click', connectButtonHandler);
shareScreen.addEventListener('click', shareScreenHandler);

```

2.7 – Running The Video Chat Server

Now that all intelligence is done, all we need is to start the server:

```
(env) $ FLASK_ENV=development flask run
```

```

1  * Environment: development
2  * Debug mode: on
3  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
4  * Restarting with stat
5  * Debugger is active!
6  * Debugger PIN: 274-913-316

```

With the server running you can connect from the same computer by entering `http://localhost:5000` on the address bar of your web browser. But without you may want connect from a second computer or a smartphone, or maybe even invite a friend to join your video chat, you won't be able to do that because this requires one more step, since the server is only running internally on your computer and is not accessible from the Internet.

2.8 – Download ngrok

Using the ngrok, a handy utility that creates a tunnel between our locally running server and a public URL on the ngrok.io domain, you will be able to invite others to join to this chat:

```
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
```

With the Flask server running, open a second terminal window and start ngrok as follows:

```
$ ngrok http 5000
```

```
ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Session Expires     7 hours, 59 minutes
Version             2.3.35
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://bbf1b72b.ngrok.io -> http://localhost:5000
Forwarding           https://bbf1b72b.ngrok.io -> http://localhost:5000

Connections         ttl    opn    rt1    rt5    p50    p90
0                0      0.00   0.00   0.00   0.00
```

Find the forwarding lines to see what's the public URL that ngrok assigned to your server. Use the one which start with `https://`, since many browsers do not allow unencrypted sites to access camera and microphone. While having flask and ngrok running, you'll be able to use the public URL to make videoconfere calls. Now, you're able to invite other to join you.