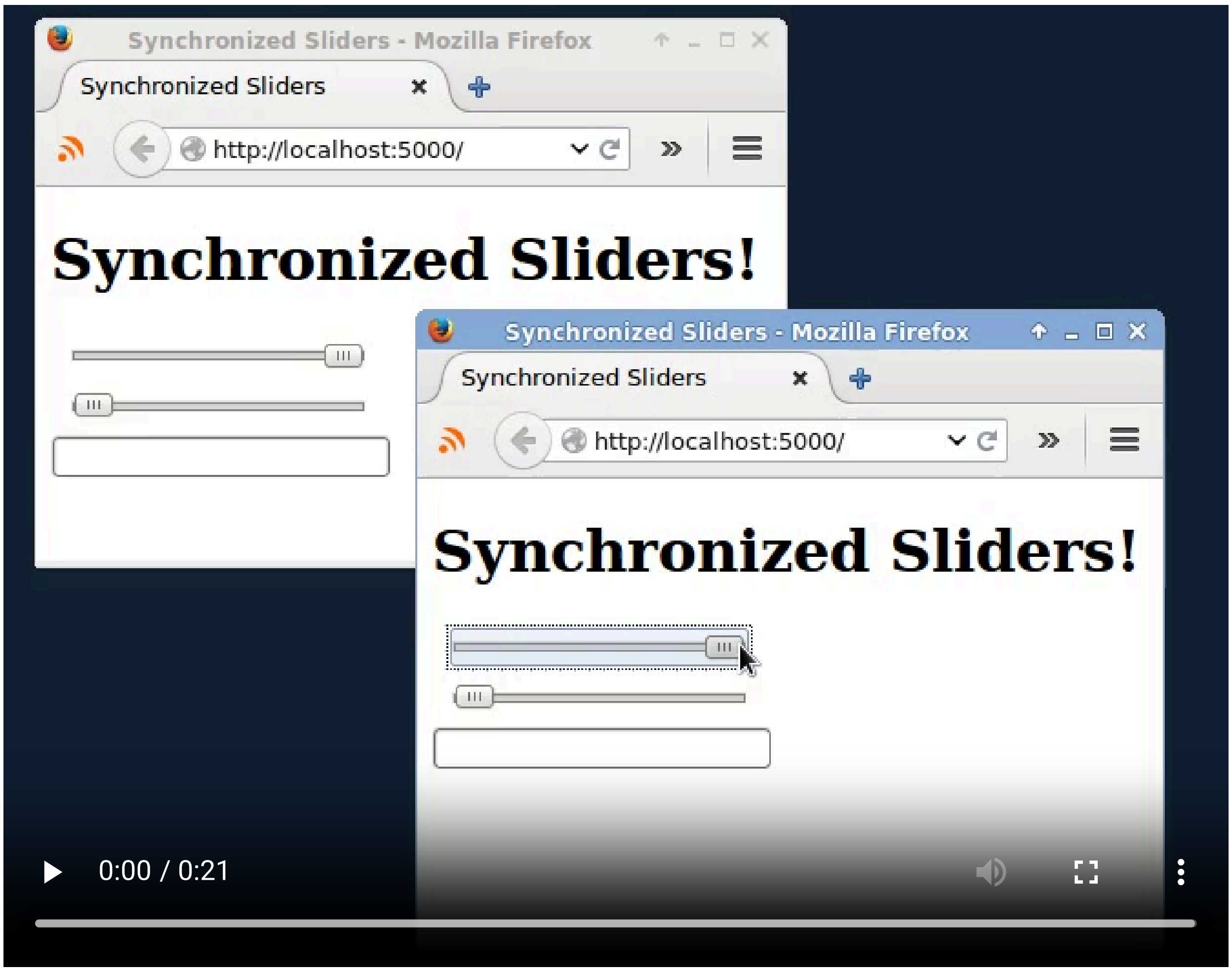
Synchronize clients of a Flask application with websockets

6 mars 2015
#Python [https://matthieuamiguet.ch//tags/python/] #Web [https://matthieuamiguet.ch//tags/web/]

tl;dr

If you' ve got a browser that supports html5 video and want to take the fast track, you could have a look at this 20-seconds demo and jump to the full code [#code] at the end of this article.



The browser as a GUI

When writing a python application for on-stage use, I' m more and more tempted to use a browser as the GUI. Using a microframework such as Flask [http://flask.pocoo.org/] , this can be done in minutes (literally), can be used locally or from a remote device (it looks much cooler to touch a tablet than grab a mouse on stage!) and it even allows several people to control the application from multiple devices.

But wait… there is a catch with this last point. If I use my browser to change something, this will not be visible to my colleagues unless they reload the page.

Websockets

There is a solution to this problem: websockets [https://www.websocket.org/] . They allow to establish a bi-directional communication channel between the server and the browser, allowing for real-time updates on both sides.

As it happens, Flask users are lucky: they can use the Flask-SocketIO [https://github.com/miguelgrinberg/Flask-SocketIO] extension on the server side, plus some jQuery [http://jquery.com/] + socket.io [http://socket.io/] on the client side to achieve this with astonishing simplicity (note that it seems [http://blog.miguelgrinberg.com/post/easy-websockets-with-flask-and-gevent] that you should use python 2.x for now, tough).

Let’ s try to develop a mini-app with a few sliders that get synchronized between clients.

Let’ s go

You’ ll probably want to create and activate a virtualenv [https://virtualenv.pypa.io/en/latest/] first (remember to use python 2.x), then install flask-socketio (which will install Flask and its dependencies as well)

```
$ pip install flask-socketio
```

Now create a file with an almost standard minimal Flask application:

```
from flask import Flask, render_template
from flask.ext.socketio import SocketIO

app = Flask(__name__)
socketio = SocketIO(app)

@app.route('/')
def index():
    return render_template('index.html')

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

This looks very much like a Flask application, except for the added `socketio = SocketIO(app)` and the way the app is run on the last line (note that the `host='0.0.0.0'` part is optional but allows to connect from another machine).

Of course you’ ll need a `templates/index.html` file:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Synchronized Sliders</title>
    </head>
    <body>
        <h1>Synchronized Sliders!</h1>

        <input id="slider1" type="range" min="0" max="50" value="25" /> <br>
        <input id="slider2" type="range" min="0" max="50" value="0" />

    </body>
</html>
```

Now run your python file and point your browser to http://localhost:5000/ and you should see a page with two sliders.

But of course, if you open a second browser and move a slider, the sliders in the first window will stay still.

Sending message from the client to the server

Now we' ll have to make these sliders alive. We' ll first add socket.io and jQuery to our web page. For now, the recommended [http://flask-socketio.readthedocs.org/en/latest/] socket.io version is 0.9.16.

In the html template file, add these lines to the header:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script type="text/javascript"
src="//cdnjs.cloudflare.com/ajax/libs/socket.io/0.9.16/socket.io.min.js"></script>
```

Note: Of course if you intend to use this on stage, you' ll have to store these files locally, but for this quick demo this is enough.

Right under these lines, add the following script:

```
<script type="text/javascript" charset="utf-8">
    $(document).ready(function(){

        var socket = io.connect();

        // the socket.io documentation recommends sending an explicit package upon
connection
        socket.on('connect', function() {
            socket.emit('connect', {data: 'I\'m connected!'});
        });

        $('input.sync').on('input', function(event) {
            socket.emit('value changed', {who: $(this).attr('id'), data: $(this).val()});
            return false;
        });

    });
</script>
```

The code is very simple:

- First we create a websocket connection to the server.
- Then we send a message to notify the server that we are connected. The first argument of `socket.emit` can be seen as the type of the message and will be used later for filtering. The second is the payload and will be available on the server side as a python dict.
- Finally, we use jQuery to emit a message on the websocket each time the value of a slider changes. We then send the id of the slider and its new value.

Note: We use the `input` event in place of the more traditional `change` event because the latter is fired only when the mouse is released as the former is fired as soon as there is a change.

The jQuery invocation only acts on `input`s with class `sync`, so we' ll have to add this class to the sliders.

```
<input id="slider1" class="sync" type="range" min="0" max="50" value="25" /> <br>
<input id="slider2" class="sync" type="range" min="0" max="50" value="0" />
```

You can now run the app again, but nothing visible will happen: the messages are sent, but we don' t do anything with them!

Getting message on the server

Now in your python file, add these (ridiculously simple) lines:

```
@socketio.on('value changed')
def value_changed(message):
    print(message)
```

Run your application: each time you move a slider, the message sent is dumped to the console.

But of course we don' t really mind the console. In your python code, add

```
values = {
    'slider1': 25,
    'slider2': 0,
}
```

and modify the value_changed function:

```
@socketio.on('value changed')
def value_changed(message):
    values[message['who']] = message['data']
```

Make this information available when rendering the template

```
@app.route('/')
def index():
    return render_template('index.html', **values)
```

and modify the template accordingly:

```
<input id="slider1" class="sync" type="range" min="0" max="50" value="{{slider1}}" />
<br>
<input id="slider2" class="sync" type="range" min="0" max="50" value="{{slider2}}" />
```

Now if you point a browser to your app, modify the sliders positions and *then* point another browser on the same page, it will display the new values rather than the default ones. Better, but this still isn't what I would call synchronization.

Full synchronization

We will need the server to send websocket messages, so the first thing we want to do is complete the import line:

```
from flask.ext.socketio import SocketIO, emit
```

Then we modify the `value_changed` function to forward the message to all connected clients:

```
@socketio.on('value changed')
def value_changed(message):
    values[message['who']] = message['data']
    emit('update value', message, broadcast=True)
```

And finally we add some code to the javascript in the template:

```
socket.on('update value', function(msg) {
    $('input#'+msg.who).val(msg.data)
});
```

This simply means: whenever you get an "update value" message `msg`, find the input with id `msg.who` and set its value to `msg.data`.

And there we are: point two browsers to your app and move the sliders: the changes are synchronized between them! You can even try it from your phone or tablet, this should work as well.

And now for the good news: nothing in our code is specific to sliders; try adding a text input under the sliders, it should work too:

```
<input id="txt1" class="sync" type="text" />
```

Full code []

This might seem complicated as we've been doing it step by step, but a look at the full python code makes it clear how easy this is:

```
from flask import Flask, render_template
from flask.ext.socketio import SocketIO, emit

app = Flask(__name__)
socketio = SocketIO(app)

values = {
    'slider1' : 25,
    'slider2' : 0,
}

@app.route('/')
def index():
    return render_template('index.html', **values)

@socketio.on('value changed')
def value_changed(message):
    values[message['who']] = message['data']
    emit('update value', message, broadcast=True)

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

The client side is slightly more verbose, but still very easy:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Synchronized Sliders</title>
        <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
        <script type="text/javascript"
src="//cdnjs.cloudflare.com/ajax/libs/socket.io/0.9.16/socket.io.min.js"></script>
        <script type="text/javascript" charset="utf-8">
            $(document).ready(function(){

                var socket = io.connect();

                socket.on('connect', function() {
                    socket.emit('connect', {data: 'I\'m connected!'});
                });

                $('input.sync').on('input', function(event) {
                    socket.emit('value changed', {who: $(this).attr('id'), data:
$(this).val()});
                    return false;
                });

                socket.on('update value', function(msg) {
                    $('input#'+msg.who).val(msg.data)
                });

            });
        </script>
    </head>
    <body>
        <h1>Synchronized Sliders!</h1>

        <input id="slider1" class="sync" type="range" min="0" max="50" value="
{{slider1}}" /> <br>
        <input id="slider2" class="sync" type="range" min="0" max="50" value="
{{slider2}}" />

        <input id="txt1" class="sync" type="text" />

    </body>
</html>
```

I think the combination socket.IO [http://socket.io/] + Flask-SocketIO [https://github.com/miguelgrinberg/Flask-SocketIO] makes all this incredibly easy and I’ d like to thank the authors of these two libraries for their great work.

Articles connexes:

- Wifi Client Mode for Mugic [/blog/wifi-client-mode-mugic/]
- Mise à jour: Sondages avec Flask [/blog/polls-flask-ethercalc/]
- Mise à jour: Sondages avec Flask [/blog/mise-jour-sondages-avec-flask/]
- Apache Log Analysis with Haskell and Python [/blog/apache-log-analysis-haskell-python/]
- Tutoriel: Sondages avec Flask [/blog/tutoriel-flask/]