Real-time Web Applications

- Collaborative Documents (Google Docs, Office 365, etc.)
- Many Browser Games
- Email Clients
- Social Media
One method systems have often used to provide the illusion of real-time applications is polling. This method generally uses jQuery and AJAX in the background.

- Client sends any necessary updates to the server when events are handled
  - Server saves data in memory or persists to database
- Client periodically asks server for updates
  - Server responds with all relevant updates since last request

This method is effective for many applications, but the more frequent polling occurs, the more load it puts on the server and the fewer clients it can reasonably support.
One way to improve on polling is to have the server respond only when it has updates.

- Client sends any necessary updates to the server when events are handled
  - Server saves data in memory or persists to database
- Client periodically asks server for updates
  - Server waits until it has an update before responding. If this takes too long, the request times out and the client will issue a new request.

This method actually achieves real-time connections, but still has a very high per client overhead cost.
In order to avoid timeouts and extra overhead, many modern browsers now support direct TCP connections between client and server in the form of Web Sockets. We will give an example using

- SocketIO (JavaScript Library)
- Flask-SocketIO (Third-Party Flask Extension)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Application</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport</td>
</tr>
<tr>
<td>IP</td>
<td>Internet/Network</td>
</tr>
<tr>
<td>Ethernet</td>
<td>Datalink</td>
</tr>
<tr>
<td>IEEE 802.3ah</td>
<td>Physical</td>
</tr>
</tbody>
</table>
SocketIO and Flask-SocketIO

- **SocketIO** | client-side JavaScript library that supports WebSockets \(^1\)
- **Flask-SocketIO** | builds server-side compatibility between SocketIO and Flask \(^2\)

We will provide a simple example of how this works by creating a multi-user live chat server.

---

\(^1\) It falls back on Long Polling when WebSockets are not supported

\(^2\) Using the development server you will actually fall back on Long Polling
Setting up Flask-Socket-IO

```python
from flask import Flask, request, session
from flask import render_template, url_for, redirect
from flask_socketio import SocketIO
from flask_socketio import emit, join_room, leave_room
from flask_wtf import FlaskForm
from wtforms.fields import StringField, SubmitField
from wtforms.validators import Required

app = Flask(__name__)
app.config['SECRET_KEY'] = 'gjr39dkjn34asdfsfsf987fh432f4'
socketio = SocketIO(app)
```
@app.route("/", methods=['GET', 'POST'])
def index():
    form = LoginForm()
    if form.validate_on_submit():
        session['name'] = form.name.data
        session['room'] = form.room.data
        return redirect(url_for("chat"))
    elif request.method == "GET":
        form.name.data = session.get("name", 
form.room.data = session.get("room", 
return render_template("index.html", form=form)
```python
@app.route("/chat", methods=["GET"])
def chat():
    name = session.get("name", None)
    room = session.get("room", None)
    if name is None or room is None:
        return redirect(url_for("index"))
    return render_template("chat.html", name=name, room=room)
```
<html>
  <head>
    <title>Flask-SocketIO-Chat: {{ room }}</title>
    <script type="text/javascript" src="jQuery CDN"></script>
    <script type="text/javascript" src="socket.io CDN"></script>
    <script type="text/javascript" charset="utf-8">
      ...
    </script>
  </head>
  <body>
    <h1>Flask-SocketIO-Chat: {{ room }}</h1>
    <textarea id="chat" cols="80" rows="20"></textarea><br><br>
    <input id="text" size="80" placeholder="Say Something"><br><br>
    <a href="#" onclick="leave_room();">Leave this room</a>
  </body>
</html>
var socket;
$(document).ready(function(){
    socket = io.connect("http://" + document.domain + ":" +
                        location.port + "/chat");

    socket.on("connect", function(){socket.emit("joined", {});});

    socket.on("status", function(data) {
        $('#chat').val($('#chat').val() + "<" + data.msg + ">
        
    $('#chat').scrollTop($('#chat')[0].scrollHeight);
    });

    socket.on("message", function(data) {
        $('#chat').val($('#chat').val() + data.msg + "\n"
        
    $('#chat').scrollTop($('#chat')[0].scrollHeight);
    });
});
```javascript
$("#text").keypress(function(e) {
    var code = e.keyCode || e.which;
    if (code == 13) {
        text = $("#text").val();
        $("#text").val('');
        socket.emit("text", {msg: text});
    }
});

function leave_room() {
    socket.emit("left", {}, function() {
        socket.disconnect();
        window.location.href = "{{ url_for('index') }}";
    });
}
```
@socketio.on("joined", namespace="/chat")
def joined(message):
    
    """
    Sent by clients when they enter a room.
    A status message is broadcast to all people in the room.
    """

    room = session.get("room")
    join_room(room)
    emit("status", {"msg": session.get("name") + " has entered the room."}, room=room)
@socketio.on("text", namespace="/chat")
def text(message):
    
    """
    Sent by a client when the user entered a new message. The message is sent to all people in the room.
    """

    room = session.get("room")
    emit("message", {"msg": session.get("name") + ":" + \n        message["msg"]}, room=room)
@socketio.on("left", namespace="/chat")
def left(message):
    ""
    Sent by clients when they leave a room.
    A status message is broadcast to all people in the room.
    ""
    room = session.get("room")
    leave_room(room)
    emit("status", {"msg": session.get("name") + " has left the room."}, room=room)
Flask-SocketIO.
Flask-socketio.

Miguel Grinberg.
Flask-socketio-chat.