Client-Side Rendering in Flask

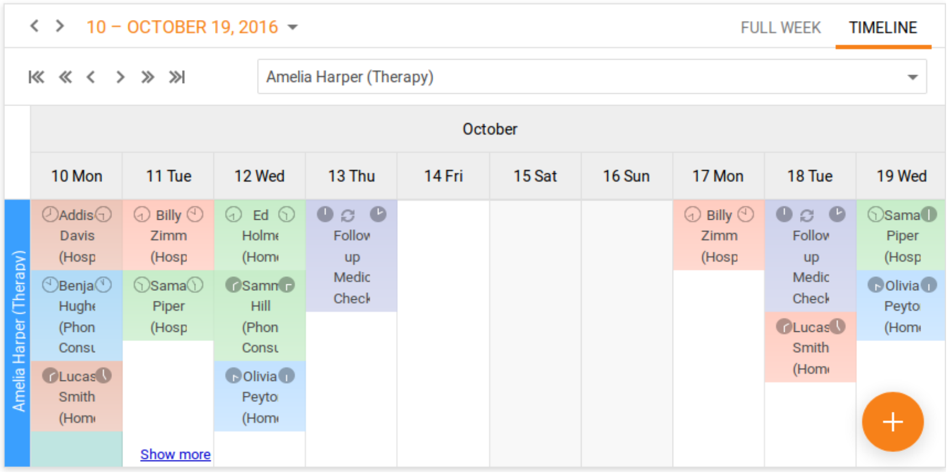
CITS3403 and CITS5505 - Agile Web Development

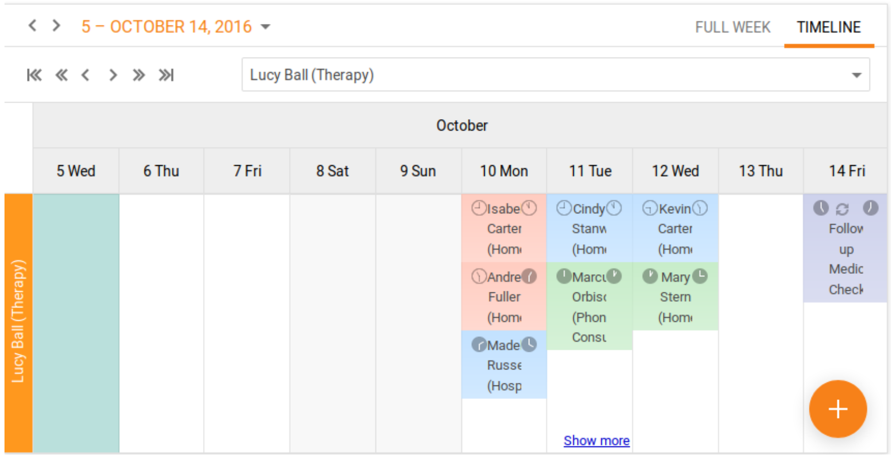
Adapted from the Flask Mega-Tutorial, by Miguel Grinberg: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial

Semester 1, 2024

Why use client-side rendering?

- So far, the JavaScript we have seen responds to local events in the browser, such as users clicking buttons, pages loading, and mouse movements.
- However, we often want to respond to remote events, such as someone sending you a message, liking a post etc.
- We also may want to dynamically respond to a local event using information on the server: if a user enters the 1st of April as a preferred appointment date, then we would like to immediately show them the available appointments.
- We could send the date to the server, have the server rebuild the page and send the entire page back, but we only require a few bytes of data!

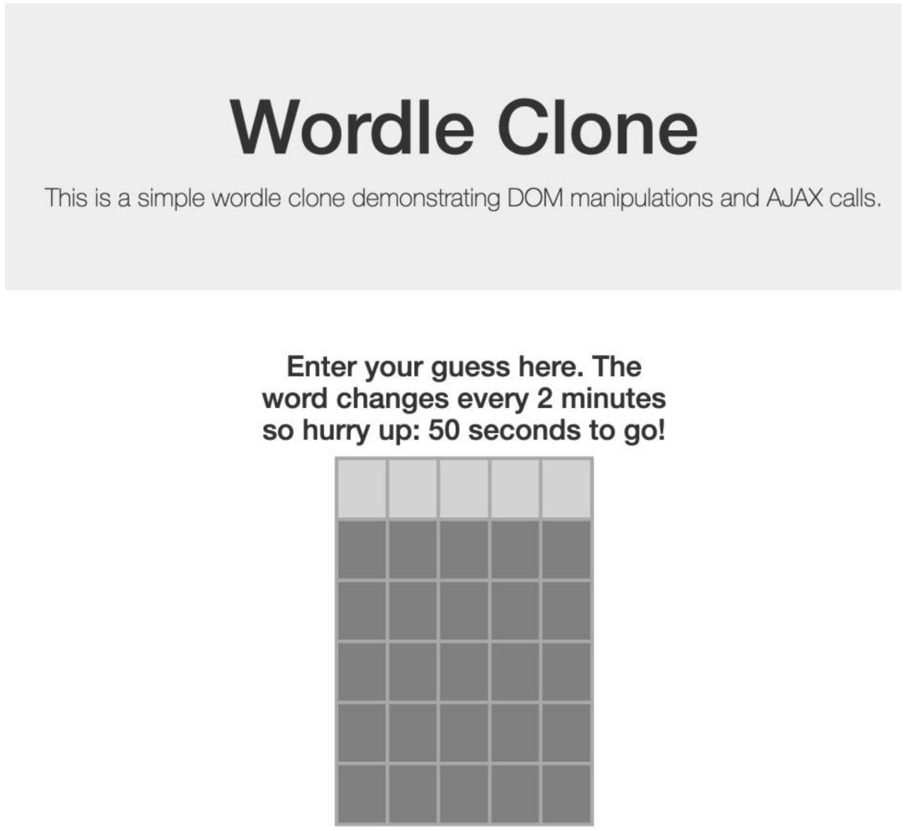




Client-side rendering with AJAX

Wordle example

- As a simple example of client-side rendering we will look at a simple clone of the popular word game Wordle.
- It will use AJAX to send and receive requests from the server.
- It will use JavaScript and the DOM to update the web page.
- Source code available at: https://drtnf.net/static/wordle.html



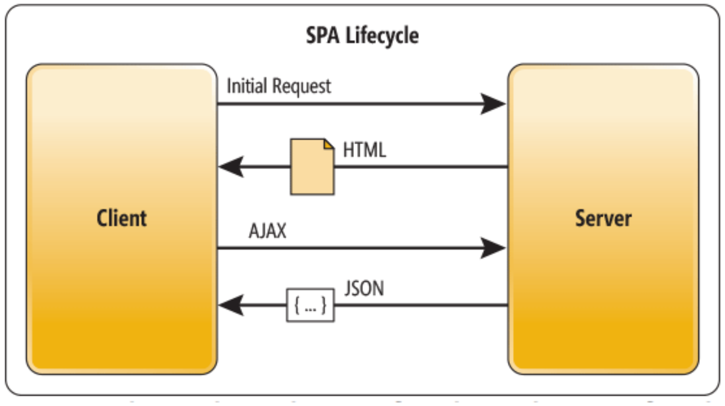
Wordle Clone

This is a simple wordle clone demonstrating DOM manipulations and AJAX calls.

Enter your guess here. The word changes every 2 minutes so hurry up: 50 seconds to go!

Initial static files

- When using client-side rendering we still need to send initial HTML/CSS/JS code.



- Unlike with server-side rendering, the same initial template is sent to every client. Therefore, such files are known as static files.

Serving static files in Flask

- Flask projects have a directory called "static" to serve static files, including HTML, CSS, JS and images.

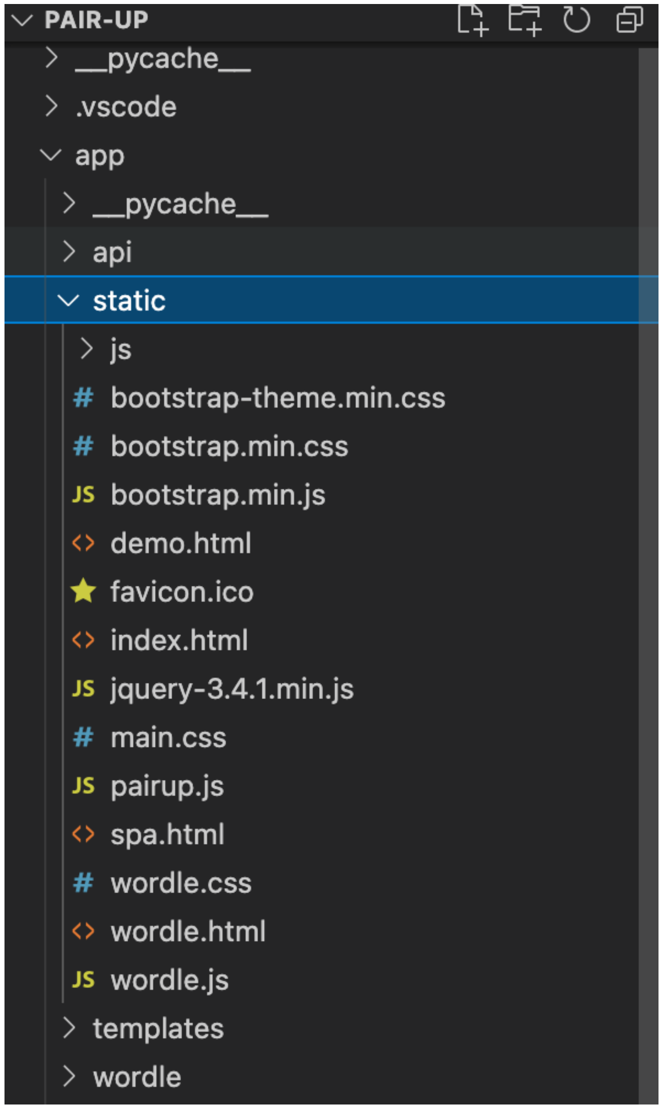
- Flask automatically creates an endpoint called `static` with the following route:

  `/static/<path:filename>`

  which loads resources from this folder.

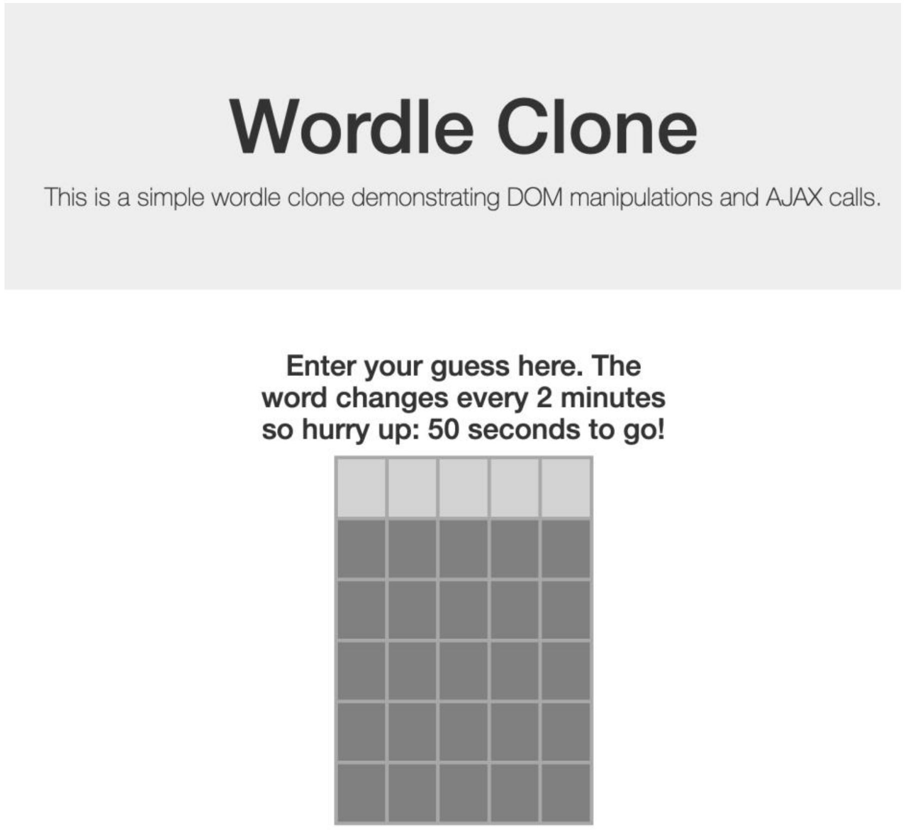
- We can then have Flask redirect requests to a given route, to the static files we want to serve using `url_for`.

```
@app.route('/speed_wordle')
def speed_wordle():
    return redirect(url_for('static', filename='wordle.html'))

```



```
PAIR-UP
  > __pycache__
  > .vscode
  v app
    > __pycache__
    > api
    v static
      > js
      bootstrap-theme.min.css
      bootstrap.min.css
      bootstrap.min.js
      demo.html
      favicon.ico
      index.html
      jquery-3.4.1.min.js
      main.css
      pairup.js
      spa.html
      wordle.css
      wordle.html
      wordle.js
    > templates
    > wordle
```

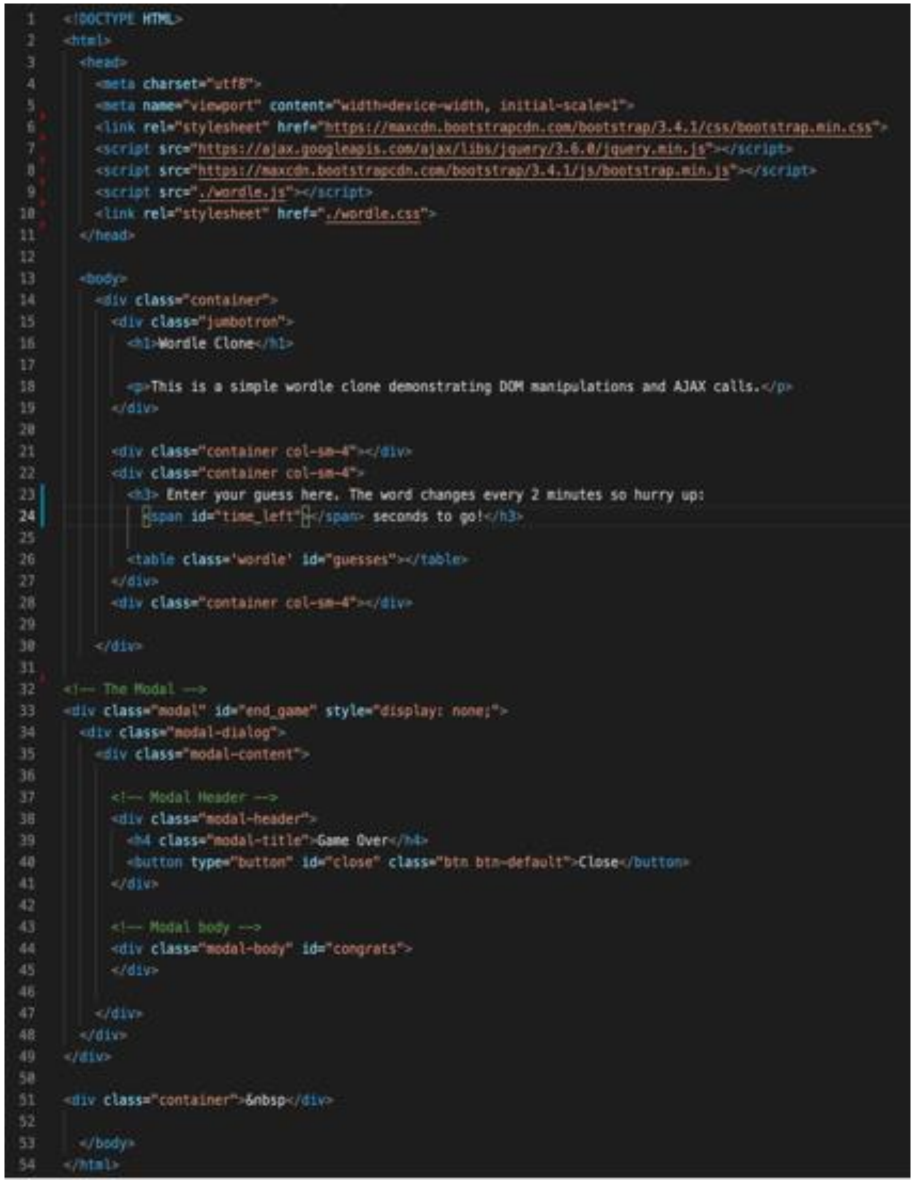
Example static HTML files



Wordle Clone

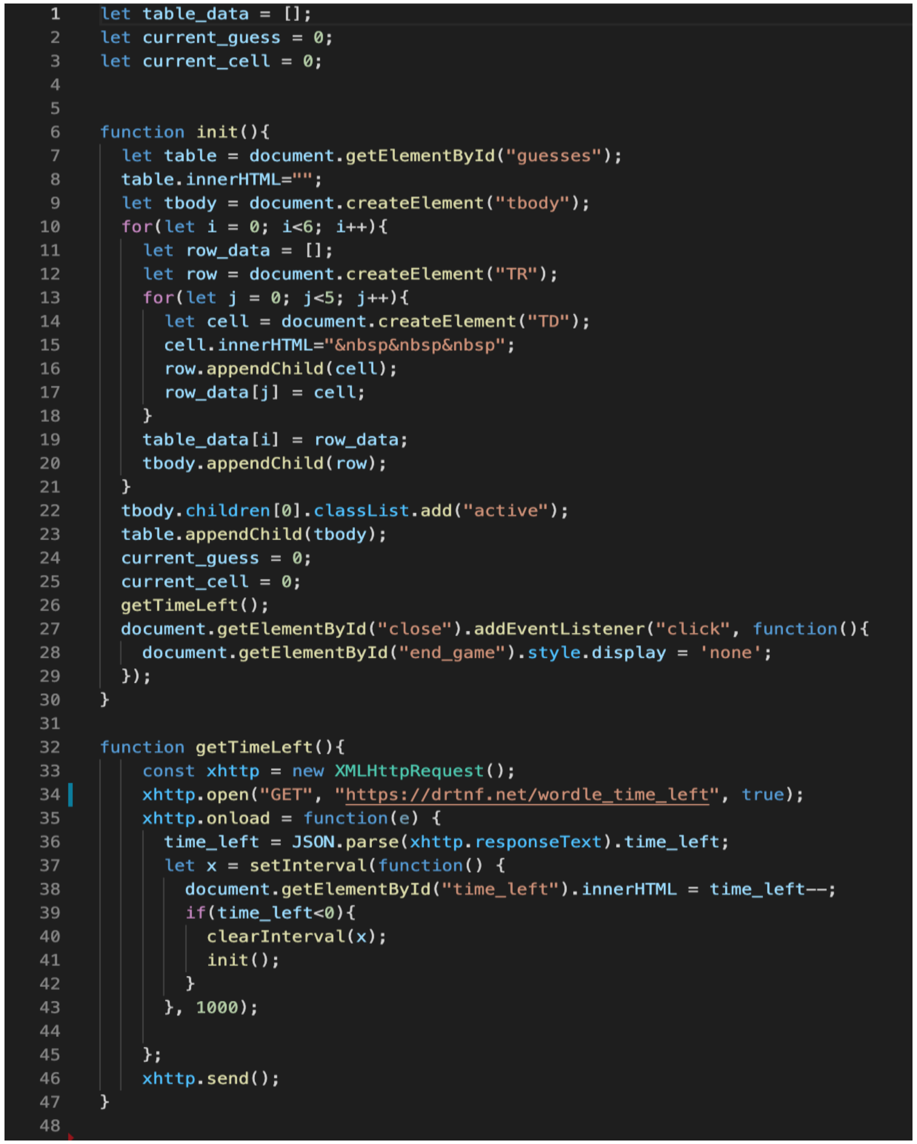
This is a simple wordle clone demonstrating DOM manipulations and AJAX calls.

Enter your guess here. The word changes every 2 minutes so hurry up: 50 seconds to go!

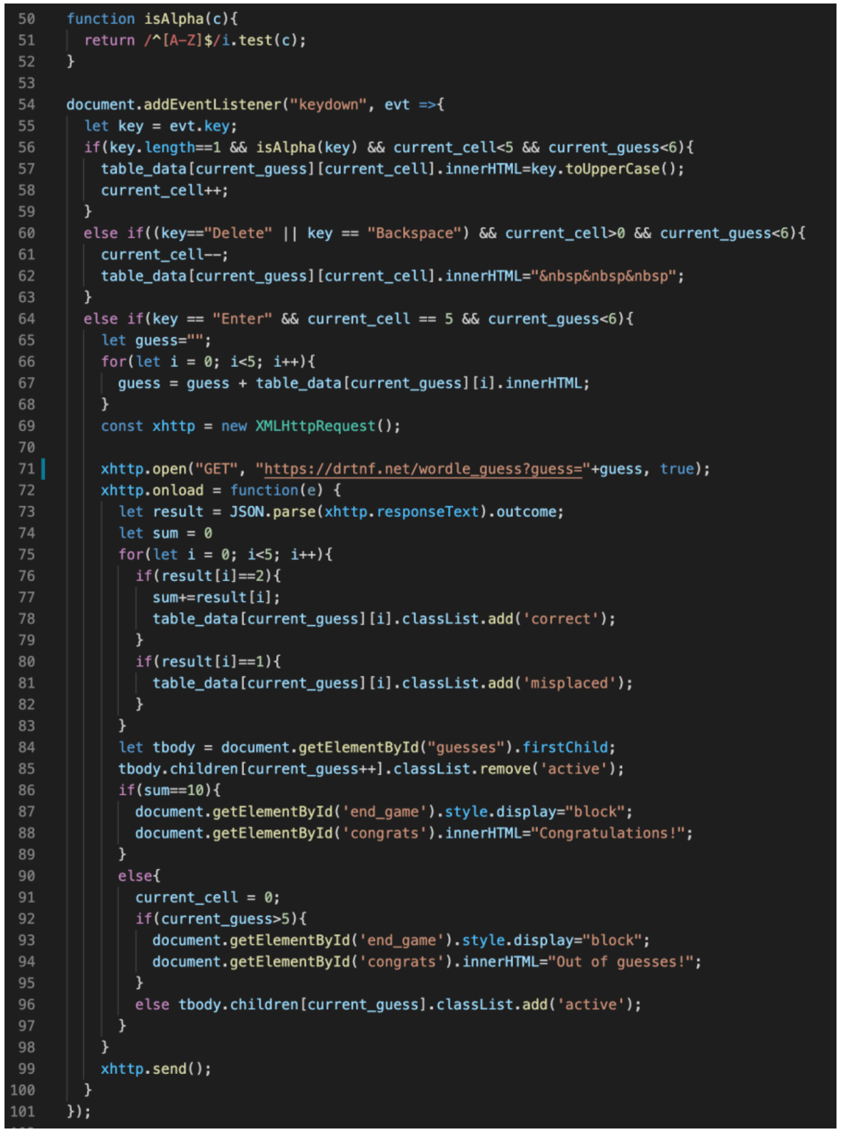


```html
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
    <script src="./wordle.js"></script>
    <link rel="stylesheet" href="./wordle.css">
  </head>

  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Wordle Clone</h1>

        <p>This is a simple wordle clone demonstrating DOM manipulations and AJAX calls.</p>
      </div>

      <div class="container col-sm-4"></div>
      <div class="container col-sm-4">
        <h3> Enter your guess here. The word changes every 2 minutes so hurry up:
          <span id="time_left"></span> seconds to go!</h3>

        <table class='wordle' id="guesses"></table>
      </div>
      <div class="container col-sm-4"></div>

    </div>

<!-- The Modal -->
<div class="modal" id="end_game" style="display: none;">
  <div class="modal-dialog">
    <div class="modal-content">

      <!-- Modal Header -->
      <div class="modal-header">
        <h4 class="modal-title">Game Over</h4>
        <button type="button" id="close" class="btn btn-default">Close</button>
      </div>

      <!-- Modal body -->
      <div class="modal-body" id="congrats">
      </div>

    </div>
  </div>
</div>

<div class="container">&nbsp</div>

  </body>
</html>
```

Example static JavaScript files



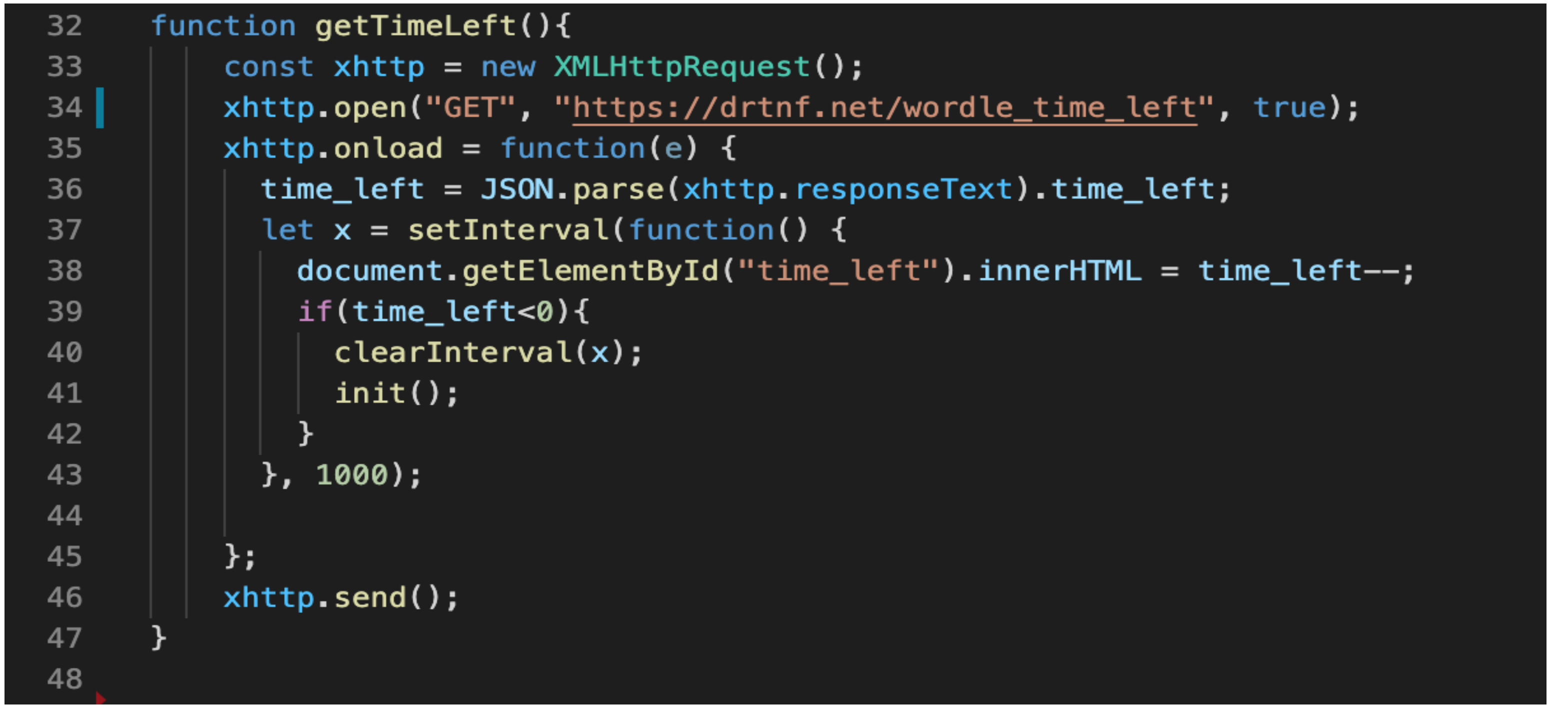
```javascript
let table_data = [];
let current_guess = 0;
let current_cell = 0;


function init(){
  let table = document.getElementById("guesses");
  table.innerHTML="";
  let tbody = document.createElement("tbody");
  for(let i = 0; i<6; i++){
    let row_data = [];
    let row = document.createElement("TR");
    for(let j = 0; j<5; j++){
      let cell = document.createElement("TD");
      cell.innerHTML="&nbsp&nbsp&nbsp";
      row.appendChild(cell);
      row_data[j] = cell;
    }
    table_data[i] = row_data;
    tbody.appendChild(row);
  }
  tbody.children[0].classList.add("active");
  table.appendChild(tbody);
  current_guess = 0;
  current_cell = 0;
  getTimeLeft();
  document.getElementById("close").addEventListener("click", function(){
    document.getElementById("end_game").style.display = 'none';
  });
}

function getTimeLeft(){
    const xhttp = new XMLHttpRequest();
    xhttp.open("GET", "https://drtnf.net/wordle_time_left", true);
    xhttp.onload = function(e) {
      time_left = JSON.parse(xhttp.responseText).time_left;
      let x = setInterval(function() {
        document.getElementById("time_left").innerHTML = time_left--;
        if(time_left<0){
          clearInterval(x);
          init();
        }
      }, 1000);

    };
    xhttp.send();
}

```



```javascript
function isAlpha(c){
  return /^[A-Z]$/i.test(c);
}

document.addEventListener("keydown", evt =>{
  let key = evt.key;
  if(key.length==1 && isAlpha(key) && current_cell<5 && current_guess<6){
    table_data[current_guess][current_cell].innerHTML=key.toUpperCase();
    current_cell++;
  }
  else if((key=="Delete" || key == "Backspace") && current_cell>0 && current_guess<6){
    current_cell--;
    table_data[current_guess][current_cell].innerHTML="&nbsp&nbsp&nbsp";
  }
  else if(key == "Enter" && current_cell == 5 && current_guess<6){
    let guess="";
    for(let i = 0; i<5; i++){
      guess = guess + table_data[current_guess][i].innerHTML;
    }
    const xhttp = new XMLHttpRequest();

    xhttp.open("GET", "https://drtnf.net/wordle_guess?guess="+guess, true);
    xhttp.onload = function(e) {
      let result = JSON.parse(xhttp.responseText).outcome;
      let sum = 0
      for(let i = 0; i<5; i++){
        if(result[i]==2){
          sum+=result[i];
          table_data[current_guess][i].classList.add('correct');
        }
        if(result[i]==1){
          table_data[current_guess][i].classList.add('misplaced');
        }
      }
      let tbody = document.getElementById("guesses").firstChild;
      tbody.children[current_guess++].classList.remove('active');
      if(sum==10){
        document.getElementById('end_game').style.display="block";
        document.getElementById('congrats').innerHTML="Congratulations!";
      }
      else{
        current_cell = 0;
        if(current_guess>5){
          document.getElementById('end_game').style.display="block";
          document.getElementById('congrats').innerHTML="Out of guesses!";
        }
        else tbody.children[current_guess].classList.add('active');
      }
    }
    xhttp.send();
  }
});
```

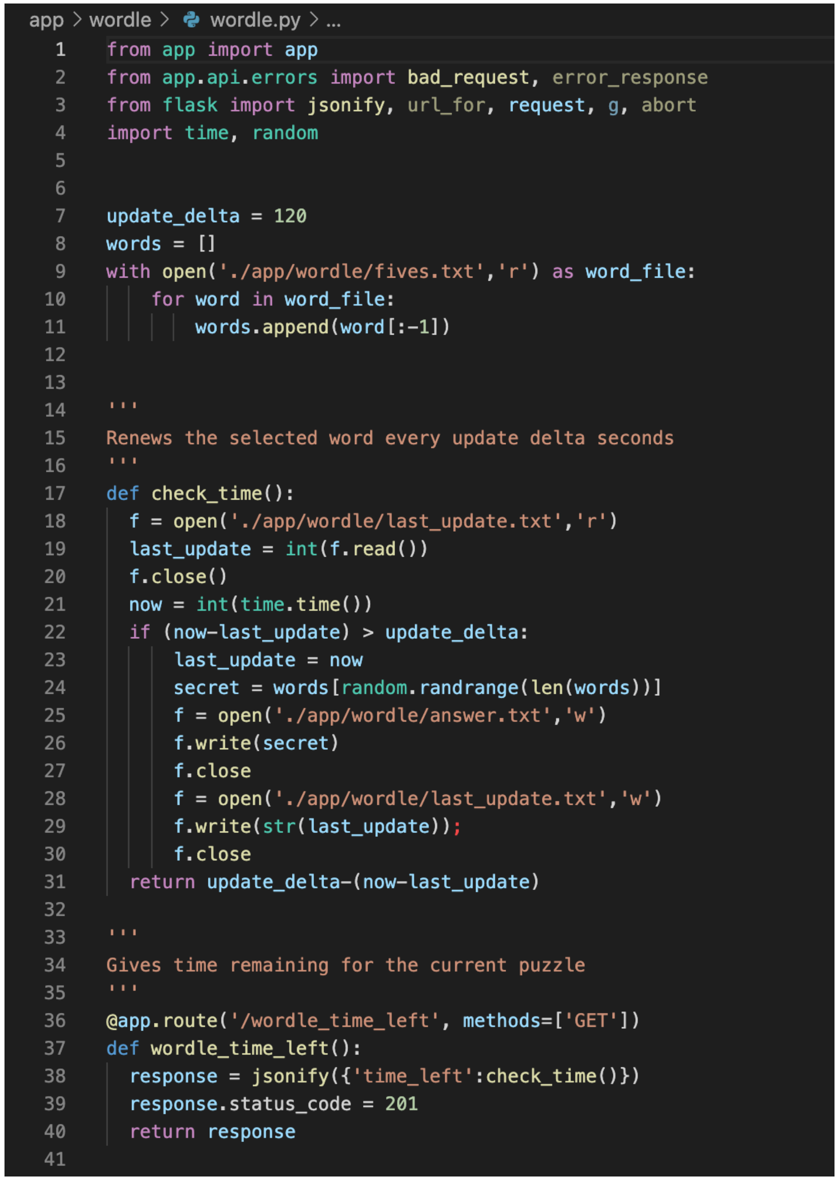
Making requests for time left

- The first example of client-side rendering is a simple request to get the time left for the current puzzle. This is a one-off request when the page is loaded.



```
function getTimeLeft(){
    const xhttp = new XMLHttpRequest();
    xhttp.open("GET", "https://drtnf.net/wordle_time_left", true);
    xhttp.onload = function(e) {
      time_left = JSON.parse(xhttp.responseText).time_left;
      let x = setInterval(function() {
        document.getElementById("time_left").innerHTML = time_left--;
        if(time_left<0){
          clearInterval(x);
          init();
        }
      }, 1000);

    };
    xhttp.send();
}
```

Responding to requests for time left



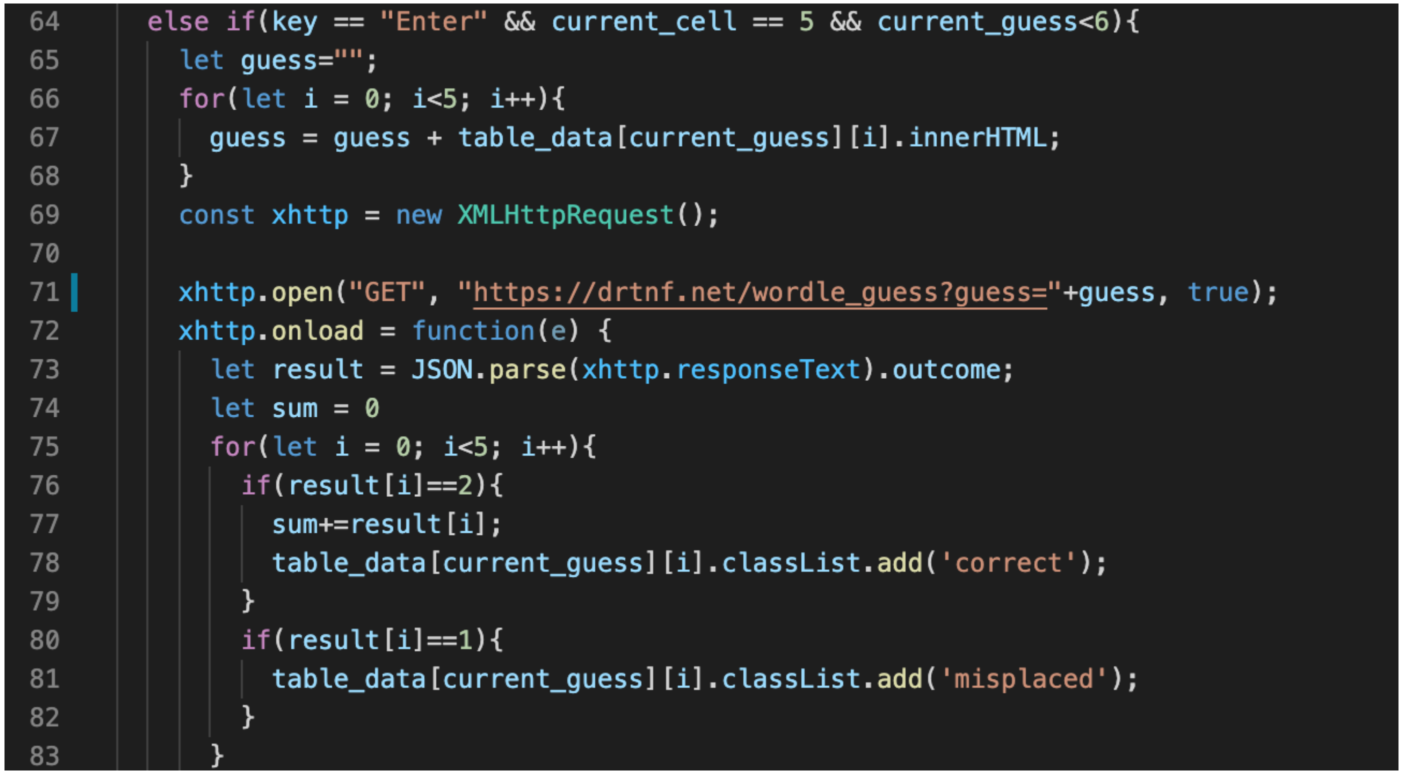
app > wordle > wordle.py > ...

```python
from app import app
from app.api.errors import bad_request, error_response
from flask import jsonify, url_for, request, g, abort
import time, random


update_delta = 120
words = []
with open('./app/wordle/fives.txt','r') as word_file:
    for word in word_file:
        words.append(word[:-1])


'''
Renews the selected word every update delta seconds
'''
def check_time():
  f = open('./app/wordle/last_update.txt','r')
  last_update = int(f.read())
  f.close()
  now = int(time.time())
  if (now-last_update) > update_delta:
    last_update = now
    secret = words[random.randrange(len(words))]
    f = open('./app/wordle/answer.txt','w')
    f.write(secret)
    f.close
    f = open('./app/wordle/last_update.txt','w')
    f.write(str(last_update));
    f.close
  return update_delta-(now-last_update)

'''
Gives time remaining for the current puzzle
'''
@app.route('/wordle_time_left', methods=['GET'])
def wordle_time_left():
  response = jsonify({'time_left':check_time()})
  response.status_code = 201
  return response

```

- When we receive a time left request, we respond to the request with a JSON object with a single field '`time_left`'.
- To respond consistently we need to persist the state, i.e. the current word and when it was created.
- As we don't have a database yet, we can store the time since the word being guessed was updated in a simple text file `` `last_update.txt` ``.
- If it's time for a new word, we write the new word into another text file '`answer.txt`'.

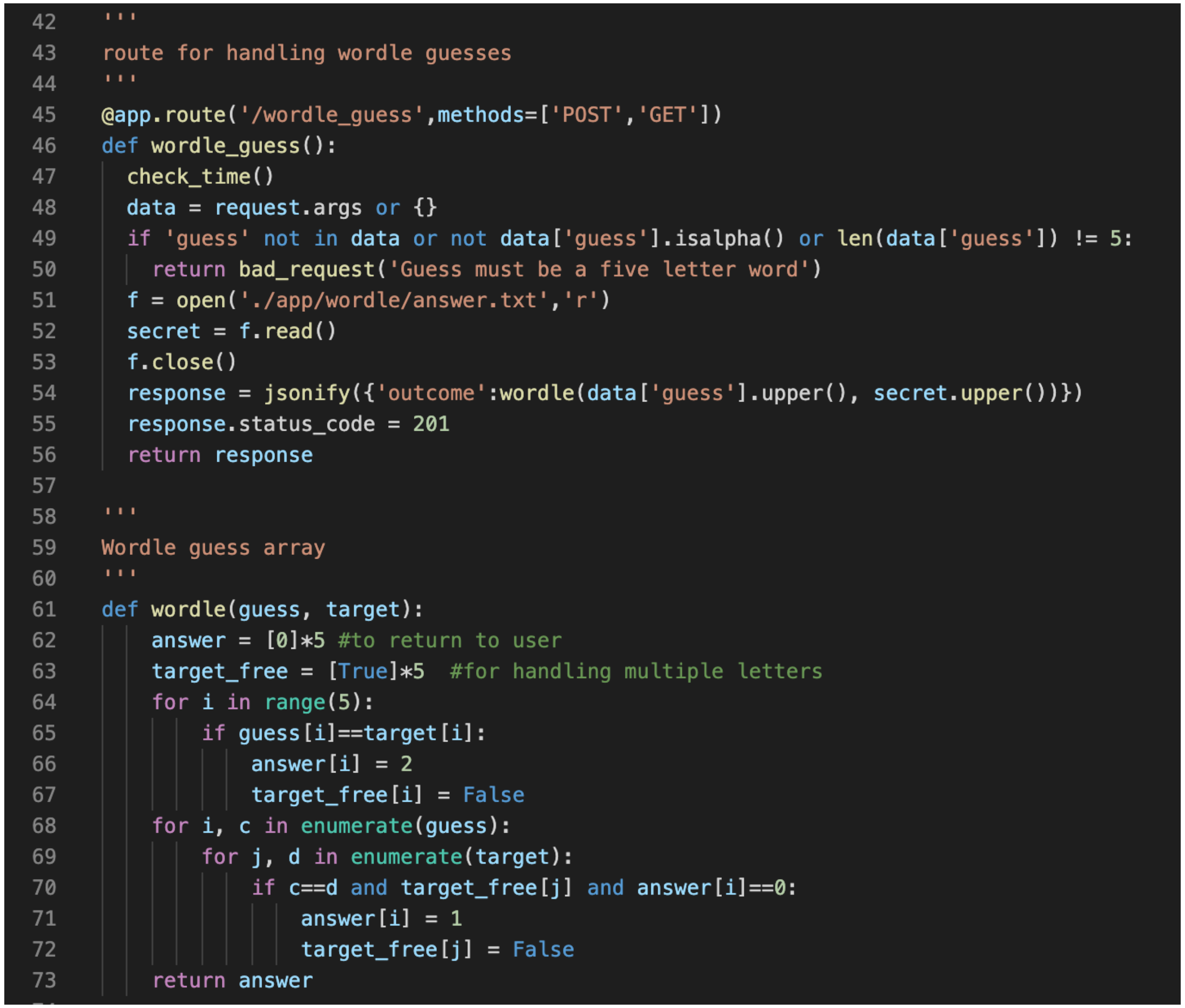
Making guess requests

- The second time we need to make a request to the server is when the user makes a guess.
- The server response tells us which letters are correct and which are misplaced.



```
else if(key == "Enter" && current_cell == 5 && current_guess<6){
  let guess="";
  for(let i = 0; i<5; i++){
    guess = guess + table_data[current_guess][i].innerHTML;
  }
  const xhttp = new XMLHttpRequest();

  xhttp.open("GET", "https://drtnf.net/wordle_guess?guess="+guess, true);
  xhttp.onload = function(e) {
    let result = JSON.parse(xhttp.responseText).outcome;
    let sum = 0
    for(let i = 0; i<5; i++){
      if(result[i]==2){
        sum+=result[i];
        table_data[current_guess][i].classList.add('correct');
      }
      if(result[i]==1){
        table_data[current_guess][i].classList.add('misplaced');
      }
    }
```

Responding to guess requests



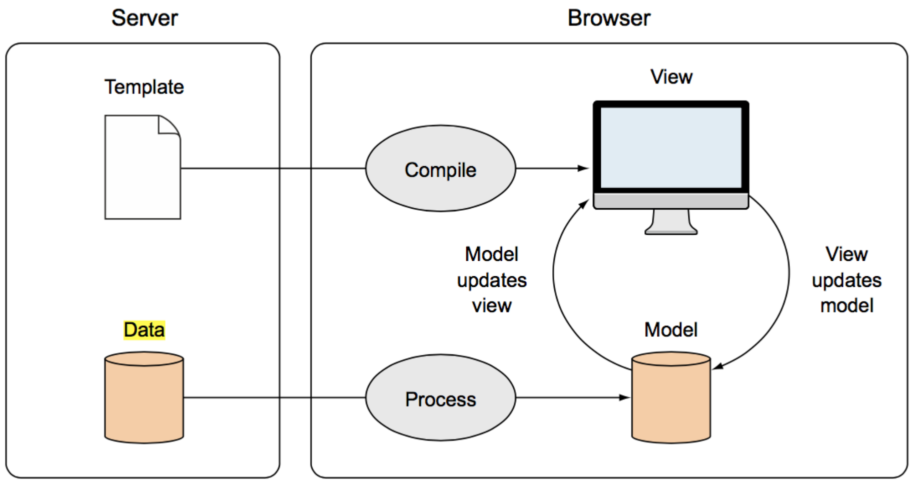
```
'''
route for handling wordle guesses
'''
@app.route('/wordle_guess',methods=['POST','GET'])
def wordle_guess():
 check_time()
 data = request.args or {}
 if 'guess' not in data or not data['guess'].isalpha() or len(data['guess']) != 5:
  return bad_request('Guess must be a five letter word')
 f = open('./app/wordle/answer.txt','r')
 secret = f.read()
 f.close()
 response = jsonify({'outcome':wordle(data['guess'].upper(), secret.upper())})
 response.status_code = 201
 return response

'''
Wordle guess array
'''
def wordle(guess, target):
  answer = [0]*5 #to return to user
  target_free = [True]*5  #for handling multiple letters
  for i in range(5):
    if guess[i]==target[i]:
      answer[i] = 2
      target_free[i] = False
  for i, c in enumerate(guess):
    for j, d in enumerate(target):
      if c==d and target_free[j] and answer[i]==0:
        answer[i] = 1
        target_free[j] = False
  return answer
```

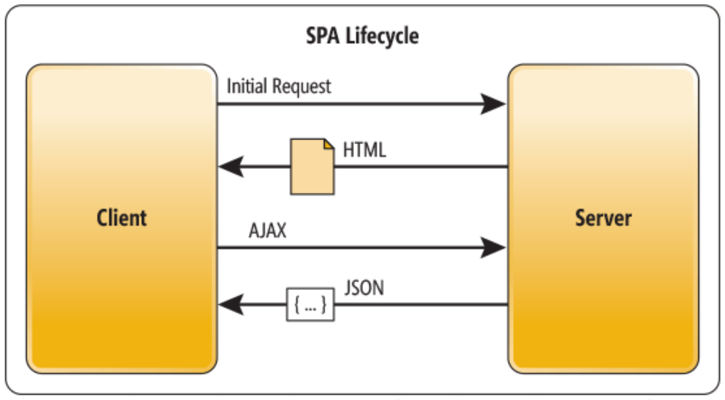
- When we receive a guess, we first check that the guess is valid and throw an appropriate error if not.
- We then consult the secret answer, compute the array of answers.
- Finally, we encode the answer array in a JSON object and send it as part of the response.

Single page applications

Single-page applications (SPA)

- Single Page Applications are services where the entire website is provided via client-side rendering:
  - The browser/client do the heavy lifting i.e. logic and rendering.
  - The server just provides the data.
  - The user never navigates to a new URL, even when they move to what looks like a new page.
  - The LMS is a great example.





Advantages and disadvantages

Pros of SPA

- Less load on the server, able to respond to more clients.
- A more responsive client. No need to wait for server responses.
- Genuine separation between content and presentation.

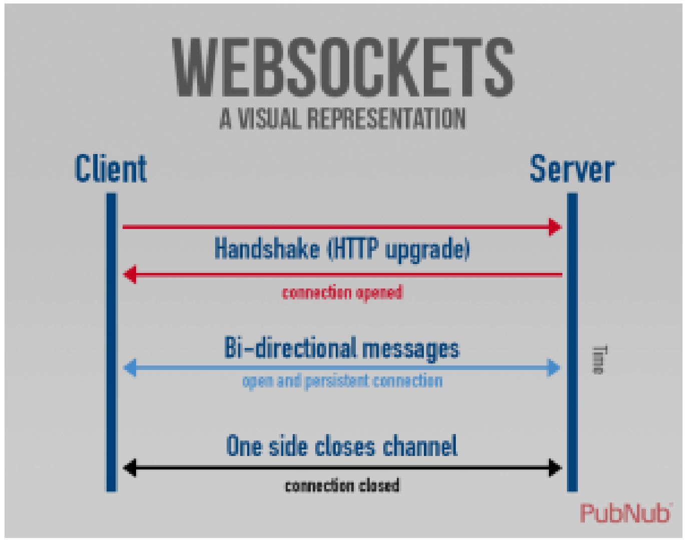
Cons of SPA

- Longer load time. A lot of JS must be transferred.
- Search engine optimisation (SEO) can be a problem. Robots won’t crawl JavaScript.
- Navigation (e.g. forward and back buttons) can be an issue.

Client-side rendering with web-sockets

Web-socket basics

- HTTP requests are useful for providing dynamic content but are heavy weight and expensive to setup.
- Many web applications depend on real time interaction.
- Web-sockets were standardised in 2011 to provide full duplex communication.
- Web-sockets allow your client-side JavaScript to open a persistent connection (stream) to the server.
- This allows real time communication in the application without having to send HTTP requests.



Web-sockets in Flask

- Web-sockets are supported in Flask via the `flask-socketIO` package

  https://flask-socketio.readthedocs.io/en/latest/

- SocketIO is good for message passing chat or distributed games.

- For direct video and audio, WebRTC can be used (peer-to-peer).

- Clients can connect to a socket on a server, and then the server can push messages to clients.

- The client has a *listener* architecture so it will respond to the push immediately.

Structuring a socket-based application

- Sockets mirror the routes architecture of a Flask project, but instead of listening for requests, they listen for messages and actions, and broadcast to all listening clients.

- The server works as a common blackboard for the session (or room) and the clients implement a listening architecture via jQuery.

- The socketIO architecture maintains rooms that users/processes can subscribe to.

- Clients and server interact by emitting events including *join, status, message,* and *leave.* You can also create customised events for clients to create and receive.

- We will follow a simple demonstration from Miguel Grinberg taken from: https://github.com/miguelgrinberg/Flask-SocketIO-Chat

Web-sockets on the server-side

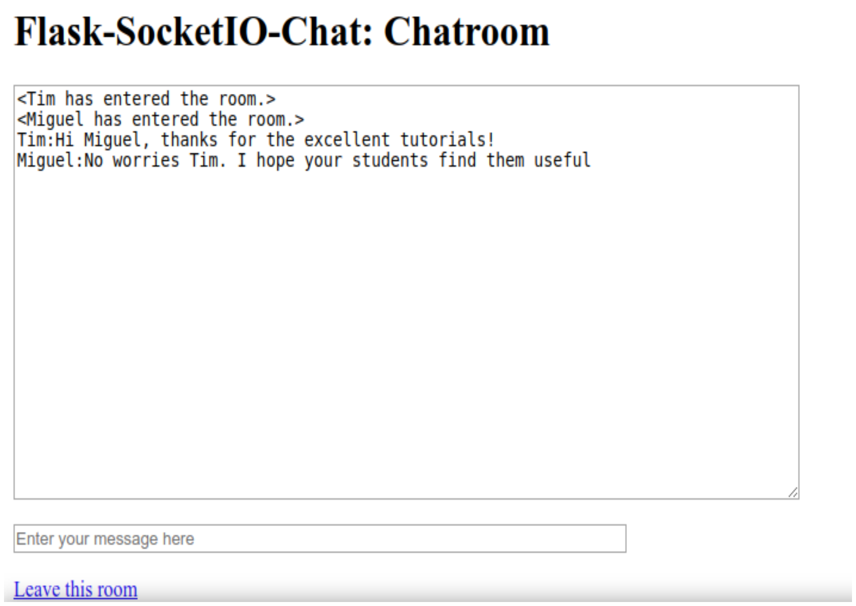
- We use a similar architecture. A main folder called *main*, containing a *forms.py* for registration, *routes.py* for handling login, and a *events.py* file for handling the socket events.

- The SocketIO package includes a decorator to match incoming messages with python methods.

```python
from flask import session
from flask_socketio import emit, join_room, leave_room
from .. import socketio


@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') + ':' + 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)

```

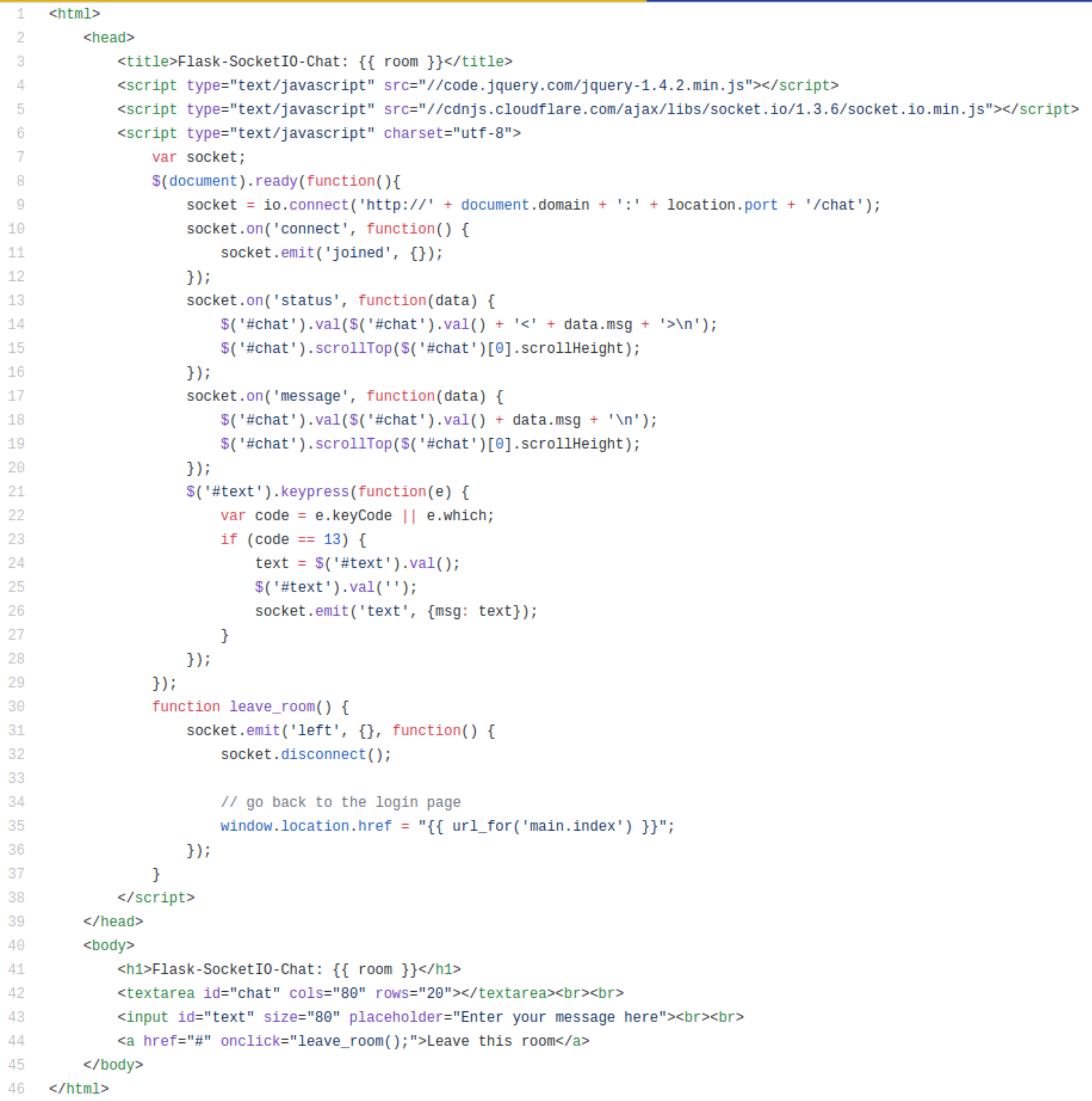
Web-sockets on the client-side

- We can use jQuery to send events to the server, listen for events coming from the server, and update the DOM accordingly.



Flask-SocketIO-Chat: Chatroom

```
<Tim has entered the room.>
<Miguel has entered the room.>
Tim:Hi Miguel, thanks for the excellent tutorials!
Miguel:No worries Tim. I hope your students find them useful
```



```
<html>
    <head>
        <title>Flask-SocketIO-Chat: {{ room }}</title>
        <script type="text/javascript" src="//code.jquery.com/jquery-1.4.2.min.js"></script>
        <script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/socket.io/1.3.6/socket.io.min.js"></script>
        <script type="text/javascript" charset="utf-8">
            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 + '>\n');
                    $('#chat').scrollTop($('#chat')[0].scrollHeight);
                });
                socket.on('message', function(data) {
                    $('#chat').val($('#chat').val() + data.msg + '\n');
                    $('#chat').scrollTop($('#chat')[0].scrollHeight);
                });
                $('#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();

                    // go back to the login page
                    window.location.href = "{{ url_for('main.index') }}";
                });
            }
        </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="Enter your message here"><br><br>
        <a href="#" onclick="leave_room();">Leave this room</a>
    </body>
</html>
```

Other applications of web-sockets

- Sockets can be used for distributing real time events such as real-time scoreboards or blogs, stock prices, weather etc.
- Implementing user-ids and sessions (next lecture) can allow you to have private chats between two users.
- Socket.io allows you to group sockets into namespaces and rooms, which allows you to control who can access and post messages.

```python
from flask_socketio import join_room, leave_room

@socketio.on('join')
def on_join(data):
    username = data['username']
    room = data['room']
    join_room(room)
    send(username + ' has entered the room.', room=room)

@socketio.on('leave')
def on_leave(data):
    username = data['username']
    room = data['room']
    leave_room(room)
    send(username + ' has left the room.', room=room)
```