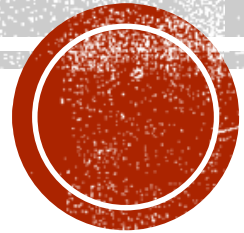
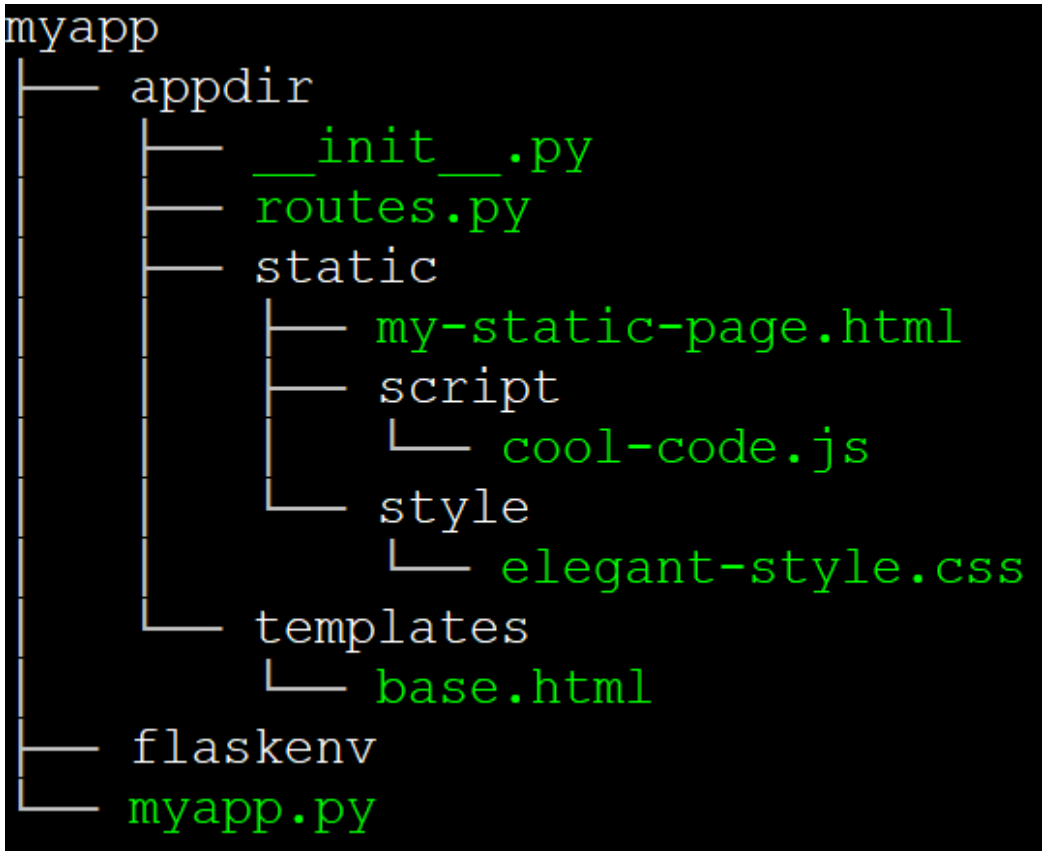
JQUERY & AJAX



ABOUT THE PROJECT - REMINDER

- Deadline: Last lab of term: 12-December-2019 5pm
- Technologies to use: Flask, Javascript, CSS, HTML
- Domain: Any domain of your choice
- How to submit:
  - All files in the following structure
  - Change file names to suit your website
  - But keep the same directory structure
  - Zip the top-level directory
  - Submit the zip file **only**



ABOUT THE PROJECT

- Grading rubric available on Moodle

- Referencing! Referencing! Referencing!
  - Code from *any* source (including code shown in class) must be referenced
  - Failure to reference will be considered plagiarism

- I will put up an interview schedule ***after*** submissions are over. Interviews will be held with myself and the head TA, in the shamrock / exchange room [will be notified in the interview schedule]

PROJECT EVALUATION

- I will extract your zip file into ***my*** `venv`
- I will run the following commands:

```
your-proj-dir> source flaskenv/bin/activate
(flaskenv)your-proj>export FLASK_APP=your-proj.py
(flaskenv)your-proj>flask shell
>>>from yourproj import db
>>> db.create_all()
```

This will create the database according to *your* schema. Your project should run after this.

AJAX

Asynchronous JavaScript with XML

Asynchronous JavaScript And XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser.

AJAX is very convenient, because you can:

- Update a web page without reloading the page
- Request / receive data from a server - after the page has loaded
- Send data to a server - in the background
- Use both XML or JSON to send data back-and-forth [we will use JSON]

AJAX

AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

JQUERY SYNTAX

$(*selector*).*action*()

- Selectors
- Actions: load(), off(), on()….

AJAX

GET Requests – formal definition

jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**)

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object.*
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.
  - **data** holding the body of the response as a string.
  - **textStatus** holding the status of the request (i.e., "success").
  - **jqXHR** holding a jqXHR (jQuery XMLHttpRequest) object, described shortly.
- **dataType** is an optional parameter to hold the type of data expected from the server.

AJAX

GET Requests – an example

```
$.get("/profile", { name:"Donald", town:"Ducktown" });
```

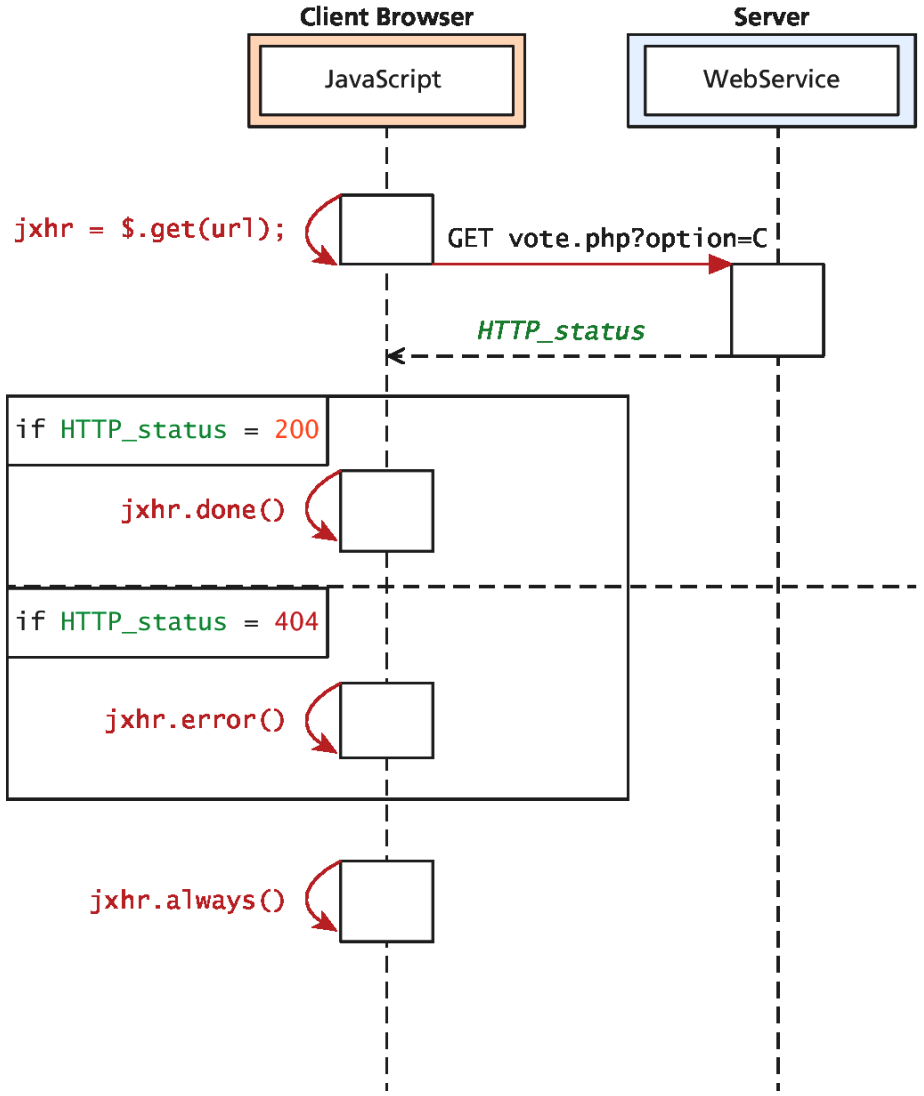
```
$.get("/fav", { 'colors[]' : ["Red","Green","Blue"] });
```

Other AJAX detail:

https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

JQXHR

Actually quite easy to use



jqXHR objects have methods

- done()
- fail()
- always()

which allow us to structure our code in a more modular way than the inline callback

```
$.ajax({
    type: {POST or GET or PUT etc.},
    url:  {server.url},
    data: {someData: true},
    statusCode: {
        404: function(responseObject, textStatus, jqXHR) {
            // No content found (404)
            // This code will be executed if the server returns a 404 response
        },
        503: function(responseObject, textStatus, errorThrown) {
            // Service Unavailable (503)
            // This code will be executed if the server returns a 503 response
        }
    }
})
.done(function(data){
    alert(data);
})
.fail(function(jqXHR, textStatus){
    alert('Something went wrong: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
   alert('Ajax request was finished')
});
```

POST REQUESTS

Via jQuery AJAX

POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.

GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.

With POST it is possible to transmit files, something which is not possible with GET.

POST REQUESTS

Via jQuery AJAX

If we were to convert our vote casting code it would simply change the first line from

```
var jqxhr = $.get("/vote?option=C");
```

to

```
var jqxhr = $.post("/vote", "option=C");
```

```
var jqxhr = $.post("/demo_test_post",
    {
        name: "Donald Duck",
        city: "Duckburg"
    });
```

POST REQUESTS

Serialize() will seriously help

serialize() can be called on any form object to return its current key-value pairing, suitable for use with post().

```
var postData = $("#voteForm").serialize();

$.post(“/vote", postData);
```

LET'S ADD IT TO THE MICROBLOG

- When the user comes to the signup page, the page should use ajax to check if the username already exists.

- Four files that we will change:
  - base.html – add jquery to the <script> section of the template
  - routes.py – create a route for the ajax code to call [`/checkuser`]
  - myjs.js – register an event handler to see when the username field changes
  - mystyle.css – add two classes to show success/failure

BASE.HTML

```
<head>
    {% if title %}
    <title>{{ title }} - microblog</title>
    {% else %}
    <title>microblog</title>
    {% endif %}
    <link rel="stylesheet" href="{{ url_for('static', filename='style/mystyle.css') }}">
    <script src="{{ url_for('static', filename='scripts/jquery.js') }}"></script>
```

routes.py

```
@app.route('/checkuser', methods=['POST'])
def check_username():
    chosen_name = request.form['username']
    user_in_db = User.query.filter(User.username == chosen_name).first()
    if not user_in_db:
        return jsonify({'text': 'Username is available',
                        'returnvalue': 0})
    else:
        return jsonify({'text': 'Sorry! Username is already taken',
                        'returvalue': 1})
```

myjs.js

- Register an event for handling the username field

```
$(document).ready(function(){
    // add all event handlers here
    console.log("Adding event handlers");
    $("#username").on("change", check_username);
    console.log("function registered");
});
```

myjs.js

```
function check_username(){
    // get the source element
    console.log("check_username called");
    var chosen_user = $(this).find("input");
    console.log("User chose: " + chosen_user.val());

    $("#checkuser").removeClass();
    $("#checkuser").html('<img src="static/style/loading.gif")>');
```

myjs.js - II

```
// ajax code
$.post('/checkuser', {
    'username': chosen_user.val() //field value being sent to the server
}).done(function (response){
    var server_response = response['text']
    var server_code = response['returnvalue']
    if (server_code == 0){ // success: Username does not exist in the database
        $("#checkuser").html('<span>' + server_response + '</span>');
    }else{ // failure: Username already exists in the database
        $("#checkuser").html('<span>' + server_response + '</span>');
    }
}).fail(function() {
    $("#checkuser").html('<span>Error contacting server</span>');
});
// end of ajax code
```

mystyle.css

```
.success {
    color: green;
}

.failure {
    color: red;
}
```

EXAM PREP TIPS!

FEEDBACK

- UCD operates an all-university, on-line anonymous feedback system
- Allows you to express your opinion about how the module went
- Helps us understand how to improve
- Go to: www.ucd.ie/survey

TIME DISTRIBUTION

- 100 marks in 120 minutes

- Spend 110 minutes answering the questions
- Spend 10 minutes reviewing your answers

- On an approximate basis, spend 1 minute per mark. So, a 3-mark question should take between 3 - 3.5 minutes.

EXAM FORMAT

- Questions in two sections:
  - Short Questions
  - Program Comprehension

TYPES OF QUESTIONS – SHORT QUESTIONS

- Theory questions
  - Explain X
  - What is the difference between X and Y

TYPES OF QUESTIONS – PROGRAM COMPREHENSION

- What does this code do?
  - Consider the following code …
  - Given the Javascript code below …
  - Take a look at the HTML given …

EXAMPLE

c. Write a function in Javascript to convert a date from DD/MM/YYYY format to YYYYMMDD format.

```
function convertDate(oldDate){
return [oldDate.slice(6),oldDate.slice(3,5),
oldDate.slice(0,2)].join('');
}
```

```
function convertDate(oldDate){
        let reg = /^(\d{2})\/(\d{2})\/(\d{4})/;
        let match = oldDate.match(reg);
        return match[3]+match[2]+match[1];
}
```

Good luck for the exams!

下课