INFO3180 - Tutorial 4

Flask, JSON, JQuery and localStorage

In this tutorial we will look using Flask to serve JSON responses. We will also look at the javascript library called jQuery and it's basic usage. This understanding is very important as it lays the foundations for interacting with common web Application Programming Interfaces (APIs) as well as creating your own.

We will explore how Javascript's built in **`localStorage`**, **`JSON.stringify`** and **`JSON.parse`** are commonly used together when working with APIs.

jQuery 101

jQuery is a javascript library designed to make DOM manipulation, event handling, animation and working with AJAX simpler. The library can be downloaded from http://jquery.com

Key features of jQuery

Selecting elements is done using the same syntax as CSS Selectors (using jQuery is comparable to using **`document.querySelectorAll()`**. jQuery provides some convenience methods such as **`getJSON()`** (which we will use later). jQuery also provides the **`jQuery`** object, for short it can be referenced using the **`$`** symbol.

Discussion

jQuery is a javascript library. How would you add jQuery to your own project? Discuss the difference between using a Content Delivery Network (CDN) to include the JQuery library and downloading and embedding the JQuery library.

Some quick jQuery Examples

Visit the front page of jquery.com and look at the quick examples in the "A Brief Look" section. You can try most of them out simply by opening the console of your browser at http://jquery.com.

`$(document).ready()` is a common jQuery pattern used to run the code within the ready function only after the page has completely loaded. It is similar to the **`window.onload`** event in vanilla JavaScript.

```
$( document ).ready(function() {
   console.log( "ready!" );
});
```

Note: Remember **`$`** is the same as **`jQuery`**

The jQuery object can be called using **`jQuery`** instead of **`$`**.

e.g.

```
jQuery( document ).ready(function() {
   console.log( "ready!" );
});
```

Javascript, JSON and localStorage

JSON: Javascript provides a object called "JSON" for working with JSON content. It makes it possible to convert JSON into a string or a string into JSON using the methods: **`JSON.stringify()`** and **`JSON.parse()`**.

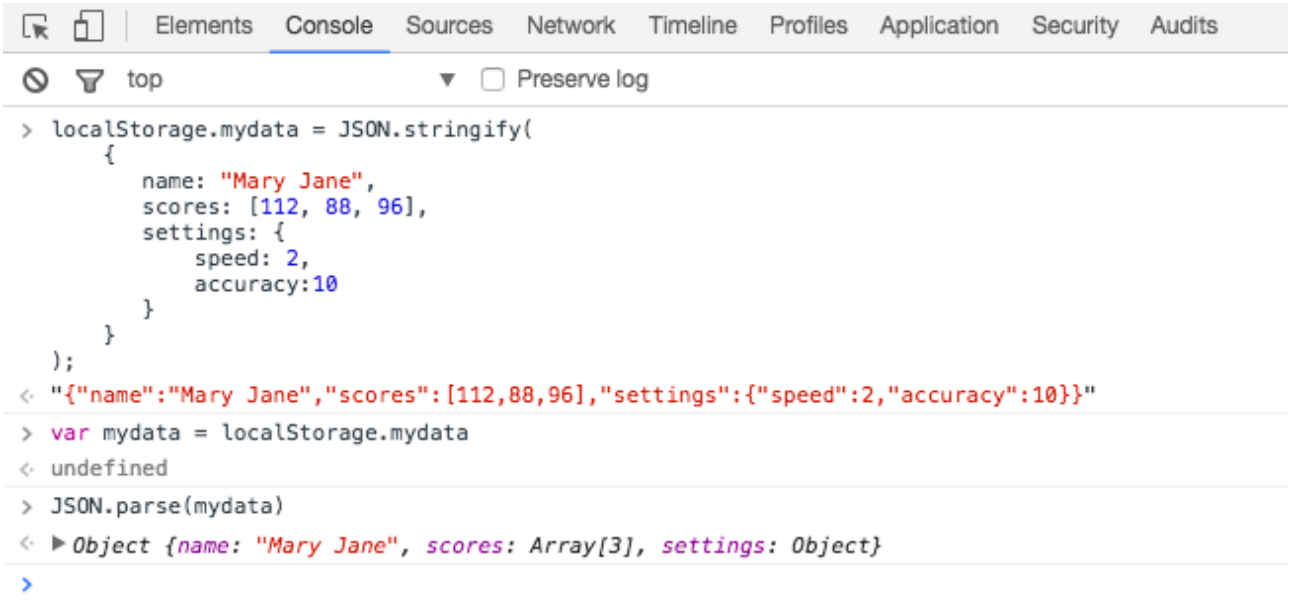
localStorage/sessionStorage: Javascript provides the **localStorage** and **sessionStorage** objects for persisting data in your client (your web browser). **sessionStorage** is similar to **localStorage**, the only difference is while data stored in **localStorage** has no expiration set, data stored in **sessionStorage** gets cleared when the page session ends. A page session lasts for as long as the browser is open and survives over page reloads and restores. Opening a page in a new tab or window will cause a new session to be initiated, which differs from how session cookies work.

In the developer tools console of your browser or in a standalone .js file try the following:

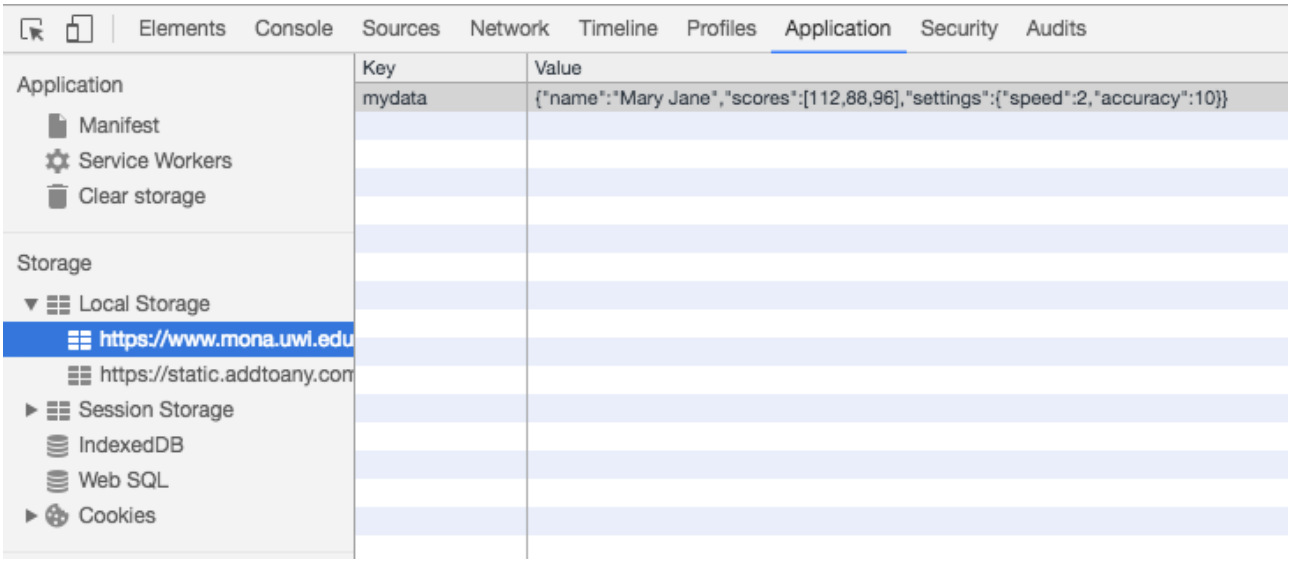
```
localStorage.mydata = JSON.stringify(
    {
       name: "Mary Jane",
       scores: [112, 88, 96],
       settings: {
           speed: 2,
           accuracy:10
       }
    }
);

var mydata = localStorage.mydata
JSON.parse(mydata)
```

The screenshot below shows an interactive session on the Chrome devtools console



Under the "Application" tab you should now see your data stored as Local Storage



Discussion

1. Why do we refer to the web browser as the client?
2. What are the two major client side APIs that we've looked at so far?
3. Would you be able to distinguish clearly between a server provided API and the client APIs we've looked at?

JSON output from Flask

In a previous tutorial we briefly looked at outputting JSON from Flask. A basic example of generating JSON from Flask is discussed in the Flask documentation.

Flask provides a method called '**jsonify**' which returns JSON formatted output. We've provided a simple example based on our flask_starter template at https://github.com/uwi-info3180/flask-json

This is what the '**views.py**' file looks like from the flask_json example:

```
from app import app
from flask import render_template, request, redirect, url_for
from flask import jsonify, session

username = "ironman"
email = "tony.stark@avengers.com"
id_number = "999123"

##
# Routing for your application.
##
@app.route('/get-current-user')
def get_current_user():
    return jsonify(username=username, email=email,
id_number=id_number)

@app.after_request
def add_header(response):
    """
```

```
    Add headers to both force latest IE rendering engine or Chrome
Frame,
    and also to cache the rendered page for 10 minutes.
    """
    response.headers['X-UA-Compatible'] = 'IE=Edge,chrome=1'
    response.headers['Cache-Control'] = 'public, max-age=600'
    return response

@app.errorhandler(404)
def page_not_found(error):
    """Custom 404 page."""
    return render_template('404.html'), 404

if __name__ == '__main__':
    app.run(debug=True,host="0.0.0.0",port="8888")
```

Visiting the '/get-current-user' URL would result in the following:

```
{
  "email": "tony.stark@avengers.com",
  "id_number": "999123",
  "username": "ironman"
}
```

You can imagine that this output might be the data from a user profile. In the next section, we'll see how javascript can interact with this JSON output.

Note:

In some browsers this may trigger a download of the resulting json. Why would this happen? **Ans.** If the web browser does not understand the **`application/json`** Content-Type it will prompt the user to download it instead of displaying it in the browser.

This example is hard coded and probably not very useful as is, except for teaching the concepts of interacting with a JSON response.

Using jQuery to interact with the JSON output

Let's create a route "**`/jquery-get-json`**" that maps to it's respective **`get_data()`** method in our Flask app which uses jQuery to interact with our **`get_current_user`** view. Also create a file in your Flask **`templates`** folder, name the file **`jquery-get-json.html`**. We're creating this so that we have access to the jQuery library.

```
# Home page route
@app.route("/jquery-get-json")
def get_data():
    return render_template('jquery-get-json.html')
```

jquery-get-json.html template file

```
<html>
<head>
  <title>Get JSON using jQuery</title>
  <script type="text/javascript" src="{{ url_for('static',
filename='jquery.js') }}"></script>
</head>
<body>
<div class="container">
<span id="title">Hello World</span>
```

```
</div>
</body>
</html>
```

Note:

This template uses '**`url_for`**' and assumes that there is a file in the static folder called **`jquery.js`**. Make sure that there is one in that location. You could just as easily have written the script tag like this: **`<script type="text/javascript" src="/static/jquery.js"></script>`**

You can quickly download the jQuery file from the website, just remember the following

1. Place it in the static folder of your Flask app
2. Rename it to jquery.js

What if you want to parse this output using javascript?

If you're working with jQuery it is possible to do the following:

```
<script>
$.getJSON('/get-current-user', function(data) {
    alert(data);
});
</script>
```

Note: You could also have done something similar with the jQuery **`$.ajax()`** method. Though there are some slight differences in the way it is used.

Discussion

How could we take the output of the "**`get_current_user`**" method and save the information to your client using javascript?

Try the following code in a console.

1. Visit the application page we just created (e.g. /jquery-get-json)
2. Then open your browser's developer tools console
3. Try using the **`$.getJSON`** to retrieve the output of the **`get_current_user`** route method.
4. The code currently returns an alert. Change the code to convert the JSON to a string instead, **Hint:** use **`JSON.stringify`**
5. Then store the "stringified" output to localStorage.

Discussion

Why do we "stringify" before storing to localStorage?

How could the techniques explored above be useful when building an application?