LECTURE 33

POST AND FORMS

MCS 275 Spring 2023

Emily Dumas

LECTURE 33: POST AND FORMS

Reminders and announcements:

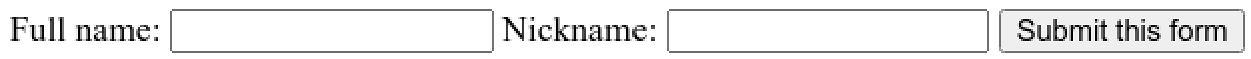
- Project 4 is due 6pm CDT Friday 28 April.
- Today is the last day to submit initial application for a non-SQLite project 4 topic.

WEB APP TODO LIST

- ☑ HTML mockup
- ☑ Stylesheet
- ☑ Learn a bit about Flask
- ☑ Database schema & test data
- ☑ Python code to generate worker view HTML from a database query
- ☐ Add page to create new work order
- ☐ Make buttons on worker view page work

FORMS

Interactive elements in an HTML document (text entry, checkbox, dropdown list, etc.)



```
<form action="https://example.com/formsub/">
    <label for="full">Full name:</label>
    <input type="text" id="full" name="full">
    <label for="nick">Nickname:</label>
    <input type="text" id="nick" name="nick">
    <input type="submit" value="Submit this form">
</form>
```

jsfiddle is a nice way to test out form designs (for code that can be public).

INPUTS NAME VS ID

Each form input should have both a `name` and `id` attribute. Usually they are equal, but they have separate roles:

- `name` is what this value is called when submitted to the server.
- `id` is used to match an input with its `<label>`.

TEXTAREA

`<input type="text">` is typically for single-line answers.

Longer text entry (multi-line) should be handled with a `<textarea>` tag.

HTTP REQUEST TYPES

`GET` - load a resource, the only action we've considered so far.

`GET` requests are supposed to be idempotent, meaning repeating the same request multiple times has the same effect as doing it once.

HTTP REQUEST TYPES

`POST` - submit data and/or request an action.

`POST` requests are not expected to be idempotent. Browsers typically prevent reloading a `POST` request, for example.

By default, forms use a `GET` request and put form data in the URL.

This is usually a bad idea, and a `POST` request is more appropriate.

Easy change: Add `method="post"` attribute to the `<form>` tag.

WHAT FORM GET REQUEST LOOKS LIKE

Form values become **query parameters**, e.g.

```
https://example.com/formsub/?full=Emily%20Dumas&nick=emmy
```

Many ascii characters appear verbatim but others* become `%` escape sequences with two hex digits. Flask decodes these and makes the parameters available as `flask.request.values.get(name)`.

* The precise encoding scheme is specified in RFC3986. Python's built-in `urllib.parse` module has

functions that perform this type of encoding/decoding: `urllib.parse.quote` and `urllib.parse.unquote`. When using Flask, you usually won't call these directly.

Form values are made available to the function handling submission through `flask.request.values.get(name)`.

Note that a Flask route must explicitly declare that it accepts POST requests:

```
from flask import Flask, request

# ... app setup ...

@app.route('/registernick',methods = ['POST', 'GET'])
def record_fullname_and_nickname():
    print("Received nickname {}".format(
        request.values.get("nick")
    ))
```

FLASK FUNCTIONS

All are in the `flask` module:

- **`redirect(url)`** - Returning this object from a route will cause the HTTP server to issue a 302 response code, telling client to load `url` instead.
- **`abort(http_error_code)`** - Immediately stop and return a HTTP error code (usually 400 bad request, 401 not authorized, 403 forbidden, or 404 not found).

ROUTES

- **`/worker/<name>/`** - (GET) worker's view of orders
- **`/new/`** - (GET) form for new order
- **`/new/submit`** - (POST) form submission destination
- **`/wo/<int:woid>/`** - (GET) work order status
- **`/wo/<int:woid>/assign_to/<str:name>/`** - (GET*) assign work order to user
- **`/wo/<int:woid>/unassign/`** - (GET*) unassign work order
- **`/wo/<int:woid>/complete/`** - (GET*) mark work order complete

* These should really be POST but we would need to use javascript or a different button markup to do it because `<a>` tags generate a GET request.

REFERENCES

- jsfiddle - Write and test HTML+CSS quickly in browser
- HTML tutorial from w3schools
- CSS tutorial from w3schools
- The Flask tutorial

REVISION HISTORY

- 2022-04-13 Last year's lecture on this topic finalized
- 2023-04-10 Updated for 2023