LECTURE 34

WEB APP WRAP-UP

MCS 275 Spring 2023
David Dumas

LECTURE 34: WEB APP WRAP-UP

Reminders and announcements:

- Work on Project 4.
- Project 4 is due 6pm CDT Friday 28 April
- Custom (non-SQLite) topics: Must have requested already, must receive approval by Friday
- Autograder opens Monday 24 April.

TODAY

This is the last in our contiguous lecture series focused on writing a Flask+SQLite application.

(We may revisit this topic a bit in the last week.)

WEB APP TODO LIST

☑ HTML + CSS mockups
☑ Database
☑ Flask application with worker view
☑ New work order form
☑ Activate "take assignment" button
☐ Activate other buttons
☐ Style the submission form
☐ Date/time formatting
☐ Work order status page
☐ Make all actions redirect to natural destinations
☐ DB initialization and connection cleanup
☐ Detect and handle errors (e.g. failure to take assignment)

WEB APP TODO LIST

- ☑ HTML + CSS mockups
- ☑ Database
- ☑ Flask application with worker view
- ☑ New work order form
- ☑ Activate "take assignment" button
- **☑ Activate other buttons**
- **☑ Style the submission form**
- ☐ Date/time formatting
- ☐ Work order status page
- ☐ Make all actions redirect to natural destinations
- ☐ DB initialization and connection cleanup
- ☐ Detect and handle errors (e.g. failure to take assignment)

WEB APP TODO LIST

- ☑ HTML + CSS mockups
- ☑ Database
- ☑ Flask application with worker view
- ☑ New work order form
- ☑ Activate "take assignment" button
- ☑ Activate other buttons
- ☑ Style the submission form
- **☑ Date/time formatting**
- **☑ Work order status page**
- **☑ Make all actions redirect to natural destinations**
- **☑ DB initialization and connection cleanup**
- **☐ Detect and handle errors (e.g. failure to take assignment)**

DB CHANGES

`createdb.py` is now a module, not just a script.

It has functions to create or clear the tables, and to add sample data.

It also now sets the database filename used everywhere.

Main web app script imports it and uses it to make sure database exists before proceeding.

DATE/TIME HANDLING

Added `timefmt.py`, module defining:

- **`ts_fmt(x)`** - Given a timestamp x, return a string like `"12:53pm on April 28, 2023"`
- **`tsdiff_fmt(t)`** - Given a number of seconds `t` since something happened, return a string like `"38 seconds ago"` or `"94 days ago"` etc.

Worker view uses this to display times. Entries in the list are sorted by how long they've been in their current state (oldest first).

WORK ORDER STATUS PAGE

Now `/wo/5/` gives you a page with status info about work order 5.

New template, new route in the main python program.

If the work order is assigned, this page links to the worker view for the assigned worker.

AFTER WORK ORDER CREATION

We redirect from `/new/submit` to the status page.

But how?

We just `INSERT`ed it so we don't know the `woid`.

LAST INSERTED ROW

After a single-row `INSERT`, how to get the primary key of the new row?

```
SELECT last_insert_rowid();
```

Implicitly refers to the most-recently executed `INSERT` on this connection.

LINKING TO STATUS PAGES

Any time a work order number appears on the worker view, the number itself is a link to the associated status page.

FORM STYLING

Larger text entry box for the description. No CSS needed!

The `size` attribute of an `input` of type `"text"` sets the width in characters.

SUMMARY: 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/<name>/`** - (GET*) take assignment
- **`/wo/<int:woid>/cancel/<name>/`** - (GET*) cancel assignment
- **`/wo/<int:woid>/complete/<name>/`** - (GET*) mark complete

* These should really be POST but we would need to use javascript or a different button markup to do it.

NEAR MISSES

Natural things we could do, but won't:

- Store work order creation username in DB
  - Fix would break existing DB files
- Admin page
  - Too similar to worker view to justify discussion

DETECT ERRORS?

We perform database queries. They might fail for some reason and raise an exception.

But we also expect some of these queries to change exactly one row. We don't check that.

NUMBER OF UPDATED ROWS

An UPDATE might match any number of rows (e.g. 0, 1, 50). The query

```
SELECT changes();
```

returns the number of rows changed by the last UPDATE on this connection.

HARD-CODED URLS

Right now, our templates contain explicit reference to the URLs our application uses.

These are also declared in the Python source (`@app.route(...)`).

It would be nice to have them in only one place, for ease of change or maintenance.

FLASK'S URL BUILDER

`flask.url_for(function_name)` - convert a function name into the URL that triggers it.

ABORT

`flask.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).

NOT-SO-NEAR MISSES

Some of the things you'd do differently in a "real" application:

- **Action history**: We have a single column for WO creation time. We should probably log every action that changes a work order in a separate table.
- **Accounts, roles, cookies**: login page checks credentials against DB, sets browser *cookie*. Auth-required pages check for it, redirect to login page if not found.
- **JavaScript**: e.g. to check for new messages in real time, post new message without loading a new page, make buttons perform POST requests without forms.
- **Pagination**: Limit number of orders we show on the main worker view. Links let you view the next or previous page.
- **Search**: Find a work order by date range, by words in description, etc.
- **Deploy**: Domain name, server, ...

REFERENCES

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

REVISION HISTORY

- 2022-04-15 Last year's lecture on this topic finalized
- 2023-04-12 Updated for 2023