Objective:

This week's iteration builds on our existing Flask web development skills and involves working through additional database interactions.

Feel free to update your database or previously created application screens during any step as you work through the assignment. For example, you might add a notification capability to the dashboard screen so that the dashboard can be used to tell the user about records not inserted due to duplication or other captured exception cases. Or you may need to make some additional changes to the data model as you work through how to make the report screen.

Step 1 - Start the feature branch

To keep the master branch in a clean and working state, it is a good habit to move your active development into a separate branch.

- Create a new branch 'assignment7':

  ```
  git branch assignment7
  ```

- Change to the branch:

  ```
  git checkout assignment7
  ```

- Push the branch to GitHub:

  ```
  git push origin assignment7
  ```

All of the assignment 7 work will be done in the assignment7 branch.

-- Feature branches are just what we do - 0pts

Step 2 - Updates to preflight for mod_wsgi

It is much easier for us to test your code if it is working with mod_wsgi. If you have not been using mod_wsgi, you will need to start this week. Your preflight script will need to copy the contents of your src directory to $HOME/wsgi.

If needed, add to preflight.sh

```
cp -R src/* $HOME/wsgi
```

To start the webserver use:

```
apachectl start
```

To restart the webserver after copying over changes use:

```
apachectl restart
```

The tracebacks that used to appear on your screen when running app.py directly will appear in

`$HOME/logs/error_log`
. You can use the 'cat' command or 'tail' commands to view these tracebacks.
-- low water mark - 0pts

Step 3 - Update the data model

The new functionality will require additional data to be kept in the database. The create_tables.sql will need to be updated.

This week we need to add a notion of roles to the users. A user can have the role of 'Logistics Officer' or 'Facilities Officer'.

Do you need a new table for roles? If so, decide on what the roles table should look like and how it should be connected to the users table. Does the users table also need to be updated? Make those changes too. Include comments in your create_tables.sql to indicate how you are handling roles.

This week we also need to start tracking assets and facilities. An asset minimally has:

- an asset tag, up to 16 characters in length.
- a description of arbitrary length.

A facility minimally has:

- a facility common name, up to 32 characters in length (e.g. Headquarters)
- a facility code, up to 6 characters in length (e.g. HQ)

Add tables to hold assets and facilities.

For this week assets are either at a specific facility or marked as disposed. Assets may be in transit in future iterations. Come up with a strategy to represent where and when an asset is located using the database (probably involves an additional table). Put a

little thought into this decision since upcoming reporting requirements will involve being able to know where the asset is currently located and where the asset has been historically. Include comments in your create_tables.sql to indicate how you are mapping assets to facilities.

Commit and push your code.

-- Database dependance - 8pts (2 role representation, 2 assets table, 2 facilities table, 2 connecting assets and facilities)

Step 4 - Update create user screen

Now that user roles are part of the system, the create user screen needs to be updated to include roles. Update the form to also allow the role to be set. Update how the POST is handled so that the role is correctly set for the user.

Test your updated create user screen using your web browser. Use psql to verify that the create user screen make the right changes to the data in the database.

Commit and push your code.

-- Minimum application code -- 2pts (all or nothing)

Step 5 -- Facility Add Screen

Since assets are generally located at facilities, facilities should probably be added before adding assets.

- Add a route for '/add_facility' to app.py
- If the request is a 'GET' type request:
    - A template with a form element that allows a facility common name, fcode, and any other facility attributes to be entered by the user should be rendered.
    - The form method should be POST and the action should go to the /add_facility route.
    - The template should also list all of the facilities currently in the database, preferably before the form element.
- If the request is a 'POST' type request:
    - The facility attributes should be read from the form.
    - The facility code and common name should be looked up in the database, if a record matches the fcode or common name:

            - The user should be redirected to a screen indicating the facility is a duplicate and will not be added.
    - A new facility record should be inserted into the database with the information from the form.
    - The user should be redirected to the /add_facility route.

Test your code using your web browser. Commit and push your code.

-- Functionality A -- 2 pts (all or nothing)

Step 6 -- Asset Add Screen

An interface is needed to add assets to be tracked.

- Add a route for '/add_asset' to app.py
- If the request is a 'GET' type request:
    - A template with a form element that allows an asset tag, asset description, and any other asset attributes to be entered by the user should be rendered.
    - The user should select the facility the asset is located at from a drop down (HTML select).
    - The user should enter a date for when the asset arrived at the facility.
    - The form method should be POST and the action should go to the /add_asset route.
    - The template should also list all of the assets currently in the database, preferable before the form element.
- If the request is a 'POST' type request:
    - The asset attributes should be read from the form.
    - If the asset tag matches an asset tag already in the database:
        - The user should be redirected to a screen indicating the asset is a duplicate and will not be added.
- A new asset record should be inserted into the database with the information from the form, including linking to the facility and storing of the arrival date.
- The user should be redirected to the /add_asset route.

Test your code using your web browser. Commit and push your code.

-- Functionality B - 2pts (all or nothing)

Step 7 -- Asset Dispose Screen

Assets must eventually be disposed of. A screen is needed for that and asset disposal is access controlled.

- Add a route for '/dispose_asset' to app.py
- Use the session information to lookup the user's role
- If the user is not a logistics manager:
    - redirect the user to a screen indicating only logistics managers can dispose of assets.
- If the request is a 'GET' type request:
    - A template with a form element to enter an asset tag and date should be rendered for the user.
    - The form method should be POST and the action should go to the /dispose_asset route.
- If the request is a 'POST' type request:
    - The database should be checked for an asset tag matching the one entered by the user.
    - If there is no matching asset tag:
        - redirect the user to a screen indicating the asset does not exist
- If there is a matching asset tag but it was disposed:
    - redirect the user to a screen indicating the asset was already disposed
- Update the database to note that the asset was disposed of (is not at any facility) on/after the date entered.
- redirect the user to the /dashboard route.

Test your code using your web browser. Commit and push your code.

-- Functionality C - 2pts (all or nothing)

Step 8 -- Report Screen

This is the major report for the application.

- Add a route for '/asset_report' to app.py
- If the request is a 'GET' type request:
    - A template with form elements for selecting a facility and date should be provided.
    - The facility may be blank, indicating all facilities are to be included in the report.
    - The date may not be blank and indicates the date of interest for the asset report.

    - The form method should be POST and the action should go to the /asset_report route.
- If the request is a 'POST' type request:
    - A template with form elements for selecting a facility and date should be provided.
    - The facility may be blank, indicating all facilities are to be included in the report.
    - The date may not be blank and indicates the date of interest for the asset report.
    - The form method should be POST and the action should go to the /asset_report route.
    - A database query should be run using the filter criteria received in the POST.
        - If a specific facility was selected, only assets at the user selected facility on the user selected day should be shown.
        - If no specific facility was selected, assets at any facility on the selected day should be shown.
    - Following the form, a report/table should be displayed where each row minimally provides the asset tag, description, facility, arrival date, and disposal date of each asset matched by the filter.

Test your code using your web browser. Commit and push your code.

-- Complete functionality - 4pts (all or nothing)

Step 9 -- Merge the feature branch

With the iteration complete, merge the work back into the master branch.

```
git checkout master
git merge assignment7
git push
```

Use the GitHub web interface to verify that the expected files are present.