Objective

This week's iteration adds an enforced workflow to the application. Feel free to update your database or previously created application screens during any step as you work through the assignment.

Step 1 - Start the feature branch

Create a new branch 'assignment8'. Change to the branch and push the branch to GitHub. Look at last week's assignment if you have forgotten the commands. All assignment8 work will be done in the assignment 8 branch.

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

Step 2 - Update the data model

The data model will need to be changed to support enforcing the approval process for asset transfers. You will likely need to look ahead at the steps regarding the screens to build a good understanding of what data model changes are needed. A table will likely be needed to track transfer requests. A transfer minimally has:

- A requester, a logistics officer submitting the request.
- The date and time the transfer request was submitted.
- The source facility.
- The destination facility.
- The asset to be transfered.
- An approver, a facilities officer approving the transfer request.
- The date and time the transfer request was approved.

A table will likely also be needed to track assets in transit. An asset in transit is minimally modeled by:

- A source facility
- A load time
- A destination facility
- An unload time

Optionally, assets in transit should track which user (should be a logistics officer) set the load and unload times. You might also consider relating the asset transit to the transfer request (in which case the source and destination information might be in the transit request rather than this other transfer table). Maybe you don't need two tables at all and everything goes in the transfer request table...

What ever the shape of the tables and however many you add, place comments in your create_tables.sql script to discuss the design decisions you are making to support asset transfers.

-- Datamodel updates - 8pts (4 for transfer request tracking, 4 for transit tracking)

Step 3 - Dashboard Updates

Several updates to the dashboard are needed to make navigation easier. You might have already added some of these last week on your own.

The dashboard screen should have links to the following routes:

- /asset_report - used by any user to get a listing of assets by facility
- /transfer_report - used by any user to get a listing of assets in transit. Route will be added in a later step.
- /add_asset - used by any user to add an asset
- /add_facility - used by any user to add a facility
- /dispose_asset - visible only to logistics officers, used to dispose of assets
- /transfer_req - visible only to logistics officers, used to initiate an asset transfer. Route will be added in a later step.

The dashboard should also have a report/table showing work that the user needs to do. For logistics officers, this will be a listing of asset transits which need to have load or unload times set. For facilities officers, this will be a listing of transfer requests needing approval. In either case, clicking on one of the rows in the report should take the user to the screen to complete the appropriate action (a parameter will be passed using a GET argument to identify the record the user will act on).

-- Site Navigation - 4 pts (all or nothing)

Step 4 - Initiate Transit Request

Transit request initiation should be access controlled, only logistics officers should be able to initiate transfers.

- Add a route for '/transfer_req'
- 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 officers can request transfers.

- If the request is a 'GET' type request:
    - A template with the form elements needed for a transfer request should be rendered for the user (minimally source facility, destination facility, and asset tag).
    - The form method should be POST and the action should go to the /transfer_req route
- If the request is a 'POST' type request:
    - The user input should be checked for validity. Does the asset tag exist? Is the asset not disposed? Is the asset not currently in transit? Do the source and destination facilities exist?
    - If any of the validity checks fail, the user should be redirected to a screen indicating which check failed.
    - A transit request record should be inserted into the DB based on the user input.
    - The user should be redirected to a page indicating that the transit request has been successfully added

-- Request Init - 2pts (all or nothing)

Step 5 - Approve the Transit Request

Now that the system can add transit requests, an interface is needed to approve/complete them.

- Add a route for '/approve_req'
- Use the session information to lookup the user's role
- If the user is not a facilities officer, redirect the user to a screen indicating only facilities officers can approve transit requests.
- If the request is a 'GET' type request:
    - Check the argument that identifies the request to be approved
    - If there is no matching request or the request has already been approved, redirect the user to a screen indicating that they have made an invalid request.
    - A template showing the transit request information should be rendered with buttons to accept or reject the request.
    - The form method should be POST and the action should go to the /approve_req route.
- If the request is a 'POST' type request:
    - If the user clicked the reject button:
        - The transit request should be marked rejected or should be deleted from the database.

        - The user should be redirected to the dashboard.
    - If the user clicked the approve button:
        - The transit request should be marked approved.
        - The data needed to start tracking an asset in transit should be added to the database (whatever needs to happen so that the system knows to ask for load/unload times).
        - The user should be redirected to the dashboard.

-- Approve Request - 2pts (all or nothing)

Step 6 - Transit Tracking

Approved requests need an interface to be serviced.

- Add a route for '/update_transit'
- Use the session information to lookup the user's role
- If the user is not a logistics officer, redirect the user to a screen indicating only logistics officers can approve transit requests.
- If the request is a 'GET' type request:
    - Check the argument that identifies the transit to be updated
    - If there is no matching transit or the transit has had an unload time set, redirect the user to a screen indicating that they have made an invalid request.
    - A form allowing the load and unload time to be entered should be rendered.
    - The form method should be POST and the action should go to the /update_transit route.
- If the request is a 'POST' type request:
    - Update the load and unload times in the database for the appropriate record.
    - Redirect the user to the dashboard.

-- Finish Transit - 2pts (all or nothing)

Step 7 - Transfer Report

Now that the system can support a process to move assets, a report showing assets in motion should be possible.

- Add a route for '/transfer_report'
- If the request is a 'GET' type request:

    - A template should be rendered to take in the date for the report.
    - The from method should be POST and the action should go to the /transfer_report route.
- If the request is a 'POST' type request:
    - A template should be rendered to take in the date for the report.
    - The from method should be POST and the action should go to the /transfer_report route.
    - A report/table should be shown minimally listing the asset tag, load time, and unload time for all assets where the load time is less than the time provided by the user and the unload time is greater than the time provided by the user. If the unload time is null, the unload time is in the future. If the load time is null then the load time is in the future.
    - A link or button should be provided to take the user back to the dashboard.

-- Complete functionality - 2pts (all or nothing)

Step 8 -- Merge the feature branch

With the iteration complete, merge the work back into the master branch. Look at last week's assignment for an example of the command sequence.

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