Final Project

Due No Due Date **Points** 150 **Available** Nov 8 at 12:01am - Nov 28 at 11:59pm 21 days

Overview

This Final Project will incorporate topics from the beginning of the term through ECMAScript. You will be building a RESTful budgeting application where a user can sign up and keep track of monthly budget categories and purchases in each of those categories. The app will also use functional programming to perform basic budget analysis in order to let the user know how much money they have left in a category for the month or if they have overspent.

Required Packages

- Google Chrome - All grading will be done in Google Chrome, so make sure your budget app works in Google Chrome.
- Python 3.x
- Flask
- Jinja2 Templating engine (included with Flask)
- Flask-SQLAlchemy
- Flask-RESTful
- This project is to be done using vanilla JavaScript with the standard library. This means that you cannot use external libraries like jQuery or React.
    - You are free to use either the Fetch API or XMLHttpRequest objects to implement AJAJ.

GitHub Classroom Link

Please click on the following link ot get your project repository
https://classroom.github.com/a/rkVpVIKL (https://classroom.github.com/a/rkVpVIKL)

Specification

1. You must build a RESTful API for accessing your budget category and purchase resources.
    - For budget categories, you must implement
        - GET "/cats" to get the list of categories
        - POST "/cats" to add a new category
        - DELETE "/cats/<categoryId>" to delete the category with the id <categoryId>
    - For purchases, you must implement
        - GET "/purchases" to get a list of purchases
        - POST "/purchases" to add a new purchase

    - You **do not** need to implement any other RESTful methods.
    - All data must be transmitted using JSON.
2. When visiting the page for the first time, users should be given the chance to create an account or login.
    - You will need to verify no duplicate usernames.
    - Any failed login or account creation should present an error message to the user.
    - User management should **not** be done with AJAJ or REST. You should implement this using normal flask routes alongside the REST resources.
3. When the user is logged in and visits the root route of your site ("/"), your Flask application should send a basic page skeleton to the user along with a JavaScript script that will make AJAJ requests to populate the page.
    - The skeleton should display the current month, a place for the budget categories to be populated and a form for adding a new purchase.
    - Once the page is loaded by the user's browser, it should make AJAJ requests for both the list of categories and list of purchases made by the user using the RESTful API.
        - You must use console.log() to print the contents of every AJAJ response you get to ensure ease of debugging and grading.
    - Upon receiving the list of categories and purchases, the page should display the status of each of the user's budget categories, and the total sum of all uncategorized purchases.
        - To perform the budget category calculations, use a combination of the JavaScript Array.map(), Array.reduce(), and/or Array.filter() methods.
        - The status of each budget category will be the budget amount minus the total of all purchases made in the current month. If the user has spent more money than was budgeted, list the status as "overspent".
        - You should **not** display the individual purchases to the user.
    - The user should always have the ability to add a new purchase with the following fields
        - The amount spent
        - What it was spent on
        - The date that it was spent
        - The category it should be counted towards
    - The user should always have the ability to add a new category (specifying name and limit), and also always have the ability to delete any existing category.
        - If a category is deleted, all of the purchases assigned to that category should become uncategorized.
    - You do not need to regularly poll the server for updates. However, once user requests any changes (e.g., add a purchase, add a category or delete a category), your application should fetch updated information via the RESTful API, and recompute the status of each of the user's budget categories.
4. The budget website should only show the current month. You **do not** need to implement switching between months.

5. All users, categories and purchases should have Models and be stored in a database using SQLAlchemy.

Notes/Hints

- Using the JavaScript Date() object, you can construct a date object, and use that to figure out the current month.
- You can use any of the examples from class for inspiration and you may use snippets from them in your code. However, **DO NOT** copy them in their entirety and edit them for this project

Submission Guidelines

- Place any comments you have for the grader to help them grade your assignment in the FOR_GRADER file
- Name the main (controller) file budget.py
- Name the sqlite database budget.db
- You must be able to initialize your database by setting the FLASK_APP environment variable to budget.py and running "flask initdb"
- You must be able to run your application by setting the FLASK_APP environment variable to budget.py and running "flask run"
- Do not submit any IDE files, feel free to add them to the .gitignore file under "# Editor files"
- Be sure to remember to push the latest copy of your code back to your GitHub repository before the the assignment is due. GitHub Classroom will not stop you from pushing additional commits after the deadline but it will make the latest commit before the deadline the submission that the grader will grade. It will also notify us if previous work is edited after the deadline.