Joseph Kramer
CS496 Mobile Web
Final Project

MOBILE APP: http://kramerj-final-project.appspot.com/
VIDEO: http://web.engr.oregonstate.edu/~kramerj/CS496_FINAL/

My app is designed within the GAE environment. I use Jinja2 to display data from the datastore and I use Python for backend control. The front end is designed with Bootstraps and mobile in mind. With Bootstraps I can manage how the frontend interacts with a variety of screen sizes. This app is tested on actual mobile phones, emulated phones, plus webpages and because of Bootstraps all screen sizes functioned normally.

The frontend interacts with the backend (API) via HTML forms; therefore POST is used to send all data to the backend. If the user would like to delete an item, it is sent as a POST. The typical DELETE and PUT are handled as a POST.

For example: To DELETE an item from the datastore. Instead of sending off a DELETE this app will send off a POST to the server. From there the Python backend will handle the process of deleting the item. The same concepts applies to the PUT command for editing, it will be sent as a POST.

However the API is still coded with GET, DELETE, & PUT, incase a developer would like to access the datastore via it's API instead of the GAE app.

This app does not utilize the GET command to display data, because data is displayed via Jinja2. Jinja2 can access WebApp2 Datastore data and display the results in HTML. The code is simple to write and easy to use via Python. A GET is still available through the API to retrieve data, if a developer would like to utilize it; however this app uses Jinja2.

In conclusion I designed the app to take full advantage of the GAE system. My API still utilizes the traditional GET, PUT, DELETE, POST calls, but my app focus on technology that makes the process of calling the datastore much easier. i.e. Jinja2, HTML forms, and Python.

Technologies & Languages:

jQuery
Python
Jinja2
HTML5
Datastore

List of API commands

GET:
/get_scientist_by_name
/get_project_by_name

DELETE:
/delete_scientist
/delete_project

PUT:
/edit_scientist
/edit_project

POST:
/add_scientist
/add_project

How to interact with the API

GET:
Call command: kramerj-final-project.appspot.com/get_scientist_by_name?
Required data: scientistName
Example: /get_scientist_by_name?scientistName=[some-name]
Notes:

- It will verify the scientist name is not empty
- If unsuccessful an error message will be sent
- If successful
  - It will return a JSON of data pertaining to scientistName

Call command: kramerj-final-project.appspot.com/get_project_by_name?
Required data: projectName
Example: /get_project_by_name?projectName=[some-name]
Notes:

- It will verify the project name is not empty
- If unsuccessful an error message will be sent
- If successful
  - It will return a JSON of data pertaining to projectName

DELETE or POST:
Call command: kramerj-final-project.appspot.com/delete_scientist?
Required data: entity_key
Example: /delete_scientist?entity_key=[entity key]
Notes:

- It will delete the scientist via the entity key
- If unsuccessful
  - POST will send error message and redirect to an error Page
  - DELETE will send error message
- If successful
  - POST will send redirect back to the refereeing (home) page
  - DELETE will send a success message

Call command: kramerj-final-project.appspot.com/delete_project?
Required data: entity_key
Example: /delete_project?entity_key=[entity key]
Notes:

- It will delete the project via the entity key
- If unsuccessful
  - POST will send error message and redirect to an error Page
  - DELETE will send error message
- If successful
  - POST will send redirect back to the refereeing (home) page
  - DELETE will send a success message

PUT or POST:
Call command: kramerj-final-project.appspot.com/edit_scientist?
Required data: dob, scientistName, inlineRadioOption
Optional data: famous_discovery
Example:/edit_scientist?dob=[dob]&scientistName=[name]&inlineRadioOption=[sex]&famous_discovery=[text]
Notes:

- Revalidates what should have been validated from the frontend
- It will validate data received and strip extra spaces from the front and back of string.
- DOB is verified to be MM-DD-YYYY
- Performs update of scientist data.
- If successful
  - POST will send a success message and redirect back to the refereeing (home) page
  - PUT will send a success message
- If unsuccessful
  - POST will send error message and redirect to an error Page

    - PUT will send a error message
- Edits the scientist names that are in the Project table

Call command: kramerj-final-project.appspot.com/edit_project?
Required data: projectName, scientistName, entity-key
Example: /edit_project?projectName=[name]&scientistName=[name]&entity-key=[entity key]
Notes:

- Will validate that projectName and scientistName are not blank.
- Will check for duplicate project names, because it doesn't make sense to assigned the same project to the same scientist twice.
- Updates the project name for that particular scientist
- If successful
    - POST will send a success message and redirect back to the refereeing (home) page
    - PUT will send a success message
- If unsuccessful
    - POST will send error message and redirect to an error Page
    - PUT will send a error message

POST:
Call command: kramerj-final-project.appspot.com/add_scientist?
Required data: scientistName, inlineRadioOptions
Optional field: famous_discovery
Example:/add_scientist?scientistName=[name]&dob=[dob]&inlineRadioOptions=[sex]&famous_discovery=[text]
Notes:

- Revalidates what should have been validated from the frontend
- It will strip access spaces from front and back of string.
- DOB is verified to be MM-DD-YYYY
- Verifies the same scientist is not entered twice
- Performs update of scientist data.
- If successful
    - POST will send a success message and redirect back to the refereeing (home) page
- If unsuccessful
    - POST will send error message and redirect to an error Page

Call command: kramerj-final-project.appspot.com/add_project?
Required data: science_key, project_name, longitude_name, latitude_name, altitude_name

Example: /add_project?science_key=[Scientist Name]&project_name=[Project Name]&longitude_name=[longitude]&latitude_name=[latitude]&altitude_name=[altitude]

Notes:

- Will validate the following are not blank:
  - projectName
  - scientistName
  - longitude_name
  - latitude_name
  - altitude_name
- Will check for duplicate project names, because it doesn't make sense to assigned the same project to the same scientist twice.
- Enters the project name for a particular scientist
- If successful
  - POST will send a success message and redirect back to the refereeing (home) page
- If unsuccessful
  - POST will send error message and redirect to an error Page

Account System

I used the Google App Engine API called "Users." This made programming a login system much easier. Only Google account users are allowed to use this app.

This is how it works:

When a user loads the webpage they are directed to the root which loads "run_app.py" and within that file is this line of code: "users.get_curent_user()." That code is ran before anything else can happen, at this point GAE API takes over and directs the new visitor to enter their Google Account information. That information is returned to a "user" object where I use it to distinguish users within the datastore.

From there I can track who adds and deletes what data. Since each user is unique, because of Google's vetting process, I don't have to worry about duplicate users or password security. The user object is tracked within the datastore for each unique login.

I think it is safe to say that Google made using their login system developer friendly. It is recommend that any application using this API allows users the ability to login via Google Accounts.