Jack Holkeboer
Oregon State University Computer Science 496
holkeboj@onid.oregonstate.edu
Assignment 3, Part 2

Public URI: http://my-recipe-tracker.appspot.com

Explanation of URL Structure

There are two main types of entities, "recipes" and "ingredients". Recipes can be accessed by the /recipes route with the recipe key supplied as a parameter. Other operations can be done by substructures under the /recipes route, such as /recipes_new and /recipes_delete. The /ingredients route allows you to look up all recipes that use a certain ingredient. See the code in main.py for details and documentation of each route.

Test Cases

I used the Chrome extension called "REST Console" to test my application. This program simulates HTTP requests and allows you to supply parameters. The below table shows the URIs for each test and the results.

Test case: Create a valid recipe
URI: /recipes/new
Request Type: POST
Parameters:
name=Enchiladas&prepTime=30&category=dinner&ingredientList=salt,pepper,tortillas&instructionList="Chop up veggies, cook and eat"
Expected Response: 200 "Recipe saved successfully"
Test Passed? Yes

Test case: Attempt to create recipe with invalid prep time
URI: /recipes/new
Request Type: POST
Parameters:
Enchiladas&prepTime=thirty&category=dinner&ingredientList=salt,pepper,tortillas&instructionList="Chop up veggies, cook and eat"
Expected Response: 500 "Invalid prep time"
Test Passed? Yes

Test case: Attempt to create recipe with invalid category
URI: /recipes/new
Request Type: POST

Parameters:
Enchiladas&prepTime=thirty&category=brunch&ingredientList=salt,pepper,tortillas&instructionList="Chop up veggies, cook and eat"
Expected Response: 500, "Invalid category. Valid categories are breakfast, lunch, and dinner"
Test Passed? Yes

Test case: Attempt to create recipe with no ingredients
URI:
Request Type: POST
Parameters: name=Enchiladas&prepTime=30&category=dinner
&instructionList="Chop up veggies, cook and eat"
Expected Response: 500, "No ingredients provided"
Test Passed? Yes

Test case: Attempt to delete resource that doesn't exist
URI: /recipes/delete
Request Type: DELETE
Parameters: recipe=garbage
Expected Response: 500, "Attempted to delete recipe that does not exist
Test Passed? Yes

Test case: Associate recipe with ingredient it already contains
URI: /recipes/add_ingredient
Request Type: PUT
Parameters: recipe=[valid-recipe-key]&ingredient=[ingredient-in-the-recipe]
Expected Response: 200, "Ingredient already in recipe"
Test Passed? Yes

Test case: Ask for list of recipes that use ingredient that doesn't exist
URI: /ingredients
Request Type: GET
Parameters: ingredientName=kryptonite
Expected Response: 500, "Ingredient not found"
Test Passed? Yes

Description of RESTful constraints

One way that my application is not strictly RESTful is that my 'resources' are stored in a database. They are not accessible directly from a URI because the server needs query parameters to know what to look for. For example, my '/recipes/all'

route does not need a parameter because it looks up all recipes. But to look up a specific recipe, you need to

It is possible in Flask to parse the URL and use the 'subfolders' as parameters. For example if you requested '/recipes/tacos', it could take the name 'tacos', find that recipe, and send it back. There are two problems with this. For one, a system like this should be able to support multiple recipes with the same name. We could have thousands of users, many of whom have a recipe called 'tacos'. This is why I use resource keys as query parameters in my '/recipes' route. The tradeoff is that these keys are not human readable, and it goes against the spirit of REST to have an unreadable URI like that.

My application is restful in that it separates client and server functions. Everything client side is handled either by the templating system, or ideally a consumer of the API could create their own interface from scratch. My Flask server code only deals with the data, it does not dictate how it is displayed to the client or how, for example, the client would parse user input and structure the requests.

Discussion of schema changes

When I was researching the data types available in Google's NDB, I came across this page from Google's documentation which suggests that there is a JsonProperty datatype in the NDB that can support Json: https://cloud.google.com/appengine/docs/python/ndb/properties.

However, it turns out that JsonProperty is not actually a method of the google.appengine.ext.db module. I'm not sure if this is a case of outdated documentation, different versioning, or something else. The closest thing NDB has is the BlobProperty, which is an unstructured blob of text that could be parsed as JSON. However, this request that the client is able to properly parse JSON and this is not an idea situation.

Because of this, I decided to form my ingredient as a list of keys, and to offload amounts of ingredients to the "instructions" property. Each entry in the Recipe.ingredients property references an entry in the Ingredient table.

Having finished the API, what would you have done differently?

I found working with Google's NDB to be very limiting. If I was doing this again, I would use SQL. SQL makes it easier to deal with structured data. Its relational capabilities would have allowed me to create an index table with each instance of a recipe-ingredient pair, and the amount used in that specific recipe. This could then be queried with a join. But in a non-relational database, this type of association requires a lot of additional work and making multiple queries.

I also find it easier to work with raw SQL queries instead of Python methods, even though similar ORM layers exist for relational databases. Google's GQL

language is decent but it behaves in some ways that are not obvious from the syntax. For example, to check if something is within a List in GQL, instead of using the IN operator like you would in SQL, you use the = symbol. This is semantically confusing because that is not what the word 'equals' means.