|  |
| --- |
| Cloud Computing: Assignment 1 |
| Electronic Vehicle Application |
| Documentation guide |

|  |
| --- |
| Jenom Yahaya Kassim - 3004361  3-31-2020 |

Contents

[Introduction 2](#_Toc36543762)

[The App.yaml file 2](#_Toc36543763)

[User Login / Logout and Jinja 2](#_Toc36543764)

[Classes and Methods 2](#_Toc36543765)

[MainPage 2](#_Toc36543766)

[AddCar 3](#_Toc36543767)

[SearchCars 3](#_Toc36543768)

[CarDetails 5](#_Toc36543769)

[CompareCars 7](#_Toc36543770)

# Introduction

EVHub is a PaaS application that houses a database of electric vehicles. The application allows users add and update vehicles in the database as well as perform comparisons of multiple cars in the database and leaving reviews on the cars in the database.

The application was built using Python 2.7, webapp2 and the jinja templating framework for deployment on the Google App Engine.

# The App.yaml file

The yaml file is responsible for informing Google App Engine about the runtimes and libraries needed for this application to run. It is also responsible for defining where requests should be routed in the application. The yaml file used in this application is called “app.yaml”.

The first part with the comment “RUNTIME USED” states which runtime was used (python 2.7 version of the runtime because the 3.0 wasn’t stable enough as at the time of development), it also states version 1 of the Google App Engine API was used (just like the python runtime, this is the only one available as at time of development), and finally the threadsafe states that our application is threadsafe so that the App Engine knows it is safe to launch multiple instances on the same server using threads and that they will synchronise correctly and will not cause data loss or race conditions.

The second part with the comment “LIBRARIES USED” states what libraries the App Engine should include to ensure the application functions properly. The libraries included are Jinja, which generates dynamically rendered HTML templates at run time to provide dynamic contents to individual users. Like the previous code blocks, we also specify which version of the library to use, which is the latest available to avoid as many security vulnerabilities as possible.

The third part with the comment “ROUTES FOR THE STATIC ASSETS USED” states which python modules and objects are responsible for handling requests sent to certain URLs. This application directs all requests to any url to the app variable that is defined in main.py as that contains our routing table. It also directs the application to the static directories housing the javascripts, css and image assets of the application when rendering of the html pages occurs.

# User Login / Logout and Jinja

The login and logout functionalities are part of the main cores of the application as they determine and control a lot of what users have access or are restricted to on the application. The Google user login / logout service was used to implement the login / logout functionality using Google user accounts as it is a proven, tried and tested authentication service.

# Classes and Methods

## MainPage

This class is responsible for the handling of the landing page. The get method first checks to make sure the user is logged into the application, this is to decide on what content is being served to the user.

It doesn’t contain a post method because no forms are being handled on this page

## AddCar

This class handles the addition of vehicles to the datastore. Like in the get method on the MainPage, it first checks to make sure a user is logged in to the application before deciding what to display. There are two versions of the page it displays depending on the login status of a user.

**Logged in user**

It displays a page with a form to allow logged in users add vehicles to the datastore. Every field in the form is required so a required attribute was included in each input field to ensure the submit button’s action can’t be invoked without valid inputs. To also ensure the integrity of inputs, fields that accept strings e.g. the name and manufacturer’s field were declared as type inputs, and input fields for integers were set to a type of number e.g. year, wltp range, power and cost, while input fields that accept float e.g. battery size was set to number with a step attribute.

**Non logged in user**

It displays a page telling the user they can’t access the page as a guest and must login to the application before they can get access to add cars to the application.

The post method of the AddCar handles the addition of cars to the datastore. On click of the submit button, a new object of the Vehicle class is created and its values are used to populated using the values gotten from all the inputs from the input field of the AddCar get page. While the values are being passed to the new object’s values, integer and float values are casted to their appropriate data types to ensure processing of the data can be done efficiently without errors.

After the values have been passed and casted successfully, a query to the datastore is made filtering by the new Vehicle object’s name, manufacturer and year. This is to ensure that the vehicle about to be added to the application does not currently exist in the datastore.

To test for this, the length of the result of the query is checked, if the length of the query is equal to zero that implies that the car does not exist in the datastore so the application can put the new vehicle into the datastore and return a success message. But if the length of the query is not equal to zero, then it means that the vehicle already exists in the datastore and an error message is passed to the application.

## SearchCars

This class is responsible for the handling of the search page of the application. Like the previous classes already mentioned, the get method first checks the status of the user i.e. whether the user is logged in or not. Although this does not affect the content of the page displayed to the user to a great extent, as all it does is to ensure that the login / logout button either shows “Logout” for logged in users along with its corresponding URL function as well as “Login” for logged out users / new users along with its corresponding login URL function.

The page displays all the vehicles in the application’s datastore and the number of vehicles in the datastore. To achieve this, the get method queries the datastore and runs a loop on the result, the values of each of the iterations are passed to a template on the html page. Also, a count with an initial value of zero is declared before the loop begins and with every iteration of the loop, the count is incremented by a value of 1. At the end of the loop, the total count is rendered into the html of the search page to display how many vehicles are in the datastore.

The post method

Like the get method, it first checks the status of the user and either shows “Logout” for logged in users along with its corresponding URL function as well as “Login” for logged out users / new users along with its corresponding login URL function. It then proceeds to passing the values from the search form field for processing. For processing to begin, the application checked that none of the numerical values are empty. This is done to ensure that the lower limit and upper limit of the numerical search parameters have values in it for a logical comparison to be performed during the query filter process.

If the any of the values passed from the numerical input fields i.e. year, battery size, wltp range, cost and power does not have any value in it, a default minimum value which is equal to the lowest lower limit value allowed by the search field for that input field is assigned to the minimum value field and a default maximum value which is equal to the highest higher limit value allowed by the search field for that input field is assigned to the maximum value field. This is done because the result of those values will always return every vehicle in the datastore.

After that is done, the application checks to ensure every numerical value passed from the input field into the lower limit and upper limit field is within the lower and upper limit range. If any of them fails, an error message is sent notifying the user which field(s) have the errors in them. If the values all conform within the lower and higher limit range, then a query is performed on the datastore.

The query is then filtered individually with conditions that satisfy the lower and upper limit of the year, battery size, range, cost and power and fetches just the keys of each of them. Then a final query is done depending on the following conditions

1. If the name and manufacturer of the vehicle is not inputted into the search field

The final query does a query that takes the results of the year query and filters it by the intersection of the battery size, range, cost and power to return the result(s) of vehicles that satisfy the search conditions Also, a count with an initial value of zero is declared at the beginning of the post method and a loop on the result of the final query is done so that with every iteration of the loop, the count is incremented by a value of 1. At the end of the loop, the total count is passed to the search page to display how many vehicles are in the datastore. In the same fashion, a loop is run (on the html page) on the final result of the query to display each of the vehicles and its details.

1. If the name of the vehicle is not provided but the manufacturer of the vehicle is provided into the search field

A query is first done on the datastore and filtered by the value of the manufacturer’s name. Then the final query does a query that takes the results of the manufacturer’s query and filters it by the intersection of the year, battery size, range, cost and power to return the result(s) of vehicles that satisfy the search conditions. Like in the first condition, a count with an initial value of zero is declared at the beginning of the post method and a loop on the result of the final query is done so that with every iteration of the loop, the count is incremented by a value of 1. At the end of the loop, the total count is passed to the search page to display how many vehicles are in the datastore. And in the same fashion, a loop is run (on the html page) on the final result of the query to display each of the vehicles and its details.

1. If the name of the vehicle is provided but the manufacturer of the vehicle is not provided into the search field

A query is first done on the datastore and filtered by the value of the name of the vehicle. Then the final query does a query that takes the results of the name of the vehicle’s query and filters it by the intersection of the year, battery size, range, cost and power to return the result(s) of vehicles that satisfy the search conditions. Like in the previous condition, a count with an initial value of zero is declared at the beginning of the post method and a loop on the result of the final query is done so that with every iteration of the loop, the count is incremented by a value of 1. At the end of the loop, the total count is passed to the search page to display how many vehicles are in the datastore. And in the same fashion, a loop is run (on the html page) on the final result of the query to display each of the vehicles and its details.

1. If the name of the vehicle is provided and the manufacturer of the vehicle is provided into the search field

A query is first done on the datastore and filtered individually by the value of the name of the vehicle and the manufacturer’s name. The final query then does a query that takes the results of the name of the vehicle’s query and filters it by the intersection of the manufacturer’s name, year, battery size, range, cost and power to return the result(s) of vehicles that satisfy the search conditions. Like in the previous condition, a count with an initial value of zero is declared at the beginning of the post method and a loop on the result of the final query is done so that with every iteration of the loop, the count is incremented by a value of 1. At the end of the loop, the total count is passed to the search page to display how many vehicles are in the datastore. And in the same fashion, a loop is run (on the html page) on the final result of the query to display each of the vehicles and its details.

## CarDetails

This class handles the display of the car details, adding a review, deleting a review, viewing a review, editing a vehicle and deleting a vehicle. Like the AddCar class, what the user sees is heavily dependent on the login status of the user. The get method can be broken down into the general view and the logged in user’s view.

**General view**

Like the previous classes already mentioned, the get method first checks the status of the user i.e. whether the user is logged in or not to ensure that the login / logout button either shows “Logout” for logged in users along with its corresponding URL function as well as “Login” for logged out users / new users along with its corresponding login URL function.

To access the car details page, you have to view via the use of a hyperlink. Embedded in the hyperlink is a urlsafe key value passed to a variable called “id”, the get method first retrieves the value in the id and decrypts the urlsafe key and the decrypted key is then used to query the datastore to retrieve the details of that particular vehicle which is then used to populate the vehicle information on the html.

To handle the population of the value of the vehicle review, a loop is made on the result of the initial query with a nested loop querying the structured property, “review” to count how many reviews are in the datastore and also add the values of the ratings in the datastore. After the loop is done, we check that the count of the review is greater than zero before doing a calculation of the addition of the total ratings of the vehicle divided by the total count of reviews of the vehicle then we pass the result as the average rating of the vehicle. If the count is not greater than zero, that means a review of the vehicle hasn’t been made yet and we return a message saying “No review”.

The get method also handles the display of all the reviews and ratings of the vehicle by running the same loop above which gets the count and values of the rating before performing the average rating score. This time it doesn’t perform any mathematical operation, rather it displays the results of the review and the ratings of each review.

**Logged in view**

The logged in view is an extension of the general view so we will only talk about the workings of the extensions since the base implementation has been handled already in the general view.

The get method which handles the display of all the reviews and ratings of the vehicle as stated above in the general view has a feature to delete reviews on the logged in view. This action is handled by the post method. A button called delete is displayed on the html which has a hidden value of the loop index minus 1. This is done because the index in the datastore for lists starts from 0. On click of the delete button, the value of the index is passed to the post method and a delete query on the review index is performed and then put back to the datastore to commit the delete.

The post method also handles the addition of review by listening for an action equals to ‘Review’, which is the value of the action that handles the addition of a review. On click of the button, the user review from the text field is gotten along with the rating and the values are appended to the review list of the vehicle and then put in the datastore to commit the addition and finally the page redirects back to itself to reload the page so the updated details can be displayed.

The edit feature is also handled by the post method and this is responsible for the updating of vehicle details in the datastore. It is invoked by the click of a button equals to “Submit” and when this button is clicked, the values of all the form fields are passed into the application. By default when the edit button on the html is clicked, it preloads the values of each field with the current values in the datastore so that updating the values will be easy in terms of user experience as well as also passing of values back to the post method so that no values are passed back blank. Just like in the AddCar class which checks if a vehicle with the same name, manufacturer and year exists in the database, the exact same query is performed and if the length of the query is equal to zero that implies that the car does not exist in the datastore so the application can update the vehicle details in the datastore and return a success message. But if the length of the query is not equal to zero, then it means that the vehicle already exists in the datastore and an error message is passed to the application saying the vehicle cannot be updated because it already exists in the database.

And finally, the post method also handles the deletion of a vehicle from the datastore. If the delete button is clicked, the delete query of the key of the vehicle is invoked and the page is redirected to the search page where it displays the list of all the current vehicles in the datastore.

## CompareCars

The class handles the comparison of vehicles in the datastore. Like the previous classes, the get method first checks the status of the user i.e. whether the user is logged in or not. Although this does not affect the content of the page displayed to the user to a great extent, as all it does is to ensure that the login / logout button either shows “Logout” for logged in users along with its corresponding URL function as well as “Login” for logged out users / new users along with its corresponding login URL function.

The post method compares a maximum of four vehicles at a time, querying the vehicle name and year against the datastore to return unique results. The first two vehicle values and year fields must be provided on the html because a comparison is done on a minimum of two values. A class called VehicleResult is created as well as a class called VehicleResultValues. These classes are created to handle the final values displayed and rendered to the html and the values that will be used for the logical operations for the comparison respectively. Also a variable called v1Link, v2Link, v3Link and v4Link are created and assigned a value of True to handle the generation of hyperlinks to the car details after the result of the compare operation has been displayed.

If the compare button is clicked, the values of all the eight input fields (name and year) are retrieved and queries are performed on the datastore. Each of these queries are done on the individual vehicles names and year and a loop is performed on each of the individual queries to get their results.

If the name field of the third and fourth vehicle aren’t inputted, then a new object of VehicleResult and VehicleResultValues are created and values representing the third and fourth vehicles are created as new objects of the VehicleResult class with default numerical values of zero. Also the v3Link and v4Link are assigned a value of False since no result will be displayed for the third and fourth vehicles hence there will be no hyperlinks generated to the car details after the result of the compare operation has been displayed.

In the same vein the length of the queries of the first and second vehicles is checked to see that it is not equal to 1. If it is not equal to 1, that means, no vehicle in the datastore exists with the name and year and so vehicle 1 and/or 2 that meets this parameter is assigned as a new object of VehicleResult with numerical values of 0 as well as the v1Link and v2Link are assigned a value of False since no result will be displayed for the first and/or second vehicles hence there will be no hyperlinks generated to the car details after the result of the compare operation has been displayed.