Cloud Computing- Assignment 1-Documentation

EV DATABASE

Motivation

This web application was developed for the requirement of Master's in Big Data Management and Analytics, Semester 2- Cloud Computing module. The end goal was to develop a database for Electric Vehicles as a PaaS application.

1. Description

This PaaS (Platform as a Service) application is built using Google App Engine with Python version 2.7. The application serves as a database for Electric Vehicles. This application deals with users who login and logout and based on this status, functionalities available to them is altered. In the application users can add more vehicles to the database and modify details of vehicles that are present in the EV database. The functionality to add, delete and modify the details are available only for logged in users. Logged in users can also submit reviews of the cars, the details of which are displayed in the information page for each EV. However, despite this status, users will be able to perform comparisons of multiple cars in the database. Similarly, all users can query the database, filtering on attributes to see the EVS that satisfy their criteria. EVS are displayed as hyperlinks which takes the users to the information page where edit, delete and submit review functionalities are provided.

2. File organization and naming conventions

All the image files are included in the 'img' folder and the CSS stylesheet is included in the 'css' folder. These folders with the other yaml, python and html files are zipped together for submission.

All files are named according to their actual functionalities making them easier to understand. Similarly, data model names and variable names are also self- explanatory.

3. Images

All images included to enhance the aesthetics of the User Interface are designed using online platform *Canva*. Images are thus not violating any copyright laws and are solely created for the purpose of this application.

4. Reference materials used for learning

In order to learn the basics of Google App Engine, the 'Google App Engine with Python' pdf provided as part of the module was utilised. To deeply understand, the concepts of entity, key, datastore and queries, Google App Engine documentation was referenced. All of the references are provided in the end of this document.

5. Code documentation

1. app.yaml

This file contains all the configuration details including name and version information about the library used (jinja2) and the static directory for ‘img’ and ‘css’ folders. Handlers transfers requests for all the URLS to the variable ‘app’ that is defined in python file (main.py) which has the routing table for this web application.

2. get() in all Classes

get() method of all classes accepts an argument of self. This means that this method may only be called on an instance of the class. This method will respond to the GET HTTP verb. It will render the result of a GET request to this class. [1] This functionality of get() method is common for all classes. Other functionalities of get() methods specific to classes are documented in appropriate sections.

3. Main page

3.1. MainPage Class in main.py

3.1.1. get()

This method checks the user’s logged in or logged out status to create a url string accordingly. The welcome message will also be displayed depending on whether the user is a guest or already logged in. Jinja environment constant get the template called main.html. The values like welcome message, user object and URL string are then rendered to the main.html page as a HashMap. Google App Engine’s default login-logout service is used for this assignment. If there is a user logged in a logout link is created (this links to the Google users API). When the user clicks that link and logs out main page with limited functionalities will be displayed to the logged-out user. Similarly, if there is no user a login URL is created that after login will redirect the user back to this page and then the user can see functionalities that are specific to logged in users. This is achieved by passing the user object to know the status of the user.

All webapp2 applications are instances of a WSGIApplication which has all of the required code to transfer requests to the appropriate classes. Application routing table is defined as a list in Python. Tuples are used in the list that states that any requests to the root of the application at ‘/’ calls the respective methods of an instance of the ‘MainPage’ class. [1]

4. Adding an EV

4.1. Add class in add.py

4.1.1. get()

This method creates default ids for EVS initially using default option. The EV object and the user status are rendered to add.html where the form for adding an EV data will be displayed.

4.1.2. post()

This method handles HTTP post request. This method gets the name, manufacturer, year, power, cost, WLTP Range which are input elements in the page for adding an EV (add.html), and stores them in the EV entity. The name, manufacturer and year are checked to see if an EV with same name, manufacturer and year already exists in the database. If that is the case then the EV will not be put to the database and an error message will be displayed. All fields are marked as mandatory in the add.html to disallow null and incomplete information. Year field is validated by providing a range from 1830-2020, as the first EV was manufactured in 1830. Fields accept only text, number or decimals depending upon the attribute, this ensures that incorrect information is not added to the database. These validations are included in add.html, and only upon the entry of correct information users can add the EV details to the database. This method also redirects the user to the Main page when cancel option is invoked

5. Querying an EV

5.1. Search class in search.py

5.1.1. get()

This method allows the users to query the EV datastore. In this method, if the action is 'Search' (From the button in search.html page), data is filtered out from the EV database. The EVS that satisfy the criteria are displayed as a list of hyperlinks with their names as links. This is done using conditional statements to check if all the fields in the search form are filled or not. This method gets the data from fields that users have inputted and performs a query operation on these attributes. If all the fields are empty, query list is empty and then the entire EV list will be displayed to the user. Users are allowed to filter on one attribute or as many attributes as they want. If the user enters more than one attribute to query or filter on, an AND operation will be performed and only the EVS that satisfy all the conditions will be displayed as list of hyperlinks.

The list is displayed on the same page(search.html) for ease of further querying. When there are no EVS that match the user's search criteria, 'Sorry, no records found' message is displayed. This is achieved in this method by using jinja2. Limits are included in search for numerical attributes and for string attributes single input string is provided. This is done using html input fields.

5.2. JavaScript check() function in search.html

Proper validation is provided in search.html search form using this JavaScript method. For instance, the ‘To’ value users enter for numerical attributes have to be greater than ‘From’, otherwise an alert will pop up. Similarly, users can search with one attribute or multiple attributes. Users have to enter both ‘from’ and ‘to’ of numerical attributes, if they have to query that attribute on the datastore. Leaving either one of them empty, will cause a dialog box to pop up to ensure that users enter both the fields. This is done to ensure that querying is properly validated and makes sense to the user.

6. Edit, Delete, Review and Information page of an EV

6.1. List class in list.py

6.1.1. get()

List of EVs is fetched from the database and stored in an object which will rendered to list.html with each EV name as hyperlink. This method also renders values to information page(evinfo.html) which is the page that is displayed when the hyperlink is clicked and URLs are generated dynamically for each EV. Id of each EV is passed using HashMap to information page where information of that particular EV will be displayed according to their id. In the information page the average rating and reviews of each EV in reverse chronological order are also displayed. For this EVReview (data object for review and rating created based on each EV’s id) query is created and filtered based on matching id. For displaying reviews in reverse chronological order, the query is rendered to the information page using jinja2 template. The average rating is calculated in get method and rendered in information page.

The presence of an id value is used to determine the page to be rendered. If an id is present EV information page(evinfo.html) is rendered, otherwise list page containing the list of EVS will be rendered.

6.1.2. post()

This method stores the button value in action variable and decides the actions to be completed based on that. Id of the EV is rendered to edit page(edit.html) to automatically display the EV info in the edit page. When action is Delete, the EV will be deleted from the database according to the id. Edit and delete options will only be available for a logged in user because, the presence of a user object is checked in the EV information page before displaying the buttons. Edit and Delete buttons will only be available for logged in user. When action is ‘Review’, EV object is rendered to the review page (reviewform.html). This method also redirects Back option to the page with list of EVs.

6.2. Edit class in list.py

6.2.1. get()

This method handles HTTP Get request for Edit class.

6.2.2. post()

When the action is update, this method gets the id of the EV to be updated and updates its attributes. Here also, it is checked if the name, manufacturer and year are same with an existing EV. Success or error message is also rendered according to this condition. EV will be updated only if the condition is met. This message is rendered to edit page. This method also redirects the user to the page where EV list is displayed, when the user clicks on the Cancel button.

6.3. JavaScript mysubmit() function in evinfo.html

This function is used to ask the user for confirmation before a delete action is performed. This is to ensure that no data gets accidentally deleted from the datastore.

Conditional checks are performed in list.html to display “There are no EVS in the database currently” message using jinja2, when the EV datastore is empty.

7. Review and rating of an EV

7.1. ReviewForm class in reviewform.py

7.1.1. get()

This method renders the current user object and the EVReview object to reviewform.html. Id of each EVReview will be set to default.

7.1.2. post()

When ‘Submit Review’ option is selected in the information page(evinfo.html) of an EV, the id of that particular EV is retrieved and it is stored as ‘ev_id’ along with the review (text area in reviewform.html) and rating (number in reviewform.html) values in the EVReview Datastore. This method also redirects user to the information page when cancel button is clicked from Review (reviewform.html) page.

8. Comparison of EVS

8.1. Compare class in compare.py

8.1.1. get()

This method renders the list of EVs to the compare page(compare.html) to be displayed as checkboxes.

8.1.2. post()

This method handles HTTP post request. Here all of the checked values in the compare page are obtained and stored in a list. When 2 or more EVs are selected for comparison, in this method their ids are obtained and their average rating, maximum and minimum of all the ratings, costs, power, WLTP_Range etc are calculated. These calculated values are rendered as HashMap to the comparison result (compare-result.html) page where comparison is displayed side by side for all EVS. In the page these maximum, minimum values will be checked for equality with the current values and colour will be decided accordingly.

8.2. JavaScript check() function in compare.html

This function is used to ensure that two or more EVs are to checked for comparison. Otherwise a dialog box will pop up in the compare page asking the user to select two or more checkboxes.

Conditional checks are performed in compare-result.html to display "There are no EVS in the database currently" message using jinja2, when the EV datastore is empty. Internal CSS is used to display red, green colours for attributes depending upon highest and lowest values.

6. Data Structure and Models

6.1. Models

Three data models are created and used for the purpose of this assignment.

1. **MyUser in myuser.py**

In this file, there is MyUser class which extends ndb Model. This data model stores the email address of the users as StringProperty.

a. 'emailaddress': Email address contains alphabets, numbers and characters like @. Thus, string is the best choice for it. It also allows for the usage of Google App Engine's default login-logout service.

2. **EV in ev.py**

EV data model in this file stores the details of the electric vehicle and extends ndb. Model.

a. 'name' is declared as StringProperty. Name of EV is stored as string, because name of an Electric Vehicle contains mostly alphabets with occasional characters or numbers
b. 'manufacturer' is declared as StringProperty. Manufacturer of vehicles will usually be alphabetic characters with occasional numbers. StringProperty accepts both.
c. 'year' is declared as IntegerProperty. In the add.html page limits are included for year field 1830-2020. This choice was made after going through information about the first Electric Vehicle. This was a choice that was made only to ensure proper validation. Incorrect integer values of year, if allowed will destroy the integrity of the data.
d. 'battery_size', 'Cost', 'Power', 'WLTP_Range' are declared as FloatProperty.
e. 'cost' is declared as FloatProperty.
f. 'power' is declared as FloatProperty.
g. 'WLTP_Range' is declared as FloatProperty.

Battery Size, Cost, Power and WLTP_Range attributes of Electric Vehicles tend to have decimal values; thus, they are declared as FloatProperty. However, for validation, only two decimal places are allowed for cost and three decimal places are allowed for the other three attributes. This is achieved using 'step' option in html.

3. **EVReview in ev.py**

EVReview data model was designed to store review and rating separately from the EV Data Model. This choice was made to ensure integrity and to allow for easier maintenance, modification and updates in the future. ID is used to associate the EV object and its review and rating.

The EVReview class extends ndb Model and has two attributes

1. ev_id - This allows the association of the id of particular EV objects to their respective review and rating
2. review- review is declared as StringProperty because the reviews submitted by users can contain different kind of characters. The maximum length of characters is 1000, this is validates using html 'maxlength' attribute in reviewform.html
3. rating- rating is declared as IntegerProperty since in the reviewform.html users are allowed to submit a value ranging from 1 to 10 as an integer only.
4. rv_date - This attribute has a DateProperty with auto_now_add set to True. This attribute was added to understand the review, rating and its association to certain time periods. Customers may have different reviews and ratings during different times of the year, with different kind of promotions or strategies etc. This also allows us to show the reviews in reverse chronological order in compare and information page of the EV.

6.2. Data Structures

6.2.1. Dictionaries

Dictionaries offer powerful functionalities like hash maps. Throughout this project, hash map of all the values that is to be rendered in the template are used. This is particularly useful with jinja2 to render content, message, entity objects etc.

6.2.2. Lists

Lists are used in Compare class to store all the numerical attributes and to find the maximum and minimum of those attributes. Lists were chosen because it is easy to implement as there are max() and min() functions which can be applied on lists.

List is also utilised in Search class, for appending the queries as the number of attributes the user wants to filter on increases. Here also, append() function of lists are easy to use and implement. Application routing table is also provided in the form of a python list.

6.2.3. Sets

In Add class, queries are converted into Sets for intersection with other queries. This is a very useful choice, because it allows for filtering of the data based on multiple queries.

6.2.4. Tuples

As part of defining the application routing table provided in the form of a python list, Tuples are given in the list to ensure that requests to the root of the application at '/' have to invoke the respective methods of an instance of the MainPage class. [1]

7. Design Decisions

Design is a very important part of building a web application. The design decision choices made here are carefully designed after thoughts about increasing the ease of the user interface and also the aesthetics of the overall application. The most important decision, is the choice of a two-column layout – one column for the image and the other for the rendered content. Since this application does minimal functionalities like add, delete, edit and compare, there is no need to allocate the entire width of a device as it is not appealing to see lots of blank space in the webpage. Images also improve the aesthetics and in compare page, image is used to ask the user to select 2 or more EVS. Navigation bar is designed to be minimalistic, hence a simple shadow affect is utilised instead of background colour. The navigation items appear above the fold for easy use.

The advantages of electric vehicles are currently being discussed everywhere alongside the discussions of climate change and environment. A large part of the design decisions of this application comes from this aspect of Electric Vehicles, that is their ability to provide sustainable transportation without releasing harmful fossil fuels. Images used in this application were designed after taking into account, the sustainability and less polluting feature of electric vehicles. Position of the images are kept as fixed in order to give the effect of a background. The background of the images and the webpages are purposefully kept plain white because white is less distracting and it brings the users attention to the content in the image and also the forms, tables and other content in the web pages. White colour also gives a simple look to the application, which goes well with the other CSS choices and prevents a crowded look.

External and internal CSS is used for building this application. However, for the purpose of maintaining a similar template for all webpages, the CSS for navigation bar, tables and input fields are kept in stylesheet.css which is then linked in all html pages. This increases the overall look and feel of the application, because consistency is the key for any good website.

Design decisions including colour choices are made after reading *The Principles of Beautiful Web Design* by Jason Beaird. A complementary colour scheme is used in this application to suit audiences of different age groups and this is interesting for particularly young users. Pink increases excitement of the users, is vibrant and blue is a very soothing and relaxing colour. Green colour is avoided in the content, because it is the most common form of colour deficiency. The colours used in the web application are shades of blue and pink, and they are chosen after understanding colour psychology. Shades of blue are used in the navigation bar and in the tables. Active pages are displayed with fixed blue colour in the navigation bar. This facilitates easy understanding for the user. When hovering on other inactive links, a pink shade is displayed which contrasts with the light blue shade. Colour codes are given as RGB values for proper rendering on all browsers. Colour shades used in the application are kept to minimum, because a lot of colour palettes can confuse and distract the user from the actual content. Subtle shades also allow increased focus on the content and better readability.

Another important design decision, is the use of tables for forms and lists with colours on the even rows. This was chosen to contrast with the plain white background and to match with the colour on the navigation bar. Tables also allow properly formatted display of content with equal spacing. Comparison results are displayed side by side so that the users can easily go through the features of each vehicle, attributes of the EVS are kept as row headers to ensure that the user can focus on the task of distinguishing between different EVS.

Buttons are designed to stand out from the rest of the layout to grab attention. The review button was chosen to be a different blue shade to avoid the monotony while maintaining simplicity. A font stack is used in order to cater to a situation of unavailable fonts with different browsers. The fonts that are chosen are Perpetua, Tahoma and Verdana. This is to match with the overall simple but elegant look of the web application.

References

1. Denby, B., 2017. *Google App Engine By Example*.
2. Google Cloud. 2020. *Python On Google App Engine | App Engine Documentation | Google Cloud*. [online] Available at: <https://cloud.google.com/appengine/docs/python> [Accessed 26 February 2020].
3. George, J., 2014. *Principles of Beautiful Web Design, 3Rd Edition*.