

ELECTRIC VEHICLE- DOCUMENTATION

Methods used in the application code

MainPage

1. **get():**

The get method in MainPage class is called initially when the app is opened. The functionalities defined in this method are:

- Load user data and return to html
The main page welcomes a user with a welcome message and renders a login/logout url. For a guest, a login url and for a logged in user, a logout url is rendered. The current user object is passed to the view (HTML) through template value.
- Create a new user by creating a new object/instance of MyUser model:
The logged in user is queried in the datastore using the MyUser model which is used to store the details (email address) of logged in users. If the user is not found in the datastore, a new MyUser object is created.
- Query an electric vehicle and return the ev object to html to display in the web page:
The query EV functionality allows a user to query an electric vehicle. This is implemented as a form with a get action corresponding to querying an ev. Users can query an ev by providing any details, where the string based properties (name and manufacturer) takes in a single input while the numerical properties takes in a minimum and maximum value, both included in the search.

2. **post():**

The post method in MainPage is executed when a post action is performed in the page. This happens in the following cases:

- When user is creating a new electric vehicle:
Clicking the button to create a new ev redirects the user to create a page.
- When user want to compare already existing electric vehicles based on their properties:
Clicking the button to compare ev's redirects the user to compare the page where the user is shown with all ev's listed, with checkboxes to select those which need to be compared.

Information

3. **get():**

This page displays the information about the electric vehicle and the facility to edit the ev, add review and rating. In the get method,

- Ev name, manufacturer, year are received by url parameters, and these are used to query the datastore to fetch the ev object. This ev object is passed to the view as template value.

- The key is also fetched for this entry which is then passed as template values to the view (html). The key is kept hidden from the user. This key is used when a user edit or delete the ev from the information page.
- Reviews and scores are fetched and passed to the view as template values. The reviews are to be displayed in reverse chronological order. Reviews and scores are stored as structured properties in the model datastore. This property which is stored by default in chronological order is retrieved as a list and function reversed() is used to reverse the list. This data structure is passed onto the view as template values. The score list is fetched and found the average of. This average score is passed to the view as template values.

User reviews for this EV (more recent first listed):
Nothing found

Add review:

Average user score for this EV is **Nothing found** out of 10

Add rating:

- A variable named 'showButtons' is also passed as template value. This variable is set to 1 if a user is logged into the system currently; otherwise it is None. So, in the html view, the functionalities that are restricted only to a logged in user are displayed only if the variable 'showButtons' is set to 1. Otherwise, the buttons are not displayed, thus a guest user will not be able to use those functionalities.

4. post()

Functionalities supported by this method are:

- Handles editing an electric vehicle
A logged in user can edit any properties of that ev other than review and score. This is possible from the Information page of the web app.
The key is used when editing an ev. This key retrieved in the post method is used to get the corresponding ev entity, which is the ev to be edited. The new details of the ev are also retrieved, and the datastore is queried with the new name, manufacturer and year to retrieve an ev object if one exists with these details.
→ If such an ev instance doesn't exist, it means that the change has happened to any of the three properties that are used to uniquely describe an ev: name, manufacturer, and year. In this case, the ev is modified by setting new values of each property, and submitted to datastore using the retrieved key.

→ If such an ev exists, there can be two possibilities. First, it could mean that these three properties are left unchanged. It could also mean that the user modifies the ev name, manufacturer, or year so that now the three of them describe another ev in the datastore. The latter should not be made possible. This is where the key is used. In this case, the key of the original ev retrieved from the html view is compared with the queried ev's (queried with the modified values) key. If both are the same, it should mean that the user is trying to modify the original ev, and the system modifies the ev and returns the user to the mainpage. Else, it means that the user accidentally is trying to give new values to the name, manufacturer, and year which is already existing in the datastore for another ev object. In this case, the user is presented with the error message 'This EV already exists with the same details (name, manufacturer, year)' by being redirected to the error page.

- **Deleting an electric vehicle**
A logged in user can delete an ev by clicking on the delete button in the information page. In the post method, the ev entity key is retrieved from the html view and is used to delete the entity from the datastore.
- **Addition of review and scores**
A logged in user can add a review for the ev, and score out of 10. Both review and score are structured properties of the ELEVEH model, and are appended, and saved.
- **Accessibility button- Back**
Functionality for redirecting users back to the main page.

Compare

5. get():

Compare page lets user compare selected ev's. All ev's will be listed with checkbox when the user goes to this page. From here, the user can select the electric vehicles to compare, and click on the compare button. The get method does the following functionalities:

- All the ev's in the datastore are fetched, and the keys for each of them are also stored in another list. The keys are retrieved by iterating through the list of ev's fetched by querying the entire datastore. The keys are rendered hidden with the checkbox (checkbox value) and are used to retrieve the ev's that the user selects to compare.

6. post():

The compare page allows the user to post to the page in two cases: first, by clicking the 'Compare' button to compare the selected ev's. Other, to go back to the root page, that is the MainPage. The functionalities written and defined in the post method of this class are:

- **Compare**
The checkboxes rendered in the html view have the corresponding keys as its value, and the name of the checkbox element is the same: 'c'. In the post method, 'self.request.get_all('c')' takes in all the selected checkboxes. Thus these keys of those ev's that are selected to be compared, are parsed, ev retrieved, and are passed to another list: 'ele2'. 'ndb.Key(urlsafe=i).get()' is used to parse an ev entity by passing

the entity key to the `ndb.Key` method. After calculating the average score, it is appended to a score list, and is passed to the html view as template value. There are two other lists: `low[]` and `high[]` which stores the lowest and highest corresponding elements for each property. These lists are also passed onto view as template values along with the list of chosen ev's: 'ele2' and the list of all entity keys.

- Finding the average score:
The average score is calculated by dividing the sum of all the scores of the ev by the number of scores added.
- Creating the minimum and maximum values list for all properties:
For highlighting the minimum and maximum values of each property (in red and green), two lists: `low[]` and `high[]` are used. Also, 6 lists are created for 6 different numerical properties of an ev. These lists are: `allyears`, `allbatterysize`, `allwltp range`, `allcost`, `allpower`, `allrating`. In the `post` method, while parsing the entities that are selected to be compared, the values of each property of each ev entity is appended to the corresponding list. Thus, when all the selected entries are parsed, `allyears` for example will contain all the years of all the ev's to be compared. The minimum and maximum value among each of these lists is found out, and all the minimum values for each of the properties will be appended to list 'low', and the highest values will be appended to the list 'high'. These lists will be used to check the value rendered in view html for `compare.html` and to decide whether to highlight the table cell red or green. In the `compare` results in `comapre.html`, the table cells are highlighted green if it's the highest value and red if it's the lowest value for all numerical properties except for cost.
- Accessibility- Back
Functionality for redirecting users back to the main page.

Create

The create page is used to create a new electricvehicle. This page takes in the different parameters of an electric vehicle which are: name, manufacturer, year, battery size, wltp range, cost and power. Name and manufacturer are strings while everything else other than year are floating point properties. Year is defined as an integer property. Reviews and ratings for an ev cannot be added when creating the ev.

7. `get()`:

The `get` method returns a view defined in `create.html` file. This file has the html defined with a form which posts the user inputs to `Create` class.

8. `post()`:

The `post` method handles the create ev functionality. When the user clicks on the create ev button, the form is posted. The user inputs containing the properties of the ev are striped and taken into each variable and saved. The major underlying condition when creating or editing an ev is that there should be no two ev's with the same name, manufacturer and year. This is checked in here by querying the entire datastore and filtering the result set for entries with the same name, manufacturer and year. These three separate result sets are retrieved as lists

and are checked to see if all of them contain at least one item. If yes, then the user is redirected to an error page which displays the error message "This EV already exists with the same details (name, manufacturer, year)". If the ev is created successfully, i.e, the ev object is created and all properties set and inserted into the datastore (*by `ele.put()`*), the page redirects to the MainPage. A guest user (no user is logged in) cannot normally access this page. But if the page was already loaded in a tab in the browser before the user logs out of the app in another tab, still the user will be shown the error message 'You are not logged in' if a guest user manages to click on the create button.

Data Structures and models used:

Datastructures used:

1. `ele` : list of users that are the result of the query `ev` functionality in the main page. This list contains all the user details, and are parsed in the html jinja code.
2. `_key` : an `ev` entity key fetched from `infomtaion.py` get method and passed to the view for rendering hidden in the page.
3. `reviews` : list of reviews that an `ev` entity has. This list is passed from `information.py` get method to its view via template values.
4. `ele2` : list of electric vehicles that are chosen by the user to be compared. This list of `ev`'s is fetched in the `compare.py` post method and passed as template values to the view.
5. `low` : list of lowest values of each property for the `ev`'s that are chosen to compare.
6. `high` : list of highest values of each property for the `ev`'s that are chosen to compare.

Models used:

1. ELEVEH

```
name=ndb.StringProperty()
manufacturer=ndb.StringProperty()
year=ndb.IntegerProperty()
batterySize=ndb.FloatProperty()
WLTPRange=ndb.FloatProperty()
cost=ndb.FloatProperty()
power=ndb.FloatProperty()
review=ndb.StringProperty(repeated=True)
score=ndb.IntegerProperty(repeated=True)
```

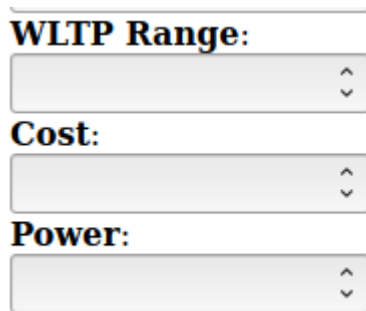
2. MyUser

```
email_address=ndb.StringProperty()
```

Every user has an email address field which is taken as input when a user logs in. The user is created with the corresponding email address.

Application design:

1. Numerical input fields in the HTML are set as type 'number' so that a check for the content in the input fields is implemented. This will inform the user of any errors, and make sure the user submits correct data types. For eg. in the following image, the WLTP Range can be incremented or decremented by the small arrow buttons on the right hand side of the input field. Users can also enter a number manually. Every property other than name and manufacturer are numerical fields; of these, year doesn't take floating point numbers as input.



The image shows three vertically stacked input fields. Each field has a label to its left and a small up/down arrow icon on its right side. The labels are 'WLTP Range:', 'Cost:', and 'Power:'.

2. For editing a data field, the text is already pre populated in the field, thus the user can easily modify or edit a small part of the original value if desired. In the information page, the information about the ev selected is shown.

View/Edit/Delete

Ev name	Ev manufacturer	Ev year	Ev battery size	
Niro	Kia	2020	1.56	28

The data values are displayed in input text fields, which enable them to be edited if the user wishes to.

3. Use of a separate error page: A html page error.html is used to redirect the user to in case any errors occur, along with showing an appropriate error message with a back button to go to the main page. User is presented with the text 'Nothing found' if no review or rating is added to the electric vehicle. This page acts as a single place of redirection for any error in the web app.