<u>Building a GPU Database</u>

Introduction: -

This documentation deals with building an interactive database about GPU's (Graphic Processing Units) and the feature's they support. Python (version 2.x) language is used for the server-side programming, NoSql is used as type database for the system, HTML is used as front-end language. The application is launched on Google cloud SDK for testing.

There are various feature's in the application:

1. Login/Logout System
2. adding GPUs and its feature
3. Viewing the different GPUs
4. Editing with GPU database
5. View by Feature for specific GPU
6. Comparison of two different GPU

Method Overview: -

1. **MainPage class:** The MainPage class is the main method of the application in **main.py** file. The method begins with routing the application to the mainpage_guest.html page by defining the route ('/') in WSGIApplication (function of webapp2). The MainPage method contains GET method which we create a Login URL for the user to login into the application. This class contain two function a GET function and POST function.
    a) **GET Method: -** This method is defined in MainPage class of the application. This method is responsible for responding to GET HTTP and rendering the result of GET request to the class. The first line in GET method is showing the content of the HTML file on the web browser. The Login/Logout service of the application is defined in the GET method. When the user starts the application the method route to **mainpage_guest.html** where a Login link will be generated for the user to login into the application. After user successfully logins into the application a Logout

link will be created on the MainPage in **mainpage.html**. The MainPage has query() variable that will display the GPU list the user has entered in the database. In GET method the user authentication is also checked that user is logged in or not.

   b) **POST Method: -** This method is defined in MainPage class of the application. This method is responsible for responding to POST HTTP and rendering the result of POST request to the class. The first line of the POST method is showing the content of the HTML file on the browser. The MainPage of the application shows the form to enter the GPU details to the database. The if-else of the POST method checks if the GPU exist (duplicates) then it will print message that "GPU already exist" else the GPU will be added to the database of the application. The append() function is used to add GPU into the datastore of the application. The application is routed to the **mainpage.html ('/').** The key of GPUs is the name of the GPU. The name of the GPU on the MainPage is a hyperlink that will redirect the application to the view page.

2. **View class: -** The View class is the view method of the application in the **view.py** file. This page is routed by ('/view') in WSGIApplication of the **main.py** file. This method has GET method that will view the GPUs detail that is stored in database of the application. The user authentication is also done in the GET method for checking if user is logged in or not, to create a logout link for the user.

   a) **GET Method: -** This method is defined in the View class. We creating a fetch() and query() variable to fetch the value of the GPUs from the database and then printing the GPUs detail in table view. An UPDATE link is created in the table column for updating the GPU details.

3. **Edit class: -** The Edit class is the edit method of the application in the **edit.py** file. This page is routed by ('/Edit') URL in the **main.py** file. The **edit.py** file has GET method and POST method. The user authentication is done in the GET method for checking if the user is logged into the application and logout link is created on **edit.html** page.

a) **GET Method: -** This method is defined in the Edit class. The HTML content of the edit.html is printed on the web browser using self.response.headers. The GET method is used to authenticate the user if the user is logged into the application and a logout link is created on the edit page. The GPUs name is the key of the model of the datastore using my_gpu variable. If there are no GPUs in the edit page it will be redirected to the MainPage ('/'). There is template value called 'checked' for the Boolean type features.

b) **POST Method: -** This method is defined in the Edit class. The post method is used to edit the GPU information. All of the GPU information can be edited but name of the GPU cannot be changed. The if-else statement of the UPDATE and CANCEL button will be redirection of the HTML pages. The UPDATE button will redirect the page to View page ('/view') to view the changed GPU information. The CANCEL button (elif part) will redirect the page to MainPage ('/'). The nested if-else of the POST method will change the value of GPUs features as TRUE or FALSE.

4. **MyFeature class: -** The MyFeature class is the view by feature of the application in the **myfeature.py** file. This page is routed by ('/myfeature') URL in the **main.py** file. The myfeature class has GET method to get GPUs for specific features. The user authentication is also in GET method and if the user is logged into the application a logout link is created on myfeature page.

   a) **GET Method: -** This method is defined in the MyFeature class. The HTML content of myfeature.html is render on web browser by using jinja. The nested if-else is used to get value of the feature of GPUs either they are TRUE or FALSE. A query variable is created to check if the feature is present in the GPU to print it. A fetch query is used to print the GPU information in the table view using **myfeature.html** file. If we input two features in the form and if there are GPU in the datastore that have that feature it can be viewed in the Myfeature page in the table view.

5. **Compare class: -** The Compare class is the comparison of the two GPUs in the application in **compare.py** file. This feature is routed by ('/compare') URL in the **main.py.** The compare class has GET method to compare two GPU information. The user authentication is also done in GET method. If the user is logged into the application a logout link created on the compare page.
   a) **GET Method: -** This method is defined in the compare class. The HTML content of compare.html is printed on the web browser. The myuser variable is created for user is the key to authenticate a user is logged into the application. A logout link is also created on compare page. The compare button is created on the main page of the application. For comparison two GPU are selected and when we click the compare button, the application is routed to ('/compare') compare page and two GPU are shown in table view. The checkbox variable is used allow GPU names to compare the values, and query variable is used only two GPU can be compared not more nor less. Data variable is used to fetch() the value of two GPU and view the value in compare page.

Model Overview: -

1. **Address class: -** The Address class is defined in the **address.py** file. The model of the application is defined in this class. To use the datastore with our own class we have hooked the data onto the ndb models. The ndb module is imported through google.appengine.ext. The ndb module is used to create, store and retrieve objects from the database. The GPU has nine properties altogether. The GPU has a name (StringProperty), manufacturer (StringProperty), dateissued (DateProperty). The GPUs has six features that are Boolean properties that can be TRUE or FALSE. the six features are geometryShader, TesselationShader, shaderInt16, sparseBinding, textureCompressionETC2, vertexPipelineStoreandAtomics all are Boolean Properties. The GPU name is the key is data model. So, all thing is retrieved from the GPU name.
2. **MyUser class: -** The MyUser class is defined in the **myuser.py** file. The ndb module is imported from google appengine. It is used to create, store and retrieve object from the

NoSql database. The username is stored in the datastore and can be retrieved. It is a String Property. We are importing Address class into the MyUser class to link both the class. The Structure Property is used as we introduce another class ‘Address’ as a property of this class. The property is also repeated=TRUE to store multiple address object into the datastore. The datastore is responsible for generating name key for each GPU (because of structure property) are linking them to parent class.

The model and datastructure used in the application are all the requirement of the application. A GPU has a name, manufacturer and when the GPU is issued (date). The GPU has some feature that can be in some GPU and not in some that why it is a boolean property. The choice of reasoning structure is structure property to hold different properties (string, date, boolean) in one structure. To store multiple value in the datastore we are using a repeated property.

User Interface (UI) Overview: - The user interface UI of the application is designed for the user-friendly interface. There are forms, table that are uniquely designed to show the data easily to the user. The form is aligned to the application and table are all horizontal and have small horizontal distance for form and table.

- **mainpage_guest.html: -** When user opens the application the first page that is rendered by MainPage class is the **mainpage_guest.html** page. A Login link is created using expression delimiters rendered by jinja and webapp2 modules. The template values are passed through the python file into the HTML file.
- **mainpage.html: -** This is the main page of the application. When a user logged into the application and a logout link is created in the mainpage using logout_url template value in main.py file. A horizontal navigation bar is also created on main page to navigate to other feature of the application e.g View by feature, view the GPU, logout. A form is also created to add the GPU information into the application. The form is perfectly aligned and there are equal gaps between form boxes and checkboxes. A table is also created below the form to view the GPU the user is added in the datastore. The GPUs name, manufacturer, dateissued can be viewed on the main page. There is column in the table

  where there are checkboxes for comparison of the two GPU and there is compare button in top the column. The labels and checkbox are perfectly placed and small distance shows that they belong together. The name of the GPU is a hyperlink that can directed to the view page of the application.

  - **view.html: -** The view page of the application is the page where user can see the GPUs whole information, he added into the datastore. The name link in mainpage.html can be redirected to this page. The page has table view of the GPU information that are added into the database. The table has css style to make table viewable and all value is assigned in the table. The last column of the table is the UPDATE link to update the information of the GPUs.
  - **Edit.html: -** The edit page of the application is the page where user can edit the information of GPU. There is a form for editing the information but name of the GPU cannot be edited. The form is aligned and all boxes and checkboxes are all aligned and equally gap. There are two button UPDATE and CANCEL. After user changes the information in the form and click UPDATE button the information gets updated and the page is redirected to view.html (to view the information of GPU). If a user clicks CANCEL button the page is redirected to the mainpage of the application.

- **Myfeature.html: -** The myfeature page of the application is the page where user can view the GPU by its feature. There is a form that has checkbox to select the feature for which the user wants see the GPU information. There is table where we can view the GPU information for which the feature is selected. The checkbox is aligned and perfectly gaped on the page. The GPU information can be shown in the table that is retrieved from the database. The navigation bar is also there to direct to the mainpage.
- **Compare.html: -** The compare page of the application is the page where user can view the two GPU information that they have selected from the mainpage and clicked the compare button. The table view is horizontal one and the table value is equally placed and the table rows are at equal gap and there also navigational bar at the top of the page to get back to the main page to select other GPUs to compare.

The User Interface is selected for the application so the user can easily access the application and can use the feature of the application. The color chosen for the application is highly contrast so anyone can view it easily and differentiate between the text and hyperlinks, table and forms.

SUBMITTED BY: - LAKSHAY CHAUHAN - 2986744