Liston Lucio Mascarenhas

Msc. Big Data Management and Analytics

2970414

**Methods**

1. **Main Page**

The get method in main.py initially allows the user to login to the application. If the authentication is successful, the users email address is stored in the NoSQL database and is shown a navigation board where all the different features of the application are shown and a list of all GPUs present in the database are displayed to the user.main.py is rendered using main.html.

In main.py we also define the application object that is responsible for this application. The routing table is specified in this object.

1. **Add GPU and Edit GPU**

addgpu.py allows a user to add a GPU to the database. On filling the form and clicking the update button, the post method is triggered, and the details of the GPU are stored in the NoSQL database. The GPU name is stored as the key and a condition is kept checking that every key is unique, so a GPU is not overwritten. The user is then redirected to the main page.

addgpu.py is also used when editing a GPU. In the get method, on accessing the GPU name from the URL parameter, we use it as a key to fetch its information and features to populate the form. The GPU name cannot be edited and is set to read only in addgpu.html. gpu\_edit is passed to the URL parameter to check if a new GPU is added or an already present GPU is being edited. addgpu.py is rendered using addgpu.html. ‘if’ conditions are used to toggle the state of the features. On clicking Update the post method is triggered where the details in the form are stored in the datastore and the user is then redirected to the main page.

If the user accesses this page from the main page, the Back button redirects the user back to the main page. If the user accesses this page from gpuInformation, the Back button redirects the user back to gpuInformation.

1. **GPU Information**

In the get method in gpuinformation.py, on accessing the GPU name from the URL parameter we use it as a key to fetch its information and features to populate the page.

In the post method, the Edit button redirects the user to addGpu. On clicking on the Back button, the user is redirected to the main page.

gpuInformation.py is rendered using gpuInformation.html.

1. **Query**

In the get method in gpuquery.py, we request all the features from the URL parameter.If there are no features, it will store the value of the variables as ‘’.We also create a function which converts the value from a string to boolean. If the search parameter is requested and the value is yes, we filter the query based on all features selected. All the GPUs that satisfy the query are displayed as a list. In the post method, on click of the Search button, all the features are redirected back to the same page i.e. gpuQuery with all the features passed as parameters. On click of the Back button, the user is then redirected to the main page. gpuquery.py is rendered using gpuquery.html.

1. **GPU Compare**

gpucompare.py allows the user to select 2 GPUs. In the get method, we access gpu\_same and send it to the html page. If the value is ‘yes’, we display a message to choose 2 different GPUs.

In the post method, we accept the values of the 2 GPUs. If they are the same,the user is redirected to the same page i.e.gpuCompare with the parameter gpu\_same with a value as yes. If the GPUs are different, the user is redirected to gpuCompareResults and passes the 2 GPU names as parameters. On clicking the back button, the user is redirected to the home page. gpucompare.py is rendered using gpucompare.html.

1. **GPU Comparison Results**

gpucompareresults.py does a feature by feature comparison of 2 distinct GPUs. In the get method, on accessing the 2 GPU names from the URL parameter we use it as keys to fetch their information and features to populate the page. In the post method, on clicking the back button, the user is redirected to the previous page which is gpuCompare. gpucompareresults.py is rendered using gpucompareresults.html.

1. **app.yaml**

app.yaml is responsible for informing Google App Engine about the runtimes and libraries needed for the application. Python version 2.7 is used as the runtime. We also state that the application is threadsafe so multiple instances can be allowed on the same server. In libraries we state that we will be using Jinja2 running on its latest version. In handlers, we state that all URLs with /css will be redirected to the static directory css. We also state that all other requests will be redirected to the app variable which is defined in main.py.

**Models and data structures**

1. **MyUser**

myuser.py contains the class MyUser which uses ndb.Model to store the email address of new users. This is possible by importing ndb from google.appengine.ext. email\_address is specified as a StringProperty() since an email address is of ASCII format.

1. **GPU Features**

gpufeatures.py contains the class GpuFeatures which uses ndb.Model to store the GPUs information and features. This is possible by importing ndb from google.appengine.ext. GPU name, GPU manufacturer are specified as StringProperty() since they are ASCII format whereas GPU date issued is specified as a DateProperty(). This is done because the GPU date issued can only allow a specific format to be stored as a date in the datastore and not just any string. geometryShader, tesselationShader, shaderInt16, sparseBinding, textureCompressionETC2 and vertexPipelineStoresAndAtomics are specified as BooleanProperty() because they are supposed to have one of only 2 values (True or false).

**Designing the UI**

The approach taken to designing this web application is minimalism. #fdfdfd which has an hsl of (0,0%,99%) is used as the background colour rather than traditional white (0,0%,100%) which gives a certain nuance.

The hyperlinks are set to blue while normal texts are black since the eyes are familiar with primary shades and colours.

Lists are displayed without any borders to keep the application minimalistic.

Buttons have a margin of 1dp to indicate they are part of the same design.

Labels and input fields have a small horizontal distance to indicate they belong together.

Pages with lists are displayed in the centre while pages with forms or information are positioned to the left with a margin of 3dp from the top and left to not look cluttered.

Warning messages are displayed in red so it is relatively visible over the background.