Hafeez

Building A GPU DATABASE

# Introduction

In this assignment I will be tasked with building an interactive database about GPUs (Graphics Processing Units) and the features they support. I will be expected to use a NoSQL type database as storage for this system.

This application capable of the following,

• Add in a GPU with a series of features.

• Edit a currently existing GPU and its features.

• Be able to choose a set of supporting features and figure out which GPUs support them.

This application will be similar to the format found at **http://vulkan. gpuinfo.org/listreports.php**. If I click on any of the GPUs listed here it will bring I to another page. The information we are interested in here is listed under the “features” tab.

I do require to map six features of a GPU which are the following:

* geometryShader
* tesselationShader
* shaderInt16
* sparseBinding
* textureCompressionETC2
* vertexPipelineStoresAndAtomics.

Each GPU stored should contain the information for these six properties. When a GPU is added a user should be able to click a series of checkboxes (or similar) to enable or disable features. When an add button is clicked there should be no duplicate names in the database or pre-existing objects in the database for that GPU.

# Technical Stack

To develop this application, we choose stack as below.

Python for programming platform and jinja2 for template programming. HTML, CSS, JavaScript for developing interactive forms and tables and navigations and hyperlinks. Bootstrap 4 for look and feel of the application. Bootstrap 4 is the newest version of Bootstrap, which is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites. Bootstrap 4 is completely free to download and use. Google cloud platform quickly reach customers and end users by deploying web apps on App Engine. With zero-config deployments and zero server management, App Engine allows you to focus on writing code. Plus, App Engine automatically scales to support sudden traffic spikes without provisioning, patching, or monitoring.

* Python.
* Jinja2.
* HTML.
* JavaScript.
* CSS.
* Bootstrap.
* GCP NoSQL Database (NDB).
* GCP Platform.(App Engine)

# Coding Brackets

## Bracket 1

* Get the shell of the application with login/logout working.
* Generate a model that will store the information of a GPU: name, manufacturer, date issued.
* Add the six properties listed in the introduction to the model of a GPU.
* the GPU name should be the key for each GPU in the database.

### HomeController: This a base controller and render welcome page for user.

#### Get():

* This method performs user credential validation and redirect to home page.
* When we load home page, Application will generate database models as we configured in models.py program.
* You can able to see two models at Database as GPUEntity and Users.
* GPUEntity model we have columns as given in project abstract.
* User model have name and email columns to store user details.

## Bracket 2

* Build a UI form that will enable the user to add a GPU and all its information.
* When the form is submitted the GPU should be added to the database

### AddGPUController:

#### Get():

* This method will not read any request param. But it checks whether current user session valid or not.
* Once user valid, then it will render Add GPU page.
* Where user can add GPU details by using this UI form.

## Bracket 3

* Prevent the overwriting of an object that is already in the database.
* Bracket Failure if object overwritten
* Display a list of GPUs that are currently in the database by name only.

### AddGPUController:

#### Post():

* This method will take form data such as name, manufacturer, date, geometryShader, tesselationShader, shaderInt16, sparseBinding, textureCompressionETC2 and vertexPipelineStoresAndAtomics.
* Once we get all required inputs, we are query to database table called GPUEntity with name filed by given name value.
* If GPU name already existing then we are simply rendering an error message and showing message to end user as **Failed! With Same GPU Name, Record Already Existing.**
* If GPU name not there then we are saving the input object in to NDB by using put method and rendering a success message as **Success! Record Inserted Successfully** to same page.
* After Insertion done successfully then we are rendering list of GPU Entities with hyperlinks to same page right panel. In left panel you can see GPU input form.

## Bracket 4

* Make the GPU name list a set of hyperlinks.
* When a GPU name is clicked it should go to a separate page showing the information and features for that GPU Bracket Failure if not on a separate page.

### GPUInfoController:

#### Get():

* List of GPU Information listed in Add GPU page itself.
* From there When user clicks the hyperlink, handle will come under GPUInfoController.
* Then, this method will take a parameter of GPU name and query to the database.
* Fetch GPU information and open a separate page and populates the GPU entity information.

## Bracket 5

* Enable editing of a GPU in the database.
* The form or editing a GPU should be prepopulated with the existing information of the GPU in the database.
* Editing must be done on a different page Bracket failure if not on a separate page

### GPUEditController:

#### Get():

* When user clicks the edit button in GPU Info page, handle will come under GPUEditController.
* Then, this method will take a parameter of GPU name and query to the database.
* Fetch GPU information and open a separate page and populates the GPU entity information and fill in form with default values.
* From there Use can modify GPU Entity information and Save the changes by clicking Save Changes button.
* Once User clicked the button then form data will posted and called POST method.

#### Post():

* This method validates the input data and update the GPU Entity in Database. Once it is successfully saved then end user will get a message like **Success! GPU Information inserted successfully.**
* Or else if any error occurred while validating then end user will get following error like **Failed! GPU Information not updated.**

## Bracket 6

* Enable the user to select the features to query by using checkboxes (or similar)
* Enable querying of the database using a Boolean combination of user selected features.

### SearchGPUController:

#### Get():

* This method will not take any query parameter. It will load the page with features as checkboxes shown in Search GPU page.
* Once user click on required check boxes then click for search GPU button. Then in the same page user can able to see associated GPU information list.
* Here end user will have provision like he can select one or more checkboxes and can see related records in same page.

## Bracket 7

* Add in a form that permits the user to choose two GPUs for comparison purposes.
* When the comparison is triggered a separate page should be displayed Bracket Failure if not separate page
* The feature-by-feature comparison of both GPUs should be shown on the separate page

### CompareGPUController:

#### Get():

* This method will load all GPU entities in a table format and with check box in each row as shown in compare gpu page.
* Here we given a proviso to end user, where he can choose any two rows in table and clicks on Compare Features button.
* Then User will navigate to separate page and will populate checked GPU Entitles list in other page.

### CompareViewController:

#### Get():

* This method will take GPU names as input parameter and Hit the database with this name using NDB.OR query and get the results of two entities.
* By that results we are populating the features in table format will all compared features as shown in compare view page.

## Bracket 8

* UI design: well, thought out UI that is easy and intuitive to use.

As a developer I take care of all user scenarios and try to visualize in user friendly manner. Where user can do his operations without any confusion. To achieve this UI, I relied on bootstrap CSS framework.

# Data Structure:

* In this application in most cases, we used for each loop and arrays. To iterate each item and compared the indexed value.
* When iterating over a dictionary, you have a few different options. You can iterate over just the keys, just the values, or both the keys and values.
* To access the values of a dictionary in a for loop, you need to use the. values () method. To access both the keys and values of a dictionary using a for loop, you can use the. items () method.
* We used arrays to store objects in it. Arrays give us a way to store and organize data, and we can use the built-in Python methods to retrieve or change that data.
* For example, if you have a list of student names that you want to store, you may want to store them in an array.

# Database Design:

* We have only two database tables in this project one is GPU Entity and User Entity.
* User entity useful for storing user information. Whereas GPU entity is used to store GPU features related information.
* There are no parent child relationship tables in the database. We are depending on GPUEntity model most of times.
* In GPU Entity we made name as unique key and which is not allowed to add duplicate values.

# Resources:

* <https://www.w3schools.com/bootstrap4>
* <https://zetcode.com/python/jinja/>
* <http://w3schools.com/python>
* <http://stackoverflow.com/python>
* <http://stackoverflow.com/jinja2python>

References:

* <https://jinja.palletsprojects.com/en/2.10.x/templates/>
* <https://gosecure.github.io/template-injection-workshop/#0>
* <https://cloud.google.com/appengine/docs/standard/python/ndb>
* <https://googleapis.dev/python/python-ndb/latest/index.html>
* <https://cloud.google.com/datastore/docs/quickstart>