**A01: Building a GPU database of Vulkan features**

**NOTE:** read the whole assignment brief \_rst before implementing it contains very important information

In this assignment you will be tasking with building an interactive database about GPUs (Graphics Processing Units) and the features they support. You

will be expected to use a NoSQL type database as storage for this system.

Your application must be 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

Your application will essential replicate the information found at http://vulkan.

gpuinfo.org/listreports.php

If you click on any of the GPUs listed it will bring you to another page. The

information we are interested in here is listed under the \features" tab. You are

required to map six features of a GPU which are the following: geometryShader,

tesselationShader, shaderInt16, sparseBinding, textureCompressionETC2, and

vertexPipelineStoresAndAtomics. Each GPU should have as its key the name

of the GPU itself and should contain the information for these six properties

When a GPU is added a user should be able to click a series of check-

boxes (or similar) to enable or disable features. When an add button is clicked

there should be no duplicate names in the database or preexisting object in the

database for that GPU.

1

For queries a user should be able to choose a combination of features and

run a query to \_nd the set of GPU names that support all the required features.

NOTE: that you are also required to show me a working example of

your brackets in labs. I will only consider marking those brackets that

I've seen in labs. If you have only shown me the \_rst four brackets

in a lab I will not consider the rest.

**Coding Brackets (70%)**

1. Bracket 1 (10%)

\_ Write the shell of an application that has a working login/logout

Service

2. Bracket 2 (20%)

\_ Generate a model that will store the information of a GPU: name,

manufacturer, date issued

\_ Add the following boolean options to the model: geometryShader,

tesselationShader, shaderInt16, sparseBinding, textureCompressionETC2,

vertexPipelineStoresAndAtomics

\_ GPU name should be the key for each object in the NoSQL database

3. Bracket 3 (30%)

\_ 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.

4. Bracket 4 (40%)

\_ Prevent the overwriting of an object that is already in the database

(Bracket failure if not done)

\_ Display a list of GPUs that are currently in the database by name

Only

5. Bracket 5 (50%)

\_ 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)

\_ Enable editing of a GPU in the database

\_ Editing must be done on a di\_erent page (Bracket failure if not on a

separate page)

6. Bracket 6 (60%)

\_ Enable the user to select the features to query by using checkboxes

\_ Enable querying of the database using a boolean combination of user

selected features

7. Bracket 7 (70%)

\_ Add in a form that permits the user to choose two GPUs for com-

parison 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

**Documentation Brackets (30%)**

8. Bracket 8 (0 to 10%): Document every method in your code from a high

level perspective. i.e. give an overview of what the method does. Do not

copy and paste code you will be penalised for this.

9. Bracket 9 (10 to 20%): Give a high level description of all models and

datastructures used. Explain your choices and reasoning for each model

and datastructure used.

10. Bracket 10 (20% to 30%): Document why you designed the UI the way

you did. This should detail your choices in widget layout and position and

how they make user interaction easier. Examples of what I'm looking for

would be the following

\_ Labels and entry boxes have a small horizontal distance to indicate

they belong together.

\_ The colour scheme was chosen to avoid the main form of colour blind-

ness and produce high contrast for the visually impaired.

\_ NOTE Do not include instructions for how to use the UI.

You will lose marks if you include such instructions.