GPU DATABASE

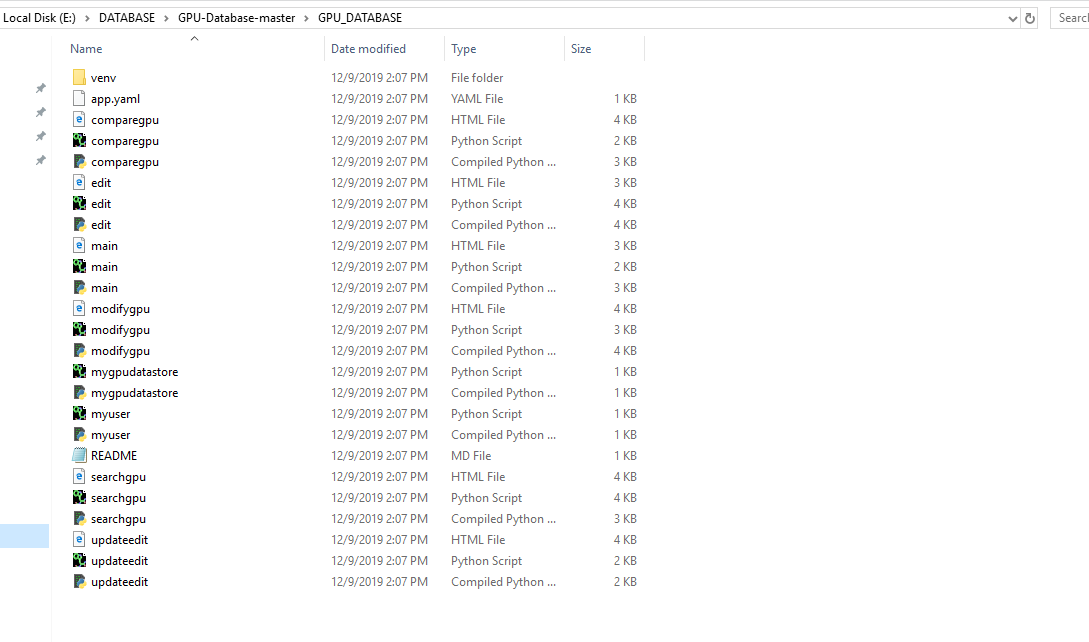
Requirement Files



CONTENT

The following is the codebase for the project

The files are arranged and adopted for modular programming to allow for code reuse and repackaging for further utilization



MAIN PAGE

import webapp2

import jinja2

from google.appengine.api import users

from google.appengine.ext import ndb

from myuser import MyUser

from edit import Edit

from updateedit import UpdateEdit

from searchgpu import SearchGPU

from mygpudatastore import MyGPUDatabase

from comparegpu import CompareGPUDisplay

from modifygpu import ModifyEdit

import os

JINJA\_ENVIRONMENT = jinja2.Environment(

loader=jinja2.FileSystemLoader(os.path.dirname(\_\_file\_\_)),

extensions=['jinja2.ext.autoescape'],

autoescape=True

)

class MainPage(webapp2.RequestHandler):

def get(self):

self.response.headers['content-Type'] = 'text/html'

url = ''

url\_string = ''

welcome = 'Welcome'

myuser = None

user = users.get\_current\_user()

if user:

url = users.create\_logout\_url(self.request.uri)

url\_string = 'Logout'

myuser\_key = ndb.Key('MyUser', user.user\_id())

myuser = myuser\_key.get()

if myuser == None:

welcome = 'Welcome to the application'

myuser = MyUser(id=user.user\_id())

myuser.put()

else:

url = users.create\_login\_url(self.request.uri)

url\_string = 'Login'

gpu\_name\_list\_compare\_query = MyGPUDatabase().query().fetch()

template\_values = {

'url' : url,

'url\_string' : url\_string,

'user' : user,

'welcome' : welcome,

'myuser' : myuser,

'gpu\_name\_list\_query' : gpu\_name\_list\_compare\_query

}

template = JINJA\_ENVIRONMENT.get\_template('main.html')

self.response.write(template.render(template\_values))

app = webapp2.WSGIApplication([

('/',MainPage),

('/edit',Edit),

('/updateedit',UpdateEdit),

('/searchgpu',SearchGPU),

('/comparegpu', CompareGPUDisplay),

('/modifygpu', ModifyEdit)

], debug=True)

MODIFFY>PY

import webapp2

import jinja2

import os

from google.appengine.api import users

from google.appengine.ext import ndb

from datetime import datetime

from myuser import MyUser

from mygpudatastore import MyGPUDatabase

JINJA\_ENVIRONMENT = jinja2.Environment(

loader=jinja2.FileSystemLoader(os.path.dirname(\_\_file\_\_)),

extensions=['jinja2.ext.autoescape'],

autoescape=True

)

class ModifyEdit(webapp2.RequestHandler):

def get(self):

self.response.headers['content-Type'] = 'text/html'

gpu\_name\_clicked = self.request.GET.get("gpu\_name\_list\_display")

gpu\_name\_clicked\_key = ndb.Key("MyGPUDatabase", gpu\_name\_clicked)

clicked\_gpuname\_key = gpu\_name\_clicked\_key.get()

template\_values = {

"clicked\_gpu\_name" : gpu\_name\_clicked,

"clicked\_gpuname\_key" : clicked\_gpuname\_key

}

template = JINJA\_ENVIRONMENT.get\_template("modifygpu.html")

self.response.write(template.render(template\_values))

def post(self):

self.response.headers["Content-Type"] = "text/html"

if self.request.get("update") == "Update":

user\_clicked\_gpuname = self.request.get("gpu\_name\_update")

user\_clicked\_gpuname\_key = ndb.Key("MyGPUDatabase", user\_clicked\_gpuname)

clicked\_gpuname\_key = user\_clicked\_gpuname\_key.get()

gpu\_name\_update\_store = self.request.get("gpu\_name\_update")

gpu\_manufacturing\_update\_store = self.request.get("gpu\_manufacturing\_update")

gpu\_date\_update\_store = datetime.strptime(self.request.get("gpu\_date\_update"),'%Y-%m-%d').date()

geometryShader\_update\_store = bool(self.request.get("geometryShader\_update"))

tesselationShader\_update\_store = bool(self.request.get("tesselationShader\_update"))

shaderInt16\_update\_store = bool(self.request.get("shaderInt16\_update"))

sparseBinding\_update = bool(self.request.get("sparseBinding\_update"))

textureCompressionETC2\_update\_store = bool(self.request.get("textureCompressionETC2\_update"))

vertexPipelineStoresAndAtomics\_update\_store = bool(self.request.get("vertexPipelineStoresAndAtomics\_update"))

clicked\_gpuname\_key.gpumanufacturing = gpu\_manufacturing\_update\_store

clicked\_gpuname\_key.gpudate = gpu\_date\_update\_store

clicked\_gpuname\_key.geometryshader = geometryShader\_update\_store

clicked\_gpuname\_key.tesselationshader = tesselationShader\_update\_store

clicked\_gpuname\_key.shaderint = shaderInt16\_update\_store

clicked\_gpuname\_key.sparsebinding = sparseBinding\_update

clicked\_gpuname\_key.texturecompressionetc = textureCompressionETC2\_update\_store

clicked\_gpuname\_key.vertexpipelinestoresandatomics = vertexPipelineStoresAndAtomics\_update\_store

clicked\_gpuname\_key.put()

self.redirect("/edit")

elif self.request.get('button') == 'Cancel':

self.redirect('/edit')

UpdateEdits

import jinja2

import os

from google.appengine.api import users

from google.appengine.ext import ndb

from datetime import datetime

from myuser import MyUser

from mygpudatastore import MyGPUDatabase

JINJA\_ENVIRONMENT = jinja2.Environment(

loader=jinja2.FileSystemLoader(os.path.dirname(\_\_file\_\_)),

extensions=['jinja2.ext.autoescape'],

autoescape=True

)

class UpdateEdit(webapp2.RequestHandler):

def get(self):

self.response.headers['content-Type'] = 'text/html'

gpu\_name\_clicked = self.request.GET.get("gpu\_name\_list\_display")

gpu\_name\_clicked\_key = ndb.Key("MyGPUDatabase", gpu\_name\_clicked)

clicked\_gpuname\_key = gpu\_name\_clicked\_key.get()

template\_values = {

"clicked\_gpu\_name" : gpu\_name\_clicked,

"clicked\_gpuname\_key" : clicked\_gpuname\_key

}

template = JINJA\_ENVIRONMENT.get\_template("updateedit.html")

self.response.write(template.render(template\_values))

def post(self):

self.response.headers["Content-Type"] = "text/html"

if self.request.get('button') == 'Cancel':

self.redirect('/edit')

GPU COMPARISON ANALYTICS

{% for gpudetails in gpu\_name\_list\_query %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.geometryshader %} {% else %} {% endif %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.tesselationshader %} {% else %} {% endif %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.shaderint %} {% else %} {% endif %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.sparsebinding %} {% else %} {% endif %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.texturecompressionetc %} {% else %} {% endif %} {% endfor %} {% for gpudetails in gpu\_name\_list\_query %} {% if gpudetails.vertexpipelinestoresandatomics %} {% else %} {% endif %} {% endfor %}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Comparsion of GPU** | | | | | |
| **GPU Name** | **{{ gpudetails.gpuname }}** |  |  |  |  |
| **GPU Manufacturing** | **{{ gpudetails.gpumanufacturing }}** |  |  |  |  |
| **GPU Date** | **{{ gpudetails.gpudate }}** |  |  |  |  |
| **Geometry Shader** | {{ gpudetails.geometryshader }} | {{ gpudetails.geometryshader }} |  |  |  |
| **Tesselation Shader** | {{ gpudetails.tesselationshader }} | {{ gpudetails.tesselationshader }} |  |  |  |
| **Shader Int16** | {{ gpudetails.shaderint }} | {{ gpudetails.shaderint }} |  |  |  |
| **Sparse Binding** | {{ gpudetails.sparsebinding }} | {{ gpudetails.sparsebinding }} |  |  |  |
| **Texture Compression ETC2** | {{ gpudetails.texturecompressionetc }} | {{ gpudetails.texturecompressionetc }} |  |  |  |
| **Vertex Pipeline Stores And Atomics** | {{ gpudetails.vertexpipelinestoresandatomics }} | {{ gpudetails.vertexpipelinestoresandatomics }} |  |  |  |

Top of Form



Bottom of Form

# **CHAPTER 3**

# **METHODOLOGY**

## 3.1Introduction.

This chapter focuses on software development techniques, methodology, and development stages with precise deliverables.

## 3.2Software Development Methodology.

The agile model will be used for the this project it is a type of incremental model. This will be appropriate since not all system requirements are clear, and the technology used is not fully understood. This methodology allows regular system adaptation to changing circumstances and even late changes in requirement are allowed.

The agile development model will consist of the following phases: disover phase, design phase, development phase, and test phase. Objective number three which is to implement an efficient data entry mechanism through ML will be divided into two steps, that is

STEP I

Retrieve data from receipts through an interactive UI

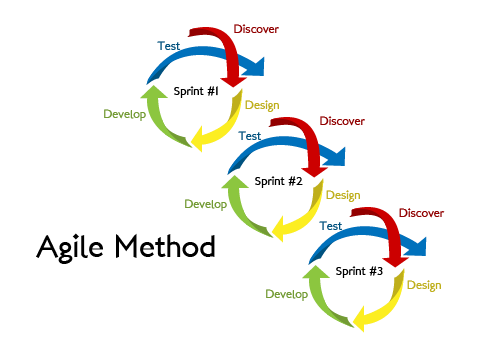
Process data through the GPU model

Populate the processed data into form input fields for data entry.

STEP II

Build the GPU model for execution

Both steps will use the Firebase ML Kit which will assist in the deployment of the model either in the Firebase cloud or on the user device. Firebase ML Kit will assist in testing the functionality of step 1.



**Figure 3.1:** Agile model.

The Agile model will have the following four stages.

### 3.2.1STAGE 1

Implement an authentication system the through implementation of a passcode.

Implement the UI framework of the SEIMMA.

Design and implement the database for the system.

Design and implement the state management system for SIEMMA

### 3.2.2STAGE 2

Implementation of Financial new and trends feature.

Carry out STEP I.

### 3.2.3STAGE 3

The following steps will be carried out during the implementation of the financial news and trends feature.

First, identify the appropriate API endpoint to use to get current financial news and trends, such as the Financial times API. API selection will be based on Ease of Use, availability of Documentation, Price, and Content provided.

Study the API documentation, interpret it, and document how data is structured, how data attributes relate to each other, and how to retrieve these attributes.

Create a Model class to Interface with the structure of the API (e.g. by use of JSON deserialization techniques) and create tests for the Model class.

Display the data to users on the news screen view and cache the retrieved data by storing them in persistent data storage.

Identify a state management library from the variety provided by the Flutter Community and use it to update the data in the cache based on API data updates and retrieve more data based on user interaction.

# CHAPTER 4

# SYSTEM ANALYSIS AND DESIGN

## 4.1Introduction.

This chapter focuses on design and analysis of SEIMMA. This section will discuss the design of the support service architecture, software architecture, use case realization, database, system and UI and system security.

## 4.2Feasibility Analysis.

### 4.2.1Technical Feasibility.

In technical feasibility, I assessed whether the technology proposed for the system was available. There is a wide use of smartphones globally hence, the technology used in this project was available to its system users.

### 4.2.2Operational Feasibility.

This was carried out to determine how the system will operate with limited failures. I decided to localize the application in order eliminate failure due to server error. This ensured this project realizes a reliable system which is only dependent on the users’ smartphone resources.

### 4.2.3Economic Feasibility.

This was done in order to assess the financial aspect of the project. This deals with the project returns’ which is compared to the project investments. The projectis purely educational

## 4.3Requirement Analysis.

Requirement analysis is the process of developing software specifications in order to determine the user expectations for a new or modified product. The activities done during requirement analysis were gathering detailed information from case studies and questionnaires, developing user interface prototypes which were used to improve to improve the system usability, design quality and evaluate requirements with users. Software system requirements were categorized into functional and non-functional requirements.

### 4.3.1Functional Requirements.

These are the processes that the system has to perform which describe the core functionalities of the system. SEIMMA had the following function requirements.

1. **Authentication –** This enabled the system user to verify their identity through a passcode entry.
2. **Generating Reports –** The system creates an overview of the monthly data entries and also provides income and expense reports for both the categories and subcategories. A quadric chart and pie charts are used to help users make sense of their data.
3. **User Functions -** These includes functions performed by the users of the system. This include creation of categories and sub Categories for both income and expense, setting up a pin for authentication and data entry into the system.
4. **Interface Requirement –** This refers to the ease of use of the application, were personal finance manager application users are able to interact with the application and use its components easily.
5. **Security Requirements –** This involves restricted access to the application for unauthorized user; however, this depends on the user’s willingness to setup an authentication system.

### 4.3.2Non-Functional Requirements.

1. Reliability - The system ensures minimum failure by ensuring a low rate of failure occurrence.
2. Usability - The system is easy to understand by new users and will provide user with textual direction on actions to be performed.
3. Availability - The system will work round the clock allowing user to access it at any time.
4. Security – The application will provide an optional authentication system for users in order to meet the user’s preference.
5. Performance - The response time of the system will meet the requirement standards with minimum delay.

Data integrity - the system will ensure overall accuracy, completeness, and consistency of data.