# PHASE 2
# System Design Document

## UniGo

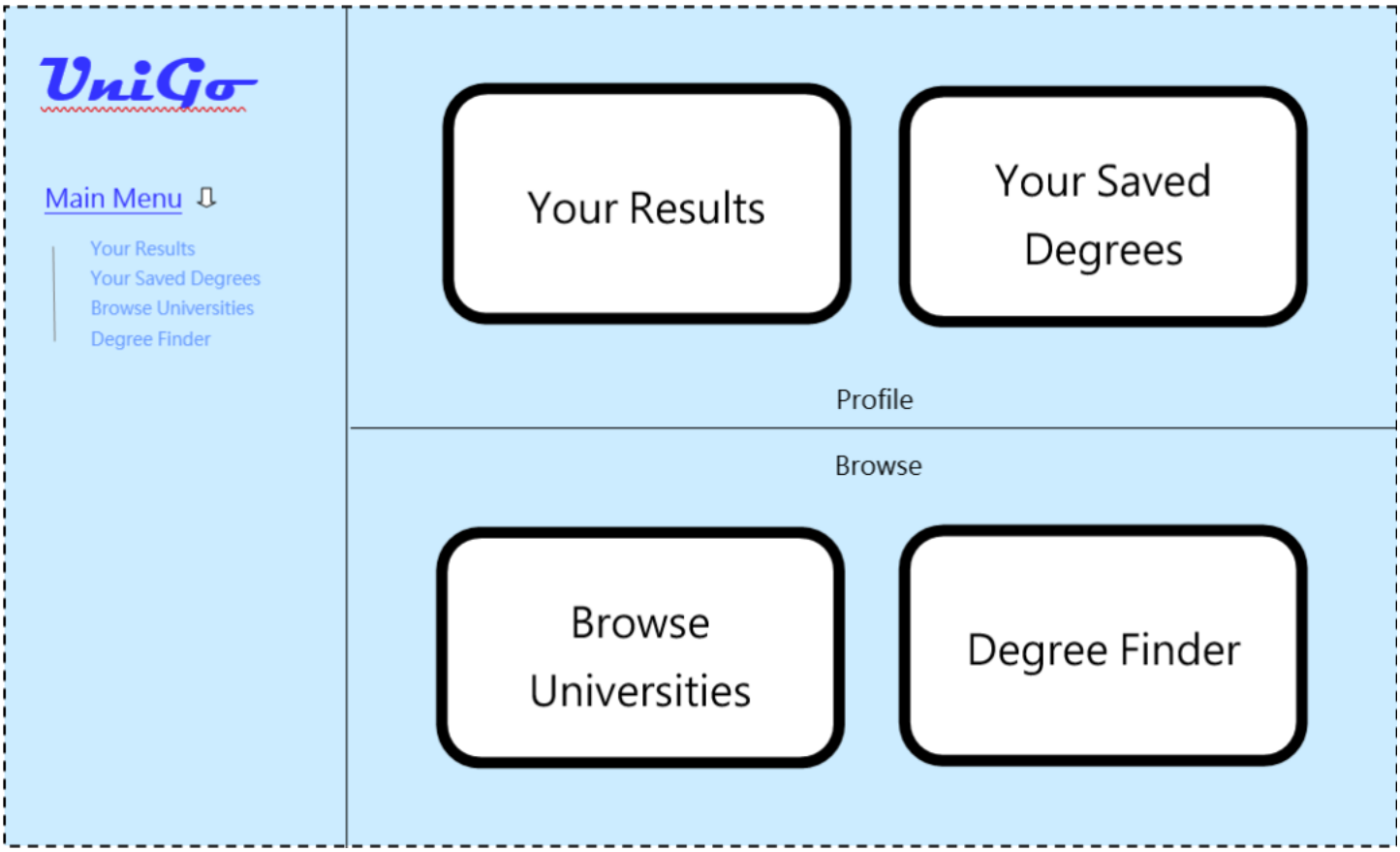

Developed by Saien Naidu

Grade 12
Information Technology
2025

# Table of Contents

# 1. User Interface Design

## 1.1 Main Menu




When the program is opened, the user will be taken to the Main Menu. From here, they can navigate to other screens by clicking on one of the four buttons.

The user opens the program:

### 1.1.1. - Clicking 'Your Results'

- Close the current screen
- Fetch the user's marks from the UserMarks text file
- Populate the screen
- Open the Result Screen

### 1.1.2 - Clicking 'Your Saved Degrees'

- Close the current screen
- Fetch the list of the IDs of the user's saved degrees from the SavedDegrees text file
- Create a list of degree names using the IDs
- Populate the screen
- Open the SavedDegrees screen
- Clear the search bar

### 1.1.3 - Clicking 'Browse Universities'

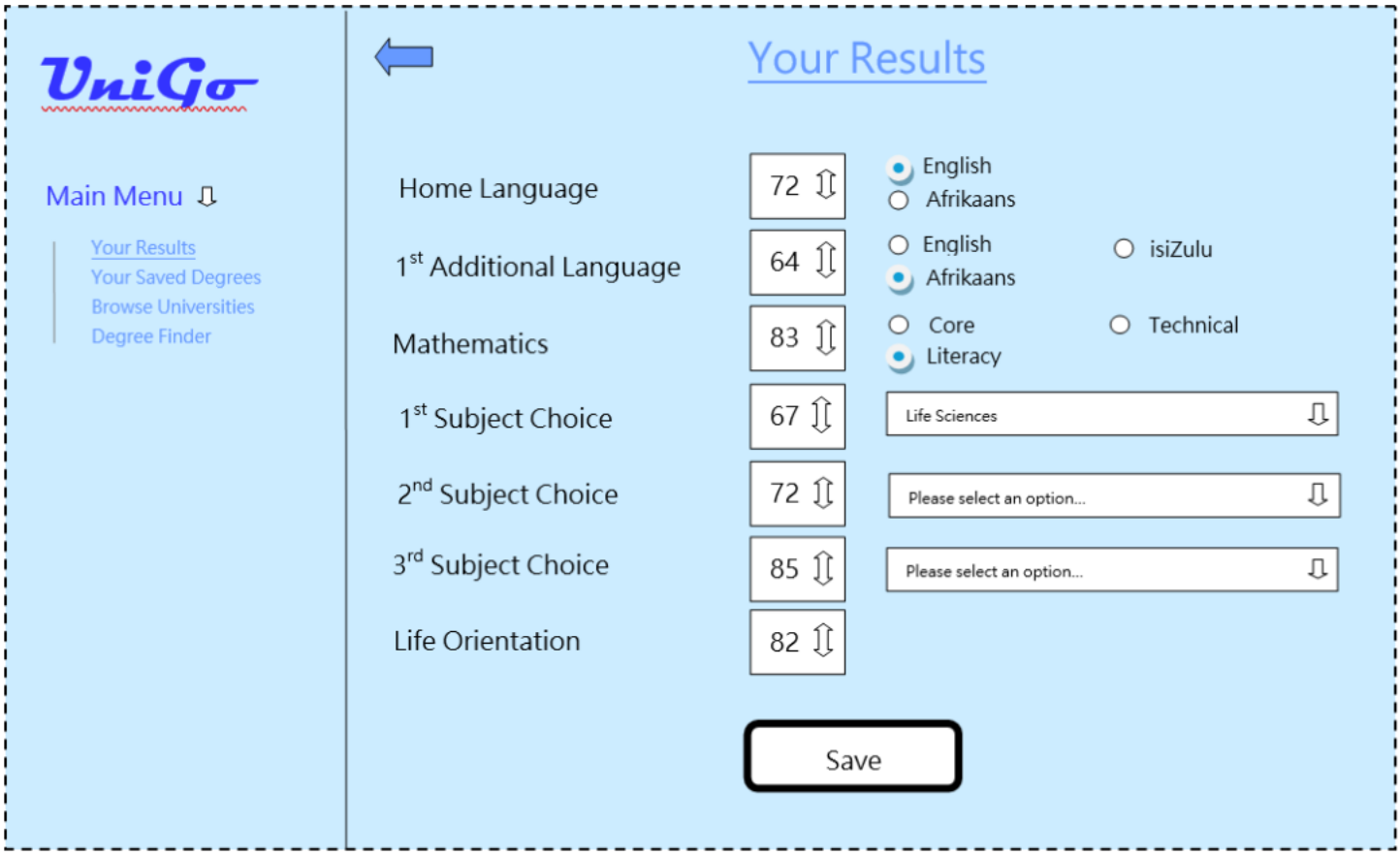
- Close the current screen

- Fetch the list of universities
- Clear the search bar
- Populate the table
- Open the BrowseUniversities screen

### 1.1.4 - Clicking ‘Degree Finder’

- Close the current screen
- Check if the ‘Use Results’ filter is applied
- If yes, use an algorithm to compare the user’s results with each degree. Degrees that the user meets the requirements of will be added to the list. Those that the user does not will be discarded.
- Check if any other filters are applied
- Remove entries from the list accordingly
- Check the search bar
- Remove entries from the list accordingly
- Open the Degree Finder screen
- Populate the table

## 1.2 Your Results

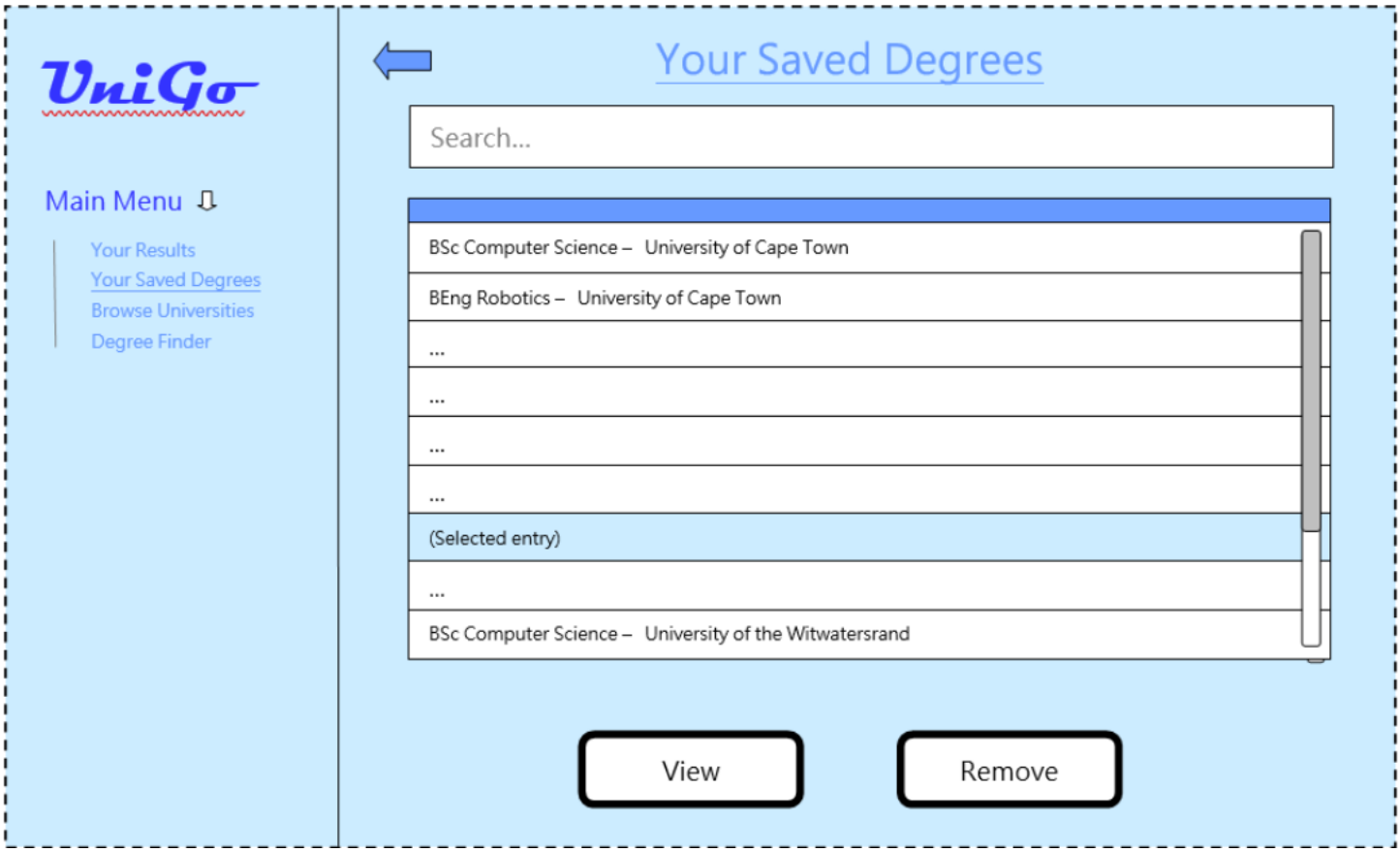



On this screen, the user will be able to input their marks and subject choices. Marks are out of 100, so a spinner will be used. For Home Language, FAL and Mathematics, there are fewer choices. This is why a combo box is being used here. For the subject choices, there are far more options. Therefore, a dropdown list has been provided.

### 1.2.1 - Clicking 'Save'

- The program will fetch all the values it needs from the UI
- Only the first 3 letters of every choice will be fetched, and it will be lower case.
- Using these values, a String will be created.
- This string is in the same format as a Requirements object
- The program will overwrite the userMarks text file with the String.

## 1.3 Your Saved Degrees




Here the user will be able to view a list of all the degrees that they have saved. Typing in the search bar will remove entries based on what has been typed. The user can select an entry, and then either go to the degree's information screen or remove the degree from the saves.

### 1.3.1 - Typing in the search bar:

- Fetch the text from the search bar and the list.
- Remove entries from the list if it is not similar to the received text.
- Displays the new list.

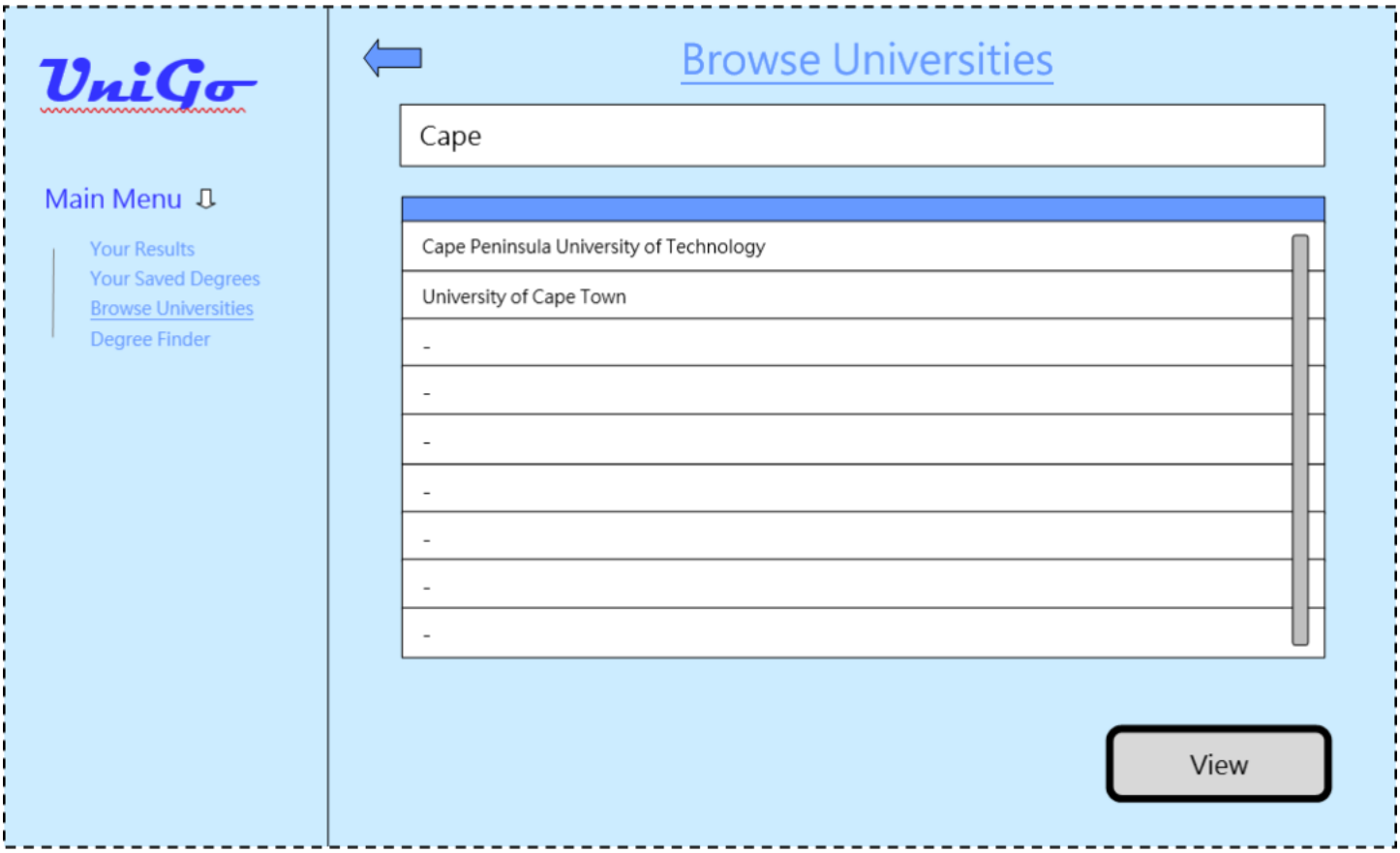
### 1.3.2 - Clicking 'View'

- Checks which entry is selected
- Finds the ID of the degree linked to the entry
- Closes the current screen
- Opens the Degree screen, and populates it using the Degree's ID.

### 1.3.3 - Clicking 'Remove'

- Checks which entry is selected
- Finds the ID of the degree linked to the entry
- Opens the SavedDegrees file
- Create a new string
- Checks each ID within the files
- If the ID in the file is not the same as the received ID, add the ID to the string
- If the ID in the file is the same as the received ID, discard the ID
- Overwrites the file with the new string
- Repopulates the table

## 1.4 Browse Universities




On this screen, the user will be given a list of all the universities within the program. They can select a university from the list, and then click a button that will take them to that university's screen. A search bar will be provided. When the user types in the search bar, the table will then only show universities whose names are like what has been searched.

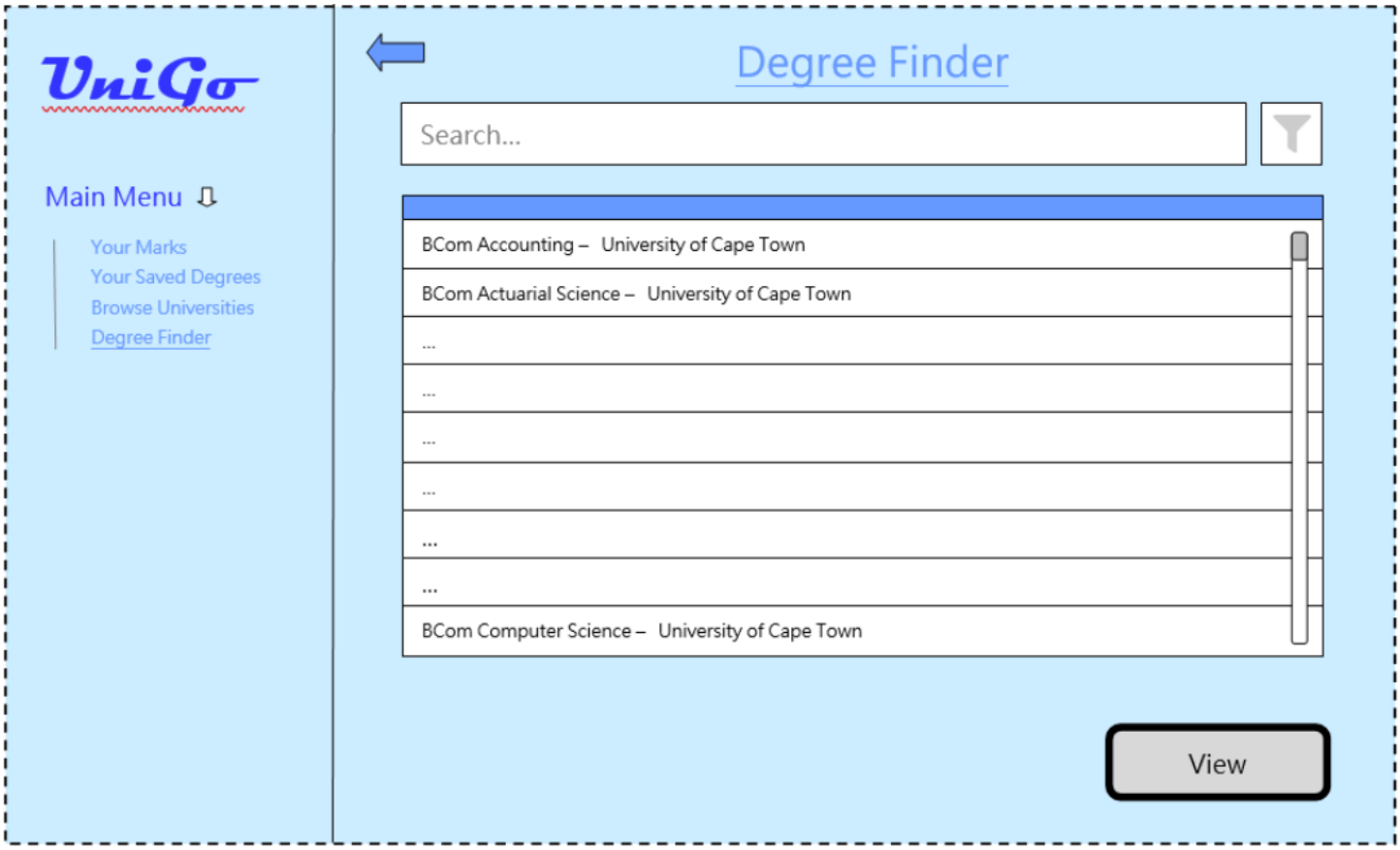
### 1.4.1 - Typing in the search bar:

- Fetch the text from the search bar and the list.
- Remove entries from the list if it is not similar to the received text.
- Displays the new list.

### 1.4.2 - Clicking on 'view'

- Fetches the ID of the university
- Closes the current screen
- Opens the University screen, and populates it using the University's ID.

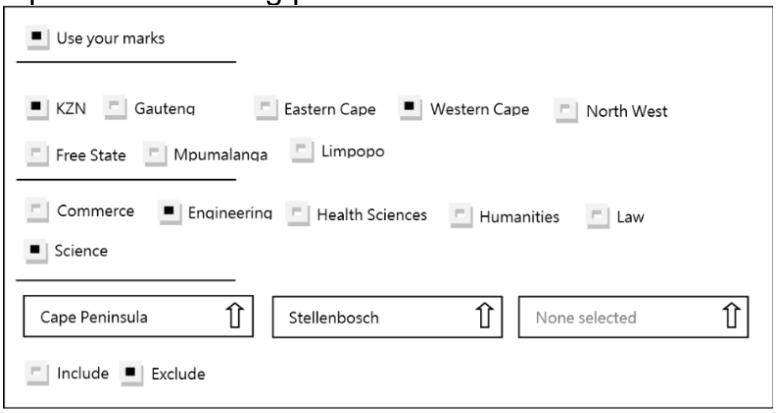
## 1.5 Degree Finder




Here, the user will be able to go to the Degree screen of any of the degrees shown in the list. The entries in this list are predetermined (go to 2.1.4). If the filters are changed, or the user types in the search bar, the list will change to reflect the user's choices.

### 1.5.1 - The filter button is clicked

- Changes the transparency of the filter icon to 100%
- Opens the following panel:



- Pressing the button again will…
    - If there is at least one filter applied (that's not 'Use your marks'), change the transparency of the filter icon to 100%, otherwise set to 50%
    - Apply filters (Check the pseudo code in 2.1.4)
    - Close the panel

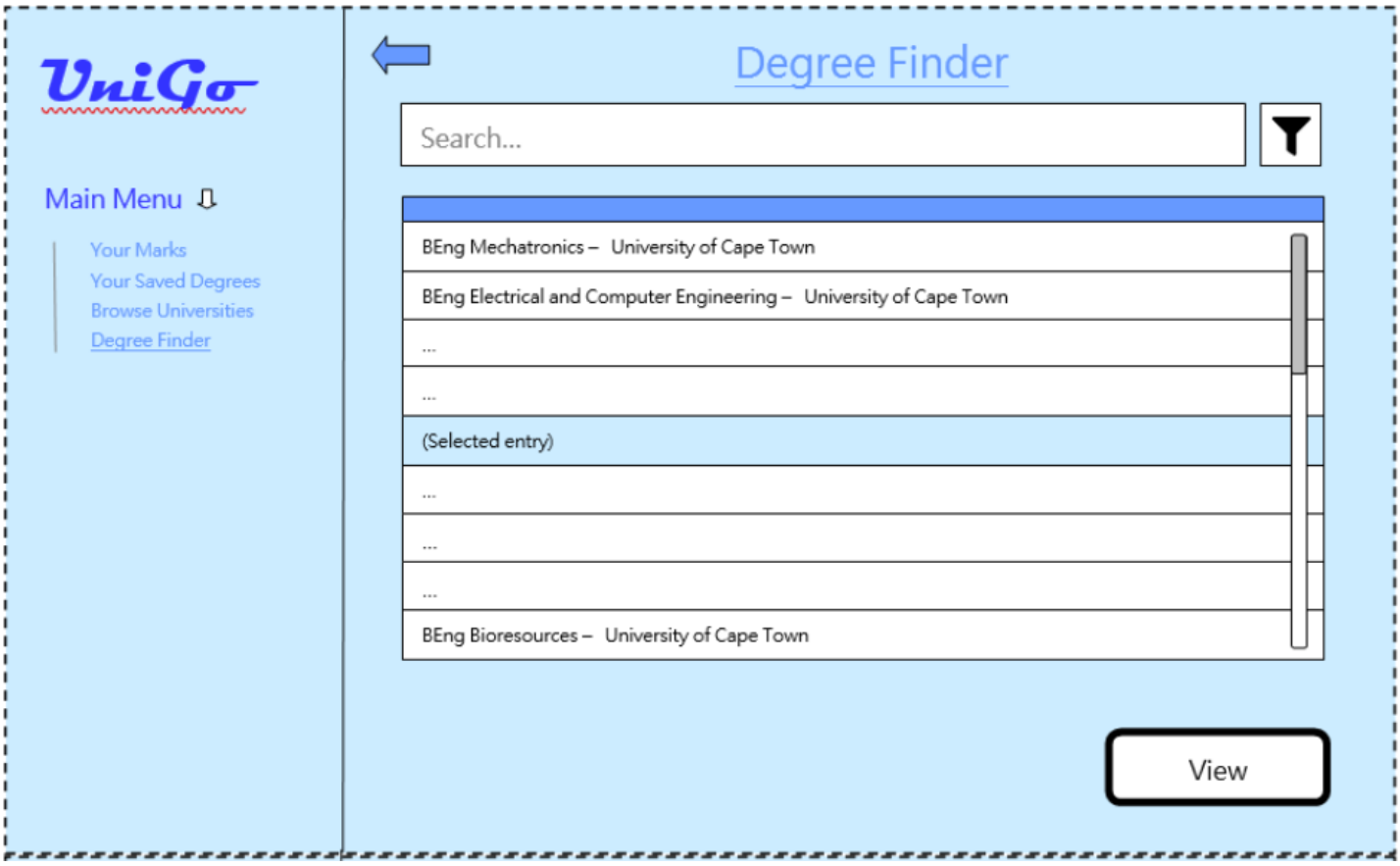
### 1.5.2 - The user types in the search bar

- Fetch the text from the search bar and the list.
- Remove entries from the list if it is not similar to the received text.
- Displays the new list.

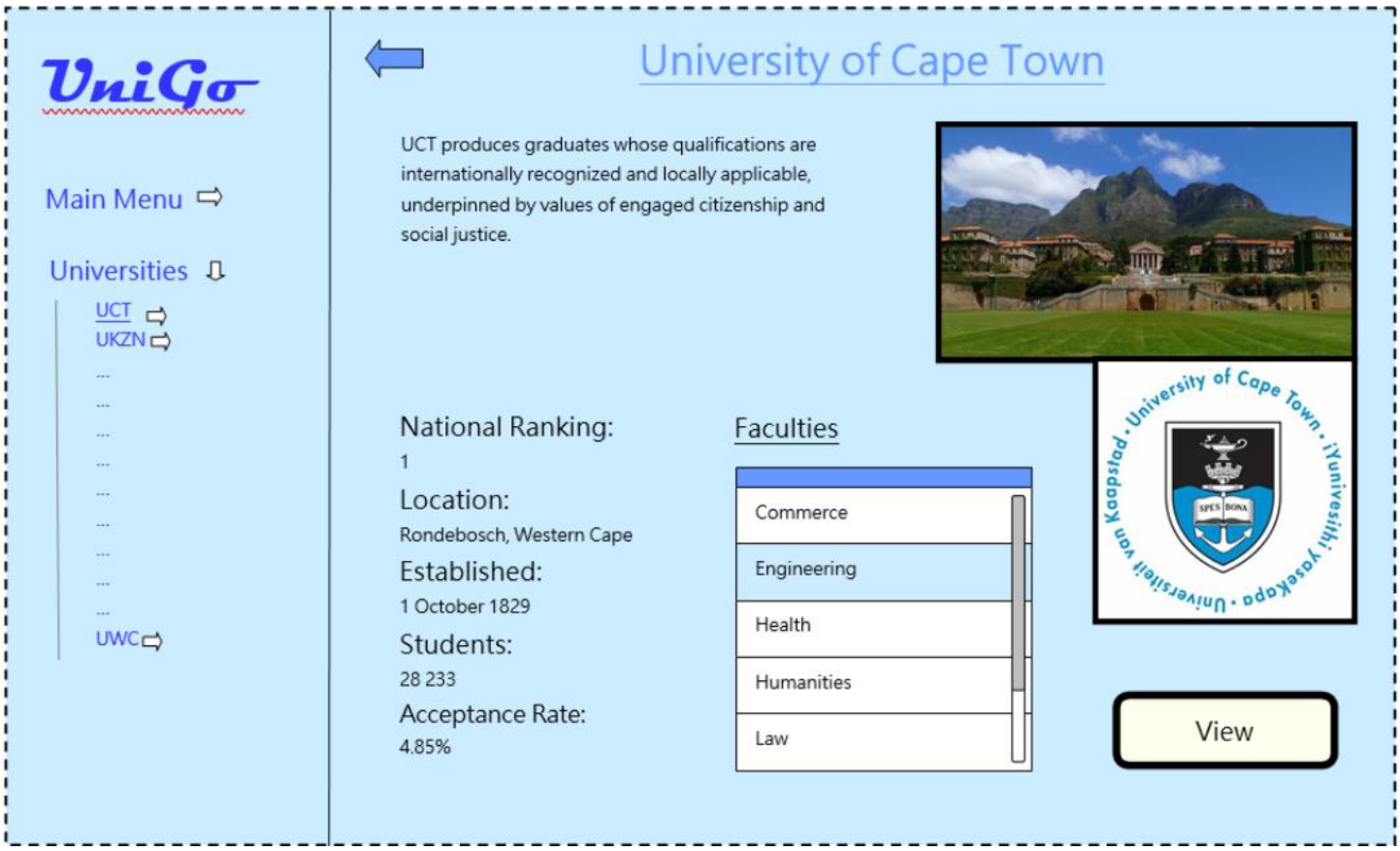
### 1.5.3 - Clicking on 'view'

- Fetches the ID of the degree
- Closes this screen and opens the Degree screen, populated using the Degree's ID.

Below is what the screen would look like with filters applied and an entry selected, search bar empty:



## 1.6 University

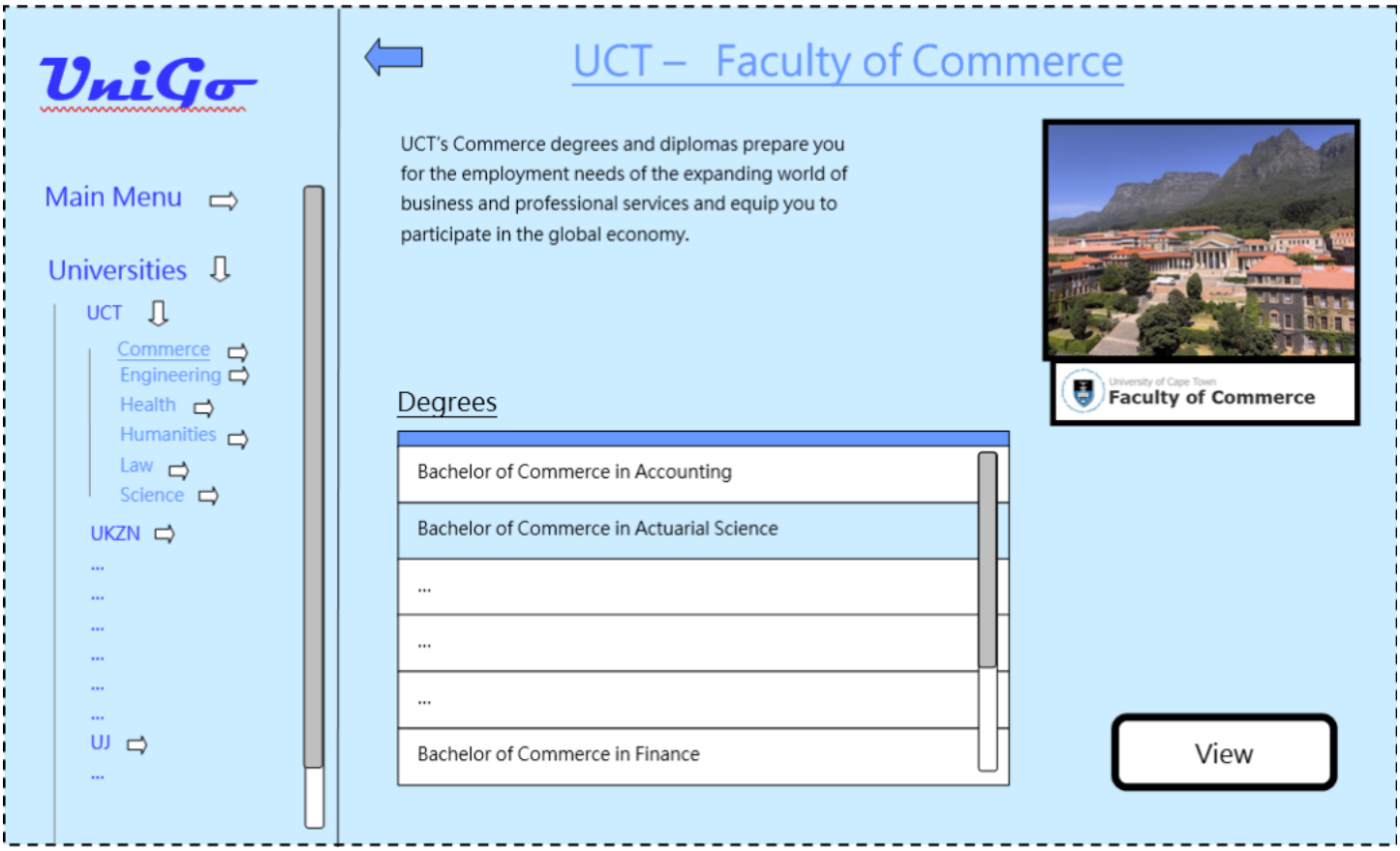



Here, the user will be able to view information on any university within the program's database. The user will also be able to view a table with all the university's faculties. Selecting an entry will allow the user to go to the selected faculty's screen.

### 1.6.1 - Clicking 'View'

- Fetches the index of the selected Faculty
- Fetches the list of faculties
- Uses the index to find the faculty's ID
- Closes this screen and opens the faculty screen using the ID.

## 1.7 Faculty

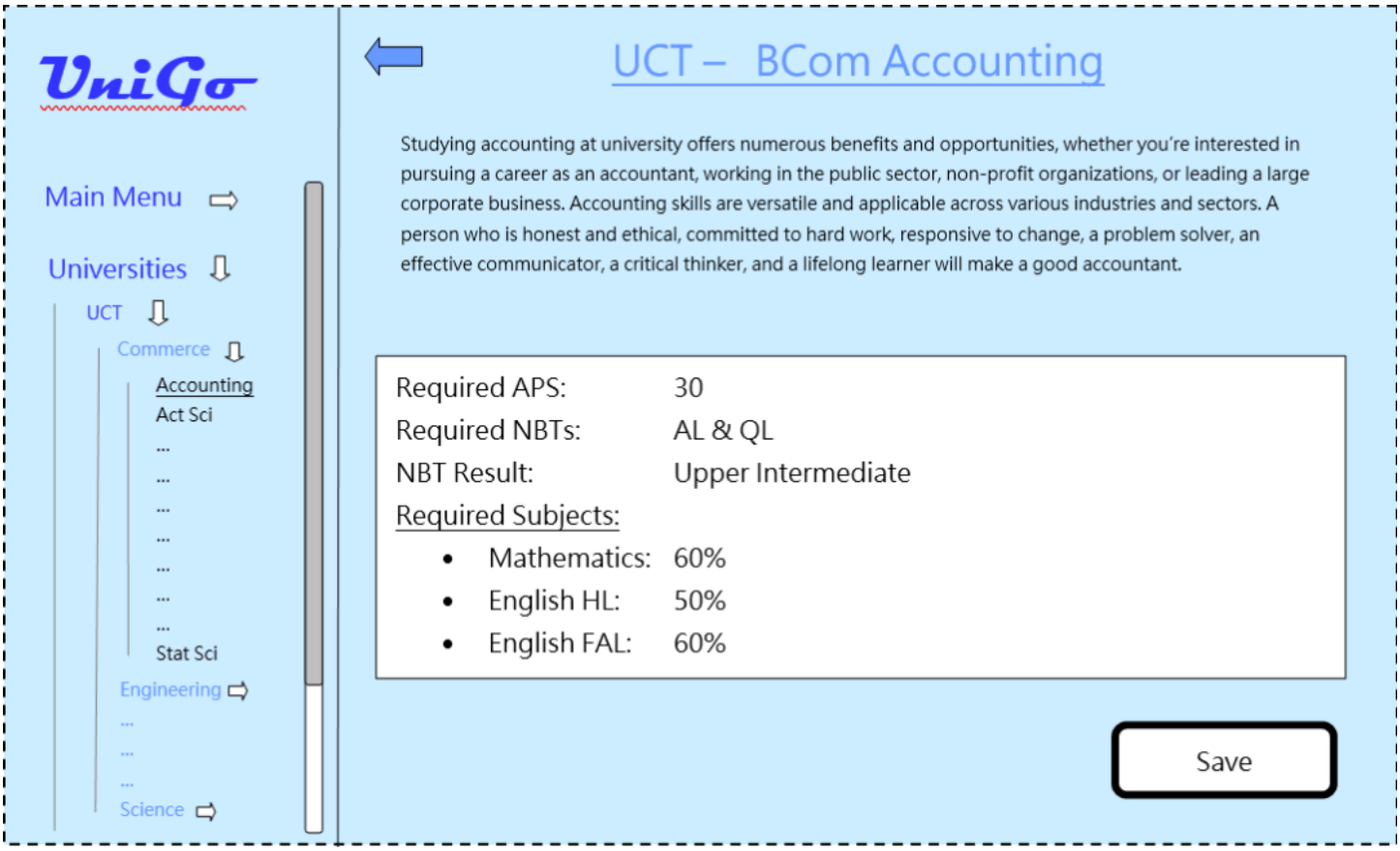



This screen shows the user all information relating to a specific faculty, including the faculty's offered degrees. Clicking on a degree will give the user the option to go to that degree's information screen.

### 1.7.1 - Clicking 'View'

- Fetches the index of the selected Faculty
- Fetches the list of faculties
- Uses the index to find the faculty's ID
- Closes this screen and opens the faculty screen using the ID.

## 1.8 Degree



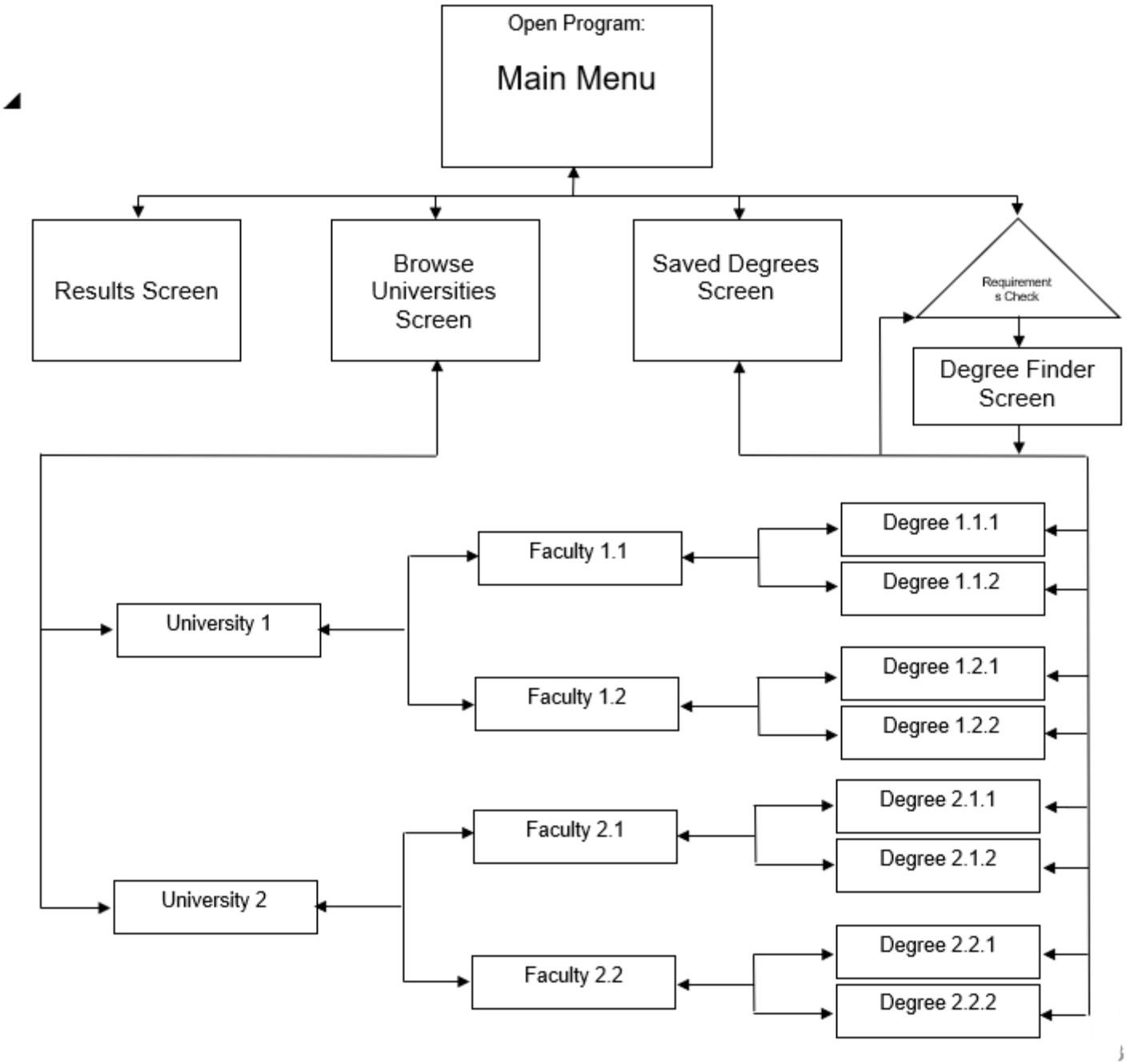

This screen shows the user a degree's requirements. It also allows them to save this degree so that they can quickly come back to it later.

### 1.8.1 - When 'Save' is clicked

- Get the degree's ID
- Append the ID to the SavedDegrees text file
- Change the color and text of the button

# 2. Program Flow

Upon opening the program, the user will be brought to the Main Menu. University, Faculty, Degree and Requirement objects will be created using all the information from the database. All objects will belong to their respective Manager classes and will be accessible by the entire program. Objects will be linked to one another, meaning that faculties can belong to universities, and degrees will belong to faculties. The Database Driver will get its own class and will be used throughout the program as well. There are 4 special screens, namely the Results, Browse Universities, Saved Degrees and Degree Finder screens. Each has their own function. Every University, Faculty and Degree will have their own screen and will be used to display information regarding the item.



## 2.1 Main Menu → Results Screen

```
setPreviousScreen(menuScreen)
close menuScreen

new UserMarksManager UMM

spn_hl.setValue(UMM.getHlMark)

if (UMM.getHlChoice = eng)
>    cbb_hlEnglish.setSelected(true)
     cbb_hlAfrikaans.setSelected(false)
< else if (UMM.getHlChoice = afr)
>    cbb_hlEnglish.setSelected(false)
     cbb_hlAfrikaans.setSelected(true)
<

(repeat blue text for every subject choice except LO)

spn_loMark.setValue(UMM.getLo)

open resultScreen
```

### 2.1.1 [Save]

```
int hl_mark = spn_hl.get
String hl_choice = cbb_hl.get
(Repeat the above 2 lines for every subject except LO)

int lo_mark = spn_lo.get

new userMarksManager uMM
uMM.save(hl_mark, hl_choice, …, lo_mark)
```

## 2.2 Main Menu → Browse Universities Screen

```
setPreviousScreen(menuScreen)
close menuScreen
UniversityManager um

string[] list = um.getUniversities("SELECT Name FROM tblUniversity;")
for (i = 0; i < list.length; i++)
>    table[i].setText(list[i]) <

clear searchBar
open browseUniScreen
```

### 2.2.1 (User types in search bar)

```
new UniversityManager um
String query = "WHERE Name LIKE '%'" +  searchBar.getText + "%' "
int[] list = um.getUniversities(query)

for (i = 0; i < list.length; i++)
>    table.setIndexAt[i](um.getUniversityName(list[i])) <
```

### 2.2.2 [View]

```
int index = table.getSelectedIndex
```

```
new UniversityManager um
int ID = um.getID(list[index])
close current screen
open University(ID) Screen
```

**Go to 2.5.1**

## 2.3 Main Menu → Saved Degrees Secreen

```
setPreviousScreen(menuScreen)
close menuScreen

UserManager um
String[] list = um.getSavedDegrees

for (i = 0; i < list.length; i++)
>    table[i].setText = list[i] <

open savedDegreesScreen
clear searchBar
```

### 2.3.1 (User types in search bar)

```
new DegreeManager dm
String query = "WHERE Name LIKE '%'" +  searchBar.getText + "%' "
int[] list = dm.getDegrees(query)

for (i = 0; i < list.length; i++)
>    table.setIndexAt[i](dm.getDegreeName(list[i])) <
```

### 2.3.2 [View]

```
int index = table.getSelectedIndex
new DegreeManager dm
int ID = dm.getID(list[index])
close current screen
open Degree(ID) Screen
```

**Go to 2.5.3**

### 2.3.3 [Remove]

```
int index = table.getSelectedIndex
new userSavedDegreesManager uSDM
uSDM.remove(index)

new DegreeManager dm

for (i = 0; i < size; i++)
>
     new Degree d = dm.getDegree(list[i])
     table.setIndexAt[i](d.getName())
<
```

## 2.4 Main Menu → Degree Finder Screen

```
setPreviousScreen(menuScreen)
close menuScreen

String query = "SELECT Name FROM tblDegree "
```

```
int size
new DegreeManager dm
boolean needsWhere = true
new Requirement r

if (clb_useMarks.isSelected)
>
     query += r.toQuery
     needsWhere = false
<

if (clb_locationKZN.isSelected)
>
     query += r.addToQuery(needsWhere, "Location = KZN ")
     needsWhere = false
< repeat if-statement for every location

if (clb_facultyCommerce.isSelected)
>
     query += r.addToQuery(needsWhere, "Faculty = Commerce ")
     needsWhere = false
<(repeat if-statement for every faculty)

string uni1 = ddl_uniChoice1.getText
string uni2 = ddl_uniChoice2.getText
string uni3 = ddl_uniChoice3.getText

if (cbb_uniInclude.isSelected)
>
     if (!uni1.isBlank)
     >
          query += r.addToQuery(needsWhere, "University = " + uni1)
          needsWhere = false
     < (Repeat this if-statement for each choice)

<else if (cbb_uniExclude.isSelected)
     >
          if (!uni1.isBlank)
          >
               query += r.addToQuery(needsWhere, "University IS NOT
                    = " + uni1)
               needsWhere = false
          < (Repeat this if-statement for each choice)
     <
<

string search = searchBar.get

if (!search.isBlank)
     >
          query += r.addToQuery(needsWhere, "Degree LIKE '%" +
               search + "%' ")
          needsWhere = false
     <

String[] list = getDegrees(query)

open degreeFinderScreen
```

```
for (i = 0; i < list.length; i++)
>    table[i].setText = list[i] <
```

### 2.4.1 [Filter]

**See 2.4**

### 2.4.2 (User types in search bar)

**See 2.4**

### 2.4.3 [View]

```
int index = table.getSelectedIndex
new DegreeManager dm
int ID = dm.getID(list[index])
close current screen
open Degree(ID) Screen
```

**Go to 2.5.3**

## 2.5 Dedicated Screens

### 2.5.1 University

```
UniversityManager um
University u = um.getUni(ID)

lblTitle.setText(u.getName)
lblDesc.setText(u.getDesc)
lblRank.setText(u.getRank)
lblLocation(u.getLocation)
lblEstb(u.getEstb_Date)
lblStudents(u.getStudents)
lblAccRate(u.getAcc_Rate)


Faculty[] f = u.getFaculties
For(i = 0; i < f.length; i++)
>    table[i] = f[i].getName <
```

#### *2.5.1.1 [View]*

```
int index = table.getSelectedIndex
fetch f[]
int ID = f[index]

Close current screen
Open Faculty(ID) screen
```

**Go to 2.5.2**

### 2.5.2 Faculty

```
FacultyManager fm
Faculty f = fm.getFaculty(ID)

lblTitle.setText(f.getName)
lblDesc.setText(f.getDesc)

Degree[] d = f.getDegrees
for(i = 0; i < d.length; i++)
>    table[i] = d[i].getName <
```

#### *2.5.2.1 [View]*

```
int index = table.getSelectedIndex
fetch d[]
int ID = d[index]

Close current screen
Open Degree(ID) screen
```

**Go to 2.5.3**

### 2.5.3 Degree

```
DegreeManager dm
Degree d = dm.getDegree(ID)

lblTitle.setText(d.getName)
lblDesc.setText(d.getDesc)

RequirementManager rm
Requirement r = rm.getReq(d)
lblReq.setText(r.toString)
```

#### *2.5.3.1 [Save]*

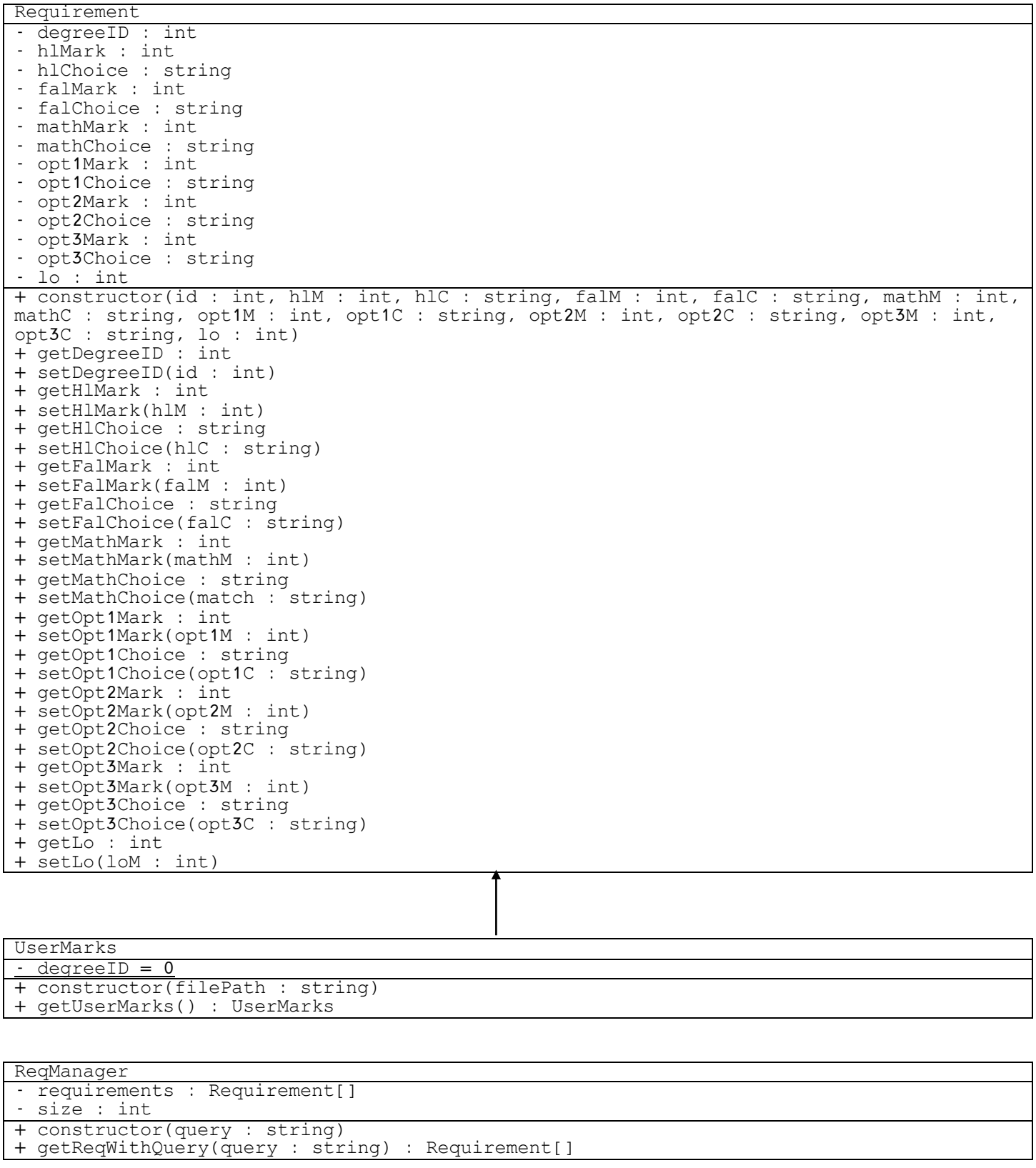
```
int ID = d.getID

new userSavedDegreesManager uSDM
uSDM.save(ID)

saveButton.setText("Saved")
saveButton.setColor("GRAY")
```

# 3. Class Diagrams and OOP Principles

## 3.1 Requirement & UserMarks



## 3.2 SavedDegrees

| SavedDegrees |
| --- |
| - degrees : int[]<br>- size : int |
| + constructor(filePath : string)<br>+ getSavedDegrees() : int[] |

## 3.3 University

| University |
| --- |
| - name : string<br>- desc : string<br>- location : string<br>- rank : int<br>- estb : int<br>- students : int<br>- accRate : float |
| + constructor(name : string, desc : string, location : string, rank : int, estb : int, students : int, accRate : float)<br>+ getName() : string<br>+ setName(name : string)<br>+ getDesc() : string<br>+ setDesc(desc : string)<br>+ getLocation() : string<br>+ setLocation(location : string)<br>+ getRank() : int<br>+ setRank(rank : int)<br>+ getEstb() : int<br>+ setEstb(estb : int)<br>+ getStudents() : int<br>+ setStudents(students : int)<br>+ getAccRate() : float<br>+ setAccRate(accRate : float) |

| UniManager |
| --- |
| - universities : University[]<br>- size : int |
| + constructor(query : string)<br>+ getUniWithID(id : int) : University<br>+ getUniWithName(name : string) : University<br>+ getUniWithLocation(location : string) : University[] |

## 3.4 Faculty

| Faculty |
| --- |
| - name : string<br>- desc : string<br>- uni : University |
| + constructor(name : string, desc : string, uni : University)<br>+ getName() : string<br>+ setName(name : string)<br>+ getDesc() : string<br>+ setDesc(desc : string)<br>+ getUni() : University<br>+ setUni(uni : University) |

| FacManager |
| --- |
| - faculties : Faculty[]<br>- size : int |
| + constructor(query : string)<br>+ getFacWithName(name : string) : Faculty |

| + getFaculties() : Faculty[] |
|---|

## 3.5 Degree

| Degree |
|---|
| - degreeID : int<br>- name : string<br>- uni : University<br>- fac : Faculty<br>- desc : string |
| + constructor(id : int, name : string, uni : University, fac : Faculty, desc : string)<br>+ getID() : int<br>+ setID(id : int)<br>+ getName() : string<br>+ setName(name : string)<br>+ getUni() : University<br>+ setUni(uni : University)<br>+ getFac() : Faculty<br>+ setFac(fac : Faculty)<br>+ getDesc() : string<br>+ setDesc(desc : string) |

| DegManager |
|---|
| - degrees : Degree[]<br>- size : int |
| + constructor(query : string)<br>+ getDegrWithName(name : string) : Degree[]<br>+ getDegWithID(id : int) : Degree |

## 3.6 Database Driver

| dbDriver |
|---|
| - DRIVER = "net.ucanaccess.jdbc.UcanaccessDriver"<br>- URL = "jdbc:ucanaccess://UniGo.accdb"<br>+ connection : Connection<br>- statement : PreparedStatement<br>- resultSet : ResultSet |
| + constructor()<br>+ query(query : string) : ResultSet<br>+ update(query : string)<br>- addToQuery(addWhere : boolean, toAdd : string) : string |

# 4. Secondary Storage Design

This program uses both databases and text files to store data. Below you will see the fields of each table along with the data type for each field. Sample data has also been provided.

## 4.1 University_Table

| ID | UniversityName | Description | Location | Rank | Established | Students | AcceptanceRate |
|---|---|---|---|---|---|---|---|
| 1 | University of Cape Town | UCT is a community of exceptionally | WC | 1 | 1829 | 28233 | 4.85 |
| 2 | University of the Witwatersrand | Internationally distinguished for rese | G | 2 | 1922 | 40259 | 34 |
| 3 | Stellenbosch University | We combine world-class knowledge | WC | 3 | 1918 | 35093 | 52 |
| 4 | University of Pretoria | The University of Pretoria (UP) is or | G | 4 | 1908 | 56409 | 57 |
| 5 | University of KwaZulu-Natal | Boasting multiple campuses, UKZN | KZN | 5 | 2004 | 46539 | 63 |
| 6 | University of Johannesburg | Recognised as the country's seconc | G | 6 | 2005 | 50786 | 60 |
| 7 | University of South Africa | The University of South Africa (Unis | G | 7 | 1873 | 420000 | 64 |
| 8 | North-West University | The NWU is committed to functionin | NW | 8 | 2004 | 56000 | 83 |

- ID : integer
- UniversityName : string
- Description : string
- Location : string
- Rank : integer
- Established : integer
- Students : integer
- AcceptanceRate : float

## 4.2 Faculty_Table

| ID | FacultyName | Description | UniversityName |
|---|---|---|---|
| 1 | Commerce | Commerce degree programmes prep | University of Cape Town |
| 2 | Engineering | The Faculty of Engineering & the Bu | University of Cape Town |
| 3 | Health Sciences | The Faculty of Health Sciences is cc | University of Cape Town |
| 4 | Humanities | With close to 7 000 students, the fac | University of Cape Town |
| 5 | Law | The Faculty of Law might be the sma | University of Cape Town |
| 6 | Science | The Faculty of Science has a long a | University of Cape Town |
| 7 | Agriculture | The Faculty of AgriSciences at Stelle | Stellenbosch University |
| 8 | Humanities | The Faculty of Arts and Social Scien | Stellenbosch University |
| 9 | Commerce | From our humble beginnings in 1925 | Stellenbosch University |
| 10 | Education | The Faculty strives to play a leading | Stellenbosch University |

- ID : integer
- FacultyName : string
- Description : string
- UniversityName : string

## 4.3 Degree_Table

| ID | DegreeName | UniversityName | FacultyName | Description |
|---|---|---|---|---|
| 1 | Accounting | University of Cape Town | Commerce | Studying accounting at university offers numerous benefits |
| 2 | Actuarial Science | University of Cape Town | Commerce | Actuaries use statistical techniques to solve financial and |
| 3 | Computer Science | University of Cape Town | Commerce | Computer Science is about the creation of computer software |
| 4 | Economics | University of Cape Town | Commerce | All Economics options give an excellent grounding in the area |
| 5 | Information Systems | University of Cape Town | Commerce | The Information Systems discipline investigates how |
| 6 | Finance | University of Cape Town | Commerce | In Finance, students develop the skills needed to source, |
| 7 | Management Studies | University of Cape Town | Commerce | The Bcom specialising in Management Studies is the |
| 8 | Marketing Studies | University of Cape Town | Commerce | Marketing is concerned with creating the revenue streams |
| 9 | Organisational Psychology | University of Cape Town | Commerce | Organisational Psychology, also known as Industrial |
| 10 | Statistical Sciences | University of Cape Town | Commerce | Statistics and data science plays a crucial role in |

- ID : integer
- DegreeName : string
- UniversityName : string
- FacultyName : string
- Description : string

## 4.4 Requirement_Table

| ID | DegreeID | HL | FAL | MATH | OPT1 | OPT2 | OPT3 | APS |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | eng50 | any00 | cor60 | any00 | any00 | any00 | 31 |
| 2 | 1 | any00 | eng60 | cor60 | any00 | any00 | any00 | 31 |
| 3 | 2 | eng60 | any00 | cor80 | any00 | any00 | any00 | 35 |
| 4 | 2 | any00 | eng80 | cor80 | any00 | any00 | any00 | 35 |
| 5 | 3 | eng50 | any00 | cor70 | any00 | any00 | any00 | 31 |
| 6 | 3 | any00 | eng60 | cor70 | any00 | any00 | any00 | 31 |
| 7 | 4 | eng50 | any00 | cor60 | any00 | any00 | any00 | 31 |
| 8 | 4 | any00 | eng60 | cor60 | any00 | any00 | any00 | 31 |
| 9 | 5 | eng50 | any00 | cor60 | any00 | any00 | any00 | 31 |
| 10 | 5 | any00 | eng60 | cor60 | any00 | any00 | any00 | 31 |

- ID : int
- DegreeID : int
- HL : string
- FAL : string
- MATH : string
- OPT1 : string
- OPT2 : string
- OPT3 : string
- APS : int

## 4.5 UserResults.txt

```
eng72#afr66#cor80#inf73#phy51#bus73#34
```

- HL : string
- FAL : string
- MATH : string

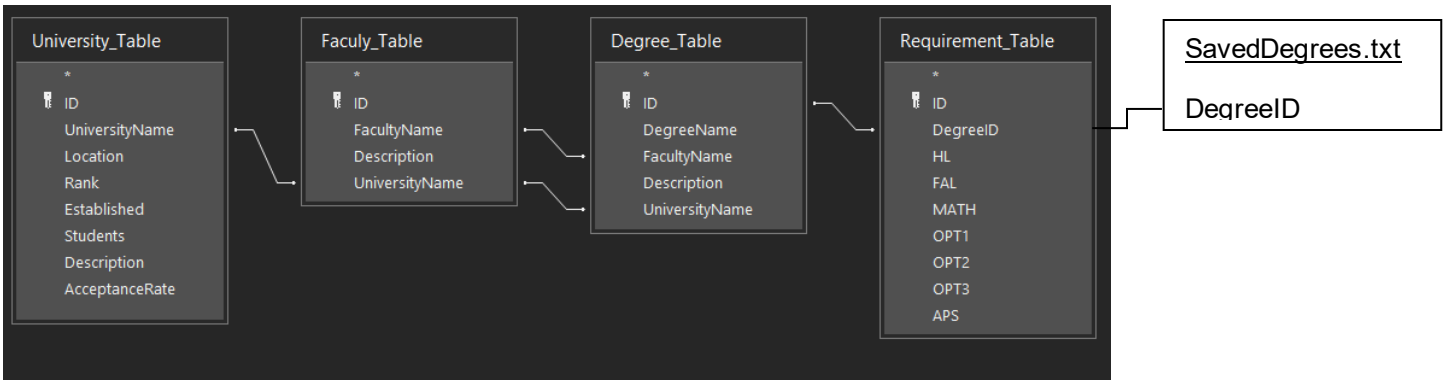
- OPT1 : string
- OPT2 : string
- OPT3 : string
- APS : int

## 4.6 SavedDegrees.txt

```
3
46
16
41
48
115
```

- DegreeID : int

## 4.7 Relationships




1. University.UniversityName – Faculty.UniversityName
2. Faculty.UniversityName – Degree.UniversityName
3. Faculty.FacultyName – Degree.FacultyName
4. Degree.ID – Requirement.DegreeID
5. Requirement.DegreeID – SavedDegrees.txt(DegreeID)

# 5. Explanation of Secondary Storage Design

## 5.1 Databases

All information that relates to the universities (that have nothing to do with the user) will be stored within a database. There are a few reasons for this choice. Firstly, the amount of data is extensive, and therefore using a database is the most practical way of storing it. Additionally, the program will sometimes need to find and return specific data based on user input. To do this, it will need to make use of a querying system. Databases are most suited for querying systems like SQL. Lastly, the data will need to be changed often and in real time as things change in real life. A database allows the administrator to easily make changes that will be made effective throughout every instance of the program. For the reasons stated above, a database has been used to store non-user related data.

The following will use a database:

- University_Table
- Faculty_Table
- Degree_Table
- Requirement_Table

## 5.2 Text files

The program will need to store some information that is unique to each user, such as the user's academic results and the degrees that they have bookmarked. This data is simple, and therefore a simple text file can be used to store the data. This data will remain on the machine and can be manipulated and utilized directly by the program. There is no need for special procedures such as SQL to process the data, since what is in the file is already how the program needs it to be. The data in the text files is small, meaning that it can be stored on the user's computer without issue.

The following will use a text file:

- UserResults.txt
- SavedDegrees.txt

# 6. Explanation of Primary Data Structure related to Secondary Storage

**All objects and arrays will remain in memory so long as the program is open.**

- When the program is opened, a University object is instantiated for every record in the University table, using the data in that table.
- An array of University objects is created. This array will contain every single University object that has been instantiated.

- When the program is opened, a Faculty object is instantiated for every record in the Faculty table, using the data in that table.
- An array of Faculty objects is created. This array will contain every single Faculty object that has been instantiated.

- When the program is opened, a Degree object is instantiated for every record in the Degree table, using the data in that table.
- An array of Degree objects is created. This array will contain every single Degree object that has been instantiated.

- When the program is opened, a Requirement object is instantiated for every record in the Requirement table, using the data in that table.
- An array of Requirement objects is created. This array will contain every single Requirement object that has been instantiated.

- Using the data from the UserResults text file, a UserMarks object will be instantiated. The object inherits its properties from the Requirement object

- An array of integers will be instantiated using the list of integers provided by the SavedDegrees text file.

- The dbDriver object is instantiated when the program is opened, providing the program with continious connection to the database.