A black and white logo

Description automatically generated

University Database Management System

BY: Mohamed Saeed

Data Management-Alex

# Brief About the Case Study

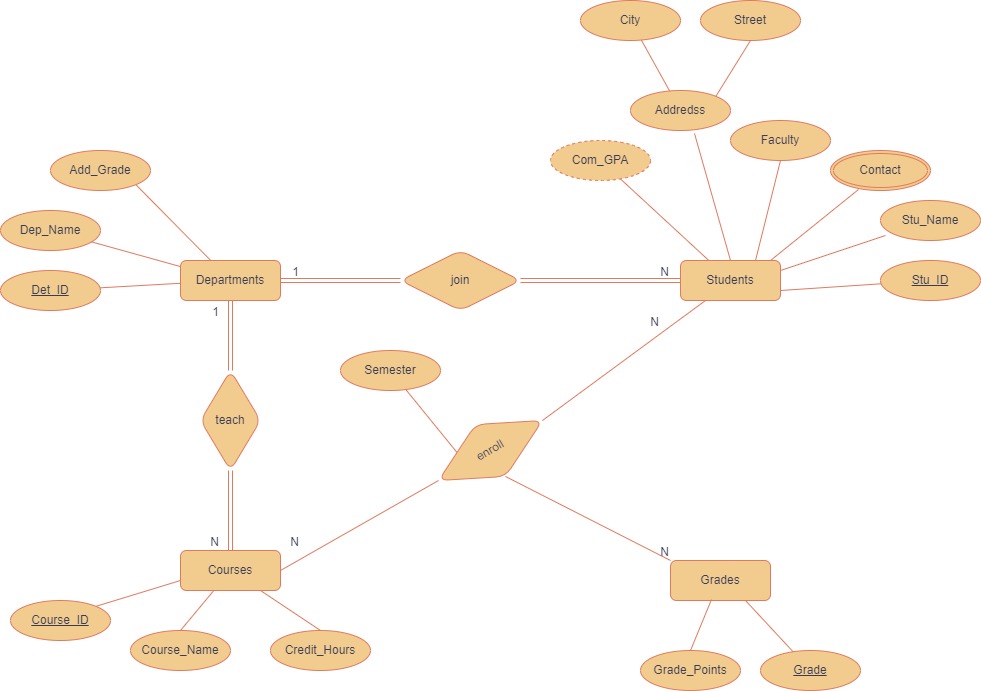
The objective of this project is to design and implement a comprehensive data management system for a university using SQL, PLSQL, Advanced PLSQL, Red Hat, Bash scripting, Java SE, and OOP principles. The project encompasses various aspects, including database design, SQL and PLSQL implementation, automation scripts, Java application development, and integration.

**Milestones:**

1. **Database Design**
2. **SQL Implementation**
3. **PLSQL Implementation**
4. **Automation Scripts**
5. **Java Application Development**
6. **Integration and Reporting**

# Database Design

**ERD:**



* Students’ Information stored in table students.
* Departments Information stored in table departments.
* Courses Information stored in table courses.
* The Grades and their points stored in grades table.
* Students must be assigned in only one department and one department must have one or more students.
* The course must be taught in only one department and department must have one or more courses.
* Many students may enroll in many courses and many courses may be enrolled by many students.

**Mapping:**

1. **Students Table:** 
   * Stu\_id (primary key)
   * Stu\_name
   * Contact (multivalued)
   * Address (city, street) (composite attribute)
   * Faculty
   * Com\_GPA
   * Dept\_ID (Foreign Key referencing Departments Table)
2. **Courses Table:** 
   * Course\_ID (Primary Key)
   * Course\_name
   * Credit\_Hours
   * Dept\_ID (foreign key referencing Departments Table)
3. **Departments Table:**
   * Dept\_ID (Primary Key))
   * Dept\_name
   * Add\_grade
4. **Grades Table:**
   * Grade (Primary Key)
   * Grade\_points
5. **Enrollments Table:**
   * Stu\_ID (primary key with course\_id)
   * Course\_ID (primary key with stu\_id)
   * Grade
   * Semester

**Normalization:**

1. **Students Table:** 
   1. Stu\_id (primary key)
   2. Stu\_name
   3. City
   4. street
   5. Faculty
   6. Com\_GPA
   7. Dept\_ID (Foreign Key referencing Departments Table)
2. **Courses Table:** 
   1. Course\_ID (Primary Key)
   2. Course\_name
   3. Credit\_Hours
   4. Dept\_ID (foreign key referencing Departments Table)
3. **Departments Table:**
   1. Dept\_ID (Primary Key))
   2. Dept\_name
   3. Add\_grade
4. **Grades Table:**
   1. Grade (Primary Key)
   2. Grade\_points
5. **Enrollments Table:**
   1. Stu\_ID (primary key with course\_id)
   2. Course\_ID (primary key with stu\_id)
   3. Grade
   4. Semester
6. **Student\_Contact:**
   1. Stu\_ID ((primary key with contact & foreign key referencing stu\_id in the students table).
   2. Contact (primary key with stu\_id).

# SQL Implementation

1. Students (Stu\_ID number primary key, Stu\_name varchar2(100) not null, City VARCHAR2(50), Street VARCHAR2(50), Faculty VARCHAR2(50), Dept\_ID number references Departments (Dept\_ID))

1. Departments (Dept\_ID number primary key, Dept\_Name varchar2(50) unique not null, Add\_Grade VARCHAR2(10)).

1. Courses (Course\_ID number primary key, Course\_Name varchar2(100) unique not null, Credit\_Hours number not null).

1. Enrollments (Stu\_ID number references Students (Stu\_ID), Course\_ID number references Courses (Course\_ID), semester varchar2(20), Grade varchar2(3)).
2. Grades (Grade VARCHAR2(2) PRIMARY KEY,

Grade\_Points NUMBER).

1. Student\_Contact (Stu\_ID NUMBER, Contact VARCHAR2(20), PRIMARY KEY (Stu\_ID, Contact),

FOREIGN KEY (Stu\_ID) REFERENCES Students (Stu\_ID)).

# PLSQL Implementation

**Package (gpa\_update\_pkg): consists of**

CREATE OR REPLACE PACKAGE gpa\_update\_pkg IS

TYPE student\_id\_list IS TABLE OF Students.stu\_ID%TYPE INDEX BY

PLS\_INTEGER;

students\_to\_update student\_id\_list;

PROCEDURE add\_student\_to\_update(stu\_id IN Students.stu\_id%TYPE);

PROCEDURE process\_updated\_students;

FUNCTION calc\_com\_GPA(v\_stu\_id NUMBER) RETURN NUMBER;

END;

* **Procedures:**

1. **add\_student\_to\_update:** make a list of students that their cumulative GPA will be updated.

PROCEDURE add\_student\_to\_update (stu\_id IN Students.stu\_id%TYPE) IS

BEGIN

students\_to\_update(stu\_id) := stu\_id;

END add\_student\_to\_update;

1. **process\_updated\_students:** update students Cumulative GPA in students’ table.

PROCEDURE process\_updated\_students IS

BEGIN

FORALL i IN students\_to\_update.FIRST..students\_to\_update.LAST

UPDATE Students

SET com\_GPA = calc\_com\_GPA(students\_to\_update(i))

WHERE stu\_id = students\_to\_update(i);

students\_to\_update.DELETE;

END process\_updated\_students;

* **Function:**

1. **calc\_com\_GPA:** to calculate the cumulative GPA for the students**.**

FUNCTION calc\_com\_GPA(v\_stu\_id NUMBER) RETURN NUMBER IS

CURSOR stu\_cursor IS

SELECT g.grade\_points, e.course\_id

FROM grades g, enrollments e

WHERE g.grade = e.grade

AND stu\_id = v\_stu\_id;

v\_total\_hours NUMBER := 0.0;

v\_points NUMBER;

v\_hours NUMBER;

v\_cour\_weight NUMBER;

v\_total\_weight NUMBER := 0.0;

v\_record stu\_cursor%ROWTYPE; -- Add this line

BEGIN

FOR v\_record IN stu\_cursor LOOP

SELECT credit\_hours

INTO v\_hours

FROM courses

WHERE course\_id = v\_record.course\_id;

v\_points := v\_record.grade\_points;

v\_cour\_weight := v\_points \* v\_hours;

v\_total\_hours := v\_total\_hours + v\_hours;

v\_total\_weight := v\_total\_weight + v\_cour\_weight;

END LOOP;

RETURN ROUND(v\_total\_weight / v\_total\_hours, 2);

END calc\_com\_GPA;

**Standalone Function:**

* **calc\_avg\_GPA:** to calculate the average GPA for the students enrolled in a specific course.

CREATE OR REPLACE FUNCTION calc\_avg\_GPA(v\_cour\_id NUMBER)

RETURN NUMBER

IS

CURSOR cour\_cursor IS

SELECT grade FROM enrollments

WHERE course\_id = v\_cour\_id;

v\_avg\_gpa NUMBER := 0.0;

v\_points NUMBER;

v\_total\_points NUMBER := 0;

v\_total\_students NUMBER := 0;

BEGIN

FOR rec IN cour\_cursor LOOP

SELECT grade\_points

INTO v\_points

FROM grades

WHERE grade = rec.grade;

v\_total\_points := v\_total\_points + v\_points;

v\_total\_students := v\_total\_students + 1;

END LOOP;

IF v\_total\_students > 0 THEN

v\_avg\_gpa := v\_total\_points / v\_total\_students;

END IF;

RETURN v\_avg\_gpa;

END;

**Trigger:**

* **Compound Trigger:** to fire when any students grade inserted or updated in the enrollments table and update the value of the cumulative GPA in students table.

CREATE OR REPLACE TRIGGER update\_GPA

FOR INSERT OR UPDATE ON ENROLLMENTS

COMPOUND TRIGGER

AFTER STATEMENT IS

BEGIN

gpa\_update\_pkg.process\_updated\_students;

END AFTER STATEMENT;

BEFORE EACH ROW IS

BEGIN

IF INSERTING OR UPDATING THEN

gpa\_update\_pkg.add\_student\_to\_update(:NEW.stu\_id);

END IF;

END BEFORE EACH ROW;

END ;

# Automation Script

1. **Database Backup:** 
   * **Purpose:** make a copy of the database and a full backup in a safe place
   * **Process:**
     + **Database Connection:**

Specifies database connection details using variables like db\_owner, PASS, and DB.

* + - **Date Formatting:**

Uses the date command to generate a timestamp in the format "YYYYMMDD\_HHMMSS" and assigns it to DATE\_FORMAT.

* + - **Export File Naming:**

Constructs the export file name with the format

"backup\_YYYYMMDD\_HHMMSS.dmp" using the previously generated timestamp.

* + - **Database Export:**

Utilizes the expdp command to perform a full database export. Connects to the specified database using the provided credentials.

Specifies the data pump directory and the dump file to store the exported data.

* + - **Status Check:**

Checks the exit status of the expdp command using $?.

A computer screen with text on it

Description automatically generated

1. **Disk Space Check:**
   * **Purpose:** monitor and check the disk space and make sure that it does not exceed the threshold
   * **Process:**
     + **Let a Threshold:**

Define a threshold value that the disk exceeds means that it is in a Critical condition.

* + - **Get Disk Usage:**

Checks current disk space usage percentage using ”df -h”

* + - **Compare Usage by Threshold:**

If the current disk space exceeds the threshold, logs a warning; otherwise, logs that the disk is safe.

* + - **Path to Log file:**

Make a path to save the log file that contains the messages.

1. **Anomalies Check:**
   * **Purpose:** check for any anomalies like CPU, Memory and Network anomalies happen to the system and make an alert for them
   * **Process:**
     + **Let Thresholds:**

Defining threshold values for each check.

* + - **Functions:**

Make a function to get each component status like CPU, Memory and Network and compare the status with the threshold value

* + - **Alerts:**

If the current Status exceeds the threshold, logs a warning; otherwise, logs that the status is safe.

* + - **Path to Log file:**

Make a path to save the log file that contains the Alerts.

* + - **Calling all the functions**

A black background with blue and red text

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen with white text

Description automatically generated

# Java Application Development

**Overview:**

The University Management System allows admins to perform operations such as adding, updating, and selecting information related to students, courses, grades, and departments. It also supports the assignment of courses to students.

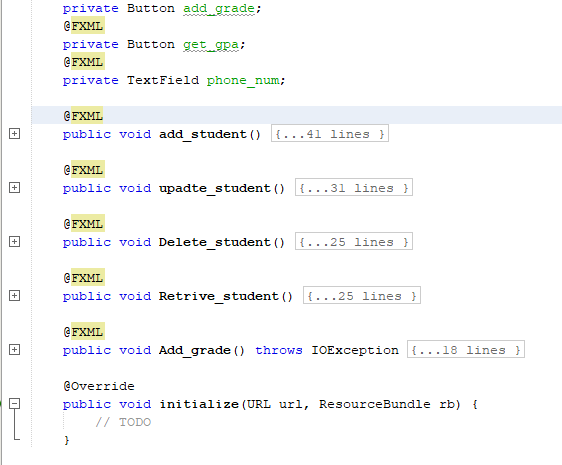
**Code Structure:**

* **FXML Files:** are the scenes, interfaces or GUI used in the application.
* **Controllers Classes:** are classes made of the FXML files used in handling each element in the scene and show information on the GUI using OOP concepts.
* **DTO Classes:** are simple Java classes that are used to transfer data between software application subsystems or layers.
* **DAO Classes:** classes typically provide methods for performing CRUD (Create, Read, Update, Delete) operations on the data source.

**We Have 6 Main Scenes:**

* **Login As Administrator**
* **Welcome Scene**
* **Students Scene (has subscene to add grade to student)**
* **Departments Scene (has 2 subscenes)**
* **Courses Scene (has 2 subscenes)**
* **Report Scene**

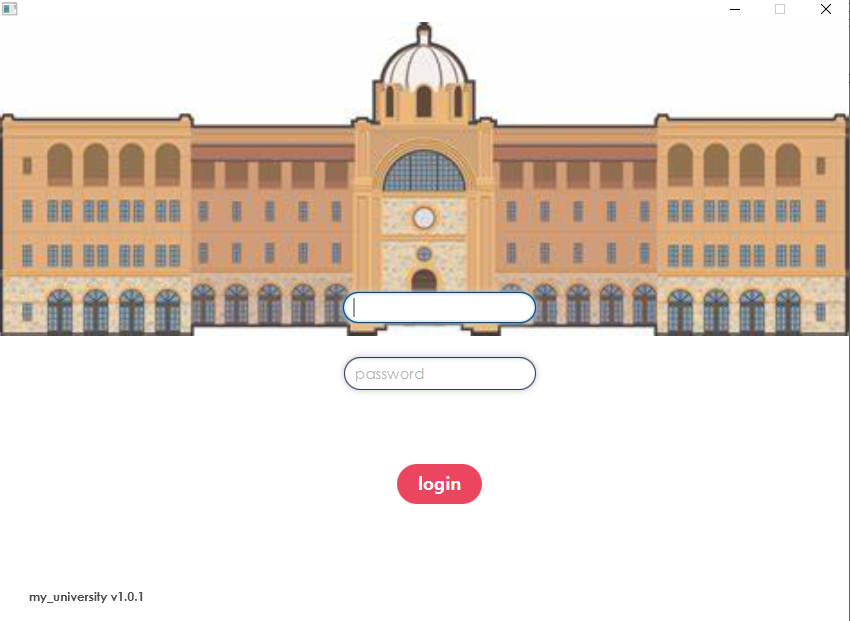
A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

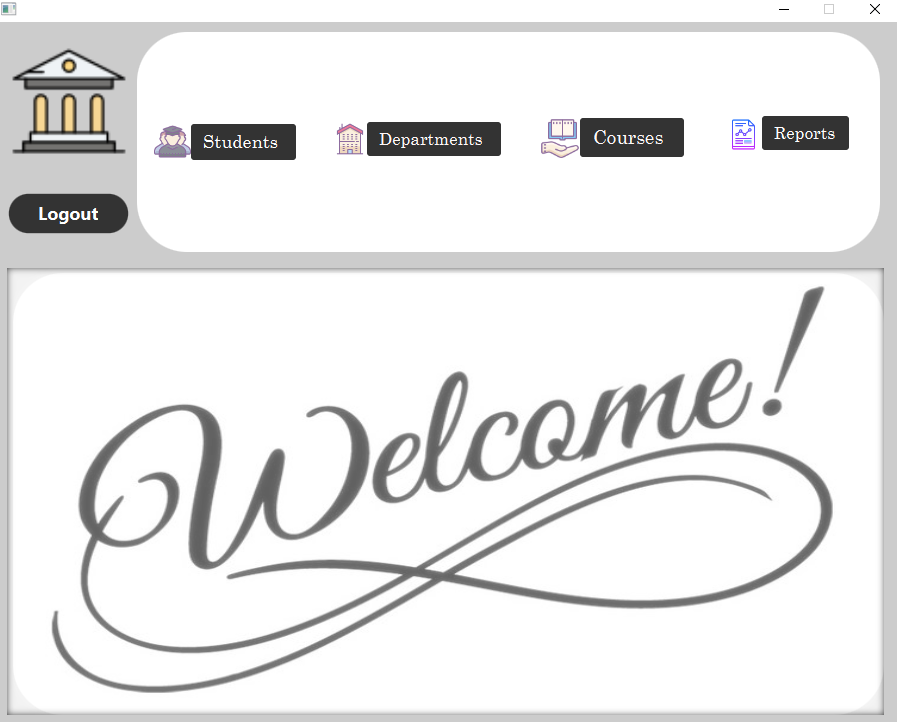
**Application Scenes:**

1. **Login Scene**



This is for admin users only that can login to the system for the staff only.

1. **Welcome Scene**



The welcome scene of our system provides welcome message to the user and the buttons to redirect the user to which he search .

1. A screenshot of a computer

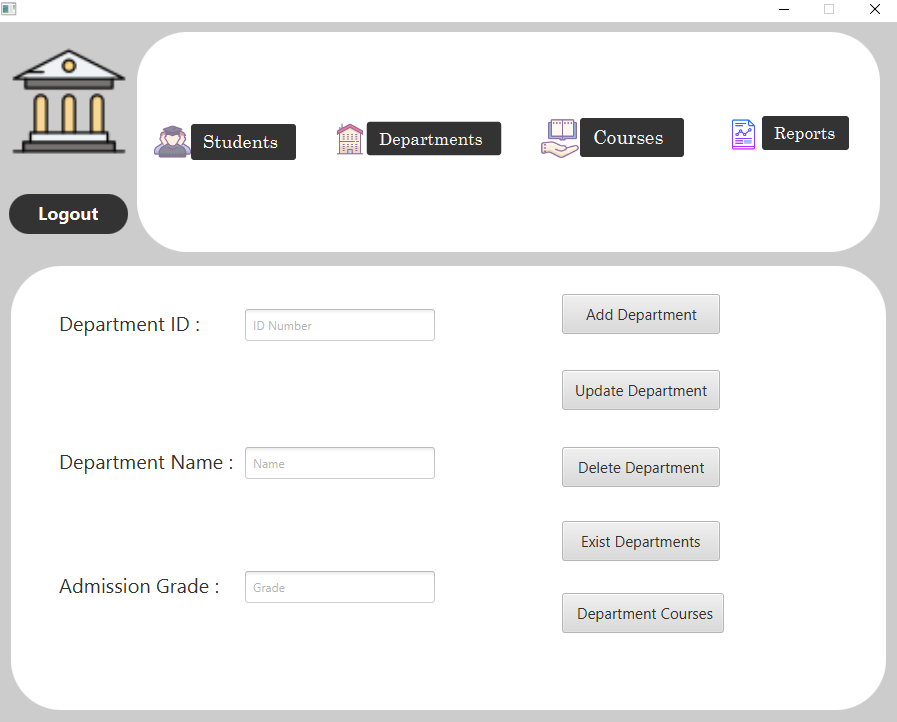
   Description automatically generated**Students Scene**

The student’s scene offers many actions to do over the student data

* **Add New Student:**
  + Fill in all the required fields for a new student.
  + Press the 'Add Student' button to incorporate the new student into the system.
* **Get Student Data:**
  + Write student ID and click “Get Student GPA” to display a brief about him.
  + The brief includes his name , his city, his faculty and his cumulative GPA.
* **Update Student:**
  + Modify the student's information by changing the data in the fields.
  + Press the 'Update Students' button to save the changes.
* **Delete Student:**
  + Press the 'Delete Student' button to remove the student's record from the system by using the student’s name.
* **Add grade:**
  + The user can add a grade to specific Student in a specific course by clicking “Add student grade”**.**
  + **A screenshot of a computer

    Description automatically generated**A subscene will pop up to fill the data and add grade.

* **Validations:**
  + All necessary fields have validations.
  + Validation for non-exist students.
  + Validation for wrong inputs.

**4. Departments Scene**

The departments scene provides information and action for departments:

* **Add New Department:**
  + Enter the necessary data in the fields.
  + Press the 'Add Departments' button to create a new department in the system.
* **Exist Departments:**
  + Click “Exist departments” displayed list of departments and their information.
* **Update Department:**
  + Change the department Info in the fields.
  + Press the 'Update Department' button to save the modifications.
* **Delete Department:**
  + When viewing a selected department, press the 'Delete Department' button to remove the Empty departments from the system.
* **Courses within Department:**
  + View the courses associated with the selected department by clicking “Department Courses”.

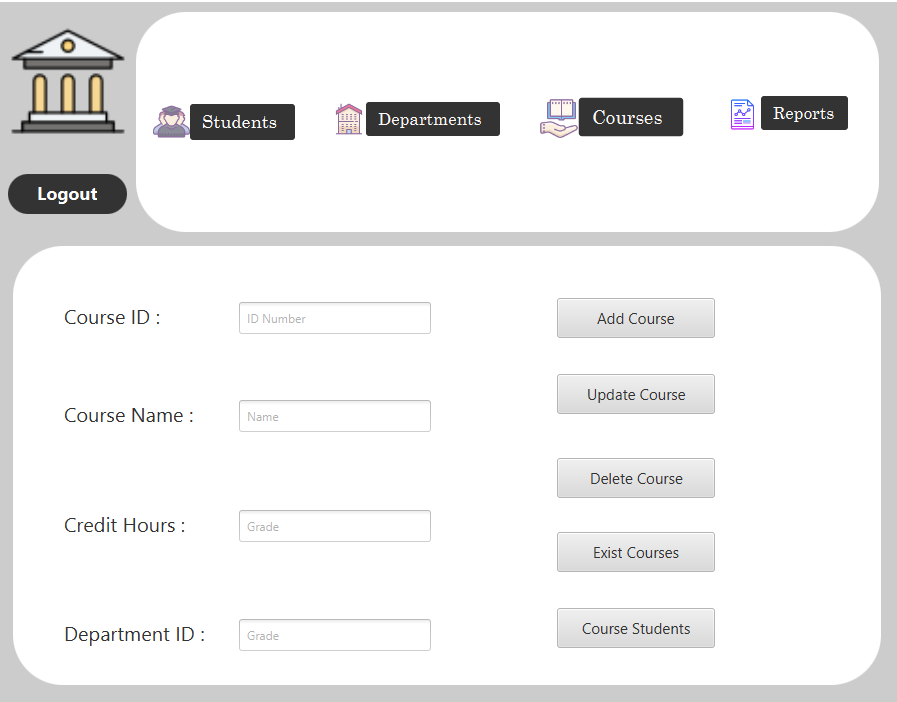
A screenshot of a computer

Description automatically generated

* **Validations:**
  + All necessary fields have validations.
  + Validation for non-exist Departments.
  + Validation for wrong inputs.
  + Can not delete department has students.

A screenshot of a computer

Description automatically generated

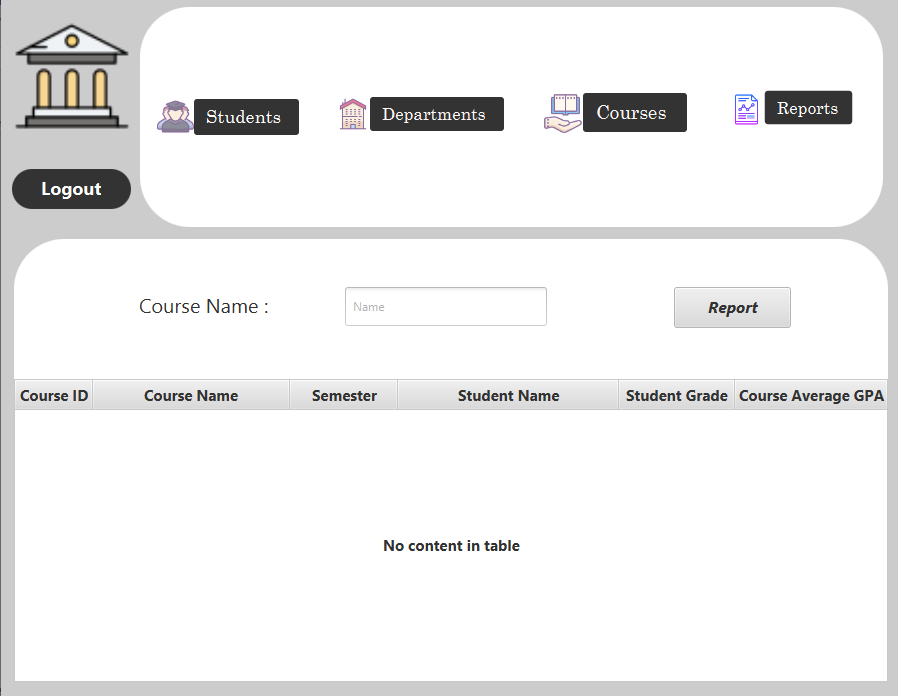
**5. Courses Scene**

The courses scene offers a straightforward interface for managing course information:

* **Add New Course:**

o Fill in the necessary fields for a new course.

* + Press the 'Add Course' button to incorporate the new course into the system.
* **Exist Course:**
  + Click “Exist Courses” displayed list of courses and their information.
* **Update Course:**
  + Write course data to modify its information.
  + Update the relevant fields.
  + Press the 'Update Course' button to save the changes.
* **Delete Course:**
  + Write course Name and press the 'Delete Course' button to remove it from the system but the courses with enrolled students cannot be deleted.
* **Students within the course:**
  + View the Students associated with the course by writing the course name and clicking “Course Students”.
* **Validations:**
  + All necessary fields have validations.
  + Validation for non-exist Courses.
  + Validation for wrong inputs.
  + Can not delete Course has students.

**6. Report Scene**

The report scene provides a summary about a specific course by its name:

* **Display Course Report:**
  + Enter any valid Course name.
  + Click “Report” to show you the summary in the table view.
* **Validations:**
  + For empty name field.
  + For non-exist course.
  + For course without students.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated