Computer and Systems Department

CSE 321: Software Engineering

Student Information System

Team 03

Name	Code
محمد جمال طلعت مطر	1601146
محمد عادل محمد علي	1601214
محمد عبد الله	1601222
محمد محمود عبد الله احمد الانصاري	1601269
مريم عبد الرحمن محمود علي	1601374
مصطفي محمد صديق	1601427
ياسمين علاء عبد الفتاح محمد عليٍ	1601683

Submitted to:

Dr. Gamal A. Ebrahim

Abstract

A "Student Information Management System" software project aims to manage all the students records and administration, in addition to varied facilities and services. Since the student information management systems are the backbone for any successful educational establishment so, this software will act as a handy tool for the whole organization through some features:

Keeping personal information of every student (name, date of birth, ID, address, year of enrolment).

- Enabling the academic staff to keep track of the students' progress.
- Allowing the instructors to record the students' attendance.
- Allowing the students to keep track of their records.
- Enabling the students to select and register their year courses.

Table of Contents

1. Introduction

The student information system is an integral part of this technology. This student information system handles most aspects of student data right from admission, academic information, subject enrolled by the student, overall student performance, and personal information of student. All these elements are integrated into a single database, accessing and tracking data of any student with this desktop base application.

The benefits of Student Information System intuitive user interface with pioneering features. Maximize school management parent’s communication.

Enhancing the efficiency of school administration and managing student data is effortless and easy with the Student Information System software. This system can be customized to include a whole range of activities. It can be easily accessed anytime. Schools can run the Student Information System on minimal hardware affordably and gain a competitive advantage of exploiting the latest in technology staying ahead in competition.

2. General Description

2.1. Product Perspective

Other systems like The University students Statistics and Questionnaires Data,

the Student Activities that may want to make use of this System data.

2.2. General Capabilities

The main capabilities of the system are to facilitate the process of managing students' info, status and courses enrollment help the academic staff to keep track of their students easily.

2.3. General Constrains

The student must provide papers to the student affairs to verify his info and accept his enrollment request.

The number of each student's courses shouldn't exceed a specific number

and this number will depend on his GPA last year.

2.4. User Characteristics

The user who uses this software is either a student, an academic staff member or a staff member

2.5. Environment Description

The user using a system on a computer platform to help him with his requests, job depending on what type of user he is.

2.6. Assumptions and Dependencies

The user chooses only the courses that he is allowed to take this year depending on his track.

2.7. Other Resources Needed

This system needs the data to be provided first to the student affairs in the university in papers to be verified and added to the Database.

3. System Requirements

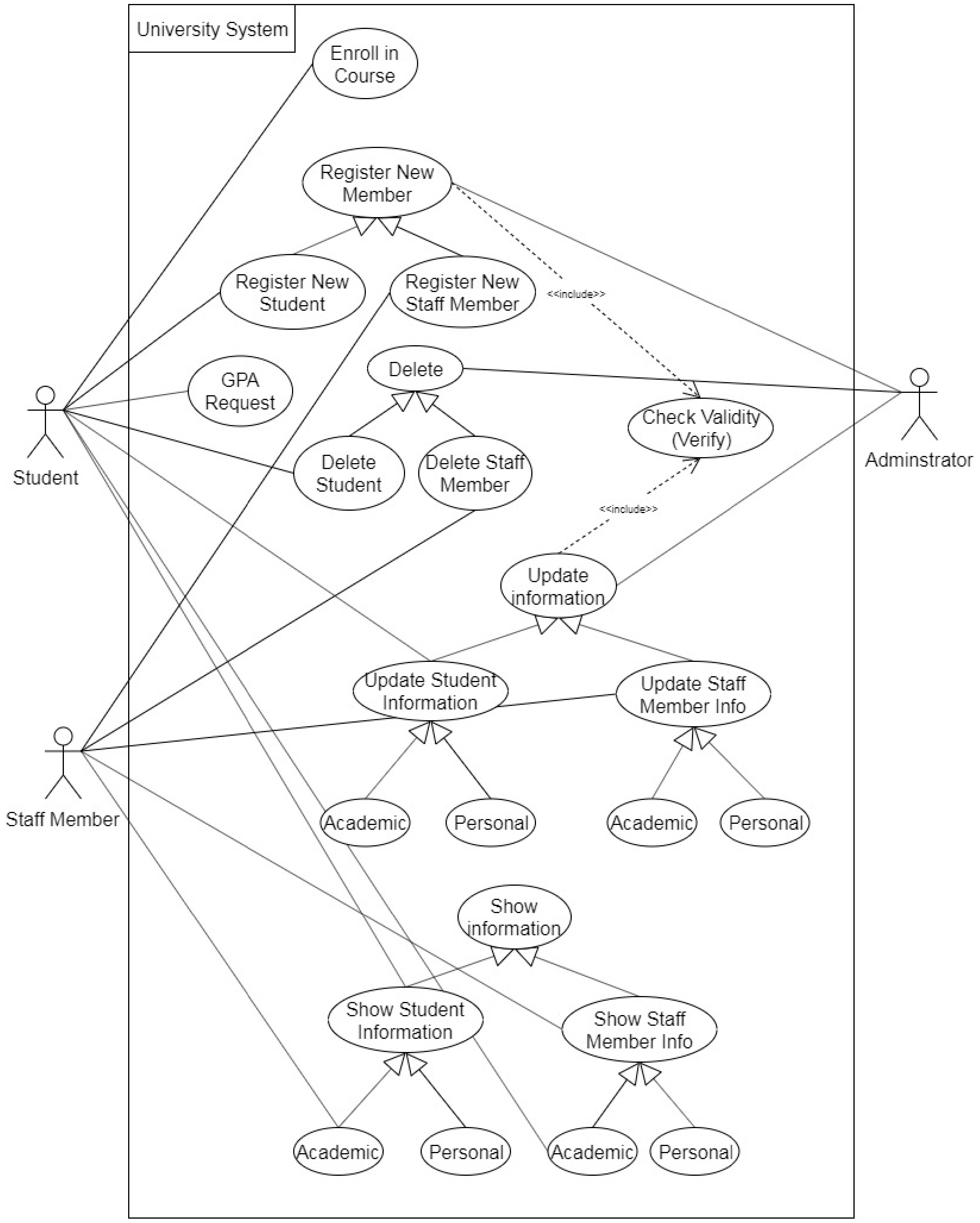
3.1. Functional Requirements

1. Register New Students.
2. Allow Students to enroll in new courses at the beginning of every semester.
3. Show every student his personal and academic information.
4. Update students' information with the supervision of the administrator.
5. Student can make a GPA request.
6. Delete a student from the database by the approve of the administrator.
7. Register new staff member.
8. Show every staff member his personal and academic information.
9. Update staff members' information with the supervision of the administrator.
10. Delete a staff member from the database by the approve of the administrator.

3.2. Non-Functional Requirements

1. The Software must be implements in C++ programming language.
2. The project development has a span of 1 Month.
3. The project will have a limited budget of $10,000.
4. If software falls for any reason it should recover in no more than 1 minute.
5. The software should be able to serve about 25,000 students and staff members.

4. Use-Case Diagram



5. Narrative Description/Swimlane Diagram of Use Cases

Use case name: Request GPA

Goal In context: Student wants to show his GPA.

Preconditions: Student must have an account.

Successful End Condition: GPA is displayed.

Failed End Condition: GPA request is failed.

Primary Actors: Student.

Secondary Actors: None.

Trigger: Student wants to know his grades.

Main Flow:

1) Student logins in system.
2) Student request system to make GPA report.
3) System gets GPA information of the student.
4) System displays the results.

Use case name: Enroll in courses

Goal In context: Student wants to enroll in courses.

Preconditions: Student must have an account.

Successful End Condition: Student enrolls in courses successfully.

Failed End Condition: Student enrollment is failed.

Primary Actors: Student.

Secondary Actors: None.

Trigger: Student asks the system to enroll in courses.

Main Flow:

1) Student Logins.
2) Student decides what courses he wants to enroll in.
3) Student tells system the courses.
4) System saves the chosen courses into student data.

Use case name: Register New Student

Goal In context: To register new student into the system.

Preconditions: Student must have his credentials.

Successful End Condition: Student is registered successfully.

Failed End Condition: Student registration failed.

Primary Actors: Student.

Secondary Actors: Admin.

Base use case: Register New Member.

Included use case: Check validity.

Trigger: Student decides to join the school.

Main Flow:

1) New student provides his credentials to admin.
2) Credentials are checked.
3) Include::Check Validity.
4) Admin enters the student information into the system.
5) System provides username and password.
6) Admin gives username and password to student.

Use case name: Register New Staff Member.

Goal In context: To register new staff member into the system.

Preconditions: Staff member must have his credentials.

Successful End Condition: Staff member is registered successfully.

Failed End Condition: Staff member registration failed.

Primary Actors: Staff member.

Secondary Actors: Admin.

Base use case: Register New Member.

Included use case: Check validity.

Trigger: New Staff member is employed in school.

Main Flow:

1) New Staff member provides his credentials to admin.
2) Credentials are checked.
3) Include::Check Validity.
4) Admin enters the Staff member information into the system.
5) System provides username and password.
6) Admin gives username and password to staff member.

Use case name: Delete Student

Goal In context: To delete student from the system.

Preconditions: Student must be registered to the system.

Successful End Condition: Student is deleted successfully.

Failed End Condition: Student deletion is failed.

Base use case: Delete.

Primary Actors: Student.

Secondary Actors: Admin.

Trigger: Student causes very big problem in school.

Main Flow:

1) Student is fired from the school.
2) Admins asks for student information.
3) Admin enters student information to system.
4) Admin asks the system to delete student.
5) System deletes student.

Use case name: Delete Staff Member

Goal In context: To delete staff member from the system.

Preconditions: Staff member must be registered to the system.

Successful End Condition: Staff member is deleted successfully.

Failed End Condition: Staff member deletion is failed.

Base use case: Delete.

Primary Actors: Staff member.

Secondary Actors: Admin.

Trigger: Student causes very big problem in school.

Main Flow:

1) Staff member apply for resignation from school.
2) Admins asks for staff member information.
3) Admin enters staff member information to system.
4) Admin asks the system to delete staff member.
5) System deletes staff member.

Use case name: Show student information

Goal In context: Student wants to see his information.

Preconditions: Student must have an account on system.

Successful End Condition: Student shows his information.

Failed End Condition: Student can’t show his information.

Primary Actors: Student.

Secondary Actors: None.

Trigger: Student asks the system to show his information.

Main Flow:

1) Student logins to the system.
2) Student asks the system to show his information.
3) Student chooses the type of information (academic/personal).
4) System shows information to student.

Use case name: Show academic student information

Goal In context: To show student academic information.

Preconditions: Staff member must have account and student must be registered to system.

Successful End Condition: Staff member shows the academic info of student.

Failed End Condition: Staff member can’t show the academic info of a student.

Base use case: Show student information.

Primary Actors: Staff member.

Secondary Actors: None.

Trigger: Staff member decides to show student information.

Main Flow:

1) Staff member logins to system.
2) Staff member chooses to show academic student info.
3) Staff member enters student name to system.
4) System shows the academic information to staff member.

Use case name: Show Staff Member Information

Goal In context: Staff member wants to see his information.

Preconditions: Staff member must have an account on system.

Successful End Condition: Staff member shows his information.

Failed End Condition: Staff member can’t show his information.

Primary Actors: Staff member.

Secondary Actors: None.

Trigger: Staff member asks the system to show his information.

Main Flow:

1) Staff member logins to the system.
2) Staff member asks the system to show his information.
3) Staff member chooses the type of information (academic/personal).
4) System shows information to Staff member.

Use case name: Show academic staff member information

Goal In context: To show staff member academic information.

Preconditions: Student must have account and Staff member must be registered to system.

Successful End Condition: Student shows the academic info of student.

Failed End Condition: Student can’t show the academic info of a student.

Base use case: Show Staff member information.

Primary Actors: Student.

Secondary Actors: None.

Trigger: Student decides to show student information.

Main Flow:

1) Student logins to system.
2) Student chooses to show academic staff member info.
3) Student enters staff member name to system.
4) System shows the academic information to student.

Use case name: Update Student Information

Goal In context: To update student information.

Preconditions: Student must have an account and registered to the system.

Successful End Condition: Information is updated.

Failed End Condition: Information isn’t updated.

Base use case: Update information.

Included use case: Check Validity.

Primary Actors: Student.

Secondary Actors: Administrator.

Trigger: Student’s information is changed or requires update.

Main Flow:

1) Student provides the new information to the admin.
2) Student information is checked.
3) Include::Check validity.
4) Admin enters updated information to the system.
5) System updates student information.

Use case name: Update Staff Member Information

Goal In context: To update Staff Member information.

Preconditions: Staff Member must have an account and registered to the system.

Successful End Condition: Information is updated.

Failed End Condition: Information isn’t updated.

Base use case: Update information.

Included use case: Check Validity.

Primary Actors: Staff Member.

Secondary Actors: Administrator.

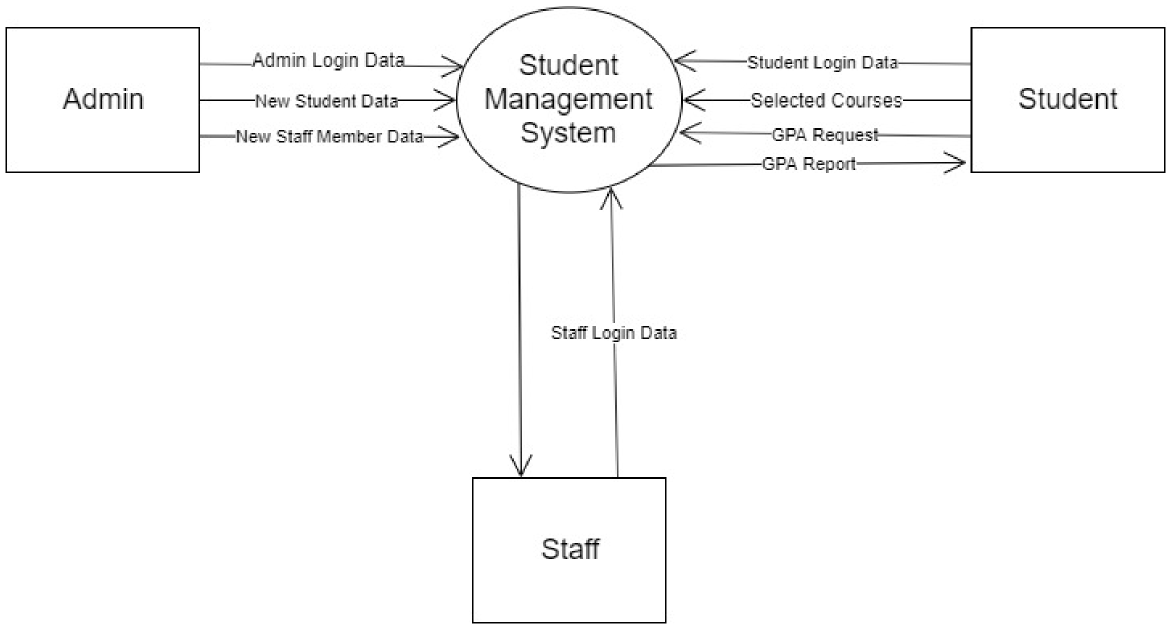
Trigger: Staff member’s information is changed or requires update.

Main Flow:

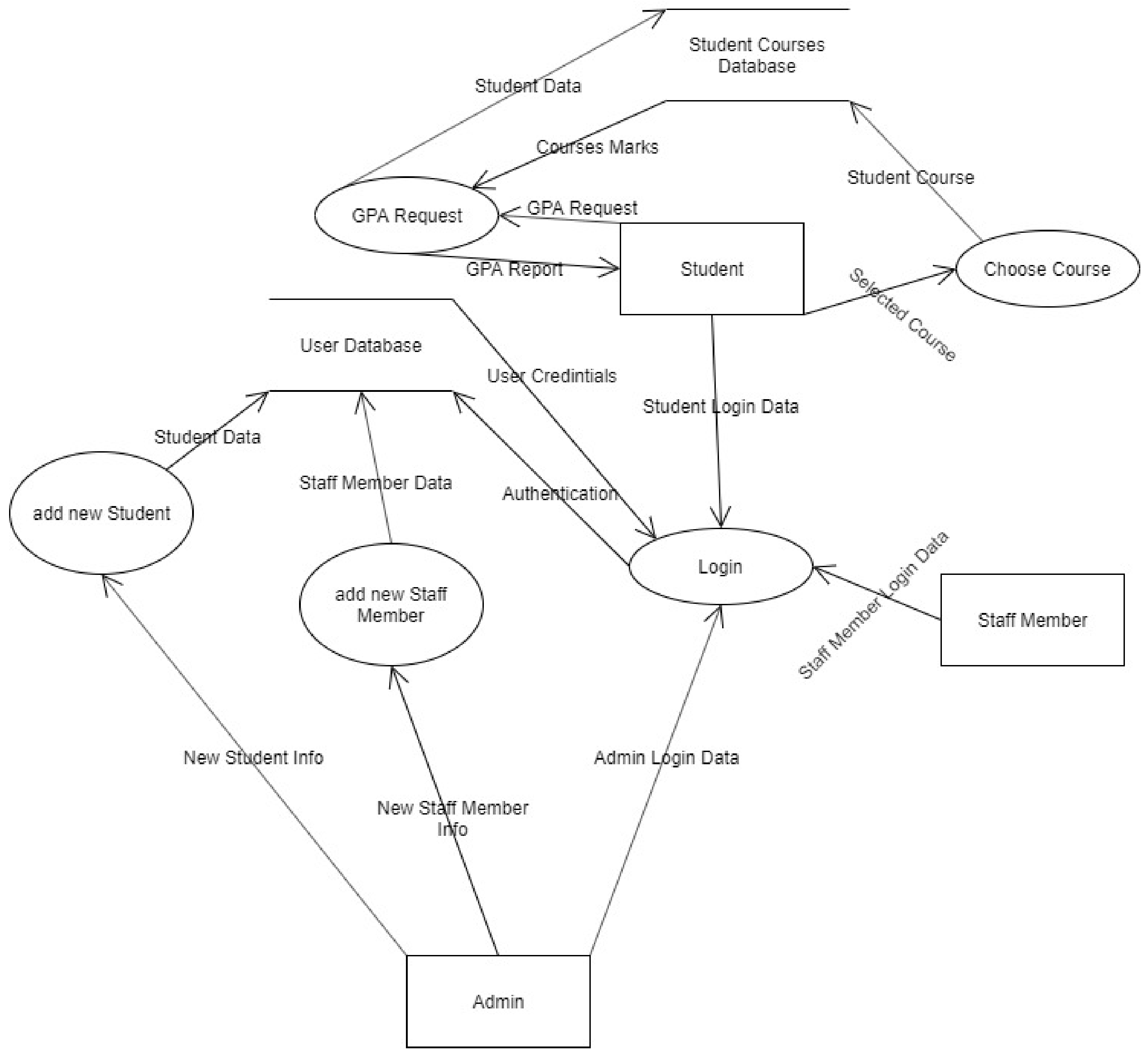
1) Staff member provides the new information to the admin.
2) Staff member information is checked.
3) Include::Check validity.
4) Admin enters updated information to the system.
5) System updates Staff member information.

6. Data Model

6.1. Context

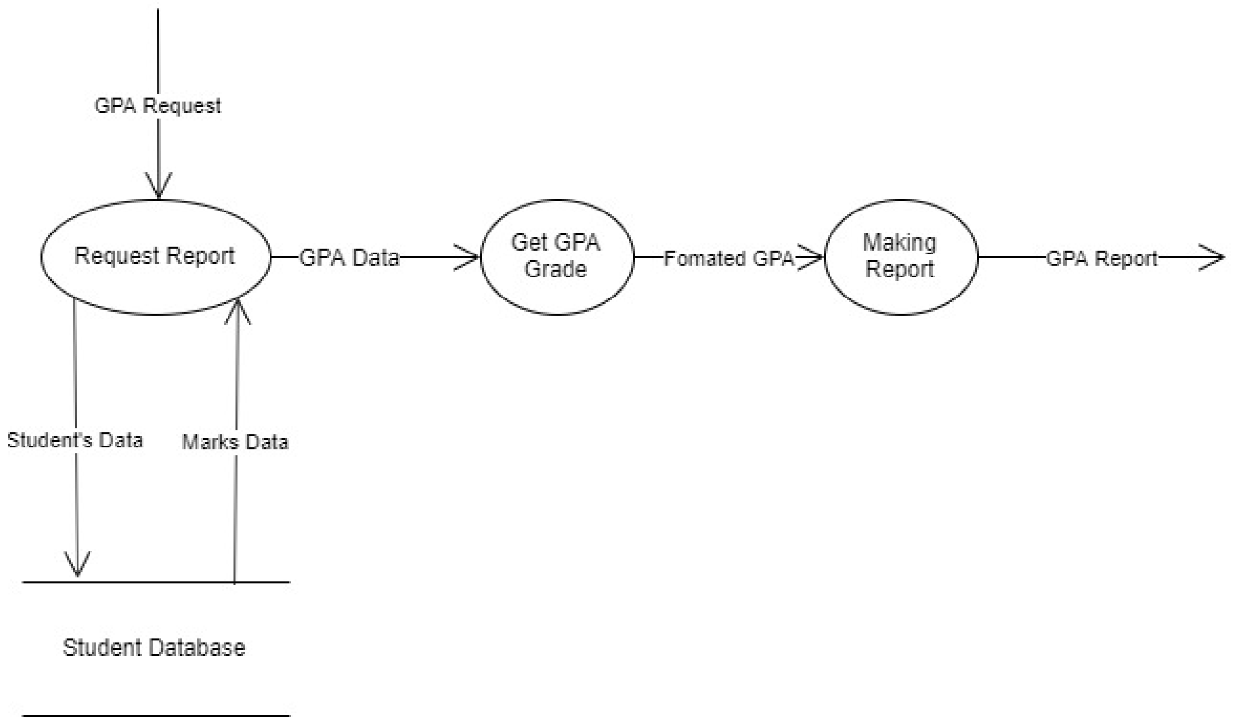


6.2. Data Flow Level 0



6.3. Data Flow Level 1

GPA Report



7. Requirements Validation

7.1. Requirement Traceability Matrix

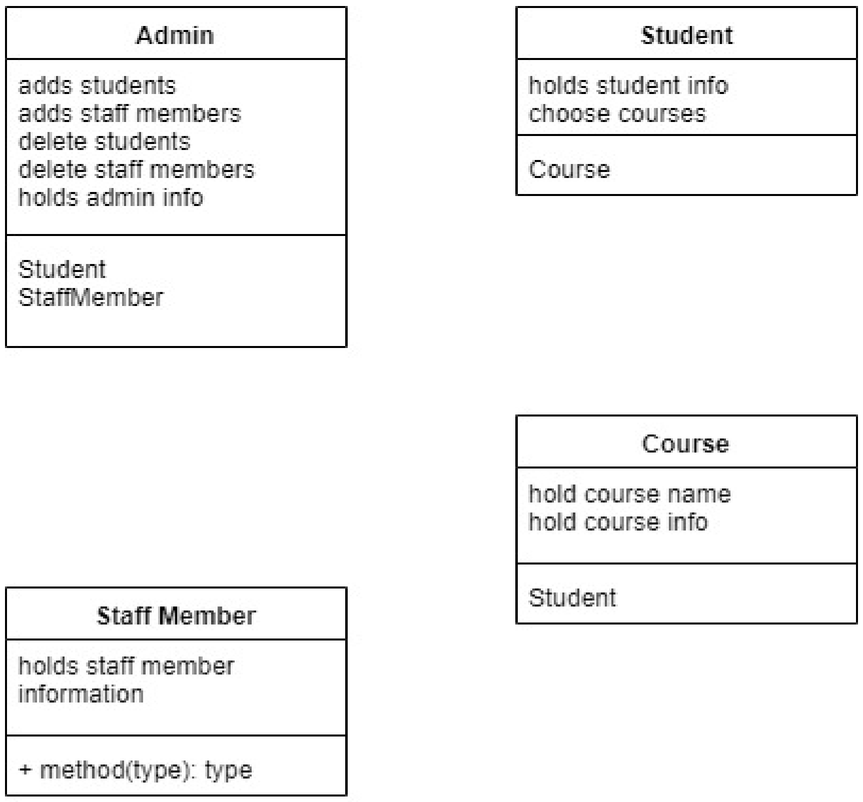
	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8	Req9	Req10
Req1										
Req2										
Req3				√						
Req4			√							
Req5										
Req6										
Req7										
Req8									√	
Req9								√		
Req10										

7.2. Source Traceability Matrix

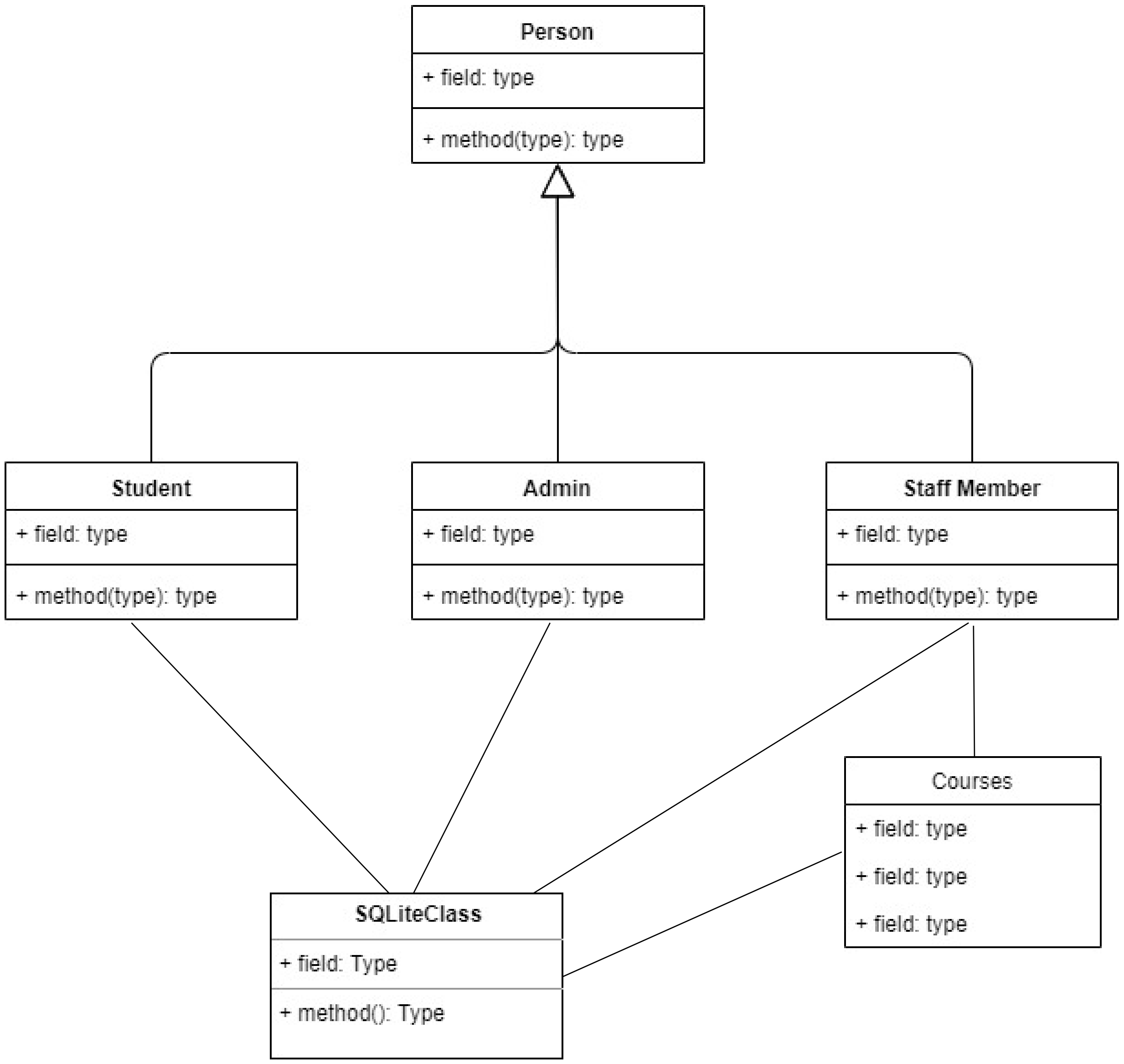
	Student	Staff	Admin	Developer	Sponsor	PM	EM
Req1	√		√				
Req2	√	√					
Req3	√						
Req4	√		√				
Req5	√						
Req6	√		√				
Req7		√	√				
Req8	√	√					
Req9		√	√				
Req10		√	√				
Req11				√		√	
Req12							√
Req13					√		√
Req14	√	√	√			√	
Req15	√	√	√			√	

8. Class Model

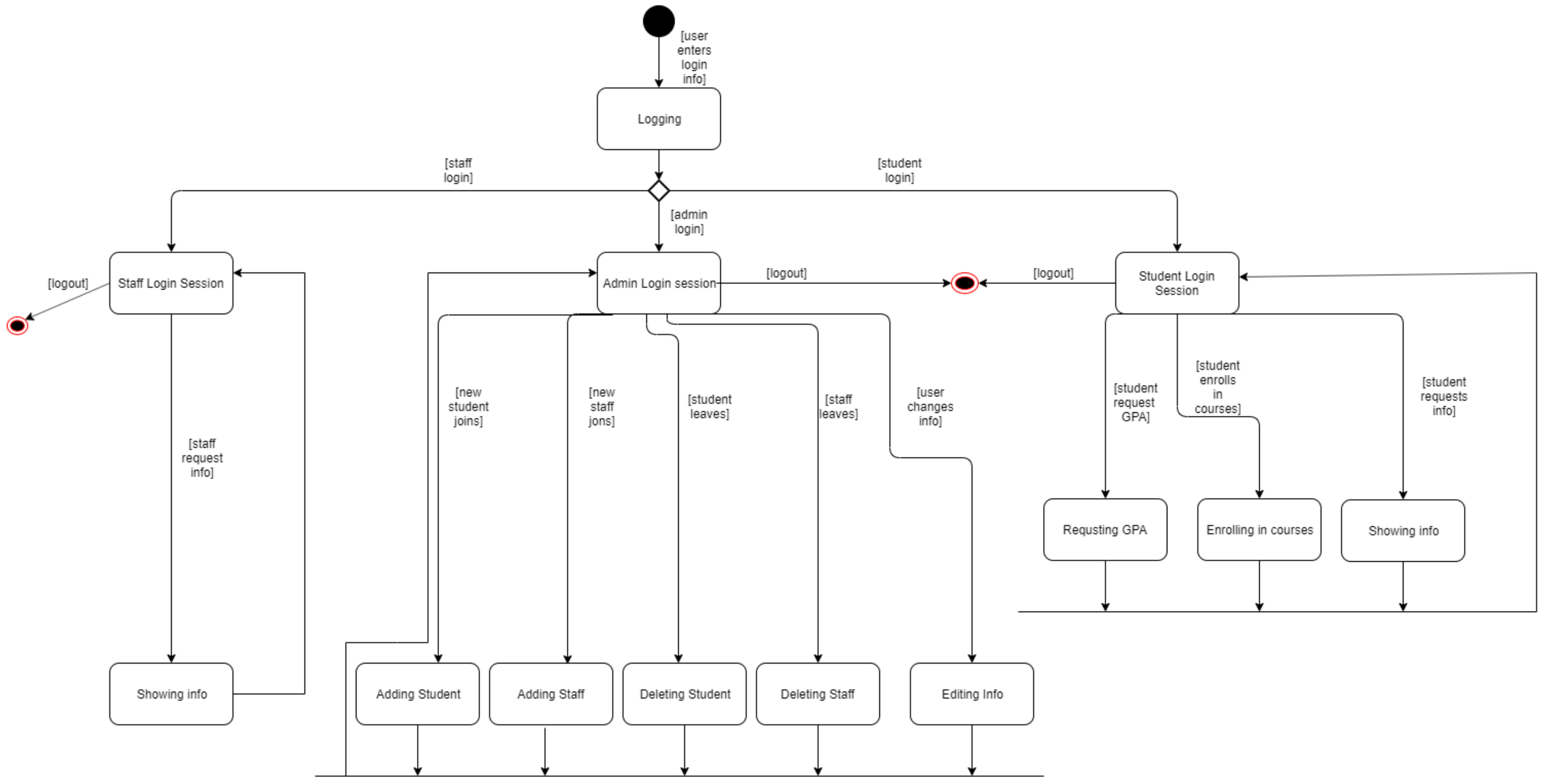
8.1. CRC



8.2. Class Diagram

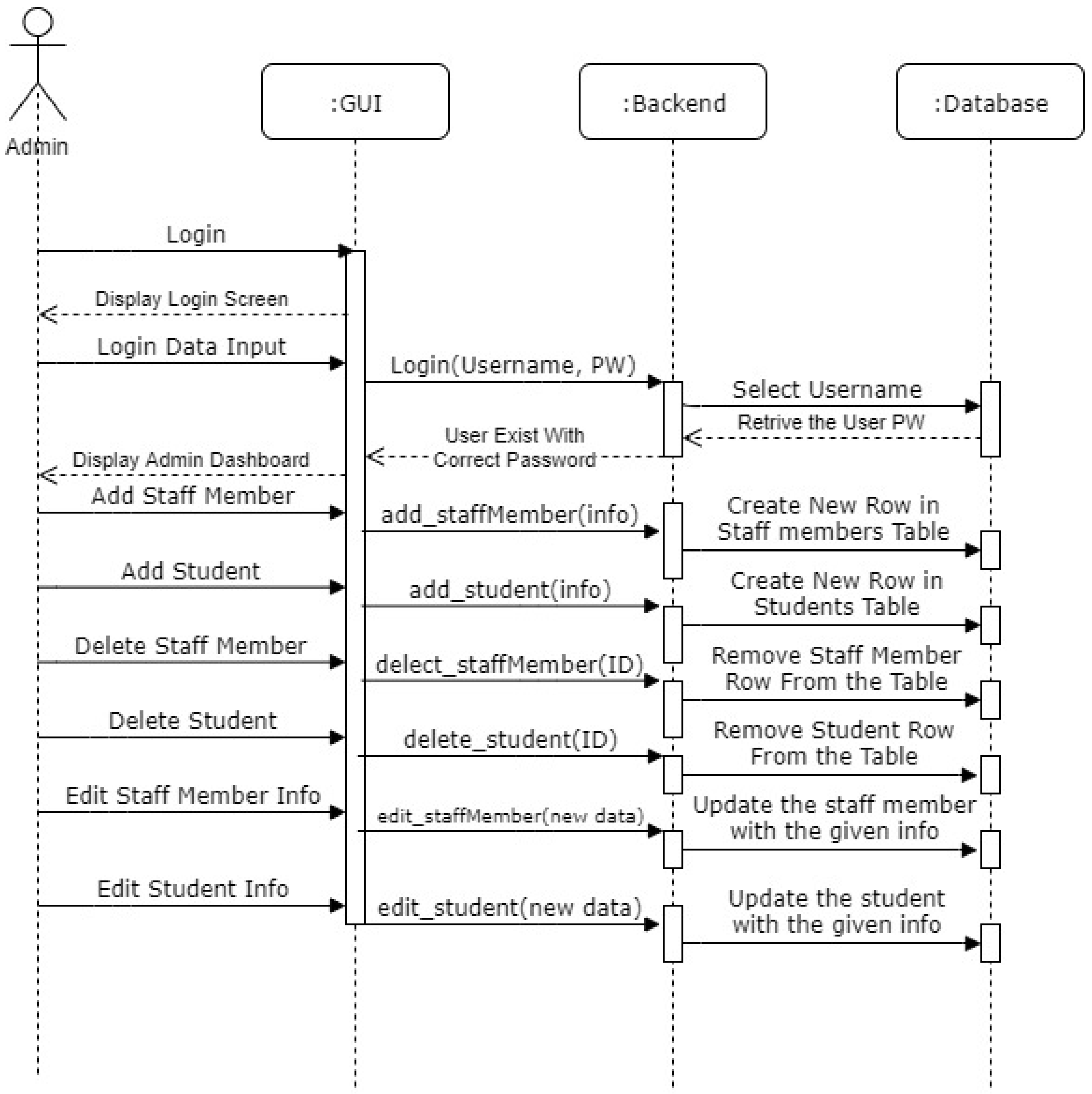


9. State Diagram

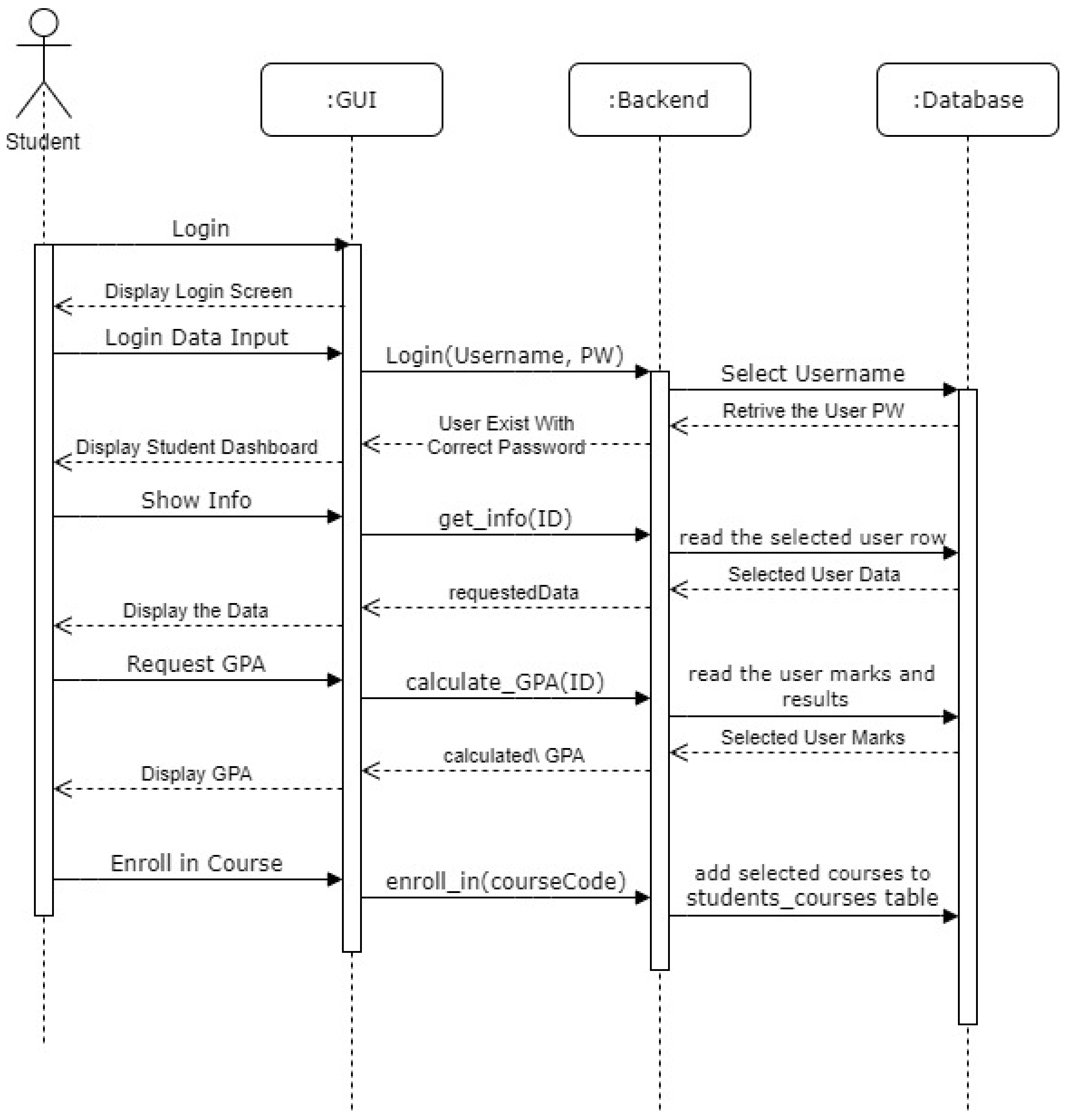


10. Interaction Diagram

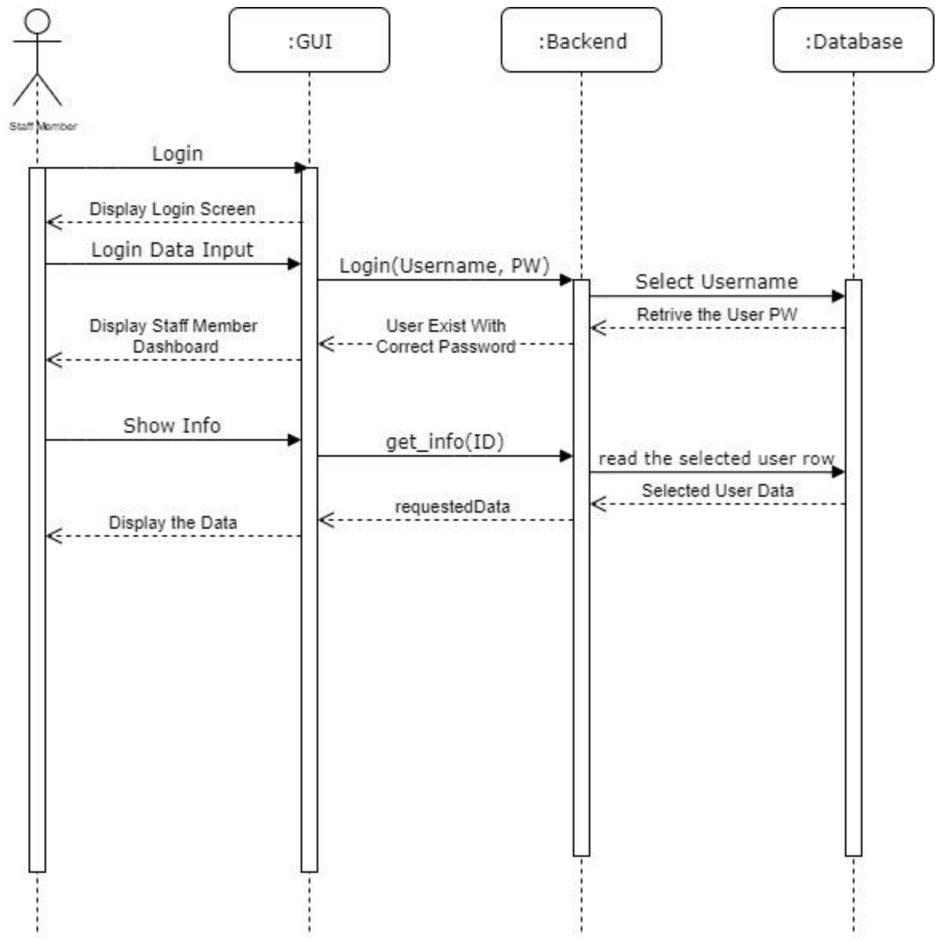
10.1. Admin



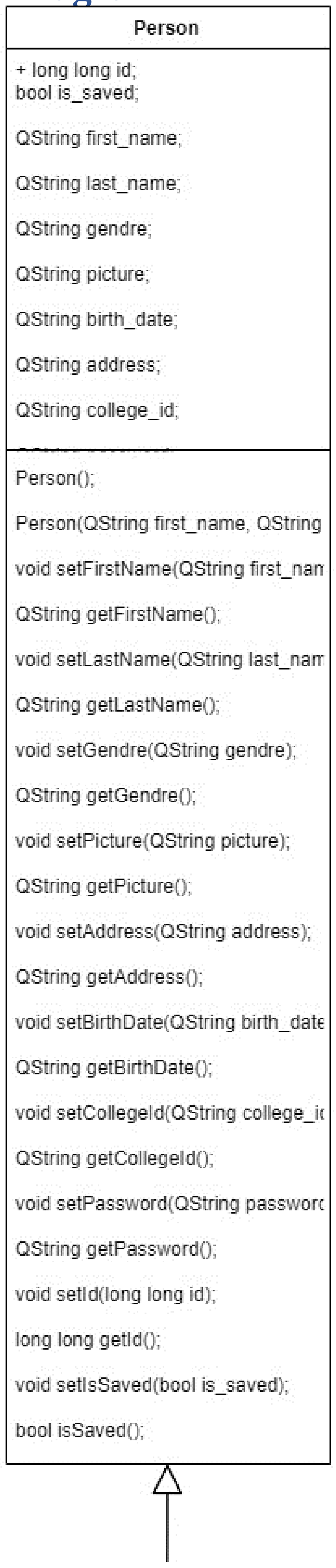
10.2. Student


Student
:GUI
:Backend
:Database
Login
Display Login Screen
Login Data Input
Login(Username, PW)
Select Username
Retrive the User PW
User Exist With Correct Password
Display Student Dashboard
Show Info
get_info(ID)
read the selected user row
Selected User Data
requestedData
Display the Data
Request GPA
calculate_GPA(ID)
read the user marks and results
Selected User Marks
calculated\ GPA
Display GPA
Enroll in Course
enroll_in(courseCode)
add selected courses to students_courses table

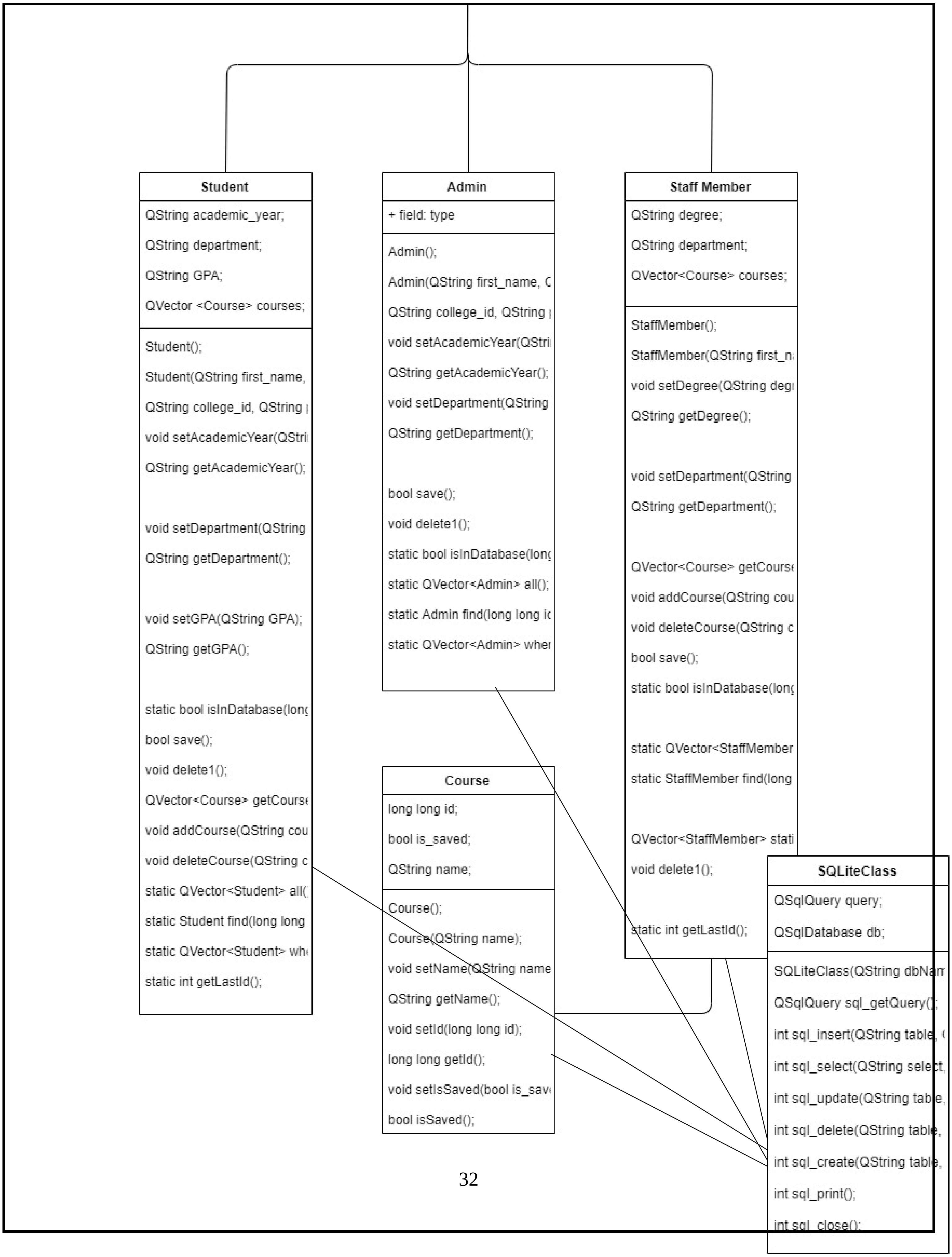
10.3. Staff Member



11. Detailed Class Diagram



```
Person

+ long long id;
bool is_saved;

QString first_name;

QString last_name;

QString gendre;

QString picture;

QString birth_date;

QString address;

QString college_id;

Person();

Person(QString first_name, QString

void setFirstName(QString first_nan

QString getFirstName();

void setLastName(QString last_nam

QString getLastName();

void setGendre(QString gendre);

QString getGendre();

void setPicture(QString picture);

QString getPicture();

void setAddress(QString address);

QString getAddress();

void setBirthDate(QString birth_date

QString getBirthDate();

void setCollegeId(QString college_i

QString getCollegeId();

void setPassword(QString passwor

QString getPassword();

void setId(long long id);

long long getId();

void setIsSaved(bool is_saved);

bool isSaved();
```


Student
QString academic_year;
QString department;
QString GPA;
QVector <Course> courses;
Student();
Student(QString first_name,
QString college_id, QString
void setAcademicYear(QStri
QString getAcademicYear();
void setDepartment(QString
QString getDepartment();
void setGPA(QString GPA);
QString getGPA();
static bool isInDatabase(long
bool save();
void delete1();
QVector<Course> getCourse
void addCourse(QString cou
void deleteCourse(QString c
static QVector<Student> all(
static Student find(long long
static QVector<Student> whe
static int getLastId();
Admin
+ field: type
Admin();
Admin(QString first_name, C
QString college_id, QString
void setAcademicYear(QStri
QString getAcademicYear();
void setDepartment(QString
QString getDepartment();
bool save();
void delete1();
static bool isInDatabase(long
static QVector<Admin> all();
static Admin find(long long ic
static QVector<Admin> wher
Staff Member
QString degree;
QString department;
QVector<Course> courses;
StaffMember();
StaffMember(QString first_n
void setDegree(QString degr
QString getDegree();
void setDepartment(QString
QString getDepartment();
QVector<Course> getCourse
void addCourse(QString cou
void deleteCourse(QString c
bool save();
static bool isInDatabase(long
static QVector<StaffMember
static StaffMember find(long
QVector<StaffMember> stati
void delete1();
static int getLastId();
Course
long long id;
bool is_saved;
QString name;
Course();
Course(QString name);
void setName(QString name
QString getName();
void setId(long long id);
long long getId();
void setIsSaved(bool is_sav
bool isSaved();
SQLiteClass
QSqlQuery query;
QSqlDatabase db;
SQLiteClass(QString dbNam
QSqlQuery sql_getQuery();
int sql_insert(QString table, (
int sql_select(QString select,
int sql_update(QString table,
int sql_delete(QString table,
int sql_create(QString table,
int sql_print();
int sql_close();

12. Data Model Design

- students table
  - Attributes: id, first_name, last_name, gendre, picture, birth_date, address, college_id, password, academic_year, department
  - Key: id

- staff_members table
  - Attributes: id, first_name, last_name, gendre, picture, birth_date, address, college_id, password, degree, department
  - Key: id

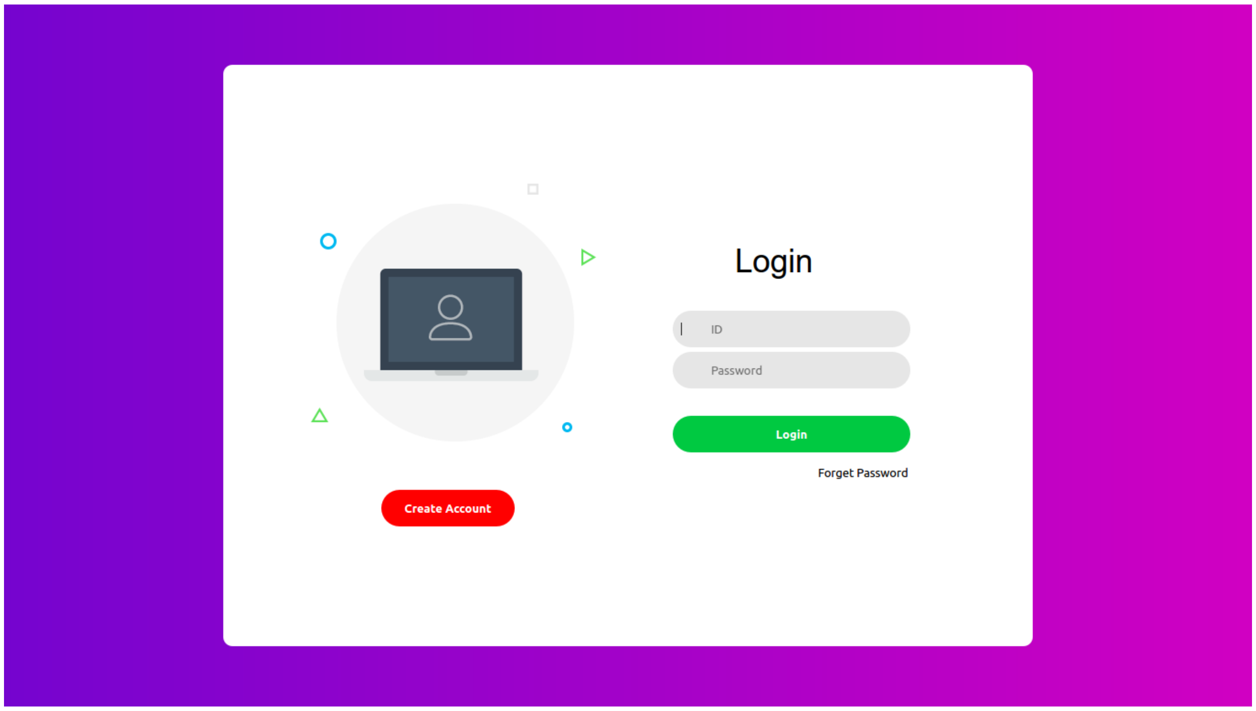
- admins table
  - Attributes: id, first_name, last_name, gendre, picture, birth_date, address, college_id, password
  - Key: id

- courses table
  - Attributes: id, name
  - Key: id

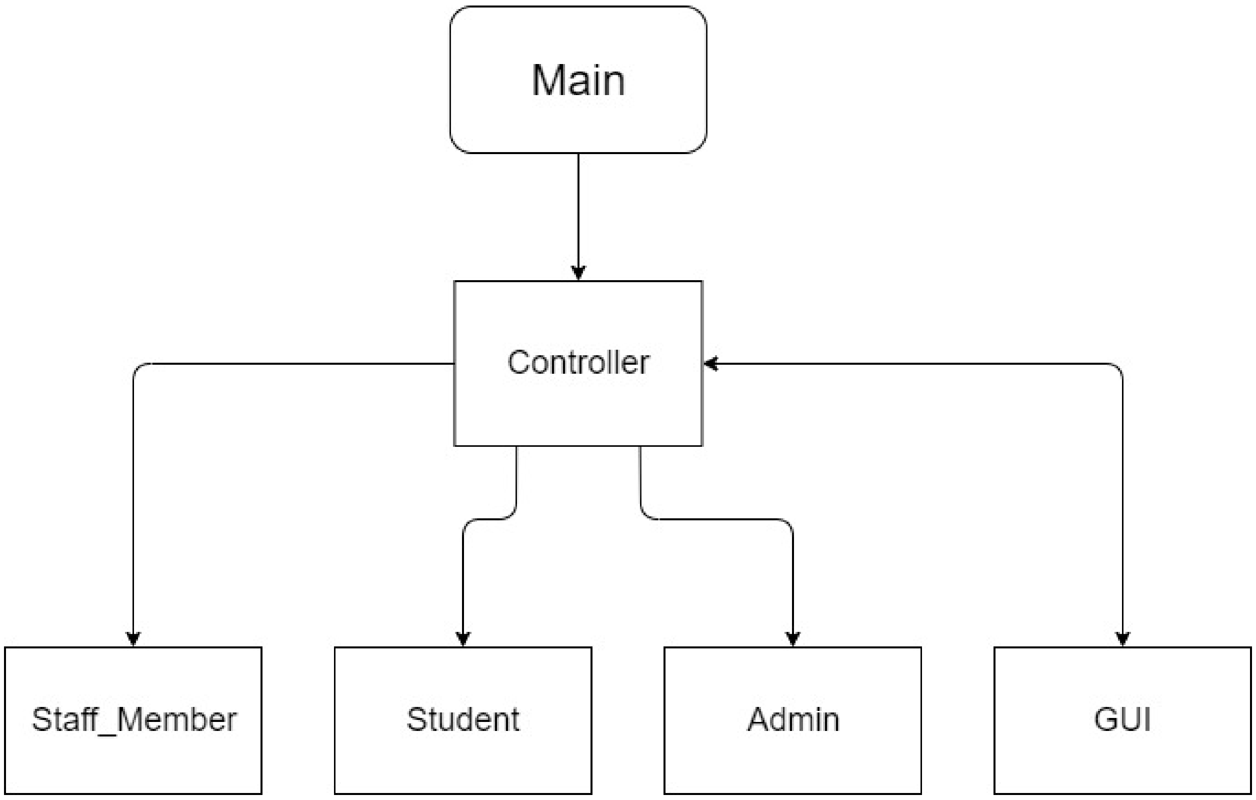
- students_courses table
  - Attributes: course_id, student_id

- courses_staff_members table
  - Attributes: course_id, staff_member_id

13. User Interface Design



14. Client-Object Relation Diagram



15. Detailed Design

15.1. Person

```
#include "person.h"
#include <QString>

Person::Person()

{

   set the id property to 0

   set the is_saved property to false

   set the first_name property to empty string

   set the last_name property to empty string

   set the gendre property to empty string

   set the picture property to empty string

   set the birth_date property to empty string

   set the address property to empty string

   set the college_id property to empty string

   set the password property to empty string

}
```

```
Person::Person(QString first_name, QString last_name, QString gendre, QString
picture, QString birth_date, QString address, QString college_id, QString
password) {
    set the id property to 0
    set the is_saved property to false
    set the first_name property to the specified value
    set the last_name property to the specified value
    set the gendre property to the specified value
    set the picture property to the specified value
    set the birth_date property to the specified value
    set the address property to the specified value
    set the college_id property to the specified value
    set the password property to the specified value
}


void Person::setFirstName(QString first_name) {
    set first_name attribute to the passed value
}


QString Person::getFirstName() {
    return the value of the first_name attribute
}
void Person::setLastName(QString last_name) {
    set last_name attribute to the passed value
}
QString Person::getLastName() {
    return the value of the last_name attribute
}
```

```
void Person::setGendre(QString gendre) {
   set gendre attribute to the passed value
}
QString Person::getGendre() {
   return the value of the gendre attribute
}
void Person::setPicture(QString picture) {
   set picture attribute to the passed value
}

QString Person::getPicture() {
   return the value of the picture attribute
}

void Person::setAddress(QString address) {
   set address attribute to the passed value
}

QString Person::getAddress() {
   return the value of the address attribute
}
void Person::setBirthDate(QString birth_date) {
   set birth_date attribute to the passed value
}
QString Person::getBirthDate() {
   return the value of the birth_date attribute
}
```

```
void Person::setCollegeId(QString college_id){
   set college_id attribute to the passed value
}

QString Person::getCollegeId(){
   return the value of the college_id attribute
}

void Person::setPassword(QString password){
   set password attribute to the passed value
}

QString Person::getPassword(){
   return the value of the password attribute
}

void Person::setId(long long id) {
   set id attribute to the passed value
}

long long Person::getId() {
   return the value of the id attribute
}

void Person::setIsSaved(bool is_saved) {
   set is_saved attribute to the passed value
}
```

```
bool Person::isSaved() {
    return the value of the is_saved attribute
}
```

15.2. Admin

```
#include "admin.h"

#include "sqliteclass.h"

#include "globalDbObject.h"


static QString admins_table = "admins";

static QStringList admins_columns = {"first_name","last_name", "gendre", "picture",

                                "birth_date", "address", "college_id", "password"};


Admin::Admin() : use the person constructor

{

}


Admin::Admin(QString first_name, QString last_name, QString gendre, QString picture, QString birth_date, QString address, QString college_id, QString password)

    : use the person constructor

{

}
```

```
QVector<Admin> Admin::all() {
    open a connection with the database
    create a vector to store all the admins in
    make a query to fetch all the admins from the database
    while(there another admin to be fetched) {
        fetch all the attributes from the table and use them to create a admin object

        make a query to fetch all the courses related to this admin from the database
                while(there another course to be fetched) {
                        fetch all the attributes from the table and use them to create
a admin object

                        make a query to fetch all the courses related to this admin
from the database
                }
    }
    close the connection with the database
    return the vector of admins
}
```

```
bool Admin::save(){
    get the id of the admin
    create a query that will fetch the admin with the id
    get the values of attributes of the object
    if(the admin exists in the database){
        update the attributes of the admin in the database

        while(there is another course in the vector){
            get the id of the course
            create a query that will fetch the row that links the admin with the course
            if(this row doesn't exist){
                insert this row in the database
            }
        }
        return true indicating that the method finished and the admin was already in
the database
    }
    insert this admin into the database
    while(there is another course in the vector){
        create a query that will link the course to the admin in the database
                        execute this query
    }
    return false indicating that the method finished and the admin wasn't in the
database
}
```

```
bool Admin::isInDatabase(long long id) {
   open a connection with the database


   create a query to fetch the data of the admin with the specified id
   if(a admin with such an id exists) {
      close the connection with the database
      return true indicating that the admin exists
   }
   close the connection with the database
   return false indicating that the admin doesn't exist
}
void Admin::delete1(){
   SQLiteDb.sql_delete(admins_table, "id = " + QString::number(getId()));
}
Admin Admin::find(long long id) {
   create a connection with the database
         create a query that will fetch the data of a admin with the specified id
   execute this query
   fetch all the data from the database
   create a admin object with all the data from the database
         create a query that will fetch all the course related to the admin from the
database
   while(there is another course to be fetched) {
      add this course to the vector of courses
   }
         close the connection with the database
   return the admin object
}
```

```
QVector<Admin> Admin::where(QString column, QString value)
{
    crate a query to fetch all the admins that have the value of the specified with the
specified value
    create a vector to store the admins in
    while (there is another admin to be fetched) {
        add this admin to the vector
    }
    return the vector
}
```

15.3. Student

```
#include "student.h"

#include "sqliteclass.h"

#include "globalDbObject.h"


static QString students_table = "students";

static QStringList students_columns = {"first_name","last_name", "gendre", "picture",

                                "birth_date", "address", "college_id", "password", "academic_year", "department"};


Student::Student() : Person()

{

   set the degree property to "0" string

   set the department property to empty string

}


Student::Student(QString first_name, QString last_name, QString gendre, QString picture, QString birth_date, QString address, QString college_id, QString password,

                 QString academic_year, QString department) : use the person constructor {

   set the academic_year property to the specified value

   set the department property to the specified value

}


void Student::setAcademicYear(QString academic_year){

   set academic_year attribute to the passed value

}
```

```
void Student::setDepartment(QString department) {
   set department attribute to the passed value
}

QString Student::getDepartment() {
   return the value of the department attribute
}

QString Student::getAcademicYear(){
   return the value of the academic_year attribute
}

QVector<Course> Student::getCourses() {
   return this->courses;
}

void Student::setCollegeId(QString college_id){
   set college_id attribute to the passed value
}

QString Student::getCollegeId(){
   return the value of the college_id attribute
}
```

```
void Student::addCourse(QString course_name) {
   while(there are more courses that hadn't been compared with) {
      if(the passed course name is the same as this course name) {
         exit the function because the course is already stored
      }
   }
   if(course is not found)
      add the course to the vector
}

void Student::deleteCourse(QString course_name) {
   while(there are more courses that hadn't been compared with) {
      if(the passed course name is the same as this course name) {
         remove this course from the vector and exit the function
      }
   }
}
```

```
QVector<Student> Student::all() {
    open a connection with the database
    create a vector to store all the students in
    make a query to fetch all the students from the database
    while(there another student to be fetched) {
        fetch all the attributes from the table and use them to create a student object

        make a query to fetch all the courses related to this student from the database
                while(there another course to be fetched) {
                        fetch all the attributes from the table and use them to create
a student object

                        make a query to fetch all the courses related to this student
from the database
                }
    }
    close the connection with the database
    return the vector of students
}

bool Student::isInDatabase(long long id) {
    open a connection with the database

    create a query to fetch the data of the student with the specified id
    if(a student with such an id exists) {
        close the connection with the database
        return true indicating that the student exists
    }
    close the connection with the database
    return false indicating that the student doesn't exist
}
bool Student::save(){
    get the id of the student
    create a query that will fetch the student with the id
    get the values of attributes of the object
    if(the student exists in the database){
        update the attributes of the student in the database

        while(there is another course in the vector){
            get the id of the course
            create a query that will fetch the row that links the student with the course
            if(this row doesn't exist){
                insert this row in the database
            }
        }
        return true indicating that the method finished and the student was already in
the database
    }
    insert this student into the database
    while(there is another course in the vector){
        create a query that will link the course to the student in the database
                    execute this query
    }
    return false indicating that the method finished and the student wasn't in the
database
}
```

```
void Student::delete1(){
   delete the student with the available id
         delete the courses related to this student
}


Student Student::find(long long id) {
         create a connection with the database
         create a query that will fetch the data of a student with the specified id
   execute this query
   fetch all the data from the database
   create a student object with all the data from the database
         create a query that will fetch all the course related to the student from the
database
   while(there is another course to be fetched) {
      add this course to the vector of courses
   }
         close the connection with the database
   return the student object
}
QVector<Student> Student::where(QString column, QString value){
   crate a query to fetch all the students that have the value of the specified with
the specified value
   create a vector to store the students in
   while (there is another student to be fetched) {
      add this student to the vector
   }
   return the vector
}
```

15.4. Staff Member

```
#include "staffmember.h"

#include <QString>

#include "course.h"


#include <QSqlDatabase>

#include <QSql>

#include <QSqlError>

#include <QDir>

#include <QFile>

#include <QDebug>

#include <QSqlQuery>

#include <QString>

#include <QStringList>

#include <QSqlError>

#include <globalDbObject.h>

static QString staff_table = "staff_members";

static QStringList staff_columns = {"first_name","last_name", "degree",
"birth_date",

                        "gendre", "address", "password", "picture", "degree",
"department"};

static QStringList staff_types = {"INTEGER PRIMARY KEY
AUTOINCREMENT", "TEXT", "TEXT", "TEXT", "TEXT",

                       "TEXT", "TEXT", "TEXT", "TEXT", "TEXT", "TEXT"};

StaffMember::StaffMember() : use the person constructor {

   set the degree property to empty string

   set the department property to empty string

}
```

```
StaffMember::StaffMember(QString first_name, QString last_name, QString
gendre, QString picture, QString birth_date, QString address, QString college_id,
QString password, QString degree, QString department)
    : use the person constructor {
    set the degree property to the specified value
    set the department property to the specified value
}

void StaffMember::setDegree(QString degree) {
    set degree attribute to the passed value
}

QString StaffMember::getDegree() {
    return the value of the degree attribute
}

void StaffMember::setDepartment(QString department) {
    set department attribute to the passed value
}

QString StaffMember::getDepartment() {
    return the value of the department attribute
}



QVector<Course> StaffMember::getCourses() {
    return the value of the courses attribute
}
```

```
void StaffMember::addCourse(QString course_name) {
    while(there are more courses that hadn't been compared with) {
        if(the passed course name is the same as this course name) {
            exit the function because the course is already stored
        }
    }
    if(course is not found)
        add the course to the vector
}

void StaffMember::deleteCourse(QString course_name) {
    while(there are more courses that hadn't been compared with) {
        if(the passed course name is the same as this course name) {
            remove this course from the vector and exit the function
        }
    }
}
```

```
QVector<StaffMember> StaffMember::all() {
    open a connection with the database
    create a vector to store all the staff members in
    make a query to fetch all the staff members from the database
    while(there another staff member to be fetched) {
        fetch all the attributes from the table and use them to create a staff member object

        make a query to fetch all the courses related to this staff member from the database
            while(there another course to be fetched) {
                fetch all the attributes of this course from the table and add it to the vector
            }
    }
    close the connection with the database
    return the vector of staff members
}
```

```
bool StaffMember::isInDatabase(long long id) {
   open a connection with the database


   create a query to fetch the data of the staff member with the specified id
   if(a staff member with such an id exists) {
      close the connection with the database
      return true indicating that the staff member exists
   }
   close the connection with the database
   return false indicating that the staff member doesn't exist
}


StaffMember StaffMember::find(long long id) {
   create a connection with the database
         create a query that will fetch the data of a staff member with the specified
id
   execute this query
   fetch all the data from the database
   create a staff member object with all the data from the database
         create a query that will fetch all the course related to the staff member
from the database
   while(there is another course to be fetched) {
      add this course to the vector of courses
   }
         close the connection with the database
   return the staff member object
}
```

```
bool StaffMember::save(){
    get the id of the staff member
    create a query that will fetch the staff member with the id
    get the values of attributes of the object
    if(the staff member exists in the database){
        update the attributes of the staff member in the database

        while(there is another course in the vector){
            get the id of the course
            create a query that will fetch the row that links the staff member with the
course
            if(this row doesn't exist){
                insert this row in the database
            }
        }
        return true indicating that the method finished and the staff member was
already in the database
    }
    insert this staff member into the database
    while(there is another course in the vector){
        create a query that will link the course to the staff member in the database
                        execute this query
    }
    return false indicating that the method finished and the staff member wasn't in
the database
}
```

```
QVector<StaffMember> StaffMember::where(QString column, QString value){
   crate a query to fetch all the staff members that have the value of the specified
with the specified value
   create a vector to store the staff members in
   while (there is another staff member to be fetched) {
      add this staff member to the vector
   }
   return the vector
}

void StaffMember::delete1(){
   delete the staff member with the available id
         delete the courses related to this staff member
}
```

16. Estimated Project Cost

According to COCOMO-II Early Cost Model:

Pm = A x Size^{B} x M

A = 2.94

B = 1.2

For M Software errors must be minimum so RCPX > 1, RCPX = 1.2

Software is made from scratch so RUSE > 1, RUSE = 3

Software hasn’t special platform so PDIF = 1

Personnel experience is low so PREX > 1, PREX = 3

Personnel capability is high so PERS = 1

Project must be delivered before 1/1/2`020 so SCED > 1, SCED = 4

Team support facilities is low, So FCIL = 3

M = RCPX x PDIF x RUSE x PREX x PERS x SCED x FCIL

M = 1.2 x 3 x 1 x 3 x 1 x 4 x 3 = 129.6

Size: LOC = 2500 = 2.5 kLOC

pm = 2.941 x Size$^{1.2}$ x 129.6

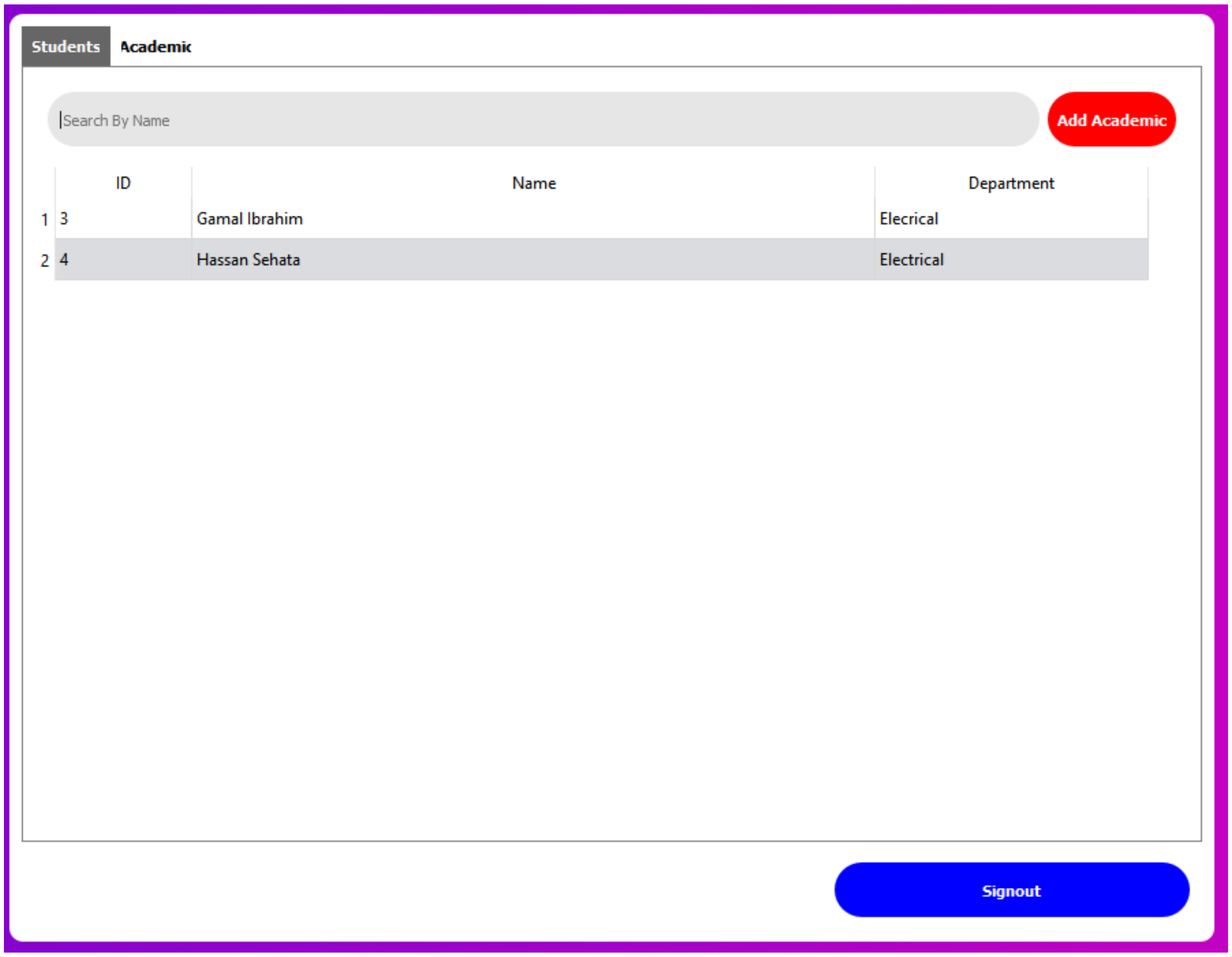
Labour rate = 100 $/pm

Cost = Labour rate x pm = 114.55 k$

17. Testing

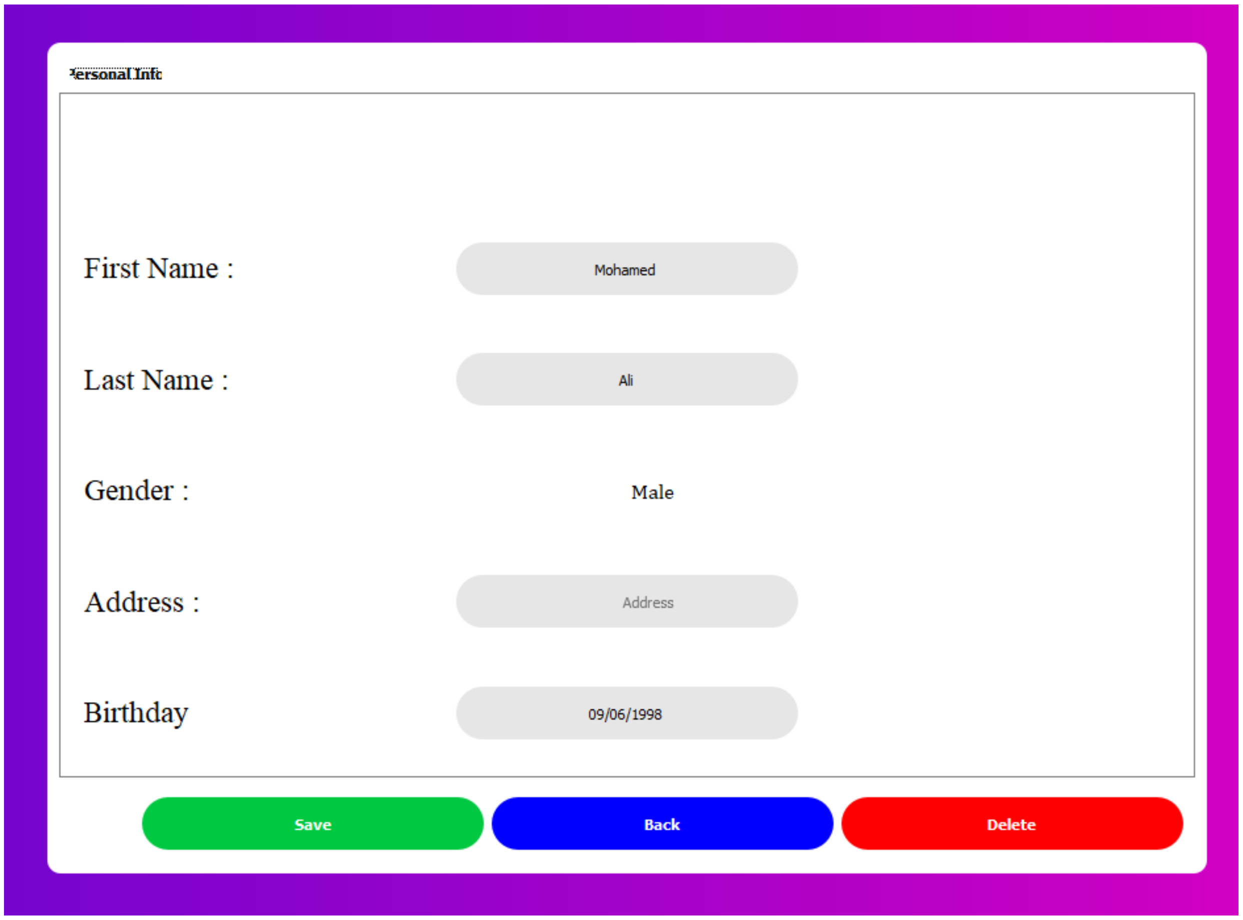
Logging in as Admin:

Viewing Academic Staff members' list:



	ID	Name	Department
1	3	Gamal Ibrahim	Elecrical
2	4	Hassan Sehata	Electrical

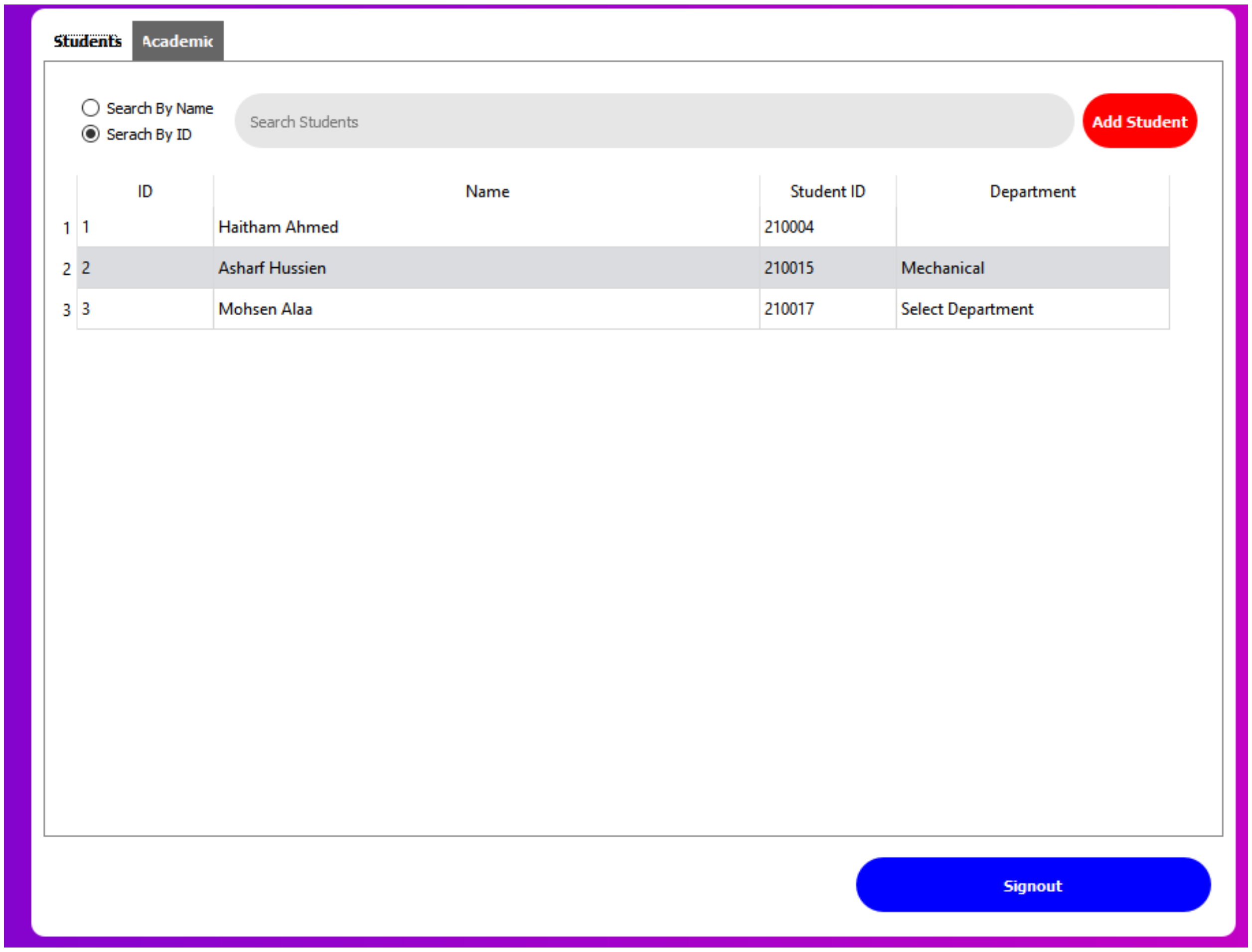
(selecting a student/staff member from students'/academic staff members' list)



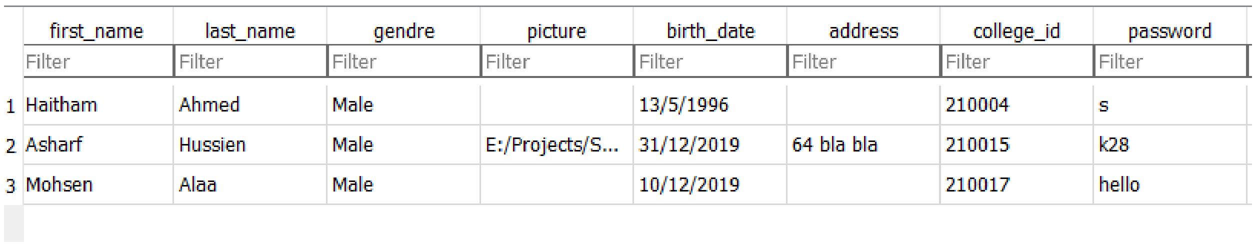
Personal Info

First Name :	Mohamed
Last Name :	Ali
Gender :	Male
Address :	Address
Birthday	09/06/1998

After deleting the student:

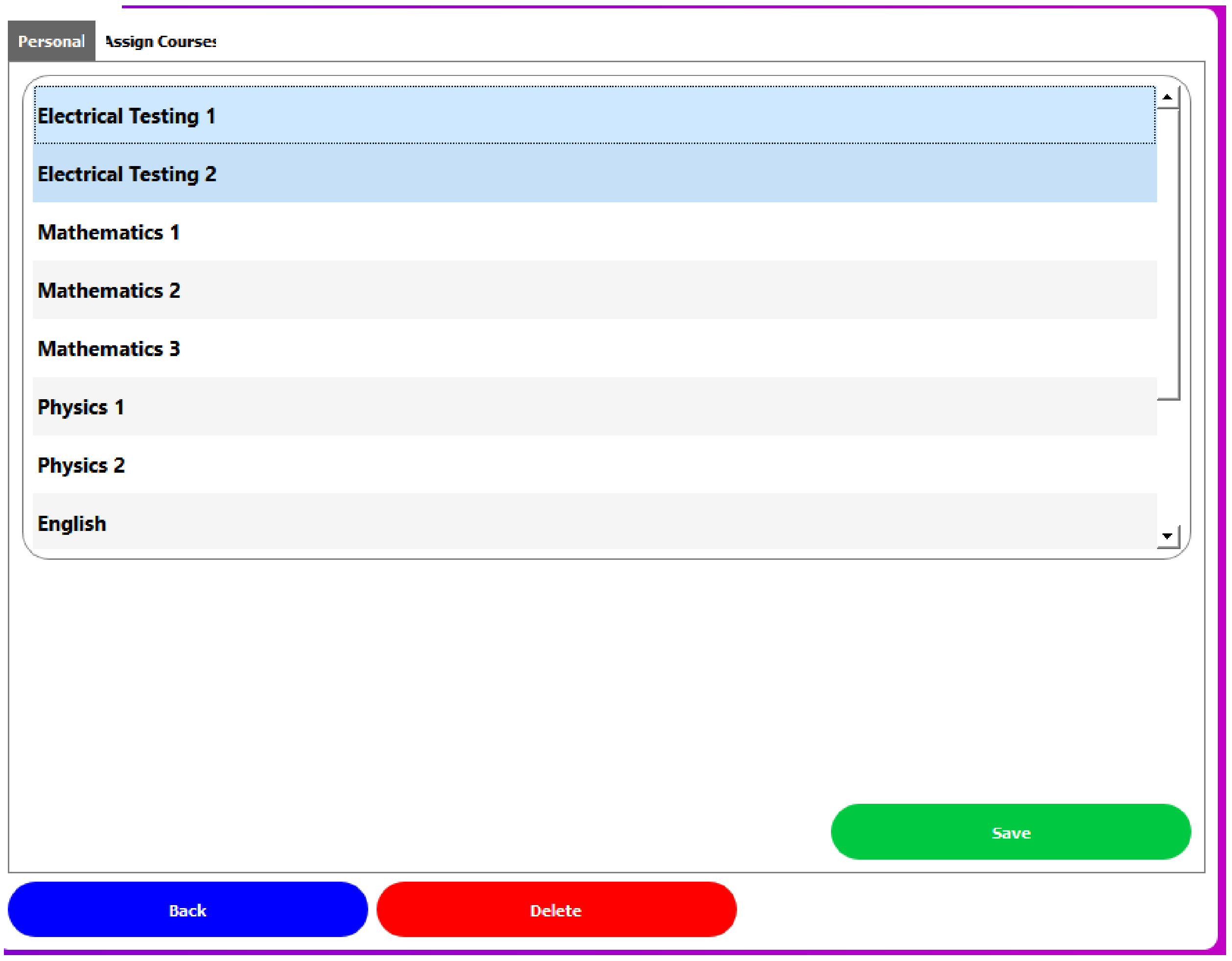


	ID	Name	Student ID	Department
1	1	Haitham Ahmed	210004	
2	2	Asharf Hussien	210015	Mechanical
3	3	Mohsen Alaa	210017	Select Department



	first_name	last_name	gendre	picture	birth_date	address	college_id	password
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Haitham	Ahmed	Male		13/5/1996		210004	s
2	Asharf	Hussien	Male	E:/Projects/S...	31/12/2019	64 bla bla	210015	k28
3	Mohsen	Alaa	Male		10/12/2019		210017	hello

Admin assign courses to an Academic Staff member:



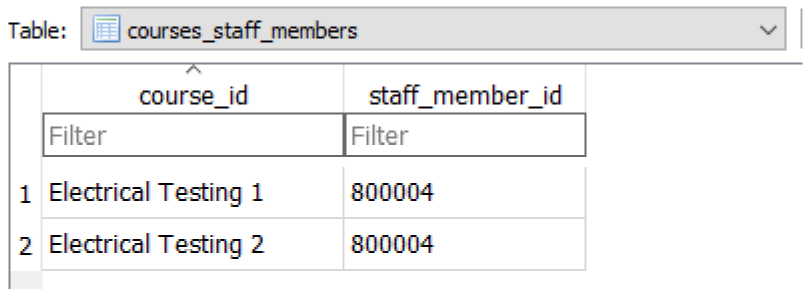
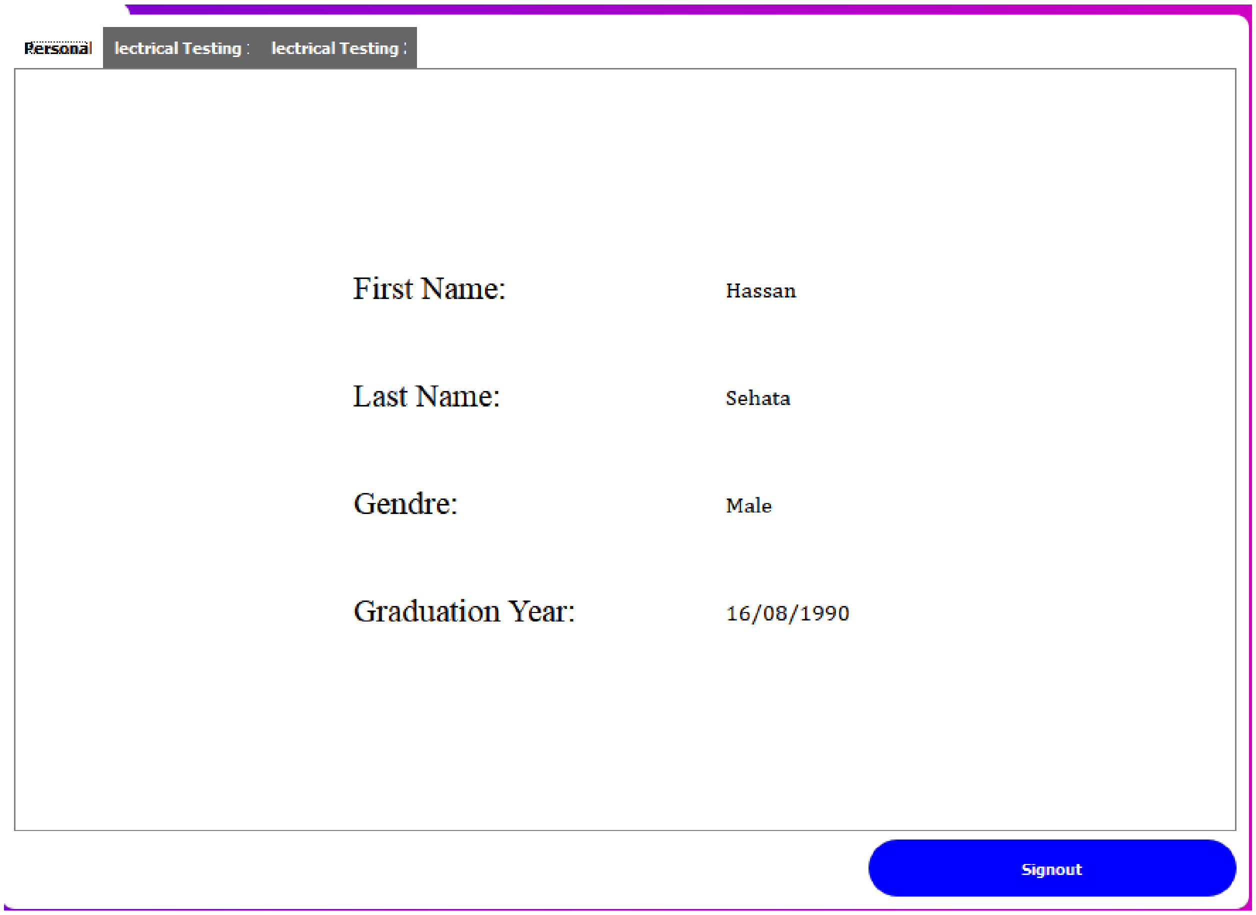


Table: courses_staff_members

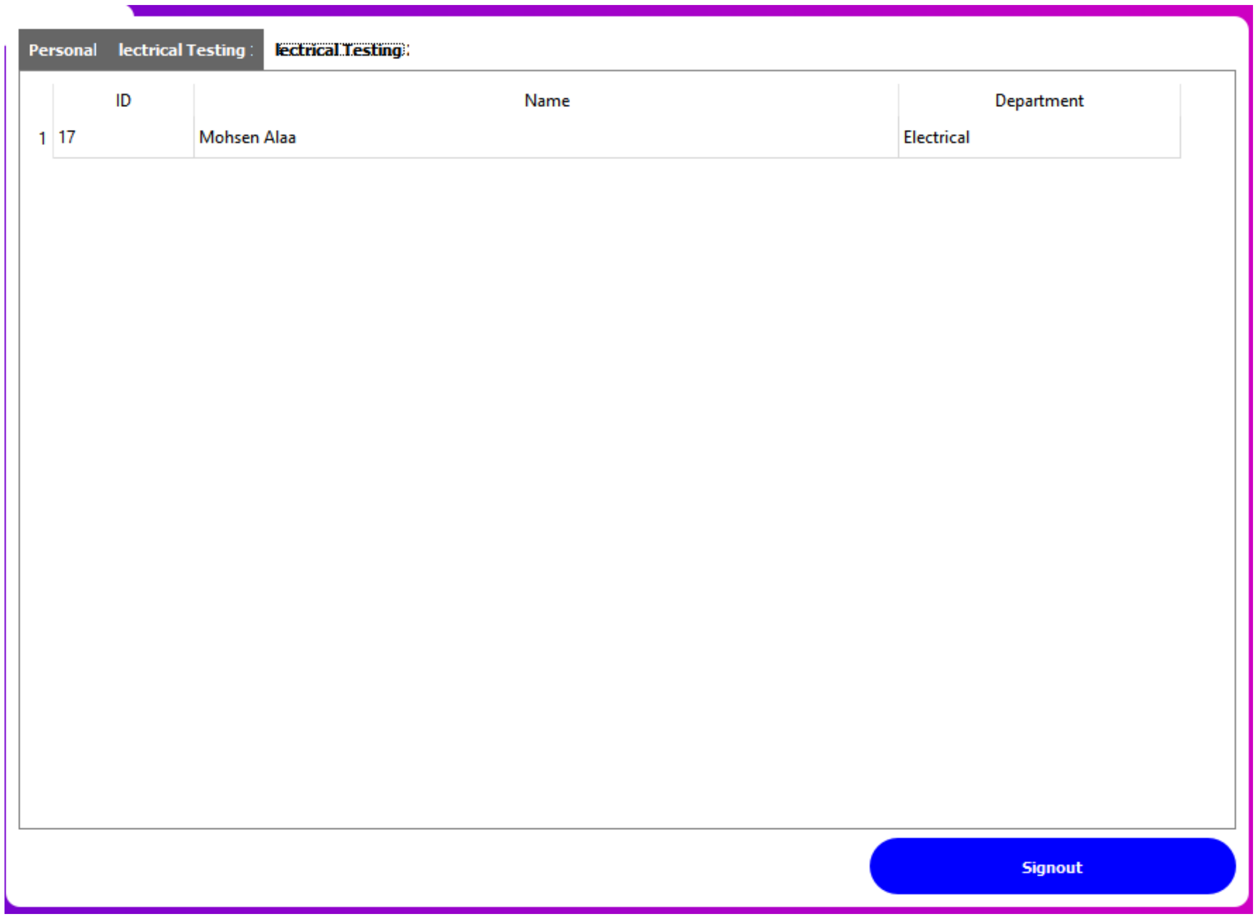
	course_id	staff_member_id
1	Electrical Testing 1	800004
2	Electrical Testing 2	800004

Logging in as staff member:



First Name:	Hassan
Last Name:	Sehata
Gendre:	Male
Graduation Year:	16/08/1990

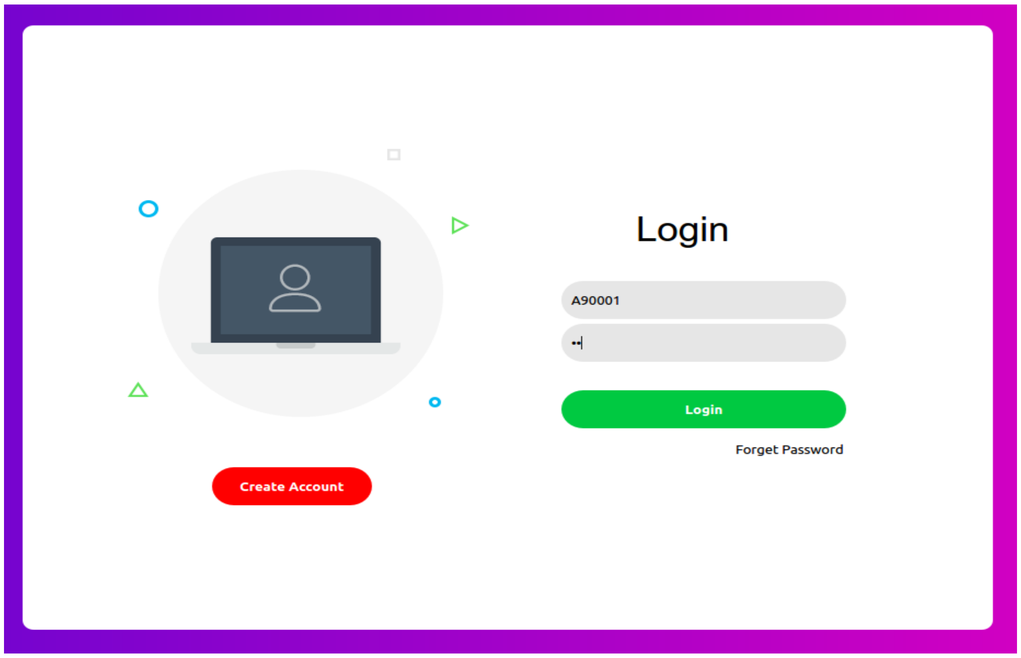
Viewing Courses:



	ID	Name	Department
1	17	Mohsen Alaa	Electrical

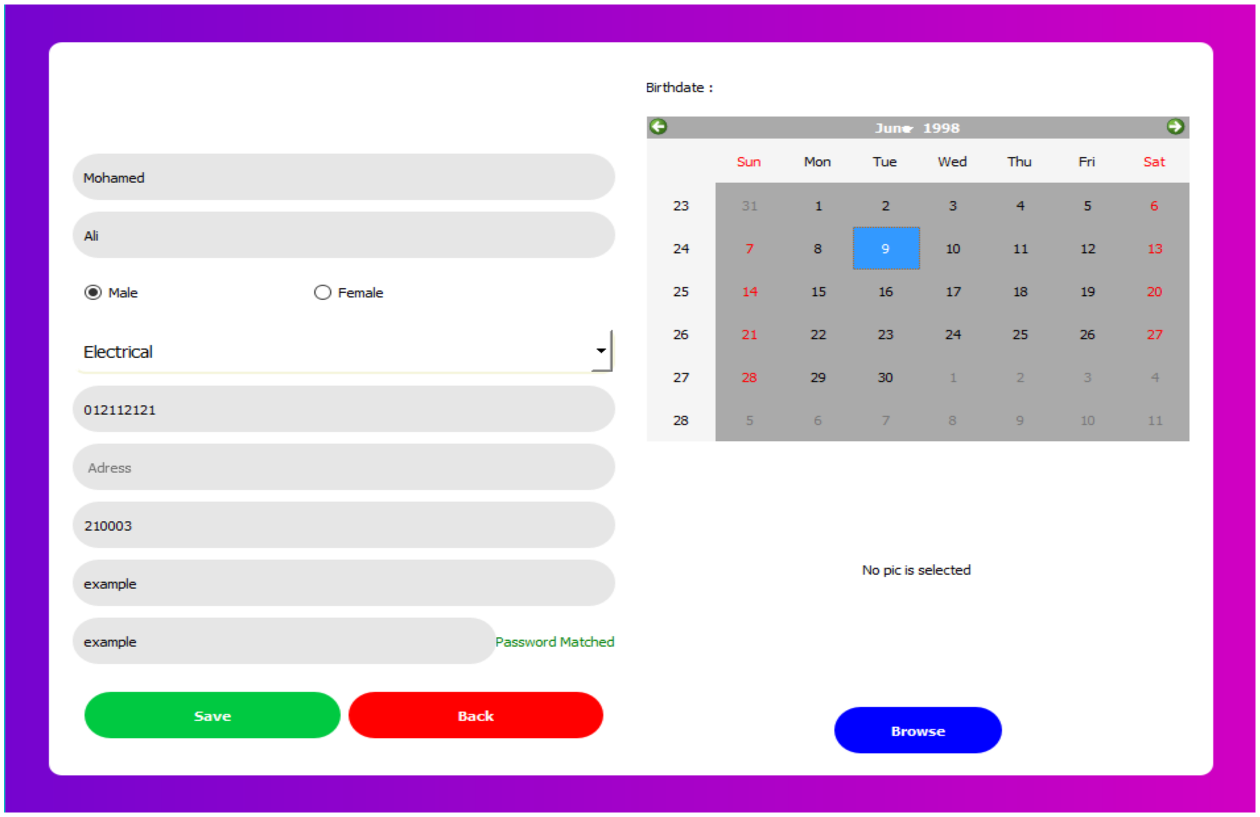
18. User Guide

Login:



Registration:

A Student can register himself if he already got the ID



Birthdate :

June 1998

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
23	31	1	2	3	4	5	6
24	7	8	9	10	11	12	13
25	14	15	16	17	18	19	20
26	21	22	23	24	25	26	27
27	28	29	30	1	2	3	4
28	5	6	7	8	9	10	11

Mohamed

Ali

● Male ○ Female

Electrical

012112121

Adress

210003

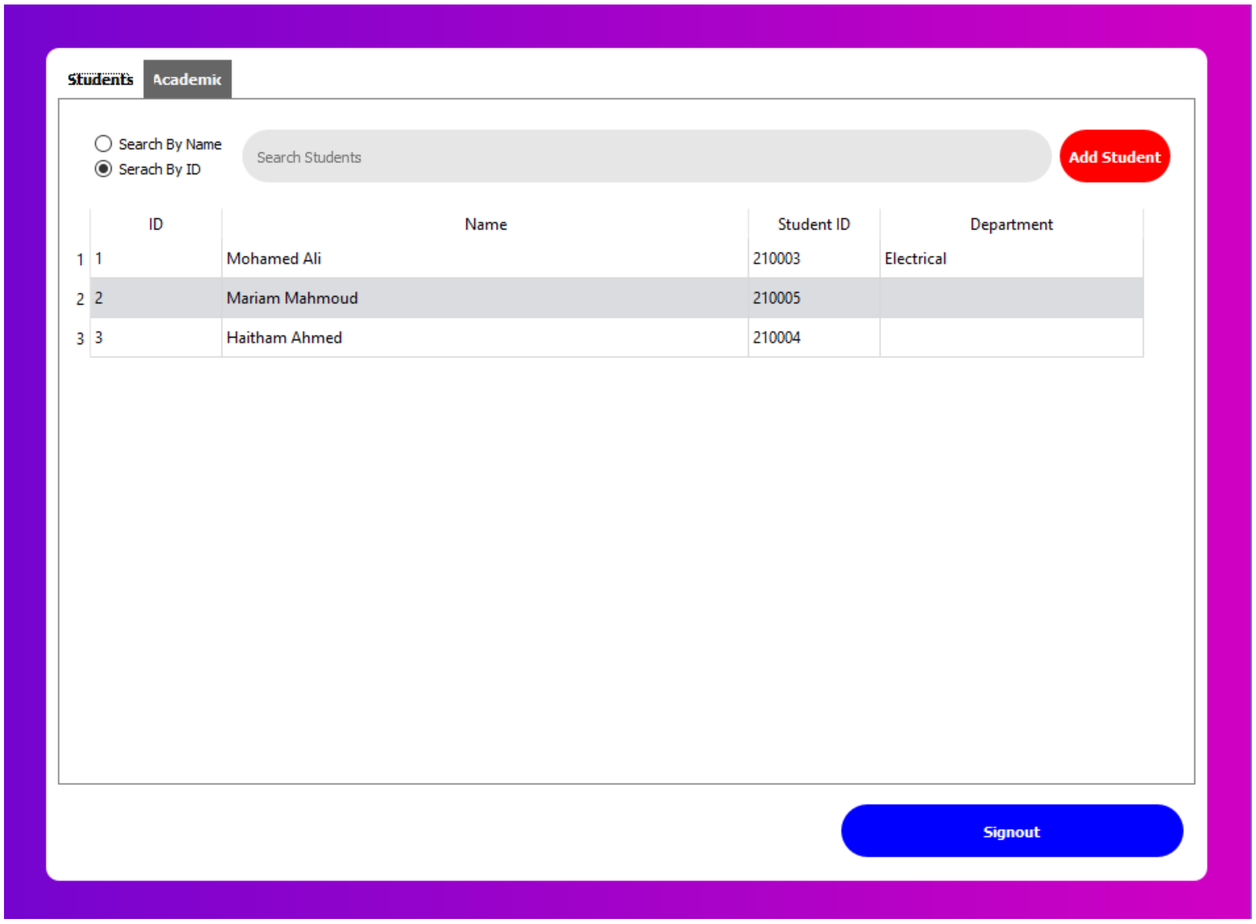
example

example Password Matched

No pic is selected

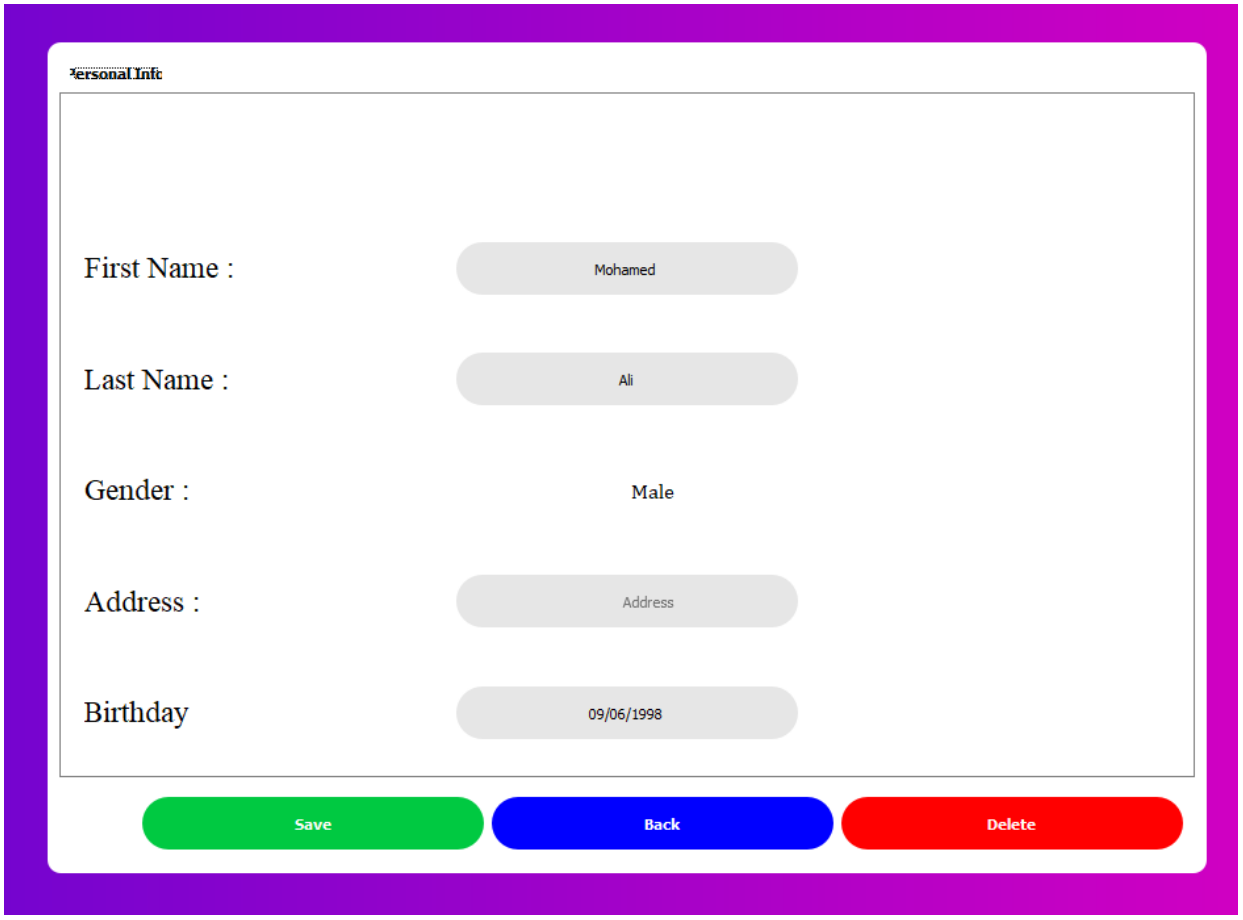
Admin Account:

1. Viewing Students' list.
2. Selecting and editing or deleting the selected student data.
3. Adding new Student.
4. Viewing Academic Staff Members' list.
5. Selecting and editing or deleting the selected member data.
6. Adding Academic Staff Member.

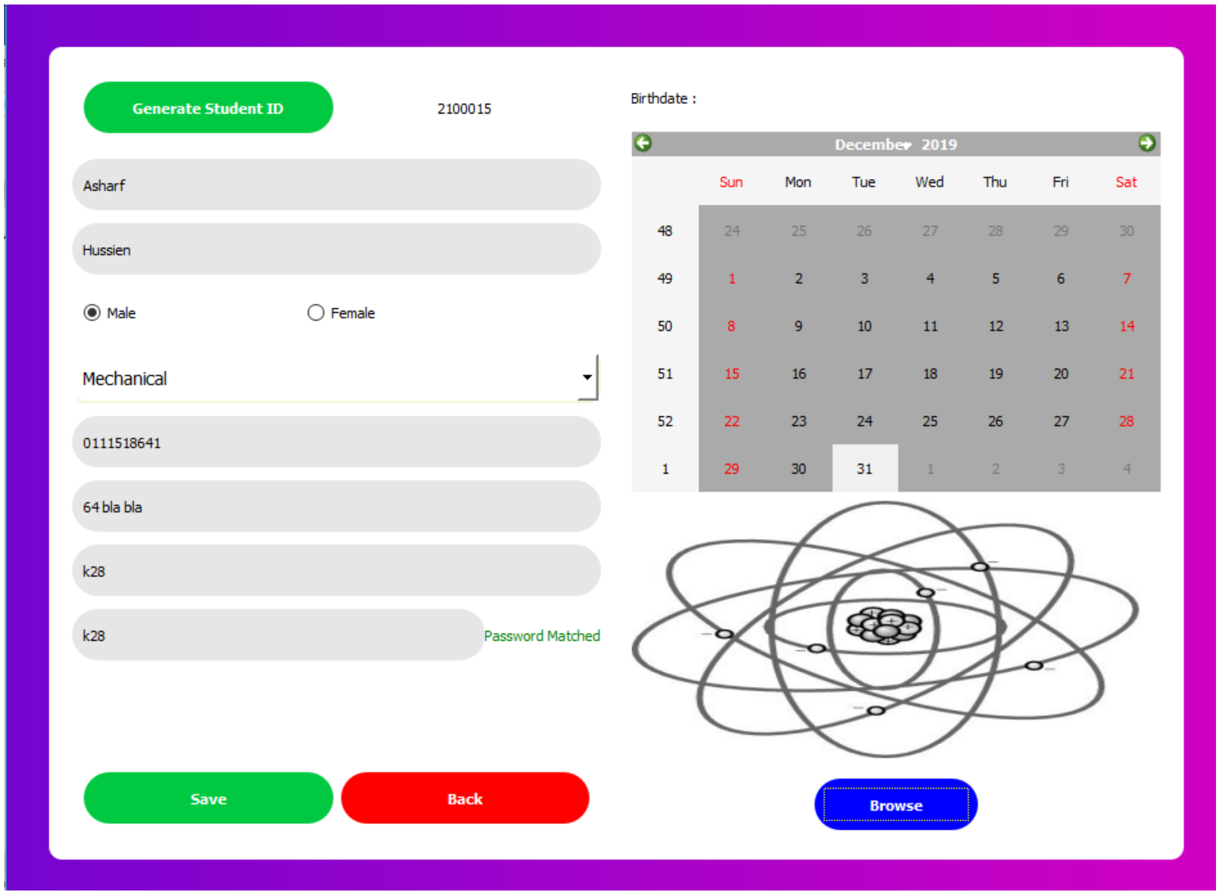


	ID	Name	Student ID	Department
1	1	Mohamed Ali	210003	Electrical
2	2	Mariam Mahmoud	210005	
3	3	Haitham Ahmed	210004	

Editing New Student:



First Name :	Mohamed
Last Name :	Ali
Gender :	Male
Address :	Address
Birthday	09/06/1998


Generate Student ID

2100015

Birthdate :

December 2019

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
48	24	25	26	27	28	29	30
49	1	2	3	4	5	6	7
50	8	9	10	11	12	13	14
51	15	16	17	18	19	20	21
52	22	23	24	25	26	27	28
1	29	30	31	1	2	3	4

Asharf

Hussien

(● Male) ○ Female

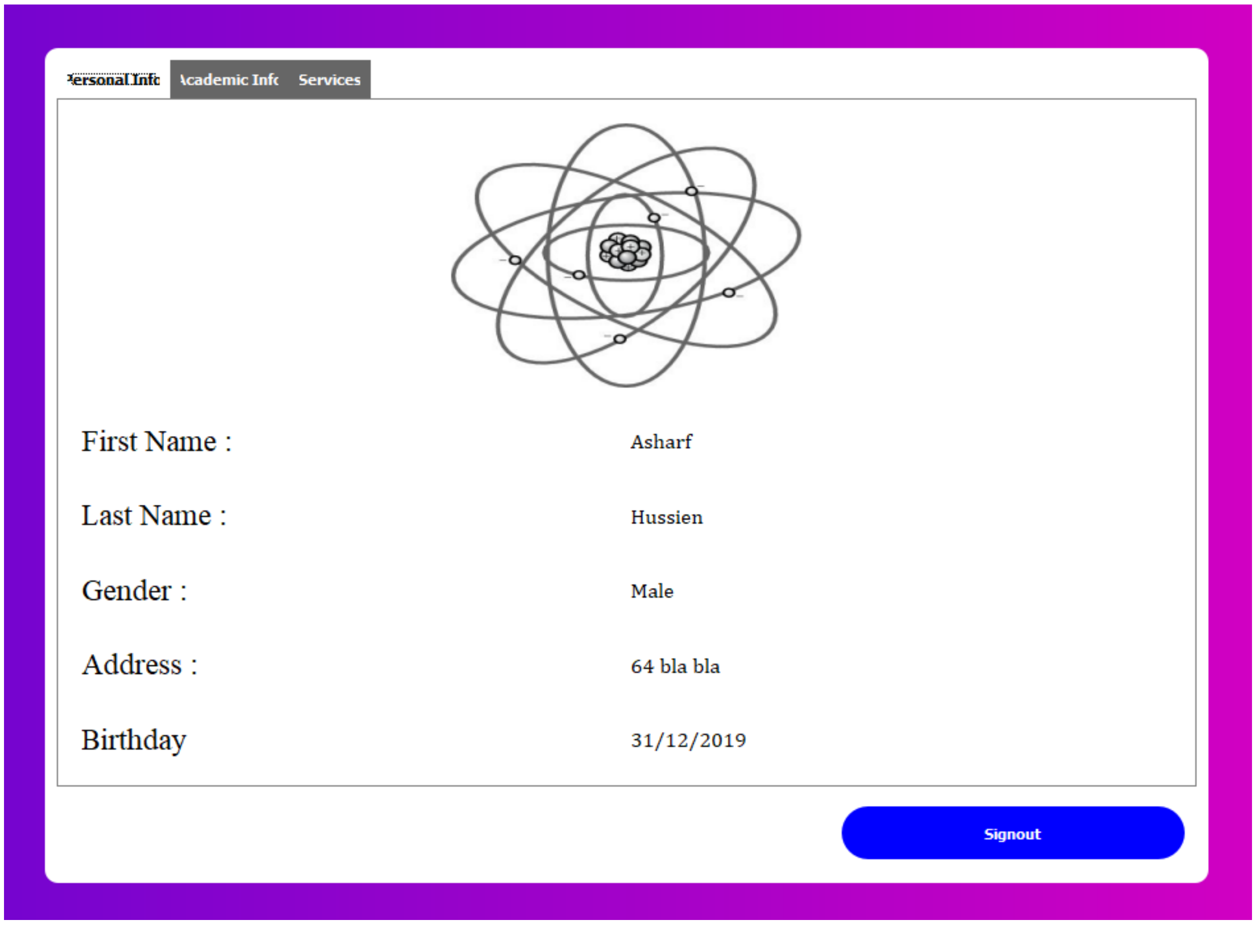
Mechanical

0111518641

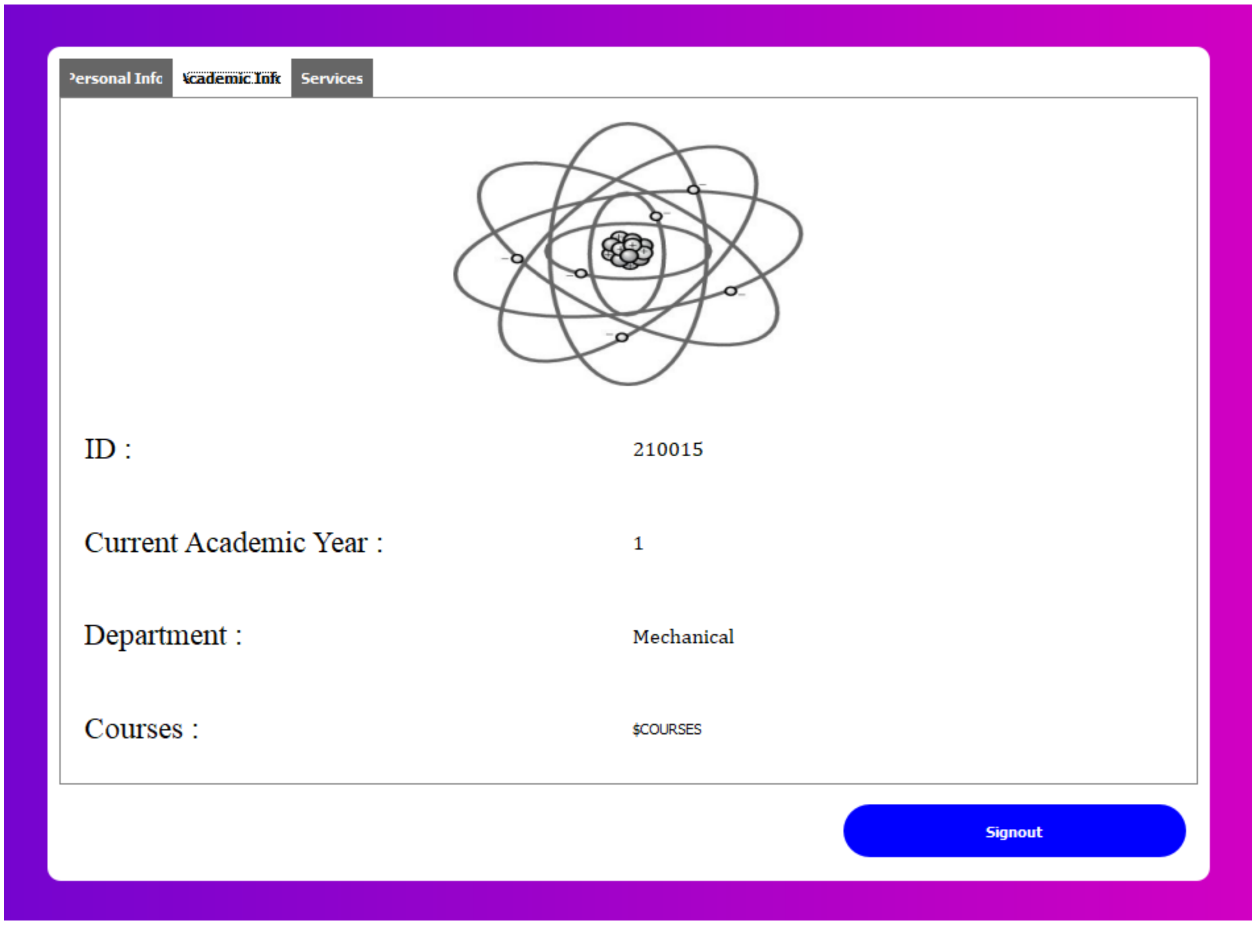
64 bla bla

k28

k28 Password Matched

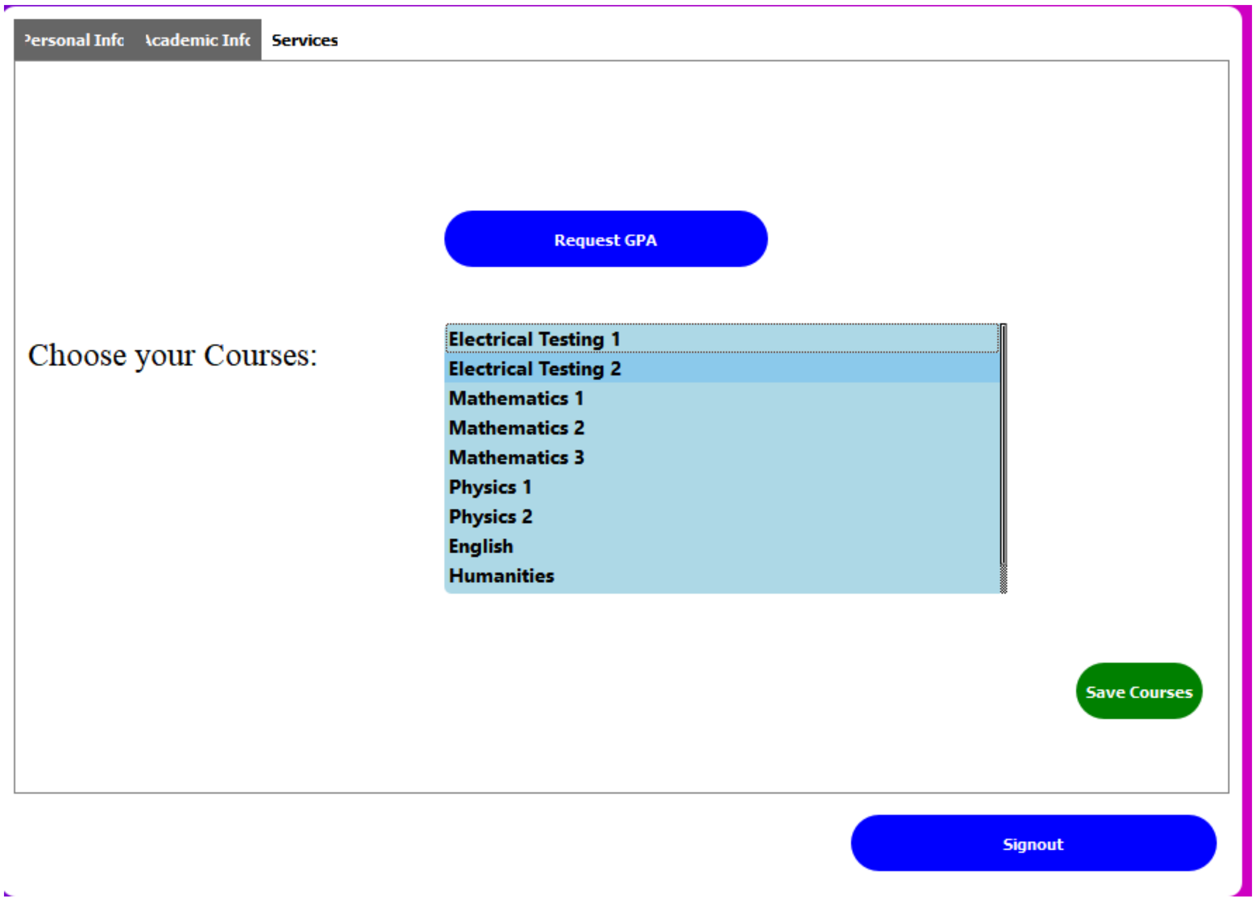


First Name :	Asharf
Last Name :	Hussien
Gender :	Male
Address :	64 bla bla
Birthday	31/12/2019



ID :	210015
Current Academic Year :	1
Department :	Mechanical
Courses :	$COURSES

Selecting Courses



Choose your Courses:

Electrical Testing 1
Electrical Testing 2
Mathematics 1
Mathematics 2
Mathematics 3
Physics 1
Physics 2
English
Humanities

Appendices

Google Drive Link for the Required Files:

https://drive.google.com/open?id=1krWatkNYrSNFHwwVYxFA4OzJ1Ctw5bSz

GitHub link for source code:

https://github.com/3rd-year-CSE-20/SIS_GUI