# Advanced programming assignment

# Student name: Monika Baricza

# Student number: 10389618

# Teacher’s name: Paul Laird

# 25/06/19

# Table of contents

[1 Introduction 3](#_Toc12441884)

[1.1 Requirements 3](#_Toc12441885)

[2 UML diagrams 3](#_Toc12441886)

[2.1 Use Case Diagram 3](#_Toc12441887)

[2.2 Activity Diagram 4](#_Toc12441888)

[2.3 Sequence diagram for deleting teacher 6](#_Toc12441889)

[2.4 Class diagram 7](#_Toc12441890)

[3 Matching the requirements 8](#_Toc12441891)

[3.1 Tiered architecture 8](#_Toc12441892)

[3.1.1 Presentation tier 8](#_Toc12441893)

[3.1.2 Business tier 11](#_Toc12441894)

[3.1.3 Data tier 13](#_Toc12441895)

[4 Final thoughts 13](#_Toc12441896)

# Table of figures

[Figure 1 Use case diagram 4](#_Toc12441916)

[Figure 2 Activity diagram 5](#_Toc12441917)

[Figure 3 Sequence diagram 6](#_Toc12441918)

[Figure 4 Class diagram 7](#_Toc12441919)

[Figure 5 Login screen 8](#_Toc12441920)

[Figure 6 Main screen 9](#_Toc12441921)

[Figure 7 Enrol students 9](#_Toc12441922)

[Figure 8 Find in database 10](#_Toc12441923)

[Figure 9 Edit Student 10](#_Toc12441924)

[Figure 10 Saving as XML 11](#_Toc12441925)

[Figure 11 Inheritance 12](#_Toc12441926)

## Introduction

The aim of the assignment was to develop a student management system for DBS. The project’s learning outcome is to get more familiar with the windows form applications and to learn how to manage databases that are connected to the application.

### Requirements

Requirement 1: the application must be built using a tiered architecture

Requirement 2: the application must contain a login screen and only a file menu should be visible

Requirement 3: a working login system that validates the username and password

Requirement 4: once the user is logged in the menu should change according to the user type

Requirement 5: a grid should be displayed with student data on the welcome screen

Requirement 6: ability to enrol a student

Requirement 7: ability to update students

Requirement 8: the system should provide for extracting data in XML format

## UML diagrams

### Use Case Diagram

The use case diagram can be found under Figure 1 below. As you can see, this diagram shows that the user can sign in as a regular user or as an admin. If the user is an admin it means that they have a much wider access to managing data in the application. They can add, edit, delete and view students, teachers or courses while a regular user can only view them. Admins can also enrol students to courses and set the fee for that student for that particular course. The user model itself can be improved further later on; at the moment the admin doesn’t manage users. There is a search engine built for the application which is used to find students, courses or teachers in the database.

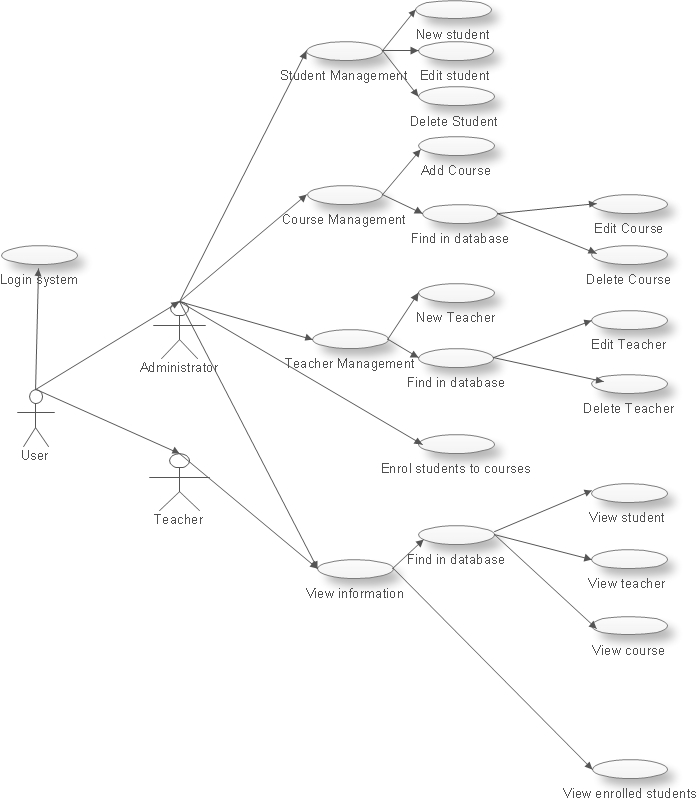


Figure Use case diagram

### Activity Diagram

The activity diagram shows what an admin can do in the application. After validating the login credentials, they can add a student, add a teacher, or add a course. After adding data to the database, they can manage the data by editing, deleting students, teachers or courses. Data can be found by entering details into the search engine which pops up after “edit”, “delete” or “view” has been clicked. The course table includes a foreign key referring to the teacher table, so if the admin deletes a teacher, it will automatically delete the course related to the teacher. The admin can also enrol a student to a course by selecting the student from the dropdown list and then selecting a course from the dropdown list. On this screen they add the fee for the course for the student as well. They can also view student and course relations under Course Management -> Course Management Info tab.

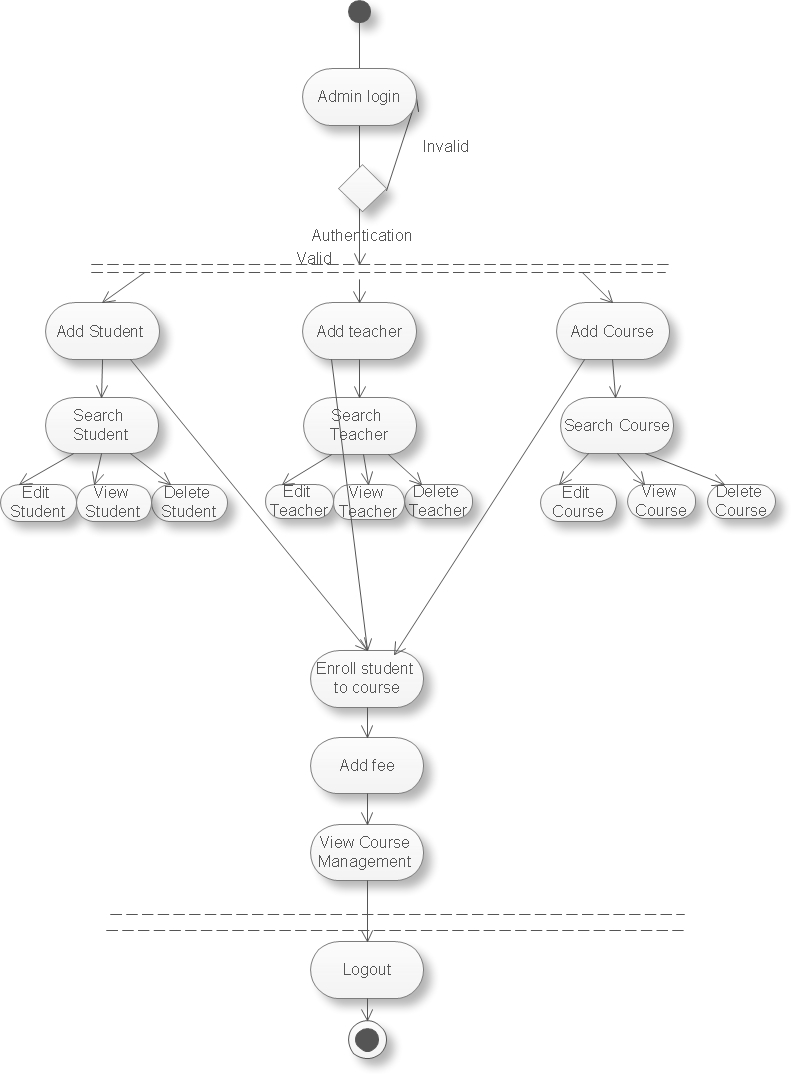


Figure Activity diagram

### Sequence diagram for deleting teacher

This is an example of how the application works. If the admin signs in with the right credentials, the main screen appears then they click Teacher -> Delete Teacher (assuming there is already at least one teacher in the database). The Find in Database screen pops up, where the admin can search by ID, first name or last name. While the ID has to be an exact number, it is enough to enter a few letters of the first name or the last name. Once they have found the teacher that they are looking for, they click on the right row in the datagrid which displays the search results. Then the delete teacher screen pops up, where if they hit “Delete”, the teacher and all associated records are going to be deleted from the database. If they decide not to go ahead with the deletion, the application returns to the main screen.

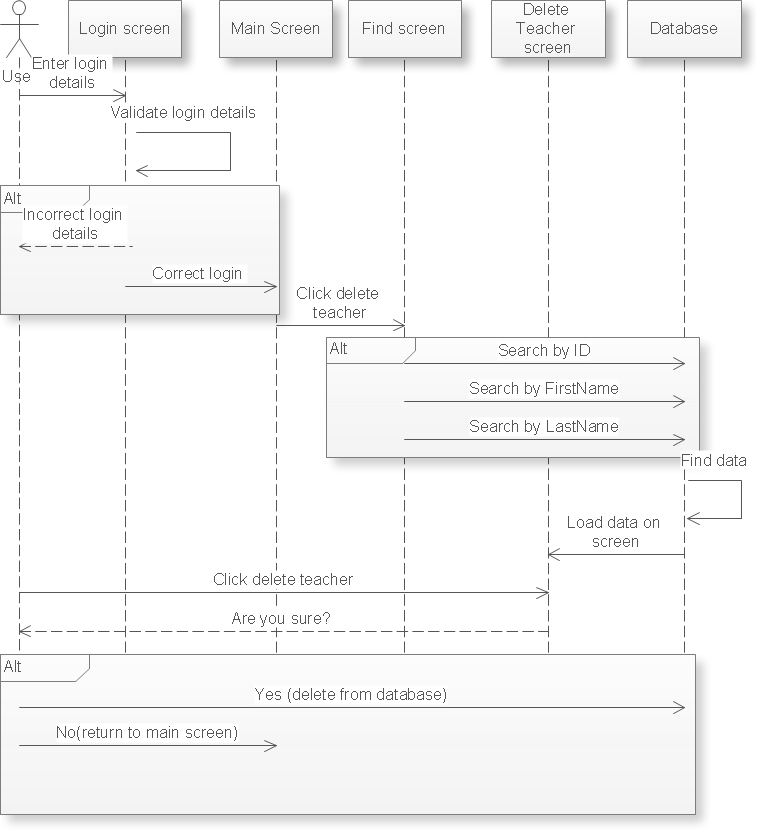


Figure Sequence diagram

### Class diagram

The following class diagram represents the database structure. The Teacher and the course have a one to many relationship as a teacher can teach many courses but a course can only have one teacher. The course and the student tables have a many to many relationship as a student can attend many courses and a course can be attended by many students. This has been resolved by adding the course management table, which brakes the many to many relationship down to two one to many relationships. As it was mentioned before the user table is not being managed by any of the users at the moment, this is subject to further improvement (a role could be added to the user table to define if the user is a teacher or an admin).

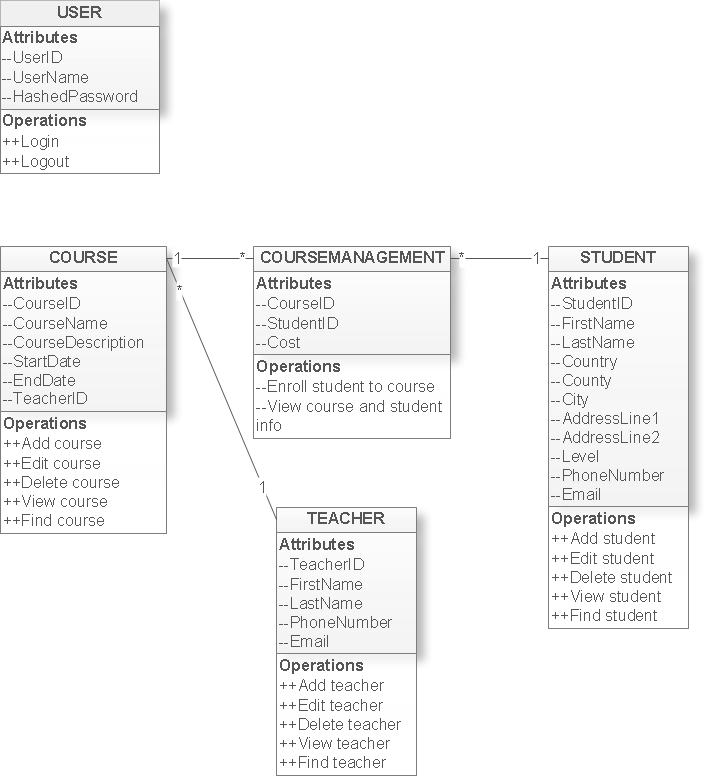


Figure Class diagram

## Matching the requirements

### Tiered architecture

I feel tiered architecture has been achieved by dividing the application into 3 tiers: presentation tier, business tier and data tier.

#### Presentation tier

I feel the user interface is quite clear and it is easy to use. I aimed to use the same simple colour scheme throughout the application.

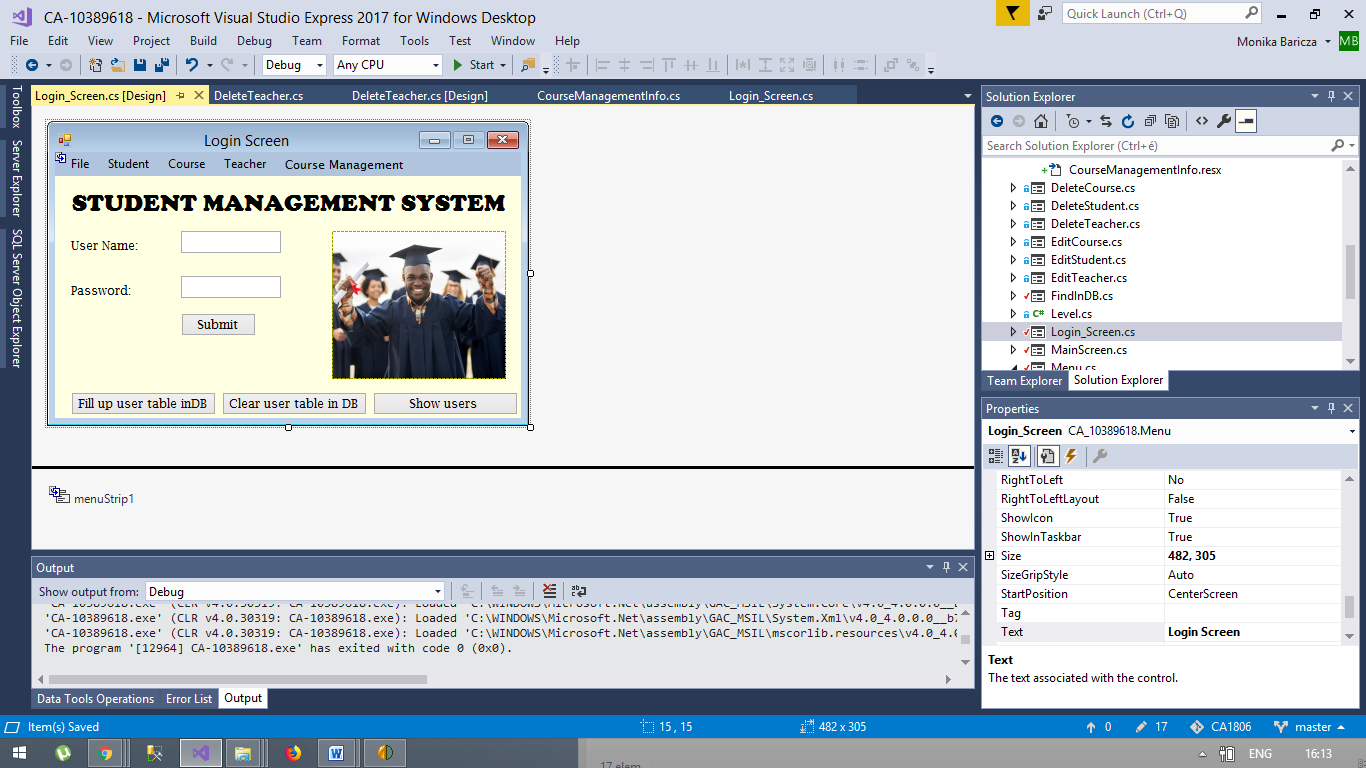


Figure Login screen

On the login screen there are three buttons. As the database cannot be zipped together with the rest of the project, it will have to be created separately which will result in an empty database. With the “Fill up user table in DB” button the user table is going to be prefilled with a set of users. With the “Show users” button, the login credentials for one admin and two regular users will be displayed in a message box. I didn’t want to create a sign up screen for the application as it is an internal application.

The passwords are securely hashed as per the requirement. In the Login\_Screen class, which is using System.Security.Cryptography, there is a method called hashPassword(string password) which adds salt to the password. There is a method called CheckSecurePassword(string password, string savedHashedPassword) which returns a Boolean if the password entered matches the hashed password stored in the database. If the user is not an admin, the menu will be reduced to views. So this fulfils requirement two, three and four.

The following screenshot is taken of the main screen, as per requirement 5 to have student data displayed in a grid on the welcome screen.

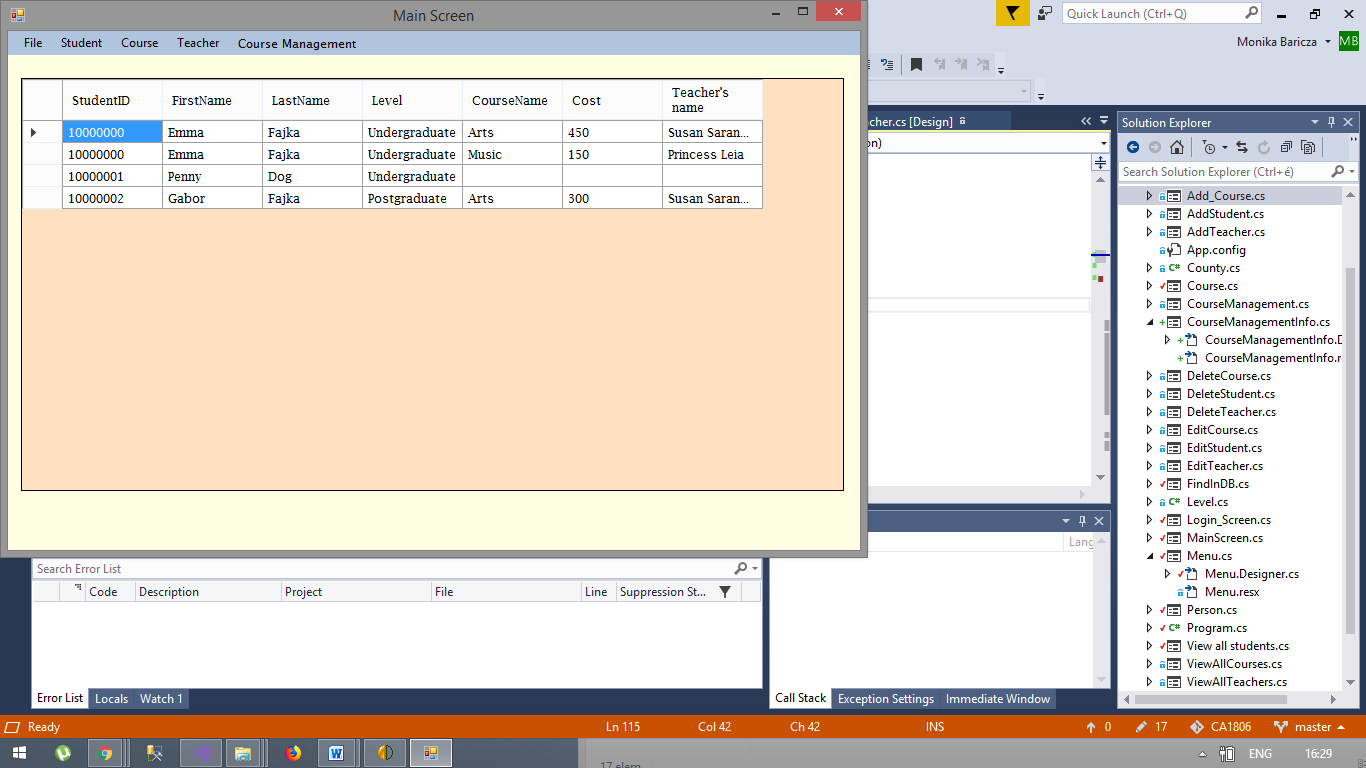


Figure Main screen

Requirement 6 was to be able to enrol a student. Once an admin is logged in, they can click on Course Management -> Enroll Student. The following screen will appear:

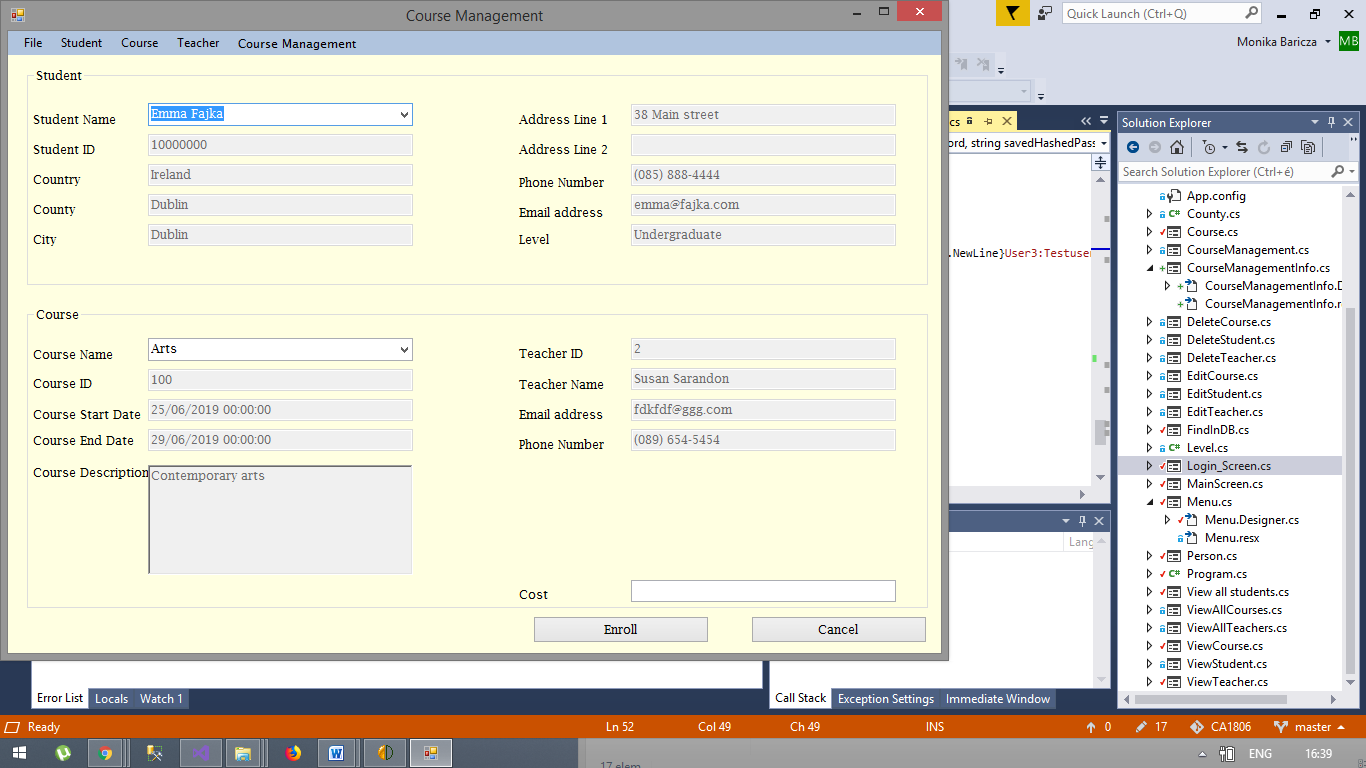


Figure Enrol students

Here the admin can choose from the students and courses in the database and can associate them with each other. A cost of the course can be added here as well.

Requirement 7 was to be able to update students. This can be achieved by clicking on Student->Edit Student. The following search engine will appear:

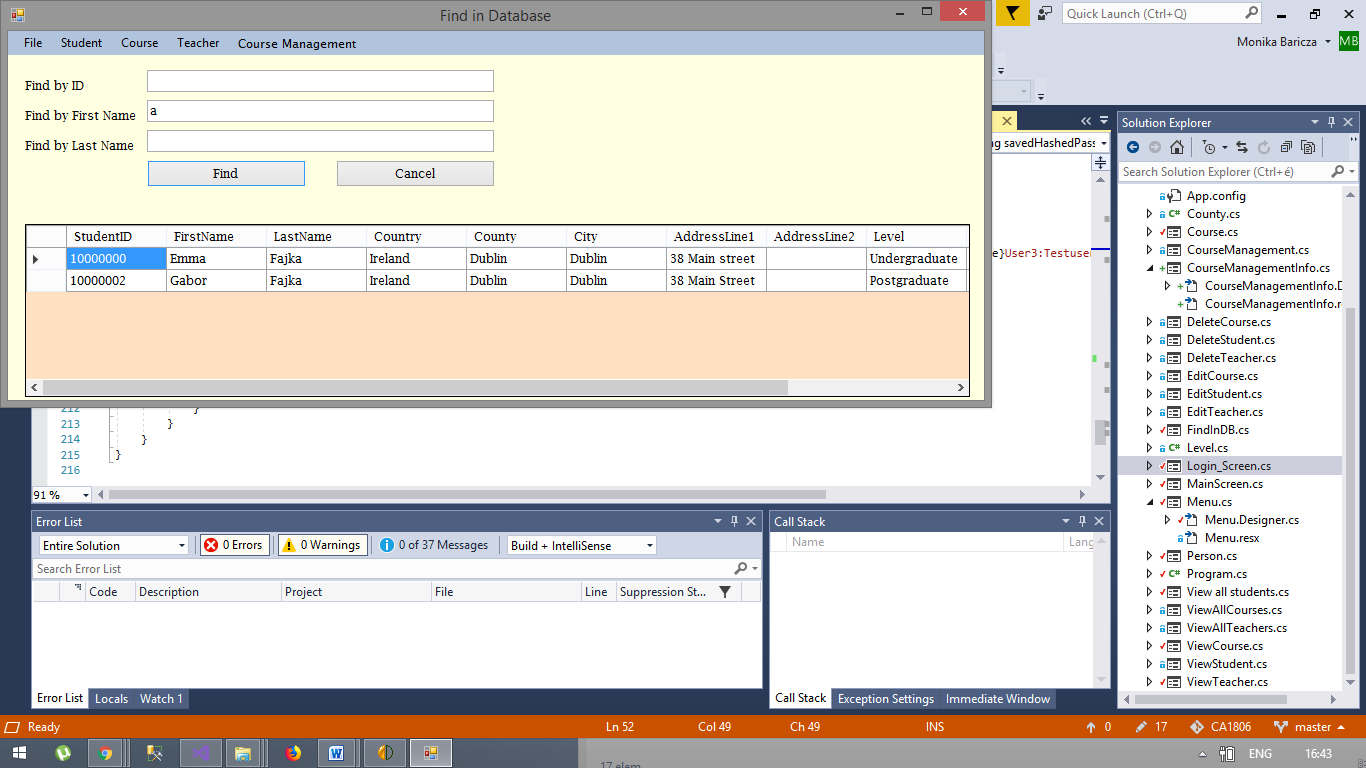


Figure Find in database

Students can be searched by ID (exact number), or any letter of the first name and last name. By clicking the right row in the datagrid which displays the search results a new “Edit student “ window will appear prefilled with the selected student’s data:

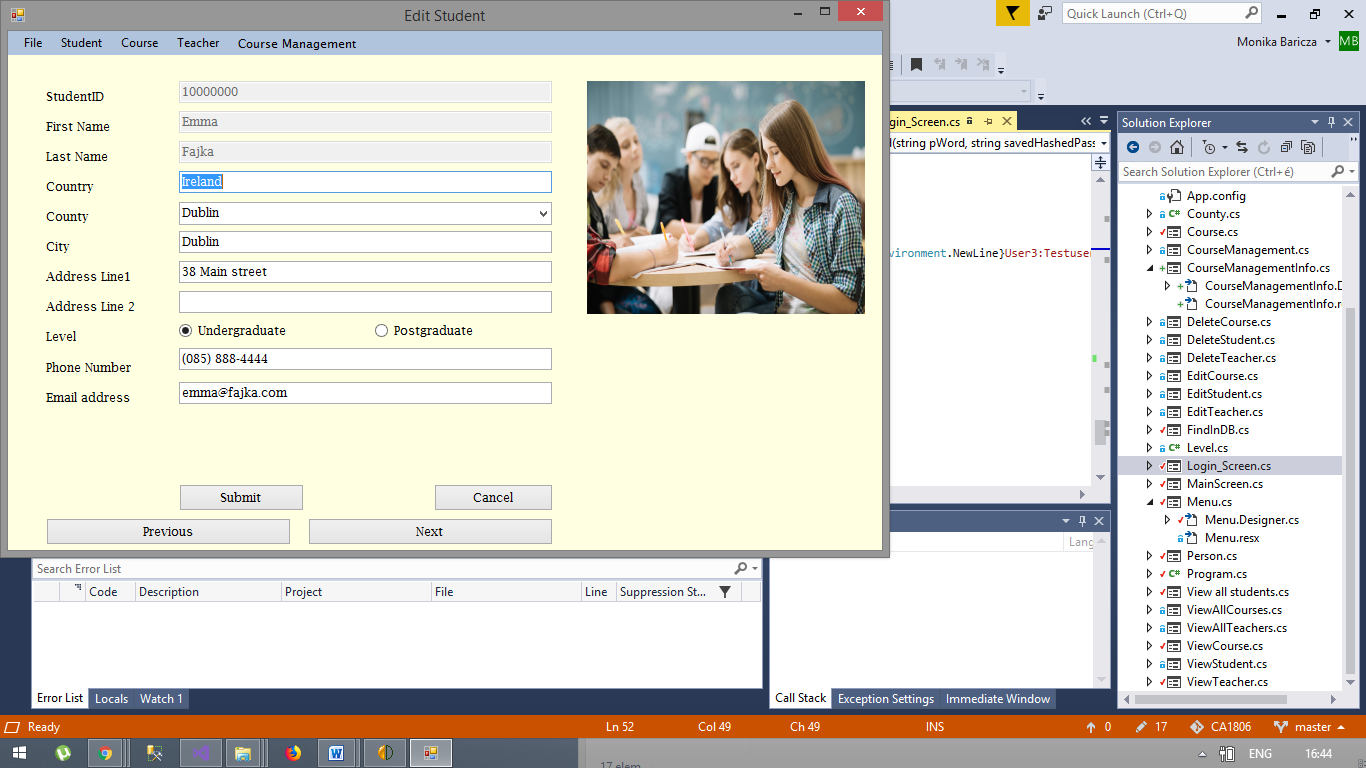


Figure Edit Student

Also we can navigate between the database records on any of these screens (delete, view or edit) with the previous and next buttons.

Requirement 8 was to be able to extract data to an xml format. Any of the search results or database histories can be extracted to XML by clicking the File->Save as button.

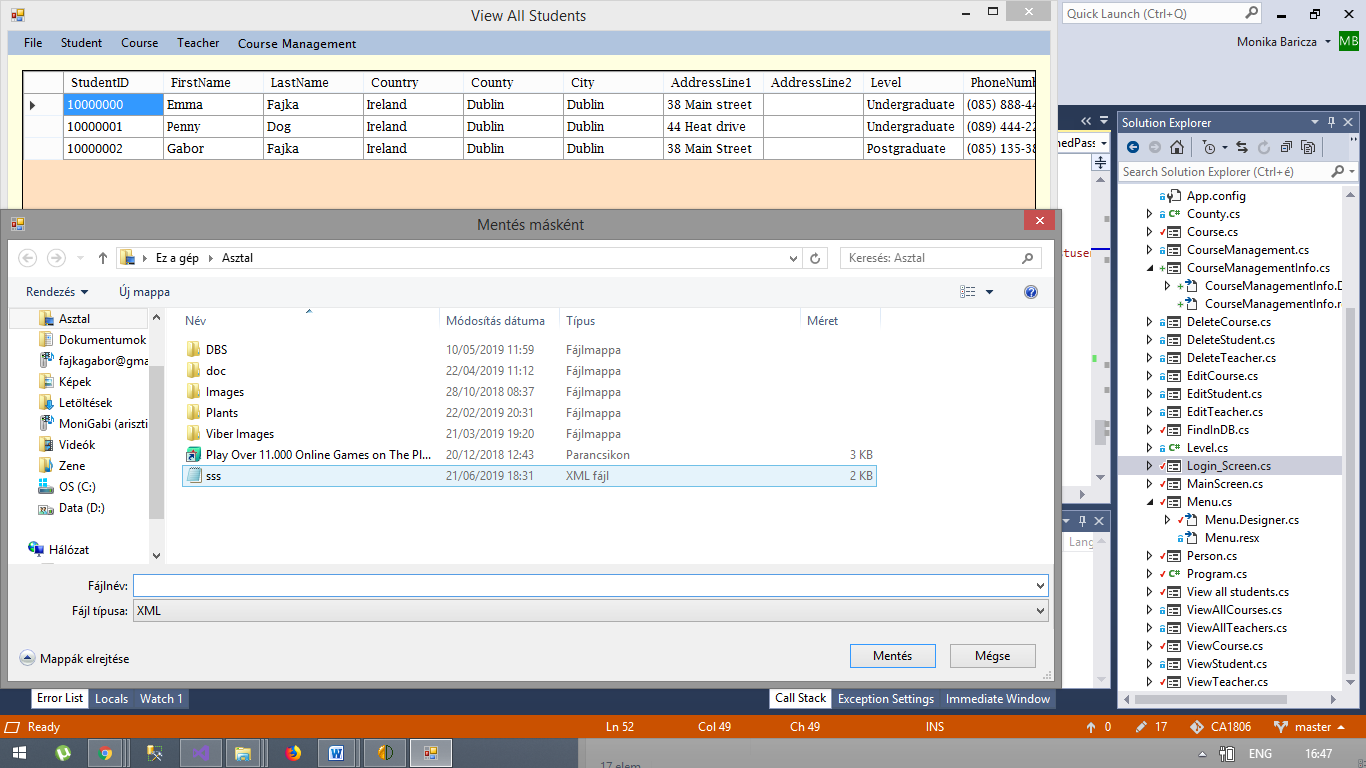


Figure Saving as XML

In addition to these requirements, option to handle teachers and courses in a similar way has been added. On my personal opinion it would have been more practical to use one form to handle one class. For example after adding the student, the same form could have been used to delete, edit or view the student. But because in the requirement it was specified that under the student menu point there should be “New student”, “Edit student”, “Delete student” and “View Database history” I designed the application this way. If I use the same form to manage the student, I would have had nothing to put under these menu points.

#### Business tier

The application itself consists of 23 windows forms and 2 enums. I will use the Student management as an example, the course and teacher management are very similar. Please see the diagram below that represents inheritance in the application. All the windows forms are inheriting from the form called “Menu”. The full menustrip is on this form and some items from the menu are hidden depending on which form is open. The Student management inherits from the Person form. In the Menu class there is an EstablishConnection() method which connects to the database. There is a GetAllIDs() method as well which is a support method for the previous and next buttons in the application. In the person form there is a ClearForm() method which clears the form and sets the StudentID to the next available ID in the database using a method GetLastID(int action). Action equals to 1 when searching for students, it equals to 2 when searching for teachers and equals to 3 when searching for courses. In the Person class there is a MustFillUp() method that throws an exception if not all the required fields are filled up. The RetrieveInfoForSelectedStudent(int ID) method gets all the information of the selected student searching by ID. In the AddStudent, EditStudent, DeleteStudent and ViewStudent forms an SQL query is executing the required command.

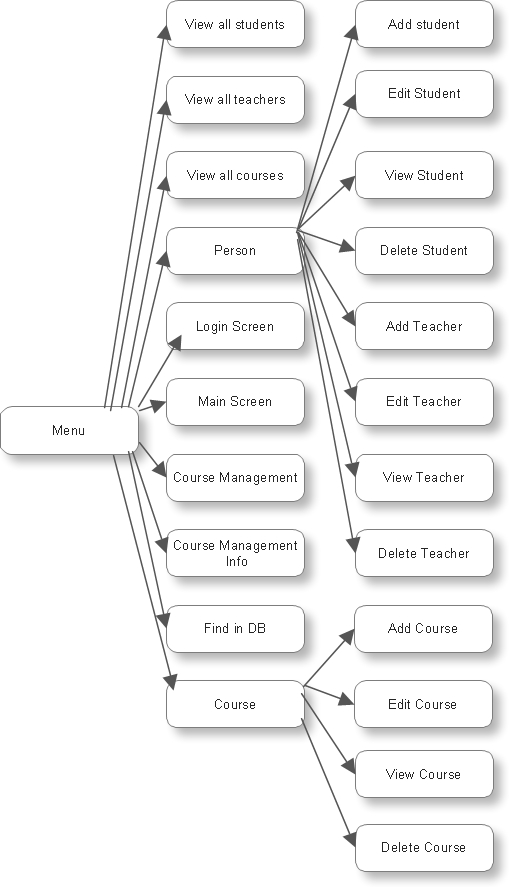


Figure Inheritance

Under the CourseManagement, the Admin can enrol a student. This associates the selected course’s CourseID with the selected student’s StudentID. If the admin clicks Course Management Info, they can cancel a student’s enrolment for a course and also they can see some statistics for the course (students enrolled, number of students enrolled, total income).

#### Data tier

To test the application, a database called StudentDB needs to be created in the MySQL LocalDB as the database cannot be zipped with the rest of the project. Once created, the queries in the SQLQueries folder need to be executed. I also included some queries to fill up the database with some data but they are not necessary for the program to run. As you can see from the Class diagram, there are four main tables. The Student table has all the student information, while the Teacher table stores the teachers’ details. In the Course table there is a foreign key referring to the teacher table. In the CourseManagement table we can enrol students to courses by associating their IDs. In this table the primary key is the CourseID and the StudentID together.

## Final thoughts

Overall, I feel the project’s aim has been achieved. Students, teachers and courses can be added, updated, searched and deleted. Also the admin has the option to enrol students to courses. User rights are restricted, so a regular user (Teacher) can only view the data in the database. I really enjoyed creating this application as it gave me a better insight on how databases are handled together with applications.