**CS 555**

**HOMEWORK 8**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** December 15, 2018

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

**[CONCLUSION:](#_Toc446970375)**21

**INTRODUCTION**

The objective of this assignment is to get used to and understand better Razor pages. This assignment consists of two parts: the first one focused on Identity Framework and the second one focused on Entity Framework (use of Linq).

**SCREEN SHOTS:**

**IDENTITY FRAMEWORK:**

The first step in order to be able to start practicing the Razor pages features was to stablish a connection with our DB named ComplexDb2. In order to do that we needed to change the appsetting.json with the following code:

{

"ConnectionStrings": {

"ComplexDb2": "Server=USUARIO\\TEW\_SQLEXPRESS; Trusted\_Connection=True;Database=ComplexDb2; MultipleActiveResultSets=true;"

},

"Logging": {

"LogLevel": {

"Default": "Warning"

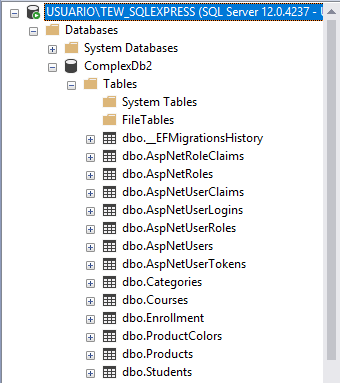
}

},

"AllowedHosts": "\*"

}

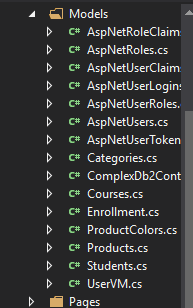
After that change was made, we needed to update our DB through the Package Manager Console using the update-database command. Once that command is run we could see some changes in our DB:



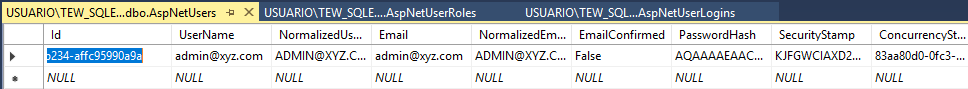
After making those changes to the DB, we now needed to be able to access to all the tables in our project. In order to be able to do that, we needed to first create a Models folder (where we would have all the models corresponding to the distinct tables) and then run the following command:

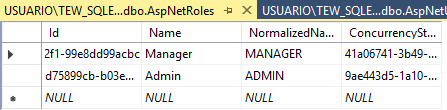
*Scaffold-DbContext "Server=USUARIO\TEW\_SQLEXPRESS;Database=ComplexDb2;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models*

That command will create all the models needed to access our DB in our Models folder as shown bellow:

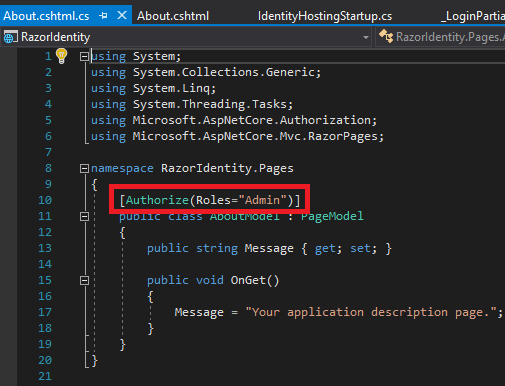


Now all we needed to do was to inject the context in our startup.cs in order to have access to our DB by just calling the context referring to it. Also, I added the logic in order to give roles to the users logged in (attached Admin role to the email registered). We can go to the DB and check that the users are getting registered properly with the defined roles:

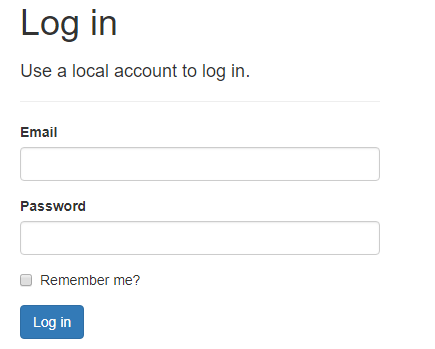




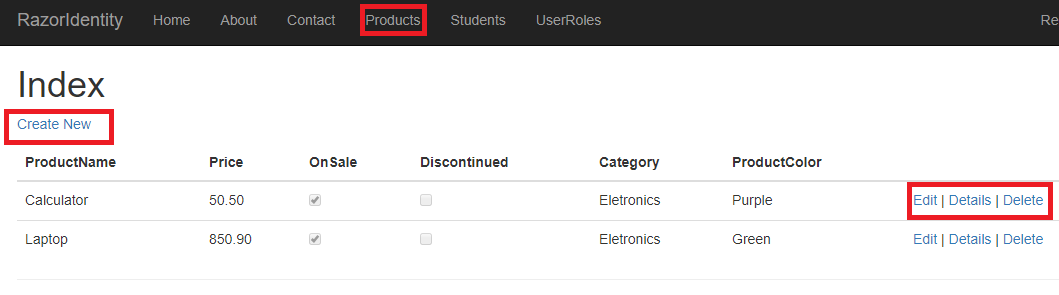
In order to check the identity framework is working properly, I protected the about page in a way that just if you have Admin role can access it. Otherwise, you will be redirected to the login page:



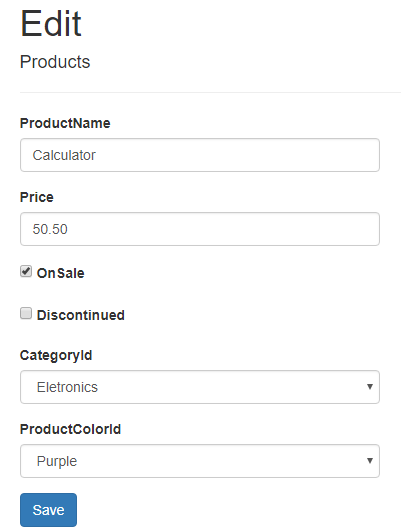
Redirected to login page since I tried to access to the About page without Admin role:

****

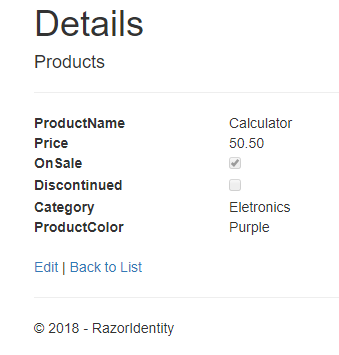
Using CRUD we created different pages for Create, Delete, Details and Edit pages. I needed to make some modifications in order to show the views the way I wanted (drop downs, tables etc.). On the products page we will add the links in order to delete, show details and create new products:



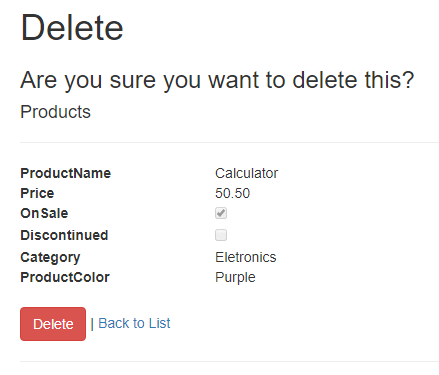
Edit:



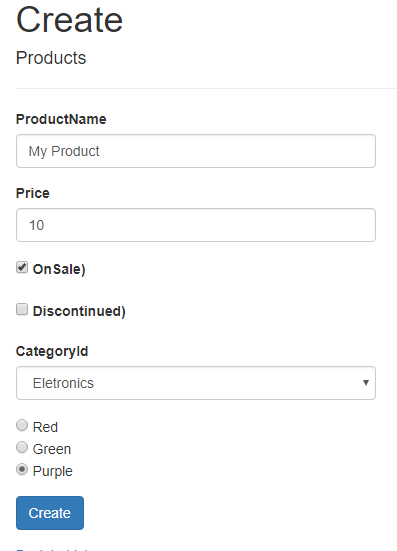
Details:



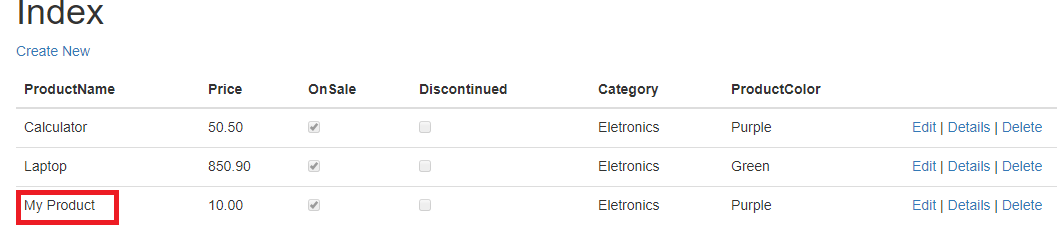
Delete:



Create:

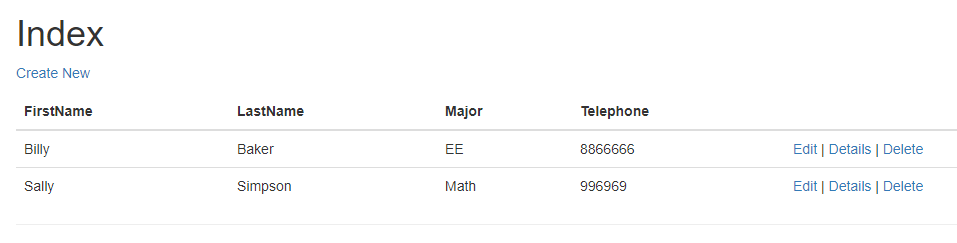


New product added to the list:

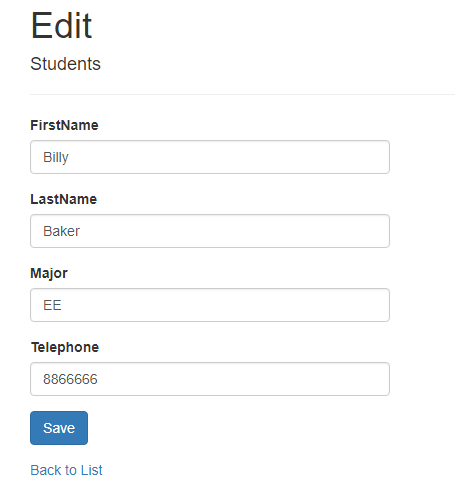


The second part was to do a similar thing but using students Models or in order to create new Roles for the last part:

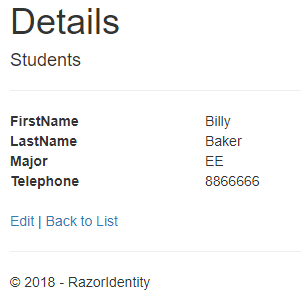
Students List:



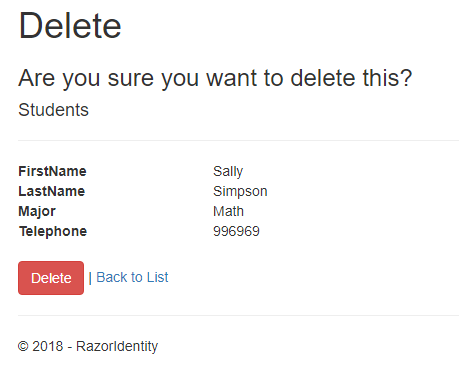
Edit Students, any change made on this page will have an effect on the list and DB of Students:



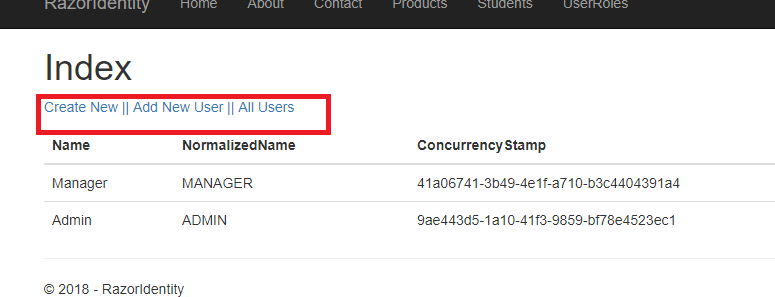
Details:



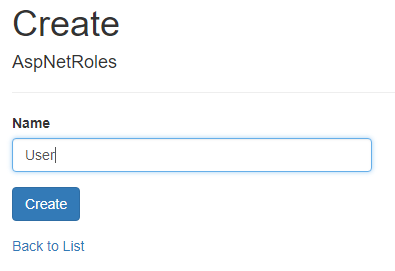
Delete Student:

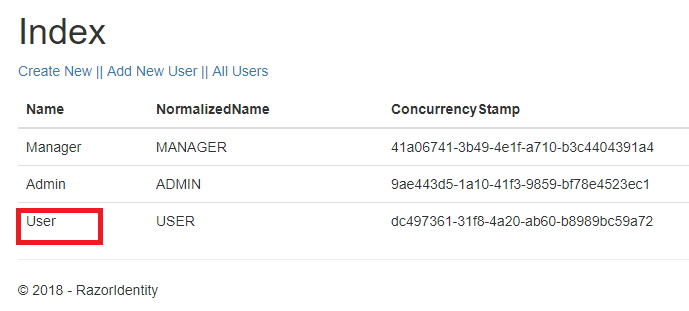


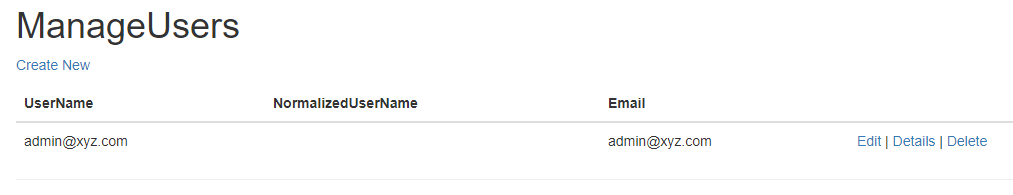
On the last part we are going to be able to list all the roles available (from DB) and on the top of the pages I put the links in order to create a new role, see the details of the roles and creating a new student:

****

Creating new role:

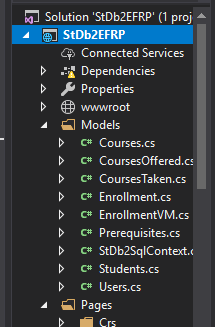
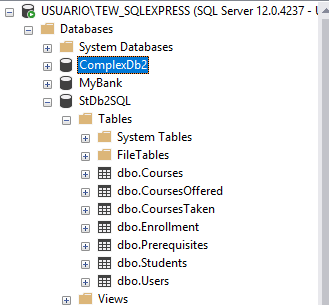
****

****

****

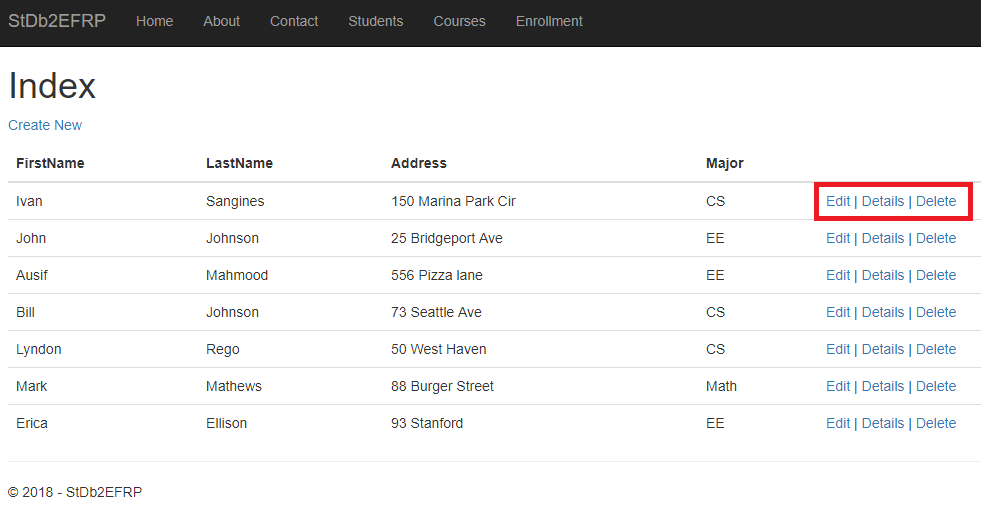
**ENTITY FRAMEWORK**

As in the first part of the assignment, we need to set up the connection with our DB and run the corresponding command in order to create the models attached to our DB tables.

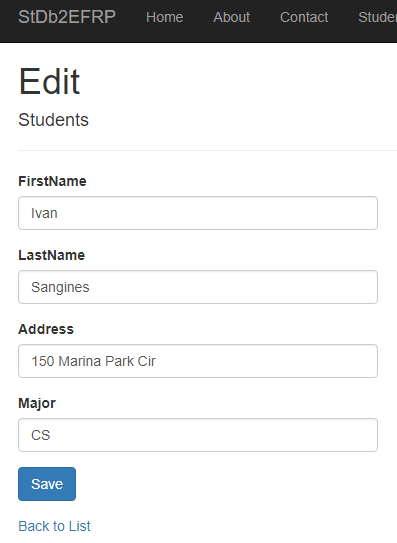


Once the connection and context with the DB is created. We can start creating the razor pages.

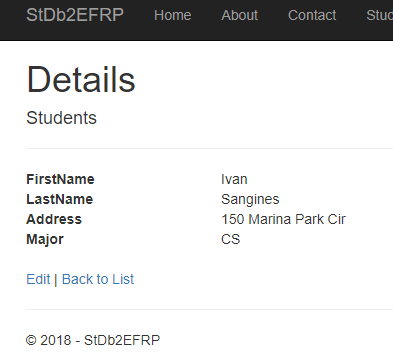
The first page I created is in order to show the list of all the students. The view for this page will be of type list and will show all the students on a table. Also three links will appear next to each student in order to delete, show details or edit the student information, this was done by using the templates CRUD:



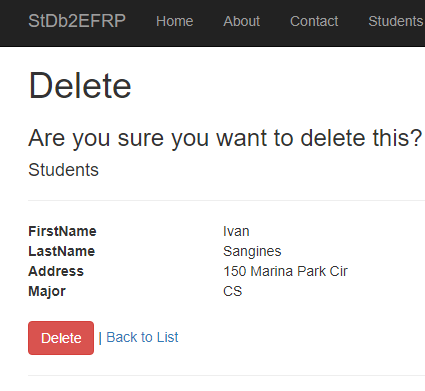
Edit student:



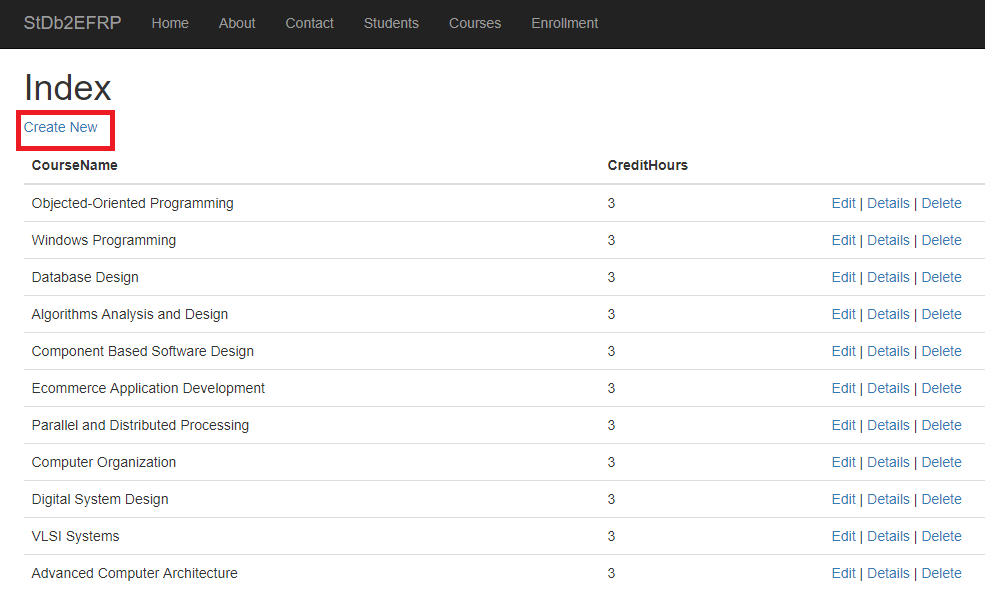
Show detail for a specific student:

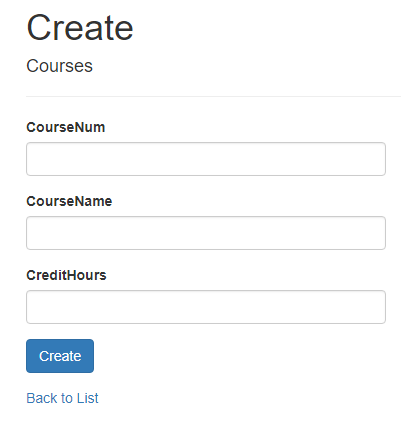


Delete student:

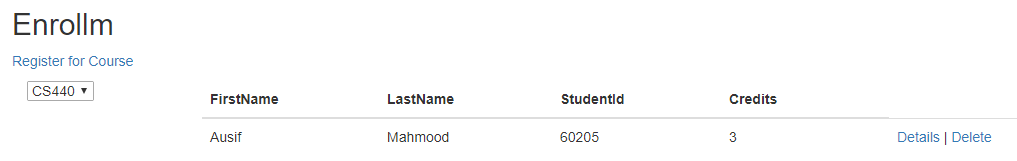


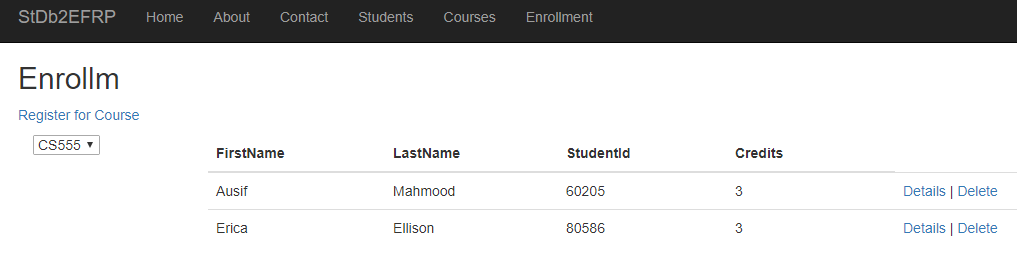
The next step was to create a similar page for the existent courses. Same as before, we will be using CRUD to create links in order to edit, delete and show details. This time in order to create a new course, I have added a link on the top of the page, this link will redirect us to a new page to provide all the information that will be matched to the courses model fields.





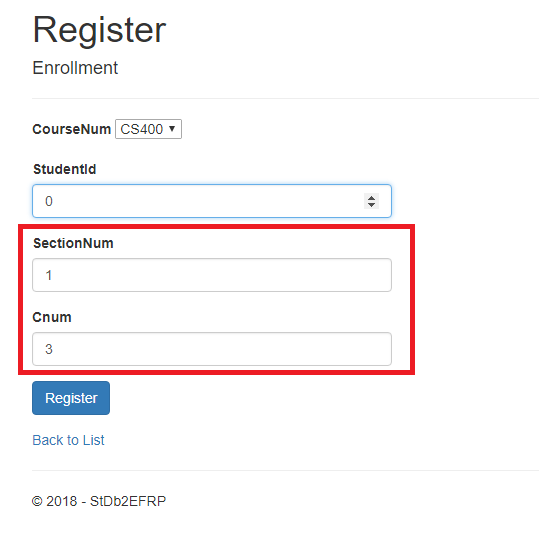
The last part of this assignment was to create a Enrollment page where we will have a dropdown button with all the available courses. Once a course is selected, all the students enrolled in that course will show up.



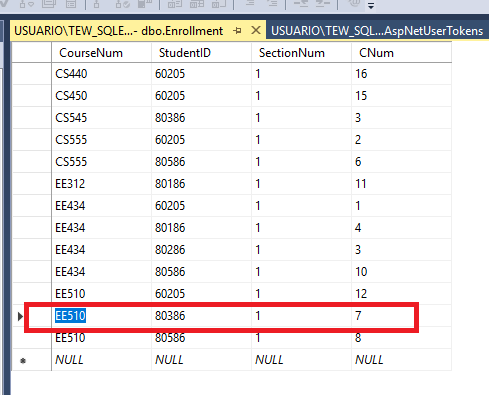


The extra part was to create a new page in order to be able to register students for a specific course. In order to do that, I have created a page where the first field required is the studentID, through the LINQ query I will check if that student exists. Once that is checked, I needed to verify that the student has all the perquisites for that course. I achieved that by creating a query with LINQ and checking all the prerequisites for the course we want to enroll the student. Then, I will get a list of courses taken by that student and compare if the prerequisites are there. If all these previous validations are correct, the student will be added to the new course. The section number and credit hours are automatically filled up by using another LINQ query for the course selected (showed in the red box). Once the student is added, it will automatically appear on the database (using context).

Register page:



New Student Added:



**Conclusion:**

After doing this assignment I was able to understand better how Razor pages work. I can say it is very similar to MVC but most of the configuration is done automatically. This assignment made me realize how great and useful are Razor pages, we can avoid most of the configurations problems since a lot of the work is already done automatically for us. Features like CRUD pages are very useful since they are already precreated useful pages that we might want to use in any web page, all they need are small changes in order to show the page as we want (add dropdowns, not show inputs for all the model fields etc.). On the other hand, this assignment helped me practice and get used to LINQ, which is very similar to SQL queries with the difference it operates on the context linked to our DB, it can be converted to a list really easy and has collection properties.