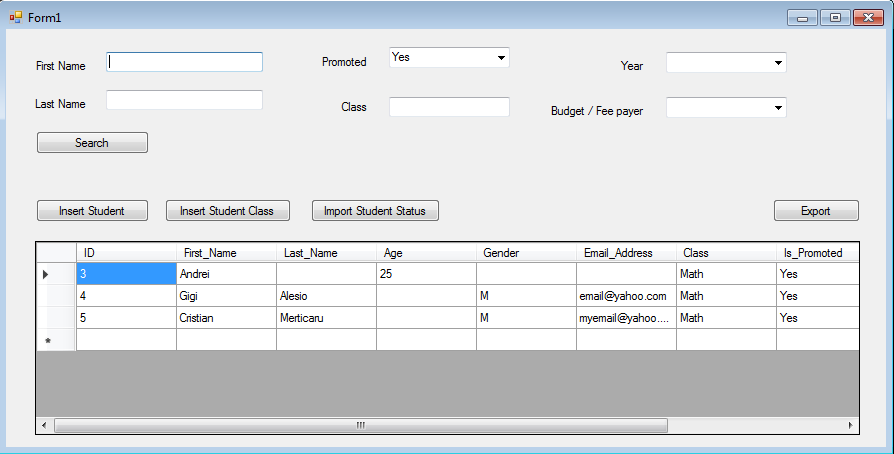
**Input – Output Management module**

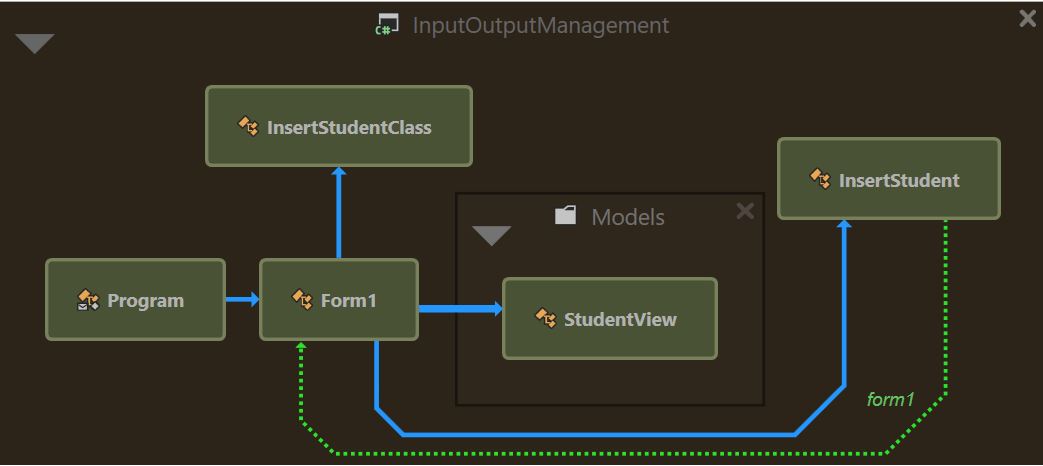
**Phase 1**

This module was build in order to present to the user an interface whereby he can execute the operations implemented in the project. Therefore the interface exposes in a facile manner all the input and the actions which the user can perform for the management of the academic records of the students, as shown in the picture below.



The input-output management module was implemented using windows forms for user dialogs. OpenFileDialogs and SaveFileDialogs importing and exporting students information to and from the database.

The overall structure of the application is relatively simple, as shown in the following diagram:



The system provides the following functionalities:

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Method Behavior** | **Return Type** | **Parameters** |
| **btnSearch\_Click** | **This method is called when “Search” button is pressed. It implements the search functionality of the students. The search functionality is a complex one that based on the input fields of the user, the search is made over 3 tables : students classes, students and student statuses and then intersect the search results from all the tables by generating the final list of the students that is finally rendered in the interface table.** | **void** | **NA** |
| **GetLastSearchResult** | **This method returns the final result list of the students that is finally rendered in the interface table. This list is generated by the intersection of the search results from all the tables. The method is caled from the btnSearch\_Click method.** | **List<StudentView>** | **NA** |
| **GetStudentClassToAdd** | **Returns the results from the table “StudentClass” by Id if the “Promoted” and /or “Class” text fields is/are filled. The method is caled from the GetLastSearchResult method.** | **List<int>** | **NA** |
| **GetStudentsStatusToAdd** | **Returns the results from the table “StudentStatus” by Id if the “Budget/Fee Payer” text field is filled The method is caled from the GetLastSearchResult method.** | **List<int>** | **NA** |
| **GetStudentsToAdd** | **Returns the results from the table “Students” by Id if the “First Name” and / or “Last Name” text field is / are filled The method is caled from the GetLastSearchResult method.** | **List<int>** | **NA** |
| **btnImport\_Click** | **Based on the input from a file, this method takes all the data that contains some students and insert them in the “Student” table. The method is called when the “Import” button from the “Insert Student” popup that manages the insertion of a student, is pressed.** | **void** | **object sender, EventArgs e** |
| **btnImportStatus\_Click** | **Based on the input from a file, this method takes all the data that contains some students statuses and insert them in the “StudentStatus” table. This method is called when the “Import Student Status” button from the main window is pressed.** | **void** | **object sender, EventArgs e** |
| **btnImportStudentClasses\_Click** | **Based on the input from a file, this method takes all the data that contains some students classes and insert them in the “StudentClasses” table. This method is called when the “Import ” button from the window that manages the insertion of the student classes is pressed.** | **void** | **object sender, EventArgs e** |
| **Insert\_Click** | **This method from Form2.cs inserts a single student based on the informations filled in the input.** | **void** | **object sender, EventArgs e** |
| **insertClass\_Click** | **This method from Form3.cs inserts a single student class based on the input from the user.** | **void** | **object sender, EventArgs e** |

**Phase 2**

In the second phase we added tests in the **InputOutputManagement.UnitTests** project, using NUnit Framework and Moq.

The tests done on this phase were done mainly on data retrieving on load, data searching based on different fields and the insertion functionality.

Each form class has it’s own test class :

1. **Form1Tests**

This class was build in order to test the operations with ”Form1” window form. Firstly, an init method is made for creating a new mock for IDatabaseContext interface. Then, a set of methods were created for tesing the behavior of this entity.

* test\_GetStudentata()
* test\_GetStudentStatusToAdd()
* test\_GetStudentClassToAdd()
* test\_SearchAfterName()
* test\_SearchAfterNameThosePromoted()
* test\_SearchShouldReturnNoValue()

1. **Form2Tests**

This class was build in order to test the operations with ”Form2” window form. Firstly, an init method is made for creating a new mock for IDatabaseContext interface. Then, a set of methods were created for tesing the behavior of this entity.

* test\_GetLastStudentId()
* test\_InsertStudent()

**Phase 3**

On the third phase assertions were added to the **Database module** to check various preconditions, post-conditions and invariants.

Example of an assertion for a **precondition** that informs the developer if a filepath can’t be found.

* Debug.Assert(Debug.Exists(filePath), "The script file must exist."));
* Debug.Assert(length < 0, "WHERE instruction must be after SET instruction");

This assertion of type **postconditions** is telling to developer that if in the UPDATE statement there is a condition stated by WHERE then the condition must come after SET from the statement. In other words the statement must follow the SQL conventions of an UPDATE statement.

- Debug.Assert(!instruction.Contains(Instructions.Set), "update instruction must specify at least a set with a new value to be changed");

This assertion of type precondition is telling to developer that the UPDATE statement should always include the new values that updates the table from database.

**Contributions**

Name: Colban Mihai

Contribution:

• wrote documentation on the **Input Output Management** module, covering phases 1, 2 and 3,

• worked on application interface

• implemented the import/export file methods for the **Input Output Management** module,

• implemented unit tests for the **Input Output Management** module,

• added precondition assertions in the database module,

• helped other team members when needed.

Name: Merticaru - Rafael Cristian - Ovidiu

• wrote documentation on the **Input Output Management** module, covering phases 1, 2 and 3

• worked on application interface

• implemented the search, insert student and insert student class methods for the **Input Output Management** module

• added precondition and postconditions assertions in the database module

• helped other team members when needed

Note: for a more detailed overview on the contributions please go to:<https://github.com/EchipaProiectCSS/AcademicManager/graphs/contributors>