# Documentation

## 1. Components / Objects

### Components

The `Components.h` file serves as a central header file for including various components and defining important structures and variables used in the C++ program.

### 1. Included Libraries

The following libraries are included in the `Components.h` file:

- `<iomanip>`: Provides functions for manipulating input/output formatting.
- `<string>`: Provides string manipulation functions and classes.
- `"Vector.h"`: User-defined header file that implements a dynamic array data structure.

### 2. Constants

- `defaultStr`: A constant string that holds the value “Error”. It is used as a default value for certain string variables.

### 3. Enumerators

The following enumerators are defined in the file:

- `Weekday`: Represents the days of the week with values from `SUN` to `SAT`.
- `Gender`: Represents gender with values `male` and `female`.
- `Session`: Represents academic sessions with values `S1`, `S2`, `S3`, and `S4`.

### 4. Structures

The following structures are forward declared in the file:

- `Date`: Represents a date.
- `Name`: Represents a person’s name.
- `User`: Represents a user.
- `Staff`: Represents a staff member.

- `SchoolYear`: Represents a school year.
- `Class`: Represents a class.
- `Student`: Represents a student.
- `AcademicYear`: Represents an academic year.
- `Semester`: Represents a semester.
- `Course`: Represents a course.
- `Scoreboard`: Represents a scoreboard for a student's scores.

## 5. Struct Operator Overloading

The file includes a template function `operator!=` that overloads the inequality operator for comparing two objects of the same type. It returns `true` if the objects are not equal and `false` otherwise.

## Global Variables

The file declares the following global variables:

- `schoolYears`: A `Vector` container that stores `SchoolYear` objects.
- `academicYears`: A `Vector` container that stores `AcademicYear` objects.
- `students`: A `Vector` container that stores `Student` objects.
- `staffs`: A `Vector` container that stores `Staff` objects.
- `ptrStaff_Global`: A pointer to a `Staff` object.
- `ptrStudent_Global`: A pointer to a `Student` object.
- `ptrSchoolYear_Global`: A pointer to a `SchoolYear` object.
- `ptrClass_Global`: A pointer to a `Class` object.
- `ptrAcademicYear_Global`: A pointer to an `AcademicYear` object.
- `ptrSemester_Global`: A pointer to a `Semester` object.
- `ptrCourse_Global`: A pointer to a `Course` object.
- `ptrScoreboard_Global`: A pointer to a `Scoreboard` object.

These global variables are used to maintain and access important data throughout the program.

The `Components.h` file serves as a central point for including necessary components and defining key structures and variables, making them easily accessible to other parts of the program.

## 1. Name

The `Name` struct represents a person's name with a first name and a last name. It provides methods for setting and retrieving the name, as well as an equality operator to compare two names.

| Attribute/Method | Description |
|---|---|
| - `first: std::string` | Private attribute that stores the first name of the person. |
| - `last: std::string` | Private attribute that stores the last name of the person. |
| + `Name(nameFirst: const std::string&, nameLast: const std::string&)` | Public constructor that initializes the `Name` object with the given first and last names. It has default parameter values of an empty string. |
| + `set(nameFirst: const std::string&, nameLast: const std::string&)` | Public method that allows setting the first and last names of the person. |
| + `get(): std::string` | Public method that returns the full name (first name followed by last name) as a string. |
| `operator==(nameA: const Name&, nameB: const Name&)` | Non-member function that compares two `Name` objects for equality. Returns `true` if the first names and last names are the same, and `false` otherwise. |

## 2. Date

The `Date` struct represents a specific date with day, month, and year values. It provides methods for setting and retrieving the date, as well as an equality operator to compare two dates.

| Attribute/Method | Description |
|---|---|
| - `day: unsigned short` | Private attribute that stores the day value of the date. |
| - `month: unsigned short` | Private attribute that stores the month value of the date. |
| - `year: unsigned int` | Private attribute that stores the year value of the date. |
| + `Date(newDay: const unsigned short&, newMonth: const unsigned short&, newYear: const unsigned int&)` | Public constructor that initializes the `Date` object with the given day, month, and year values. It has default parameter values of 0. |
| + `set(newDay: const unsigned short&, newMonth: const unsigned short&, newYear: const unsigned int&)` | Public method that allows setting the day, month, and year values of the date. |

| | |
|---|---|
| + `get(): std::string` | Public method that returns the date in the format “dd/mm/yyyy” as a string. |
| `operator==(dateA: const Date&, dateB: const Date&)` | Non-member function that compares two `Date` objects for equality. Returns `true` if the day, month, and year values are the same, and `false` otherwise. |

## 3. User

The `User` struct represents a user with a name, ID, and password. It provides various methods for setting and retrieving user information, as well as password encryption and comparison.

| Attribute/Method | Description |
|---|---|
| - `password: uint64_t` | Private attribute that stores the hashed password of the user. |
| - `hash(left: uint64_t, right: uint64_t): uint64_t` | Private method that performs a custom hash operation on two `uint64_t` values and returns the result. |
| - `hash(str: std::string): uint64_t` | Private method that performs a custom hash operation on a string and returns the result. |
| + `name: Name` | Public attribute of type `Name` that represents the user’s name. |
| + `ID: std::string` | Public attribute that stores the user’s ID. |
| + `User(name: const Name&, ID: const std::string&, passwordStr: const std::string&)` | Public constructor that initializes the `User` object with the given name, ID, and password. The default password value is “123456”. |
| + `setFirstName(firstName: const std::string&)` | Public method that allows setting the first name of the user. |
| + `setLastName(lastName: const std::string&)` | Public method that allows setting the last name of the user. |
| + `setName(first: const std::string&, last: const std::string&)` | Public method that allows setting the first and last names of the user. |
| + `setPassword(str: const std::string&)` | Public method that encrypts and sets the user’s password using the given string. |
| + `isPassword(str: const std::string&): bool` | Public method that compares the given string with the user’s password. Returns `true` if the passwords match, and `false` otherwise. |
| + `setID(ID: const std::string)` | Public method that allows setting the user’s ID. |
| + `getHashedPass(): uint64_t` | Public method that returns the hashed password of the user. |
| `operator==(userA: const User&, userB: const User&): bool` | Non-member function that compares two `User` objects for equality. Returns `true` if the IDs and names of the users are the same, and `false` otherwise. |

## 4. Staff

The `Staff` struct is a derived struct from `User` and represents a staff member. It inherits the attributes and methods from `User`.

## 5. Student

The `Student` struct is defined in the Student.h file and inherits from the User class. It contains data related to a student such as their name, ID, password, gender, date of birth, social ID, class, and scoreboards.

The following table provides an overview of the attributes and methods of the Student struct:

| Attribute / Method | Description |
| --- | --- |
| `+ Gender gender` | A public attribute representing the gender of the student. |
| `+ Date birth` | A public attribute representing the date of birth of the student. |
| `+ std::string socialID` | A public attribute representing the social ID of the student. |
| `+ Class* ptrClass` | A public attribute representing a pointer to the class that the student belongs to. |
| `+ Vector<Scoreboard*> scoreboards` | A public attribute representing a vector of scoreboards for the student. |
| `+ Student(const Name& name, const std::string& id, const std::string& password, const Gender& gender = male, const Date& birth = {0, 0, 0}, const std::string& socialID = defaultStr, Class* ptrClass = nullptr, const Vector<Scoreboard*>& scoreboards = Vector<Scoreboard*>())` | A public constructor that takes in the name, ID, password, gender, date of birth, social ID, class, and scoreboards of a student. |
| `+ Vector<Scoreboard*> getScoreboards(const Semester& semester) const` | A public method that takes in a semester and returns a vector of scoreboards for the student in that semester. |
| `+ Scoreboard* getScoreboard(const string& courseID) const` | A public method that takes in a course ID and returns a |

| | |
|---|---|
| | pointer to the scoreboard of that course for the student. |
| `+ Scoreboard* getScoreboard(Course& course) const` | A public method that takes in a course and returns a pointer to the scoreboard of that course for the student. |
| `+ void set(const Name& name, const std::string& id, const std::string& password, const Gender& gender, const Date& birth, const std::string& socialID, Class* ptrClass, const Vector<Scoreboard*>& scoreboards)` | A public method that sets the name, ID, password, gender, date of birth, social ID, class, and scoreboards of a student. |
| `+ void setName(const Name& name)` | A public method that sets the name of a student. |
| `+ void setGender(const Gender& gender)` | A public method that sets the gender of a student. |
| `+ void setBirth(const Date& birth)` | A public method that sets the date of birth of a student. |
| `+ void setSocialID(const std::string& socialID)` | A public method that sets the social ID of a student. |
| `+ void setClass(Class* ptrClass)` | A public method that sets the class of a student. |
| `+ void setScoreboards(Vector<Scoreboard*>& scoreboards)` | A public method that sets the scoreboards of a student. |
| `+ float getGPA() const` | A public method that returns the overall GPA of a student. |
| `+ float getGPA(const Semester& semester) const` | A public method that takes in a semester and returns the GPA of the student for that semester. |
| `+ void setInfoToClass(std::ifstream &ifs)` | A public method that reads student information from a file and sets it to |

## 6. SchoolYear

The `SchoolYear` struct represents a school year in a educational system. It contains information about the start year and the classes associated with that year.

| Attribute/Method | Description |
|---|---|
| - `start` : unsigned int | The starting year of the school year. |

| | |
|---|---|
| - `classes` : Vector<Class> | A vector of `Class` objects representing the classes associated with the school year. |
| + `getStartYear()` : unsigned int | Returns the starting year of the school year. |
| + `set(const unsigned int& start, const Vector<Class>& classes)` | Sets the start year and classes of the school year. |
| + `update(const unsigned int& start)` | Updates the start year of the school year. |
| + `update(Vector<Class>& classes)` | Updates the classes of the school year. |
| + `getStudent(const std::string& studentID)` : Student* | Returns a pointer to a `Student` object with the given student ID, if found in any of the classes within the school year. |
| + `getClass(const std::string& className)` : Class* | Returns a pointer to a `Class` object with the given class name, if found within the school year. |
| + `addClass(Class& CLASS)` | Adds a `Class` object to the school year and assigns the school year as its parent. |
| + `removeClass(Class& CLASS)` | Removes a `Class` object from the school year and removes the school year reference from the class. |
| + `removeAllClass()` | Removes all classes from the school year and removes the school year reference from each class. |
| + `getPeriod()` : std::string | Returns a string representation of the school year period, indicating the start year and end year. |

## 7. Class

The `Class` struct represents a class within a school year in a C++ program. It contains information about the class name, the associated school year, and the students enrolled in the class.

| Attribute / Method | Description |
|---|---|
| - `ptrSchoolYear: SchoolYear*` | A pointer to the associated school year for the class. |
| - `name: std::string` | The name of the class. |
| - `students: Vector<Student*>` | A vector containing pointers to the students enrolled in the class. |
| + `Class(SchoolYear* ptrSchoolYear = nullptr, const std::string& name = defaultStr, const Vector<Student*>& students = Vector<Student*>())` | Constructor for the `Class` struct. Initializes the `ptrSchoolYear`, `name`, and `students` attributes. |
| + `void set(SchoolYear* ptrSchoolYear, const std::string& name, const Vector<Student*>& students)` | Sets the `ptrSchoolYear`, `name`, and `students` attributes of the class. |
| + `void update(SchoolYear* ptrSchoolYear)` | Updates the `ptrSchoolYear` attribute of the |

| | |
|---|---|
| | class. |
| + `void update(const std::string& name)` | Updates the `name` attribute of the class. |
| + `void addStudent(Student*& student)` | Adds a student to the class by appending the student pointer to the `students` vector. |
| + `void removeStudent(Student*& student)` | Removes a student from the class by removing the corresponding student pointer from the `students` vector. |
| + `Vector<string> getListCourse(const Semester& semester) const` | Retrieves a vector of course IDs associated with the specified semester for the class. |
| + `Vector<std::string> getListCourse() const` | Retrieves a vector of all course IDs for the class. |
| + `void removeAllStudent()` | Removes all students from the class by resetting the `students` vector. |
| + `Student* getStudent(const std::string& studentID)` | Retrieves a pointer to a student in the class based on their student ID. |
| + `void displayScoreboardScreen(const Semester& semester)` | Displays the scoreboard of the class for the specified semester on the console. |
| + `void displayScoreboardFile(const Semester& semester, std::ofstream& ofs)` | Writes the scoreboard of the class for the specified semester to a file. |
| **Friend Functions** | |
| bool operator==(const Class& classA, const Class& classB) | Checks whether two `Class` objects are equal by comparing their `name` and `ptrSchoolYear` attributes. |

## 8. AcademicYear

The `AcademicYear` struct represents an academic year, which is defined by a start year and a collection of semesters. It contains methods for adding, removing, and getting a semester by its ID.

| Attribute/Method | Description |
|---|---|
| - `unsigned int start` | The start year of the academic year. |
| - `Vector<Semester> semesters` | A vector of semesters in the academic year. |
| + `AcademicYear(const unsigned int& start = 0, const Vector<Semester>& semester = Vector<Semester>())` | Constructor to create an academic year with a start year and a vector of semesters. |
| + `Semester* getSemester(const std::string& semesterID)` | Returns a pointer to the semester with the |

| | |
|---|---|
| | specified ID, or nullptr if it does not exist. |
| + `void addSemester(Semester& semester)` | Adds a semester to the academic year. |
| + `void removeSemester(Semester& semester)` | Removes a semester from the academic year. |
| + `void removeAllSemester()` | Removes all semesters from the academic year. |
| + `std::string getPeriod() const` | Returns a string representing the period of the academic year, in the format “start year - end year”. |
| - `bool operator==(const AcademicYear& yearA, const AcademicYear& yearB)` | Returns true if two academic years are equal (have the same start year), false otherwise. |

## 9. Semester

The `Semester` struct represents a semester in an academic year. It contains information about the semester ID, start and end dates, a collection of courses, and a pointer to the academic year it belongs to. It provides methods for managing courses within the semester.

| Attribute/Method | Description |
|---|---|
| + `std::string semesterID` | The ID of the semester. |
| + `Date startDate` | The start date of the semester. |
| + `Date endDate` | The end date of the semester. |
| + `Vector<Course> courses` | A vector of courses in the semester. |
| + `AcademicYear* ptrAcademicYear` | A pointer to the academic year that the semester belongs to. |
| + `Semester(const std::string& semesterID = defaultStr, const Date& startDate = { 0, 0, 0 }, const Date& endDate = { 0, 0, 0 }, const Vector<Course>& course = Vector<Course>(), AcademicYear* ptrAcademicYear = nullptr)` | Constructor to create a semester with the specified semester ID, start and end dates, a vector of courses, and a pointer to the academic year. |
| + `void set(const std::string& semesterID, const Date& startDate, const Date& endDate, const Vector<Course>& course, AcademicYear* ptrAcademicYear)` | Sets the attributes of the semester to the specified values. |
| + `void updateSemesterID(const std::string& semesterID)` | Updates the semester ID. |
| + `void updateStartDate(const Date& startDate)` | Updates the start date of the semester. |
| + `void updateEndDate(const Date& endDate)` | Updates the end date of the semester. |
| + `void updateCourses(Vector<Course>& course)` | Updates the courses in the semester. |
| | |

| | |
|---|---|
| + `void updateAcademicYear(AcademicYear* ptrAcademicYear)` | Updates the academic year that the semester belongs to. |
| + `Course* getCourse(const std::string& courseID)` | Returns a pointer to the course with the specified ID, or nullptr if it does not exist. |
| + `void addCourse(Course& course)` | Adds a course to the semester. |
| + `void removeCourse(Course& course)` | Removes a course from the semester. |
| + `void removeAllCourse()` | Removes all courses from the semester. |
| - `bool operator==(const Semester& semA, const Semester& semB)` | Returns true if two semesters are equal (have the same semester ID, start and end dates), false otherwise. |

## 10. Course

The `Course` struct in the C++ programming language represents a course offered in an educational system. It contains various attributes and methods to manage and manipulate course information. Below is a description of the attributes and methods of the `Course` struct.

| Attribute/Method | Description |
|---|---|
| - `ID` (`string`) | The ID of the course. |
| - `classID` (`string`) | The ID of the class associated with the course. |
| - `name` (`string`) | The name of the course. |
| - `teacher` (`string`) | The name of the teacher who teaches the course. |
| - `credits` (`int`) | The number of credits assigned to the course. |
| - `maxEnroll` (`int`) | The maximum number of students that can enroll in the course. |
| - `weekday` (`Weekday`) | The day of the week when the course is scheduled. |
| - `session` (`Session`) | The session of the course (e.g., morning, afternoon). |
| - `ptrSemester` (`Semester*`) | A pointer to the semester associated with the course. |
| - `scoreboards` (`Vector<Scoreboard*>`) | A vector of pointers to scoreboards, |

| | |
|---|---|
| | representing the performance of students in the course. |
| + `Course()` | Constructor of the `Course` struct. |
| + `void set(ID, classID, name, teacher, credits, maxEnroll, weekday, session, ptrSemester, scoreboards)` | Sets the values of the attributes of the course. |
| + `void updateID(ID)` | Updates the ID of the course. |
| + `void updateClassID(classID)` | Updates the class ID of the course. |
| + `void updateName(name)` | Updates the name of the course. |
| + `void updateTeacher(teacher)` | Updates the name of the teacher of the course. |
| + `void updateCredits(credits)` | Updates the number of credits assigned to the course. |
| + `void updateMaxEnroll(maxEnroll)` | Updates the maximum number of students that can enroll in the course. |
| + `void updateWeekday(weekday)` | Updates the day of the week when the course is scheduled. |
| + `void updateSession(session)` | Updates the session of the course. |
| + `void updateSemester(ptrSemester)` | Updates the semester associated with the course. |
| + `void updateScoreboard(scoreboards)` | Updates the scoreboards associated with the course. |
| + `Student* getStudent(studentID)` | Retrieves a pointer to a student enrolled in the course based on their ID. |
| + `Scoreboard* getScoreboard(studentID)` | Retrieves a pointer to the scoreboard of a student in the course based on their ID. |
| + `void addStudent(student)` | Adds a student to the course by creating a new scoreboard for the student. |
| + `void removeStudent(student)` | Removes a student from the course and deletes their associated scoreboard. |
| + `void removeAllStudent()` | Removes all students from the course and deletes their associated scoreboards. |
| + `void displayInfoFile(ofs)` | Displays the course information in a formatted manner in the specified output stream. |
| + `void displayInfoTable(table)` | Displays the course information in a tabular format using the provided table. |
| + `void importScoreBoards(ifs)` | Imports scoreboards for the course from the |

| | |
|---|---|
| | specified input file stream. |
| + `bool operator==(courseA, courseB)` | Compare two Course variable |

### 11. Scoreboard

The `Scoreboard` struct represents a score record for a specific course and student. It contains information about the scores obtained by the student in different assessments of the course.

| Attribute / Method | Description |
|---|---|
| - `midterm` (float) | The score obtained by the student in the midterm assessment. |
| - `final` (float) | The score obtained by the student in the final assessment. |
| - `other` (float) | The score obtained by the student in other assessments. |
| - `total` (float) | The total score obtained by the student in the course. |
| - `ptrCourse` (Course*) | A pointer to the course associated with the scoreboard. |
| - `ptrStudent` (Student*) | A pointer to the student associated with the scoreboard. |

## 2. Components Functions

### 1. Display functions

The following functions are used for displaying the respective structs as a drop box.

| Function | Description |
|---|---|
| `Vector<string> getListSchoolYear()` | return the list of school years (as the start year of each school year) to display to the screen |
| `Vector<string> getListAcademicYear()` | return the list of academic years (as the start year of each academic year) to display to the screen |
| `Vector<string> getListClass(const SchoolYear& schoolYear)` | return the list of classes inside a school year |
| `Vector<string> getListSemester(const AcademicYear& academicYear)` | return the list of semesters inside an academic year |
| `Vector<string> getListSemester(const Class& CLASS)` | return the list of semesters inside a class |
| `Vector<string> getListSemester(const Student &student)` | return the list of semester of a student |
| `Vector<string> getListCourse(const Semester& semester)` | return the list of courses from a semester |

The following functions are used for displaying the respective structs as a table.

| Function | Description |
|---|---|
| | |

| | |
|---|---|
| `Vector<Vector<string>> getTableContentOfListSchoolYear()` | return the list of school years with number of classes in each year |
| `Vector<Vector<string>> getTableContentOfSchoolYear(const SchoolYear& schoolYear)` | return the list of classes with number of student in each class |
| `Vector<Vector<string>> getTableContentOfListStudentInClass(const Class& CLASS)` | return the list of student with their information |
| `Vector<Vector<string>> getTableContentOfListScoreboardInSemesterInClass(const Class& CLASS, const Semester& semester)` | return the scoreboard in a semester of a class |
| `Vector<Vector<string>> getTableContentOfListScoreboardInClass(const Class& CLASS)` | return the scoreboard in a year of a class |
| `Vector<Vector<string>> getTableContentOfScoreboardOfStudent(const Student& student)` | return the scoreboard of a student in a year |
| `Vector<Vector<string>> getTableContentOfScoreboardInSemesterOfStudent(const Student& student, const Semester& semester)` | return the scoreboard of a student in a semester |
| `Vector<Vector<string>> getTableContentOfListAcademicYear()` | return the list of academic years with number of semesters in each year |
| `Vector<Vector<string>> getTableContentOfAcademicYear(const AcademicYear& academicYear)` | return the list of semesters and information of each in an academic year |
| `Vector<Vector<string>> getTableContentOfSemester(const Semester& semester)` | return the information of a semester |
| `Vector<Vector<string>> getTableContentOfListStudentInCourse(const Course& course)` | return the information of students in a course |

## 2. Download functions

The following functions download the data into a CSV file.

| Function | Description |
|---|---|
| `bool downloadAllData()` | download all the data |
| `bool downloadListStaff()` | download a list of staves |
| `bool downloadListStudent()` | download a list of students |
| `bool downloadListSchoolYear()` | download a list of school years |
| `bool downloadSchoolYear(SchoolYear& schoolYear)` | download the selected school year |
| `bool downloadClass(Class& CLASS)` | download the selected class |
| `bool downloadListAcademicYear()` | download a list of academic years |
| `bool downloadAcademicYear(AcademicYear&` | download the selected academic year |

| | |
|---|---|
| `academicYear)` | |
| `bool downloadSemester(Semester& semester)` | download the selected semester information |
| `bool downloadCourse(Course& course)` | download the selected course information |

## 3. File and Directory functions

The following functions help create the directory for importing and exporting functions.

| Function | Description |
|---|---|
| `void createDirectoryIfNotExists(const string& dirPath)` | create a new directory if it doesn't exist |
| `string getListStaffFilePath()` | return the directory of the list of staves file |
| `string getListStudentFilePath()` | return the directory of the list of student file |
| `string getListSchoolYearFilePath()` | return the directory of the list of school year file |
| `string getListAcademicYearFilePath()` | return the directory of the list of academic year file |
| `string getSchoolYearFolderPath(const SchoolYear& schoolyear)` | return the folder path of the start of school year |
| `string getSchoolYearFilePath(const SchoolYear& schoolyear)` | return the file path of the start of school year |
| `string getClassFolderPath(const Class& CLASS)` | return the folder path of the class |
| `string getClassFilePath(const Class& CLASS)` | return the file path of the class |
| `string getAcademicYearFolderPath(const AcademicYear& academicYear)` | return the folder path of the start of academic year |
| `string getAcademicYearFilePath(const AcademicYear& academicYear)` | return the file path of the start of academic year |
| `string getSemesterFolderPath(const Semester& semester)` | return the folder path of the semester |
| `string getSemesterFilePath(const Semester& semester)` | return the file path of the semester |
| `string getCourseFolderPath(const Course& course)` | return the folder path of the course |
| `string getCourseFilePath(const Course& course)` | return the file path of the course |
| `string getExportFolderPath()` | add prefix for data exporting path |
| `string getImportFolderPath()` | add prefix for data importing path |

## 4. Import and Export functions

The following functions import from and export to a CSV file.

| Function | Description |
|---|---|
| `bool importStudentListOfClassFromFile(const string& filename, Class& actClass, string& outStr)` | import the student list in a class |
| `bool importStudentListOfCourseFromFile(const string& filename, Course& course, string& outStr)` | import the student list in a course |
| `bool exportListOfStudentInCourse(const string& filename, Course& course, string& outStr)` | export the student list in a course |
| `bool importScoreBoardOfCourse(const string& filename, Course& course, string& outStr)` | import the scoreboard of a course |
| `bool exportListScoreboardOfStudent(const string& filename, Student& student, string& outStr)` | export the scoreboard in an academic year of a student |
| `bool exportListScoreboardInSemesterOfStudent(const string& filename, Student& student, Semester& semester, string& outStr)` | export the scoreboard in a semester of a student |
| `bool exportListSchoolYear(const string& filename, string& outStr)` | export the list of school years |
| `bool exportListClassInSchoolYear(const string& filename, SchoolYear& schoolYear, string& outStr)` | export the list of classes |
| `bool exportListStudentInClass(const string& filename, Class& CLASS, string& outStr)` | export the list of students in a class |
| `bool exportListScoreboardInSemesterOfClass(const string& filename, Class& CLASS, Semester& semester, string& outStr)` | export the scoreboard in a semester of a class |
| `bool exportListScoreboardOfClass(const string& filename, Class& CLASS, string& outStr)` | export the scoreboard of a class |
| `bool exportListAcademicYear(const string& filename, string& outStr)` | export the list of academic years |
| `bool exportListSemesterInAcademicYear(const string& filename, AcademicYear& academicYear, string& outStr)` | export the list of semesters in an academic year |
| `bool exportListCourseInSemester(const string& filename, Semester& semester, string& outStr)` | export the list of courses in a semester |
| `bool exportListScoreboardOfCourse(const string& filename, Course& course, string& outStr)` | export the scoreboard of the course |

## 5. Insert functions

The following functions insert data.

| Function | Description |
|---|---|

| | |
|---|---|
| `bool addStudent(const string& ID, const string& firstName, const string& lastName, const string& genderStr, const string& birthStr, const string& socialID, const string& password, string& outStr)` | add a new student |
| `bool addStaff(string curStaffID, const string& ID, const string& password, const string& firstName, const string& lastName, string& outStr)` | add a new staff |
| `bool addSchoolYear(const string& start, string& outStr)` | add a new school year |
| `bool addClass(SchoolYear& schoolYear, const string& className, string& outStr)` | add a new class for the school year |
| `bool addStudentToClass(Class& actClass, const string& studentID, string& outStr)` | add a new student to the class |
| `bool addAcademicYear(const string& start, string& outStr)` | add a new academic year |
| `bool addSemester(AcademicYear& newYear, const string& semesterID, string& outStr)` | add a new semester to the academic year |
| `bool addCourse(Semester& semester, const string& courseID, string& outStr)` | add a new course to the semester |
| `bool addStudentToCourse(Course& course, const string& studentID, string& outStr)` | add a new student to the course |

## 6. Remove functions

The following functions remove data and free memory in the process.

| Function | Description |
|---|---|
| `bool removeListStudent()` | remove the list of students |
| `bool removeListStaff()` | remove the list of staves |
| `bool removeListSchoolYear()` | remove the list of school year |
| `bool removeListAcademicYear()` | remove the list of academic years |
| `bool freeMemory()` | remove everthing |
| `bool removeStudent(const string& studentID, string& outStr)` | remove a student from school |
| `bool removeStaff(string curStaffID, const string& staffID, string&` | remove a staff from |

| | |
|---|---|
| `outStr)` | school |
| `bool removeSchoolYear(const string& start, string& outStr)` | remove a school year |
| `bool removeClass(SchoolYear& schoolYears, const string& className, string& outStr)` | remove a class |
| `bool removeStudentFromClass(Class& CLASS, const string& studentID, string& outStr)` | remove a student from the class |
| `bool removeAcademicYear(const string& start, string& outStr)` | remove an academic year |
| `bool removeSemester(AcademicYear& academicYear, const string& semesterID, string& outStr)` | remove a semester |
| `bool removeCourse(Semester& semester, const string& courseID, string& outStr)` | remove a course |
| `bool removeStudFromCourse(Course& course, const string& studentID, string& outStr)` | remove a student from the course |

## 7. Search functions

The following functions search the content inside the data.

| Function | Description |
|---|---|
| `Student* getStudent(const string& studentID)` | find a student |
| `Staff* getStaff(const string& staffID)` | find a staff |
| `SchoolYear* getSchoolYear(const string& start)` | find a school year |
| `AcademicYear* getAcademicYear(const string& start)` | find an academic year |
| `Class* getClass(SchoolYear& schoolYear, const string& className)` | find a class inside the school year |
| `Semester* getSemester(const string& semesterID)` | find a semester |
| `Semester* getSemester(AcademicYear& academicYear, const string& semesterID)` | find a semester inside the academic year |
| `Course* getCourse(Semester& semester, const string& courseID)` | find a course inside a semester |
| `Scoreboard* getScoreboard(Course& course, const string& studentID)` | find the scoreboard of the student inside his/her course |
| `bool isCorrectStaffAccount(const string& staffID, const string& password, string& outStr)` | check the validity of the staff account |
| `bool isCorrectStudentAccount(const string& studentID, const string& password, string& outStr)` | check the validity of the student account |

## 8. Sorting functions

The following functions are based on quicksort algorithm.

| | |
|---|---|

| Function | Description |
| --- | --- |
| `void sortStudentList(Vector<Student>& students, const int& left, const int& right)` | sort the list of students |
| `void sortStaffList(Vector<Staff>& staffs, const int& left, const int& right)` | sort the list of staves |
| `void sortSchoolYearList(Vector<SchoolYear>& schoolYears, const int& left, const int& right)` | sort the list of school year |
| `void sortAcademicYearList(Vector<AcademicYear>& academicYears, const int& left, const int& right)` | sort the list academic year |
| `void sortSemesters(Vector<Semester>& semesters, const int& left, const int& right)` | sort the list of semesters |
| `void sortClasses(Vector<Class>& classes, const int& left, const int& right)` | sort the list of classes |
| `void sortCourses(Vector<Course>& courses, const int& left, const int& right)` | sort the list of courses |
| `void sortStudentsInClass(Class& CLASS, const int& left, const int& right)` | sort the list of students inside a class |
| `void sortStudentsInCourse(Course& course, const int& left, const int& right)` | sort the list of students inside a course |

## 9. Update functions

The following functions update the data.

| Function | Description |
| --- | --- |
| `bool updateStudentIn4(Student& student, const string& ID, const string& firstName, const string& lastName, const string& genderStr, const string& birthStr, const string& socialID, const string& password, string& outStr)` | update the student information |
| `bool updateStaffIn4(Staff& staff, const string& ID, const string& firstName, const string& lastName, const string& password, string& outStr)` | update the staff information |
| `bool updateSchoolYear(SchoolYear& schoolYear, const string& newStartYear, string& outStr)` | update the school year information |
| `bool updateClass(Class& CLASS, const string& newClassName, string& outStr)` | update the class information |
| `bool updateAcademicYear(AcademicYear& academicYear, const string& newStartYear, string& outStr)` | update the academic year information |
| `bool updateSemester(Semester& semester, const string& semesterID, const string startDate, const string endDate, string& outStr)` | update the semester information |
| `bool updateCourse(Course& course, const string& courseID, const string& classID, const string& name, const string& teacher, const string& cre, const string& maxEnroll, const string& day, const string& ss, string &outStr)` | update the course |

| | |
|---|---|
| | information |
| `bool updateScoreboard(Course& course, const string& studentID, const string& midTerm, const string& other, const string& final, const string& total, string& outStr)` | update the student scoreboard |

### 10. Upload functions

The following functions upload the data from CSV files.

| Function | Description |
|---|---|
| `bool uploadAllData()` | upload all the data |
| `bool uploadListStaff()` | upload the list of staves |
| `bool uploadListStudent()` | upload the list of students |
| `bool uploadListSchoolYear()` | uplooad the list of school years |
| `bool uploadSchoolYear(SchoolYear& schoolYear)` | upload the school year |
| `bool uploadStudent(Class& actClass, Student& student, std::string id)` | upload the student to a class |
| `bool uploadClass(Class& actClass)` | upload the class |
| `bool uploadListAcademicYear()` | upload the list of academic years |
| `bool uploadAcademicYear(AcademicYear& academicYear)` | upload the academic year |
| `bool uploadSemester(Semester& semester)` | upload the semester |
| `bool uploadCourse(Course& course)` | upload the course |

## 3. Graphics / Objects

### Graphics

### 1. Constant namespace

(Define in Constant.h)

Contains constants for various elements used in a Course Management System. It includes constants for application, box, button, and text related components.

- **“app_const” namespace**: the constants are defined for the width, height, and frames per second of the application window. Additionally, the title of the window and the path of the application directory are also defined.

- **“box_const” namespace**: the constants are defined for the width, height, and thickness of the box, as well as the roundness and number of segments for the box corners. The fill and border color of the box are also defined.
- **“button_const” namespace**: the constants are defined for the colors of the button on hover and press.
- **“text_const” namespace**: the constants are defined for the font path, space between lines, size, and color of the text. The padding for the text is also defined.

## 2. Scissor

(Define in Scissor.h and implement in Scissor.cpp)

| Function | Description |
| --- | --- |
| `StartScissor(Rectangle rect)` | Starts a new scissor rectangle with the given `Rectangle` object. It first checks if there is any previously defined scissor rectangle and modifies the current rectangle to be the intersection of the previous and new rectangle. It then calls the `EndScissorMode()` function from the “raylib.h” library and begins a new scissor mode with the modified rectangle. The rectangle is also pushed onto a stack to keep track of the previous scissor rectangles. |
| `StartScissor(Vector2 pos, Vector2 size)` | Starts a new scissor rectangle with the given position and size as `Vector2` objects. It creates a new `Rectangle` object with these values and calls the `StartScissor(Rectangle rect)` function. |
| `StartScissor(float x, float y, float width, float height)` | Starts a new scissor rectangle with the given position and size as separate float values. It creates a new `Rectangle` object with these values and calls the `StartScissor(Rectangle rect)` function. |
| `EndScissor()` | Pops the top scissor rectangle off the stack and calls the `EndScissorMode()` function to end the current scissor mode. It then begins a new scissor mode with the previous scissor rectangle on top of the stack, if there is one. |

The source file “Scissor.cpp” defines the scissorStack variable as an external Stack object, which is used to keep track of the current scissor rectangle.

Overall, this Scissor class provides a convenient way to handle scissoring of graphics to a certain area in a 2D graphics program. It also allows for nested scissor rectangles, with the ability to modify the current rectangle to the intersection of the previous and new rectangle.

## 3. Application

(Define in Application.h and implement in Application.cpp)

An `Application` class that serves as the entry point for running the program. The class initializes the window and sets the target frame rate using the Raylib library.

| Attribute / Method | Description |
|---|---|
| `- Vector2 mousePoint` | A private member variable representing the current mouse position. |
| `- Scene\* scene` | A private member variable representing the current scene that the application is rendering. |
| `- void render()` | A private member function that renders the current scene. |
| `- void process()` | A private member function that updates the current scene. |
| `+ Application()` | A public constructor that initializes the application window and the scenes. |
| `+ ~Application()` | A public destructor that cleans up the application window and the scenes. |
| `+ Application(const Application &other) = delete;` | A public copy constructor that is disabled. |
| `+ Application& operator=(const Application &other) = delete;` | A public assignment operator that is disabled. |
| `+ bool shouldClose() const` | A public member function that returns true if the window should close, false otherwise. |
| `+ void run()` | A public member function that runs the application. |

(All objects is defined and implemented in directory *Graphics/Objects*)

## 1. Text

(Define in Text.h and implement in Text.cpp)

The `Text` class is a simple class used to create text objects. It contains attributes for the text content, font size, additional space between characters, font, and color. The `Text` object can be initialized using either a C-style string or an `std::string` object.

| Attribute / Method | Description |
|---|---|
| `+ std::string text` | The text content of the Text object. |
| `+ float font_size` | The size of the font used in the text. |
| `+ float space` | The additional space between |

| | |
|---|---|
| | characters. |
| `+ Font font` | The font used in the text. |
| `+ Color color` | The color of the text. |
| `+ Text()` | Default constructor. |
| `+ Text(const char* text, float fsize = text_const::size, Font font = LoadFontEx(text_const::font_path.c_str(), 128, 0, 0), float space = text_const::space, Color color = text_const::color)` | Constructor that initializes the Text object with the given values. |
| `+ Text(std::string text, float fsize = text_const::size, Font font = LoadFontEx(text_const::font_path.c_str(), 128, 0, 0), float space = text_const::space, Color color = text_const::color)` | Constructor that initializes the Text object with the given values. |
| `+ void operator=(std::string text)` | Overloaded assignment operator that sets the text content of the Text object to the given string. |
| `+ void operator=(const char* text)` | Overloaded assignment operator that sets the text content of the Text object to the given C-style string. |
| `+ Vector2 size()` | Returns a Vector2 containing the width and height of the Text object based on the current font and font size. |

## 2. TextBox

(Define in TextBox.h and implement in TextBox.cpp)

The `TextBox` class represents a text box that can be rendered on the screen using `raylib`. The class provides various methods to set and get properties of the text box, and also to render it on the screen.

| Attribute / Method | Description |
|---|---|
| - `Vector2 pos` | The position of the top-left corner of the text box |
| - `Text content` | The content of the text box (i.e. the actual text to be displayed) |
| - `Rectangle bound` | The rectangular bounds of the text box (including any padding or border) |
| - `Color color_box` | The color of the text box's background |
| + `TextBox()` | Default constructor that initializes an empty text box with position at the origin and default background color |
| | |

| | |
|---|---|
| + `TextBox(Text content, Vector2 pos = {0, 0}, Color color_box = box_const::fill_color)` | Constructor that initializes a text box with the specified content, position, and background color |
| + `TextBox(std::string text, Vector2 pos = {0, 0}, Color color_box = box_const::fill_color)` | Constructor that initializes a text box with the specified text string, position, and background color |
| + `TextBox(const char* text, Vector2 pos = {0, 0}, Color color_box = box_const::fill_color)` | Constructor that initializes a text box with the specified text C-string, position, and background color |
| + `TextBox& operator=(Text content)` | Copy assignment operator that sets the content of the text box |
| + `TextBox& operator=(std::string text)` | Copy assignment operator that sets the content of the text box from a string |
| + `TextBox& operator=(const char* text)` | Copy assignment operator that sets the content of the text box from a C-string |
| + `TextBox& operator + (const std::string& text)` | Concatenation operator that appends a string to the content of the text box |
| + `void setContent(const std::string& content)` | Sets the content of the text box from a string |
| + `void setContent(const Text& content)` | Sets the content of the text box from a `Text` object |
| + `std::string& getContent()` | Returns a reference to the content of the text box |
| + `void setX(float x)` | Sets the x-coordinate of the position of the text box |
| + `void setY(float y)` | Sets the y-coordinate of the position of the text box |
| + `void setPos(Vector2 pos)` | Sets the position of the text box |
| + `Vector2 getPos()` | Returns the position of the text box |
| + `void centerX()` | Centers the text box horizontally on the screen |
| + `void centerY()` | Centers the text box vertically on the screen |
| + `void setSize(float size)` | Sets the font size of the text box |
| + `void setColor(Color color)` | Sets the background color of the text box |
| + `void render()` | Renders the text box on the screen |
| + `void clear()` | Clears the content of the text box |

## 3. InputBox

(Define in InputBox.h and implement in InputBox.cpp)

The `InputBox` class provides a customizable input box to get text input from users. It contains several properties such as the text content, position, size, border and fill

colors. It also supports several input events such as hovering and clicking.

| Attribute/Method | Description |
| --- | --- |
| `+defaultText: Text` | The default text displayed inside the InputBox |
| `+content: Text` | The text content of the InputBox |
| `+fill_color: Color` | The color of the fill inside the InputBox |
| `+border_color: Color` | The color of the border of the InputBox |
| `+hover_color: Color` | The color of the InputBox when the mouse is hovering over it |
| `+press_color: Color` | The color of the InputBox when it is being clicked |
| `+pos: Vector2` | The position of the InputBox |
| `+size: Vector2` | The size of the InputBox |
| `+selected: bool` | A flag indicating whether the InputBox is currently selected by the user |
| `+frameCount: int` | The frame count since the creation of the InputBox |
| `+refreshText(): void` | Recalculates the size and position of the text displayed inside the InputBox |
| `+refresh(): void` | Recalculates the position and size of the InputBox |
| `+InputBox()` | Constructs an InputBox with default values |
| `+InputBox(x: float, y: float, width: float, height: float, text: string, fill: Color, hover: Color, press: Color, border: Color)` | Constructs an InputBox with the given parameters |
| `+InputBox(pos: Vector2, size: Vector2, text: string, fill: Color, hover: Color, press: Color, border: Color)` | Constructs an InputBox with the given parameters |
| `+render(mouse: Vector2): void` | Renders the InputBox |
| `+process(mouse: Vector2): void` | Processes input events for the InputBox |
| `+setX(x: float): void` | Sets the x position of the InputBox |
| `+setY(y: float): void` | Sets the y position of the InputBox |

| | |
|---|---|
| `+setWidth(width: float): void` | Sets the width of the InputBox |
| `+setHeight(height: float): void` | Sets the height of the InputBox |
| `+setPos(pos: Vector2): void` | Sets the position of the InputBox |
| `+setSize(size: Vector2): void` | Sets the size of the InputBox |
| `+getPos(): Vector2` | Returns the position of the InputBox |
| `+getSize(): Vector2` | Returns the size of the InputBox |
| `+clearContent(): void` | Clears the text content of the InputBox |
| `+centerX(): void` | Centers the InputBox horizontally |
| `+centerY(): void` | Centers the InputBox vertically |
| `+getContent(): string` | Returns the text content of the InputBox |
| `+clicked(mouse: Vector2): bool` | Returns `true` if the InputBox was clicked |
| `+pressed(mouse: Vector2): bool` | Returns `true` if the InputBox was being clicked |
| `+hovering(mouse: Vector2): bool` | Returns `true` if the mouse is hovering over the InputBox |

## 4. Button

(Define in Button.h and implement in Button.cpp)

The `Button` class is used to create clickable buttons in graphical user interface (GUI) applications. The class is defined in the `Button.h` header file and implemented in the `Button.cpp` source file.

The `Button` class inherits from the `Text` class, which represents a block of text that can be rendered on the screen. It has a `label` property that stores the text to be displayed on the button, and it can adjust the font size of the text to fit inside the button's bounds.

| Attribute / Method | Description |
|---|---|
| `- Vector2 pos` | The position of the button. |
| `- Vector2 size` | The size of the button. |
| `- Vector2 textPos` | The position of |

| | |
|---|---|
| | the text label inside the button. |
| `- Rectangle border_bound` | The bounding rectangle of the button’s border. |
| `- Rectangle fill_bound` | The bounding rectangle of the button’s fill. |
| `- Rectangle text_bound` | The bounding rectangle of the button’s text. |
| `- virtual void refreshText()` | Calculates the font size and position of the text label. |
| `- virtual void refresh()` | Calculates the bounding rectangles of the button. |
| `+ Text label` | The text label to be displayed on the button. |
| `+ float roundness` | The roundness of the button’s corners. |
| `+ Color fill_color` | The fill color of the button. |
| `+ Color border_color` | The border color of the button. |
| `+ Color hover_color` | The fill color of the button when the mouse is hovering over it. |
| `+ Color press_color` | The fill color of the button when it is being pressed. |
| `+ Button()` | Constructor for an empty button |

| | |
|---|---|
| | with default properties. |
| `+ Button(float x, float y, float width, float height, std::string text = "", float roundness = box_const::roundness, Color fill = box_const::fill_color, Color hover = button_const::hover_color, Color press = button_const::press_color, Color border = box_const::border_color)` | Constructor for a button with custom properties. |
| `+ Button(Vector2 pos, Vector2 size, std::string text = "", float roundness = box_const::roundness, Color fill = box_const::fill_color, Color hover = button_const::hover_color, Color press = button_const::press_color, Color border = box_const::border_color)` | Constructor for a button with custom properties. |
| `+ virtual void render(const Vector2 &mouse) const` | Renders the button on the screen. |
| `+ void setX(float x)` | Sets the x-coordinate of the button’s position. |
| `+ void setY(float y)` | Sets the y-coordinate of the button’s position. |
| `+ void setWidth(float width)` | Sets the width of the button. |
| `+ void setHeight(float height)` | Sets the height of the button. |
| `+ void setPos(Vector2 pos)` | Sets the position of the button. |
| `+ void setSize(Vector2 size)` | Sets the size of the button. |
| `+ Vector2 getPos()` | Gets the position of the button. |
| `+ Rectangle getRec()` | Gets the rectangle of the button. |
| `+ void setViewColor()` | Sets color of the button. |
| `+ void setRemoveColor()` | Sets color of the button. |
| `+ void setInsertColor()` | Sets color of the button. |

| | |
|---|---|
| `+ centerX(): void` | Centers the Button horizontally |
| `+ centerY(): void` | Centers the Button vertically |
| `+ clicked(mouse: Vector2): bool` | Returns `true` if the Button was clicked. |
| `+ pressed(mouse: Vector2): bool` | Returns `true` if the Button was being clicked. |
| `+ hovering(mouse: Vector2): bool` | Returns `true` if the mouse is hovering over the Button. |

## 5 Option

(Define in Option.h and implement in Option.cpp)

This class `Option` extends the `Button` class and includes the `Scissor` and `raylib` libraries. It is used to create an option that can be clicked and interacted with. It has protected attributes for the left and right padding of the option, and public methods for getting these values, refreshing the text and bounds of the option, and rendering the option.

| Attribute / Method | Description |
|---|---|
| `# float right_padding` | Right padding of the option. |
| `# float left_padding` | Left padding of the option, default is `text_const::padding`. |
| `# void refreshText()` | Refreshes the text size and position of the label. |
| `# void refresh()` | Refreshes the bounds and text of the option. |
| `+ float getLeftPad()` | Returns the left padding of the option. |
| `+ float getRightPad()` | Returns the right padding of the option. |
| `+ void render(const Vector2 &mouse) const` | Renders the option with the given mouse position. |

## 6. Equilateral

(Define in Equilareral.h and implement in Equilateral.cpp)

The `Equilateral` class represents an equilateral triangle with a specified center, side length, angle, and color.

| Attribute/Method | Description |
|---|---|
| - `float rootInv3` | A constant float value equal to the inverse square root of 3. |
| `public` | |
| + `Vector2 center` | The center point of the equilateral triangle. |
| + `float length` | The length of one side of the equilateral triangle. |
| + `float angle` | The angle of rotation of the equilateral triangle in degrees. |
| + `Color color` | The color of the equilateral triangle. |
| + `Equilateral()` | The default constructor of the Equilateral class, which initializes all attributes to their default values. |
| + `Equilateral(Vector2 center, float length, float angle, Color color)` | The constructor of the Equilateral class, which initializes the attributes with the given values. |
| + `void render()` | A method to render the equilateral triangle on the screen. It uses the raylib library's `DrawTriangle()` function to draw the triangle on the screen. |

## 7. Scrollbar

(Define in Scrollbar.h and implement in Scrollbar.cpp)

The `Scrollbar` class represents a user interface component that allows scrolling through a large content area that doesn't fit within a smaller window.

| Attribute / Method | Description |
|---|---|
| - `Vector2 pos` | The position of the scrollbar. |
| - `Vector2 cur_pos` | The current position of the scrollbar. |
| - `Vector2 last_mouse` | The position of the mouse during the last update. |
| - `float len` | The length of the scrollbar. |
| - `bool pressing` | A boolean value indicating whether the scrollbar is currently being pressed. |
| + `float pos_min` | The minimum position value for the scrollbar. |
| + `float pos_max` | The maximum position value |

| | |
|---|---|
| | for the scrollbar. |
| + `float thickness` | The thickness of the scrollbar. |
| + `float content_len` | The length of the content to be scrolled through. |
| + `float content_max_len` | The maximum length of the content to be scrolled through. |
| + `Color fill` | The color of the scrollbar. |
| + `Color press` | The color of the scrollbar when it is being pressed. |
| + `bool horizontal` | A boolean value indicating whether the scrollbar is horizontal. |
| + `Scrollbar()` | Default constructor for the Scrollbar class. |
| + `Scrollbar(Vector2 pos, float pos_min, float pos_max, float content_len, float content_max_len, float thickness, bool horizontal, Color fill, Color press)` | Constructor for the Scrollbar class. |
| + `Rectangle getRect()` | Returns a rectangle representing the current position and size of the scrollbar. |
| + `void setPos(Vector2 newPos)` | Sets the position of the scrollbar. |
| + `float content_height()` | Returns the height of the content that can be displayed by the scrollbar. |
| + `bool clicked(const Vector2 &mouse)` | Returns true if the scrollbar has been clicked by the mouse. |
| + `bool pressed(const Vector2 &mouse)` | Returns true if the scrollbar is currently being pressed by the mouse. |
| + `bool currentlyPressed(const Vector2 &mouse)` | Returns true if the scrollbar is currently being pressed. |
| + `void render(const Vector2 &mouse)` | Renders the scrollbar. |
| + `void process(const Vector2 &mouse)` | Processes the scrollbar for the current frame. |

## 8. DropBox

(Define in DropBox.h and implement in DropBox.cpp)

The `DropBox` class represents a graphical user interface (GUI) component that displays a list of options and allows the user to select one of them. This class provides methods to add, remove, and reset the options, as well as to set the position, size, and label of the component.

| Attribute / Method | Description |
|---|---|
| `- float textSize` | Font size of the label inside the dropdown box. |
| `- bool selected` | Indicates whether the dropdown box is currently selected. |
| `- int curIndex` | Index of the currently selected option. If no option is selected, `curIndex` is set to -1. |
| `- Option current` | The currently selected option. |
| `- Scrollbar bar` | The scrollbar object used in the dropdown box. |
| `- Vector2 pos` | The position of the dropdown box. |
| `- Vector2 size` | The size of the dropdown box. |
| `- Rectangle bound` | The bounding rectangle of the dropdown box, including all the options. |
| `- Vector<Option> options` | The options in the dropdown box. |
| `+ float roundness` | The roundness of the border of the dropdown box. |
| `+ Color fill_color` | The fill color of the dropdown box. |
| `+ Color border_color` | The border color of the dropdown box. |
| `+ Color hover_color` | The hover color of the options in the dropdown box. |
| `+ Color press_color` | The press color of the options in the dropdown box. |
| `+ Color text_color` | The color of the text in the dropdown box. |
| `+ Equilateral arrow` | The arrow object used in the dropdown box. |
| `+ DropBox()` | Constructor of the `DropBox` class. Initializes `curIndex`, `selected`, `current`, `pos`, and `size`. |
| `+ void refresh()` | Updates the position, size, and other properties of the dropdown box and all its options. |
| `+ void setLabel(std::string label)` | Sets the label of the currently selected option. |
| `+ void setX(float x)` | Sets the x-coordinate of the position of the dropdown box. |
| `+ void setY(float y)` | Sets the y-coordinate of the position of the dropdown box. |
| `+ void setWidth(float width)` | Sets the width of the dropdown box. |
| `+ void setHeight(float height)` | Sets the height of the dropdown box. |

| | |
|---|---|
| `+ void setPos(Vector2 pos)` | Sets the position of the dropdown box. |
| `+ void setSize(Vector2 size)` | Sets the size of the dropdown box. |
| `+ void add(const std::string &label)` | Adds an option to the dropdown box with the given label. |
| `+ void add(const Vector<std::string> &labels)` | Adds multiple options to the dropdown box with the given labels. |
| `+ string getCurLabel() const` | Returns the label of the currently selected option. |
| `+ void remove(const std::string label)` | Removes the option with the given label from the dropdown box. |
| `+ void reset()` | Resets the dropdown box by removing all the options except the default option. |
| `+ void clear()` | Clears the dropdown box by removing all the options. |
| `+ int getSelected()` | Returns the index of the currently selected option. If no option is selected, returns -1. |
| `+ void render(const Vector2 &mouse)` | enders the component to the screen. |
| `+ bool process(const Vector2 &mouse)` | Processes user input and returns true if the user has selected an option. |

# 4. Graphics / Scene

## 1. Scene

(Defined in Scene.h)

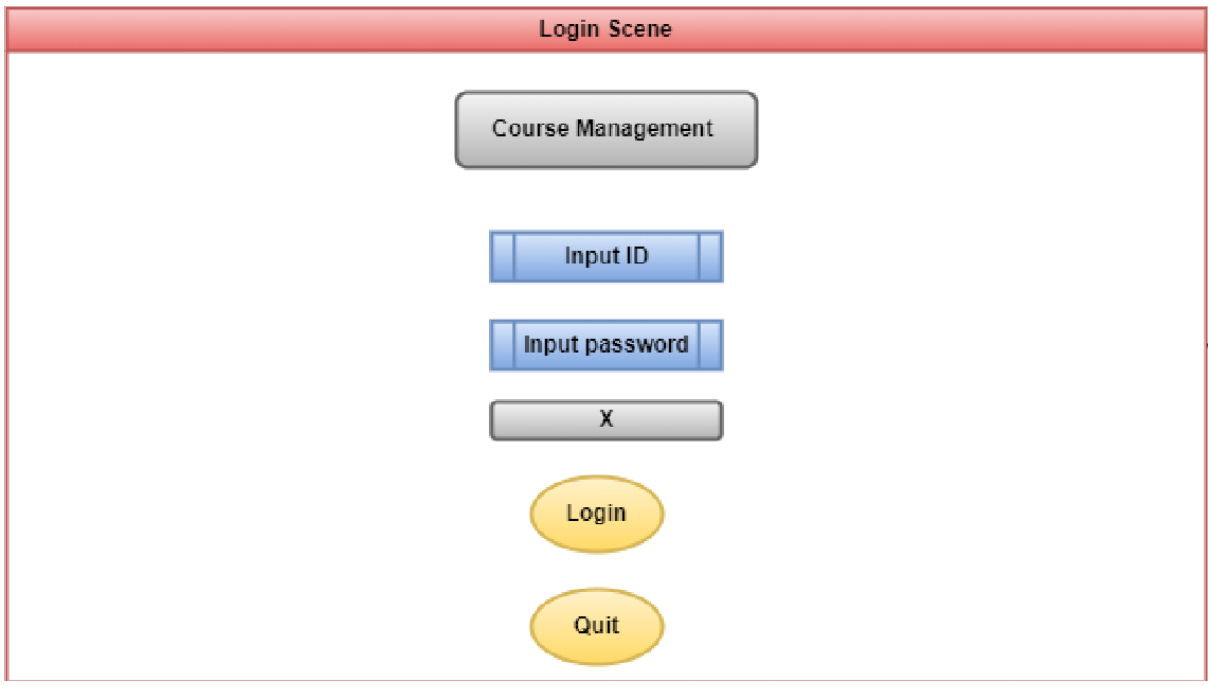
`Scene` is a base class that can be inherited by the derived classes, these classes inherite the attributes and methods in `Scene` class, which is used to create specific scenes.

| Attribute / Method | Description |
|---|---|
| `Vector2 mousePoint` | The protected attribute locate the mouse cursor position in the scene. |
| `virtual void render()` | Renders all the graphic objects to the scene. |
| `virtual Scene* process()` | Updates and returns the scene to the application. |

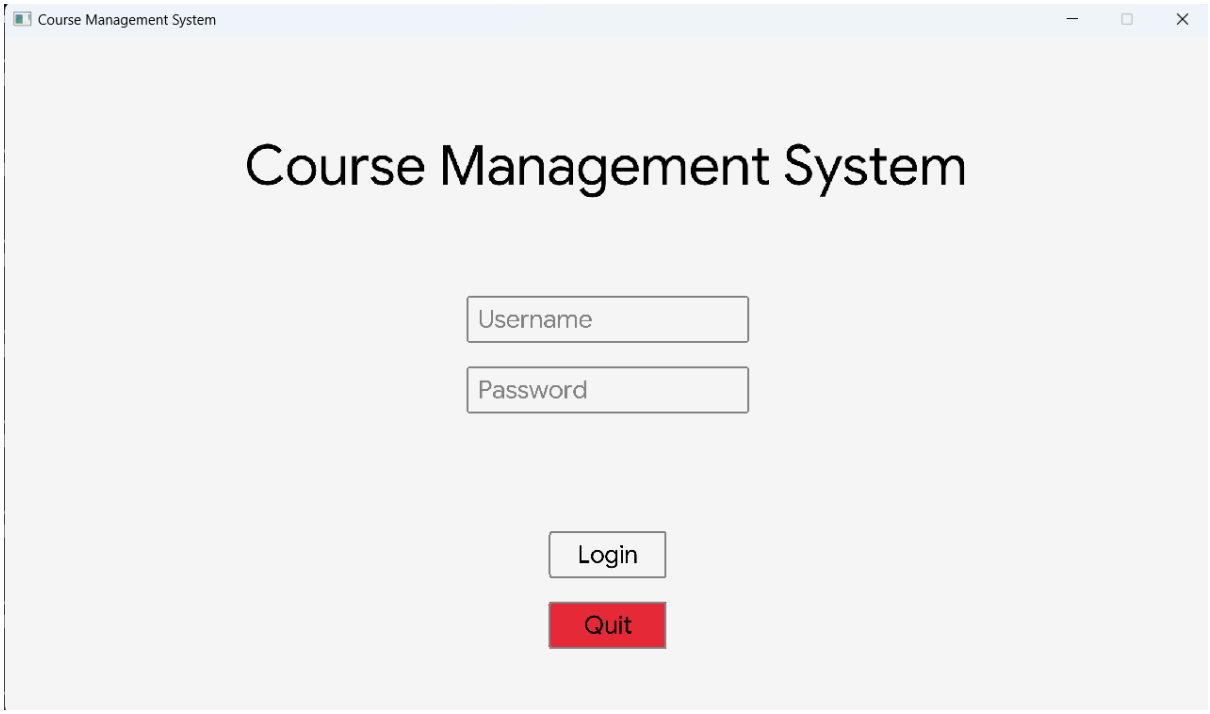
## 2. Login

```
(Define in Login.h and implemented in Login.cpp)
```

Theoretic scene




Practical scene



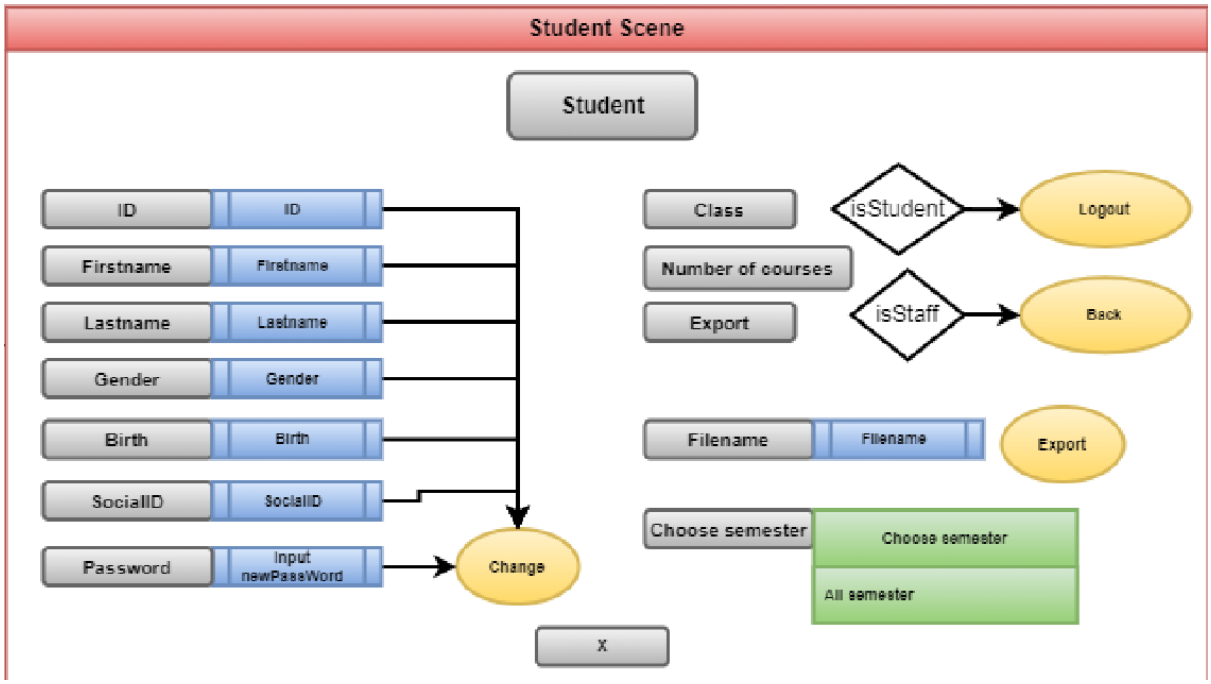

`Login` is a derived class inherited from `Scene` class used to create a login scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
| --- | --- |
| `Login()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `login` button, if the username and password are valid, it will return the next scene (Student Scene / Staff Scene). Otherwise the warning text will occur. + Click on `Quit` button, the application will be closed. |

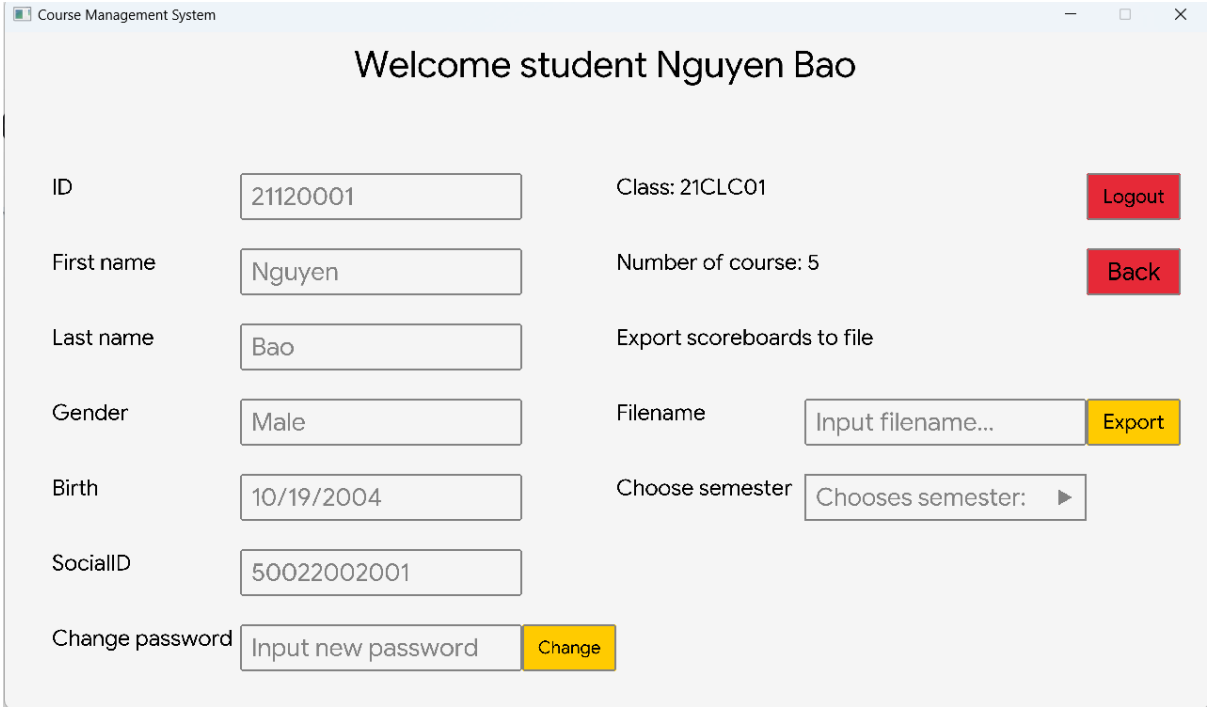
## 3. StudentScene

```
(Defined in StudentScene.h and implemented in StudentScene.cpp)
```

Theoretic scene




Practical scene



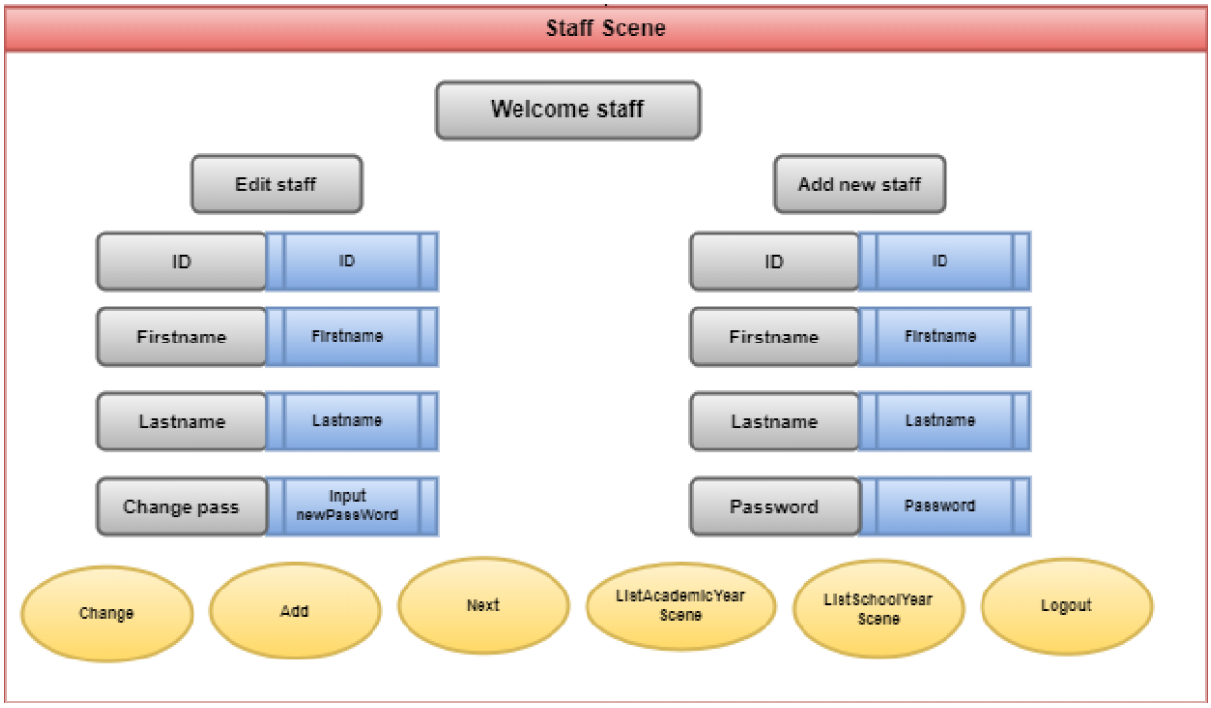

`StudentScene` is a derived class inherited from `Scene` class used to create a student scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
|---|---|
| `StudentScene()` | The constructor initializes all private graphic objects. |
| `void render()` | Renders all the objects implemented in the private class to the student scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: +Click on `Change` button, if the inputs are valid, it will access to `updateStudentIn4()` function, then the information will be updated and return the current scene (Student Scene). Otherwise the warning text will occur. +Click on `Export` button, if the input in the file name input box is valid, it will access to the `exportListScoreboardOfStudnet()` function, then a `.csv` file will be exported. In addition, if user choose a semester from semester drop box, then it will access to the `exportListScoreboardInSemesterOfStudent()`, then a `.csv` file will be exported. Otherwise, the warning text will occur. It also returns the current scene. +Click on `Back` button (it only appears when user access this scene through `StaffScene2` scene, the application will turn to `StaffScene2` scene. +Click on `Logout` button, the application will turn to `Login` scene. |

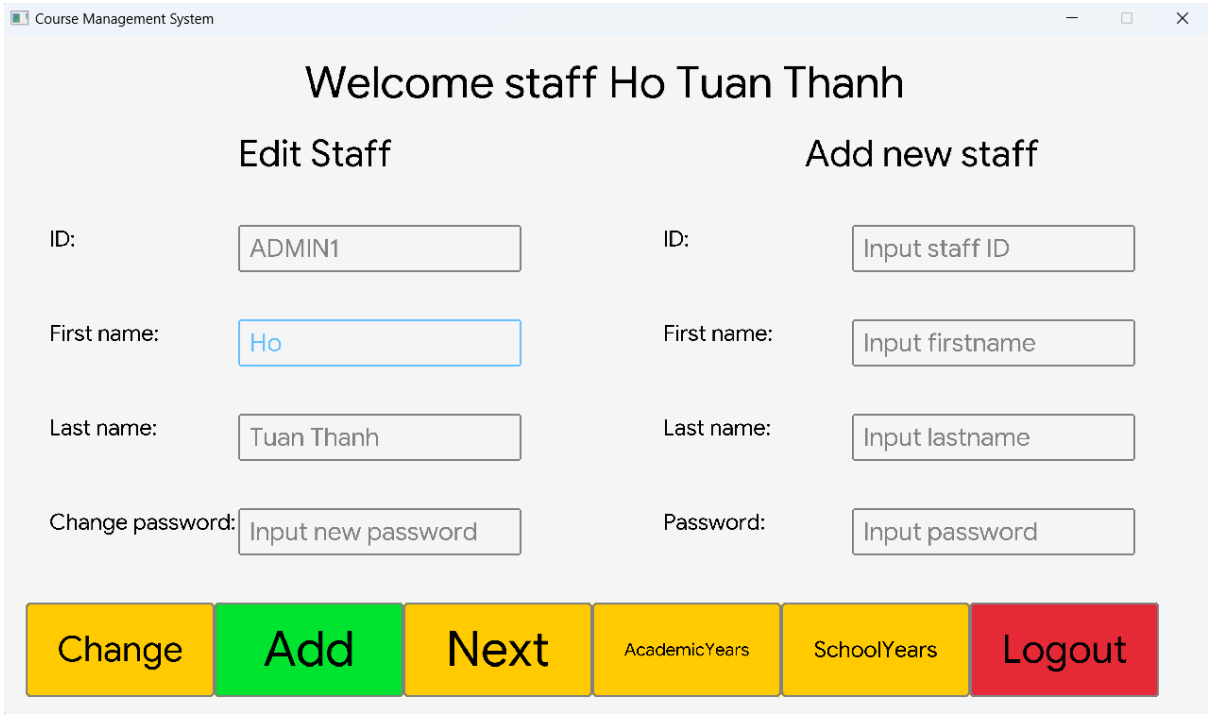
## 4. StaffScene

```
(Defined in StaffScene.h and implemented in StaffScene.cpp)
```

Theoretic scene




Practical scene



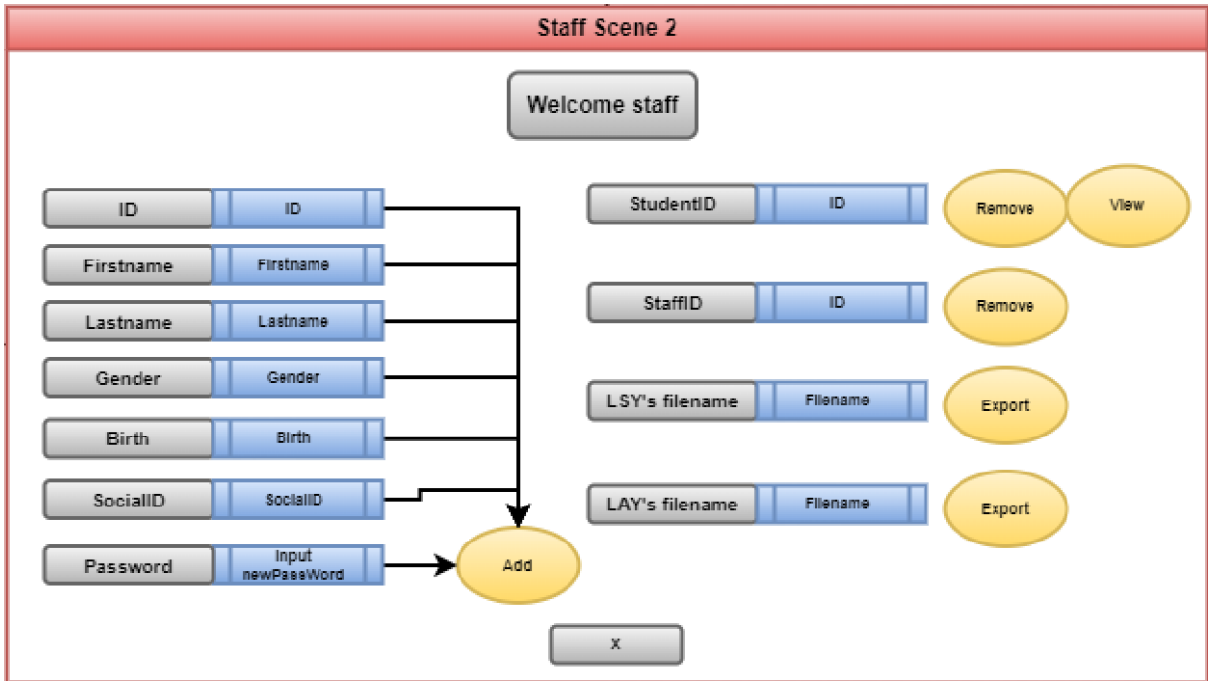
`StaffScene` is a derived class inherited from `Scene` class used to create a staff scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
|---|---|
| `StaffScene()` | The constructor initializes all private graphic objects. |
| `void render()` | Renders all the objects implemented in the private class to the student scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: +Click on `Change` button, if the inputs in `Edit Staff` tab are valid, it will access to `updateStaffIn4()` function, then the information will be updated and return the current scene (Staff Scene). Otherwise the warning text will occur. +Click on `Add` button, if the inputs in `Add new staff` tab are valid, it will access to `addStaff()` function, then a new staff will be added and it returns the current scene. Otherwise, the warning text will occur. +Click on `Next` button, the application will turn to the `StaffScene2` scene. +Click on `SchoolYears` button, the application will turn to `ListSchoolYearScene` scene. +Click on `AcademicYears` button, the application will turn to `ListAcademicYearScene` scene. +Click on `Logout` button, the application will turn to `Login` scene. |

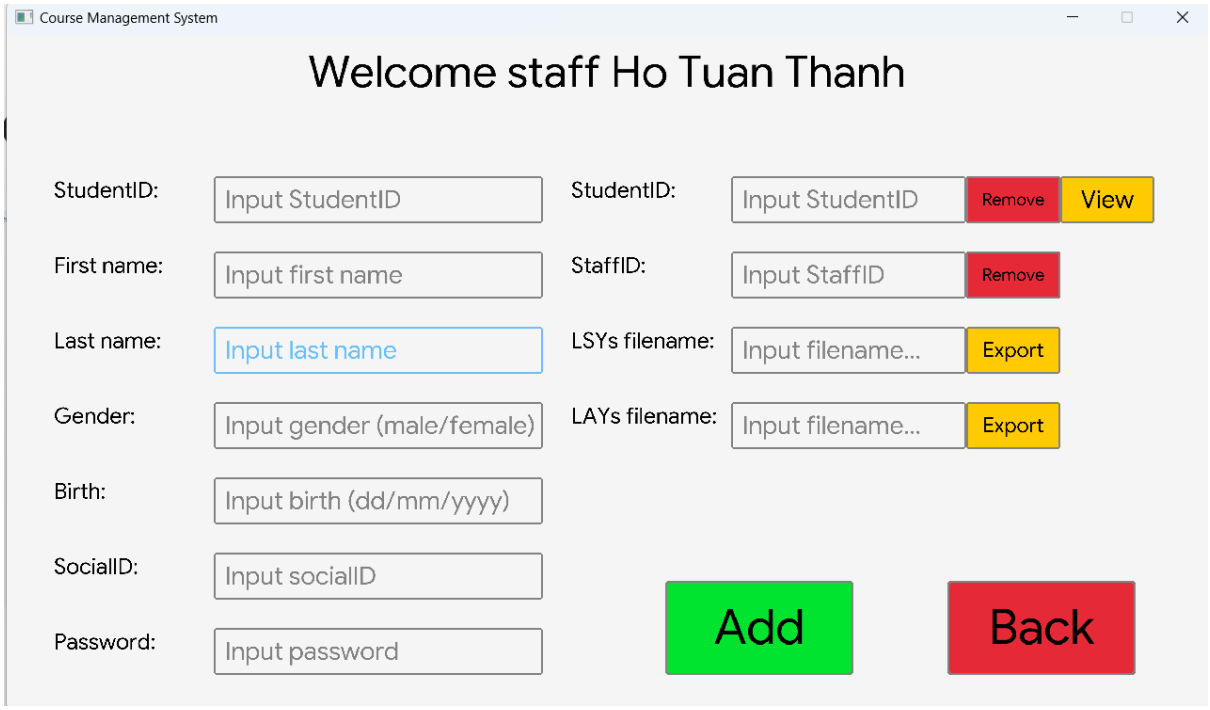
## 5. StaffScene2

```
(Defined in StaffScene2.h and implemented in StaffScene2.cpp)
```

Theoretic scene




Practical scene




`StaffScene2` is a derived class inherited from `Scene` class used to create a additional staff scene for the application. All the members in the private class are the variables for the objects for the scene.

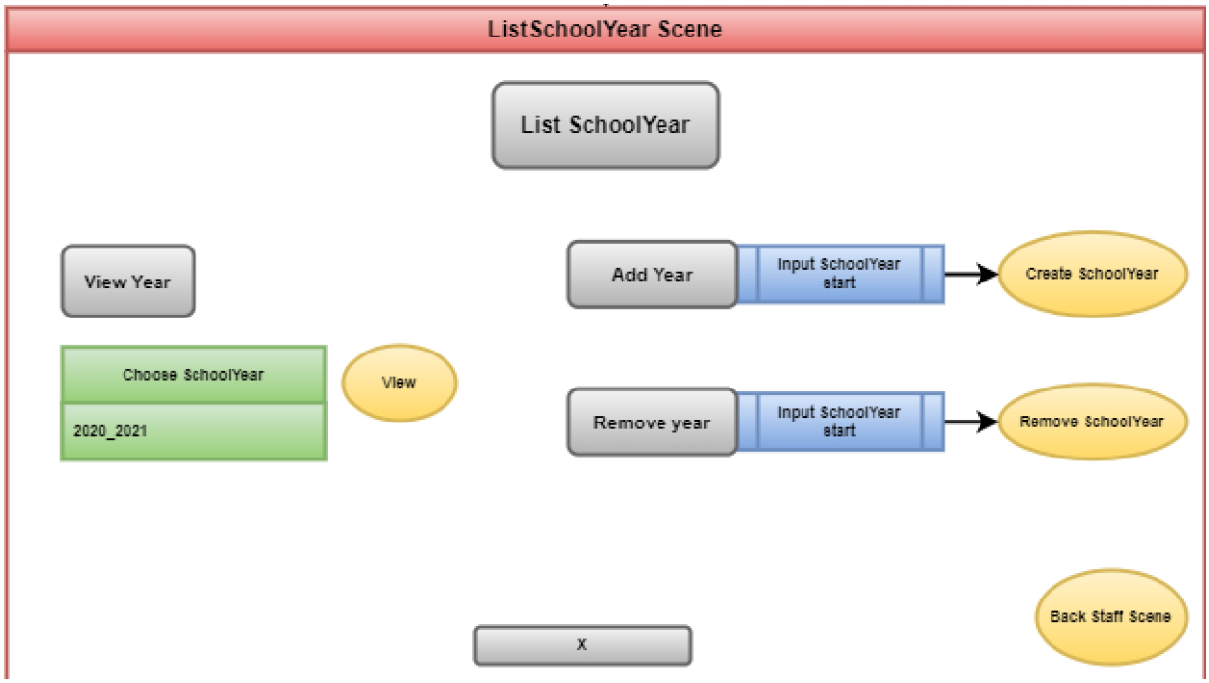
| Attribute / Method | Description |
| --- | --- |

| | |
|---|---|
| `StaffScene2()` | The constructor initializes all private graphic objects. |
| `void render()` | Renders all the objects implemented in the private class to the student scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: +Click on `Add` button, if the inputs of student's information are valid, it will access to `addStudent()` function, then a new student will be added and it returns the current scene (Staff Scene). Otherwise the warning text will occur. +Click on `Remove` button next to the `inputStudentID` input box, if the input is valid, it will access to `removeStudent()` function, then that student will be removed from school and it returns the current scene. Otherwise, the warning text will occur. +Click on `Remove` button next to `inputStaffID` input box, if the input is valid, it will access to `removeStaff()` function, then that staff will be removed and it returns the current scene. Otherwise, the warning text will occur. +Click on `Export` button next to `inputLSYsFilePath` input box, it will access to `exportListSchoolYear()` function, then a `.csv` file will be exported and the application returns the current scene. Otherwise, the warning text will occur. +Click on `Export` button next to `inputLAYsFilePath` input box, it will access to `exportListAcademicYear()` function, then a `.csv` file will be exported and the application returns the current scene. Otherwise, the warning text will occur. +Click on `View` button, if the input in the `inputStudnetID` is valid, the application will turn to `StudentScene` scene. Otherwise, the warning text will occur. +Click on `Back` button, the application will turn to `StaffScene` scene. |

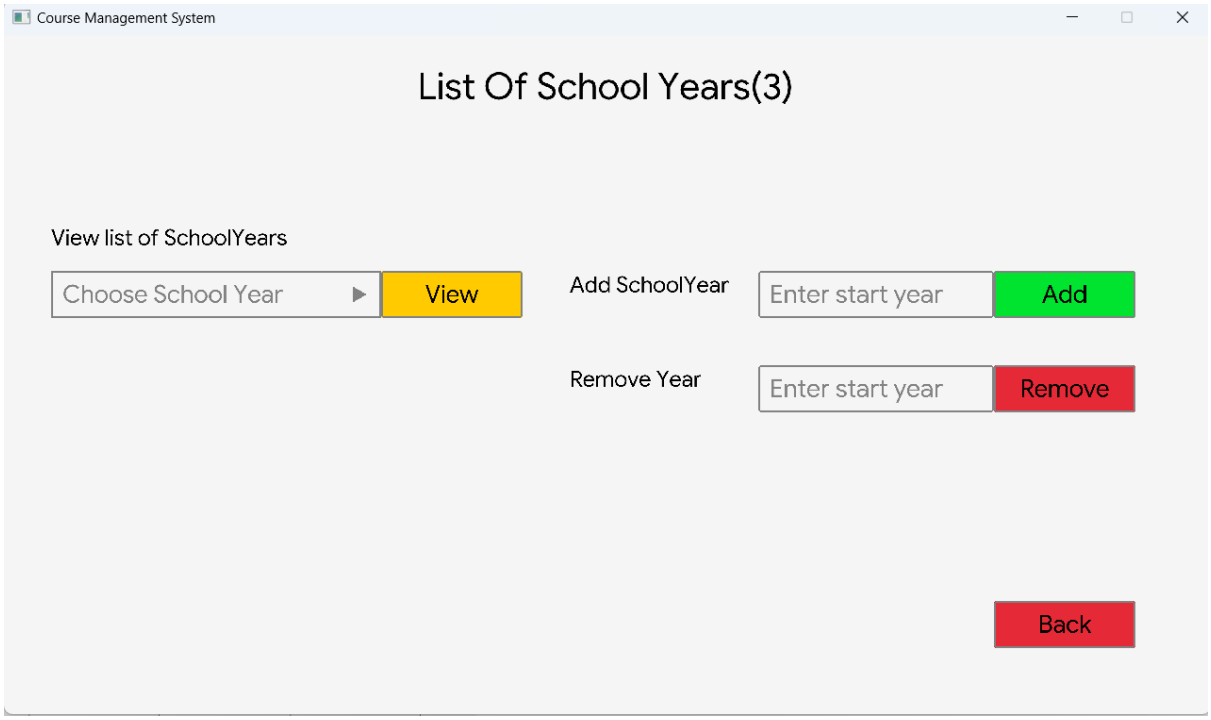
## 6. ListSchoolYearScene

```
(Defined in ListSchoolYearScene.h and implemented in ListSchoolYearScene.cpp)
```

Theoretic scene




Practical scene




`ListSchoolYearScene` is a derived class inherited from `Scene` class used to create a list school year scene for the application. All the members in the private class are the variables for the objects for the scene.

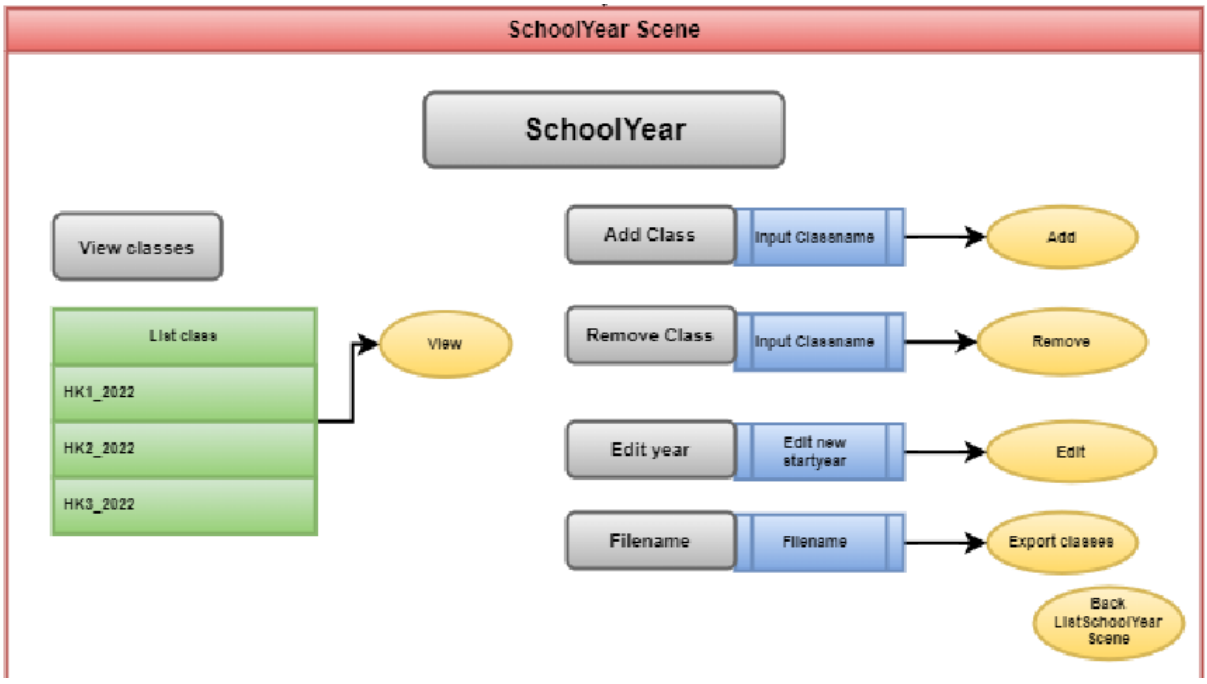
| Attribute / Method | Description |
| --- | --- |
| `ListSchoolYearScene()` | The constructor initializes all private graphic objects above. |

| | |
|---|---|
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a start year is valid, the program will access to `addSchoolYear()` function, then a school year will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a start year is valid, the program will access to `removeSchoolYear()` then a school year will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the school year drop box, then the function will return `SchoolYearScene` scene. +Click on `Back` button, the application will turn to `StaffScene` scene. |

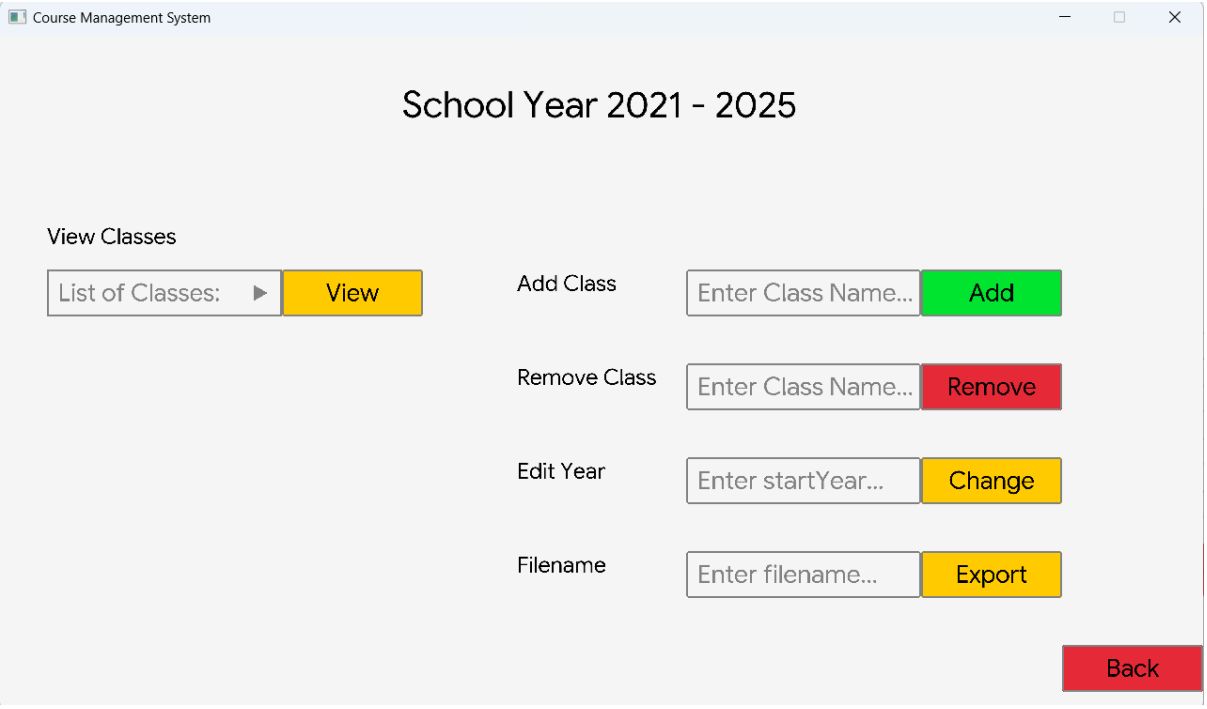
## 7. SchoolYearScene

```
(Defined in SchoolYearScene.h and implemented in SchoolYearScene.cpp)
```

Theoretic scene




Practical scene




`SchoolYearScene` is a derived class inherited from `Scene` class used to create a school year scene for the application. All the members in the private class are the variables for the objects for the scene.

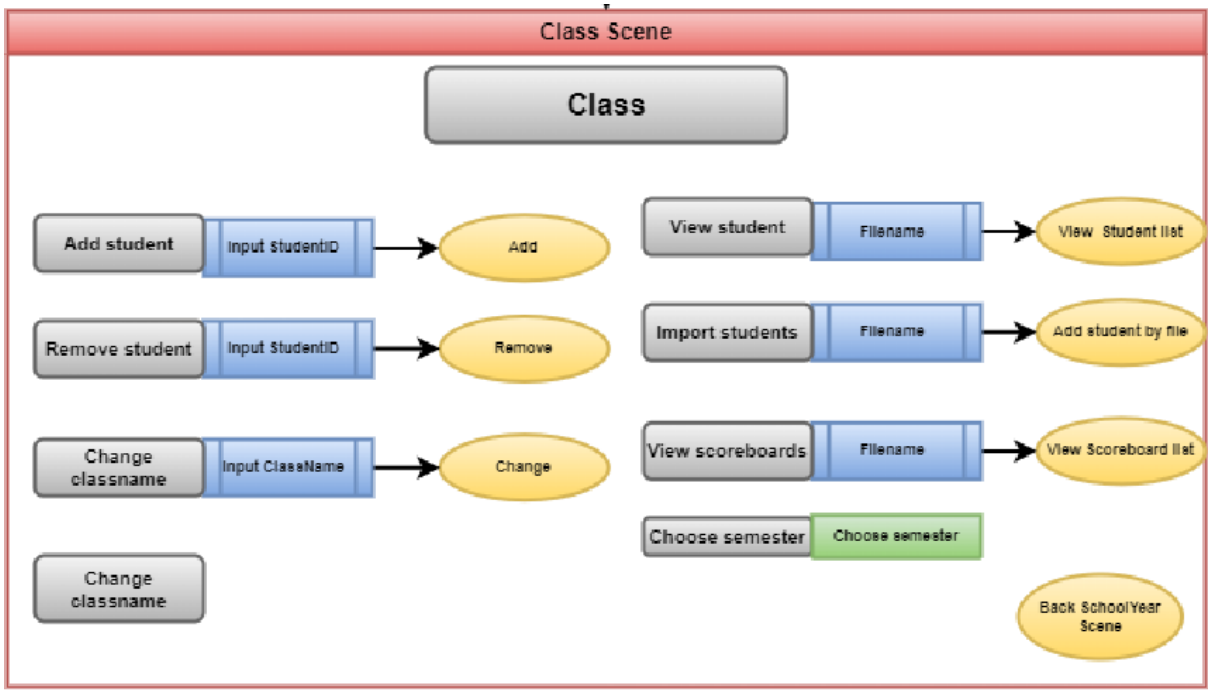
| Attribute / Method | Description |
|---|---|
| `SchoolYearScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a class name is valid, the program will access to `addSemester()` function, then a class will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a class name is valid, the program will access to `removeSemester()`, then a class will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the class drop box, then the function will return `SemesterScene` scene. +Click on `Change` button, if the start year is valid, the program will access to `updateAcademicYearYear()`, then this schoolYear will be updated and the program will return this scene. Otherwise the warning text will occur. +Click on `Export` button, if the exported file name is valid, the program will access to `exportListSemesterInAcademicYear()`, then a `.csv` file will be exported and the function returns current scene. Otherwise, the warning text will occur. |

| | |
|---|---|
| | +Click on `Back` button, the application will turn to `ListAcademicYearScene` scene |

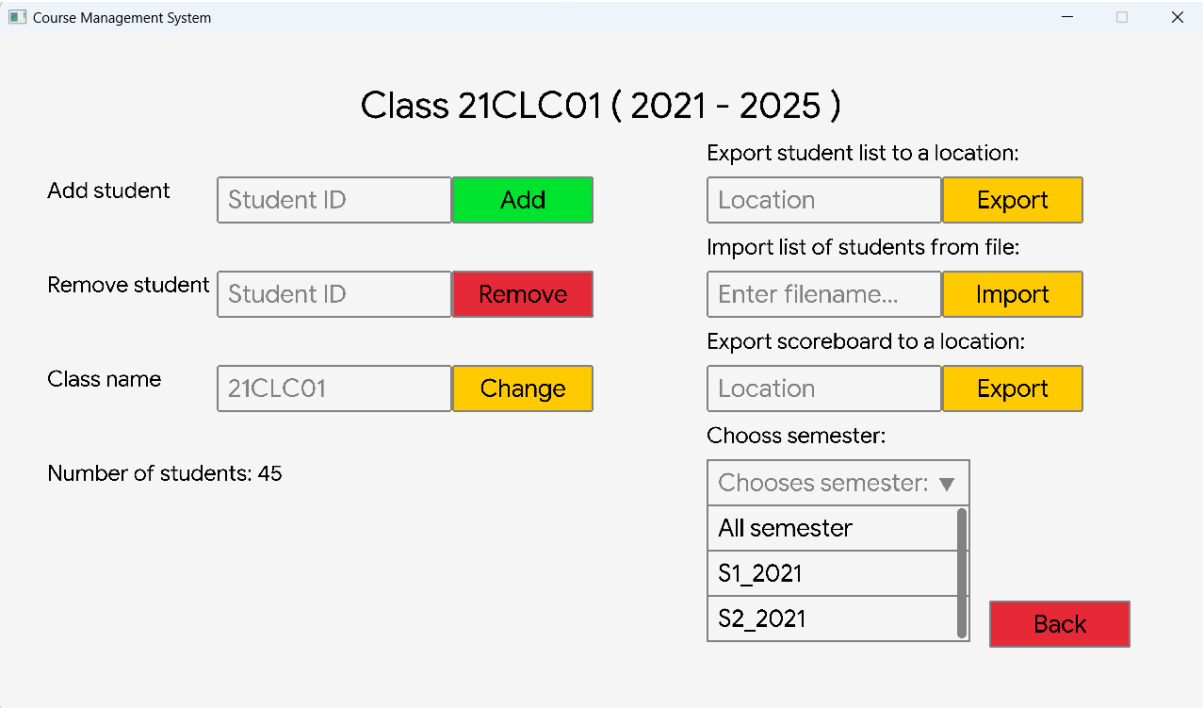
## 8. ClassScene

```
(Defined in ClassScene.h and implemented in ClassScene.cpp)
```

Theoretical scene




Practical scene




`ClassScene` is a derived class inherited from `Scene` class used to create a class scene for the application. All the members in the private class are the variables for the objects for the scene.

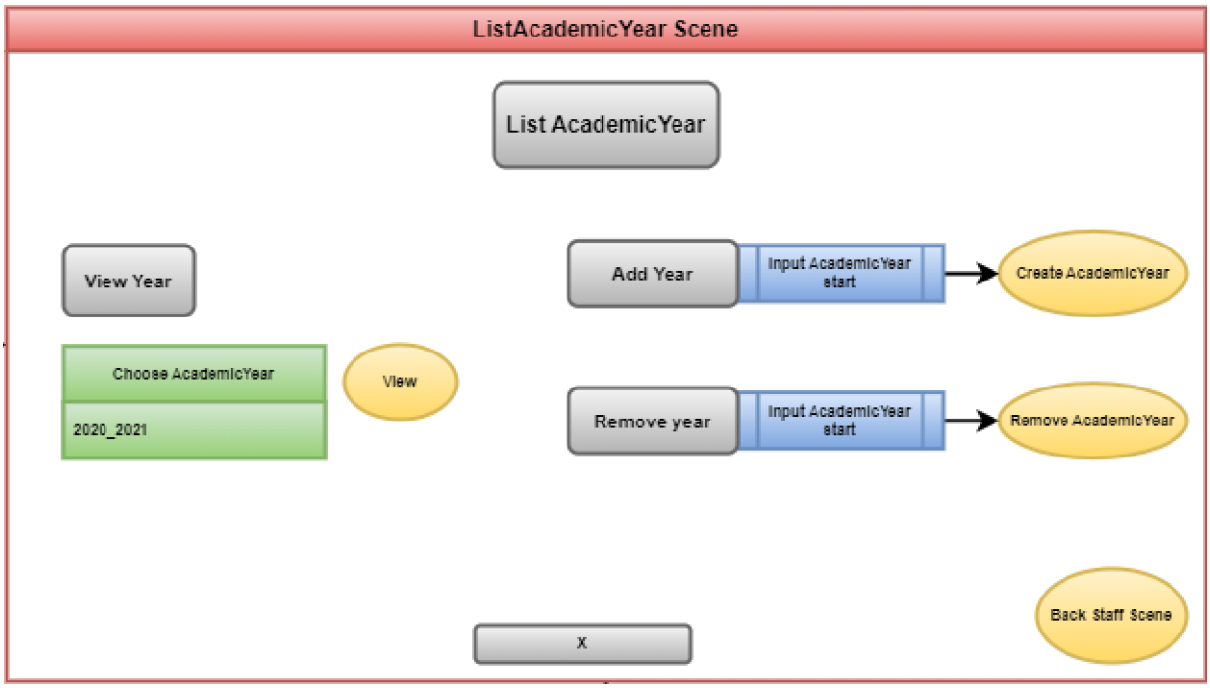
| Attribute / Method | Description |
|---|---|
| `ClassScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a student ID is valid, the program will access to `addStudentToClass()` function, then a student will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a student ID is valid, the program will access to `removeStudentFromClass()`, then a student will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the class drop box, then the function will return `ClassScene` scene. +Click on `Change` button, if the class name is valid, the program will access to `updateClass()`, then this class will be updated and the program will return this scene. Otherwise the warning text will occur. +Click on `Export` button, if the exported file name input is valid, the program will access to `exportListStudentInClass()`, In addition, if user choose a semester in semester drop box, the program will access to `exportListScoreboardInSemesterOfClass()`, then a `.csv` file will be exported and the function returns current scene. Otherwise, the warning text will |

| | |
|---|---|
| | occur. +Click on `Import` button, if the imported file name input is valid, the program will access to `importStudentListOfClassFromFile()`, then the function returns the current scene. Otherwise the warning will occur. +Click on `Back` button, the application will turn to `ListSchoolYearScene` scene. |

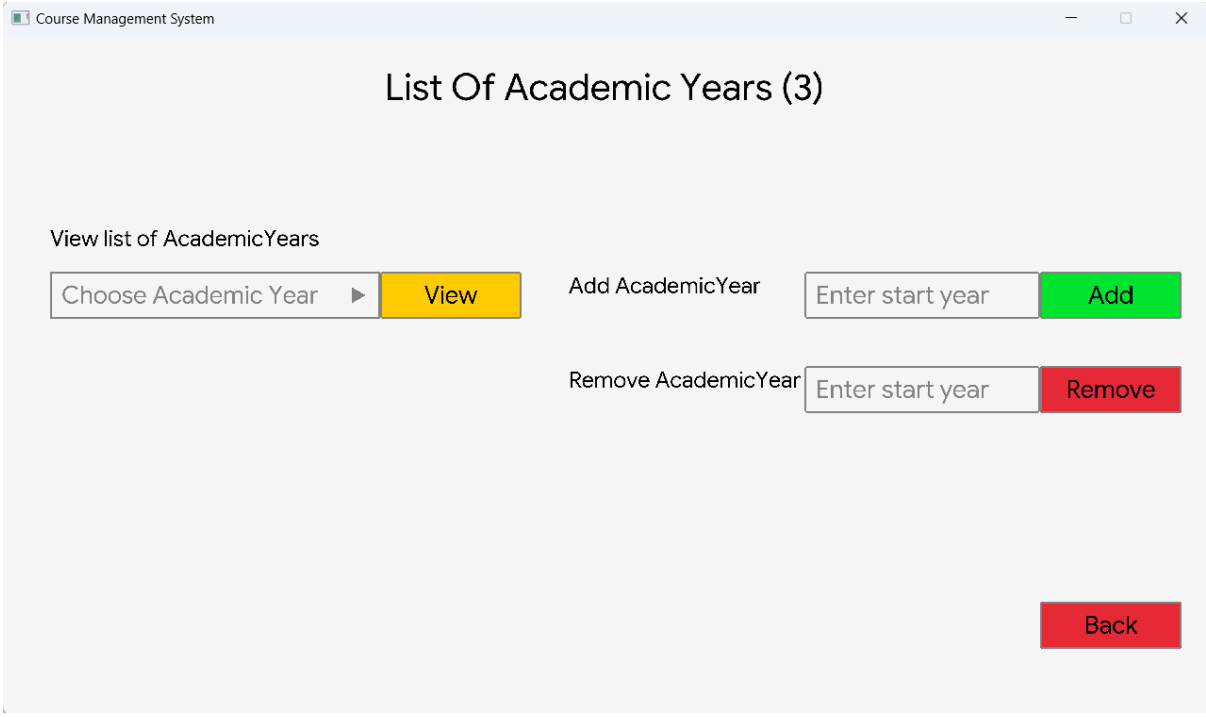
## 9. ListAcademicYearScene

```
(Defined in ListAcademicYearScene.h and implemented in ListAcademicYearScene.cpp)
```

Theoretic scene




Practical scene



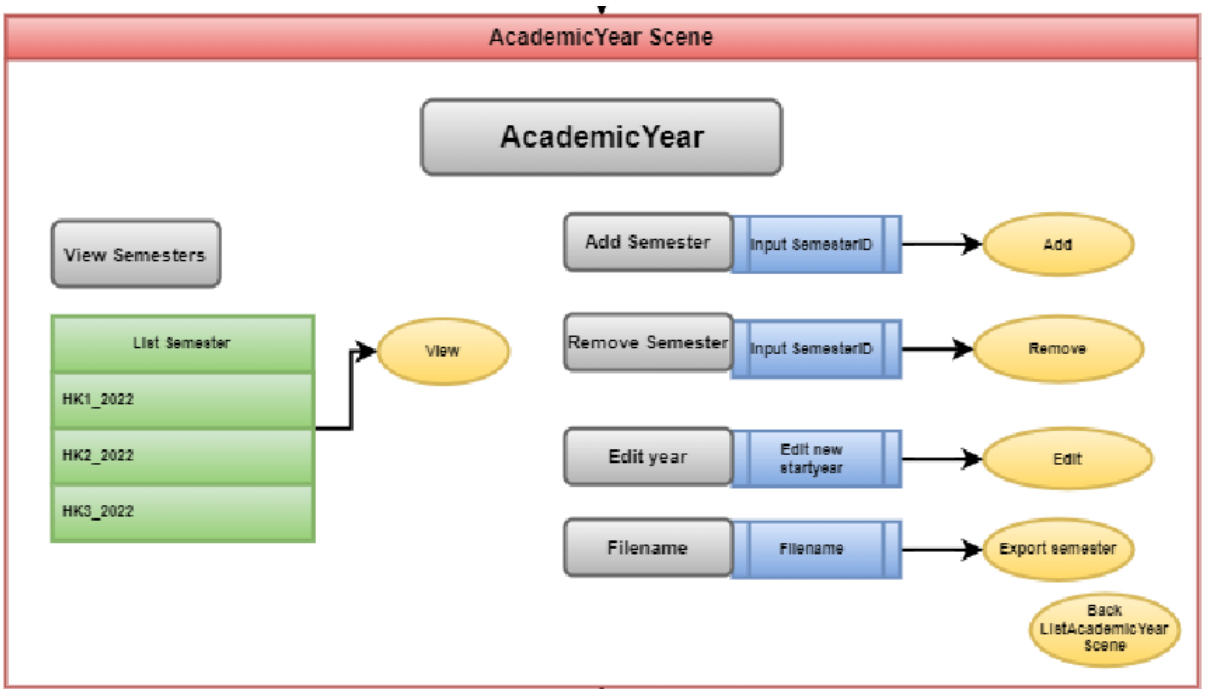

`ListAcademicYearScene` is a derived class inherited from `Scene` class used to create list academic year scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
|---|---|
| `ListAcademicYearScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a start year is valid, the program will access to `addAcademicYear()` function, then a school year will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a start year is valid, the program will access to `removeAcademicYear()` then a school year will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the school year drop box, then the function will return `AcademicYearScene` scene. +Click on `Back` button, the application will turn to `StaffScene` scene. |

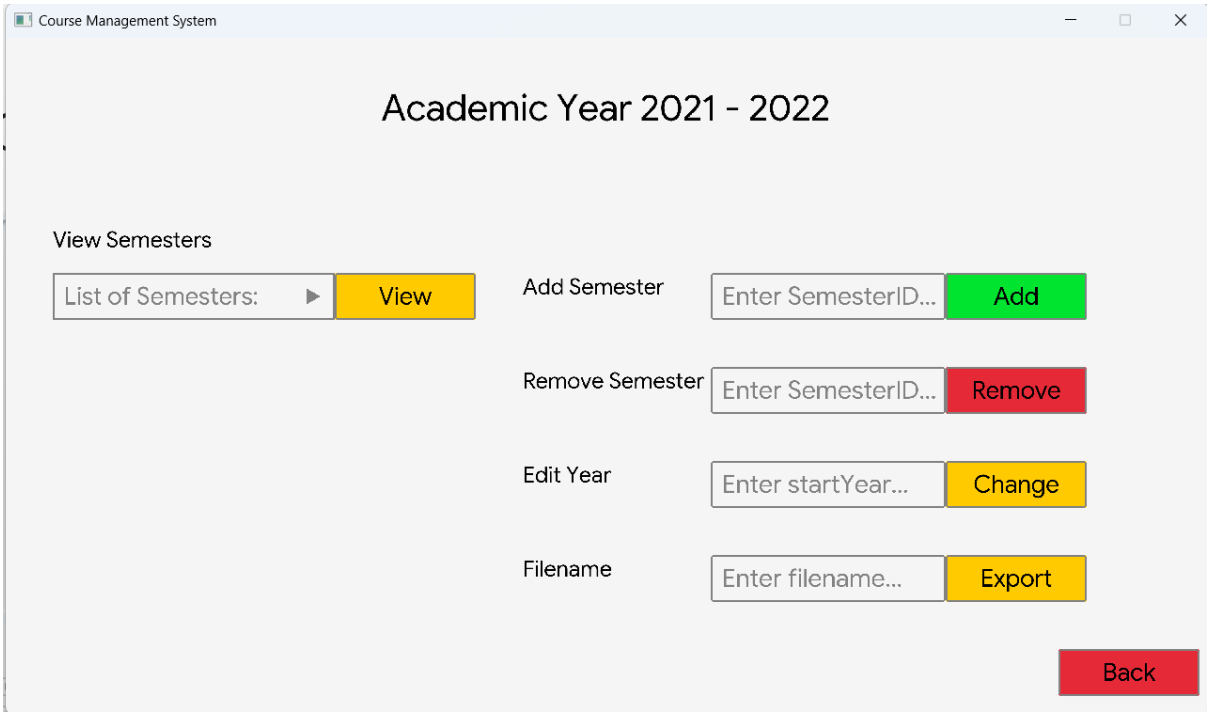
## 10. AcademicYearScene

```
(Defined in AcademicYearScene.h and implemented in AcademicYearScene.cpp)
```

Theoretic scene




Practical scene



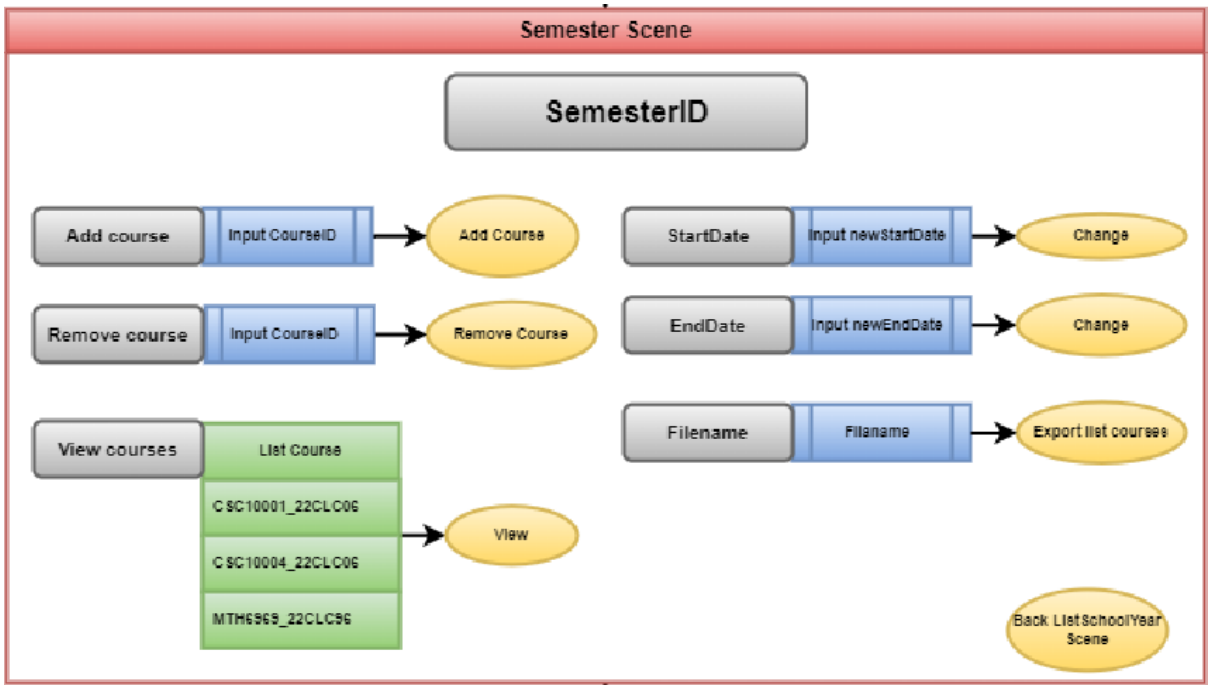

`AcademicYearScene` is a derived class inherited from `Scene` class used to create a academic year scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
| --- | --- |
| `AcademicYearScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a semester ID input is valid, the program will access to `addClass()` function, then a semester will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a semester ID input is valid, the program will access to `removeClass()`, then a semester will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the semester drop box, then the function will return `ClassScene` scene. +Click on `Change` button, if the start year is valid, the program will access to `updateAcademicYear()`, then this academicYear will be updated and the program will return this scene. Otherwise the warning text will occur. +Click on `Export` button, if the exported file name is valid, the program will access to `exportListClassInAcademicYear()`, then a `.csv` file will be exported and the function returns current scene. Otherwise, the warning text will occur. +Click on `Back` button, the application will turn to `ListAcademicYearScene` scene. |

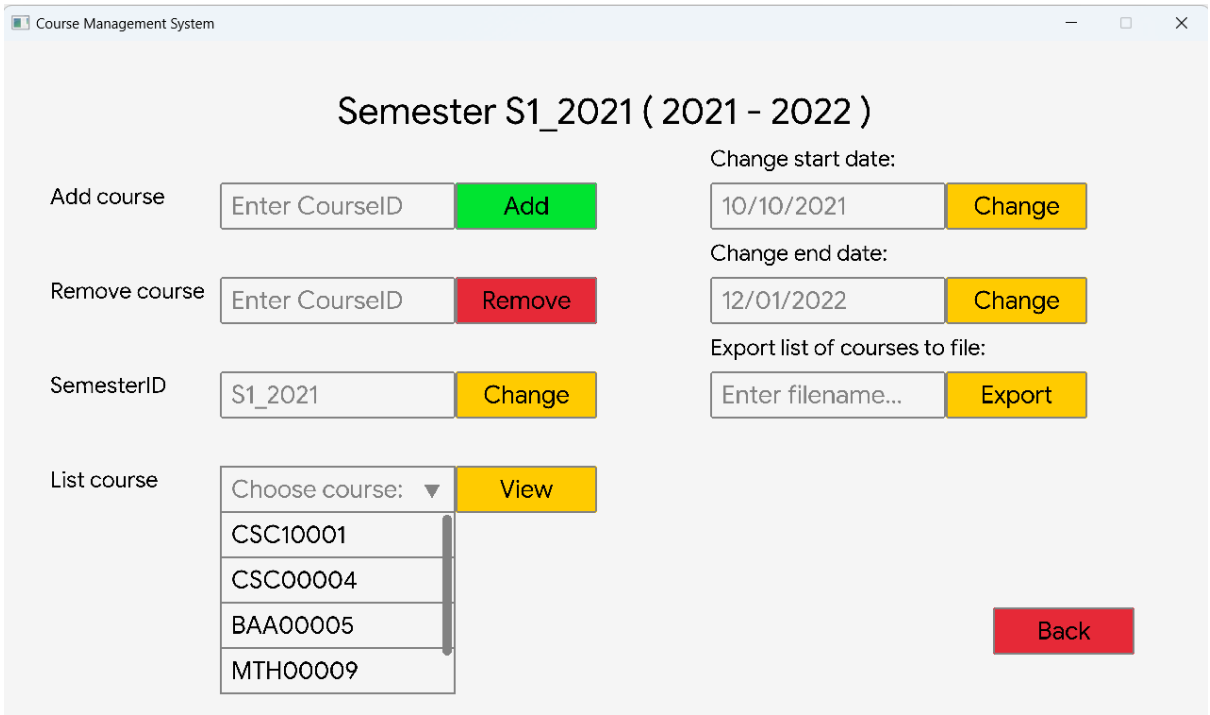
## 11. SemesterScene

```
(Defined in SemesterScene.h and implemented in SemesterScene.cpp)
```

Theoretic scene




Practical scene




`SemesterScene` is a derived class inherited from `Scene` class used to create a semester scene for the application. All the members in the private class are the variables for the objects for the scene.

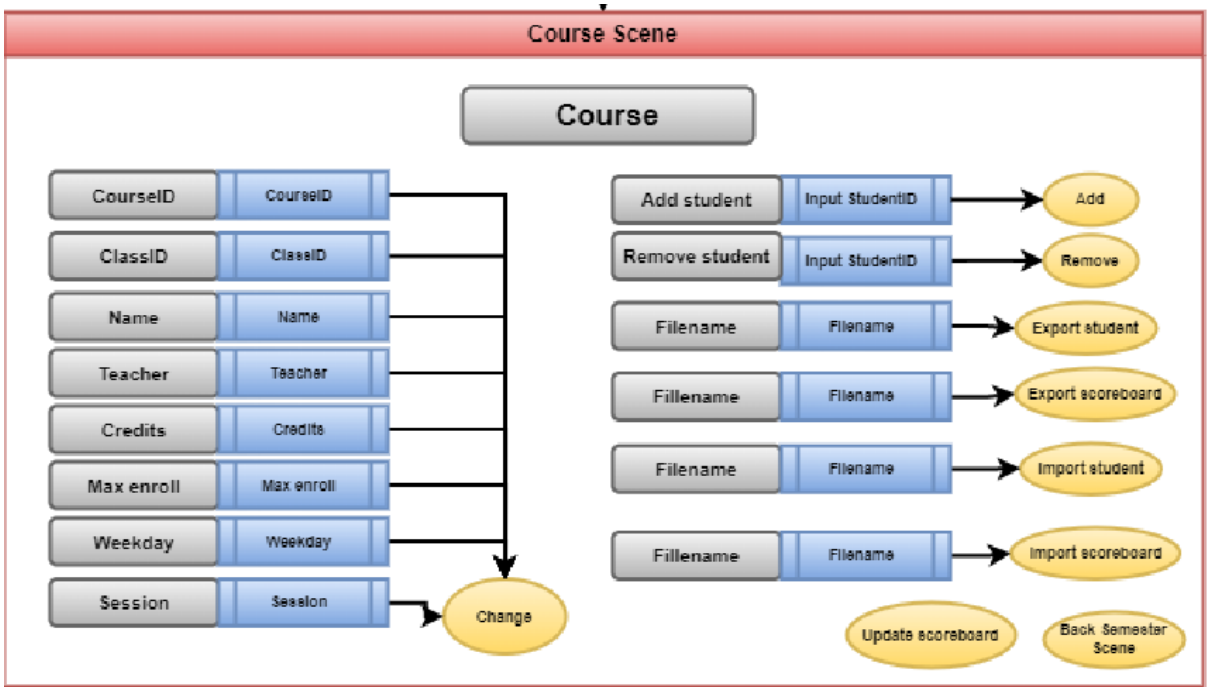
| Attribute / Method | Description |
| --- | --- |

| | |
|---|---|
| `SemesterScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a course ID input is valid, the program will access to `addCourse()` function, then a course will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a course ID input is valid, the program will access to `removeCourse()`, then a course will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to choose the course drop box, then the function will return `CourseScene` scene. +Click on `Change` button, if the start date input, end date input or semesterID input are valid, the program will access to `updateSemester()`, `Semester::updateStartDate()` or `Semester::updateEndDate()`, then this semester will be updated and the program will return this scene. Otherwise the warning text will occur. +Click on `Export` button, if the exported file name is valid, the program will access to `exportListCourseInSemester()`, then a `.csv` file will be exported and the function returns current scene. Otherwise, the warning text will occur. +Click on `Back` button, the application will turn to `AcademicYearScene` scene. |

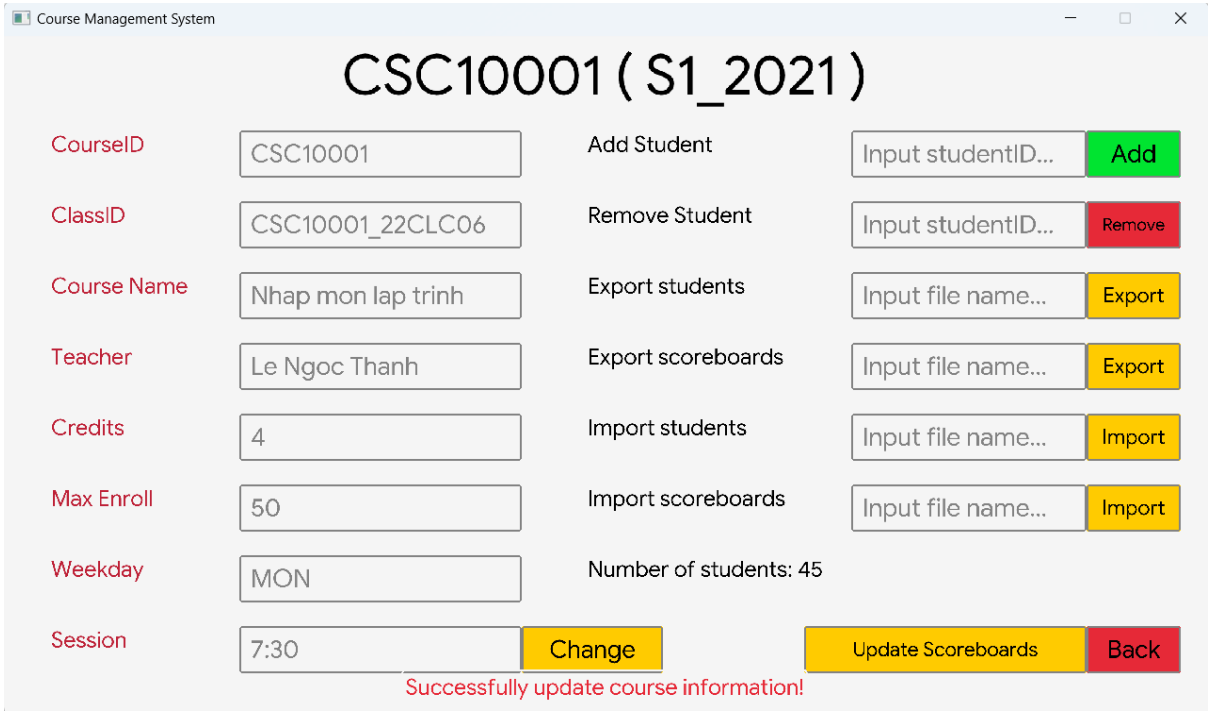
## 12. CourseScene

```
(Defined in ClassScene.h and implemented in CourseScene.cpp)
```

Theoretic scene




Practical scene




`CourseScene` is a derived class inherited from `Scene` class used to create a course scene for the application. All the members in the private class are the variables for the objects for the scene.

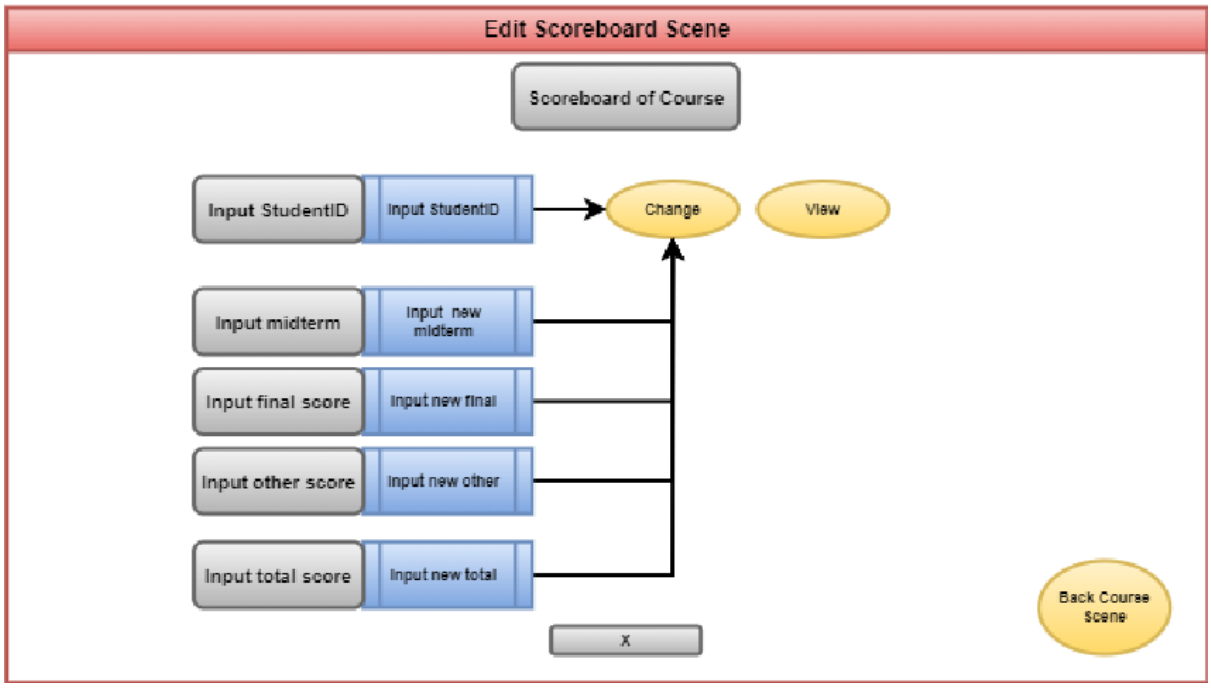
| Attribute / Method | Description |
| --- | --- |

| | |
|---|---|
| `CourseScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Add` button, if a student ID is valid, the program will access to `addStudentToCourse()` function, then a student will be added and the function will return the current scene. Otherwise the warning text will occur. + Click on `Remove` button, if a student ID is valid, the program will access to `removeStudentFromCourse()`, then a student will be removed, the function will return the current scene. Otherwise the warning text will occur. +Click on `UpdateScoreboard` button, the function will return `EditCourseScene` scene. +Click on `Change` button, if the course information inputs are valid, the program will access to `updateCourse()`, then this class will be updated and the program will return this scene. Otherwise the warning text will occur. +Click on `Export` button, if the exported file name inputs are valid, the program will access to `exportListStudentInCourse()` or `exportListScoreboardOfCourse()`, then a `.csv` file will be exported and the function returns current scene. Otherwise, the warning text will occur. +Click on `Import` button, if the imported file name inputs are valid, the program will access to `importStudentListOfCourseFromFile()` or `importScoreBoardOfCourse()`, then the function returns the current scene. Otherwise the warning will occur. +Click on `Back` button, the application will turn to `SemesterScene` scene. |

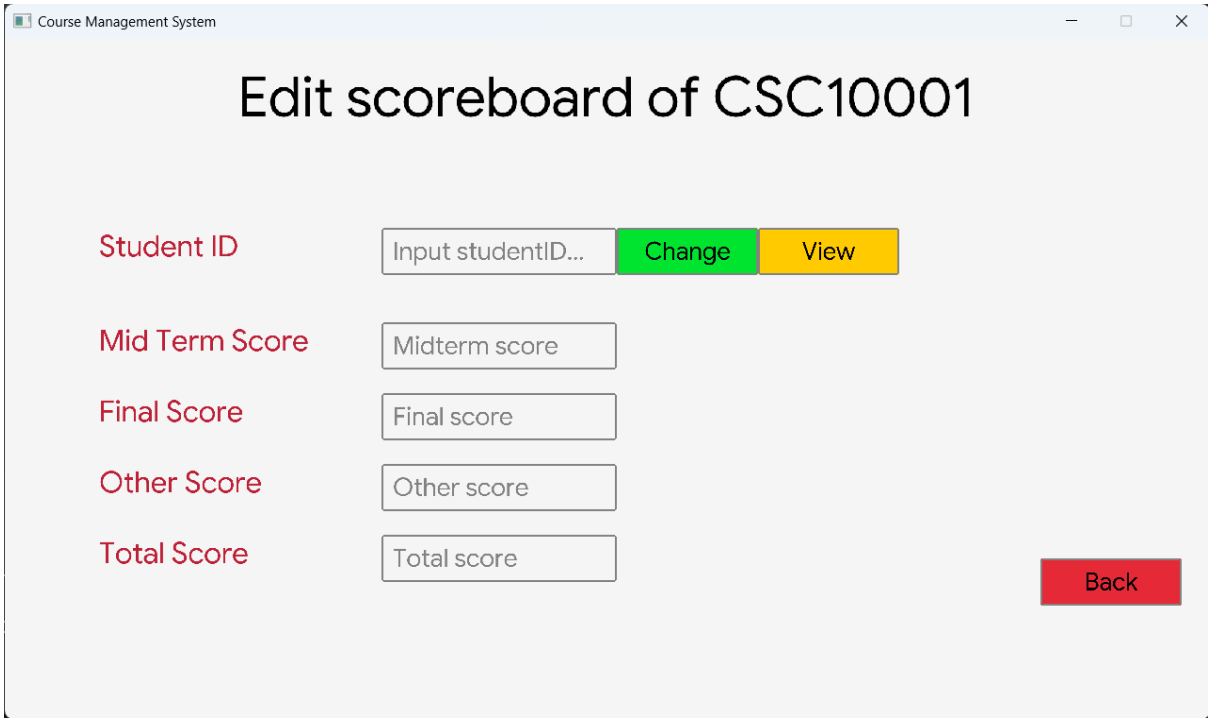
## 13. EditCourseScene

```
(Defined in EditCourseScene.h and implemented in EditCourseScene.cpp)
```

Theoretic scene




Practical scene




`EditCourseScene` is a derived class inherited from `Scene` class used to create a edit course scene for the application. All the members in the private class are the variables for the objects for the scene.

| Attribute / Method | Description |
| --- | --- |

| | |
|---|---|
| `EditCourseScene()` | The constructor initializes all private graphic objects above. |
| `void render()` | Renders all the objects implemented in the private class to the login scene. |
| `Scene* process()` | Updates the scene when user work on the scene and returns the scene to the application: + Click on `Change` button, if score inputs are valid and student ID input is valid, the program will access to `updateScoreboard()` function, then a scoreboard of a student will be updated and the function will return the current scene. Otherwise the warning text will occur. +Click on `View` button, the user had to enter student ID into input box, then the scoreboard of that student is shown on the screen and the function will return `SchoolYearScene` scene. +Click on `Back` button, the application will turn to `CourseScene` scene. |

## 14. Registry

```
(Defined in Registry.h and implemented in Registry.cpp)
```

`Registry` is a class used to store all the scenes have been implemented. All attributes in `Registry` class are the pointers of the `Scene` to all the scenes.