CSC10002 Group Project Report - Group08

Information

Course: CSC10002

Class: 22CLC06

Project: Course Management System

Group number: 08

Members:

- Đào Việt Hoàng - 22127121
- Nguyễn Gia Huy - 22127154
- Phan Hải Minh - 22127273
- Quách Trần Quán Vinh - 22127460

Features

In the current version

- User-friendly interface
- User permissions
- Add, delete, search, update objects
- Import and export data from file

Update in future version

- Directly display the table on the software interface
- Allows users to select files directly from the computer
- Report errors when users enter serious errors

How to run

- Download & install the latest version of CMake.
- Clone this repository.
- Open console exactly in this repository's directory and type the following:
  - `cmake -S . -B ./build` (this step might require internet connection if Raylib hasn't been installed before)
  - `cmake --build ./build -j 10`
- The executable `coursemgr` will appear, run it by typing `./coursemgr` in your console.

References

- Grading scheme
- Project contribution
- Trello board
- Youtube playlist
- Documentation

Acknowledgements

This section is made for the acknowledgements of our lecturers and teaching assistants. We would like to thank them for their help and support throughout the course, as well as their enlightening lectures and tutorials:

- Dr. Đinh Bá Tiến - Our beloved theory lecturer
- Mr. Hồ Tuấn Thanh - Our lab instructor
- Mr. Nguyễn Lê Hoàng Dũng - Our lab instructor

Class: 22CLC06

Group number: 08

Members:

- Đào Việt Hoàng - 22127121

- Nguyễn Gia Huy - 22127154
- Phan Hải Minh - 22127273
- Quách Trần Quán Vinh - 22127460

Features

In the current version

- User-friendly interface
- User permissions
- Add, delete, search, update objects
- Import and export data from file

Update in future version

- Directly display the table on the software interface
- Allows users to select files directly from the computer
- Report errors when users enter serious errors

How to run

- Download & install the latest version of CMake.
- Clone this repository.
- Open console exactly in this repository's directory and type the following:
  - `cmake -S . -B ./build` (this step might require internet connection if Raylib hasn't been installed before)
  - `cmake --build ./build -j 10`
- The executable `coursemgr` will appear, run it by typing `./coursemgr` in your console.

References

- Grading scheme
- Project contribution

- Trello board
- Youtube playlist
- Documentation

Acknowledgements

This section is made for the acknowledgements of our lecturers and teaching assistants. We would like to thank them for their help and support throughout the course, as well as their enlightening lectures and tutorials:

- Dr. Đinh Bá Tiến - Our beloved theory lecturer
- Mr. Hồ Tuấn Thanh - Our lab instructor
- Mr. Nguyễn Lê Hoàng Dũng - Our lab instructor

Documentation

Table of contents

I. Underlying data structures

Node

(Defined as `struct Node<Type>` in `Node.h`)

A simple node type. The basis of linked lists, stacks and other various data structures.

Attributes

Name	Description	Default Value
`Type data`	A variable of the type `Type` to store the data	
`Node* next`	A pointer that points to the next node	

Methods

Name	Description
`Node()`	Default constructor

Vector

(Defined as `class Vector<Type>` in `Vector.h`)

A vector type, which is a dynamic array that can be resized easily and efficiently.

Attributes

Name	Description	Default Value
`size_t length`		

	The size of the vector	0
`size_t capacity`	The capacity of the vector	1
`Type* array`	A pointer to the underlying dynamic array	`nullptr`

Methods

Name	Description
`Vector()`	Default constructor
`Vector(size_t size)`	Constructor with size
`Vector(const Vector<Type>& vec)`	Copy constructor
`void reallocate(size_t newCapacity)`	Reallocates the vector to a new capacity
`size_t size()`	Returns the size of the vector
`void operator=(const Vector<Type>& vec)`	Copy assignment operator
`Type& operator[](size_t pos)`	Index operator
`Type* begin()`	Returns a pointer to the first element
`Type* end()`	Returns a pointer to the last element
`void resize(size_t size)`	Resizes the vector to a new size
`void append(Type value)`	Pushes an element to the back of the vector
`void remove(const Type& value, int amount = -1)`	Removes elements from the vector with a given value. `-1` removes all elements
`void remove(Type* const ptr)`	Removes a given pointer from the vector
`Type* find(const Type& value)`	Finds and returns the pointer to the first element with a given value
`~Vector()`	Destructor

Stack

(Defined as `struct Stack<Type>` in `Stack.h`)

A stack type, which is a LIFO (Last In First Out) data structure.

Attributes

Name	Description	Default Value

`Node<Type>* head`	A pointer to the top node of the stack	`nullptr`

Methods

Name	Description
`Stack()`	Default constructor
`bool empty()`	Returns whether the stack is empty
`void push(const Type &value)`	Pushes an element to the top of the stack
`void pop()`	Pops the top element of the stack
`Type top()`	Returns the top element of the stack
`~Stack()`	Destructor

II. Core user-defined data types

Components

The `Components.h` header (and its source file `Components.cpp`) serves as a mediator file for other self-defined data types by forward-declaring them. It also includes comparing operator definition for the aformentioned types, enumerators and defines global variables.

Enumerators

Name	Description	Values
`Weekday`	Days of the week	`SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, `SAT`
`Gender`	Genders	`male`, `female`
`Session`	Academic sessions	`S1`, `S2`, `S3`, `S4`

Variables

Name	Description	Default Value
`std::string defaultStr`	A default string used to display in graphics objects	`"Error"`
`Vector<SchoolYear> schoolYears`	A global vector of school years	

`Vector<AcademicYear> academicYears`	A global vector of academic years	
`Vector<Student> students`	A global vector of students	
`Vector<Staff> staffs`	A global vector of staffs	
`Staff* ptrStaff_Global`	A global pointer to the currently interacting staff	`nullptr`
`Student* ptrStudent_Global`	A global pointer to the currently interacting student	`nullptr`
`SchoolYear* ptrSchoolYear_Global`	A global pointer to the current school year	`nullptr`
`Class* ptrClass_Global`	A global pointer to the current class	`nullptr`
`AcademicYear* ptrAcademicYear_Global`	A global pointer to the current academic year	`nullptr`
`Semester* ptrSemester_Global`	A global pointer to the current semester	`nullptr`
`Course* ptrCourse_Global`	A global pointer to the current course	`nullptr`
`Scoreboard* ptrScoreboard_Global`	A global pointer to the current scoreboard	`nullptr`

Functions

Name	Description
`bool operator!=(const Type& typeA, const Type& typeB)`	A template function to compare if compared variables differ from each other

Name

(Defined as `struct Name` in `Name.h`)

The `Name` struct represents one's name with a first and a last name. It provides methods for name setting and retrieval.

Attributes

Name	Description	Default Value
`std::string first`	Stores the first name of the person	
`std::string last`	Stores the first name of the person	

Methods

Name	Description
`Name(const std::string& nameFirst = defaultStr, const std::string& nameLast = defaultStr)`	Constructor with arguments for first & last names
`void set(const std::string& nameFirst, const std::string& nameLast)`	Modifies the first & last names
`std::string get() const`	Returns the full name (concatenation of first & last names)

Operators

Name	Description
`bool operator==(const Name& nameA, const Name& nameB)`	Compares two names

Date

(Defined as `struct Date` in `Date.h`)

The `Date` struct stores a specific date with day, month, and year values, while also provides methods for setting and retrieving the date.

Attributes

Name	Description	Default Value
`unsigned short day`	Stores the day value of the date	
`unsigned short month`	Stores the month value of the date	
`unsigned int year`	Stores the year value of the date	

Methods

Name	Description
`Date(const unsigned short& newDay = 0, const unsigned short& newMonth = 0, const unsigned int& newYear = 0)`	Constructor with arguments for day, month, and year
`void set(const unsigned short& newDay, const unsigned short& newMonth, const unsigned int& newYear)`	Modifies the day, month, and year values
`std::string get() const`	Returns the date in the format “dd/mm/yyyy”

Operators

Name	Description
`bool operator==(const Date& dateA, const Date& dateB)`	Compares two dates

User

(Defined as `struct User` in `User.h`)

The `User` struct represents a user with a name, ID, and password.

Attributes

Name	Description	Default Value
`Name name`	Stores the name of the user	
`std::string ID`	Stores the ID of the user	
`uint64_t password`	Stores the hashed password of the user	

Methods

Name	Description
`User(const Name& name = { defaultStr, defaultStr }, const std::string& ID = defaultStr, const std::string& passwordStr = "123456")`	Constructor with arguments for name, ID, and password
`uint64_t hash(uint64_t left, uint64_t right)`	Performs a custom hash operation on two `uint64_t` values and returns the result
`uint64_t hash(std::string str)`	Performs a custom hash operation on a string and returns the result
`void setFirstName(const std::string& firstName)`	Modifies the first name of the user
`void setLastName(const std::string& lastName)`	Modifies the last name of the user
`void setName(const std::string& first, const std::string& last)`	Modifies the name of the user
`void setPassword(const std::string& str)`	Modifies the password of the user
`void setPassWordUpLoad(const std::string& passwordInFile)`	Converts the hashed password in the file to a `uint64_t` value and stores it in the `password` attribute
`bool isPassword(const std::string& str)`	Checks if the given string matches the user's password

`void setID(const std::string& ID)`	Modifies the ID of the user
`uint64_t getHashedPass()`	Returns the hashed password of the user

Operators

Name	Description
`bool operator==(const User& userA, const User& userB)`	Compares two users

Staff

(Defined as `struct Staff` in `User.h`)

The `Staff` struct, which inherits from the `User` struct, represents a staff member with a name, ID, and password. Since the `Staff` struct inherits from the `User` struct, it also inherits all of the attributes and methods of the `User` struct, but it has no additional attributes or methods.

Student

(Defined as `struct Student` in `Student.h`)

The `Student` struct, which inherits from the `User` struct, represents a student with a name, ID, password and additional attributes. Since the `Student` struct inherits from the `User` struct, it also inherits all of the attributes and methods of the `User` struct, with additional attributes and methods.

Additional attributes

Name	Description	Default Value
`Gender gender`	Stores the gender of the student	
`Date birth`	Stores the date of birth of the student	
`std::string socialID`	Stores the social ID of the student	
`Class* ptrClass`	Stores a pointer to the class that the student belongs to	
`Vector<Scoreboard*> scoreboards`	Stores a vector of scoreboards of the student	

Additional methods

Name	Description
`Student(const Name& name = { defaultStr, defaultStr }, const std::string& id = defaultStr, 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*>())`	Constructor with arguments for attributes
`Vector<Scoreboard*> getScoreboards(const Semester& semester) const`	Returns a vector of scoreboards of the student in the given semester
`Scoreboard* getScoreboard(const std::string& courseID) const`	Returns the scoreboard of the student in the given course with the given ID
`Scoreboard* getScoreboard(Course& course) const`	Returns the scoreboard of the student in the given course
`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)`	Modifies the attributes of the student
`void setName(const Name& name)`	Modifies the name of the student
`void setGender(const Gender& gender)`	Modifies the gender of the student
`void setBirth(const Date& birth)`	Modifies the date of birth of the student
`void setSocialID(const std::string& socialID)`	Modifies the social ID of the student
`void setClass(Class* ptrClass)`	Modifies the class of the student
`void setScoreboards(Vector<Scoreboard*>& scoreboards)`	Modifies the scoreboards of the student
`float getGPA() const`	Returns the overall

	GPA of the student
`float getGPA(const Semester& semester) const`	Returns the GPA of the student in the given semester
`void setInfoToClass(std::ifstream &ifs)`	Reads the information of the student from a file stream and stores it in the student
`void setInfoToCourseCSV(std::ifstream &ifs)`	Reads the information of the student from a CSV file and stores it in the student
`void setInfoCourseConsole(std::string &actClass)`	Reads the information of the student from the console and stores it in the student

Additional operators

Name	Description
`bool operator==(const Student& studentA, const Student& studentB)`	Compares two students

School year

(Defined as `struct SchoolYear` in `SchoolYear.h`)

The `SchoolYear` struct represents a generation of students, with a starting year and its classes.

Attributes

Name	Description	Default Value
`unsigned int start`	The starting year of the school year	0
`Vector<Class> classes`	A vector of `Class` objects representing the classes associated with the school year	

Methods

Name	Description
`SchoolYear(const int& start = 0, const Vector<Class>& classes = Vector<Class>())`	Constructor with arguments for attributes
`unsigned int getStartYear()`	Returns the starting year of the school year
`void set(const unsigned int& start, const Vector<Class>& classes)`	Sets the start year and classes of the school year
`void update(const unsigned int& start)`	Updates the start year of the school year
`void update(Vector<Class>& classes)`	Updates the classes of the school year
`Student* getStudent(const std::string& studentID)`	Returns a pointer to a `Student` object with the given student ID, if found in any of the classes within the school year
`Class* getClass(const std::string& className)`	Returns a pointer to a `Class` object with the given class name, if found in the school year
`void addClass(Class& CLASS)`	Adds a new class to the school year
`void removeClass(Class& CLASS)`	Removes a class from the school year
`void removeAllClass()`	Removes all classes from the school year
`std::string getPeriod(std::string& period) const`	Gets the period of the school year

Operators

Name	Description
`bool operator==(const SchoolYear& yearA, const SchoolYear& yearB)`	Compares two school years

Class

(Defined as `struct Class` in `Class.h`)

The `Class` struct represents a class within a school year with students.

Attributes

Name	Description	Default Value
`SchoolYear* ptrSchoolYear`	A pointer to the associated school year for the class	
`std::string name`	The name of the class	

`Vector<Student*> students`	A vector containing pointers to the students in the class	

Methods

Name	Description
`Class(SchoolYear* ptrSchoolYear = nullptr, const std::string& name = defaultStr, const Vector<Student*>& students = Vector<Student*>())`	Constructor for the `Class` struct with arguments to initialize its 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 year associated to this class
`void update(const std::string& name)`	Updates the name of the class
`void update(Vector<Student*>& students)`	Updates the students in the class
`void addStudent(Student*& student)`	Adds a student to the class
`void removeStudent(Student*& student)`	Removes a student from the class
`void removeAllStudent()`	Removes all students from the class
`Vector<string> getListCourse() const`	Returns a vector of courses of the class that its students have enrolled
`Vector<string> getListCourse(const Semester& semester) const`	Returns a vector of courses of the class that its students have enrolled in the given semester
`Student* getStudent(const std::string& studentID)`	Returns a pointer to a `Student` object with the given student ID, if found in the class
`void displayScoreboardScreen(const Semester& semester)`	Displays the scoreboard of the class in the given semester
`void displayScoreboardFile(const Semester& semester, std::ofstream& ofs)`	Export the scoreboard of the class in the given semester to a file stream

Operators

Name	Description
`bool operator==(const Class& classA, const Class& classB)`	Compares two classes

Academic year

The `AcademicYear` struct represents an academic year, which has a period of 12 months divided into 3 semesters.

Attributes

Name	Description	Default Value
`unsigned int start`	The start year of the academic year	
`Vector<Semester> semesters`	A list of semesters in the academic year	

Methods

Name	Description
`AcademicYear(const unsigned int& start = 0, const Vector<Semester>& semester = Vector<Semester>())`	Constructor with arguments to initialize attributes
`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

Operators

Name	Description
`bool operator==(const AcademicYear& yearA, const AcademicYear& yearB)`	Compares two academic years

Semester

(Defined as `struct Semester` in `Semester.h`)

The `Semester` struct represents a semester in a typical year, which has its own start and end dates, and a list of courses.

Attributes

Name	Description	Default Value
`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 list of courses in the semester	
`AcademicYear* ptrAcademicYear`	A pointer to the academic year associated with the semester	

Methods

Name	Description
`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 with arguments to initialize attributes
`void set(const std::string& semesterID, const Date& startDate, const Date& endDate, const Vector<Course>& course, AcademicYear* ptrAcademicYear)`	Sets the attributes of the semester
`void updateSemesterID(const std::string& semesterID)`	Updates the ID of the semester
`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 list of courses in the semester
`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

`Course* getCourse(const std::string& courseID)`	Returns a pointer to the course with the specified ID, or nullptr if it does not exist

Operators

Name	Description
`bool operator==(const Semester& semA, const Semester& semB)`	Compares two semesters

Course

(Defined as `struct Course` in `Course.h`)

The `Course` struct represents a course in a standard university, which has its own ID, name, teacher, number of credits, maximum number of students, and a list of students enrolled in the course.

Attributes

Name	Description	Default Value
`std::string ID`	The ID of the course	
`std::string classID`	The ID of the class associated with the course	
`std::string name`	The name of the course	
`std::string teacher`	The name of the teacher teaching the course	
`int credits`	The number of credits assigned to the course	
`int maxEnroll`	The maximum number of students that can enroll in the course	
`Weekday weekday`	The course's scheduled day of the week	
`Session session`	The course's scheduled session of the day (morning, afternoon)	
`Semester* ptrSemester`	A pointer to the semester associated with the course	
`Vector<Scoreboard*> scoreboards`	A vector of pointers to scoreboards, which store the scores of students in this particular course	

Methods

Name	Description
`Course(const std::string& ID = defaultStr, const std::string& classID = defaultStr, const std::string& name = defaultStr, const std::string& teacher = defaultStr, const int& credits = 0, const int& maxEnroll = 50, const Weekday& weekday = MON, const Session& session = S1, Semester* ptrSemester = nullptr, const Vector<Scoreboard*>& scoreboards = Vector<Scoreboard*>())`	Constructor with arguments to initialize attributes
`void set(const std::string& ID, const std::string& classID, const std::string& name, const std::string& teacher, const int& credits, const int& maxEnroll, const Weekday& weekday, const Session& session, Semester* ptrSemester, const Vector<Scoreboard*>& scoreboards)`	Sets the attributes of the course
`void updateID(const std::string& ID)`	Updates the ID of the course
`void updateClassID(const std::string& classID)`	Updates the ID of the class associated with the course
`void updateName(const std::string& name)`	Updates the name of the course
`void updateTeacher(const std::string& teacher)`	Updates the name of the teacher teaching the course
`void updateCredits(const int& credits)`	Updates the number of credits assigned to the course
`void updateMaxEnroll(const int& maxEnroll)`	Updates the maximum number of students that can enroll in the course
`void updateWeekday(const Weekday& weekday)`	Updates the scheduled day of the week
`void updateSession(const Session& session)`	Updates the scheduled session of the day
`void updateSemester(Semester* ptrSemester)`	Updates the semester

	associated with the course
`void updateScoreboard(const Vector<Scoreboard*>& scoreboards)`	Updates the list of scoreboards associated with the course
`Student* getStudent(const std::string& studentID)`	Returns a pointer to the student with the specified ID, or nullptr if it does not exist
`Scoreboard* getScoreboard(const std::string& studentID)`	Returns a pointer to the scoreboard with the specified student ID, or nullptr if it does not exist
`void addStudent(Student& student)`	Adds a student to the course
`void removeStudent(Student& student)`	Removes a student from the course
`void removeAllStudent()`	Removes all students from the course
`void displayInfoFile(std::ofstream& ofs)`	Writes the course's information to a file stream
`void displayInfoTable(Vector<Vector<std::string>>& table) const`	Displays the course information in a tabular format using the provided table
`void importScoreBoards(std::ifstream& ifs)`	Imports the scores of students in the course from a file

Operators

Name	Description
`bool operator==(const Course& courseA, const Course& courseB)`	Compares two courses

Scoreboard

(Defined as `struct Scoreboard` in `Scoreboard.h`)

The `Scoreboard` struct represents a score record for a specific course and student. It contains information about the scores of the student in various examinations of the course.

Attributes

Name	Description	Default Value
`float midterm`	The midterm result	
`float final`	The final result	
`float other`	Other results in total	
`float total`	The total result	
`Course* ptrCourse`	A pointer to the course associated with the scoreboard	
`Student* ptrStudent`	A pointer to the student associated with the scoreboard	

Methods

Name	Description
`Scoreboard(Course* ptrCourse = nullptr, Student* ptrStudent = nullptr, const float& midterm = -1, const float& final = -1, const float& other = -1, const float& total = -1)`	Constructor with arguments to initialize attributes
`void setScore(const float& midterm, const float& final, const float& other, const float& total)`	Sets the scores of the student in the course
`void updateMidterm(const float& midterm)`	Updates the midterm result
`void updateFinal(const float& final)`	Updates the final result

`void updateOther(const float& other)`	Updates the other result
`void updateTotal(const float& total)`	Updates the total result

III. Helper Functions

Display functions

(Defined in `DisplayFunction.h`)

The following functions are used to convert the following data types into a string vector to be used in dropboxes.

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

The following functions are used to convert the following data types into a 2-dimensional string vector to be used in tables.

Function	Description
`Vector<Vector<string>> getTableContentOfListSchoolYear()`	Returns a list of school years with the number of classes in each year
`Vector<Vector<string>> getTableContentOfSchoolYear(const SchoolYear& schoolYear)`	Returns the list of classes with the number of students in each class
`Vector<Vector<string>> getTableContentOfListStudentInClass(const Class& CLASS)`	Returns a list of students with their information

`Vector<Vector<string>> getTableContentOfListScoreboardInSemesterInClass(const Class& CLASS, const Semester& semester)`	Returns the scoreboard in the given semester of the given class
`Vector<Vector<string>> getTableContentOfListScoreboardInClass(const Class& CLASS)`	Returns the scoreboard in the given year of the given class
`Vector<Vector<string>> getTableContentOfScoreboardOfStudent(const Student& student)`	Returns the scoreboard of a student in the given year
`Vector<Vector<string>> getTableContentOfScoreboardInSemesterOfStudent(const Student& student, const Semester& semester)`	Returns the scoreboard of a student in the given semester
`Vector<Vector<string>> getTableContentOfListAcademicYear()`	Returns the list of academic years with the number of semesters in each year
`Vector<Vector<string>> getTableContentOfAcademicYear(const AcademicYear& academicYear)`	Returns the list of semesters and information of each in an academic year
`Vector<Vector<string>> getTableContentOfSemester(const Semester& semester)`	Returns the information of a semester
`Vector<Vector<string>> getTableContentOfListStudentInCourse(const Course& course)`	Returns the information of students in a course

Download functions

(Defined in `DownLoadFunction.h`)

The following functions save the data into CSV files when the program is closed.

Name	Description
`bool downloadAllData()`	Downloads all the data to the default CSV files
`bool downloadListStaff()`	Downloads all staffs to the default CSV files
`bool downloadListStudent()`	Downloads all students to the default CSV files
`bool downloadListSchoolYear()`	Downloads all school years to the default CSV files
`bool downloadSchoolYear(SchoolYear& schoolYear)`	Downloads the given school year to the default CSV files
`bool downloadClass(Class& CLASS)`	Downloads the given class to the default CSV files
`bool downloadListAcademicYear()`	Downloads all academic years to the default CSV files
`bool downloadAcademicYear(AcademicYear& academicYear)`	Downloads the given academic year to the default CSV files

`bool downloadSemester(Semester& semester)`	Downloads the given semester to the default CSV files
`bool downloadCourse(Course& course)`	Downloads the given course to the default CSV files

File and directory functions

(Defined in `FileAndDirFunction.h`)

The following functions create necessary directories for importing and exporting functions.

Name	Description
`void createDirectoryIfNotExists(const string& dirPath)`	Creates a new directory if it does not exist yet
`string getListStaffFilePath()`	Returns the directory of the staff list file
`string getListStudentFilePath()`	Returns the directory of the student list file
`string getListSchoolYearFilePath()`	Returns the directory of the school year list file
`string getListAcademicYearFilePath()`	Returns the directory of the academic year list file
`string getSchoolYearFolderPath(const SchoolYear& schoolyear)`	Returns the folder path to the given school year
`string getSchoolYearFilePath(const SchoolYear& schoolyear)`	Returns the file path to the given school year
`string getClassFolderPath(const Class& CLASS)`	Returns the folder path to the given class
`string getClassFilePath(const Class& CLASS)`	Returns the file path to the given class
`string getAcademicYearFolderPath(const AcademicYear& academicYear)`	Returns the folder path to the given academic year
`string getAcademicYearFilePath(const AcademicYear& academicYear)`	Returns the file path to the given academic year
`string getSemesterFolderPath(const Semester& semester)`	Returns the folder path to the given semester
`string getSemesterFilePath(const Semester& semester)`	Returns the file path to the given semester
`string getCourseFolderPath(const Course& course)`	Returns the folder path to the given course

`string getCourseFilePath(const Course& course)`	Returns the file path to the given course
`string getExportFolderPath()`	Adds prefix to data exporting path
`string getImportFolderPath()`	Adds prefix to data importing path

Import and export functions

(Defined in `ImportAndExportFunction.h`)

The following functions import data from and export data to a CSV files.

Name	Description
`bool importStudentListOfClassFromFile(const string& filename, Class& actClass, string& outStr)`	Imports students to a class
`bool importStudentListOfCourseFromFile(const string& filename, Course& course, string& outStr)`	Imports students to a course
`bool exportListOfStudentInCourse(const string& filename, Course& course, string& outStr)`	Exports students in a course
`bool importScoreBoardOfCourse(const string& filename, Course& course, string& outStr)`	Imports scoreboards of a course
`bool exportListScoreboardOfStudent(const string& filename, Student& student, string& outStr)`	Exports scoreboards of a student in an academic year
`bool exportListScoreboardInSemesterOfStudent(const string& filename, Student& student, Semester& semester, string& outStr)`	Exports scoreboards of a student in a semester
`bool exportListSchoolYear(const string& filename, string& outStr)`	Exports all school years
`bool exportListClassInSchoolYear(const string& filename, SchoolYear& schoolYear, string& outStr)`	Exports classes of a school year
`bool exportListStudentInClass(const string& filename, Class& CLASS, string& outStr)`	Exports students in a class
`bool exportListScoreboardInSemesterOfClass(const string& filename, Class& CLASS, Semester& semester, string& outStr)`	Exports scoreboards of a class in a semester
`bool exportListScoreboardOfClass(const string& filename, Class& CLASS, string& outStr)`	Exports scoreboards of a class
`bool exportListAcademicYear(const string& filename, string& outStr)`	Exports all academic years
`bool exportListSemesterInAcademicYear(const string& filename, AcademicYear& academicYear, string& outStr)`	Exports semesters in an academic year
`bool exportListCourseInSemester(const string& filename, Semester& semester, string& outStr)`	Exports courses in a semester
`bool exportListScoreboardOfCourse(const string& filename, Course&`	Exports scoreboards of a

`course, string& outStr)`	course

Insert functions

(Defined in `InsertFunction.h`)

The following functions create new values of various data types then add them to given lists.

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

Remove functions

(Defined in `RemoveFunction.h`)

The following functions remove data, freeing memory in the process.

Name	Description
`bool removeListStudent()`	Removes students
`bool removeListStaff()`	Removes staffs
`bool removeListSchoolYear()`	Removes school years
`bool removeListAcademicYear()`	Removes academic years
`bool freeMemory()`	Removes all data
`bool removeStudent(const string& studentID, string& outStr)`	Removes a student
`bool removeStaff(string curStaffID, const string& staffID, string& outStr)`	Removes a staff
`bool removeSchoolYear(const string& start, string& outStr)`	Removes a school year
`bool removeClass(SchoolYear& schoolYears, const string& className, string& outStr)`	Removes a class
`bool removeStudentFromClass(Class& CLASS, const string& studentID, string& outStr)`	Removes a student from the class
`bool removeAcademicYear(const string& start, string& outStr)`	Removes an academic year
`bool removeSemester(AcademicYear& academicYear, const string& semesterID, string& outStr)`	Removes a semester
`bool removeCourse(Semester& semester, const string& courseID, string& outStr)`	Removes a course
`bool removeStudFromCourse(Course& course, const string& studentID, string& outStr)`	Removes a student from the course

Search functions

(Defined in `SearchFunction.h`)

The following functions search for necessary content from the data.

Name	Description
`Student* getStudent(const string& studentID)`	Finds a student
`Staff* getStaff(const string& staffID)`	Finds a staff
`SchoolYear* getSchoolYear(const string& start)`	Finds a school year
`AcademicYear* getAcademicYear(const string& start)`	Finds an academic year
`Class* getClass(SchoolYear& schoolYear, const string& className)`	Finds a class from the given school year

`Semester* getSemester(const string& semesterID)`	Finds a semester
`Semester* getSemester(AcademicYear& academicYear, const string& semesterID)`	Finds a semester from the given academic year
`Course* getCourse(Semester& semester, const string& courseID)`	Finds a course from a semester
`Scoreboard* getScoreboard(Course& course, const string& studentID)`	Finds the scoreboard of a given student from their course
`bool isCorrectStaffAccount(const string& staffID, const string& password, string& outStr)`	Authenticates the staff account
`bool isCorrectStudentAccount(const string& studentID, const string& password, string& outStr)`	Authenticates the student account

Sort functions

(Defined in `SortFunction.h`)

The following functions sort various data type lists using the Quicksort algorithm.

Name	Description
`void sortStudentList(Vector<Student>& students, const int& left, const int& right)`	Sorts students
`void sortStaffList(Vector<Staff>& staffs, const int& left, const int& right)`	Sorts staffs
`void sortSchoolYearList(Vector<SchoolYear>& schoolYears, const int& left, const int& right)`	Sorts school years
`void sortAcademicYearList(Vector<AcademicYear>& academicYears, const int& left, const int& right)`	Sorts academic years
`void sortSemesters(Vector<Semester>& semesters, const int& left, const int& right)`	Sorts semesters
`void sortClasses(Vector<Class>& classes, const int& left, const int& right)`	Sorts classes
`void sortCourses(Vector<Course>& courses, const int& left, const int& right)`	Sorts courses
`void sortStudentsInClass(Class& CLASS, const int& left, const int& right)`	Sorts students of a class
`void sortStudentsInCourse(Course& course, const int& left, const int& right)`	Sorts students of a course

Update functions

(Defined in `UpdateFunction.h`)

The following functions update the data.

Name	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)`	Updates the student information
`bool updateStaffIn4(Staff& staff, const string& ID, const string& firstName, const string& lastName, const string& password, string& outStr)`	Updates the staff information
`bool updateSchoolYear(SchoolYear& schoolYear, const string& newStartYear, string& outStr)`	Updates the school year information
`bool updateClass(Class& CLASS, const string& newClassName, string& outStr)`	Updates the class information
`bool updateAcademicYear(AcademicYear& academicYear, const string& newStartYear, string& outStr)`	Updates the academic year information
`bool updateSemester(Semester& semester, const string& semesterID, const string startDate, const string endDate, string& outStr)`	Updates 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)`	Updates the course information
`bool updateScoreboard(Course& course, const string& studentID, const string& midTerm, const string& other, const string& final, const string& total, string& outStr)`	Updates the student scoreboard

Upload functions

(Defined in `UpLoadFunction.h`)

The following functions upload the data from CSV files when the program starts.

Name	Description
`bool uploadAllData()`	Uploads all the data
`bool uploadListStaff()`	Uploads the list of staffs
`bool uploadListStudent()`	Uploads the list of students
`bool uploadListSchoolYear()`	Uploads the list of school years
`bool uploadSchoolYear(SchoolYear& schoolYear)`	

	Uploads the school year
`bool uploadStudent(Class& actClass, Student& student, std::string id)`	Uploads the student to the given class
`bool uploadClass(Class& actClass)`	Uploads the class
`bool uploadListAcademicYear()`	Uploads the list of academic years
`bool uploadAcademicYear(AcademicYear& academicYear)`	Uploads the academic year
`bool uploadSemester(Semester& semester)`	Uploads the semester
`bool uploadCourse(Course& course)`	Uploads the course

IV. Graphics

Constants

(Defined in Constant.h)

Contains default values for graphical elements & objects used in the program. The constants are divided into these following namespaces:

- **“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.

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.

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*)

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.

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

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

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.

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.

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.

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.

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.

V. Graphical scenes

Scene

(Defined in Scene.h)

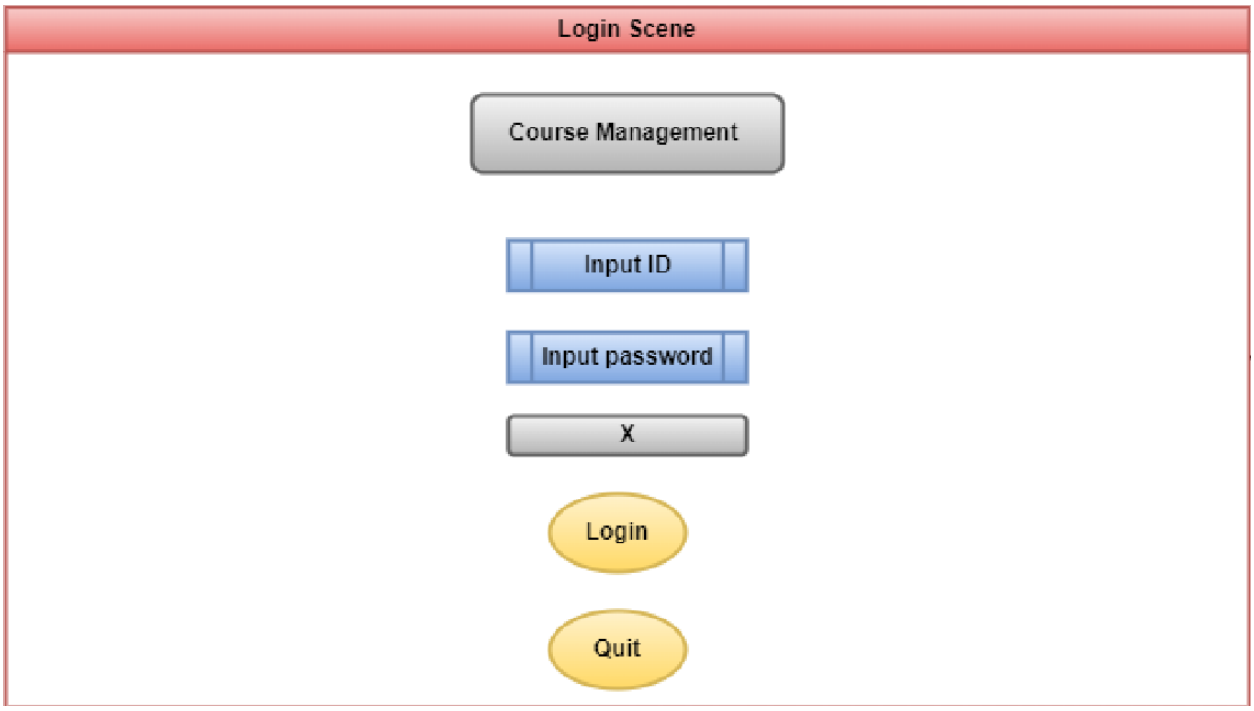
`Scene` is a base class that can be inherited by the derived classes, these classes inherite the attributes and methods in `Scene` class, which is used to create specific scenes.

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.

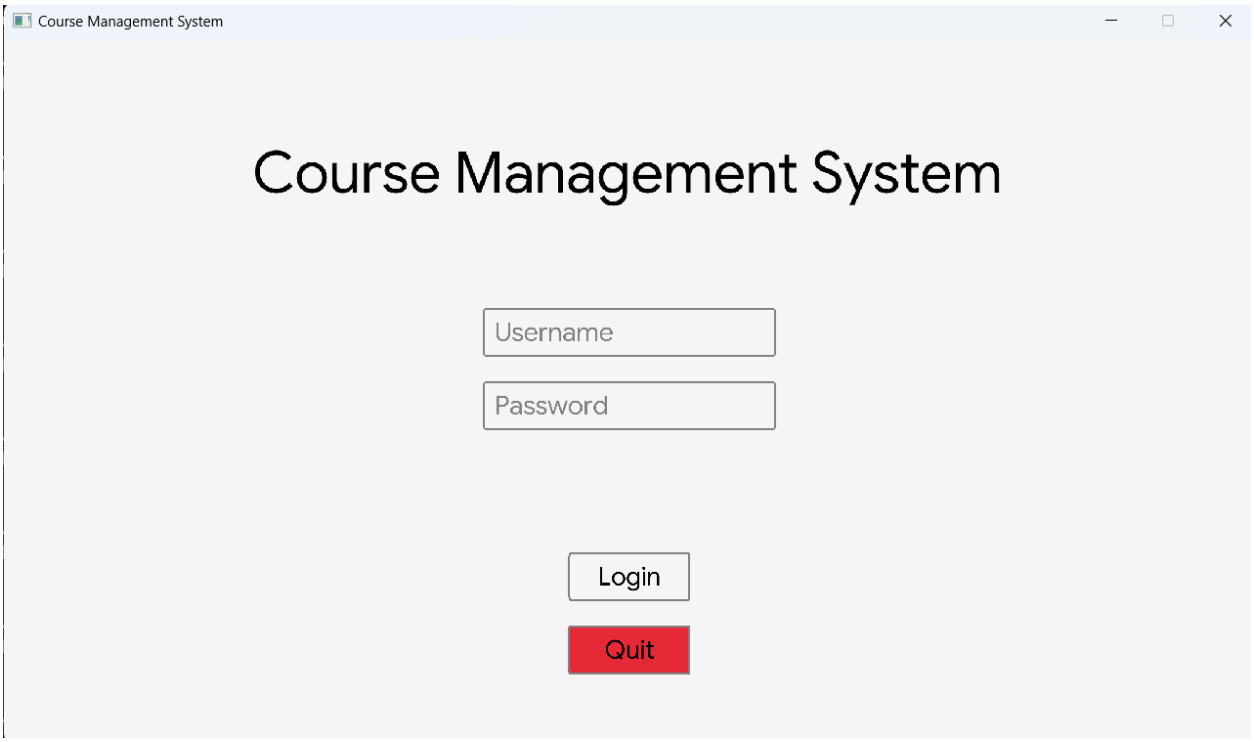
Login

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

Theoretic scene



Practical scene



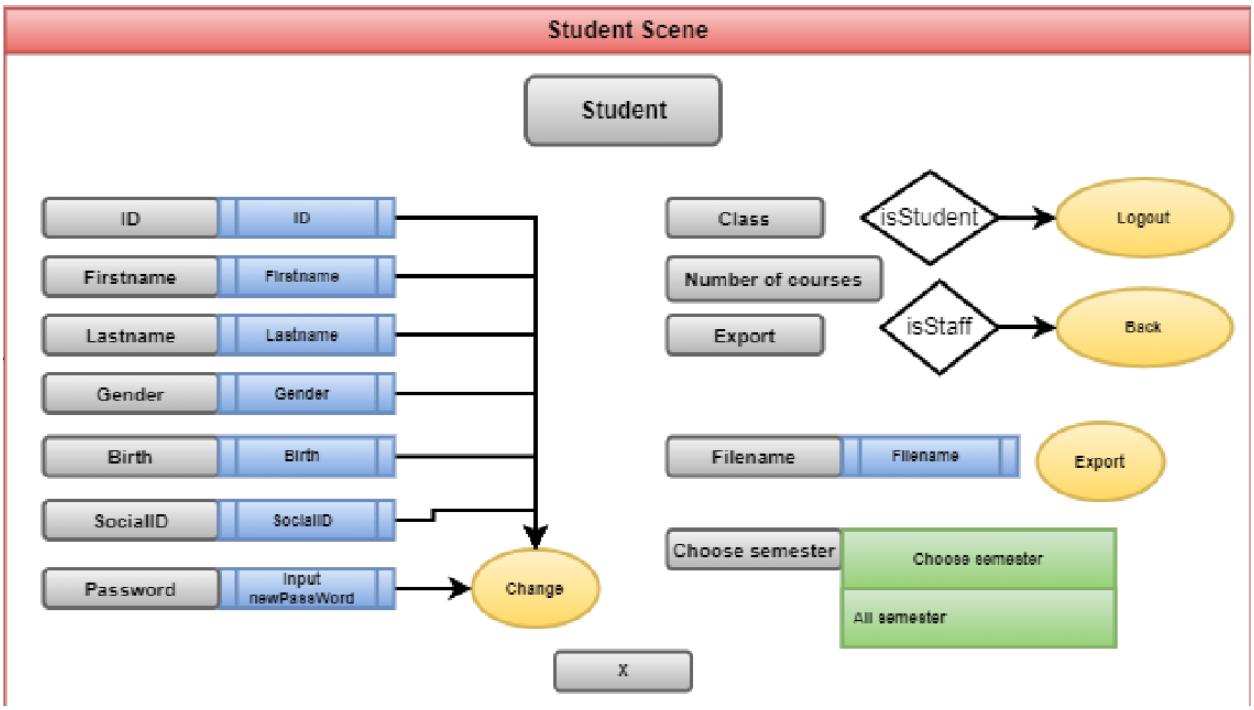
`Login` is a derived class inherited from `Scene` class used to create a login scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

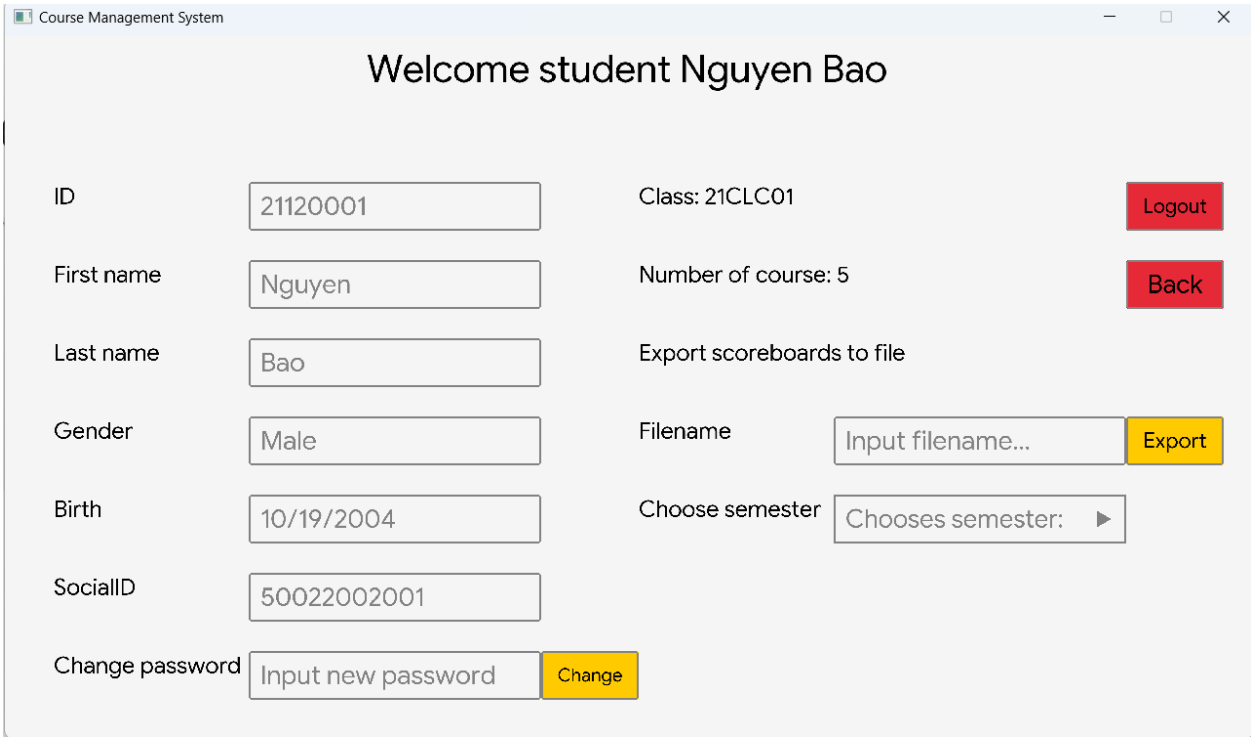
Student scene

```
(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

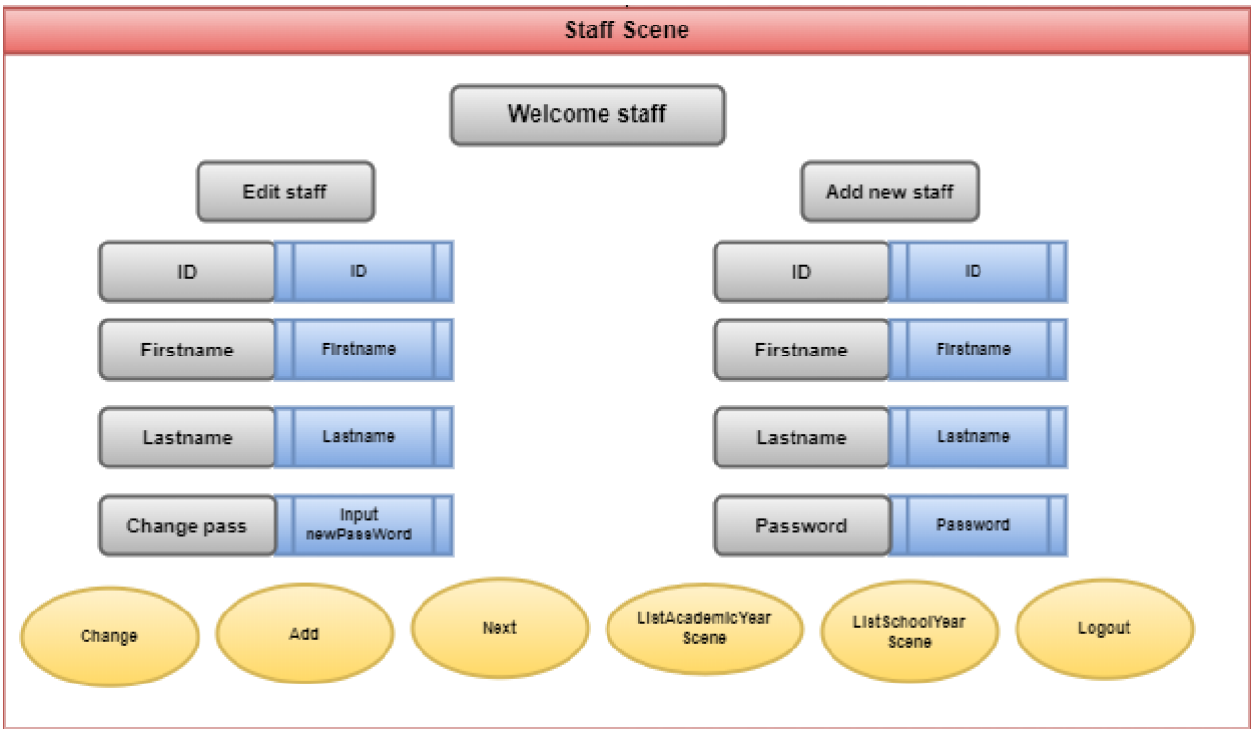
objects for the scene.

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.

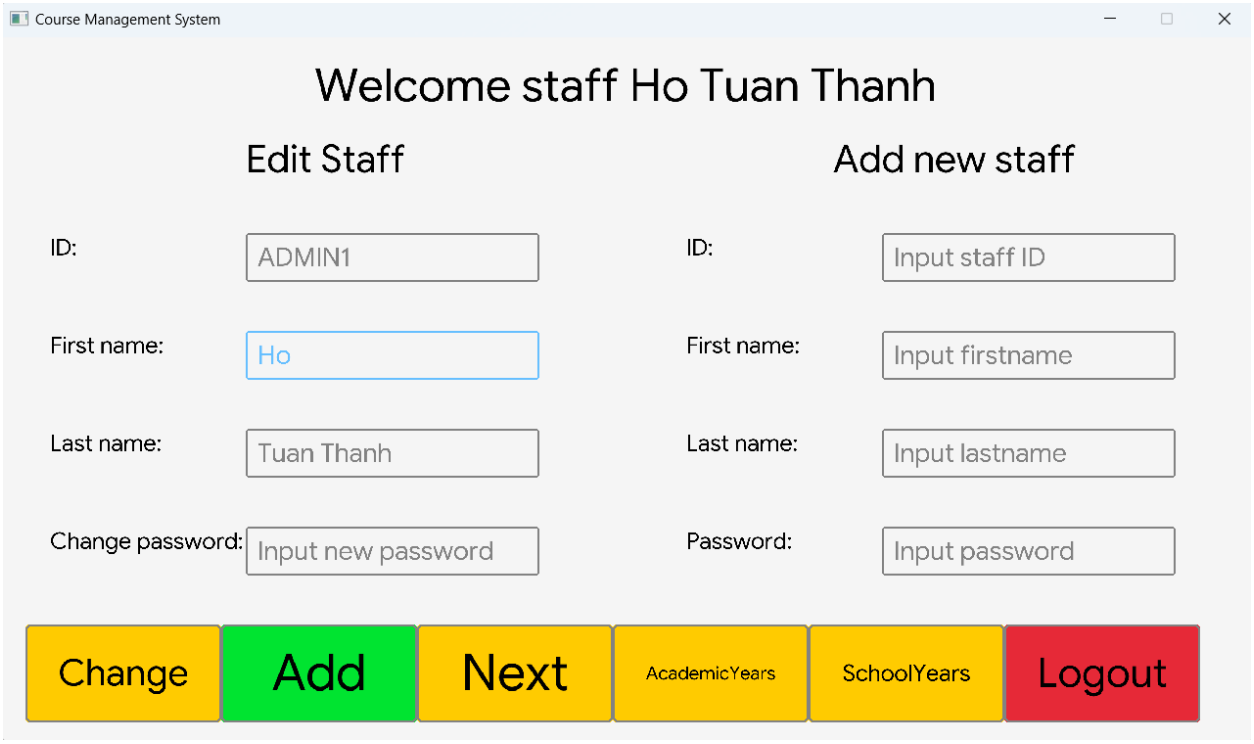
Staff scene

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

Theoretic scene



Practical scene



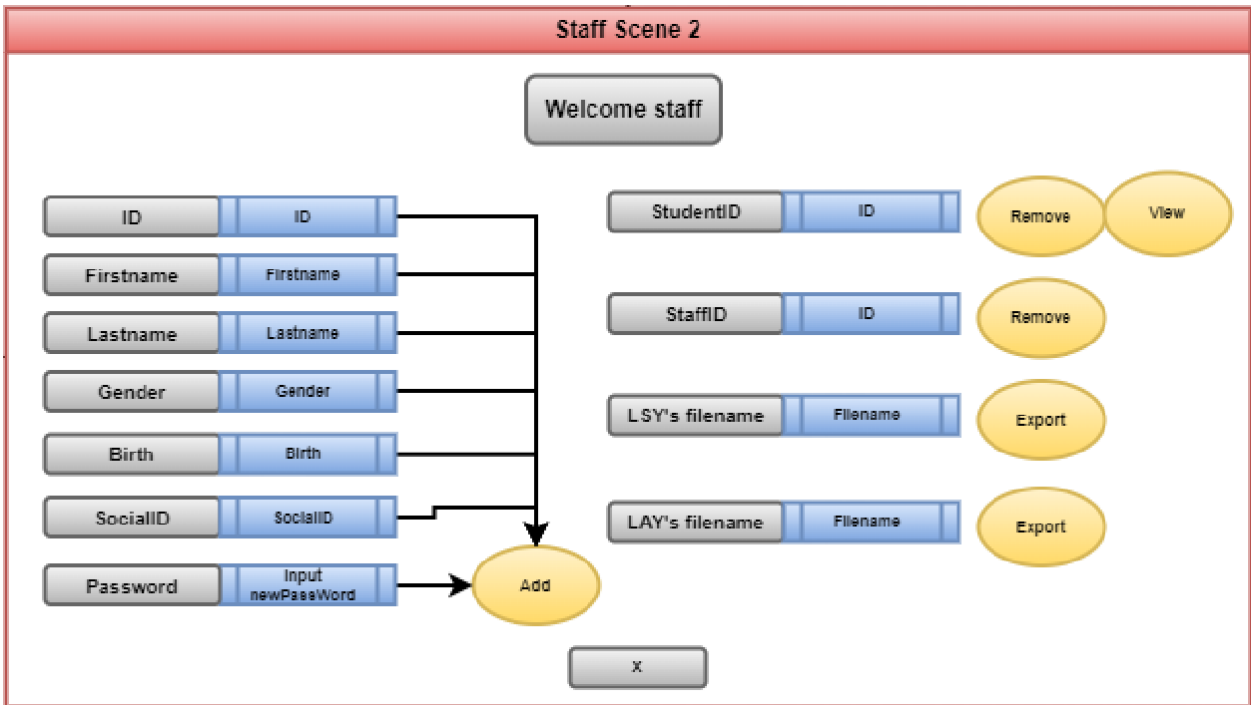
`StaffScene` is a derived class inherited from `Scene` class used to create a staff scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

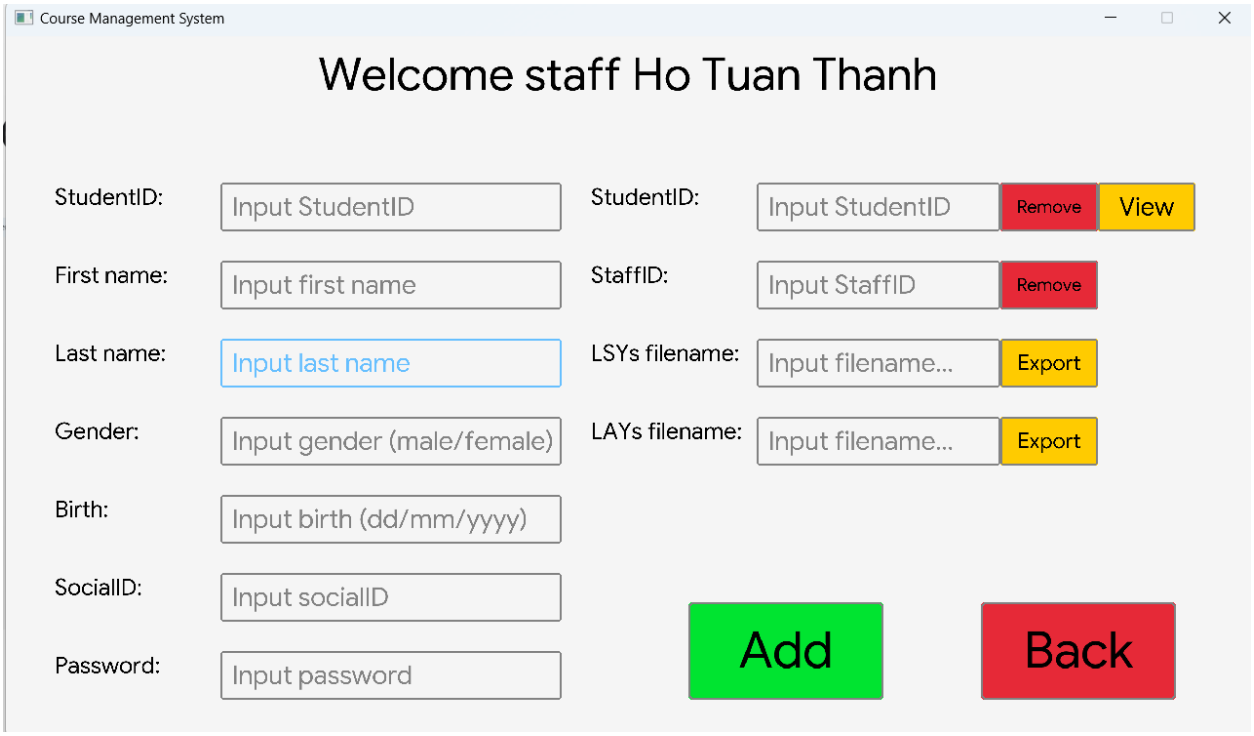
Add student from staff scene

```
(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

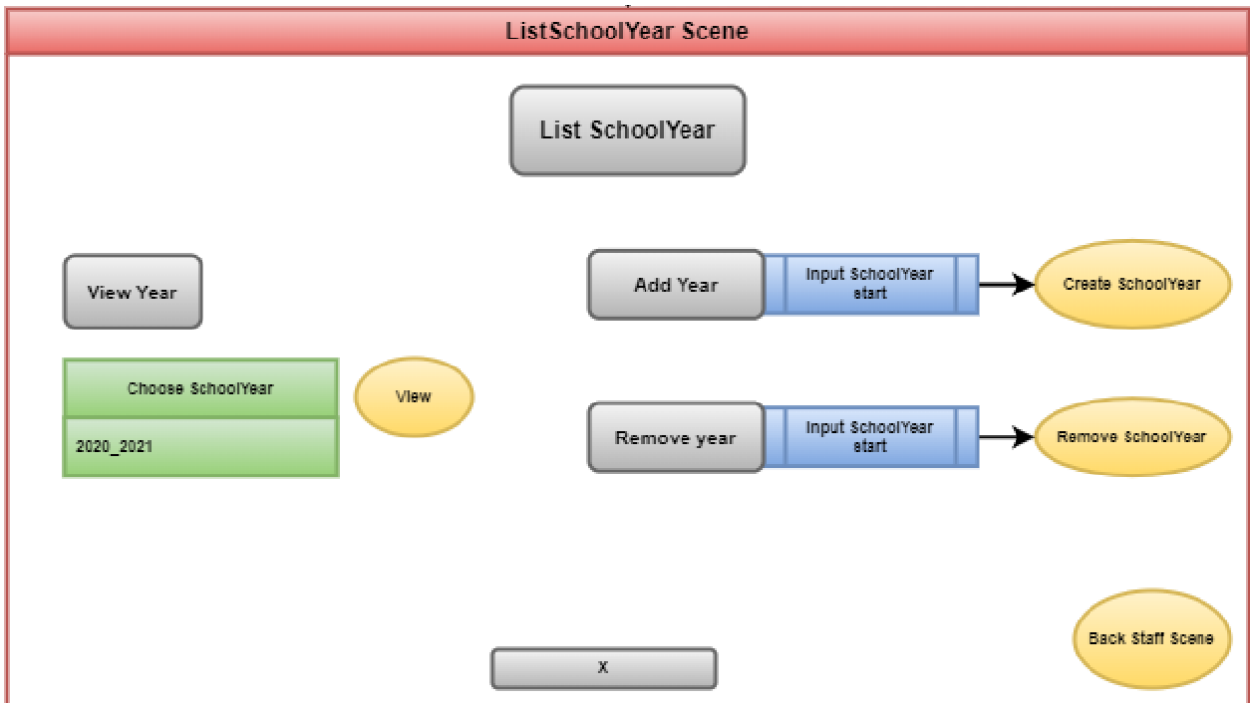
the objects for the scene.

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.

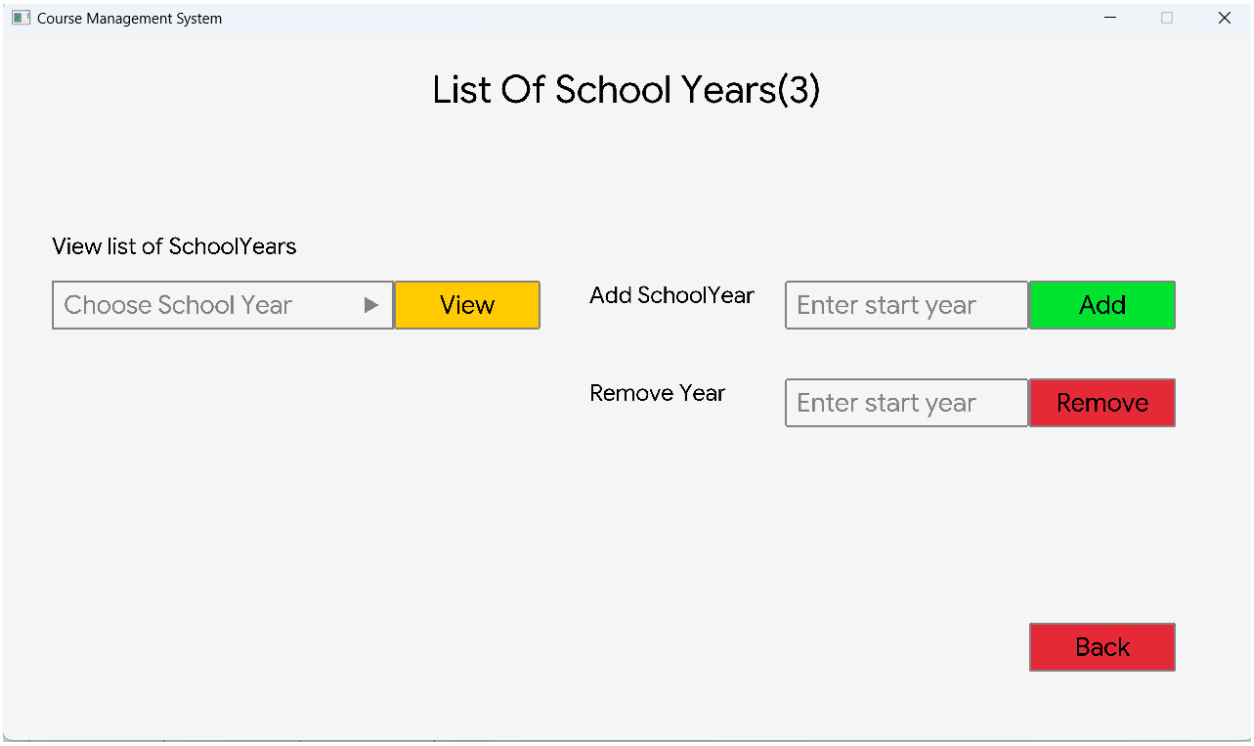
List school year scene

```
(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

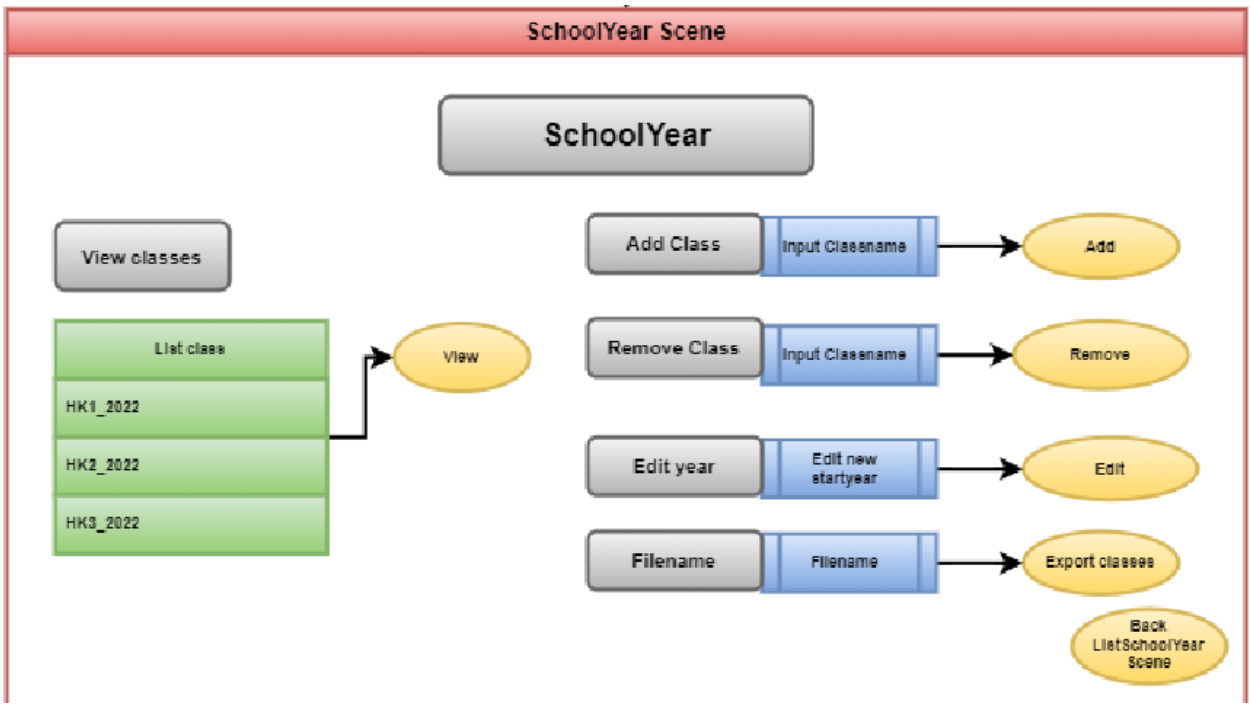
variables for the objects for the scene.

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.

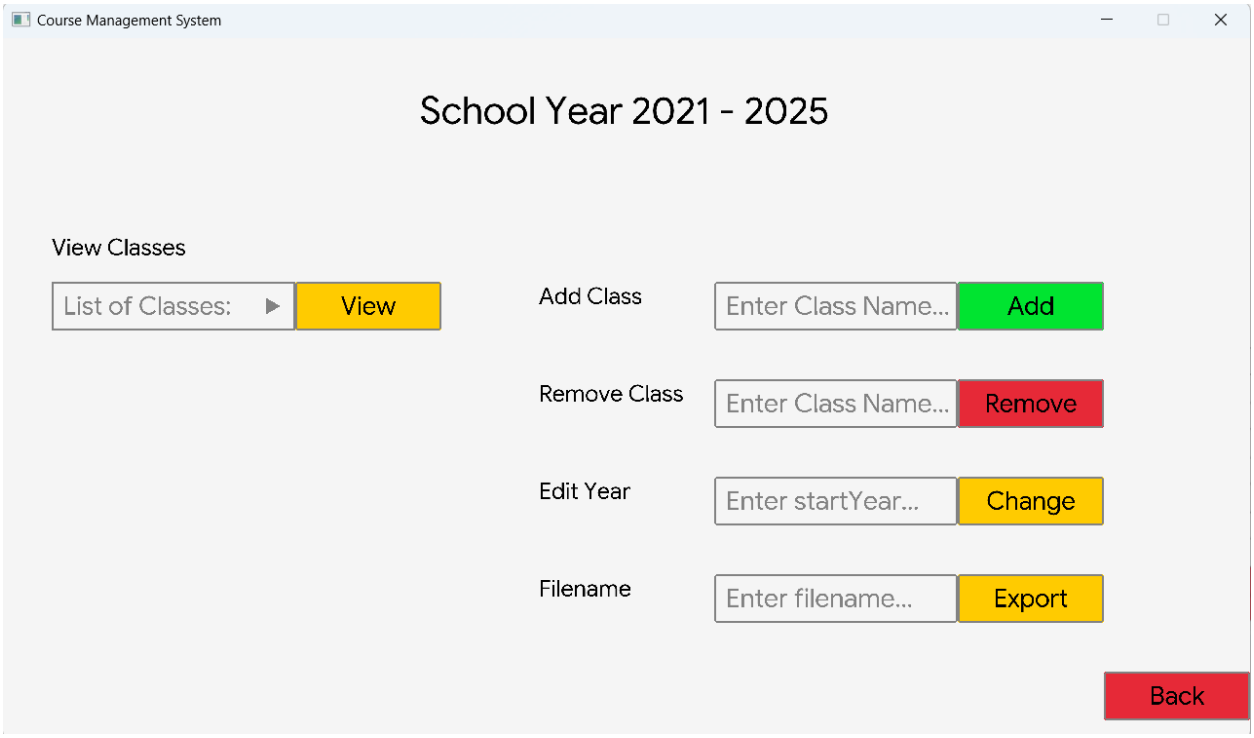
School year scene

```
(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

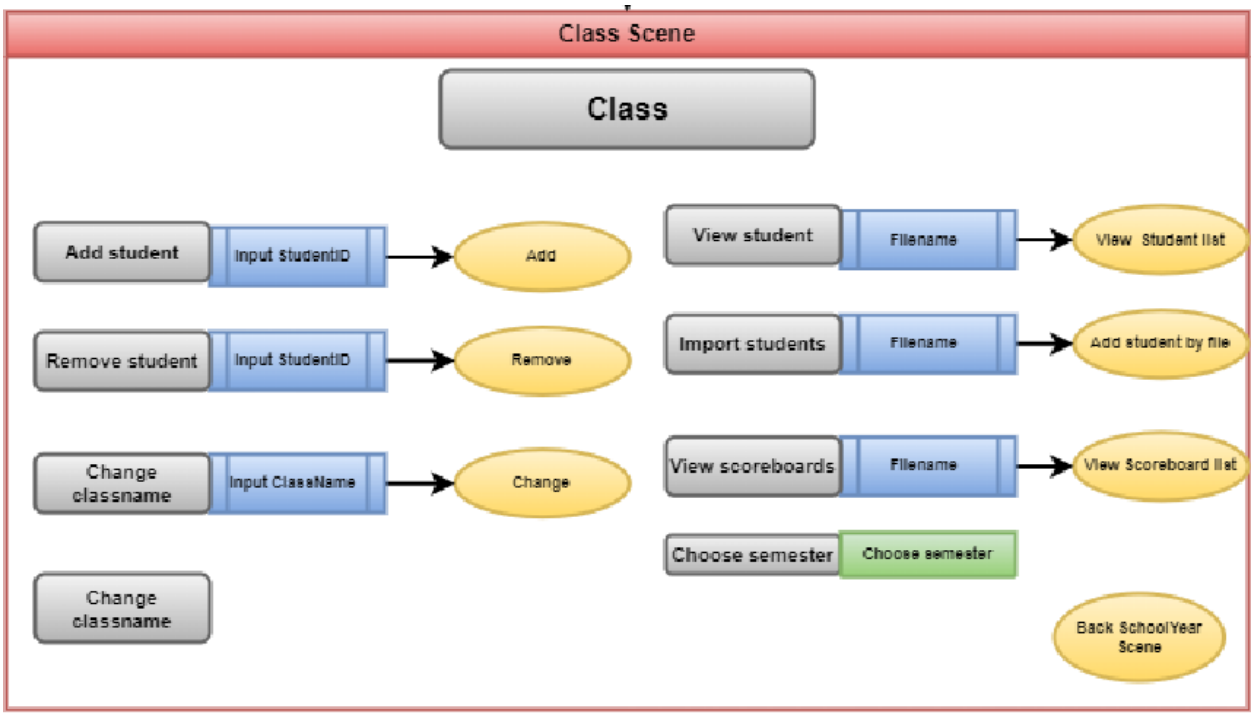
the objects for the scene.

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

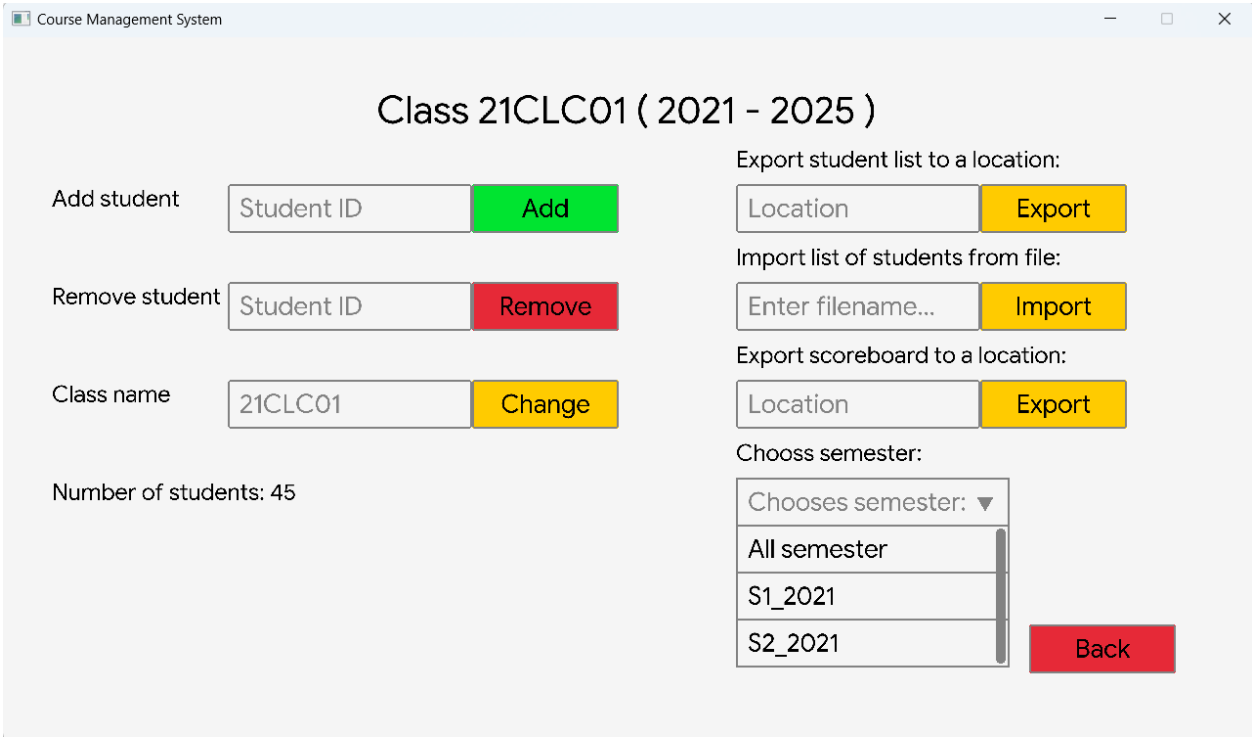
Class scene

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

Theoretical scene



Practical scene



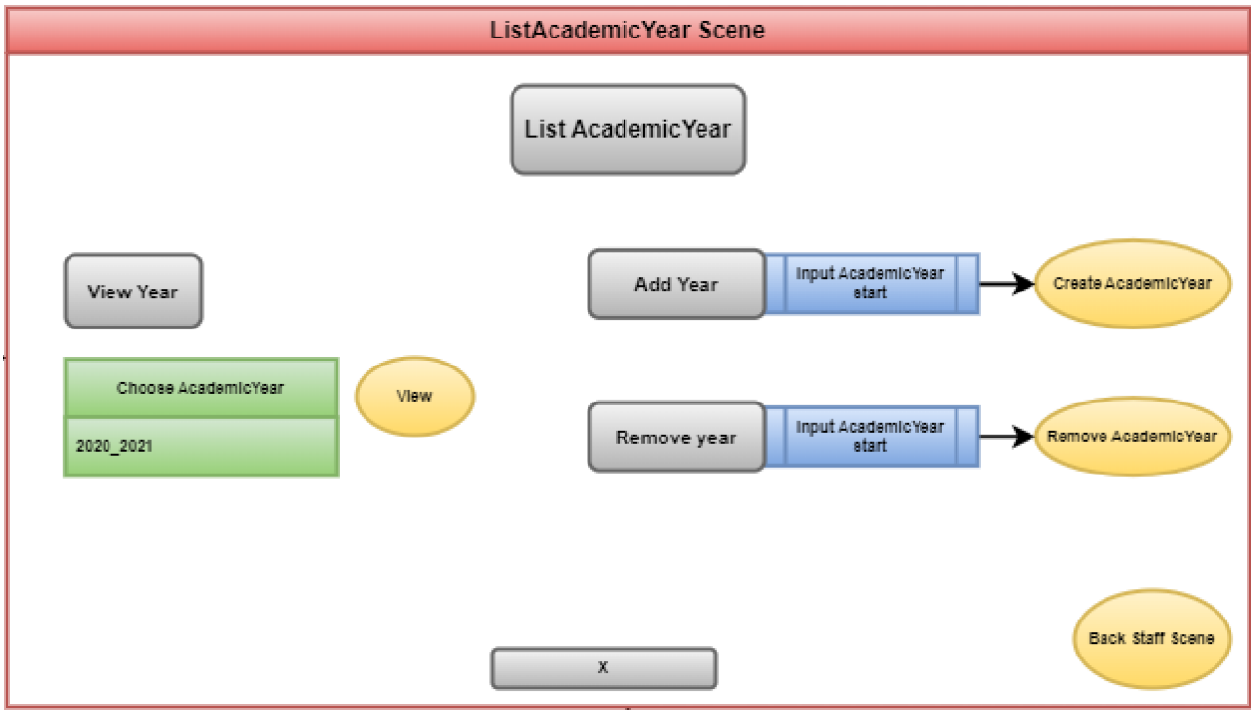
`ClassScene` is a derived class inherited from `Scene` class used to create a class scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

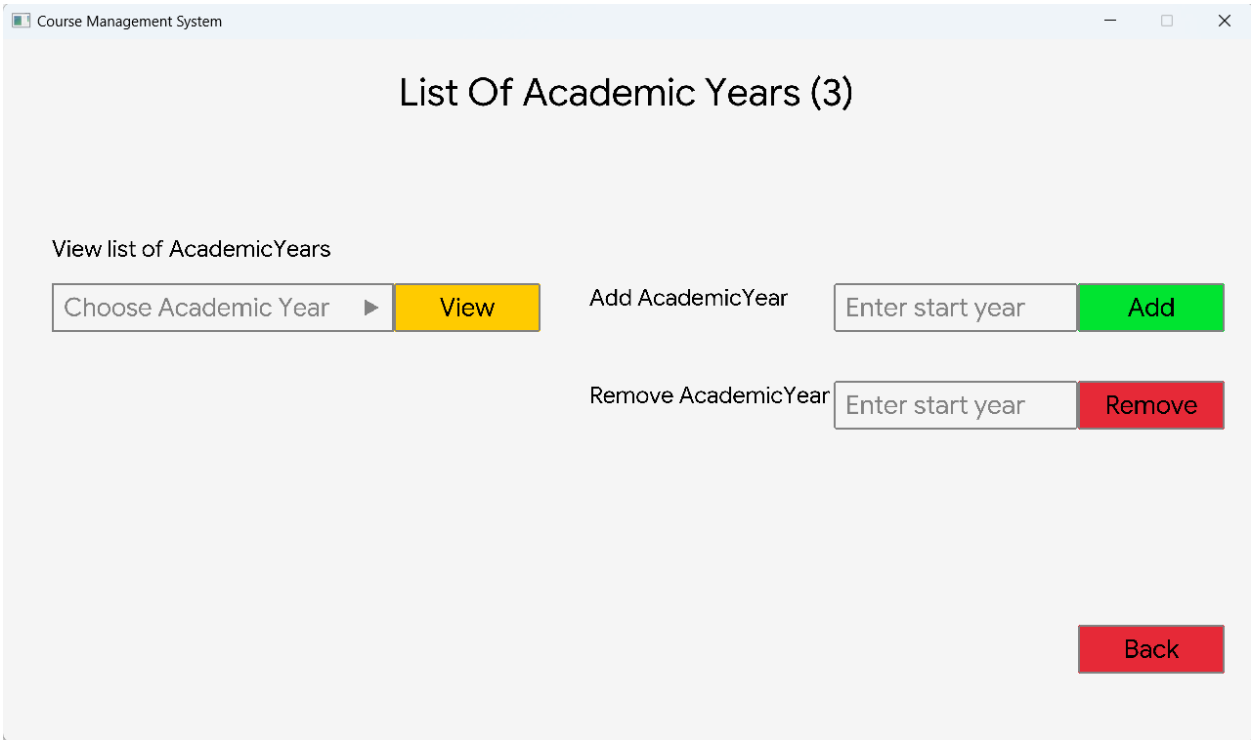
List academic year scene

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

Theoretic scene



Practical scene



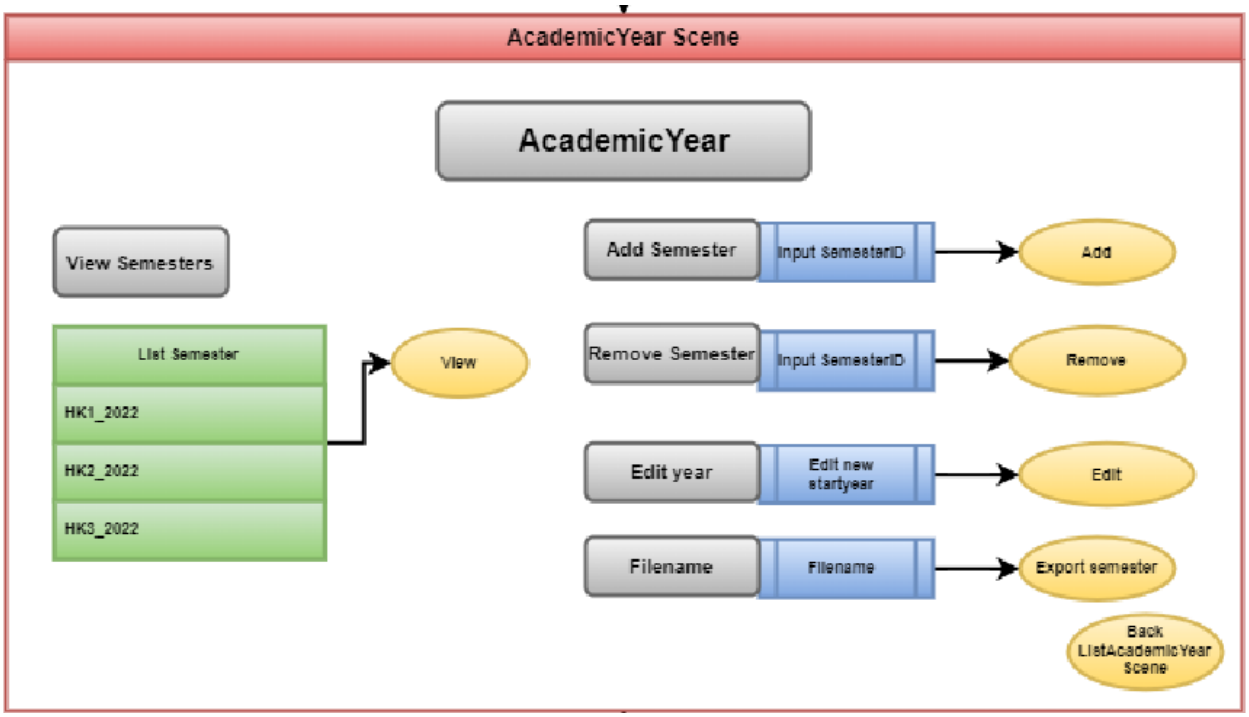
`ListAcademicYearScene` is a derived class inherited from `Scene` class used to create list academic year scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

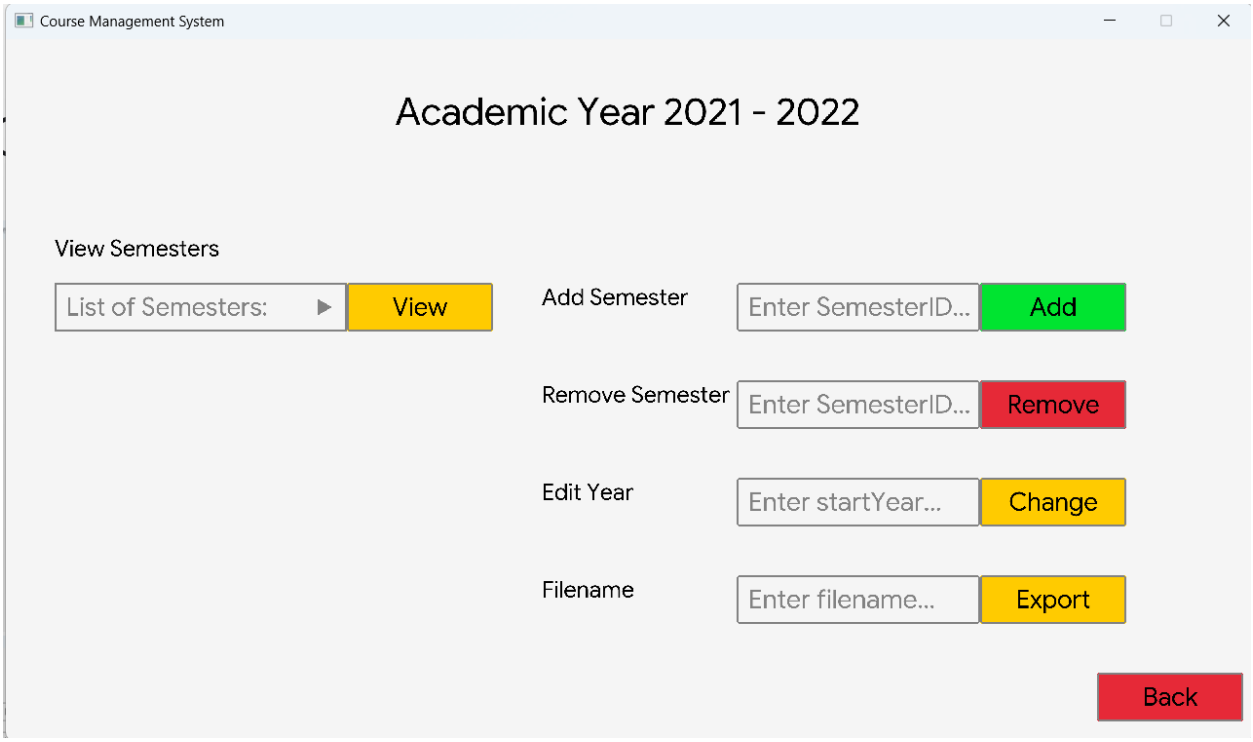
Academic year scene

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

Theoretic scene



Practical scene



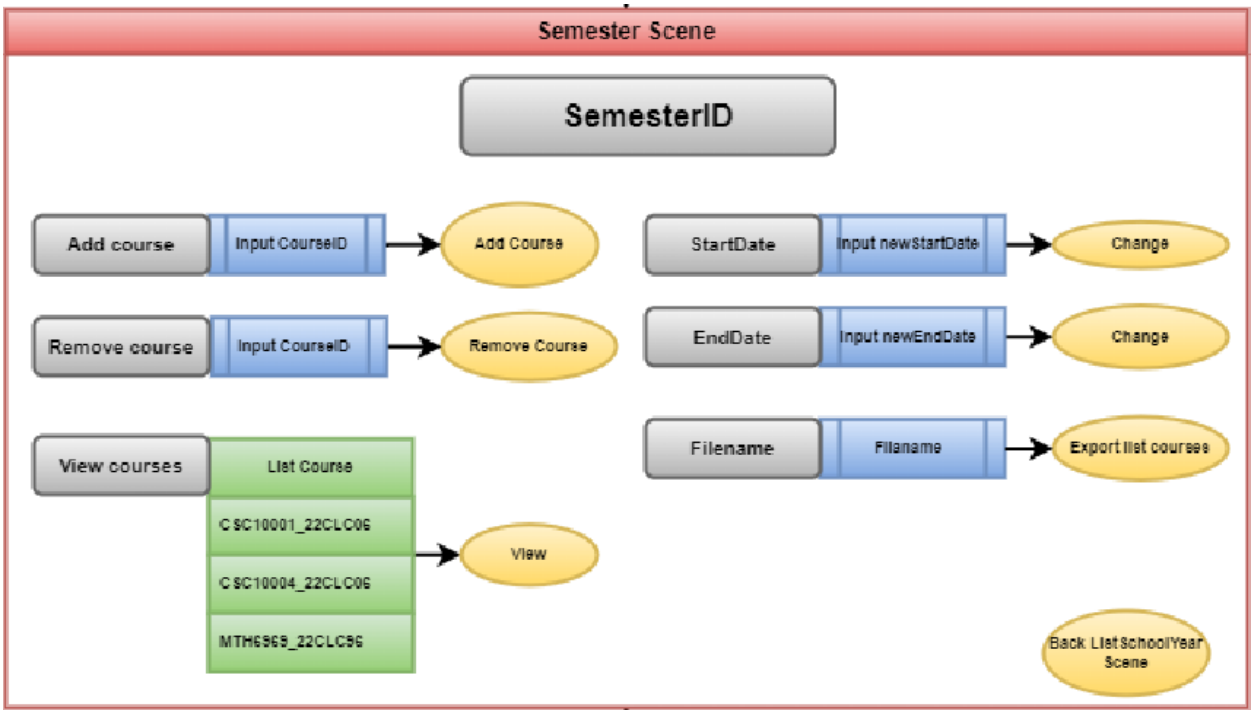
`AcademicYearScene` is a derived class inherited from `Scene` class used to create a academic year scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

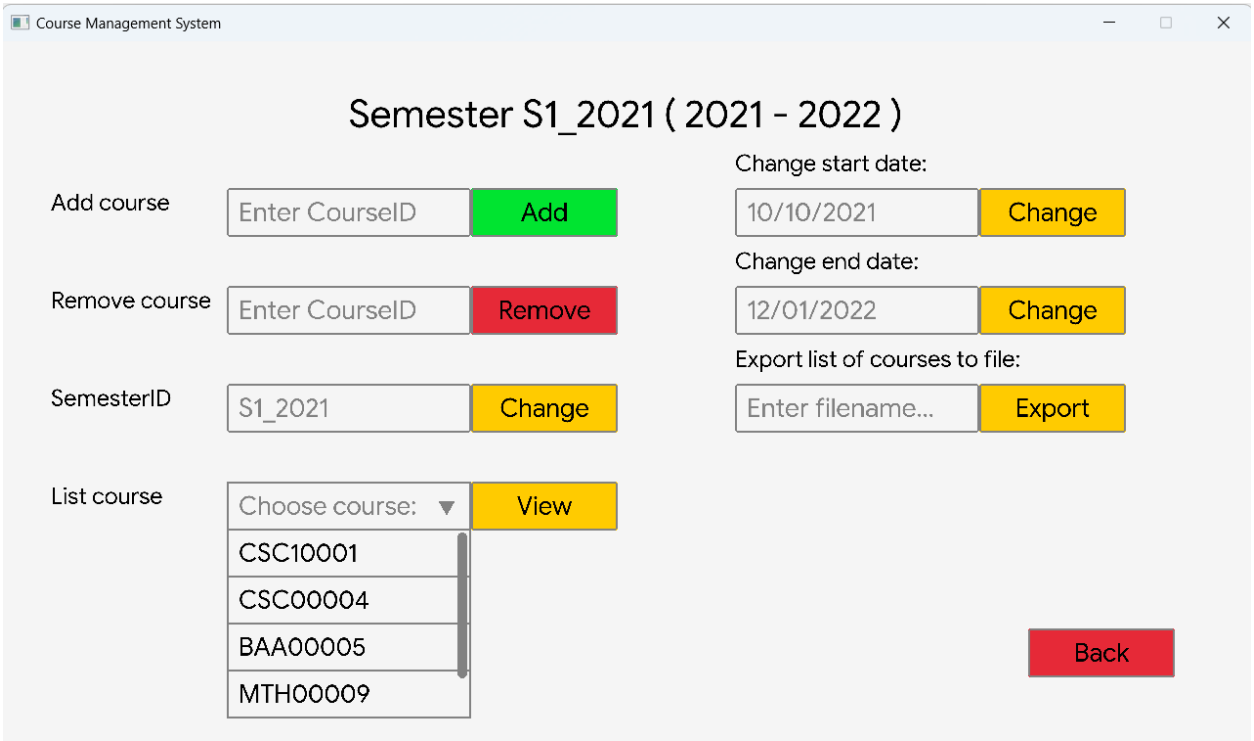
Semester scene

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

Theoretic scene



Practical scene



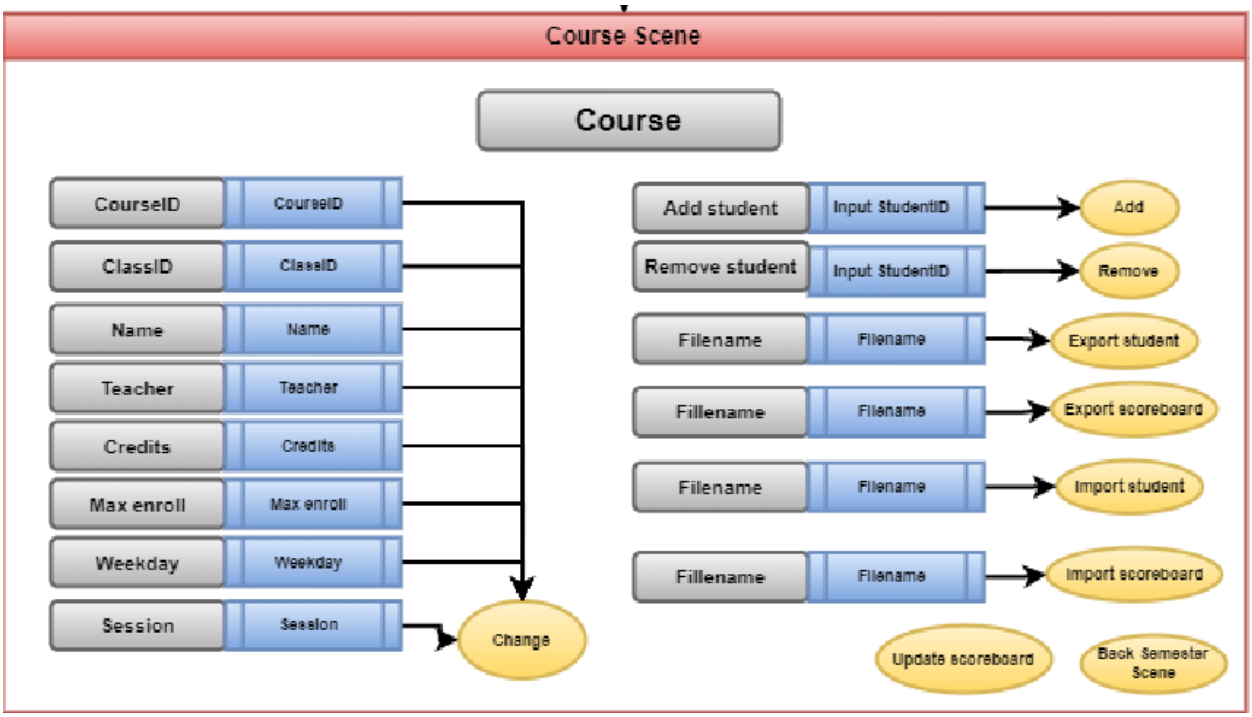
`SemesterScene` is a derived class inherited from `Scene` class used to create a semester scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

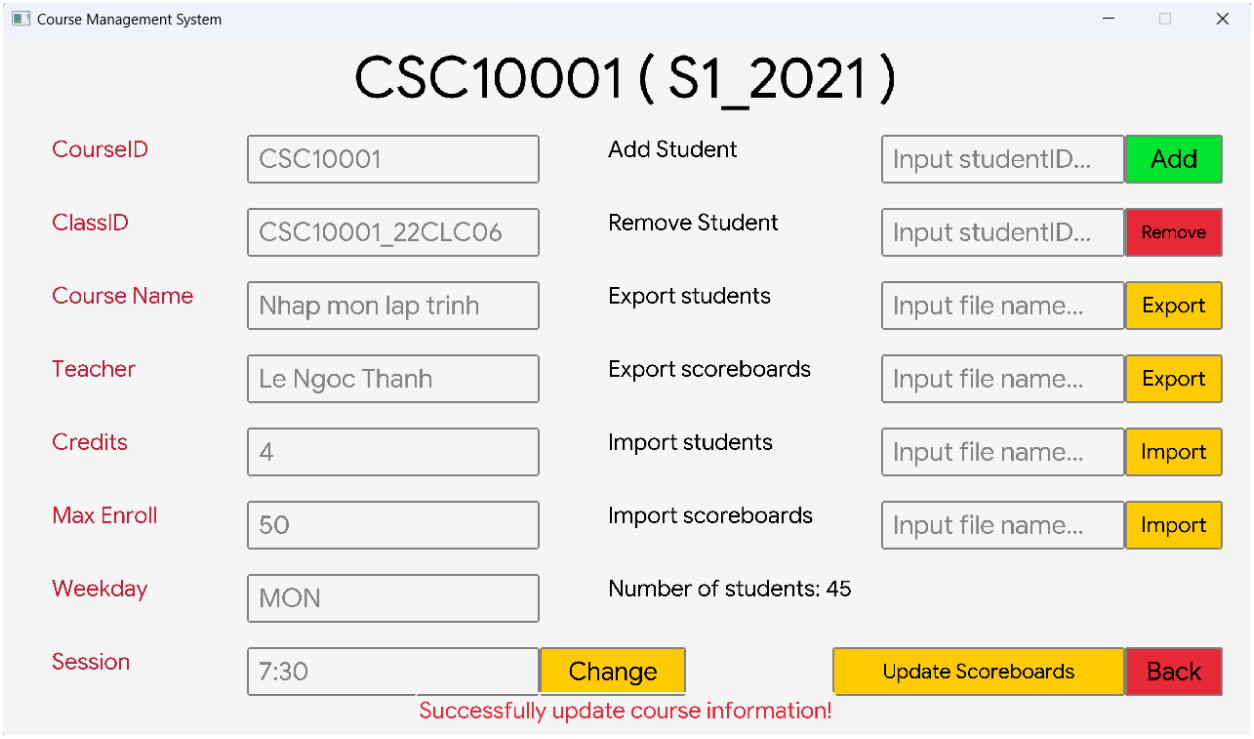
Course scene

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

Theoretic scene



Practical scene



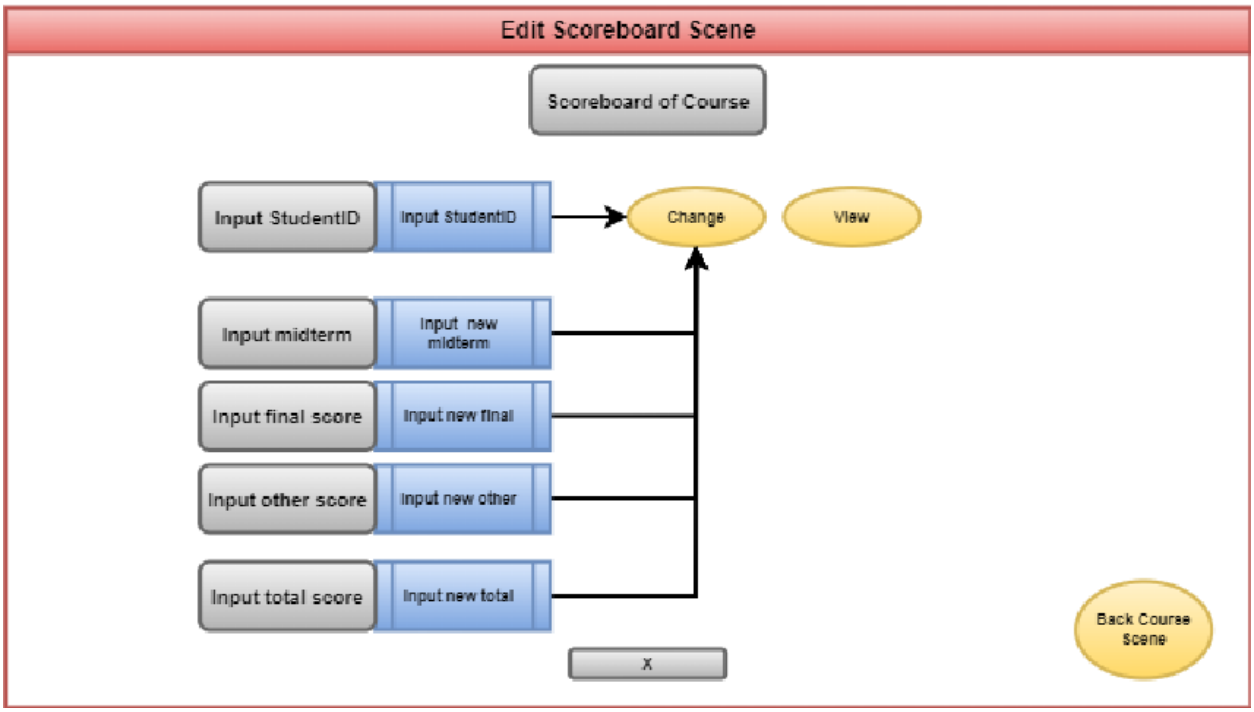
`CourseScene` is a derived class inherited from `Scene` class used to create a course scene for the application. All the members in the private class are the variables for the objects for the scene.

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.

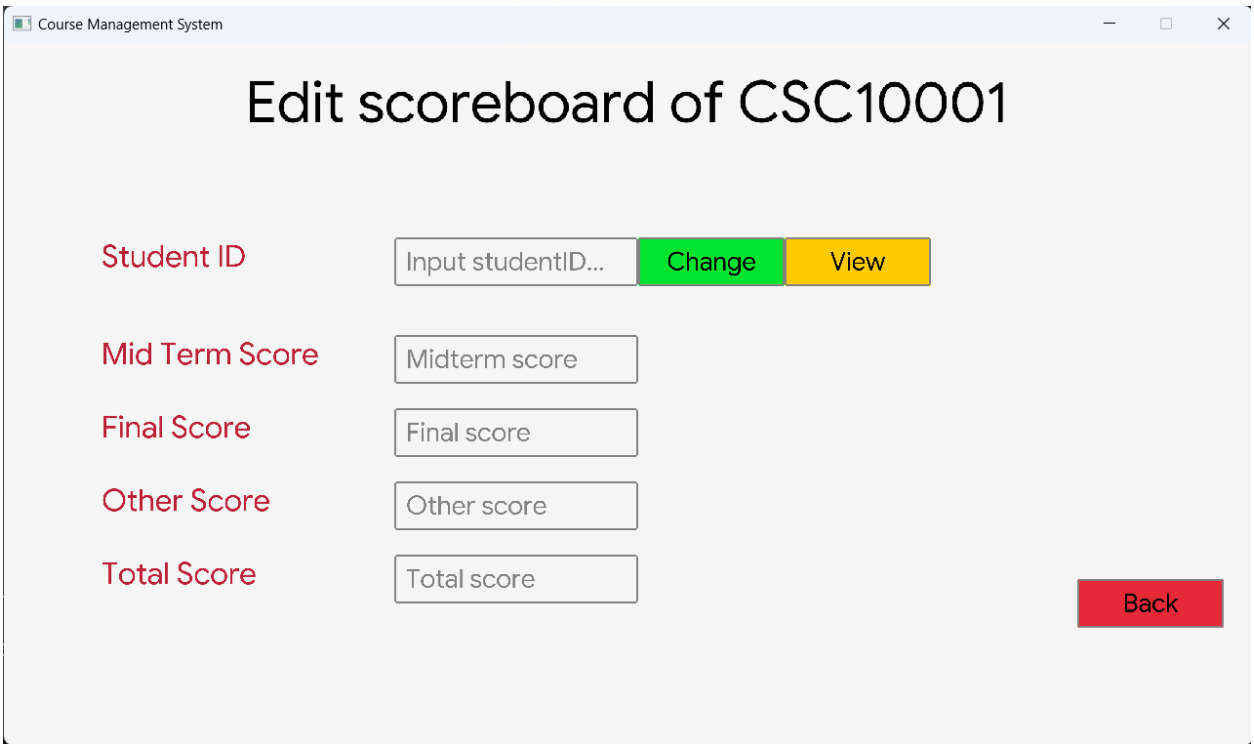
Edit course scene

```
(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.

Scene 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.