L.EIC Schedules

1.0

Generated by Doxygen 1.10.0

Chapter 1

L.EIC Schedules Management System

The L.EIC Schedules Management System project was developed for the Algorithms and Data Structures course in the 2023/24 academic year of the 2nd year of L.EIC at FEUP.

1.1 Project Description

Elaborating schedules for L.EIC classes can be a complex task. The purpose of this project is not the creation of the schedules, but rather the development of a system to manage schedules after they have been elaborated. The system must include various functionalities related to schedules, such as modifying, searching, viewing, sorting, listing, among others.

1.2 Dataset

The project uses a provided dataset available in `schedule.zip`, which contains real information about L.EIC's schedules for the 1st semester of the academic year 2022/2023 with anonymized student data. The dataset is split into three CSV files:

1. `classes_per_uc.csv`: Contains the existing classes in each course unit (UC).
2. `classes.csv`: Contains the schedules of classes.
3. `students_classes.csv`: Contains the classes of the students in each UC.

The dataset provides information about classes, students, and their schedules, which is essential for the functionality of the system.

1.3 Classes

1.3.1 Uc Class

The `Uc` class represents a course unit (UC) and provides methods for managing UC information.

1.3.2 Student Class

The `Student` class represents a student and provides methods for managing student information, including their schedule.

1.3.3 Lecture Class

The `Lecture` class represents a class lecture and includes information about the UC, class code, students enrolled, day, start time, duration, and lecture type.

1.3.4 Script Class

The `Script` class reads and processes data from CSV files, allowing the system to handle students and their schedules.

1.3.5 Request Class

The `Request` class handles various operations related to student enrollments, including adding, removing, and switching courses and classes.

1.4 Results

The program allows for the registration and management of various entities, making use of both linear (vector, list, stack, queue) and hierarchical data structures (binary search tree). Important information is saved in files for future use. The program also includes documentation for the code, generated using Doxygen, and indicates the time complexity of the most relevant functions and algorithms.

1.4.1 To run the project, run the following commands:

```
mkdir build
cd build
cmake ..
make
./aed_project
```

1.4.1.1 All documentation can be found inside the docs folder

1.4.1.2 To access the Adminstrator area, use the following credentials:

Login	Password
adm	123

1.4.2 Authors

Leonardo Garcia
Marcel Medeiros
Pedro Castro

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Student::Hash Struct Reference

Hash function for Student objects.
`#include <Student.hpp>`

Public Member Functions

- std::size_t operator() (const Student &student) const

4.1.1 Detailed Description

Hash function for Student objects.
Definition at line 85 of file Student.hpp.

4.1.2 Member Function Documentation

4.1.2.1 operator()()

```
std::size_t Student::Hash::operator() (
            const Student & student ) const  [inline]
```

Definition at line 87 of file Student.hpp.

```
00088         {
00089             // Combine the hash values of studentCode and studentName to create a unique hash for each
      student.
00090             return std::hash<std::string>{}(student.studentCode) ^
00091                    std::hash<std::string>{}(student.studentName);
00092         }
```

The documentation for this struct was generated from the following file:

- inc/Student.hpp

4.2 Lecture Class Reference

Represents a lecture.
`#include <Lecture.hpp>`

Public Member Functions

- Lecture (const std::string &ucCode)

  Constructor for Lecture with a UcCode.
- Lecture (const std::string &ucCode, const std::string &classCode, const std::string &weekDay, const double &startHour, const double &duration, const std::string &type)

  Constructor for Lecture with specific details.
- Uc getUc ()

  Get the Uc associated with this lecture.

- std::string getClassCode ()

  Get the class code of the lecture.
- void setUc (const Uc &uc)

  Set the Uc object associated with this lecture.
- void addStudent (Student &student)

  Add a student to the lecture.
- void removeStudent (const Student &student)

  Remove a student from the lecture.
- std::vector< Student > getStudents ()

  Get a vector of students enrolled in the lecture.
- std::string getWeekDay () const

  Get the day of the week when the lecture occurs.
- void setWeekDay (const std::string &weekDay)

  Set the day of the week when the lecture occurs.
- double getStartHour () const

  Get the starting hour of the lecture.
- void setStartHour (const double &startHour)

  Set the starting hour of the lecture.
- double getDuration () const

  Get the duration of the lecture.
- void setDuration (const double &duration)

  Set the duration of the lecture.
- std::string getType () const

  Get the type of the lecture.
- void setType (const std::string &type)

  Set the type of the lecture.
- bool operator== (Lecture &other)

  Overloaded equality operator to compare two lectures for equality.
- bool operator< (const Lecture &other) const

  Compare two lectures to determine their order.
- bool overlay (Lecture &other)

  Check if this lecture's time slot overlaps with another lecture's time slot.

Private Attributes

- Uc uc
- std::string classCode
- std::vector< Student > students
- std::string weekDay
- double startHour
- double duration
- std::string type

4.2.1 Detailed Description

Represents a lecture.
Definition at line 10 of file Lecture.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Lecture() [1/2]

```
Lecture::Lecture (
            const std::string & ucCode )
```

Constructor for Lecture with a UcCode.

Parameters

ucCode	The UcCode associated with the lecture.

4.2.2.2 Lecture() [2/2]

```
Lecture::Lecture (
            const std::string & ucCode,
            const std::string & classCode,
            const std::string & weekDay,
            const double & startHour,
            const double & duration,
            const std::string & type )
```

Constructor for Lecture with specific details.

Parameters

ucCode	The UcCode associated with the lecture.
classCode	The class code.
weekDay	The day of the week when the lecture occurs.
startHour	The starting hour of the lecture.
duration	The duration of the lecture.
type	The type of lecture.

Definition at line 8 of file Lecture.cpp.

```
00008
      : uc(ucCode)
00009 {
00010     this->uc.setUcCode(ucCode);
00011     this->uc.addClass(classCode);
00012     this->classCode = classCode;
00013     this->weekDay = weekDay;
00014     this->startHour = startHour;
00015     this->duration = duration;
00016     this->type = type;
00017 }
```

4.2.3 Member Function Documentation

4.2.3.1 addStudent()

```
void Lecture::addStudent (
            Student & student )
```

Add a student to the lecture.

Parameters

student	The student to add.

Definition at line 34 of file Lecture.cpp.

```
00035 {
00036     for (auto it = this->students.begin(); it != this->students.end(); it++)
00037         if (*it == student)
00038             return;
00039     this->students.push_back(student);
00040 }
```

4.2.3.2 getClassCode()

```
string Lecture::getClassCode ( )
```

Get the class code of the lecture.

Returns

The class code as a string.

Definition at line 19 of file Lecture.cpp.

```
00020 {
00021     return this->classCode;
00022 }
```

4.2.3.3 getDuration()

`double Lecture::getDuration ( ) const`

Get the duration of the lecture.

Returns

The duration as a double.

Definition at line 82 of file Lecture.cpp.

```
00083 {
00084     return this->duration;
00085 }
```

4.2.3.4 getStartHour()

`double Lecture::getStartHour ( ) const`

Get the starting hour of the lecture.

Returns

The start hour as a double.

Definition at line 72 of file Lecture.cpp.

```
00073 {
00074     return this->startHour;
00075 }
```

4.2.3.5 getStudents()

`vector< Student > Lecture::getStudents ( )`

Get a vector of students enrolled in the lecture.

Returns

A vector of Student objects.

Definition at line 57 of file Lecture.cpp.

```
00058 {
00059     return this->students;
00060 }
```

4.2.3.6 getType()

`string Lecture::getType ( ) const`

Get the type of the lecture.

Returns

The type as a string.

Definition at line 92 of file Lecture.cpp.

```
00093 {
00094     return this->type;
00095 }
```

4.2.3.7 getUc()

`Uc Lecture::getUc ( )`

Get the Uc associated with this lecture.

Returns

The Uc object.

Definition at line 24 of file Lecture.cpp.

```
00025 {
00026     return this->uc;
00027 }
```

4.2.3.8 getWeekDay()

```
string Lecture::getWeekDay ( ) const
```

Get the day of the week when the lecture occurs.

Returns

The week day as a string.

Definition at line 62 of file Lecture.cpp.

```
00063 {
00064     return this->weekDay;
00065 }
```

4.2.3.9 operator<()

```
bool Lecture::operator< (
            const Lecture & other ) const
```

Compare two lectures to determine their order.

Parameters

other	The lecture to compare with.

Returns

True if this lecture starts before the other, otherwise false.

Definition at line 112 of file Lecture.cpp.

```
00113 {
00114 std::map<std::string, int> dayValues = {
00115             {"Monday", 0},
00116             {"Tuesday", 1},
00117             {"Wednesday", 2},
00118             {"Thursday", 3},
00119             {"Friday", 4},
00120             {"Saturday", 5},
00121             {"Sunday", 6}
00122         };
00123
00124 if(weekDay != other.getWeekDay()){
00125     return dayValues.at(weekDay) < dayValues.at(other.getWeekDay());
00126 }
00127 return startHour < other.getStartHour();
00128 }
```

4.2.3.10 operator==()

```
bool Lecture::operator== (
            Lecture & other )
```

Overloaded equality operator to compare two lectures for equality.

Parameters

other	The lecture to compare with.

Returns

True if the lectures are equal, otherwise false.

Definition at line 102 of file Lecture.cpp.

```
00103 {
00104     if ((this->uc.getUcCode() == other.getUc().getUcCode()) && (this->classCode ==
      other.getClassCode()))
00105     {
00106         return true;
00107     }
00108     else
00109         return false;
00110 }
```

4.2.3.11 overlay()

```
bool Lecture::overlay (
            Lecture & other )
```

Check if this lecture's time slot overlaps with another lecture's time slot.

Parameters

other	The other lecture to check for overlap.

Returns

True if there is an overlap, otherwise false.

Definition at line 130 of file Lecture.cpp.

```
00130                                       {
00131     if(weekDay != other.getWeekDay()) return false;
00132
00133     string o_type = other.getType();
00134     if((type == "TP" && o_type == "TP") || (type == "PL" && o_type == "PL") || (type == "TP" && o_type
      == "PL") || (type == "PL" && o_type == "TP")){
00135         if((startHour >= other.getStartHour()) && startHour <
      (other.getStartHour()+other.getDuration())) return true;
00136         if((startHour < other.getStartHour()) && (startHour + duration) > other.getStartHour()) return
      true;
00137         if(startHour==other.getStartHour()) return true;
00138     }
00139
00140     return false;
00141 }
```

4.2.3.12 removeStudent()

```
void Lecture::removeStudent (
            const Student & student )
```

Remove a student from the lecture.

Parameters

student	The student to remove.

Definition at line 42 of file Lecture.cpp.

```
00043 {
00044     int mark;
00045     for (size_t i = 0; i < this->students.size(); i++)
00046     {
00047         if (this->students.at(i) == student)
00048         {
00049             mark = i;
00050         }
00051     }
00052     auto it = this->students.begin();
00053     advance(it, mark);
00054     this->students.erase(it);
00055 }
```

4.2.3.13 setDuration()

```
void Lecture::setDuration (
            const double & duration )
```

Set the duration of the lecture.

Parameters

duration	The duration to set.

Definition at line 87 of file Lecture.cpp.

```
00088 {
00089     this->duration = duration;
00090 }
```

4.2.3.14 setStartHour()

```
void Lecture::setStartHour (
            const double & startHour )
```

Set the starting hour of the lecture.

Parameters

startHour	The start hour to set.

Definition at line 77 of file Lecture.cpp.

```
00078 {
00079     this->startHour = startHour;
00080 }
```

4.2.3.15 setType()

```
void Lecture::setType (
            const std::string & type )
```

Set the type of the lecture.

Parameters

type	The type to set.

Definition at line 97 of file Lecture.cpp.

```
00098 {
00099     this->type = type;
00100 }
```

4.2.3.16 setUc()

```
void Lecture::setUc (
            const Uc & uc )
```

Set the Uc object associated with this lecture.

Parameters

uc	The Uc object to set.

Definition at line 29 of file Lecture.cpp.

```
00030 {
00031     this->uc = uc;
00032 }
```

4.2.3.17 setWeekDay()

```
void Lecture::setWeekDay (
```

```
            const std::string & weekDay )
```

Set the day of the week when the lecture occurs.

Parameters

weekDay	The week day to set.

Definition at line 67 of file Lecture.cpp.

```
00068 {
00069     this->weekDay = weekDay;
00070 }
```

4.2.4 Member Data Documentation

4.2.4.1 classCode

```
std::string Lecture::classCode  [private]
```

The class code of the lecture.
Definition at line 141 of file Lecture.hpp.

4.2.4.2 duration

```
double Lecture::duration  [private]
```

The duration of the lecture.
Definition at line 153 of file Lecture.hpp.

4.2.4.3 startHour

```
double Lecture::startHour  [private]
```

The starting hour of the lecture.
Definition at line 150 of file Lecture.hpp.

4.2.4.4 students

```
std::vector<Student> Lecture::students  [private]
```

The list of students enrolled in the lecture.
Definition at line 144 of file Lecture.hpp.

4.2.4.5 type

```
std::string Lecture::type  [private]
```

The type of lecture.
Definition at line 156 of file Lecture.hpp.

4.2.4.6 uc

```
Uc Lecture::uc  [private]
```

The Uc associated with the lecture.
Definition at line 138 of file Lecture.hpp.

4.2.4.7 weekDay

```
std::string Lecture::weekDay  [private]
```

The day of the week when the lecture occurs.
Definition at line 147 of file Lecture.hpp.
The documentation for this class was generated from the following files:

- inc/Lecture.hpp
- src/Lecture.cpp

4.3 Request Class Reference

Represents a request to perform various operations related to student enrollments.
`#include <Request.hpp>`

Public Member Functions

- Request ()

  Default constructor for the Request class.
- Request (std::string studentCode, char type)

  Constructor for the Request class to create a new request.
- bool addUc (std::string ucCodeDestination)

  Adds a student to a specified course and class.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- bool removeUc (std::string ucCode)

  Removes a student from a specified course.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- bool switchUc (std::string ucOrigin, std::string ucDestination)

  Switches a student from one course to another, preserving their schedule.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- bool switchClass (std::string uc, std::string classOrigin, std::string classDestination)

  Switches a student from one class to another within the same course.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- void studentRequests (const std::string &studentCode)

  Displays the list of requests for a specific student.
  Time complexity: O(N), where N is the number of lines in the requests_log.csv file.
- void adminRequests ()

  Displays all requests.
  Time complexity: O(N), where N is the number of lines in the requests_log.csv file.
- void undoRequest (unsigned id)

  Undoes a specific request by its ID.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- bool classesCheck (std::string uc, std::queue< std::string > &eligibleClasses)

  Checks the eligibility of available classes for a student's UC request.
  It evaluates if the student can be assigned to a class without violating capacity, balance, and schedule constraints.
  Time complexity: O(N^2), where N is the number of lines in the classes.csv file.

Private Attributes

- unsigned id
- std::string studentCode
- char type
- bool flag = false

4.3.1 Detailed Description

Represents a request to perform various operations related to student enrollments.
Request class provides methods for adding, removing, and switching courses and classes for students.
Definition at line 14 of file Request.hpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Request() [1/2]

```
Request::Request ( )  [inline]
```

Default constructor for the Request class.
Definition at line 18 of file Request.hpp.

```
00018 {};
```

4.3.2.2 Request() [2/2]

```
Request::Request (
            std::string studentCode,
            char type )
```

Constructor for the Request class to create a new request.

Parameters

studentCode	The student's unique code.
type	The type of request (1: Add Uc, 2: Remove Uc, 3: Switch Uc, 4: Switch Class).

4.3.3 Member Function Documentation

4.3.3.1 addUc()

```
bool Request::addUc (
            std::string ucCodeDestination )
```

Adds a student to a specified course and class.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

ucCodeDestination	The code of the course to add the student.

Returns

True if the student is successfully added, false otherwise.

Definition at line 66 of file Request.cpp.

```
00067 {
00068     Script script;
00069     Student newStudent = script.loadStudent(this->studentCode);
00070     map<std::string, std::string> new_schedule = newStudent.getSchedule();
00071     if (new_schedule.find(ucCodeDestination) != new_schedule.end())
00072     {
00073         throw runtime_error("Student already registered in this UC");
00074         return this->flag;
00075     }
00076
00077     if (newStudent.getSchedule().size() >= 7)
00078     {
00079         throw runtime_error("Student registered in maximum number of UC's");
00080         return this->flag;
00081     }
00082
00083     int max = 0;
00084     int min = 100;
00085     queue<string> eligibleClasses = {};
00086     if(!classesCheck(ucCodeDestination, eligibleClasses)) return this->flag;
00087
00088     ofstream outFile("../data/students_classes.csv", ios::app);
00089
00090     if (!outFile.is_open())
00091     {
00092         cerr << "Couldnt open file." << endl;
00093         return this->flag;
00094     }
00095
00096     outFile << newStudent.getstudentCode() << ',' << newStudent.getstudentName() << ',' <<
      ucCodeDestination << ',' << eligibleClasses.front() << endl;
00097
00098     outFile.close();
00099
00100     ofstream write_log("../requests_log.csv", ios::app);
00101     write_log << id << ',' << type << ',' << studentCode << ',' << ucCodeDestination << ',' <<
      eligibleClasses.front() << endl;
00102     write_log.close();
00103
00104     this->flag = true;
00105     return this->flag;
00106 }
```

4.3.3.2 adminRequests()

```
void Request::adminRequests ( )
```

Displays all requests.
Time complexity: O(N), where N is the number of lines in the requests_log.csv file.
Definition at line 329 of file Request.cpp.

```
00330 {
00331     ifstream read_file("../requests_log.csv");
00332     string line;
00333     while (getline(read_file, line))
00334     {
00335         istringstream iss(line);
00336         string id_, type_, studentCode_;
00337 
00338         getline(getline(getline(iss, id_, ','), type_, ','), studentCode_, ',');
00339 
00340         if (type_ == "1")
00341         {
00342             string ucCode_, classCode_;
00343             getline(getline(iss, ucCode_, ','), classCode_, '\r');
00344             cout « "Operation ID: " « id_ « " | Student " « studentCode_ « " added the UC " « ucCode_
      « " and entered the class " « classCode_ « endl;
00345         }
00346         else if (type_ == "2")
00347         {
00348             string ucCode_;
00349             getline(iss, ucCode_, '\r');
00350             cout « "Operation ID: " « id_ « " | Student " « studentCode_ « " removed the UC " «
      ucCode_ « endl;
00351         }
00352         else if (type_ == "3")
00353         {
00354             string ucOrigin_, ucDestination_, classCode_;
00355             getline(getline(getline(iss, ucOrigin_, ','), ucDestination_, ','), classCode_, '\r');
00356             cout « "Operation ID: " « id_ « " |  Student " « studentCode_ « " switched from UC " «
      ucOrigin_ « " to the UC " « ucDestination_ « " and was added to the class " « classCode_ « endl;
00357         }
00358         else if (type_ == "4")
00359         {
00360 
00361             string ucOrigin_, classOrigin_, classDestination_;
00362             getline(getline(getline(iss, ucOrigin_, ','), classOrigin_, ','), classDestination_,
      '\r');
00363             cout « "Operation ID: " « id_ « " |  Student " « studentCode_ « " switched from class " «
      classOrigin_ « " of the UC " « ucOrigin_ « " to the class " « classDestination_ « endl;
00364         }
00365     }
00366     read_file.close();
00367 }
```

4.3.3.3 classesCheck()

```
bool Request::classesCheck (
            std::string uc,
            std::queue< std::string > & eligibleClasses )
```

Checks the eligibility of available classes for a student's UC request.
It evaluates if the student can be assigned to a class without violating capacity, balance, and schedule constraints.
Time complexity: $O(N^2)$, where N is the number of lines in the classes.csv file.

Parameters

ucDestination	The UC code for which the student is requesting enrollment.
eligibleClasses	A queue containing eligible class codes for the student's request.

Returns

true if the student can be assigned to a class, false otherwise.

Exceptions

std::runtime_error	if any of the following conditions are met: • All classes in the UC have reached maximum occupancy. • Adding the student would disturb the balance of class occupancy in this UC. • There are no available classes in this UC. • Enrolling in a class would conflict with the student's existing schedule.

Definition at line 369 of file Request.cpp.

```
00370 {
00371     Script script;
00372     Uc destination = Uc(ucDestination);
00373     script.loadClasses(destination);
00374     int max = 0;
00375     int min = 100;
00376
00377     for (string currClass : destination.getClasses())
00378     {
00379         int classSize = script.studentsinClass(destination.getUcCode(), currClass).size();
00380         if (classSize + 1 > max)
00381         {
00382             max = classSize + 1;
00383         }
00384         else if (classSize + 1 < min)
00385         {
00386             min = classSize + 1;
00387         }
00388
00389         if (classSize + 1 <= MAXIMO && (max - classSize - 1) <= 4)
00390         {
00391             eligibleClasses.push(currClass);
00392         }
00393     }
00394
00395     if (max > MAXIMO)
00396     {
00397         throw runtime_error("All classes with maximum occupancy");
00398         return this->flag;
00399     }
00400
00401     if ((max - min) > 4)
00402     {
00403         throw runtime_error("Adding the student would affect the balance of classes in this UC");
00404         return this->flag;
00405     }
00406     if (eligibleClasses.size() < 1)
00407     {
00408         throw runtime_error("This UC hasn't avaiable classes");
00409         return this->flag;
00410     }
00411
00412     bool check = false;
00413     for (Lecture currentLecture : script.loadLecture(ucDestination, eligibleClasses.front()))
00414     {
00415         for (Lecture studentLecture : script.getSchedule(studentCode))
00416         {
00417             if (studentLecture.overlay(currentLecture))
00418             {
00419                 eligibleClasses.pop();
00420                 check = true;
00421                 break;
00422             }
00423         }
00424         if (eligibleClasses.empty())
00425         {
00426             throw runtime_error("This UC will disturb the student's schedule");
00427             return this->flag;
00428         }
00429         if (check)
00430             continue;
00431     }
00432     return true;
00433 }
```

4.3.3.4 removeUc()

```
bool Request::removeUc (
            std::string ucCode )
```

Removes a student from a specified course.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

ucCode	The code of the course to remove the student from.

Returns

True if the student is successfully removed, false otherwise.

Definition at line 20 of file Request.cpp.

```
00021 {
00022     ifstream read_file("../data/students_classes.csv");
00023     string line;
00024     queue<string> lines;
00025     while (getline(read_file, line))
00026     {
00027         istringstream iss(line);
00028         string StudentCode, StudentName, UcCode, classCode;
00029 
00030         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00031 
00032         if (StudentCode == studentCode && UcCode == ucCode)
00033         {
00034             this->flag = true;
00035             continue;
00036         }
00037 
00038         lines.push(line);
00039     }
00040     read_file.close();
00041 
00042     if (this->flag)
00043     {
00044         ofstream write_log("../requests_log.csv", ios::app);
00045         write_log « id « ',' « type « ',' « studentCode « ',' « ucCode « endl;
00046         write_log.close();
00047     }
00048     else
00049     {
00050         throw runtime_error("You are not enrolled at this Uc.");
00051         return this->flag;
00052     }
00053 
00054     size_t count = lines.size();
00055     ofstream write_file("../data/students_classes.csv");
00056     for (int i = 0; i < count; i++)
00057     {
00058         write_file « lines.front() « endl;
00059         lines.pop();
00060     }
00061     write_file.close();
00062 
00063     return this->flag;
00064 }
```

4.3.3.5 studentRequests()

```
void Request::studentRequests (
            const std::string & studentCode )
```

Displays the list of requests for a specific student.
Time complexity: O(N), where N is the number of lines in the requests_log.csv file.

Parameters

studentCode	The student's unique code.

Definition at line 180 of file Request.cpp.

```
00181 {
00182     ifstream read_file("../requests_log.csv");
00183     string line;
00184     while (getline(read_file, line))
00185     {
00186         istringstream iss(line);
00187         string id_, type_, studentCode_;
00188 
00189         getline(getline(getline(iss, id_, ','), type_, ','), studentCode_, ',');
00190 
00191         if (studentCode_ == studentCode)
00192         {
00193             if (type_ == "1")
00194             {
00195                 string ucCode_, classCode_;
00196                 getline(getline(iss, ucCode_, ','), classCode_, '\r');
00197                 cout << "Operation ID: " << id_ << " | Student added the UC " << ucCode_ << " and entered
      the class " << classCode_ << endl;
00198             }
00199             else if (type_ == "2")
00200             {
00201                 string ucCode_;
00202                 getline(iss, ucCode_, '\r');
00203                 cout << "Operation ID: " << id_ << " | Student removed the UC " << ucCode_ << endl;
00204             }
00205             else if (type_ == "3")
00206             {
00207                 string ucOrigin_, ucDestination_, classCode_;
00208                 getline(getline(getline(iss, ucOrigin_, ','), ucDestination_, ','), classCode_, '\r');
00209                 cout << "Operation ID: " << id_ << " |  Student switched from UC " << ucOrigin_ << " to the
      UC " << ucDestination_ << " and was added to the class " << classCode_ << endl;
00210             }
00211             else if (type_ == "4")
00212             {
00213                 string ucOrigin_, classOrigin_, classDestination_;
00214                 getline(getline(getline(iss, ucOrigin_, ','), classOrigin_, ','), classDestination_,
      '\r');
00215                 cout << "Operation ID: " << id_ << " |  Student switched from class " << classOrigin_ <<
      "Of the UC " << ucOrigin_ << " to the class " << classDestination_ << endl;
00216             }
00217         }
00218     }
00219     read_file.close();
00220 }
```

4.3.3.6 switchClass()

```
bool Request::switchClass (
            std::string uc,
            std::string classOrigin,
            std::string classDestination )
```

Switches a student from one class to another within the same course.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

uc	The course code.
classOrigin	The original class code.
classDestination	The destination class code.

Returns

True if the student's class is successfully switched, false otherwise.

Definition at line 221 of file Request.cpp.

```
00222 {
00223 
00224     Script script;
00225     Student newStudent = script.loadStudent(this->studentCode);
00226     int max = 0;
00227     int min = 100;
00228     queue<string> eligibleClasses = {};
00229     if(!classesCheck(currentUc, eligibleClasses)) return this->flag;
00230 
00231     while(!eligibleClasses.empty()){
00232         if(eligibleClasses.front()==classDestination){
00233             this->flag = true;
```

```
00234             break;
00235         }
00236         eligibleClasses.pop();
00237     }
00238     if (!(this->flag))
00239     {
00240         throw runtime_error("The selected UC is unavaiable");
00241         return this->flag;
00242     }
00243 
00244     ifstream read_file("../data/students_classes.csv");
00245     string line;
00246     queue<string> lines;
00247     while (getline(read_file, line))
00248     {
00249         istringstream iss(line);
00250         string StudentCode, StudentName, UcCode, classCode;
00251         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
    classCode, '\r');
00252 
00253         if (StudentCode == this->studentCode && UcCode == currentUc && classCode == classOrigin)
00254         {
00255             this->flag = true;
00256             continue;
00257         }
00258         lines.push(line);
00259     }
00260     read_file.close();
00261 
00262     size_t count = lines.size();
00263     ofstream write_file("../data/students_classes.csv");
00264     for (int i = 0; i < count; i++)
00265     {
00266         write_file << lines.front() << endl;
00267         lines.pop();
00268     }
00269     write_file << this->studentCode << ',' << newStudent.getstudentName() << ',' << currentUc << ',' <<
    classDestination << '\r';
00270     write_file.close();
00271 
00272     ofstream write_log("../requests_log.csv", ios::app);
00273     write_log << id << ',' << type << ',' << studentCode << ',' << currentUc << ',' << classOrigin << ',' <<
    classDestination << endl;
00274     write_log.close();
00275 
00276     return this->flag;
00277 }
```

4.3.3.7 switchUc()

```
bool Request::switchUc (
            std::string ucOrigin,
            std::string ucDestination )
```

Switches a student from one course to another, preserving their schedule.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

ucOrigin	The original course code.
ucDestination	The destination course code.

Returns

True if the student's course is successfully switched, false otherwise.

Definition at line 108 of file Request.cpp.

```
00109 {
00110     Script script;
00111     Student newStudent = script.loadStudent(this->studentCode);
00112 
00113     map<std::string, std::string> new_schedule = newStudent.getSchedule();
00114 
00115     if (new_schedule.find(ucOrigin) == new_schedule.end())
00116     {
00117         throw runtime_error("You are not enrolled in this UC");
00118         return this->flag;
00119     }
00120     if (new_schedule.find(ucDestination) != new_schedule.end())
```

```
00121     {
00122         throw runtime_error("Student already registered in this UC");
00123         return this->flag;
00124     }
00125 
00126     int max = 0;
00127     int min = 100;
00128     queue<string> eligibleClasses = {};
00129     if(!classesCheck(ucDestination, eligibleClasses)) return this->flag;
00130 
00131     ifstream read_file("../data/students_classes.csv");
00132     string line;
00133     queue<string> lines;
00134 
00135     while (getline(read_file, line))
00136     {
00137         istringstream iss(line);
00138         string StudentCode, StudentName, UcCode, classCode;
00139 
00140         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00141 
00142         if (StudentCode == studentCode && UcCode == ucOrigin)
00143         {
00144             this->flag = true;
00145             continue;
00146         }
00147 
00148         lines.push(line);
00149     }
00150     read_file.close();
00151 
00152     size_t count = lines.size();
00153     ofstream write_file("../data/students_classes.csv");
00154     for (int i = 0; i < count; i++)
00155     {
00156         write_file « lines.front() « endl;
00157         lines.pop();
00158     }
00159     write_file.close();
00160 
00161     ofstream outFile("../data/students_classes.csv", ios::app);
00162 
00163     if (!outFile.is_open())
00164     {
00165         cerr « "Couldnt open file." « endl;
00166         return this->flag;
00167     }
00168     outFile « newStudent.getstudentCode() « ',' « newStudent.getstudentName() « ',' « ucDestination «
      ',' « eligibleClasses.front() « endl;
00169 
00170     outFile.close();
00171 
00172     ofstream write_log("../requests_log.csv", ios::app);
00173     write_log « id « ',' « type « ',' « studentCode « ',' « ucOrigin « ',' « ucDestination « ',' «
      eligibleClasses.front() « endl;
00174     write_log.close();
00175 
00176     this->flag = true;
00177     return this->flag;
00178 }
```

4.3.3.8 undoRequest()

```
void Request::undoRequest (
            unsigned id )
```

Undoes a specific request by its ID.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

id	The ID of the request to undo.

Definition at line 279 of file Request.cpp.

```
00280 {
00281     ifstream read_file("../requests_log.csv");
00282     string line;
00283     while (getline(read_file, line))
00284     {
00285         istringstream iss(line);
00286         string idFromFile, typeFromFile, studentCodeFromFile;
```

```
00287         getline(getline(getline(iss, idFromFile, ','), typeFromFile, ','), studentCodeFromFile, ',');
00288         if (idFromFile == to_string(id))
00289         {
00290             if (typeFromFile == "1")
00291             {
00292                 string ucCodeFromFile, classCodeFromFile;
00293                 getline(getline(iss, ucCodeFromFile, ','), classCodeFromFile, '\r');
00294 
00295                 Request(studentCodeFromFile, '2').removeUc(ucCodeFromFile);
00296                 break;
00297             }
00298             else if (typeFromFile == "2")
00299             {
00300                 string ucCodeFromFile;
00301                 getline(iss, ucCodeFromFile, '\r');
00302 
00303                 Request(studentCodeFromFile, '1').addUc(ucCodeFromFile);
00304                 break;
00305             }
00306             else if (typeFromFile == "3")
00307             {
00308                 string originFromFile, destinationFromFile, classCodeFromFile;
00309                 getline(getline(getline(iss, originFromFile, ','), destinationFromFile, ','),
      classCodeFromFile, '\r');
00310 
00311                 Request(studentCodeFromFile, '3').switchUc(destinationFromFile, originFromFile);
00312                 break;
00313             }
00314             else if (typeFromFile == "4")
00315             {
00316                 string ucCodeFromFile, originFromFile, destinationFromFile;
00317                 getline(getline(getline(iss, ucCodeFromFile, ','), originFromFile, ','),
      destinationFromFile, '\r');
00318 
00319                 Request(studentCodeFromFile, '4').switchClass(ucCodeFromFile, destinationFromFile,
      originFromFile);
00320                 break;
00321             }
00322         }
00323     }
00324 
00325     if (read_file.eof())
00326         throw runtime_error("This request does not exist.");
00327 }
```

4.3.4 Member Data Documentation

4.3.4.1 flag

`bool Request::flag = false  [private]`

A flag indicating the success status of the request.

Definition at line 110 of file Request.hpp.

4.3.4.2 id

`unsigned Request::id  [private]`

Unique ID of the request.

Definition at line 101 of file Request.hpp.

4.3.4.3 studentCode

`std::string Request::studentCode  [private]`

Student's unique code.

Definition at line 104 of file Request.hpp.

4.3.4.4 type

`char Request::type  [private]`

Type of request (1: Add Uc, 2: Remove Uc, 3: Switch Uc, 4: Switch Class).

Definition at line 107 of file Request.hpp.

The documentation for this class was generated from the following files:

- inc/Request.hpp
- src/Request.cpp

4.4 Script Class Reference

Reads and processes data from the CSV files.
`#include <Script.hpp>`

Public Member Functions

- Student loadStudent (const std::string &studentCode)

  Loads a student based on the student code.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- std::list< Lecture > loadLecture (std::string ucCode_, std::string classCode_)

  Loads a list of lectures for a specific UC and class code.
  Time complexity: O(N), where N is the number of lines in the classes.csv file.
- void loadClasses (Uc &uc_)

  Loads classes into a Uc object.
  Time complexity: O(N), where N is the number of lines in the classes_per_uc.csv file.
- void studentsInLecture (Lecture &oneLecture_)

  Populates a lecture with students who are enrolled in it.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- std::set< Lecture > getSchedule (const std::string &studentCode)

  Gets the schedule of lectures for a student based on their student code.
  Time complexity: O(N), where N is the number of lines in the classes.csv file.
- std::vector< Student > studentsinUc (Uc &uc)

  Retrieves a list of students enrolled in a specific UC.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- std::vector< Student > studentsinClass (std::string ucCode_, std::string classCode_)

  Retrieves a list of students enrolled in a specific class.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- std::unordered_set< Student, Student::Hash > studentsInYear (const std::string &year)

  Retrieves a set of students based on their enrollment year.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- int studentsInNUc (int number)

  Counts the number of students enrolled in at least 'number' UCs.
  Time complexity: O(N), where N is the number of lines in the students_classes.csv file.
- std::vector< std::pair< std::string, int > > ucsWithMostStudents ()

  Retrieves a list of UCs with the most students, along with the number of students in each UC.
  Time complexity: O(N log N), where N is the number of lines in the students_classes.csv file.

4.4.1 Detailed Description

Reads and processes data from the CSV files.
Definition at line 16 of file Script.hpp.

4.4.2 Member Function Documentation

4.4.2.1 getSchedule()

```
set< Lecture > Script::getSchedule (
            const std::string & studentCode )
```

Gets the schedule of lectures for a student based on their student code.
Time complexity: O(N), where N is the number of lines in the classes.csv file.

Parameters

*studentCode_*	The student code.

Returns

A set of Lecture objects representing the student's schedule.

Definition at line 138 of file Script.cpp.

```
00139 {
00140     Script script;
00141     Student oneStudent_ = script.loadStudent(studentCode_);
00142     set<Lecture> result = {};
00143 
00144     ifstream file("../data/classes.csv");
00145     if (!file.is_open())
00146     {
00147         cout « "Failed to open the file." « endl;
00148         return result;
00149     }
00150 
00151     string line;
00152 
00153     while (getline(file, line))
00154     {
00155         istringstream iss(line);
00156         string ClassCode, UcCode, Weekday, strStarHour, strDuration, Type;
00157         double StartHour, Duration;
00158 
00159         getline(getline(getline(getline(getline(getline(iss, ClassCode, ','), UcCode, ','), Weekday,
      ','), strStarHour, ','), strDuration, ','), Type, '\r');
00160 
00161         try
00162         {
00163             StartHour = stod(strStarHour);
00164             Duration = stod(strDuration);
00165         }
00166         catch (const std::invalid_argument &e)
00167         {
00168         }
00169         catch (const std::out_of_range &e)
00170         {
00171             std::cerr « "Erro: Conversão fora do alcance. O número é muito grande ou muito pequeno." «
     std::endl;
00172         }
00173 
00174         if (oneStudent_.inClass(UcCode, ClassCode))
00175         {
00176 
00177             Lecture lecture(UcCode, ClassCode, Weekday, StartHour, Duration, Type);
00178             result.insert(lecture);
00179         }
00180     }
00181 
00182     file.close();
00183 
00184     return result;
00185 }
```

4.4.2.2 loadClasses()

```
void Script::loadClasses (
            Uc & uc_ )
```

Loads classes into a Uc object.
Time complexity: O(N), where N is the number of lines in the classes_per_uc.csv file.

Parameters

*uc_*	The Uc object to load classes into.

Definition at line 82 of file Script.cpp.

```
00083 {
00084     ifstream file;
00085     file.open("../data/classes_per_uc.csv", std::ios::in);
00086 
00087     if (!file.is_open())
00088         cout « "not open";
00089     string line;
00090 
00091     while (getline(file, line))
00092     {
00093         istringstream stream(line);
00094         string Code, ClassCode;
```

```
00095
00096         if (getline(stream, Code, ','))
00097         {
00098             if (Code == uc_.getUcCode())
00099             {
00100                 if (getline(stream, ClassCode, '\r'))
00101                 {
00102                     uc_.addClass(ClassCode);
00103                 }
00104             }
00105         }
00106     }
00107     file.close();
00108 }
```

4.4.2.3 loadLecture()

```
list< Lecture > Script::loadLecture (
            std::string ucCode_,
            std::string classCode_ )
```

Loads a list of lectures for a specific UC and class code.
Time complexity: O(N), where N is the number of lines in the classes.csv file.

Parameters

*ucCode_*	The UC code.
*classCode_*	The class code.

Returns

A list of loaded Lecture objects.

Definition at line 36 of file Script.cpp.

```
00037 {
00038     list<Lecture> result = {};
00039     ifstream file("../data/classes.csv");
00040     if (!file.is_open())
00041     {
00042         cout « "Failed to open the file." « endl;
00043         return result;
00044     }
00045
00046     string line;
00047
00048     while (getline(file, line))
00049     {
00050         istringstream iss(line);
00051         string ClassCode, UcCode, Weekday, strStarHour, strDuration, Type;
00052         double StartHour, Duration;
00053
00054         getline(getline(getline(getline(getline(getline(iss, ClassCode, ','), UcCode, ','), Weekday,
      ','), strStarHour, ','), strDuration, ','), Type, '\r');
00055
00056         try
00057         {
00058             StartHour = stod(strStarHour);
00059             Duration = stod(strDuration);
00060         }
00061         catch (const std::invalid_argument &e)
00062         {
00063         }
00064         catch (const std::out_of_range &e)
00065         {
00066             std::cerr « "Erro: Conversão fora do alcance. O número é muito grande ou muito pequeno." «
      std::endl;
00067         }
00068
00069         if (ucCode_ == UcCode && classCode_ == ClassCode)
00070         {
00071
00072             Lecture lecture(UcCode, ClassCode, Weekday, StartHour, Duration, Type);
00073             result.push_back(lecture);
00074         }
00075     }
00076
00077     file.close();
00078
```

```
00079     return result;
00080 }
```

4.4.2.4 loadStudent()

```
Student Script::loadStudent (
            const std::string & studentCode )
```

Loads a student based on the student code.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

studentCode	The student code to load.

Returns

The loaded Student object.

Definition at line 4 of file Script.cpp.

```
00005 {
00006     Student student;
00007
00008     ifstream file("../data/students_classes.csv");
00009
00010     if (!file.is_open())
00011     {
00012         return student;
00013     }
00014
00015     string line;
00016     getline(file, line);
00017     while (getline(file, line))
00018     {
00019         istringstream iss(line);
00020         string studentCodeFromFile, studentNameFromFile, ucCodeFromFile, classCodefromFile;
00021         getline(getline(getline(getline(iss, studentCodeFromFile, ','), studentNameFromFile, ','),
      ucCodeFromFile, ','), classCodefromFile, '\r');
00022
00023         if (studentCodeFromFile == studentCode)
00024         {
00025             student.setstudentCode(studentCodeFromFile);
00026             student.setstudentName(studentNameFromFile);
00027             student.addClass(pair<string, string>{ucCodeFromFile, classCodefromFile});
00028         }
00029     }
00030
00031     file.close();
00032
00033     return student;
00034 }
```

4.4.2.5 studentsinClass()

```
vector< Student > Script::studentsinClass (
            std::string ucCode_,
            std::string classCode_ )
```

Retrieves a list of students enrolled in a specific class.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

*ucCode_*	The UC code.
*classCode_*	The class code.

Returns

A vector of Student objects.

Definition at line 217 of file Script.cpp.

```
00218 {
00219     vector<Student> students;
00220
00221     ifstream file("../data/students_classes.csv");
00222     if (!file.is_open())
00223     {
00224         cout « "Failed to open the file." « endl;
00225     }
00226
00227     string line;
00228
00229     while (getline(file, line))
00230     {
00231         istringstream iss(line);
00232         string StudentCode, StudentName, UcCode, classCode;
00233
00234         getline(getline(getline(getline(iss, StudentCode, ',', StudentName, ',', UcCode, ',',
      classCode, '\r');
00235
00236         if (UcCode == ucCode_ && classCode == classCode_)
00237         {
00238             Student student{StudentCode, StudentName};
00239             students.push_back(student);
00240         }
00241     }
00242
00243     file.close();
00244     return students;
00245 }
```

4.4.2.6 studentsInLecture()

```
void Script::studentsInLecture (
            Lecture & oneLecture_ )
```

Populates a lecture with students who are enrolled in it.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

*oneLecture_*	The lecture to populate with students.

Definition at line 110 of file Script.cpp.

```
00111 {
00112
00113     ifstream file("../data/students_classes.csv");
00114     if (!file.is_open())
00115     {
00116         cout « "Failed to open the file." « endl;
00117     }
00118
00119     string line;
00120
00121     while (getline(file, line))
00122     {
00123         istringstream iss(line);
00124         string StudentCode, StudentName, UcCode, classCode;
00125
00126         getline(getline(getline(getline(iss, StudentCode, ',', StudentName, ',', UcCode, ',',
      classCode, '\r');
00127
00128         if ((UcCode == oneLecture_.getUc().getUcCode()) && (classCode == oneLecture_.getClassCode()))
00129         {
00130             Student student(StudentCode, StudentName);
00131             oneLecture_.addStudent(student);
00132         }
00133     }
00134
00135     file.close();
00136 }
```

4.4.2.7 studentsInNUc()

```
int Script::studentsInNUc (
            int number )
```

Counts the number of students enrolled in at least 'number' UCs.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

number	The minimum number of UCs for a student to be counted.

Returns

The count of students meeting the criteria.

Definition at line 275 of file Script.cpp.

```
00276 {
00277     int result = 0;
00278     int aux = 0;
00279     ifstream file("../data/students_classes.csv");
00280     if (!file.is_open())
00281     {
00282         cout « "Failed to open the file." « endl;
00283     }
00284
00285     unordered_map<string, unordered_map<string, bool» studentUCs;
00286     string line;
00287
00288     while (std::getline(file, line))
00289     {
00290         istringstream iss(line);
00291         string studentCode, studentName, ucCode, classCode;
00292         getline(getline(getline(getline(iss, studentCode, ','), studentName, ','), ucCode, ','),
      classCode, '\r');
00293
00294         studentUCs[studentCode][ucCode] = true;
00295     }
00296
00297     int count = 0;
00298
00299     for (const auto &student : studentUCs)
00300     {
00301         if (student.second.size() >= number)
00302         {
00303             count++;
00304         }
00305     }
00306
00307     return count;
00308 }
```

4.4.2.8 studentsinUc()

```
vector< Student > Script::studentsinUc (
            Uc & uc )
```

Retrieves a list of students enrolled in a specific UC.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

uc	The Uc object representing the UC.

Returns

A vector of Student objects.

Definition at line 187 of file Script.cpp.

```
00188 {
00189     vector<Student> students;
00190
00191     ifstream file("../data/students_classes.csv");
00192     if (!file.is_open())
```

```
00193     {
00194         cout « "Failed to open the file." « endl;
00195     }
00196
00197     string line;
00198
00199     while (getline(file, line))
00200     {
00201         istringstream iss(line);
00202         string StudentCode, StudentName, UcCode, classCode;
00203
00204         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00205
00206         if (UcCode == uc.getUcCode())
00207         {
00208             Student student{StudentCode, StudentName};
00209             students.push_back(student);
00210         }
00211     }
00212
00213     file.close();
00214     return students;
00215 }
```

4.4.2.9 studentsInYear()

```
unordered_set< Student, Student::Hash > Script::studentsInYear (
            const std::string & year )
```

Retrieves a set of students based on their enrollment year.
Time complexity: O(N), where N is the number of lines in the students_classes.csv file.

Parameters

year	The year for which to retrieve students.

Returns

An unordered set of Student objects.

Definition at line 247 of file Script.cpp.

```
00248 {
00249     unordered_set<Student, Student::Hash> students;
00250     ifstream file("../data/students_classes.csv");
00251     if (!file.is_open())
00252     {
00253         cout « "Failed to open the file." « endl;
00254     }
00255
00256     string line;
00257     while (getline(file, line))
00258     {
00259         istringstream iss(line);
00260
00261         if (line.substr(0, 4) == year)
00262         {
00263             string StudentCode, StudentName, UcCode, classCode;
00264             getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00265             Student student{StudentCode, StudentName};
00266
00267             students.insert(student);
00268         }
00269     }
00270
00271     file.close();
00272     return students;
00273 }
```

4.4.2.10 ucsWithMostStudents()

```
vector< pair< string, int > > Script::ucsWithMostStudents ( )
```

Retrieves a list of UCs with the most students, along with the number of students in each UC.
Time complexity: O(N log N), where N is the number of lines in the students_classes.csv file.

Returns

A vector of pairs, where the first element is the UC code, and the second element is the number of students.

Definition at line 310 of file Script.cpp.

```
00311 {
00312     map<string, int> aux = {};
00313 
00314     ifstream file("../data/students_classes.csv");
00315     if (!file.is_open())
00316     {
00317         cout « "Failed to open the file." « endl;
00318     }
00319 
00320     string line;
00321     getline(file, line);
00322     while (getline(file, line))
00323     {
00324         istringstream iss(line);
00325         string studentCode, studentName, ucCode, classCode;
00326         getline(getline(getline(getline(iss, studentCode, ','), studentName, ','), ucCode, ','),
      classCode, '\r');
00327 
00328         aux[ucCode]++;
00329     }
00330     file.close();
00331 
00332     vector<pair<string, int» result = {};
00333 
00334     for (const pair<string, int> &p : aux)
00335         result.push_back(p);
00336 
00337     sort(result.begin(), result.end(), [](pair<string, int> p1, pair<string, int> p2) -> bool
00338          { return p1.second > p2.second; });
00339 
00340     return result;
00341 }
```

The documentation for this class was generated from the following files:

- inc/Script.hpp
- src/Script.cpp

4.5 Student Class Reference

Represents a student.

`#include <Student.hpp>`

Classes

- struct Hash

  Hash function for Student objects.

Public Member Functions

- Student ()

  Default constructor for the Student class.
- Student (const Student &student_)

  Copy constructor for the Student class.
- Student (const std::string &studentCode, const std::string &studentName)

  Constructor for the Student class with a student code and name.
- std::string getstudentCode ()

  Get the student code associated with this student.
- void setstudentCode (const std::string &studentCode)

  Set the student code for this student.
- std::string getstudentName ()

  Get the name of the student.
- std::map< std::string, std::string > getSchedule ()

  Get the schedule of the student as a map of UcCode to classCode.

- void setstudentName (const std::string &studentName)

  Set the name of the student.
- void addClass (const std::pair< std::string, std::string > &Class)

  Add a class to the student's schedule.
- bool inClass (const std::string &ucCode, const std::string &classCode)

  Check if the student is enrolled in a specific class.
- bool operator== (const Student &other) const

  Overloaded equality operator to compare two students for equality.

Private Attributes

- std::string studentCode
- std::string studentName
- std::map< std::string, std::string > schedule

4.5.1 Detailed Description

Represents a student.
Definition at line 11 of file Student.hpp.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Student() [1/3]

```
Student::Student ( )
```

Default constructor for the Student class.
Definition at line 4 of file Student.cpp.

```
00005 {
00006     this->studentName = "NO_NAME";
00007     this->studentCode = "NO_CODE";
00008     this->schedule = {};
00009 }
```

4.5.2.2 Student() [2/3]

```
Student::Student (
            const Student & student_ )
```

Copy constructor for the Student class.

Parameters

*student_*	The student to copy.

Definition at line 11 of file Student.cpp.

```
00012 {
00013     this->studentName = student_.studentName;
00014     this->studentCode = student_.studentCode;
00015     this->schedule = student_.schedule;
00016 }
```

4.5.2.3 Student() [3/3]

```
Student::Student (
            const std::string & studentCode,
            const std::string & studentName )
```

Constructor for the Student class with a student code and name.

Parameters

studentCode	The code associated with the student.
studentName	The name of the student.

4.5.3 Member Function Documentation

4.5.3.1 addClass()

```
void Student::addClass (
            const std::pair< std::string, std::string > & Class )
```

Add a class to the student's schedule.

Parameters

Class	A pair of UcCode and classCode to add to the schedule.

Definition at line 40 of file Student.cpp.

```
00041 {
00042
00043     schedule.insert(Class);
00044 }
```

4.5.3.2 getSchedule()

```
map< string, string > Student::getSchedule ( )
```

Get the schedule of the student as a map of UcCode to classCode.

Returns

A map representing the student's schedule.

Definition at line 46 of file Student.cpp.

```
00046                                          {
00047     return this->schedule;
00048 }
```

4.5.3.3 getstudentCode()

```
string Student::getstudentCode ( )
```

Get the student code associated with this student.

Returns

The student code as a string.

Definition at line 24 of file Student.cpp.

```
00025 {
00026     return this->studentCode;
00027 }
```

4.5.3.4 getstudentName()

```
string Student::getstudentName ( )
```

Get the name of the student.

Returns

The student name as a string.

Definition at line 32 of file Student.cpp.

```
00033 {
00034     return this->studentName;
00035 }
```

4.5.3.5 inClass()

```
bool Student::inClass (
            const std::string & ucCode,
            const std::string & classCode )
```

Check if the student is enrolled in a specific class.

Parameters

ucCode	The UcCode to check.
classCode	The classCode to check.

Returns

True if the student is enrolled in the class, otherwise false.

Definition at line 50 of file Student.cpp.

```
00051 {
00052     auto it = schedule.find(ucCode_);
00053     if (it != schedule.end())
00054     {
00055 
00056         if (it->second == classCode_)
00057             return true;
00058         else
00059             return false;
00060     }
00061     else
00062     {
00063         return false;
00064     }
00065 }
```

4.5.3.6 operator==()

```
bool Student::operator== (
            const Student & other ) const
```

Overloaded equality operator to compare two students for equality.

Parameters

other	The student to compare with.

Returns

True if the students are equal, otherwise false.

Definition at line 67 of file Student.cpp.

```
00068 {
00069     return this->studentCode == other.studentCode && this->studentName == other.studentName;
00070 }
```

4.5.3.7 setstudentCode()

```
void Student::setstudentCode (
            const std::string & studentCode )
```

Set the student code for this student.

Parameters

studentCode	The student code to set.

Definition at line 28 of file Student.cpp.

```
00029 {
00030     this->studentCode = studentCode;
00031 }
```

4.5.3.8 setstudentName()

```
void Student::setstudentName (
            const std::string & studentName )
```

Set the name of the student.

Parameters

studentName	The name to set for the student.

Definition at line 36 of file Student.cpp.

```
00037 {
00038     this->studentName = studentName;
00039 }
```

4.5.4 Member Data Documentation

4.5.4.1 schedule

```
std::map<std::string, std::string> Student::schedule  [private]
```

The student's schedule, mapping UcCode to classCode.
Definition at line 103 of file Student.hpp.

4.5.4.2 studentCode

```
std::string Student::studentCode  [private]
```

The code associated with the student.
Definition at line 97 of file Student.hpp.

4.5.4.3 studentName

```
std::string Student::studentName  [private]
```

The name of the student.
Definition at line 100 of file Student.hpp.
The documentation for this class was generated from the following files:

- inc/Student.hpp
- src/Student.cpp

4.6 Uc Class Reference

Represents a Uc.

```
#include <Uc.hpp>
```

Public Member Functions

- Uc ()

  Default constructor for the Uc class.
- Uc (const std::string &UcCode)

  Constructor for the Uc class with a given UcCode.
- std::string getUcCode ()

  Get the UcCode associated with this Uc.
- void setUcCode (const std::string &UcCode)

  Set the UcCode for this Uc.
- void addClass (const std::string &UcClass)

  Add a class to the Uc.
- void printClasses (const std::string &SortMethod)

  Print the list of classes in this Uc, sorted by SortMethod.
- unsigned int classesCount ()

  Get the number of classes in this Uc.
- std::vector< std::string > getClasses ()

  Get a vector of class codes associated with this Uc.

Private Attributes

- std::string UcCode
- std::vector< std::string > UcClasses

4.6.1 Detailed Description

Represents a Uc.
Definition at line 14 of file Uc.hpp.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Uc() [1/2]

```
Uc::Uc ( )
```

Default constructor for the Uc class.
Definition at line 5 of file Uc.cpp.

```
00006 {
00007     UcCode = "NO_NAME";
00008     UcClasses = vector<string>();
00009 }
```

4.6.2.2 Uc() [2/2]

```
Uc::Uc (
            const std::string & UcCode )
```

Constructor for the Uc class with a given UcCode.

Parameters

UcCode	The code associated with this Uc.

4.6.3 Member Function Documentation

4.6.3.1 addClass()

```
void Uc::addClass (
            const std::string & UcClass )
```

Add a class to the Uc.

Parameters

UcClass	The name of the class to add.

Definition at line 25 of file Uc.cpp.

```
00026 {
00027     for (vector<string>::iterator it = UcClasses.begin(); it != UcClasses.end(); it++)
00028         if (*it == UcClass)
00029             return;
00030     UcClasses.push_back(UcClass);
00031     sort(UcClasses.begin(), UcClasses.end());
00032 }
```

4.6.3.2 classesCount()

```
unsigned int Uc::classesCount ( )
```

Get the number of classes in this Uc.

Returns

The number of classes as an unsigned integer.

Definition at line 57 of file Uc.cpp.

```
00058 {
```

```
00059     return UcClasses.size();
00060 }
```

4.6.3.3 getClasses()

```
vector< string > Uc::getClasses ( )
```

Get a vector of class codes associated with this Uc.

Returns

A vector of strings containing class codes.

Definition at line 53 of file Uc.cpp.

```
00053                                {
00054     return this->UcClasses;
00055 }
```

4.6.3.4 getUcCode()

```
string Uc::getUcCode ( )
```

Get the UcCode associated with this Uc.

Returns

The UcCode as a string.

Definition at line 15 of file Uc.cpp.

```
00016 {
00017     return UcCode;
00018 }
```

4.6.3.5 printClasses()

```
void Uc::printClasses (
            const std::string & SortMethod )
```

Print the list of classes in this Uc, sorted by SortMethod.

Parameters

SortMethod	The method for sorting classes ("1" for ascending and "2" for descending).

Definition at line 34 of file Uc.cpp.

```
00035 {
00036     if (SortMethod == "1") {
00037         for (const string &turma : UcClasses){
00038             cout « "|" « turma « "|" « endl;
00039         }
00040     } else if (SortMethod == "2") {
00041         stack<string> reverse;
00042         for(const string &turma : UcClasses) reverse.push(turma);
00043         while(!reverse.empty()){
00044
00045             cout « "|" «reverse.top() « endl;
00046             reverse.pop();
00047         }
00048     } else {
00049         cout « "Selecione um método de ordenação válido" « endl;
00050     }
00051 }
```

4.6.3.6 setUcCode()

```
void Uc::setUcCode (
            const std::string & UcCode )
```

Set the UcCode for this Uc.

Parameters

UcCode	The UcCode to set.

Definition at line 20 of file Uc.cpp.

```
00021 {
00022     this->UcCode = UcCode;
00023 }
```

4.6.4 Member Data Documentation

4.6.4.1 UcClasses

```
std::vector<std::string> Uc::UcClasses  [private]
```

Definition at line 67 of file Uc.hpp.

4.6.4.2 UcCode

```
std::string Uc::UcCode  [private]
```

Definition at line 64 of file Uc.hpp.

The documentation for this class was generated from the following files:

- inc/Uc.hpp
- src/Uc.cpp

Chapter 5

File Documentation

5.1 Lecture.hpp

```
00001 #ifndef AED_PROJECT_LECTURE_H
00002 #define AED_PROJECT_LECTURE_H
00003
00004 #include "Student.hpp"
00005
00010 class Lecture
00011 {
00012 public:
00017     Lecture(const std::string &ucCode);
00018
00028     Lecture(const std::string &ucCode, const std::string &classCode, const std::string &weekDay,
00029             const double &startHour, const double &duration, const std::string &type);
00030
00035     Uc getUc();
00036
00041     std::string getClassCode();
00042
00047     void setUc(const Uc &uc);
00048
00053     void addStudent(Student &student);
00054
00059     void removeStudent(const Student &student);
00060
00065     std::vector<Student> getStudents();
00066
00071     std::string getWeekDay() const;
00072
00077     void setWeekDay(const std::string &weekDay);
00078
00083     double getStartHour() const;
00084
00089     void setStartHour(const double &startHour);
00090
00095     double getDuration() const;
00096
00101     void setDuration(const double &duration);
00102
00107     std::string getType() const;
00108
00113     void setType(const std::string &type);
00114
00120     bool operator==(Lecture &other);
00121
00127     bool operator<(const Lecture &other) const;
00128
00134     bool overlay(Lecture &other);
00135
00136 private:
00138     Uc uc;
00139
00141     std::string classCode;
00142
00144     std::vector<Student> students;
00145
00147     std::string weekDay;
00148
00150     double startHour;
00151
00153     double duration;
00154
00156     std::string type;
00157 };
```

```
00158
00159 #endif
```

5.2 Request.hpp

```
00001 #ifndef AED_PROJECT_REQUEST_H
00002 #define AED_PROJECT_REQUEST_H
00003
00004 #include "../inc/Script.hpp"
00005 #include <queue>
00006 #include <list>
00007
00014 class Request
00015 {
00016 public:
00018     Request(){};
00019
00025     Request(std::string studentCode, char type);
00026
00033     bool addUc(std::string ucCodeDestination);
00034
00041     bool removeUc(std::string ucCode);
00042
00050     bool switchUc(std::string ucOrigin, std::string ucDestination);
00051
00060     bool switchClass(std::string uc, std::string classOrigin, std::string classDestination);
00061
00067     void studentRequests(const std::string &studentCode);
00068
00072     void adminRequests();
00073
00079     void undoRequest(unsigned id);
00080
00097     bool classesCheck(std::string uc, std::queue<std::string> &eligibleClasses);
00098
00099 private:
00101     unsigned id;
00102
00104     std::string studentCode;
00105
00107     char type;
00108
00110     bool flag = false;
00111 };
00112
00113 #endif
```

5.3 Script.hpp

```
00001 #ifndef AED_PROJECT_SCRIPT_H
00002 #define AED_PROJECT_SCRIPT_H
00003
00004 #include "Lecture.hpp"
00005 #include <fstream>
00006 #include <sstream>
00007 #include <unordered_map>
00008 #include <unordered_set>
00009 #include <set>
00010 #include <list>
00011
00016 class Script
00017 {
00018 public:
00025     Student loadStudent(const std::string &studentCode);
00026
00034     std::list<Lecture> loadLecture(std::string ucCode_, std::string classCode_);
00035
00041     void loadClasses(Uc &uc_);
00042
00048     void studentsInLecture(Lecture &oneLecture_);
00049
00056     std::set<Lecture> getSchedule(const std::string &studentCode);
00057
00064     std::vector<Student> studentsinUc(Uc &uc);
00065
00073     std::vector<Student> studentsinClass(std::string ucCode_, std::string classCode_);
00074
00081     std::unordered_set<Student, Student::Hash> studentsInYear(const std::string &year);
00082
00089     int studentsInNUc(int number);
00090
00096     std::vector<std::pair<std::string, int» ucsWithMostStudents();
```

```
00097 };
00098
00099 #endif
```

5.4 Student.hpp

```
00001 #ifndef AED_PROJECT_STUDENT_H
00002 #define AED_PROJECT_STUDENT_H
00003
00004 #include "Uc.hpp"
00005 #include <map>
00006
00011 class Student
00012 {
00013 public:
00015     Student();
00016
00021     Student(const Student &student_);
00022
00028     Student(const std::string &studentCode, const std::string &studentName);
00029
00034     std::string getstudentCode();
00035
00040     void setstudentCode(const std::string &studentCode);
00041
00046     std::string getstudentName();
00047
00052     std::map<std::string, std::string> getSchedule();
00053
00058     void setstudentName(const std::string &studentName);
00059
00064     void addClass(const std::pair<std::string, std::string> &Class);
00065
00072     bool inClass(const std::string &ucCode, const std::string &classCode);
00073
00079     bool operator==(const Student &other) const;
00080
00085     struct Hash
00086     {
00087         std::size_t operator()(const Student &student) const
00088         {
00089             // Combine the hash values of studentCode and studentName to create a unique hash for each
      student.
00090             return std::hash<std::string>{}(student.studentCode) ^
00091                    std::hash<std::string>{}(student.studentName);
00092         }
00093     };
00094
00095 private:
00097     std::string studentCode;
00098
00100     std::string studentName;
00101
00103     std::map<std::string, std::string> schedule;
00104 };
00105
00106 #endif
```

5.5 Uc.hpp

```
00001 #ifndef AED_PROJECT_UC_H
00002 #define AED_PROJECT_UC_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <vector>
00007 #include <stack>
00008 #include <algorithm>
00009
00014 class Uc
00015 {
00016 public:
00018     Uc();
00019
00024     Uc(const std::string &UcCode);
00025
00030     std::string getUcCode();
00031
00036     void setUcCode(const std::string &UcCode);
00037
00042     void addClass(const std::string &UcClass);
00043
```

```
00048     void printClasses(const std::string &SortMethod);
00049
00054     unsigned int classesCount();
00055
00060     std::vector<std::string> getClasses();
00061
00062 private:
00063     /* The code associated with this Uc. */
00064     std::string UcCode;
00065
00066     /* The list of classes within this Uc. */
00067     std::vector<std::string> UcClasses;
00068 };
00069
00070 #endif
```

5.6 Lecture.cpp

```
00001 #include "../inc/Lecture.hpp"
00002 using namespace std;
00003
00004 Lecture::Lecture(const string &ucCode) : uc(ucCode)
00005 {
00006 }
00007
00008 Lecture::Lecture(const std::string &ucCode, const std::string &classCode, const std::string &weekDay,
      const double &startHour, const double &duration, const std::string &type) : uc(ucCode)
00009 {
00010     this->uc.setUcCode(ucCode);
00011     this->uc.addClass(classCode);
00012     this->classCode = classCode;
00013     this->weekDay = weekDay;
00014     this->startHour = startHour;
00015     this->duration = duration;
00016     this->type = type;
00017 }
00018
00019 string Lecture::getClassCode()
00020 {
00021     return this->classCode;
00022 }
00023
00024 Uc Lecture::getUc()
00025 {
00026     return this->uc;
00027 }
00028
00029 void Lecture::setUc(const Uc &uc)
00030 {
00031     this->uc = uc;
00032 }
00033
00034 void Lecture::addStudent(Student &student)
00035 {
00036     for (auto it = this->students.begin(); it != this->students.end(); it++)
00037         if (*it == student)
00038             return;
00039     this->students.push_back(student);
00040 }
00041
00042 void Lecture::removeStudent(const Student &student)
00043 {
00044     int mark;
00045     for (size_t i = 0; i < this->students.size(); i++)
00046     {
00047         if (this->students.at(i) == student)
00048         {
00049             mark = i;
00050         }
00051     }
00052     auto it = this->students.begin();
00053     advance(it, mark);
00054     this->students.erase(it);
00055 }
00056
00057 vector<Student> Lecture::getStudents()
00058 {
00059     return this->students;
00060 }
00061
00062 string Lecture::getWeekDay() const
00063 {
00064     return this->weekDay;
00065 }
00066
```

```
00067 void Lecture::setWeekDay(const string &weekDay)
00068 {
00069     this->weekDay = weekDay;
00070 }
00071
00072 double Lecture::getStartHour() const
00073 {
00074     return this->startHour;
00075 }
00076
00077 void Lecture::setStartHour(const double &startHour)
00078 {
00079     this->startHour = startHour;
00080 }
00081
00082 double Lecture::getDuration() const
00083 {
00084     return this->duration;
00085 }
00086
00087 void Lecture::setDuration(const double &duration)
00088 {
00089     this->duration = duration;
00090 }
00091
00092 string Lecture::getType() const
00093 {
00094     return this->type;
00095 }
00096
00097 void Lecture::setType(const string &type)
00098 {
00099     this->type = type;
00100 }
00101
00102 bool Lecture::operator==(Lecture &other)
00103 {
00104     if ((this->uc.getUcCode() == other.getUc().getUcCode()) && (this->classCode ==
      other.getClassCode()))
00105     {
00106         return true;
00107     }
00108     else
00109         return false;
00110 }
00111
00112 bool Lecture::operator<(const Lecture &other) const
00113 {
00114 std::map<std::string, int> dayValues = {
00115             {"Monday", 0},
00116             {"Tuesday", 1},
00117             {"Wednesday", 2},
00118             {"Thursday", 3},
00119             {"Friday", 4},
00120             {"Saturday", 5},
00121             {"Sunday", 6}
00122         };
00123
00124 if(weekDay != other.getWeekDay()){
00125     return dayValues.at(weekDay) < dayValues.at(other.getWeekDay());
00126 }
00127 return startHour < other.getStartHour();
00128 }
00129
00130 bool Lecture::overlay(Lecture &other){
00131     if(weekDay != other.getWeekDay()) return false;
00132
00133     string o_type = other.getType();
00134     if((type == "TP" && o_type == "TP") || (type == "PL" && o_type == "PL") || (type == "TP" && o_type
      == "PL") || (type == "PL" && o_type == "TP")){
00135         if((startHour >= other.getStartHour()) && startHour <
      (other.getStartHour()+other.getDuration())) return true;
00136         if((startHour < other.getStartHour()) && (startHour + duration) > other.getStartHour()) return
      true;
00137         if(startHour==other.getStartHour()) return true;
00138     }
00139
00140     return false;
00141 }
```

5.7 main.cpp

```
00001 #include "../inc/Uc.hpp"
00002 #include "../inc/Student.hpp"
00003 #include "../inc/Lecture.hpp"
```

```
00004 #include "../inc/Script.hpp"
00005 #include "../inc/Request.hpp"
00006 #include "../inc/App.hpp"
00007
00008
00009 //g++ -o main main.cpp Uc.cpp Lecture.cpp Student.cpp Script.cpp Request.cpp App.cpp
00010
00011
00012 int main()
00013 {
00014     App app;
00015     app.mainMenu();
00016     return 0;
00017 }
```

5.8 Request.cpp

```
00001 #include "../inc/Request.hpp"
00002 using namespace std;
00003
00004 #define MAXIMO 40
00005
00006 Request::Request(string studentCode, char type)
00007 {
00008     this->studentCode = studentCode;
00009     this->type = type;
00010
00011     ifstream log("../requests_log.csv");
00012     string line;
00013     char count = 0;
00014     while (getline(log, line))
00015         count++;
00016     this->id = count;
00017     log.close();
00018 }
00019
00020 bool Request::removeUc(string ucCode)
00021 {
00022     ifstream read_file("../data/students_classes.csv");
00023     string line;
00024     queue<string> lines;
00025     while (getline(read_file, line))
00026     {
00027         istringstream iss(line);
00028         string StudentCode, StudentName, UcCode, classCode;
00029
00030         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00031
00032         if (StudentCode == studentCode && UcCode == ucCode)
00033         {
00034             this->flag = true;
00035             continue;
00036         }
00037
00038         lines.push(line);
00039     }
00040     read_file.close();
00041
00042     if (this->flag)
00043     {
00044         ofstream write_log("../requests_log.csv", ios::app);
00045         write_log « id « ',' « type « ',' « studentCode « ',' « ucCode « endl;
00046         write_log.close();
00047     }
00048     else
00049     {
00050         throw runtime_error("You are not enrolled at this Uc.");
00051         return this->flag;
00052     }
00053
00054     size_t count = lines.size();
00055     ofstream write_file("../data/students_classes.csv");
00056     for (int i = 0; i < count; i++)
00057     {
00058         write_file « lines.front() « endl;
00059         lines.pop();
00060     }
00061     write_file.close();
00062
00063     return this->flag;
00064 }
00065
00066 bool Request::addUc(string ucCodeDestination)
00067 {
```

```
00068     Script script;
00069     Student newStudent = script.loadStudent(this->studentCode);
00070     map<std::string, std::string> new_schedule = newStudent.getSchedule();
00071     if (new_schedule.find(ucCodeDestination) != new_schedule.end())
00072     {
00073         throw runtime_error("Student already registered in this UC");
00074         return this->flag;
00075     }
00076 
00077     if (newStudent.getSchedule().size() >= 7)
00078     {
00079         throw runtime_error("Student registered in maximum number of UC's");
00080         return this->flag;
00081     }
00082 
00083     int max = 0;
00084     int min = 100;
00085     queue<string> eligibleClasses = {};
00086     if(!classesCheck(ucCodeDestination, eligibleClasses)) return this->flag;
00087 
00088     ofstream outFile("../data/students_classes.csv", ios::app);
00089 
00090     if (!outFile.is_open())
00091     {
00092         cerr « "Couldnt open file." « endl;
00093         return this->flag;
00094     }
00095 
00096     outFile « newStudent.getstudentCode() « ',' « newStudent.getstudentName() « ',' «
      ucCodeDestination « ',' « eligibleClasses.front() « endl;
00097 
00098     outFile.close();
00099 
00100     ofstream write_log("../requests_log.csv", ios::app);
00101     write_log « id « ',' « type « ',' « studentCode « ',' « ucCodeDestination « ',' «
      eligibleClasses.front() « endl;
00102     write_log.close();
00103 
00104     this->flag = true;
00105     return this->flag;
00106 }
00107 
00108 bool Request::switchUc(string ucOrigin, string ucDestination)
00109 {
00110     Script script;
00111     Student newStudent = script.loadStudent(this->studentCode);
00112 
00113     map<std::string, std::string> new_schedule = newStudent.getSchedule();
00114 
00115     if (new_schedule.find(ucOrigin) == new_schedule.end())
00116     {
00117         throw runtime_error("You are not enrolled in this UC");
00118         return this->flag;
00119     }
00120     if (new_schedule.find(ucDestination) != new_schedule.end())
00121     {
00122         throw runtime_error("Student already registered in this UC");
00123         return this->flag;
00124     }
00125 
00126     int max = 0;
00127     int min = 100;
00128     queue<string> eligibleClasses = {};
00129     if(!classesCheck(ucDestination, eligibleClasses)) return this->flag;
00130 
00131     ifstream read_file("../data/students_classes.csv");
00132     string line;
00133     queue<string> lines;
00134 
00135     while (getline(read_file, line))
00136     {
00137         istringstream iss(line);
00138         string StudentCode, StudentName, UcCode, classCode;
00139 
00140         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00141 
00142         if (StudentCode == studentCode && UcCode == ucOrigin)
00143         {
00144             this->flag = true;
00145             continue;
00146         }
00147 
00148         lines.push(line);
00149     }
00150     read_file.close();
00151 
```

```
00152     size_t count = lines.size();
00153     ofstream write_file("../data/students_classes.csv");
00154     for (int i = 0; i < count; i++)
00155     {
00156         write_file « lines.front() « endl;
00157         lines.pop();
00158     }
00159     write_file.close();
00160 
00161     ofstream outFile("../data/students_classes.csv", ios::app);
00162 
00163     if (!outFile.is_open())
00164     {
00165         cerr « "Couldnt open file." « endl;
00166         return this->flag;
00167     }
00168     outFile « newStudent.getstudentCode() « ',' « newStudent.getstudentName() « ',' « ucDestination «
    ',' « eligibleClasses.front() « endl;
00169 
00170     outFile.close();
00171 
00172     ofstream write_log("../requests_log.csv", ios::app);
00173     write_log « id « ',' « type « ',' « studentCode « ',' « ucOrigin « ',' « ucDestination « ',' «
    eligibleClasses.front() « endl;
00174     write_log.close();
00175 
00176     this->flag = true;
00177     return this->flag;
00178 }
00179 
00180 void Request::studentRequests(const string &studentCode)
00181 {
00182     ifstream read_file("../requests_log.csv");
00183     string line;
00184     while (getline(read_file, line))
00185     {
00186         istringstream iss(line);
00187         string id_, type_, studentCode_;
00188 
00189         getline(getline(getline(iss, id_, ','), type_, ','), studentCode_, ',');
00190 
00191         if (studentCode_ == studentCode)
00192         {
00193             if (type_ == "1")
00194             {
00195                 string ucCode_, classCode_;
00196                 getline(getline(iss, ucCode_, ','), classCode_, '\r');
00197                 cout « "Operation ID: " « id_ « " | Student added the UC " « ucCode_ « " and entered
    the class " « classCode_ « endl;
00198             }
00199             else if (type_ == "2")
00200             {
00201                 string ucCode_;
00202                 getline(iss, ucCode_, '\r');
00203                 cout « "Operation ID: " « id_ « " | Student removed the UC " « ucCode_ « endl;
00204             }
00205             else if (type_ == "3")
00206             {
00207                 string ucOrigin_, ucDestination_, classCode_;
00208                 getline(getline(getline(iss, ucOrigin_, ','), ucDestination_, ','), classCode_, '\r');
00209                 cout « "Operation ID: " « id_ « " |  Student switched from UC " « ucOrigin_ « " to the
    UC " « ucDestination_ « " and was added to the class " « classCode_ « endl;
00210             }
00211             else if (type_ == "4")
00212             {
00213                 string ucOrigin_, classOrigin_, classDestination_;
00214                 getline(getline(getline(iss, ucOrigin_, ','), classOrigin_, ','), classDestination_,
    '\r');
00215                 cout « "Operation ID: " « id_ « " |  Student switched from class " « classOrigin_ «
    "Of the UC " « ucOrigin_ « " to the class " « classDestination_ « endl;
00216             }
00217         }
00218     }
00219     read_file.close();
00220 }
00221 bool Request::switchClass(std::string currentUc, std::string classOrigin, std::string
    classDestination)
00222 {
00223 
00224     Script script;
00225     Student newStudent = script.loadStudent(this->studentCode);
00226     int max = 0;
00227     int min = 100;
00228     queue<string> eligibleClasses = {};
00229     if(!classesCheck(currentUc, eligibleClasses)) return this->flag;
00230 
00231     while(!eligibleClasses.empty()){
```

```
00232         if(eligibleClasses.front()==classDestination){
00233             this->flag = true;
00234             break;
00235         }
00236         eligibleClasses.pop();
00237     }
00238     if (!(this->flag))
00239     {
00240         throw runtime_error("The selected UC is unavaiable");
00241         return this->flag;
00242     }
00243 
00244     ifstream read_file("../data/students_classes.csv");
00245     string line;
00246     queue<string> lines;
00247     while (getline(read_file, line))
00248     {
00249         istringstream iss(line);
00250         string StudentCode, StudentName, UcCode, classCode;
00251         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00252 
00253         if (StudentCode == this->studentCode && UcCode == currentUc && classCode == classOrigin)
00254         {
00255             this->flag = true;
00256             continue;
00257         }
00258         lines.push(line);
00259     }
00260     read_file.close();
00261 
00262     size_t count = lines.size();
00263     ofstream write_file("../data/students_classes.csv");
00264     for (int i = 0; i < count; i++)
00265     {
00266         write_file « lines.front() « endl;
00267         lines.pop();
00268     }
00269     write_file « this->studentCode « ',' « newStudent.getstudentName() « ',' « currentUc « ',' «
      classDestination « '\r';
00270     write_file.close();
00271 
00272     ofstream write_log("../requests_log.csv", ios::app);
00273     write_log « id « ',' « type « ',' « studentCode « ',' « currentUc « ',' « classOrigin « ',' «
      classDestination « endl;
00274     write_log.close();
00275 
00276     return this->flag;
00277 }
00278 
00279 void Request::undoRequest(unsigned id)
00280 {
00281     ifstream read_file("../requests_log.csv");
00282     string line;
00283     while (getline(read_file, line))
00284     {
00285         istringstream iss(line);
00286         string idFromFile, typeFromFile, studentCodeFromFile;
00287         getline(getline(getline(iss, idFromFile, ','), typeFromFile, ','), studentCodeFromFile, ',');
00288         if (idFromFile == to_string(id))
00289         {
00290             if (typeFromFile == "1")
00291             {
00292                 string ucCodeFromFile, classCodeFromFile;
00293                 getline(getline(iss, ucCodeFromFile, ','), classCodeFromFile, '\r');
00294 
00295                 Request(studentCodeFromFile, '2').removeUc(ucCodeFromFile);
00296                 break;
00297             }
00298             else if (typeFromFile == "2")
00299             {
00300                 string ucCodeFromFile;
00301                 getline(iss, ucCodeFromFile, '\r');
00302 
00303                 Request(studentCodeFromFile, '1').addUc(ucCodeFromFile);
00304                 break;
00305             }
00306             else if (typeFromFile == "3")
00307             {
00308                 string originFromFile, destinationFromFile, classCodeFromFile;
00309                 getline(getline(getline(iss, originFromFile, ','), destinationFromFile, ','),
      classCodeFromFile, '\r');
00310 
00311                 Request(studentCodeFromFile, '3').switchUc(destinationFromFile, originFromFile);
00312                 break;
00313             }
00314             else if (typeFromFile == "4")
```

```
00315             {
00316                 string ucCodeFromFile, originFromFile, destinationFromFile;
00317                 getline(getline(getline(iss, ucCodeFromFile, ',', originFromFile, ',',
      destinationFromFile, '\r');
00318
00319                 Request(studentCodeFromFile, '4').switchClass(ucCodeFromFile, destinationFromFile,
      originFromFile);
00320                 break;
00321             }
00322         }
00323     }
00324
00325     if (read_file.eof())
00326         throw runtime_error("This request does not exist.");
00327 }
00328
00329 void Request::adminRequests()
00330 {
00331     ifstream read_file("../requests_log.csv");
00332     string line;
00333     while (getline(read_file, line))
00334     {
00335         istringstream iss(line);
00336         string id_, type_, studentCode_;
00337
00338         getline(getline(getline(iss, id_, ',', type_, ',', studentCode_, ',');
00339
00340         if (type_ == "1")
00341         {
00342             string ucCode_, classCode_;
00343             getline(getline(iss, ucCode_, ',', classCode_, '\r');
00344             cout « "Operation ID: " « id_ « " | Student " « studentCode_ « " added the UC " « ucCode_
      « " and entered the class " « classCode_ « endl;
00345         }
00346         else if (type_ == "2")
00347         {
00348             string ucCode_;
00349             getline(iss, ucCode_, '\r');
00350             cout « "Operation ID: " « id_ « " | Student " « studentCode_ « " removed the UC " «
      ucCode_ « endl;
00351         }
00352         else if (type_ == "3")
00353         {
00354             string ucOrigin_, ucDestination_, classCode_;
00355             getline(getline(getline(iss, ucOrigin_, ',', ucDestination_, ',', classCode_, '\r');
00356             cout « "Operation ID: " « id_ « " |  Student " « studentCode_ « " switched from UC " «
      ucOrigin_ « " to the UC " « ucDestination_ « " and was added to the class " « classCode_ « endl;
00357         }
00358         else if (type_ == "4")
00359         {
00360
00361             string ucOrigin_, classOrigin_, classDestination_;
00362             getline(getline(getline(iss, ucOrigin_, ',', classOrigin_, ',', classDestination_,
      '\r');
00363             cout « "Operation ID: " « id_ « " |  Student " « studentCode_ « " switched from class " «
      classOrigin_ « " of the UC " « ucOrigin_ « " to the class " « classDestination_ « endl;
00364         }
00365     }
00366     read_file.close();
00367 }
00368
00369 bool Request::classesCheck(std::string ucDestination, std::queue<std::string> &eligibleClasses)
00370 {
00371     Script script;
00372     Uc destination = Uc(ucDestination);
00373     script.loadClasses(destination);
00374     int max = 0;
00375     int min = 100;
00376
00377     for (string currClass : destination.getClasses())
00378     {
00379         int classSize = script.studentsinClass(destination.getUcCode(), currClass).size();
00380         if (classSize + 1 > max)
00381         {
00382             max = classSize + 1;
00383         }
00384         else if (classSize + 1 < min)
00385         {
00386             min = classSize + 1;
00387         }
00388
00389         if (classSize + 1 <= MAXIMO && (max - classSize - 1) <= 4)
00390         {
00391             eligibleClasses.push(currClass);
00392         }
00393     }
00394
```

```
00395     if (max > MAXIMO)
00396     {
00397         throw runtime_error("All classes with maximum occupancy");
00398         return this->flag;
00399     }
00400
00401     if ((max - min) > 4)
00402     {
00403         throw runtime_error("Adding the student would affect the balance of classes in this UC");
00404         return this->flag;
00405     }
00406     if (eligibleClasses.size() < 1)
00407     {
00408         throw runtime_error("This UC hasn't avaiable classes");
00409         return this->flag;
00410     }
00411
00412     bool check = false;
00413     for (Lecture currentLecture : script.loadLecture(ucDestination, eligibleClasses.front()))
00414     {
00415         for (Lecture studentLecture : script.getSchedule(studentCode))
00416         {
00417             if (studentLecture.overlay(currentLecture))
00418             {
00419                 eligibleClasses.pop();
00420                 check = true;
00421                 break;
00422             }
00423         }
00424         if (eligibleClasses.empty())
00425         {
00426             throw runtime_error("This UC will disturb the student's schedule");
00427             return this->flag;
00428         }
00429         if (check)
00430             continue;
00431     }
00432     return true;
00433 }
```

5.9 Script.cpp

```
00001 #include "../inc/Script.hpp"
00002 using namespace std;
00003
00004 Student Script::loadStudent(const string &studentCode)
00005 {
00006     Student student;
00007
00008     ifstream file("../data/students_classes.csv");
00009
00010     if (!file.is_open())
00011     {
00012         return student;
00013     }
00014
00015     string line;
00016     getline(file, line);
00017     while (getline(file, line))
00018     {
00019         istringstream iss(line);
00020         string studentCodeFromFile, studentNameFromFile, ucCodeFromFile, classCodefromFile;
00021         getline(getline(getline(getline(iss, studentCodeFromFile, ','), studentNameFromFile, ','),
      ucCodeFromFile, ','), classCodefromFile, '\r');
00022
00023         if (studentCodeFromFile == studentCode)
00024         {
00025             student.setstudentCode(studentCodeFromFile);
00026             student.setstudentName(studentNameFromFile);
00027             student.addClass(pair<string, string>{ucCodeFromFile, classCodefromFile});
00028         }
00029     }
00030
00031     file.close();
00032
00033     return student;
00034 }
00035
00036 list<Lecture> Script::loadLecture(string ucCode_, string classCode_)
00037 {
00038     list<Lecture> result = {};
00039     ifstream file("../data/classes.csv");
00040     if (!file.is_open())
00041     {
00042         cout « "Failed to open the file." « endl;
```

```
00043         return result;
00044     }
00045
00046     string line;
00047
00048     while (getline(file, line))
00049     {
00050         istringstream iss(line);
00051         string ClassCode, UcCode, Weekday, strStarHour, strDuration, Type;
00052         double StartHour, Duration;
00053
00054         getline(getline(getline(getline(getline(getline(iss, ClassCode, ','), UcCode, ','), Weekday,
      ','), strStarHour, ','), strDuration, ','), Type, '\r');
00055
00056         try
00057         {
00058             StartHour = stod(strStarHour);
00059             Duration = stod(strDuration);
00060         }
00061         catch (const std::invalid_argument &e)
00062         {
00063         }
00064         catch (const std::out_of_range &e)
00065         {
00066             std::cerr « "Erro: Conversão fora do alcance. O número é muito grande ou muito pequeno." «
      std::endl;
00067         }
00068
00069         if (ucCode_ == UcCode && classCode_ == ClassCode)
00070         {
00071
00072             Lecture lecture(UcCode, ClassCode, Weekday, StartHour, Duration, Type);
00073             result.push_back(lecture);
00074         }
00075     }
00076
00077     file.close();
00078
00079     return result;
00080 }
00081
00082 void Script::loadClasses(Uc &uc_)
00083 {
00084     ifstream file;
00085     file.open("../data/classes_per_uc.csv", std::ios::in);
00086
00087     if (!file.is_open())
00088         cout « "not open";
00089     string line;
00090
00091     while (getline(file, line))
00092     {
00093         istringstream stream(line);
00094         string Code, ClassCode;
00095
00096         if (getline(stream, Code, ','))
00097         {
00098             if (Code == uc_.getUcCode())
00099             {
00100                 if (getline(stream, ClassCode, '\r'))
00101                 {
00102                     uc_.addClass(ClassCode);
00103                 }
00104             }
00105         }
00106     }
00107     file.close();
00108 }
00109
00110 void Script::studentsInLecture(Lecture &oneLecture_)
00111 {
00112
00113     ifstream file("../data/students_classes.csv");
00114     if (!file.is_open())
00115     {
00116         cout « "Failed to open the file." « endl;
00117     }
00118
00119     string line;
00120
00121     while (getline(file, line))
00122     {
00123         istringstream iss(line);
00124         string StudentCode, StudentName, UcCode, classCode;
00125
00126         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
```

```
00127
00128         if ((UcCode == oneLecture_.getUc().getUcCode()) && (classCode == oneLecture_.getClassCode()))
00129         {
00130             Student student(StudentCode, StudentName);
00131             oneLecture_.addStudent(student);
00132         }
00133     }
00134
00135     file.close();
00136 }
00137
00138 set<Lecture> Script::getSchedule(const string &studentCode_)
00139 {
00140     Script script;
00141     Student oneStudent_ = script.loadStudent(studentCode_);
00142     set<Lecture> result = {};
00143
00144     ifstream file("../data/classes.csv");
00145     if (!file.is_open())
00146     {
00147         cout « "Failed to open the file." « endl;
00148         return result;
00149     }
00150
00151     string line;
00152
00153     while (getline(file, line))
00154     {
00155         istringstream iss(line);
00156         string ClassCode, UcCode, Weekday, strStarHour, strDuration, Type;
00157         double StartHour, Duration;
00158
00159         getline(getline(getline(getline(getline(getline(iss, ClassCode, ','), UcCode, ','), Weekday,
    ','), strStarHour, ','), strDuration, ','), Type, '\r');
00160
00161         try
00162         {
00163             StartHour = stod(strStarHour);
00164             Duration = stod(strDuration);
00165         }
00166         catch (const std::invalid_argument &e)
00167         {
00168         }
00169         catch (const std::out_of_range &e)
00170         {
00171             std::cerr « "Erro: Conversão fora do alcance. O número é muito grande ou muito pequeno." «
    std::endl;
00172         }
00173
00174         if (oneStudent_.inClass(UcCode, ClassCode))
00175         {
00176
00177             Lecture lecture(UcCode, ClassCode, Weekday, StartHour, Duration, Type);
00178             result.insert(lecture);
00179         }
00180     }
00181
00182     file.close();
00183
00184     return result;
00185 }
00186
00187 vector<Student> Script::studentsinUc(Uc &uc)
00188 {
00189     vector<Student> students;
00190
00191     ifstream file("../data/students_classes.csv");
00192     if (!file.is_open())
00193     {
00194         cout « "Failed to open the file." « endl;
00195     }
00196
00197     string line;
00198
00199     while (getline(file, line))
00200     {
00201         istringstream iss(line);
00202         string StudentCode, StudentName, UcCode, classCode;
00203
00204         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
    classCode, '\r');
00205
00206         if (UcCode == uc.getUcCode())
00207         {
00208             Student student{StudentCode, StudentName};
00209             students.push_back(student);
00210         }
```

```
00211     }
00212
00213     file.close();
00214     return students;
00215 }
00216
00217 vector<Student> Script::studentsinClass(string ucCode_, string classCode_)
00218 {
00219     vector<Student> students;
00220
00221     ifstream file("../data/students_classes.csv");
00222     if (!file.is_open())
00223     {
00224         cout « "Failed to open the file." « endl;
00225     }
00226
00227     string line;
00228
00229     while (getline(file, line))
00230     {
00231         istringstream iss(line);
00232         string StudentCode, StudentName, UcCode, classCode;
00233
00234         getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00235
00236         if (UcCode == ucCode_ && classCode == classCode_)
00237         {
00238             Student student{StudentCode, StudentName};
00239             students.push_back(student);
00240         }
00241     }
00242
00243     file.close();
00244     return students;
00245 }
00246
00247 unordered_set<Student, Student::Hash> Script::studentsInYear(const std::string &year)
00248 {
00249     unordered_set<Student, Student::Hash> students;
00250     ifstream file("../data/students_classes.csv");
00251     if (!file.is_open())
00252     {
00253         cout « "Failed to open the file." « endl;
00254     }
00255
00256     string line;
00257     while (getline(file, line))
00258     {
00259         istringstream iss(line);
00260
00261         if (line.substr(0, 4) == year)
00262         {
00263             string StudentCode, StudentName, UcCode, classCode;
00264             getline(getline(getline(getline(iss, StudentCode, ','), StudentName, ','), UcCode, ','),
      classCode, '\r');
00265             Student student{StudentCode, StudentName};
00266
00267             students.insert(student);
00268         }
00269     }
00270
00271     file.close();
00272     return students;
00273 }
00274
00275 int Script::studentsInNUc(int number)
00276 {
00277     int result = 0;
00278     int aux = 0;
00279     ifstream file("../data/students_classes.csv");
00280     if (!file.is_open())
00281     {
00282         cout « "Failed to open the file." « endl;
00283     }
00284
00285     unordered_map<string, unordered_map<string, bool» studentUCs;
00286     string line;
00287
00288     while (std::getline(file, line))
00289     {
00290         istringstream iss(line);
00291         string studentCode, studentName, ucCode, classCode;
00292         getline(getline(getline(getline(iss, studentCode, ','), studentName, ','), ucCode, ','),
      classCode, '\r');
00293
00294         studentUCs[studentCode][ucCode] = true;
```

```
00295     }
00296
00297     int count = 0;
00298
00299     for (const auto &student : studentUCs)
00300     {
00301         if (student.second.size() >= number)
00302         {
00303             count++;
00304         }
00305     }
00306
00307     return count;
00308 }
00309
00310 vector<pair<string, int» Script::ucsWithMostStudents()
00311 {
00312     map<string, int> aux = {};
00313
00314     ifstream file("../data/students_classes.csv");
00315     if (!file.is_open())
00316     {
00317         cout « "Failed to open the file." « endl;
00318     }
00319
00320     string line;
00321     getline(file, line);
00322     while (getline(file, line))
00323     {
00324         istringstream iss(line);
00325         string studentCode, studentName, ucCode, classCode;
00326         getline(getline(getline(getline(iss, studentCode, ','), studentName, ','), ucCode, ','),
      classCode, '\r');
00327
00328         aux[ucCode]++;
00329     }
00330     file.close();
00331
00332     vector<pair<string, int» result = {};
00333
00334     for (const pair<string, int> &p : aux)
00335         result.push_back(p);
00336
00337     sort(result.begin(), result.end(), [](pair<string, int> p1, pair<string, int> p2) -> bool
00338          { return p1.second > p2.second; });
00339
00340     return result;
00341 }
```

5.10 Student.cpp

```
00001 #include "../inc/Student.hpp"
00002 using namespace std;
00003
00004 Student::Student()
00005 {
00006     this->studentName = "NO_NAME";
00007     this->studentCode = "NO_CODE";
00008     this->schedule = {};
00009 }
00010
00011 Student::Student(const Student &student_)
00012 {
00013     this->studentName = student_.studentName;
00014     this->studentCode = student_.studentCode;
00015     this->schedule = student_.schedule;
00016 }
00017
00018 Student::Student(const string &studentCode, const string &studentName)
00019 {
00020     this->studentCode = studentCode;
00021     this->studentName = studentName;
00022     this->schedule = {};
00023 }
00024 string Student::getstudentCode()
00025 {
00026     return this->studentCode;
00027 }
00028 void Student::setstudentCode(const string &studentCode)
00029 {
00030     this->studentCode = studentCode;
00031 }
00032 string Student::getstudentName()
00033 {
00034     return this->studentName;
```

```
00035 }
00036 void Student::setstudentName(const string &studentName)
00037 {
00038     this->studentName = studentName;
00039 }
00040 void Student::addClass(const pair<string, string> &Class)
00041 {
00042
00043     schedule.insert(Class);
00044 }
00045
00046 map<string, string> Student::getSchedule(){
00047     return this->schedule;
00048 }
00049
00050 bool Student::inClass(const string &ucCode_, const string &classCode_)
00051 {
00052     auto it = schedule.find(ucCode_);
00053     if (it != schedule.end())
00054     {
00055
00056         if (it->second == classCode_)
00057             return true;
00058         else
00059             return false;
00060     }
00061     else
00062     {
00063         return false;
00064     }
00065 }
00066
00067 bool Student::operator==(const Student &other) const
00068 {
00069     return this->studentCode == other.studentCode && this->studentName == other.studentName;
00070 }
```

5.11 Uc.cpp

```
00001 #include "../inc/Uc.hpp"
00002 #include <iomanip>
00003 using namespace std;
00004
00005 Uc::Uc()
00006 {
00007     UcCode = "NO_NAME";
00008     UcClasses = vector<string>();
00009 }
00010
00011 Uc::Uc(const string &UcCode) : UcCode(UcCode)
00012 {
00013 }
00014
00015 string Uc::getUcCode()
00016 {
00017     return UcCode;
00018 }
00019
00020 void Uc::setUcCode(const string &UcCode)
00021 {
00022     this->UcCode = UcCode;
00023 }
00024
00025 void Uc::addClass(const string &UcClass)
00026 {
00027     for (vector<string>::iterator it = UcClasses.begin(); it != UcClasses.end(); it++)
00028         if (*it == UcClass)
00029             return;
00030     UcClasses.push_back(UcClass);
00031     sort(UcClasses.begin(), UcClasses.end());
00032 }
00033
00034 void Uc::printClasses(const string &SortMethod)
00035 {
00036     if (SortMethod == "1") {
00037         for (const string &turma : UcClasses){
00038             cout « "|" « turma « "|" « endl;
00039         }
00040     } else if (SortMethod == "2") {
00041         stack<string> reverse;
00042         for(const string &turma : UcClasses) reverse.push(turma);
00043         while(!reverse.empty()){
00044
00045             cout « "|" «reverse.top() « endl;
00046             reverse.pop();
```

```
00047         }
00048     } else {
00049         cout « "Selecione um método de ordenação válido" « endl;
00050     }
00051 }
00052
00053 vector<string> Uc::getClasses(){
00054     return this->UcClasses;
00055 }
00056
00057 unsigned int Uc::classesCount()
00058 {
00059     return UcClasses.size();
00060 }
```

Index