Sistema de Gestão de Horários

Turma 2LEIC02
Bruno Ferreira | Eduardo Portugal | Xavier Martins

AED 2023/24

Contexto do Projeto

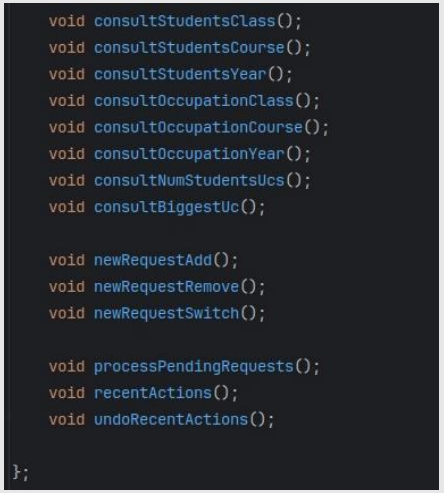
Neste projeto fomos desafiados a criar um sistema de gestão de horários.
Este deveria ter presente várias funcionalidades, entre elas a modificação, visualização e ordenação.
A par disto, devíamos optar sempre por utilizar as estruturas de dados mais adequadas ao problema em questão.

Classes

- **App**

Classe responsável por controlar os menus do programa, interligando as ações pretendidas por parte do utilizador com a visualização das mesma.

```
class App {
  private:
   Data data;

  public:
   App();
   void run();
   void close();
   void mainMenu();
   bool tryAgainMenu();
   void handleErrors(const string& error);

   void consultMenu();
   void newRequestMenu();
   void processRequestMenu();

   void consultStudentSchedule();
   void consultClassSchedule();
```



```
   void consultStudentsClass();
   void consultStudentsCourse();
   void consultStudentsYear();
   void consultOccupationClass();
   void consultOccupationCourse();
   void consultOccupationYear();
   void consultNumStudentsUcs();
   void consultBiggestUc();

   void newRequestAdd();
   void newRequestRemove();
   void newRequestSwitch();

   void processPendingRequests();
   void recentActions();
   void undoRecentActions();

};
```

Classes

- **Data**

Classe responsável por guardar e processar toda a informação do programa.

Classes

```
class Data {
   private:
    const string CLASSES_FILENAME = "classes.csv";
    const string CLASSES_PER_UC_FILENAME = "classes_per_uc.csv";
    const string REQUESTS_HISTORY_FILENAME = "request_history.csv";
    const string PENDENT_REQUESTS_FILENAME = "pendent_requests.csv";
    const string STUDENTS_CLASS_FILENAME = "students_classes.csv";
    const string DIRECTORY_PATH = "/home/xavier/Desktop/AED/Projeto/feup-aed-project-1/schedule/";

private:
    map<string, Uc> ucs;
    unordered_map<string, set<string>> ucsCodesByClassCode;
    map<int, Student> students;
    queue<Request> pendentRequests;
    list<Request> requestHistory;

   private:
    bool isNumeric(const string &str);

   public:
    // Data constructor
    Data();

    // ucs related methods
    map<string, Uc> getAllUcs() const;
    Uc &getUc(const string &ucCode);
    bool ucExists(const string &ucCode);
    string consultStudentsClass(const string &ucCode, const string &classCode);
    string consultStudentsCourse(const string &ucCode);
    string consultStudentsYear(const string &year);
    string consultOccupationClass(const string &ucCode, const string &classCode);
    string consultOccupationCourse(const string &ucCode);
    string consultOccupationYear(const string &year);
    string consultNumStudentsUcs(const string &nUcs);
    string consultBiggestUc(bool ascendingOrder);

    set<string> getUcsByClassCode(const string &classCode) const;
```

```
bool classExists(const string &ucCode, const string &classCode);

// students related methods
map<int, Student> getAllStudents() const;
Student &getStudent(int studentCode);
bool studentExists(int studentCode);

// requests related methods

void createAddRequest(const string &studentCode, const string &ucCode,
                      const string &destinyclassCode);
void createRemoveRequest(const string &studentCode, const string &ucCode);
void createSwitchRequest(const string &studentCode, const string &ucCode,
                         const string &destinyClassCode);

string processRequests();

string undoRequest(int requestNumber);

string validRequest(const Request &request) const;
void applyRequest(const Request &request);

queue<Request> &getPendentRequests();
string getRequestHistory() const;

// Read and Write data
void loadData();
void saveData();

// requests Pendent files
void readPendentRequestsFile();
void writePendentRequestsFile();

// requests history file
void readRequestHistoryFile();
void writeRequestHistoryFile();
```

```
// classes_per_uc.csv
void readClassesPerUcFile();
void writeClassesPerUcFile();

// classes.csv
void readClassesFile();
void writeClassesFile();

// students_classes.csv
void readStudentsClassesFile();
void writeStudentsClassesFile();
```

Classes

- **UserInterface**

Classe responsável pela *interface*, mostra os menus e todas as informações pretendidas pelo utilizador.

```
class UserInterface
{
public:
    static void printMainMenu();
    static char readOption(const string& msg);
    static string readCode(const string& msg);
    static void printError(const string& error);
    static void printMessage(const string& msg);
    static void printStudentSchedule(const Student& student);
    static void printClassSchedule(const string& classCode, const set<string>& ucs, const map<int,set<string>>& schedulesByDay);
    static void pressEnterToContinue();
    static void printConsultMenu();
    static void printTryAgainMenu();
    static void printNewRequestMenu();
    static void printProcessRequestMenu();

};
```

Classes

- **Uc**

Classe que representa uma Uc. Esta guarda todas as turmas da respetiva Uc.

```
class Uc {
   private:
    string ucCode;
    // classCode -> class
    map<string, Class> classes;

   public:
    // Uc constructor
    Uc();
    Uc(const string &ucCode);

    // Set & Getter -> ucCode
    string getUcCode() const;

    // classes related methods
    void addClass(const Class &newClass);
    Class &getClass(const string &classCode);
    const map<string, Class> &getAllClasses() const;
    bool hasClass(const string &classCode) const;
    bool hasVacancies() const;
    bool balancedClasses(const string &originClassCode,
                         const string &destinyClassCode) const;
};
```

Classes

- **Class**

Classe que representa uma turma de uma dada Uc.

```
class Class {
   private:
    string ucCode;
    string classCode;
    vector<ClassSession> schedule;
    set<int> students;
    const static int MAX_STUDENTS = 30;

   public:
    Class();
    Class(const string& ucCode, const string& classCode);
    string getUcCode() const;
    string getClassCode() const;
    void addClassSchedule(const ClassSession& newClassSchedule);
    vector<ClassSession> getAllClassSchedules() const;
    bool invalidOverlaps(const Class& other) const;
    void addStudent(int newStudentCode);
    void removeStudent(int studentCode);
    set<int> getAllStudents() const;
    int numberStudents() const;
    bool hasVacancies() const;
    int numberVacancies() const;
};
```

Classes

- **Student**

Classe que representa um estudante e toda a informação do mesmo.

```
class Student {
   private:
    int studentCode;
    string studentName;
    // ucCode -> class
    map<string, Class> classes;

   public:
    // CONSTRUCTORS
    Student();
    Student(int studentCode, const string &studentName);

    // GETTERS
    int getStudentCode() const;
    string getStudentName() const;
    map<string, Class> getAllClasses() const;


    void addClass(const Class &newClass);
    void removeClass(const string &ucCode);
    string findConflictClass(const string& ucCode, const Class& destinyClass) const;
    string getUcClassCode(const string &ucCode) const;
    int numberOfUcs() const;
    bool hasUc(const string &ucCode) const;
    string getStudentAsString() const;
};
```

Classes

- **ClassSession**

Classe responsável por guardar o horário para uma dada turma.

```
class ClassSession {
   private:
    string weekday;
    TimeInterval timeInterval;
    string type;

   public:
    // Class constructor constructor
    ClassSession(const string &weekday, const TimeInterval &time,
                 const string &type);

    // Set & Getter -> weekday
    const string &getWeekday() const;

    // Set & Getter -> timeInterval
    TimeInterval getTimeInterval() const;
    double getStartHour() const;
    double getDuration() const;

    // Set & Getter -> type
    string getType() const;
    void setType(const string &newType);

    // Compare Schedules
    bool overlaps(const ClassSession &schedule) const;
    bool invalidOverlaps(const ClassSession &schedule) const;

    bool operator<(const ClassSession &other) const;
};
```

Classes

- **TimeInterval**

Estrutura que guarda informação relativa a um intervalo de tempo.

```
struct TimeInterval
{
    int startHour;
    int startMinute;
    int endHour;
    int endMinute;

    TimeInterval(double startTime, double duration);
    string getTimeIntervalAsString() const;
};
```

Classes

- **Request**

Estrutura que armazena informação relativa a um pedido.

```
struct Request {
    char type;
    string originClassCode;
    string destinyClassCode;
    string ucCode;
    int studentCode;

   public:
    Request(int studentCode, char type, string ucCode, string originClassCode,
            string destinyClassCode);
    string stringInfo() const;
};
```

Estruturas de Dados

- **Set**

Estrutura usada essencialmente no armazenamento dos estudantes, por exemplo numa turma.
As principais vantagens desta estrutura são garantir que não há repetição de nenhum valor, algo crucial nesta operação, pois não queremos que um estudante seja invocado mais que uma vez e os valores estarem ordenados.

Estruturas de Dados

- **Vetor**

Estrutura usada para guardar o horário de uma dada turma.
As principais vantagens desta estrutura são não ter um tamanho fixo, ser uma estrutura simples e ser bastante eficiente ao iterar.

Estruturas de Dados

- **Map**

Estrutura importante do tipo chave-valor, permitindo a associação entre duas informações, usada em casos como associar um estudante ao seu número.
As principais vantagens desta estrutura são a garantia de que a chave é única e utilizar pesquisa binaria quando se insere, apaga ou procura alguma informação.

Estruturas de Dados

- **Queue**

Estrutura usada para guardar os pedidos pendentes do utilizador.
As principais vantagens desta estrutura são a sua ordenação do tipo FIFO, e ainda a sua eficiência ao inserir e eliminar.

Estruturas de Dados

- **List**

Estrutura usada para guardar o histórico de pedidos aprovados de forma a permitir a funcionalidade de reverter um destes pedidos.
As principais vantagens desta estrutura são não ter uma tamanho fixo e o a inserção e eliminação ser bastante eficiente.

Funcionalidades Implementadas

Leitura de dados

Esta é das primeiras funcionalidades do programa, onde é chamado o método loadData(), responsável por chamar outros métodos que vão ler as informações dos ficheiros onde constam os alunos, turmas, ucs e horários.

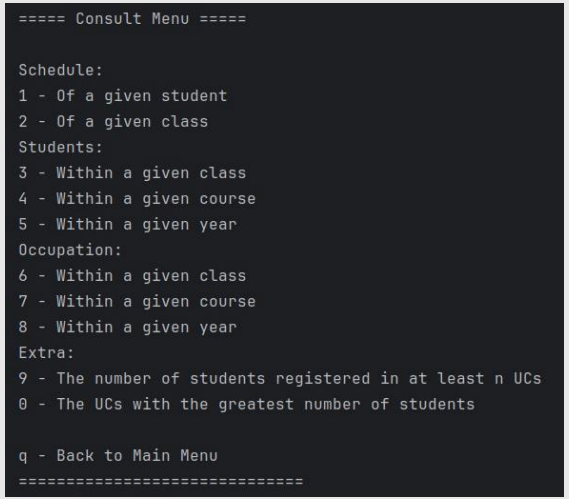
```
void Data::loadData() {
    readClassesPerUcFile();
    readClassesFile();
    readStudentsClassesFile();
    readRequestHistoryFile();
    readPendentRequestsFile();
}
```

Funcionalidades Implementadas

Consulta de dados

Esta funcionalidade permite visualizar diversos tipos de informação, nomeadamente:

- Os horários de um estudante ou de uma turma;
- Os estudantes de uma turma, uc ou de um ano;
- A ocupação de uma turma, uc ou de um ano;
- O nº de estudantes registados em pelo menos n Ucs;
- O nº de estudantes por uc.



Funcionalidades Implementadas

```
Student Code (i.e.: 123456789): 201920727

#########################################################################################################################
Student Code:   201920727
Student Name:   Ines
Enrolled in 3 UCs:  [L.EIC002-1LEIC05]  [L.EIC003-1LEIC07]  [L.EIC005-1LEIC01]
#########################################################################################################################

Monday    : [08:30-10:30] L.EIC002-1LEIC05 (T)      [10:30-12:30] L.EIC002-1LEIC05 (TP)
Wednesday : [09:30-10:30] L.EIC003-1LEIC07 (T)      [10:30-12:30] L.EIC003-1LEIC07 (TP)
Friday    : [08:00-10:00] L.EIC005-1LEIC01 (T)      [10:00-11:00] L.EIC003-1LEIC07 (T)      [11:00-13:00] L.EIC005-1LEIC01 (TP)
```

```
#########################################################################################################################
Class Code: 1LEIC01
UCs: L.EIC001   L.EIC002    L.EIC003    L.EIC004    L.EIC005    UP001
#########################################################################################################################

Monday    : [08:30-10:30] L.EIC002 (T)      [10:30-12:00] L.EIC001 (TP)
Tuesday   : [08:30-10:30] L.EIC002 (TP)     [10:30-12:00] L.EIC001 (T)
Wednesday : [09:30-10:30] L.EIC003 (T)      [10:30-12:30] L.EIC004 (TP)     [14:00-15:30] UP001 (TP)
Thursday  : [09:00-11:00] L.EIC003 (TP)     [11:00-13:00] L.EIC004 (T)
Friday    : [08:00-10:00] L.EIC005 (T)      [10:00-11:00] L.EIC003 (T)      [11:00-13:00] L.EIC005 (TP)
```

```
Insert the UC code (i.e.: L.EIC001): L.EIC001

Insert the Class Code (i.e.: 1LEIC01): 1LEIC02
#########################################################################################################################

202021152 - Alice :                 [L.EIC001-1LEIC02]  [L.EIC002-1LEIC08]  [L.EIC003-1LEIC08]  [L.EIC005-1LEIC08]  [L.EIC013-2LEIC14]
202021407 - Diana :                 [L.EIC001-1LEIC02]  [L.EIC002-1LEIC08]  [L.EIC003-1LEIC08]  [L.EIC004-1LEIC05]  [L.EIC005-1LEIC08]
202022172 - Sara :                  [L.EIC001-1LEIC02]  [L.EIC002-1LEIC08]  [L.EIC003-1LEIC08]  [L.EIC004-1LEIC05]  [L.EIC005-1LEIC08]

#########################################################################################################################
```

```
Insert the UC code (i.e.: L.EIC001): L.EIC001

Insert the Class Code (i.e.: 1LEIC01): 1LEIC02

Uc: L.EIC001   Class: 1LEIC02
Number of students: 3
Number of vacancies: 27
```

```
How many Ucs: 3

There is 648 students registered in at least 3 Ucs
```

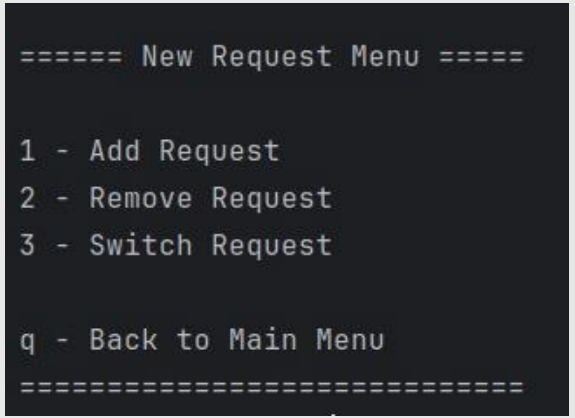
```
q - Back to Main Menu
==============================
Select an option: 0
L.EIC013 -- 400 students
L.EIC015 -- 357 students
L.EIC014 -- 352 students
L.EIC012 -- 342 students
L.EIC011 -- 329 students
L.EIC023 -- 281 students
L.EIC025 -- 273 students
L.EIC021 -- 262 students
L.EIC022 -- 223 students
L.EIC024 -- 209 students
L.EIC002 -- 85 students
L.EIC003 -- 74 students
L.EIC004 -- 71 students
L.EIC005 -- 50 students
L.EIC001 -- 31 students
UP001 -- 0 students

[1] Ascending Order    [2] Descending Order    [q] Go back
```

Funcionalidades Implementadas

Pedidos de Mudança

Esta funcionalidade permite o utilizador fazer alguns tipos de pedidos, nomeadamente:

- Pedido de entrada;
- Pedido de saída;
- Pedido de troca.



Funcionalidades Implementadas

```
Student Code (i.e.: 123456789): 201920727

Insert the UC code (i.e.: L.EIC001): L.EIC001

Insert the Class Code (i.e.: 1LEIC01): 1LEIC01
Your request was saved in the pending request list.
```

```
Student Code (i.e.: 123456789): 201920727

Insert the UC code (i.e.: L.EIC001): L.EIC002
Your request was saved in the pending request list.
```

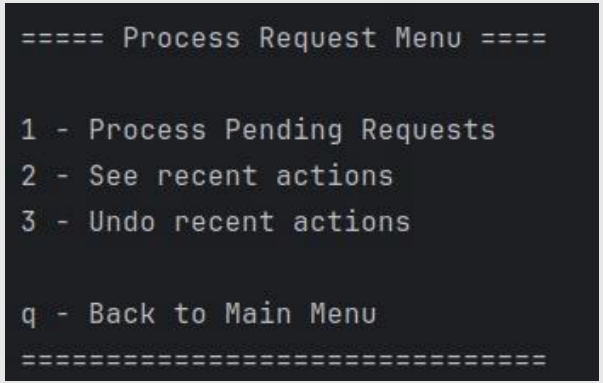
```
Insert the UC code (i.e.: L.EIC001): L.EIC003

Insert the Class Code Destiny (i.e.: 1LEIC01): 1LEIC01
Your request was saved in the pending request list.
```

Funcionalidades Implementadas

Processamento de Pedidos

Esta funcionalidade permite o processamento dos pedidos de diferentes formas, nomeadamente:

- Processar os pedidos pendentes;
- Ver as ações mais recentes;
- Reverter ações recentes.



Funcionalidades Implementadas

```
Process finished!
3 total requests processed. 2 sucessfully accepted!

    Processes log:

1-  Add Request - Student: 201920727
Joined Uc: L.EIC001. In class: 1LEIC01
Status: Not accepted.
Info: Class to enter conflicts with student's current schedule. There is an overlap with the class L.EIC002-1LEIC05

2-  Remove Request - Student: 201920727
Exited Uc: L.EIC002. Leaving the class: 1LEIC05
Status: Successful

3-  Switch Request - Student: 201920727
In Uc: L.EIC003. Exited: 1LEIC07. Joined: 1LEIC01
Status: Successful
```

```
1-  Switch Request - Student: 201920727
In Uc: L.EIC003. Exited: 1LEIC07. Joined: 1LEIC01

2-  Remove Request - Student: 201920727
Exited Uc: L.EIC002. Leaving the class: 1LEIC05
```

```
Select a request to undo: 1
Processing undo request...
Status: Not accepted.
Info: Operation affects Class balance.
```

Gestão de erros

```
Student Code (i.e.: 123456789): 201920727

Insert the UC code (i.e.: L.EIC001): L.EIC002

Insert the Class Code (i.e.: 1LEIC01): 1LEIC02
ERROR: Student is already enrolled in that Unit Course

---------------
1 - Try again
q - Go back
---------------
```

```
Insert the UC code (i.e.: L.EIC001): Llllw

Insert the Class Code (i.e.: 1LEIC01): awdq
ERROR: This UC doesn't exist in the database.

---------------
1 - Try again
q - Go back
---------------
```