SEP1Y Final Project

Time Scheduling at VIA Made Easy

Daniel Lopes Adrião, 315274

Dragos Daniel Bonaparte, 315261

Laura do Bem Rebelo, 315174

Matas Armonaitis, 315263

Supervisors:

Steffen Andersen (SVA), Mona Andersen (MWA)

REPORT: 35523 characters

Software Technology Engineering

1st Semester

15.12.2021

Table of contents

Abstract

The aim of this project is to provide VIA's timetable manager with a tool to aid his job: scheduling classes for students and teachers at VIA.

In order to achieve this, a Java application with a graphical user interface was developed. This application is capable of retrieving and processing information about the current semester from comma-separated files, thus enabling the scheduling of sessions by the timetable manager according to real information. The sessions booked through the program are saved in files after every change so that no changes are lost. The product also includes a responsive website that can display any timetable that has been generated inside of the Java application.

The outcome of the project was positive: both the desired application and website were developed with satisfactory functionality and pleasing visuals. Although not all of the initially set requirements were met, the highest-priority ones were accomplished, meaning that the program fulfils its purpose without any major problems.

1 Introduction

VIA's timetable manager is currently unable to satisfactorily produce timetables for his institution. This is due to the lack of a tool that can automatically factor everything that needs to be taken into consideration when scheduling sessions for a large university such as VIA. These considerations are mainly the overlapping of timeframes and the vacancy of classrooms, because a student or teacher cannot attend two sessions at once, and two sessions cannot be held in the same classroom at the same time.

The method presently adopted by the scheduler is lengthy and prone to error: before the semester starts, the Head of Department provides him with a list of every course, student, class, and teacher. With that information, he makes use of spreadsheets to manually create a timetable for each class. After the manager has produced the first draft, a "test week" is conducted and the timetables are uploaded to a website where they can be accessed by students and teachers. Not only is this procedure extremely time-consuming, it is also unreasonable for one person to manually check for the availability of classrooms whenever they schedule a session.

Moreover, in higher educational institutions, classes are often cancelled for unpredictable reasons. This leaves a vacant classroom and a gap in students and teachers' schedules. The lack of a system that properly displays this information to whoever is scheduling the classes may lead to teachers not knowing that there is an available room and giving up a class due to the inaccessibility of this information.

A possible application for the manager to use would be Google Calendars, which is easy to sync with multiple devices. However, it lacks a feature that is imperative when booking classes for VIA. The University's new Campus has a shortage of classrooms due to the high number of students currently enrolled, and therefore it is difficult for a timetable manager to know which classrooms are available to book for a certain class. (What Can You Do with Calendar? - Google Workspace Learning Center, 2021)

In summary, the existing problem is the inefficiency of the time schedule manager in producing and providing well-organized timetables to students and teachers.

With this in mind, it is reasonable to infer the need for a tool that can aid the scheduler by automatically checking for timeframe and classroom overlaps, and also graphically display a timetable for easy access for students and teachers, making it so that there is no need to manually create them through spreadsheets.

This tool will:

- Enable the scheduler's work to be less time-consuming and intellectually demanding.
- Prevent VIA's timetables to have overlaps by mistake.
- Enable the reduced number of existing classrooms in the current campus to be used to their full potential.

Delimitation-wise, the tool referred to in this report does not support timetables for teachers, does not feature a login feature, and does not take students with credits into consideration. This means that the only timetables that will be generated are for whole classes and never for individual students.

In order for the tool to meet its purpose, a few mandatory functionalities must be implemented. These will be analyzed in the following chapters.

2 Analysis

Having fully understood the problem, it was possible to infer what the requirements for the desired tool were. They are as follows:

Requirements

CRITICAL PRIORITY

1. As a timetable manager, I want to schedule a session so that teachers and students know when to get together for a session.
2. As a timetable manager, I want students and teachers to be assigned to the courses that they learn or teach so that these appear on their timetables.
3. As a timetable manager, I want to book classrooms for sessions so that students and teachers have a place to get together.
4. As a timetable manager, I want to be able to cancel sessions so that there are more classrooms available for other sessions.
5. As a timetable manager, the access to the planning tool should be limited to me so that I am the only one able to modify schedules.
6. As a timetable manager, I want to see only classrooms which are available when scheduling a class so that there are no overlaps.
7. As a student, I want to see my timetable so that I know when and where I have sessions.

HIGH PRIORITY

8. As a timetable manager, I want to reschedule sessions so that unforeseen events (illness, personal problems, events) do not get in the way of the intended number of lessons for each course.
9. As a timetable manager, I want the changes made to the schedules to be immediately available for students and teachers to see, so that they are always up to date with their timetables.
10. As a timetable manager, I want to remove students from a class so that I can assign them to another class if they have requested a switch.
11. As a timetable manager, I want to add students to a class so that they can have access to a new timetable if they have requested to switch classes.
12. As a timetable manager, I want to be able to assign two teachers to the same course when the course requires it.
13. As a timetable manager, I want to remove students from a course so that it will not appear in their timetable.
14. I want to be able to edit enrollment in a course so that I can add individual students to specific courses.

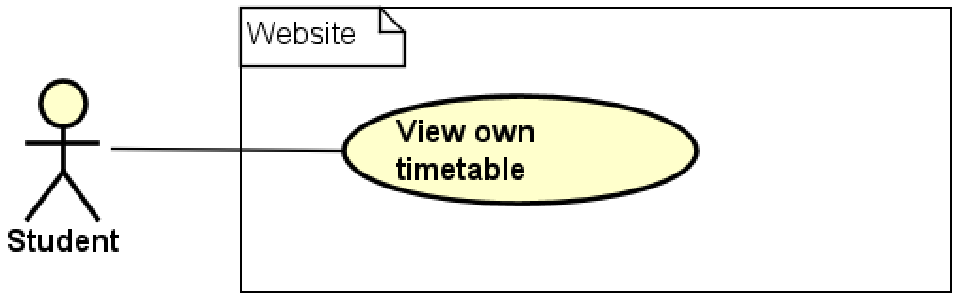
LOW PRIORITY

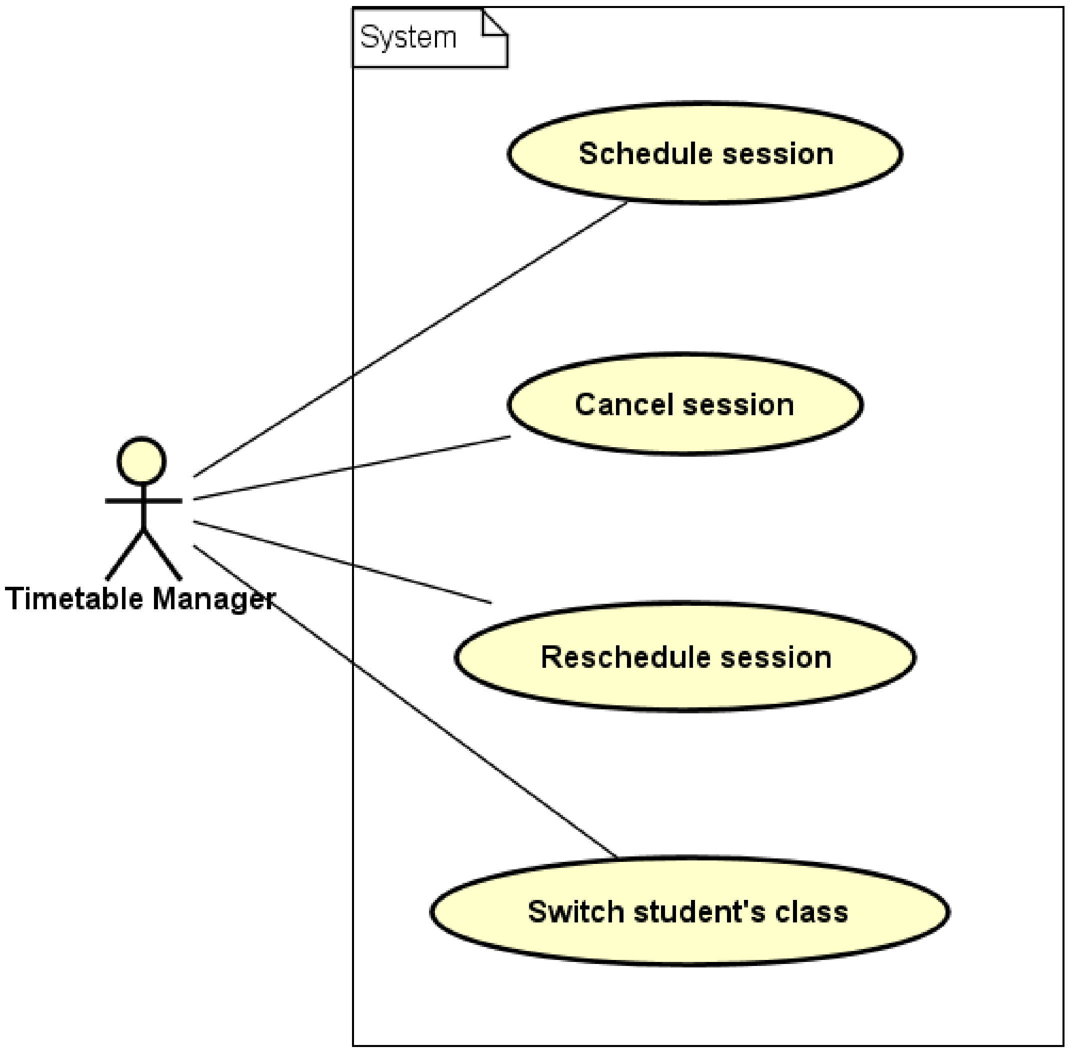
15. As a timetable manager, I want to be able to double the capacity of a room and merge two classes if it has a foldable wall.
16. As a user of this system, I want to see my schedule on a weekly format, so that I can see the sessions for each day.
17. As a user of this system, I want to be warned when sudden changes appear in the schedules.

NON-FUNCTIONAL REQUIREMENTS:

18. As a timetable manager, when assigning a student with credits to a course from another semester, I want the system to check if a timetable without overlaps is available for this student.

Given these requirements, the following use cases were formulated:





What follows is a table illustrating the relationship between the requirements and the use cases. The numbers in bold represent critical requirements.

Use cases	Covered requirements
Schedule session	**1**, **3**, **6**, 9
Reschedule session	8, 9
Cancel session	**4**, 9
Switch student's class	10, 11
View own timetable	**7**, 16, 17
No use case in particular	**2**, **5**

When it is stated that two requirements were met but not by any use case in particular, it is because:

- **Requirement number 2** states that students and teachers should be automatically assigned to the courses that they learn or teach. This is fulfilled not through a use case, but rather by the system itself as soon as it runs, so long as it is given the necessary information inside of a file.

- **Requirement number 5** states that the access to the planning tool should be limited to the timetable manager. This is achieved not through any of the use cases, but rather through the fact that the only person who can make changes to the timetables is the one in possession of the program, and that is intended to be the scheduler.

The use case that will serve as an example throughout this report will be the one that meets the most requirements: **Schedule session**. This use case is the most crucial because the other ones cannot be initiated without it happening first.

Here follows its use case description:

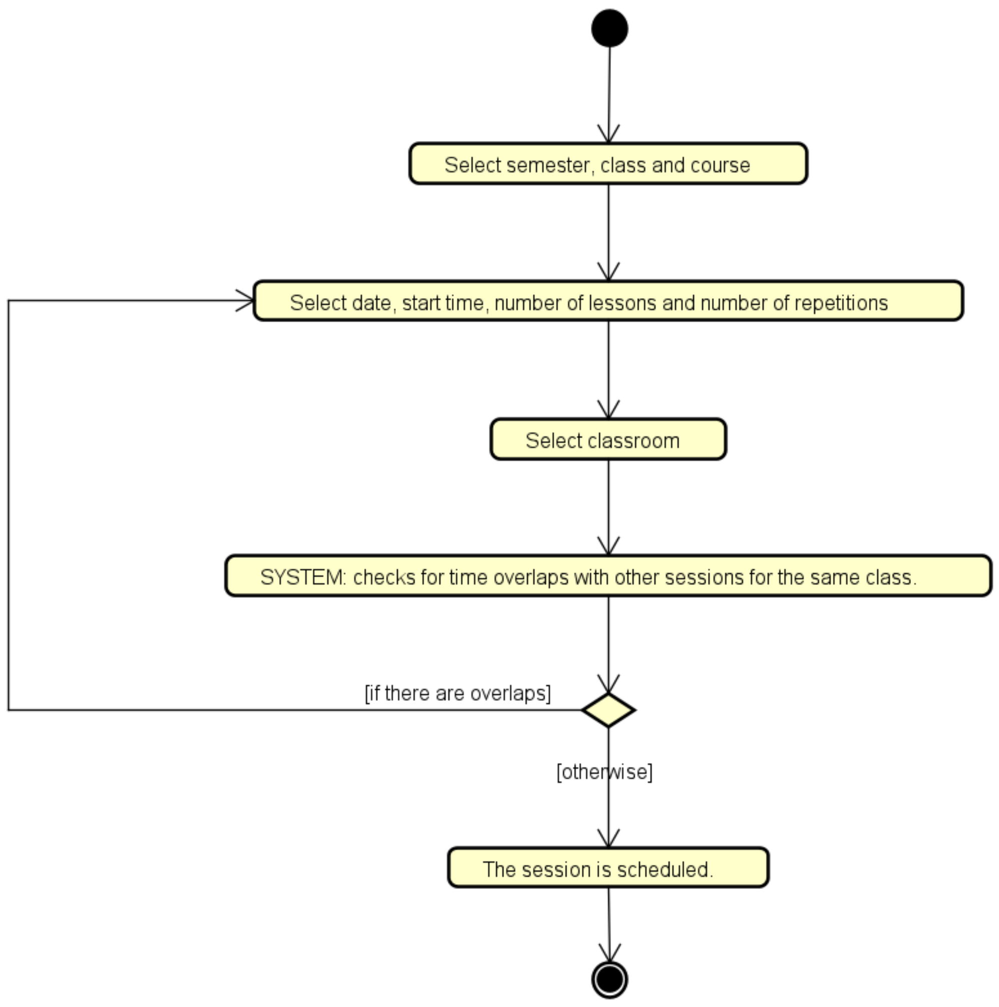
Use case	Schedule session
Summary	Schedule a session for a certain timeframe and book a classroom for that timeframe.
Actor	Timetable Manager
Precondition	The courses, students and classrooms must be registered in the system, given the information from the head of department.
Postcondition	The timetable is updated for the students affected by the change. The classroom associated with the session is marked as booked for that timeframe.
Base Sequence	1. Select semester, class, course. 2. Select date, start time, number of lessons and number of repetitions (how many copies of the session will be created for future weeks). 3. Select classroom. 4. SYSTEM: checks for time overlaps with other sessions for the same class. If there are time overlaps, go back to 2. 5. The session is scheduled.
Note	After changes have been made, the schedules are updated and whoever accesses the website will see the new changes. The requirements met by this Use Case are 1, 3, 6, 9

The reason why it the base sequence is not very lengthy and why there are very few checks for the validity of user input is because, for example, when prompted to select a semester, class or course, the only choices available are the ones retrieved from the Head of Department file, meaning that they will all be valid.

As soon as a session is scheduled, all of the changes are saved.

The rest of the use case descriptions can be found in APPENDIX 1B - ANALYSIS DOCUMENT.

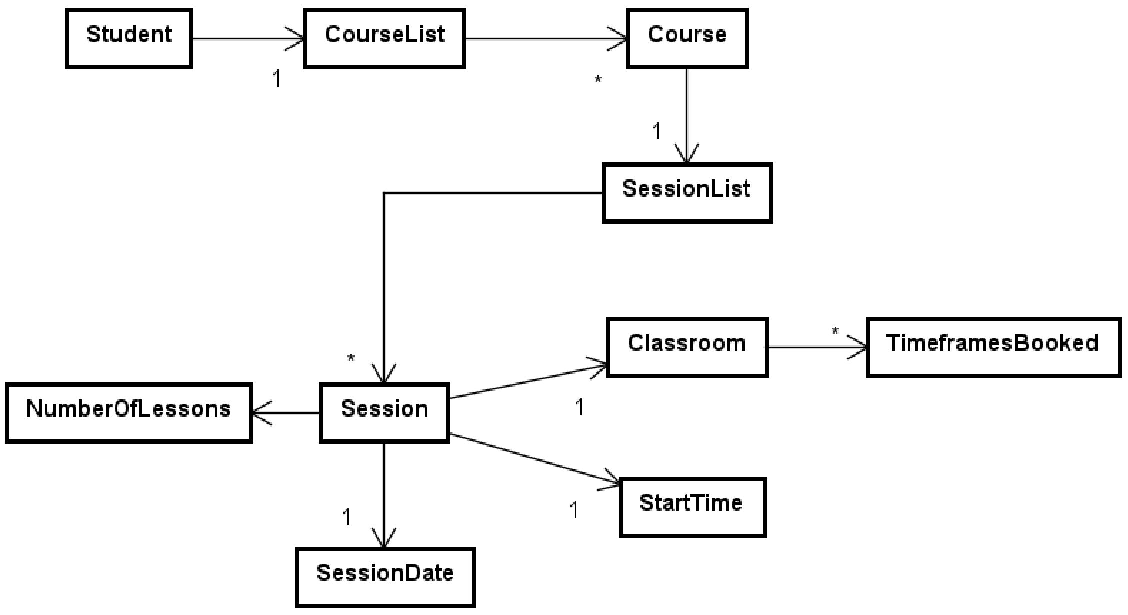
The activity diagram for this use case is as follows:


Select semester, class and course
Select date, start time, number of lessons and number of repetitions
Select classroom
SYSTEM: checks for time overlaps with other sessions for the same class.
[if there are overlaps]
[otherwise]
The session is scheduled.

The reason why a classroom can only be selected after a date, start time and number of lessons have been picked is because before prompting the user with classroom options, the system checks which classrooms are available for a certain timeframe. Therefore, the timeframe needs to be provided beforehand.

The rest of the activity diagrams can be found in APPENDIX 1B - ANALYSIS DOCUMENT.

Below can be found the domain model for this system:



When it is stated that a Student has a Course List, it means that they will be assigned to the Course List corresponding to a class. For example:

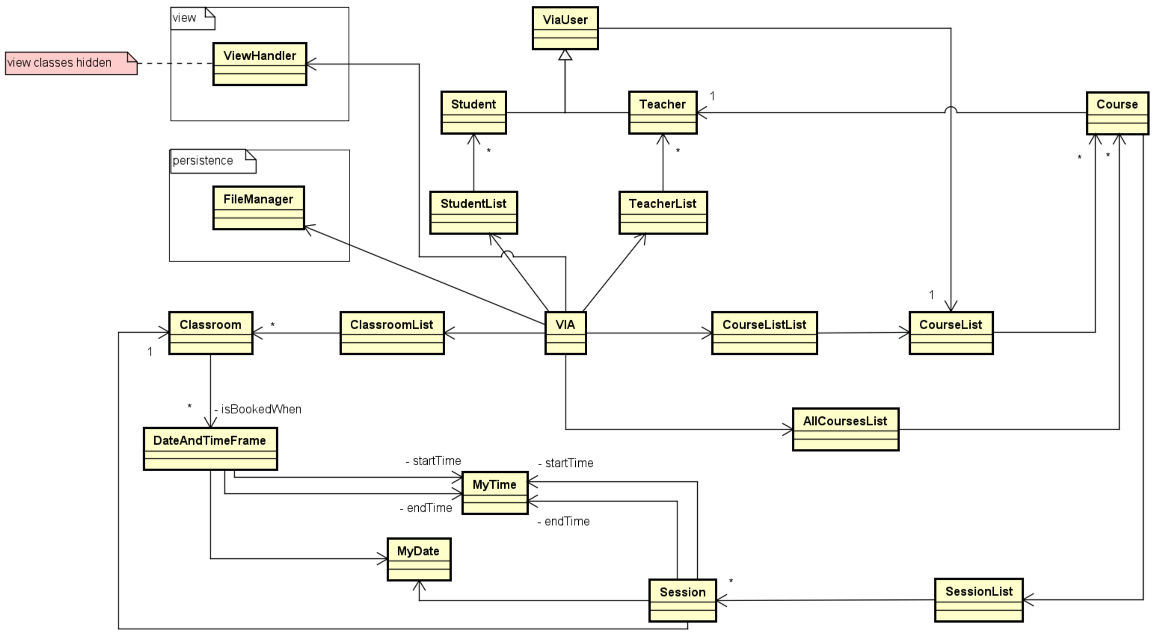
If Student 123456 is assigned to the Course List corresponding to Class **1X**, he will have the courses SDJ**1X**, DMA**1X**, RWD**1X** and SEP**1X** in his Course List.

Each Course, in turn, has a Session List. This way, whenever a timetable for a student is fetched, it will look through the courses in the student's Course List, fetch each Course's sessions and arrange them in a timetable.

The Session, in turn, is the center piece of the Domain model. It contains a Session Date, Start Time and Number of Lessons – these variables define a timeframe for the Session, which enables a check for overlaps inside of the system. Then, each session has a classroom, which has a list of Timeframes Booked. The existence of this list is also crucial: it will make it possible for the availability of a classroom in a certain timeframe to be checked and displayed whenever appropriate.

3 Design

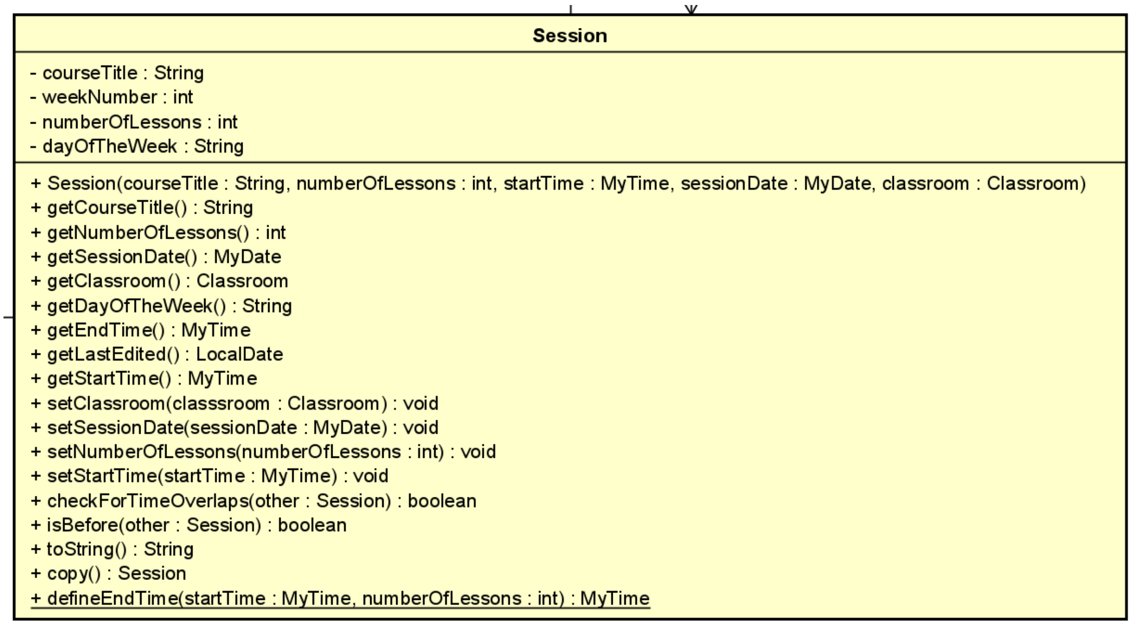
After analyzing the problem and its desired use cases, it was possible for a Class Diagram to be designed.



The central piece of the Class Diagram, VIA, which can also be referred to as Model Manager, has an instance variable of a variety of container classes. Inside it is stored all of the information for the current semester: all Classrooms, Students, Teachers, Courses and Course Lists (grouped by Classes (1X,1Y,2X…). The existence of a Model Manager enables the possibility to implement methods that retrieve information from anywhere inside of the program.

A full class diagram can be found in APPENDIX 2B – CLASS DIAGRAM.

A class that is worth diving into detail regarding the Schedule Session use case is the `Session` class. Its class diagram is in the following page.


Session
- courseTitle : String
- weekNumber : int
- numberOfLessons : int
- dayOfTheWeek : String
+ Session(courseTitle : String, numberOfLessons : int, startTime : MyTime, sessionDate : MyDate, classroom : Classroom)
+ getCourseTitle() : String
+ getNumberOfLessons() : int
+ getSessionDate() : MyDate
+ getClassroom() : Classroom
+ getDayOfTheWeek() : String
+ getEndTime() : MyTime
+ getLastEdited() : LocalDate
+ getStartTime() : MyTime
+ setClassroom(classsroom : Classroom) : void
+ setSessionDate(sessionDate : MyDate) : void
+ setNumberOfLessons(numberOfLessons : int) : void
+ setStartTime(startTime : MyTime) : void
+ checkForTimeOverlaps(other : Session) : boolean
+ isBefore(other : Session) : boolean
+ toString() : String
+ copy() : Session
+ defineEndTime(startTime : MyTime, numberOfLessons : int) : MyTime

Although it is not shown in this picture, `Session` also has:

- Two instance variables of type `MyTime` (startTime and endTime);
- One instance variable of type `Classroom` (classroom);
- Two instance variables of type `MyDate` (sessionDate and lastEdited);

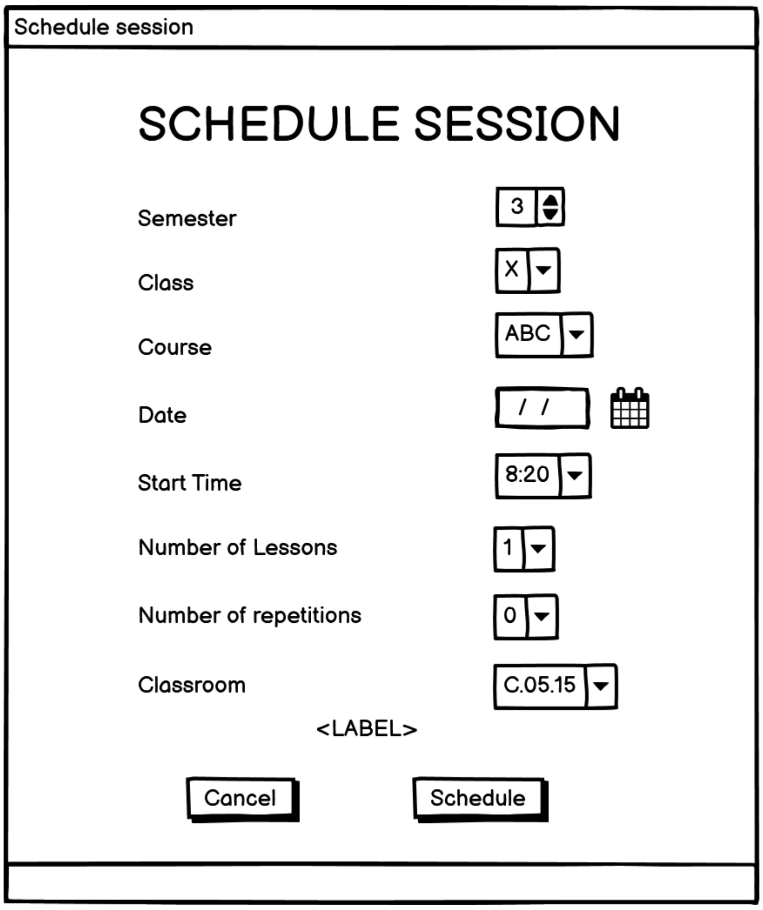
This is a class with many instance variables due to the fact that it requires storing a lot of different information. There is no method inside of it that directly regards scheduling, rescheduling or cancelling, but this is because those methods are, instead, implemented inside of the model manager class, VIA. The reason why is that VIA has access to every class in the diagram, meaning it can add a `Session` to a `SessionList`, for example, while a `Session` cannot schedule itself.

One method worth noting in the `Session` class, though, is “`checkForTimeOverlaps`”. This method is relevant to the use case “Schedule Session” in a sense that two sessions cannot be scheduled for the same class within the same time period. So, what the method does is compare the start and end time of both sessions and see if they overlap. If they overlap, it will return true, and so the system will know not to allow the scheduling of these two sessions for the same class.

It could also be relevant to talk about “`defineEndTime`”. As is observable, the constructor for Session does not take “endTime” as an argument, but rather an int numberOfLessons and a `MyTime` variable called startTime. What the constructor does in order to initialize the “endTime” variable is call `defineEndTime`, which will use both “startTime” and “numberOfLessons” to determine the appropriate end time according to the default start and end times at VIA.

Moreover, another instance variable worth noticing is “`weekNumber`”. This variable also does not appear in the constructor as an argument, but is fetched through a static method inside of the `MyDate` class, called “`getWeekNumber`” for a certain date. The need for this instance variable arose from the necessity to display timetables in a weekly manner inside of the website. With the sessions having a “week number” variable, it would then be possible to fetch this information using JavaScript and then navigate from week to week inside of the website without overlaps or conflicts.

The GUI design for the window related to the use case at hand is as follows:

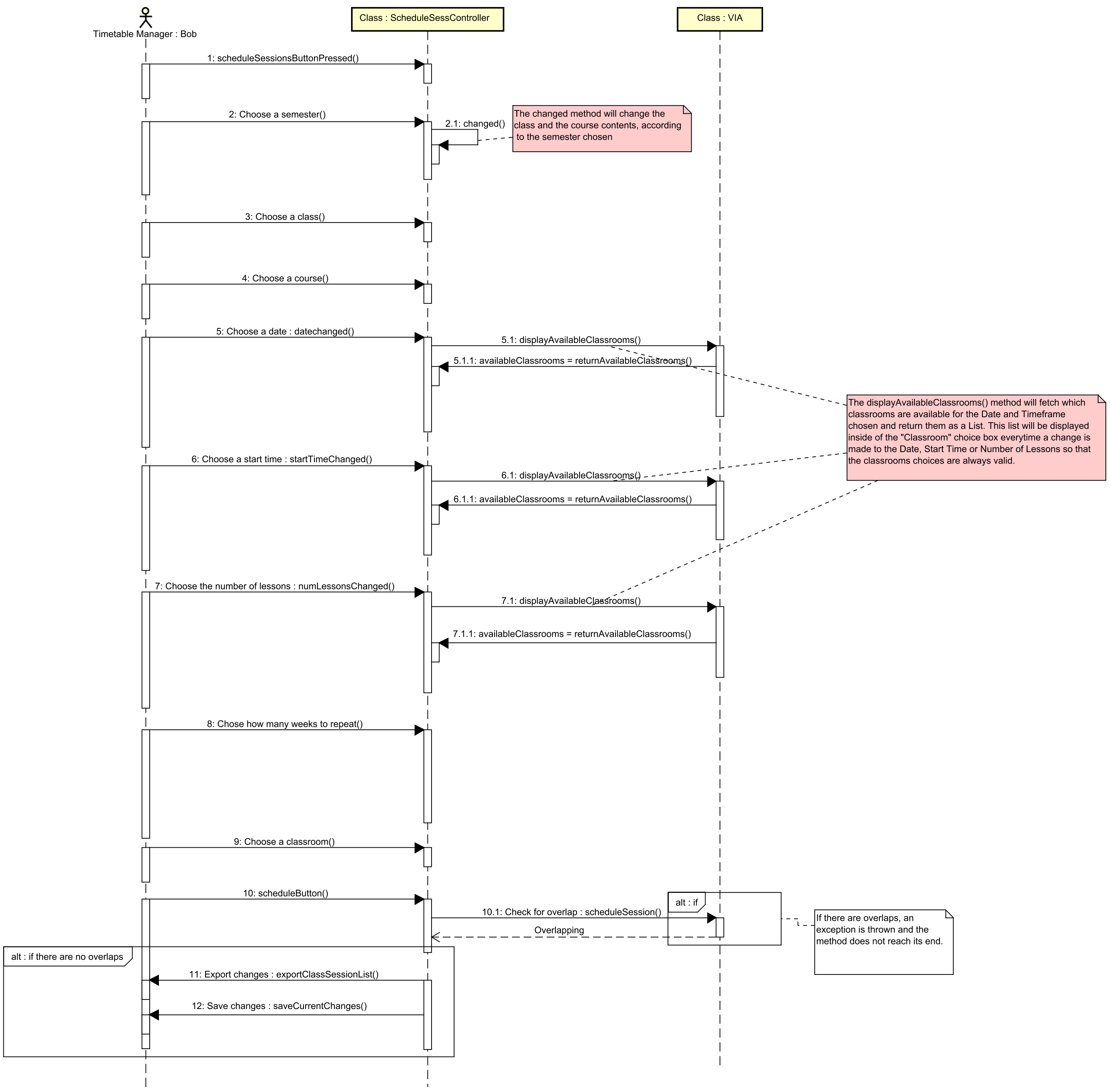


In summary, it starts by requesting the information for a class (semester and class), and afterwards, all of the arguments requested by a `Session` object constructor. This is because, in practice, when scheduling a session, a `Session` object will be created inside of a `SessionList` corresponding to a `Course` of a certain class (e.g., `SessionList` of SEP1Y).

The label at the bottom of the window is used to display errors.

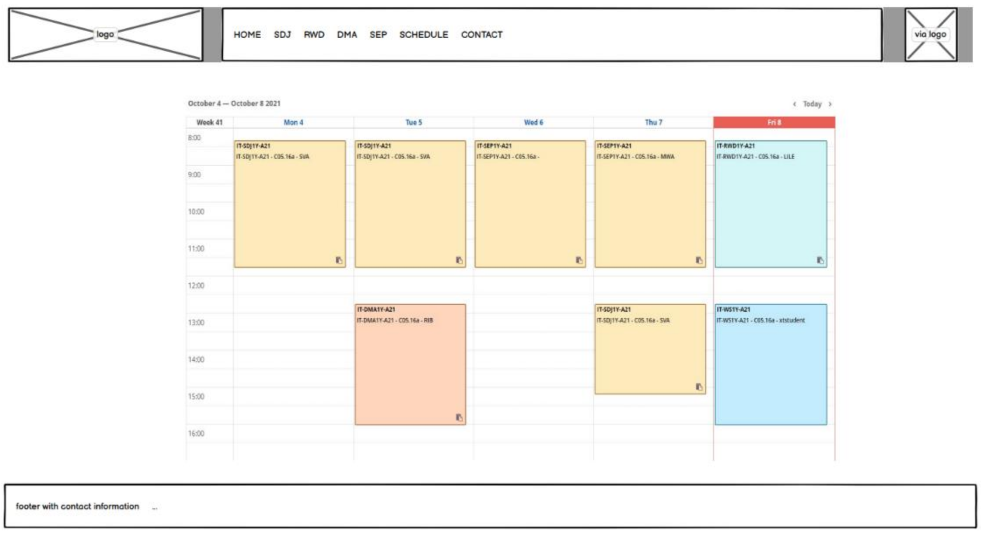
The rest of the Balsamiq Wireframes designs can be found in APPENDIX FOLDER 3 – BALSAMIQ.

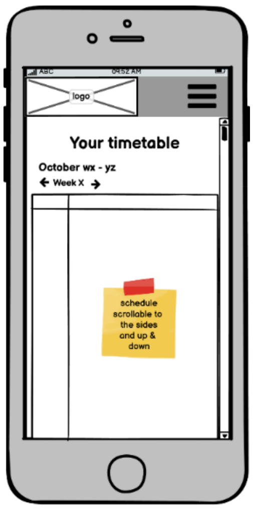
What follows is a sequence diagram for Scheduling a Session. To put it briefly, the Timetable Manager fills out fields with information about the Session at hand, the system checks for overlaps and, if everything is in order, a Session is scheduled.


Timetable Manager : Bob
Class : ScheduleSessController
Class : VIA
1: scheduleSessionsButtonPressed()
2: Choose a semester()
2.1: changed()
The changed method will change the class and the course contents, according to the semester chosen
3: Choose a class()
4: Choose a course()
5: Choose a date : datechanged()
5.1: displayAvailableClassrooms()
5.1.1: availableClassrooms = returnAvailableClassrooms()
The displayAvailableClassrooms() method will fetch which classrooms are available for the Date and Timeframe chosen and return them as a List. This list will be displayed inside of the "Classroom" choice box everytime a change is made to the Date, Start Time or Number of Lessons so that the classrooms choices are always valid.
6: Choose a start time : startTimeChanged()
6.1: displayAvailableClassrooms()
6.1.1: availableClassrooms = returnAvailableClassrooms()
7: Choose the number of lessons : numLessonsChanged()
7.1: displayAvailableClassrooms()
7.1.1: availableClassrooms = returnAvailableClassrooms()
8: Chose how many weeks to repeat()
9: Choose a classroom()
10: scheduleButton()
alt : if
10.1: Check for overlap : scheduleSession()
Overlapping
If there are overlaps, an exception is thrown and the method does not reach its end.
alt : if there are no overlaps
11: Export changes : exportClassSessionList()
12: Save changes : saveCurrentChanges()

The website also required designing before being implemented. As for the part of it that is most relevant to this project, the Schedule page, two layouts were set out for different screen sizes: Desktop and Mobile. On Desktop, the timetable shows in full extension, while on the phone, it is scrollable to the sides due to the small size of the screen. This does not hinder navigability, however, as the timetable cells maintain a considerable size as opposed to shrinking unreasonably.

The designs produced on Balsamiq Wireframes are below. They can be found in more detail in APPENDIX FOLDER 3 – BALSAMIQ.

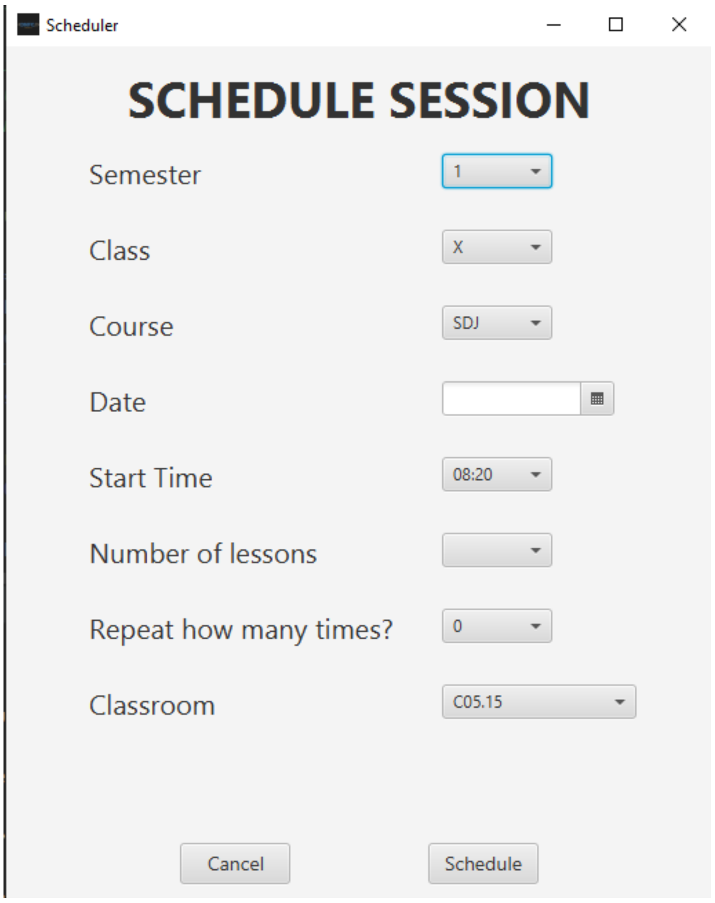

logo
HOME
SDJ
RWD
DMA
SEP
SCHEDULE
CONTACT
via logo
October 4 — October 8 2021
‹ Today ›
Week 41
Mon 4
Tue 5
Wed 6
Thu 7
Fri 8
8:00
9:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
IT-SDJ1Y-A21
IT-SDJ1Y-A21 - C05.16a - SVA
IT-SDJ1Y-A21
IT-SDJ1Y-A21 - C05.16a - SVA
IT-SEP1Y-A21
IT-SEP1Y-A21 - C05.16a -
IT-SEP1Y-A21
IT-SEP1Y-A21 - C05.16a - MWA
IT-RWD1Y-A21
IT-RWD1Y-A21 - C05.16a - LILE
IT-DMA1Y-A21
IT-DMA1Y-A21 - C05.16a - RIB
IT-SDJ1Y-A21
IT-SDJ1Y-A21 - C05.16a - SVA
IT-WS1Y-A21
IT-WS1Y-A21 - C05.16a - xtstudent
footer with contact information ...



4 Implementation

Implementation was conducted in a structured manner:

In the first place, the GUI windows were created in Scene Builder according to the design initially proposed. The window for "Schedule Session" turned out to be as follows:

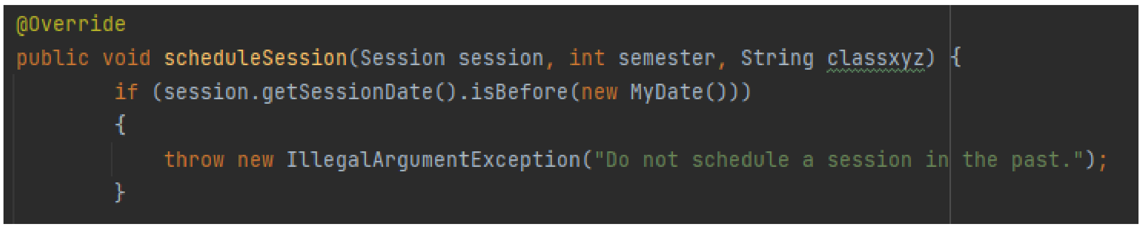

Scheduler

SCHEDULE SESSION

Semester	1
Class	X
Course	SDJ
Date	
Start Time	08:20
Number of lessons	
Repeat how many times?	0
Classroom	C05.15

This window makes use of a multiplicity of Choice Boxes that display and retrieve the information relevant for scheduling a session, one Date Picker and two Buttons: one for going back and one for confirming the Scheduling.

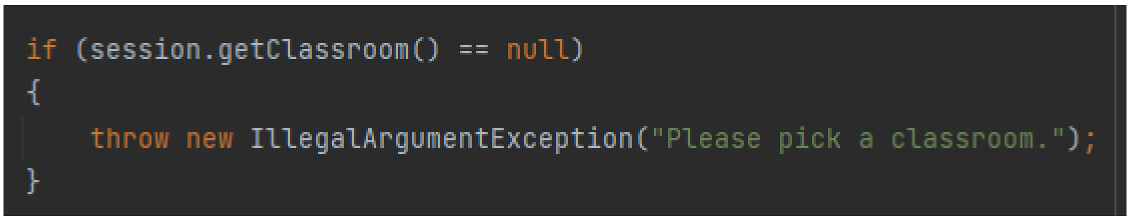
The next step was to implement the Model classes. This was done inside of IntelliJ. They were all implemented according to the Class Diagram shown in the Design chapter. A method that could be worth discussing in detail could be "`scheduleSession`" inside of the VIA class.



```
@Override
public void scheduleSession(Session session, int semester, String classxyz) {
        if (session.getSessionDate().isBefore(new MyDate()))
        {
            throw new IllegalArgumentException("Do not schedule a session in the past.");
        }
```

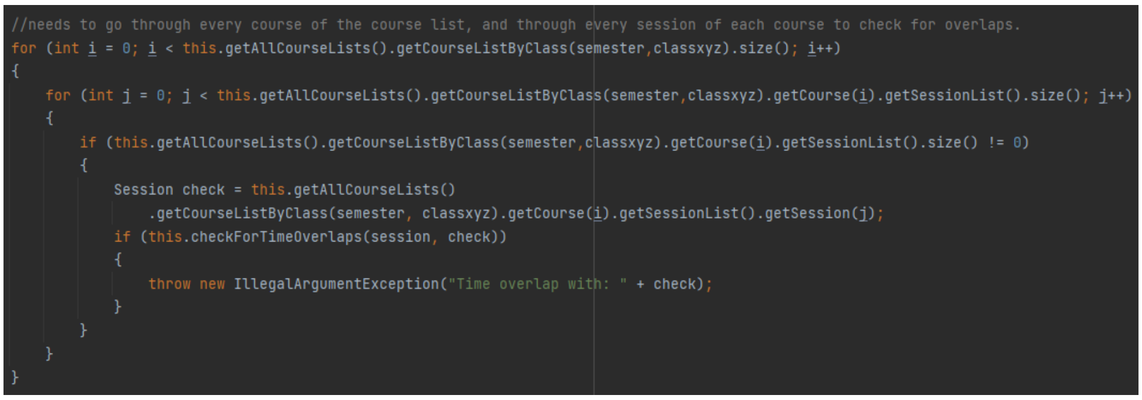
The method takes as an argument a Session object, an int for the semester and a String for the class (x,y,z...). This data is all fetched inside of the Schedule Session window, as mentioned before.

The first thing it does is check whether the date trying to be scheduled is in the past. If so, the program will throw an exception warning the user not to do this.



```
if (session.getClassroom() == null)
{
    throw new IllegalArgumentException("Please pick a classroom.");
}
```

Next, it checks whether the Classroom field is empty. Whenever the Start Time, Number of Lessons or Date are changed, the Classroom field is updated and set to empty for the user to pick a classroom once again. This is why this check was necessary.

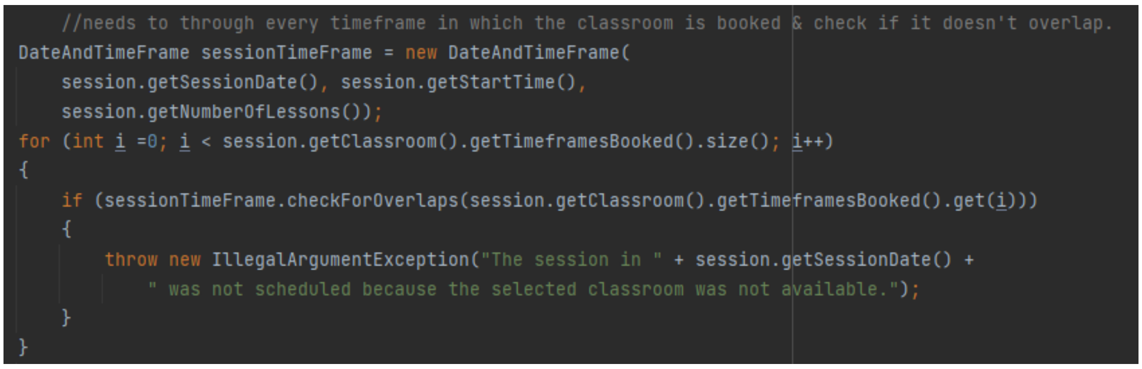


```
//needs to go through every course of the course list, and through every session of each course to check for overlaps.
for (int i = 0; i < this.getAllCourseLists().getCourseListByClass(semester,classxyz).size(); i++)
{
    for (int j = 0; j < this.getAllCourseLists().getCourseListByClass(semester,classxyz).getCourse(i).getSessionList().size(); j++)
    {
        if (this.getAllCourseLists().getCourseListByClass(semester,classxyz).getCourse(i).getSessionList().size() != 0)
        {
            Session check = this.getAllCourseLists()
                .getCourseListByClass(semester, classxyz).getCourse(i).getSessionList().getSession(j);
            if (this.checkForTimeOverlaps(session, check))
            {
                throw new IllegalArgumentException("Time overlap with: " + check);
            }
        }
    }
}
```

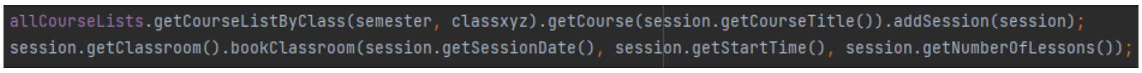
What follows is a lengthier check: in order for there not to be time overlaps, the method goes through every Session featured in that class's `CourseList` to check if any of them overlap with the Session trying to be booked. If they do overlap, an exception will be thrown telling the user which session is overlapping with the one at hand.

The **time complexity** of this method is O(n*m), being "n" the number of `Courses` inside of the class's `CourseList`, and "m" the amount of `Sessions` inside of each `Course's` `SessionList`. The reason why it is O(n*m) is because there are two nested for loops. There are two nested "if" statements inside of the nested "for" loop, but they only represent one time unit each, so each of them will happen, at most, (n*m) times. This does not affect the final time complexity.

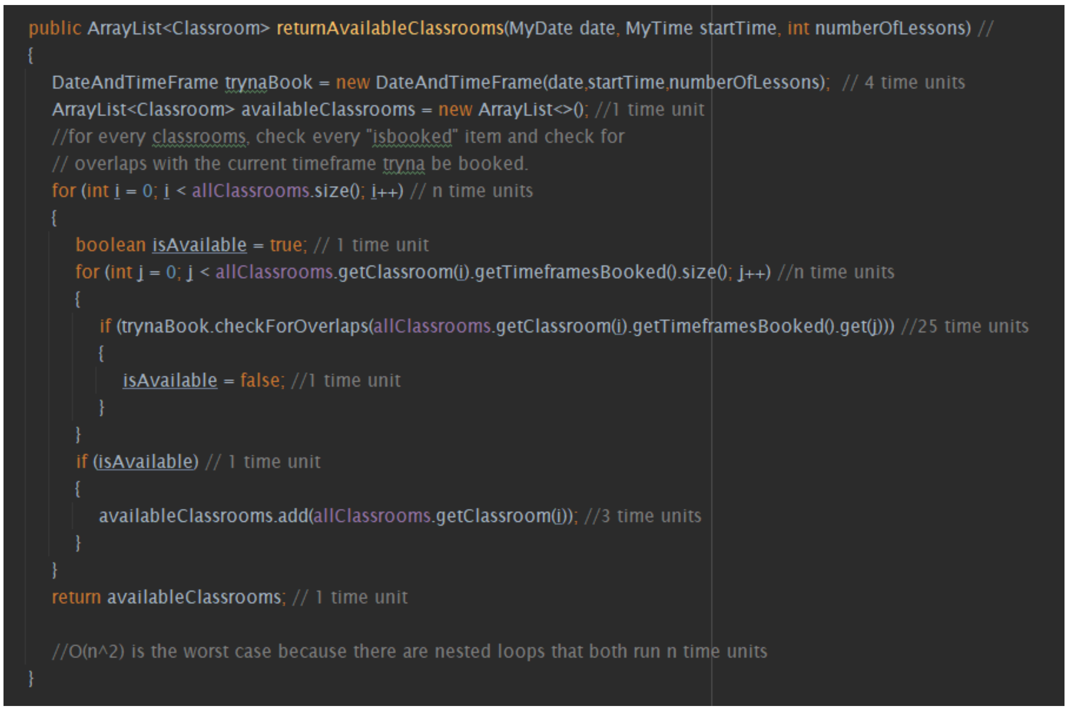
Lastly, the method goes through all of the “timeframes booked” for the picked classroom to see if it is available for the timeframe in which the session is trying to be booked. If it is unavailable, an exception will be thrown.


```
    //needs to through every timeframe in which the classroom is booked & check if it doesn't overlap.
DateAndTimeFrame sessionTimeFrame = new DateAndTimeFrame(
    session.getSessionDate(), session.getStartTime(),
    session.getNumberOfLessons());
for (int i =0; i < session.getClassroom().getTimeframesBooked().size(); i++)
{
    if (sessionTimeFrame.checkForOverlaps(session.getClassroom().getTimeframesBooked().get(i)))
    {
        throw new IllegalArgumentException("The session in " + session.getSessionDate() +
            " was not scheduled because the selected classroom was not available.");
    }
}
```

And finally, the session is booked: it is added to the chosen `Course’s SessionList` and the according `Classroom` is booked for the timeframe picked.


```
allCourseLists.getCourseListByClass(semester, classxyz).getCourse(session.getCourseTitle()).addSession(session);
session.getClassroom().bookClassroom(session.getSessionDate(), session.getStartTime(), session.getNumberOfLessons());
```

Another method whose time complexity could be analysed is "`returnAvailableClassrooms`", due to the fact that this is a method that is ran every time anything related to time is changed inside of the "Schedule Session" window.



```java
public ArrayList<Classroom> returnAvailableClassrooms(MyDate date, MyTime startTime, int numberOfLessons) //
{
    DateAndTimeFrame trynaBook = new DateAndTimeFrame(date,startTime,numberOfLessons);  // 4 time units
    ArrayList<Classroom> availableClassrooms = new ArrayList<>(); //1 time unit
    //for every classrooms, check every "isbooked" item and check for
    // overlaps with the current timeframe tryna be booked.
    for (int i = 0; i < allClassrooms.size(); i++) // n time units
    {
        boolean isAvailable = true; // 1 time unit
        for (int j = 0; j < allClassrooms.getClassroom(i).getTimeframesBooked().size(); j++) //n time units
        {
            if (trynaBook.checkForOverlaps(allClassrooms.getClassroom(i).getTimeframesBooked().get(j))) //25 time units
            {
                isAvailable = false; //1 time unit
            }
        }
        if (isAvailable) // 1 time unit
        {
            availableClassrooms.add(allClassrooms.getClassroom(i)); //3 time units
        }
    }
    return availableClassrooms; // 1 time unit

    //O(n^2) is the worst case because there are nested loops that both run n time units
}
```

This method returns the available classrooms and it has big O(n^2) time complexity because of the nested loops.

First, the local variable `trynaBook` is created and initialized, which takes 4 time units.

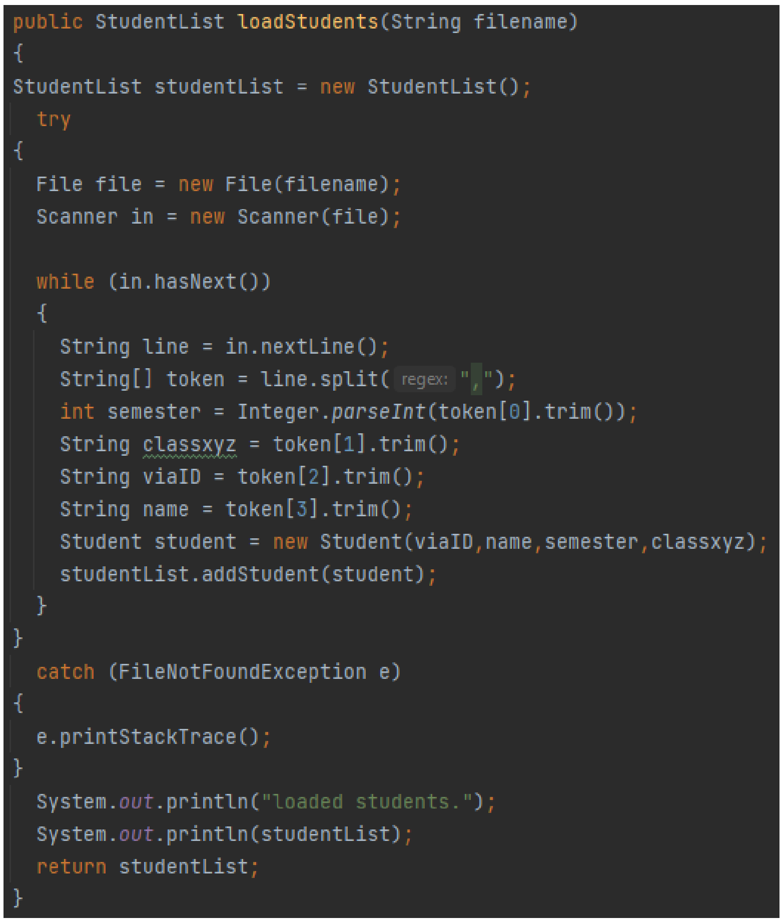
Afterwards, `ArrayList` is initialized, which takes 1 time unit.

The first "for" loop will run `allClassrooms.size()` times, will be simplified as being said to run "n" times. It will first initialize `isAvailable` with the value true, and then it is going to enter another for loop which will run `allClassrooms.getClassroom(i).getTimeFramesBooked().size()` times. In order to simplify as much as possible, and because the for-loop unit times are changing every increment of the first for loop, this loop is also going to be said to run for "n" time units.

The second loop will run an "if" statement **n** times, each time generating 25 time units in the background, because this is the runtime of `checkForOverlaps`. If `isAvailable` remains as it is, it is going to be added to the local variable `availableClassrooms`. If not, then it will not be added.

At the end, the method is going to return the `availableClassrooms` ArrayList.

It could also be relevant to talk about how the information is retrieved from the external .txt files and incorporated inside of the program.



```
public StudentList loadStudents(String filename)
{
StudentList studentList = new StudentList();
  try
{
  File file = new File(filename);
  Scanner in = new Scanner(file);

  while (in.hasNext())
  {
    String line = in.nextLine();
    String[] token = line.split( regex: ",");
    int semester = Integer.parseInt(token[0].trim());
    String classxyz = token[1].trim();
    String viaID = token[2].trim();
    String name = token[3].trim();
    Student student = new Student(viaID,name,semester,classxyz);
    studentList.addStudent(student);
  }
}
  catch (FileNotFoundException e)
{
  e.printStackTrace();
}
  System.out.println("loaded students.");
  System.out.println(studentList);
  return studentList;
}
```

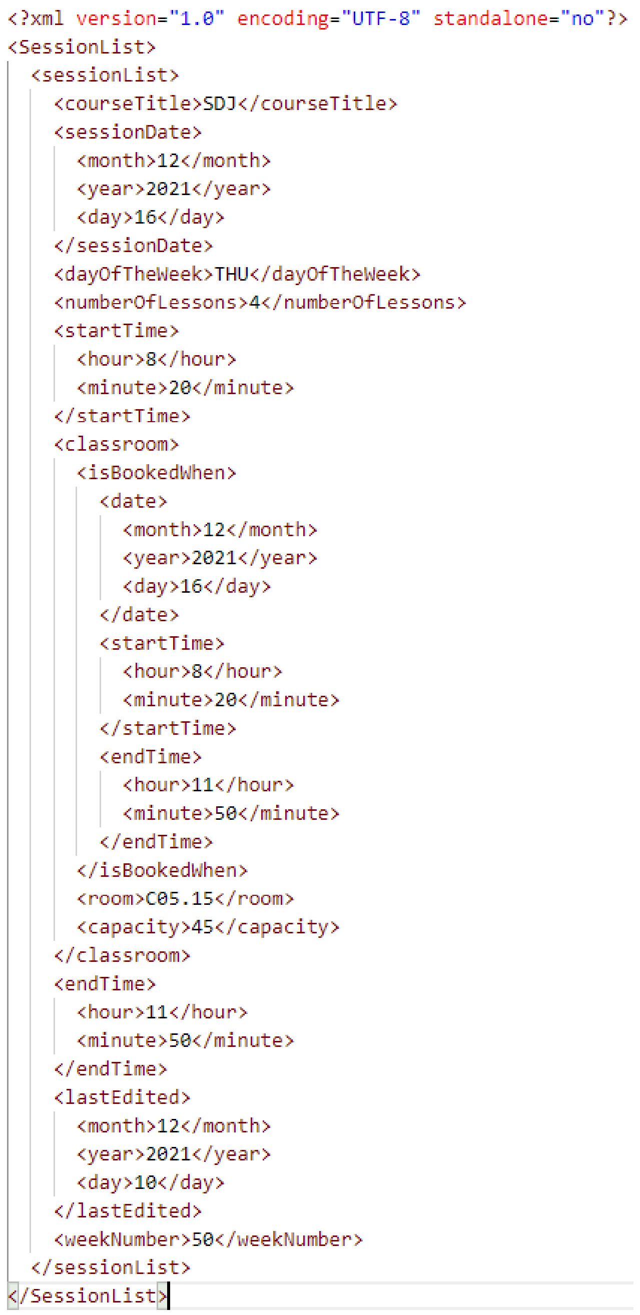
On the picture above is a method called "`loadStudents`". This method exists inside of the class "`FileManager`", part of the "`persistence`" package, which is responsible for the exchange of information between the program and the .txt external files.

What this method does is go through every line of the .txt file, parse information from there, create a `Student` object for each line and add it to a `StudentList` that is returned at the end of the method. This method is useful for the constructor of the model manager, VIA. Inside of the constructor, VIA's `StudentList` is initialized to the one that is returned by this method from the file.

The implementation about how the website is able to display information about schedules created inside of the program is also worth analysing in detail.

Inside of the Java application, there is a function that exports the current Sessions scheduled for a class to an XML file. This XML file is tailored to be read and displayed in the website by JavaScript.

An example of one of these XML files could look something like this:



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SessionList>
  <sessionList>
    <courseTitle>SDJ</courseTitle>
    <sessionDate>
      <month>12</month>
      <year>2021</year>
      <day>16</day>
    </sessionDate>
    <dayOfTheWeek>THU</dayOfTheWeek>
    <numberOfLessons>4</numberOfLessons>
    <startTime>
      <hour>8</hour>
      <minute>20</minute>
    </startTime>
    <classroom>
      <isBookedWhen>
        <date>
          <month>12</month>
          <year>2021</year>
          <day>16</day>
        </date>
        <startTime>
          <hour>8</hour>
          <minute>20</minute>
        </startTime>
        <endTime>
          <hour>11</hour>
          <minute>50</minute>
        </endTime>
      </isBookedWhen>
      <room>C05.15</room>
      <capacity>45</capacity>
    </classroom>
    <endTime>
      <hour>11</hour>
      <minute>50</minute>
    </endTime>
    <lastEdited>
      <month>12</month>
      <year>2021</year>
      <day>10</day>
    </lastEdited>
    <weekNumber>50</weekNumber>
  </sessionList>
</SessionList>
```

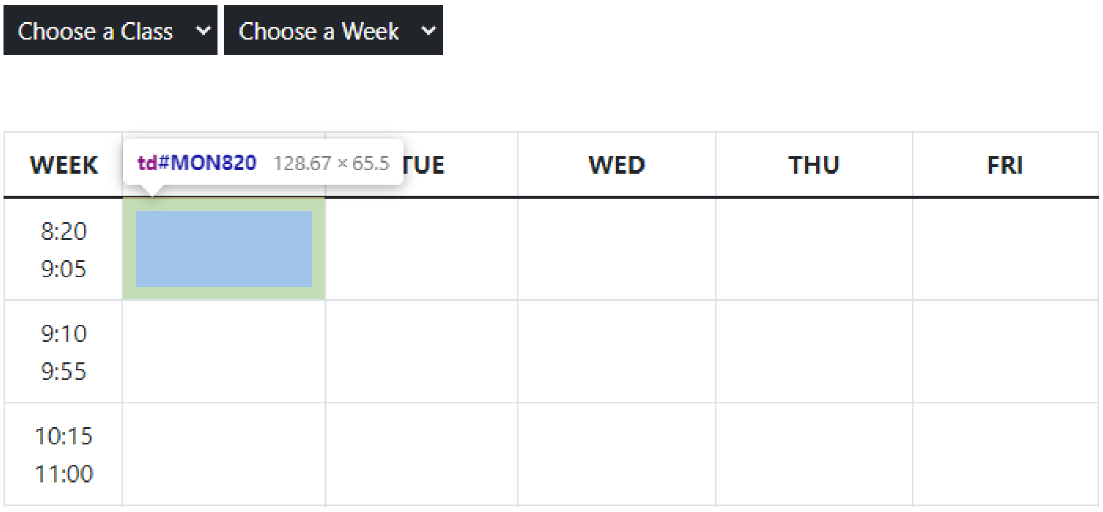
The XML file in the previous page contains only one session. Inside of each session is stored a lot of information:

- Session Date
- Start Time
- Day of the Week
- Number of Lessons
- Classroom
- End Time
- Last Edited Date
- Week Number

All of these variables will be read by JavaScript and displayed inside of the timetable.

In order for them to be displayed accurately, all of the table cells of the timetable inside of the website were given a unique ID.

For example, the table cell that is the lesson on Monday at 8:20 was given the ID "#MON820".

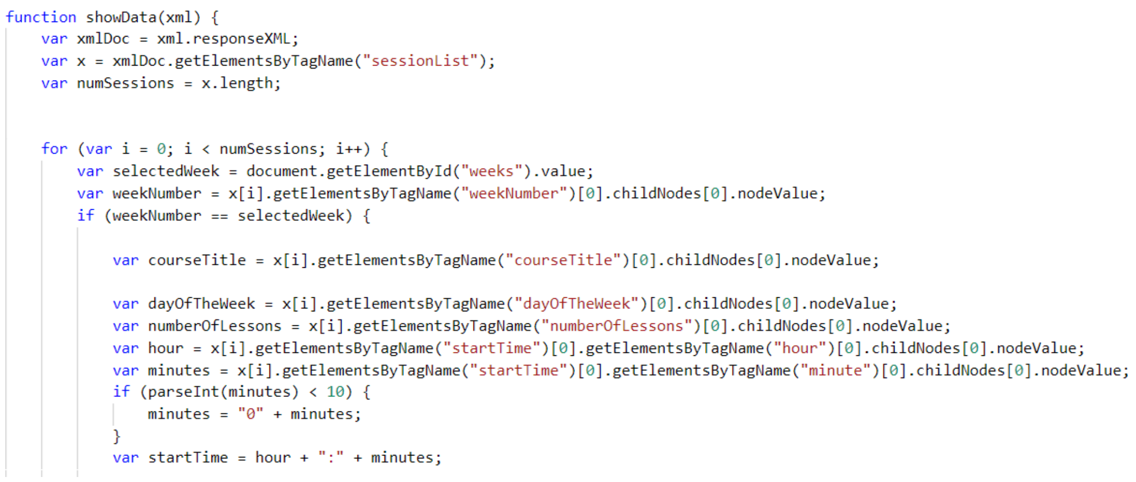


WEEK		TUE	WED	THU	FRI
8:20 9:05					
9:10 9:55					
10:15 11:00					

With these ID's, it was possible to infer where each session should be placed inside of the table. However, some sessions may have the length of more than one lesson. For this to be displayed in the timetable, the solution found was to:

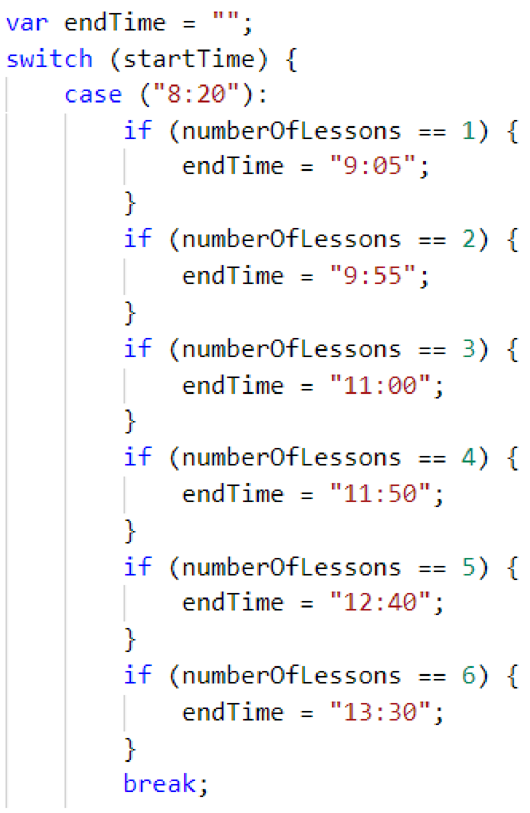
- Set the "rowspan" attribute of the table cell to the number of lessons. This way, the cell would be taller the longer the session.
- Delete the table cells that were "overridden" by the longer lesson. This was necessary because otherwise, they would be shifted to the right side and the timetable would become inaccurate.

The following screenshot portrays how the information was fetched for each session:



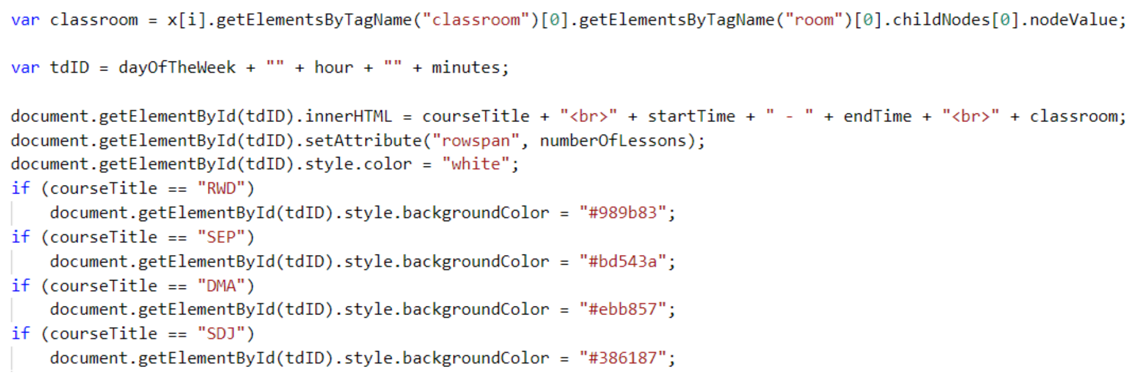
```
function showData(xml) {
    var xmlDoc = xml.responseXML;
    var x = xmlDoc.getElementsByTagName("sessionList");
    var numSessions = x.length;


    for (var i = 0; i < numSessions; i++) {
        var selectedWeek = document.getElementById("weeks").value;
        var weekNumber = x[i].getElementsByTagName("weekNumber")[0].childNodes[0].nodeValue;
        if (weekNumber == selectedWeek) {

            var courseTitle = x[i].getElementsByTagName("courseTitle")[0].childNodes[0].nodeValue;

            var dayOfTheWeek = x[i].getElementsByTagName("dayOfTheWeek")[0].childNodes[0].nodeValue;
            var numberOfLessons = x[i].getElementsByTagName("numberOfLessons")[0].childNodes[0].nodeValue;
            var hour = x[i].getElementsByTagName("startTime")[0].getElementsByTagName("hour")[0].childNodes[0].nodeValue;
            var minutes = x[i].getElementsByTagName("startTime")[0].getElementsByTagName("minute")[0].childNodes[0].nodeValue;
            if (parseInt(minutes) < 10) {
                minutes = "0" + minutes;
            }
            var startTime = hour + ":" + minutes;
```

The end time was fetched through a “switch” statement due to the fact that JavaScript was not accurately fetching its value.



```
var endTime = "";
switch (startTime) {
    case ("8:20"):
        if (numberOfLessons == 1) {
            endTime = "9:05";
        }
        if (numberOfLessons == 2) {
            endTime = "9:55";
        }
        if (numberOfLessons == 3) {
            endTime = "11:00";
        }
        if (numberOfLessons == 4) {
            endTime = "11:50";
        }
        if (numberOfLessons == 5) {
            endTime = "12:40";
        }
        if (numberOfLessons == 6) {
            endTime = "13:30";
        }
        break;
```

The screenshot below portrays how the information came to be inside of the correct table cell.



```
var classroom = x[i].getElementsByTagName("classroom")[0].getElementsByTagName("room")[0].childNodes[0].nodeValue;

var tdID = dayOfTheWeek + "" + hour + "" + minutes;

document.getElementById(tdID).innerHTML = courseTitle + "<br>" + startTime + " - " + endTime + "<br>" + classroom;
document.getElementById(tdID).setAttribute("rowspan", numberOfLessons);
document.getElementById(tdID).style.color = "white";
if (courseTitle == "RWD")
    document.getElementById(tdID).style.backgroundColor = "#989b83";
if (courseTitle == "SEP")
    document.getElementById(tdID).style.backgroundColor = "#bd543a";
if (courseTitle == "DMA")
    document.getElementById(tdID).style.backgroundColor = "#ebb857";
if (courseTitle == "SDJ")
    document.getElementById(tdID).style.backgroundColor = "#386187";
```

For example, if the session starts on Monday at 8:20:

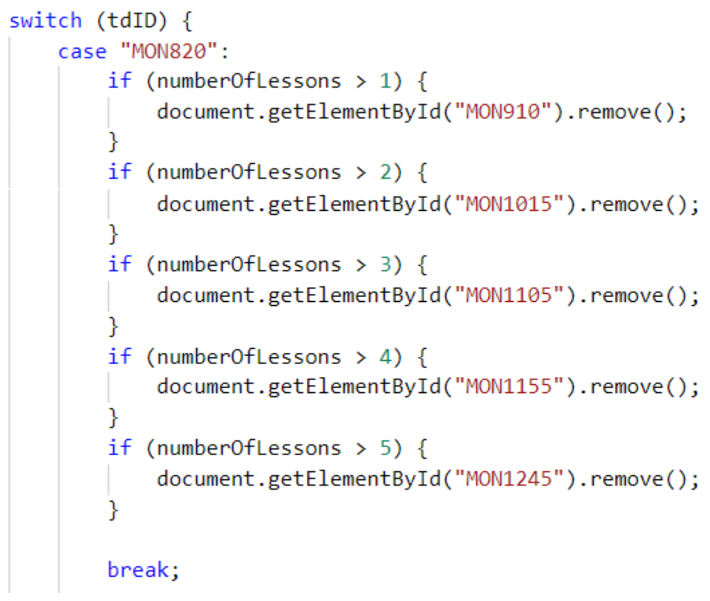
A variable called “tdID” will be created with the day of the week (“MON”), the starting hour (“8”) and the starting minutes (“20”).

Therefore, its final value will be “MON820”.

It was mentioned before that each table cell has its unique ID, and they are also in this format. Therefore, “tdID” is the method’s way of finding in which cell to insert the information for the current session through its unique ID.

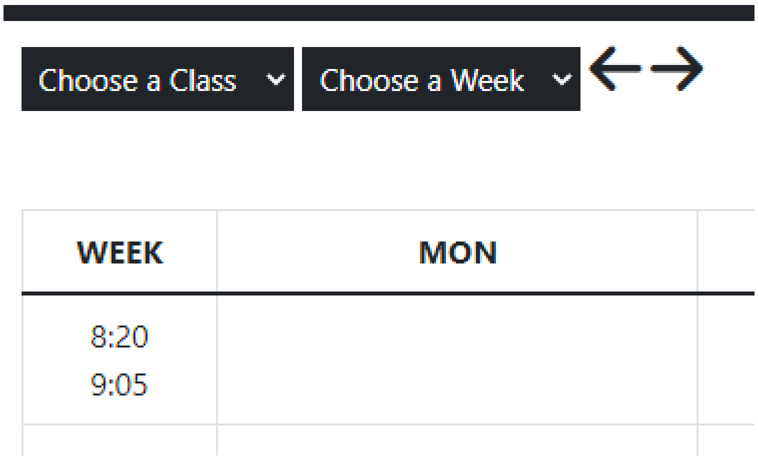
After that, the information is put inside of the table cell, its text color is set to white and the background color is set differently depending on the course. The only courses taken into account were the ones included in the “sample” .txt files.

Lastly, the removing of the “extra” table cells:



```
switch (tdID) {
    case "MON820":
        if (numberOfLessons > 1) {
            document.getElementById("MON910").remove();
        }
        if (numberOfLessons > 2) {
            document.getElementById("MON1015").remove();
        }
        if (numberOfLessons > 3) {
            document.getElementById("MON1105").remove();
        }
        if (numberOfLessons > 4) {
            document.getElementById("MON1155").remove();
        }
        if (numberOfLessons > 5) {
            document.getElementById("MON1245").remove();
        }

        break;
```

The Schedule page of the website also includes two dropdown menus and two buttons designed for a better navigability of the timetables.

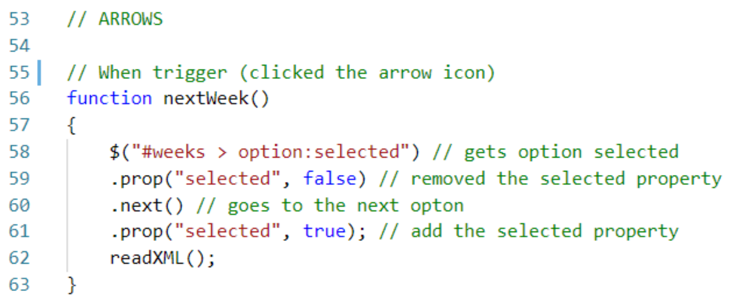


The dropdown menu "Choose a Class" gets its options from an XML file that is also generated inside of the application and tailored for this purpose: it solely contains the existing classes: e.g., 1X , 1Y, 1Z… When a class is picked, the Session XML file that is read is the one corresponding to it (e.g., "1X.xml").

```
//this function filters the information of the xml
function readClasses(xml) {
    var xmlDoc = xml.responseXML; // gets the response
    var Classs = $(xmlDoc).find("Class"); // finds the "Class" Object
    var classxyz = $("#classxyz"); // Finds where to add that information, in this case it's a select box
    Classs.each(function () { // For every found "Class" will add an option for the select
        var option = $("<option />");
        option.html($(this).find("ClassName").text());
        classxyz.append(option);
    });
}
```

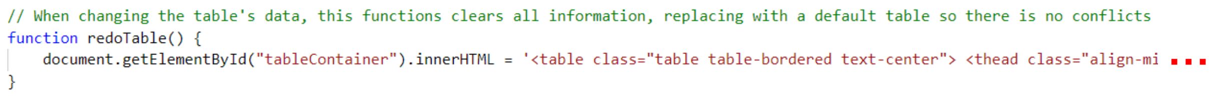
Additionally, the dropdown menu "Choose a Week" has the purpose of preventing the overlapping of Sessions from different weeks. For example, if there were two sessions: one on the 7/12/21 (TUE) and 22/12/21 (WED) and this option did not exist, it would seem in the timetable that they existed in the same week, when in reality, they are two weeks apart. In order to achieve this, the variable "`weekNumber`" inside of `Session` was used. So, when the selected week corresponds to a Session's week number, it will be displayed inside of the timetable. This way, it is possible to navigate through weeks without overlaps.

The two arrow buttons exist also to facilitate navigation: instead of having to open the dropdown menu every time, the user can just go to the previous or next week with one click.



```
// ARROWS

// When trigger (clicked the arrow icon)
function nextWeek()
{
    $("#weeks > option:selected") // gets option selected
    .prop("selected", false) // removed the selected property
    .next() // goes to the next opton
    .prop("selected", true); // add the selected property
    readXML();
}
```

Lastly, another need that arose was to “redo” the table every time a new option was selected. This is because some schedules delete table cells, as mentioned before, so when another schedule was selected, it might not have all of the cells to its disposal and could appear inaccurately. In order to prevent this, a function “redoTable()” was created, which clears the content of the container table and recreates all of the table cells.



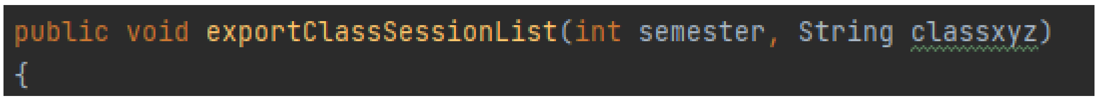
```
// When changing the table's data, this functions clears all information, replacing with a default table so there is no conflicts
function redoTable() {
    document.getElementById("tableContainer").innerHTML = '<table class="table table-bordered text-center"> <thead class="align-mi ...
}
```

To wrap up the Implementation chapter, it might be interesting to talk about how a high priority requirement related to both the “Schedule Session” use case and the website implementation:

9. As a timetable manager, I want the changes made to the schedules to be immediately available for students and teachers to see, so that they are always up to date with their timetables.

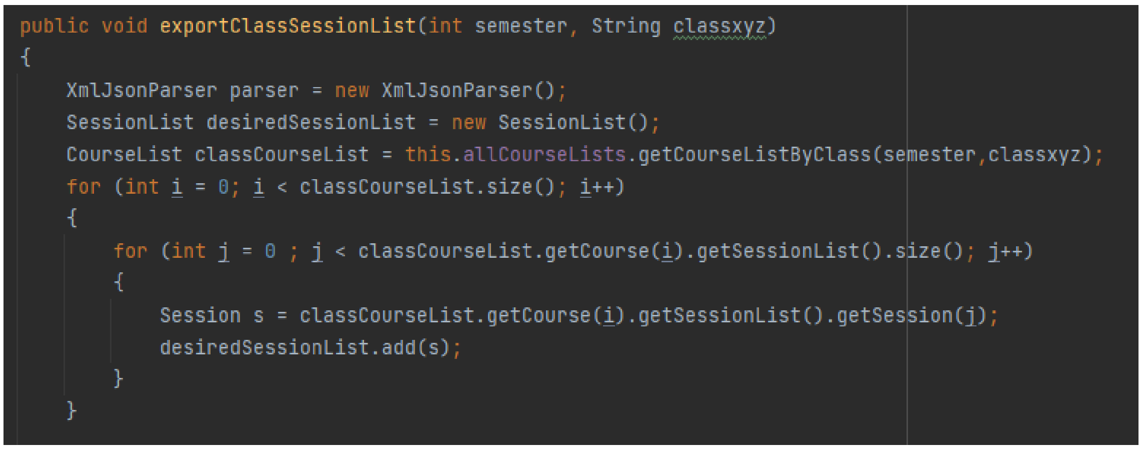
It was achieved in the following way:

Every time any changes are made to the schedule, this method is called:



```
public void exportClassSessionList(int semester, String classxyz)
{
```

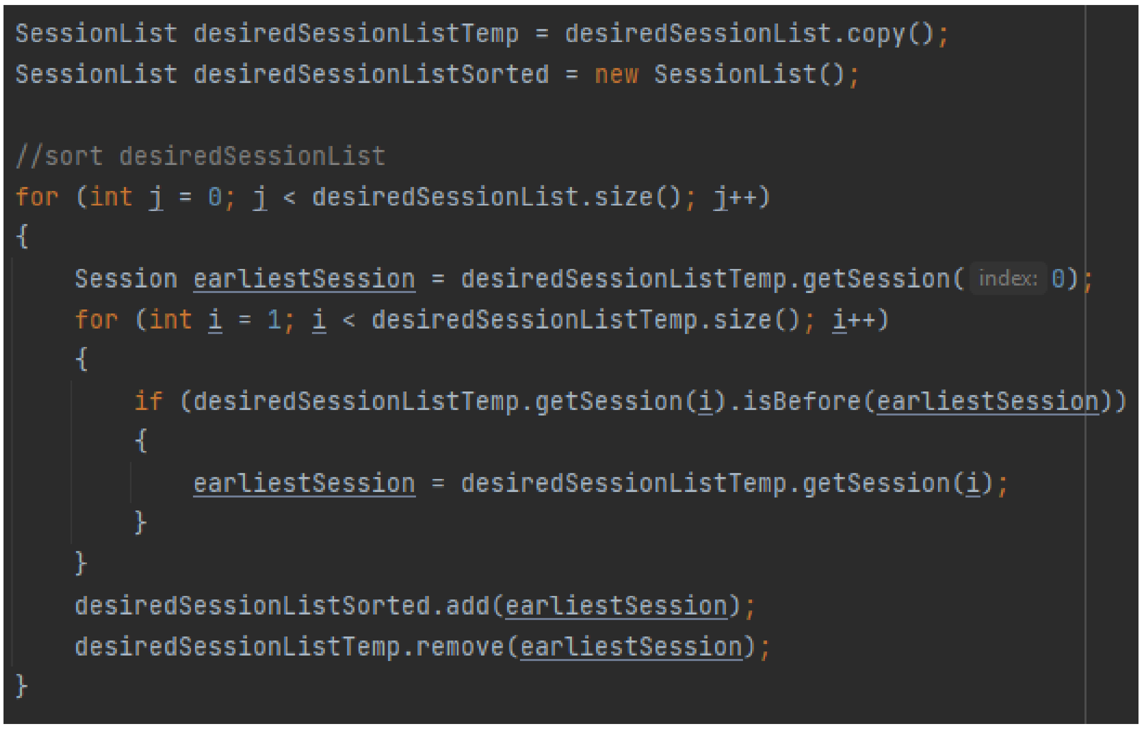
What this method does is access the website directory and export an XML file there. The XML filename depends on the semester and class at hand: for example, if a session was scheduled for class 1X, the XML file edited would be “1X.xml”. The direct consequence of this is that the next time the website is opened, the changes made are immediately available, and the requirement at hand is met. Code snippets for the method will be presented on the following page:



```java
public void exportClassSessionList(int semester, String classxyz)
{
    XmlJsonParser parser = new XmlJsonParser();
    SessionList desiredSessionList = new SessionList();
    CourseList classCourseList = this.allCourseLists.getCourseListByClass(semester,classxyz);
    for (int i = 0; i < classCourseList.size(); i++)
    {
        for (int j = 0 ; j < classCourseList.getCourse(i).getSessionList().size(); j++)
        {
            Session s = classCourseList.getCourse(i).getSessionList().getSession(j);
            desiredSessionList.add(s);
        }
    }
```

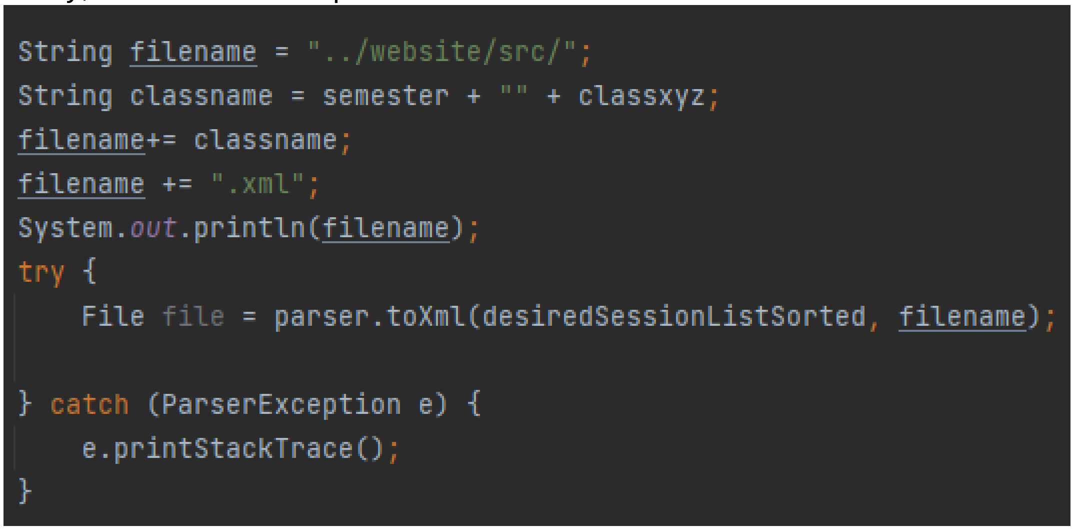
Firstly, all of the Sessions for the desired class are put inside of a "`desiredSessionList`" that will be exported to XML at the end of the method.

Then, they are sorted from earliest to latest and put inside of a sorted version of `desiredSessionList`.



```java
SessionList desiredSessionListTemp = desiredSessionList.copy();
SessionList desiredSessionListSorted = new SessionList();

//sort desiredSessionList
for (int j = 0; j < desiredSessionList.size(); j++)
{
    Session earliestSession = desiredSessionListTemp.getSession( index: 0);
    for (int i = 1; i < desiredSessionListTemp.size(); i++)
    {
        if (desiredSessionListTemp.getSession(i).isBefore(earliestSession))
        {
            earliestSession = desiredSessionListTemp.getSession(i);
        }
    }
    desiredSessionListSorted.add(earliestSession);
    desiredSessionListTemp.remove(earliestSession);
}
```

Lastly, the sorted list is exported to an XML file inside of the source folder for the website.



```
String filename = "../website/src/";
String classname = semester + "" + classxyz;
filename+= classname;
filename += ".xml";
System.out.println(filename);
try {
    File file = parser.toXml(desiredSessionListSorted, filename);

} catch (ParserException e) {
    e.printStackTrace();
}
```

So if any changes were made to the 1X Session List, whoever opened the website and picked 1X would see the new changes in the timetable.

5 Test

In order for the implemented system to be tested, the approach chosen was to go through each use case, define an expected end result, test it and document whether the desired end result was met or not.

The results can be observed in the table below.

The "Reschedule Session table" is referenced frequently because it is a way to see information about the scheduled sessions.

Use Case	Expected Result	Works as expected?
Schedule Session	The session scheduled appears in the "Reschedule Session" table when correctly searched for, meaning it was successfully created.	Yes
Reschedule Session	The session rescheduled has its information updated according to what was rescheduled. This can be seen in the "Reschedule Session" table.	Yes
Cancel Session	The session cancelled does no longer appear in the "Reschedule Session" table.	Yes
Switch student's class	The student's class is updated to the new one, which can be observed in the "Manage Students" window.	Yes
View own timetable	The sessions scheduled appear in the website when the right class and week are selected.	Yes

6 Results and Discussion

When contemplating the final product, it was possible to determine which initially set requirements were met.

Requirement	Met?	Description
		CRITICAL PRIORITY
1	Yes	As a timetable manager, I want to schedule a session so that teachers and students know when to get together for a session.
2	Yes	As a timetable manager, I want students and teachers to be assigned to the courses that they learn or teach so that these appear on their timetables
3	Yes	As a timetable manager, I want to book classrooms for sessions so that students and teachers have a place to get together.
4	Yes	As a timetable manager, I want to be able to cancel sessions so that there are more classrooms available for other sessions.
5	Yes	As a timetable manager, the access to the planning tool should be limited to me so that I am the only one able to modify schedules.
6	Yes	As a timetable manager, I want to see only which classrooms are available when scheduling a class so that there are no overlaps
7	Yes	As a student, I want to see my timetable so that I know when and where I have sessions.
		HIGH PRIORITY
8	Yes	As a timetable manager, I want to reschedule sessions so that unforeseen events (illness, personal problems, events) do not get in the way of the intended number of lessons for each course.
9	Yes	As a timetable manager, I want the changes made to the schedules to be immediately available for students and teachers to see, so that they are always up to date with their timetables.
10	Yes	As a timetable manager, I want to remove students from a class so that I can assign them to another class if they have requested a switch.
11	Yes	As a timetable manager, I want to add students to a class so that they can have access to a new timetable if they have requested to switch classes.
12	No	As a timetable manager, I want to be able to assign two teachers to the same course when the course requires it.

13	No	As a timetable manager, I want to remove students from a course so that it will not appear in their timetable.
14	No	I want to be able to edit enrollment in a course so that I can add individual students to specific courses.
LOW PRIORITY		
15	No	As a timetable manager, I want to be able to double the capacity of a room and merge two classes if it has a foldable wall.
16	Yes	As a user of this system, I want to see my schedule on a weekly format, so that I can see the sessions for each day.
17	Yes	As a user of this system, I want to be able to filter only the schedules that correspond to my class.
18	No	As a user of this system, I want to be warned when sudden changes appear in the schedules.
NON-FUNCTIONAL REQUIREMENTS		
19	No	As a timetable manager, when assigning a student with credits to a course from another semester, I want the system to check if a timetable without overlaps is available for this student.

Although not all initially set requirements were met, the end result was still satisfactory. By this, it is meant that every critical requirement was fulfilled, and also that the final product is a good foundation for a possible future version of the program in which all other requirements can be implemented.

Looking back at it critically, a few aspects could have been implemented in more convenient ways:

- The `Session` class could have had a `DateAndTimeFrame` instance variable instead of two `MyTime` and one `MyDate` variable to define the timeframe in which it is booked.
- Instead of the `Courses` having a `SessionList`, there could have been a class for each Class (e.g., 1X, 1Y…), and only inside that class would there be a `SessionList`. It was implemented in this non-optimal way because the first plan was for each student to have an individual timetable, in which their `Sessions` would be fetched from the `Courses` to which the `Student` was assigned.

However, given the way the system ended up, `Sessions` could have been booked for directly for each Class instead of for each `Course`.

- The use of files for persistence is far from optimal. Every time any change is made to the system, a new .json file and new .xml are created with all of the information, overriding the previous files, as opposed to simply appending to what was previously there, no matter how little the change.

7 Conclusions

In summary, each step of the process of developing the project at hand build on top of the previous one, eventually leading to a satisfactory end product. It was first necessary to formulate the Problem Description for the client's present situation, extract the requirements for the system during the Analysis, and only then was it possible to become solution-oriented and start designing how to implement these use cases through a Java Application and Website that came to existence during the Implementation phase. Then, in possession of the product, a test phase was conducted, and the results were positive: both application and website functioned as intended, and enough requirements were fulfilled by the project for the solution to be considered viable.

8 Sources and References

Flaticon. 2021. Detailed Flat Circular Icon Style / Flat - 39,741 vector icons available in SVG, EPS, PNG, PSD files and Icon Font.. [online] Available at: <https://www.flaticon.com/authors/detailed-flat-circular/flat> [Accessed 14 December 2021].

Gandy, D., 2021. Font Awesome. [online] Fontawesome.com. Available at: <https://fontawesome.com/> [Accessed 14 December 2021].

Miranda, R., 2021. Javafx Cascading dropdown based on selection. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/20794359/javafx-cascading-dropdown-based-on-selection> [Accessed 14 December 2021].

Stack Overflow. 2021. JavaFX- how to save the selected from ChoiceBox. [online] Available at: <https://stackoverflow.com/questions/31605585/javafx-how-to-save-the-selected-from-choicebox> [Accessed 14 December 2021].

Stack Overflow. 2021. Making a search bar in javafx. [online] Available at: <https://stackoverflow.com/questions/47559491/making-a-search-bar-in-javafx> [Accessed 14 December 2021].

Gaddis, T., 2015. Starting Out with Java: Early Objects. 5th ed. Boston [etc.]: Pearson.

Duckett., J., 2011. HTML & CSS: Design and Build Websites. John Wiley & Sons Incorporated.

Duckett, J., Ruppert, G. and Moore, J., 2014. JavaScript & jQuery.

LaGrone, B., 2013. HTML5 and CSS3 Responsive Web Design Cookbook.

Weiss, M., 2012. Data Structures and Algorithm Analysis in Java. Boston, Mass: Addison-Wesley.

Support.google.com. 2021. What can you do with Calendar? - Google Workspace Learning Center. [online] Available at:

<https://support.google.com/a/users/answer/9302892?hl=en> [Accessed 8 October 2021].

W3schools.com. 2021. W3Schools Online Web Tutorials. [online] Available at: <https://www.w3schools.com/>.

9 Appendices

Appendices can be found inside of the APPENDICES folder inside of the handed-in .zip file.

The ones that were referenced in this report will be highlighted in yellow.

They are structured by folders in the following manner:

APPENDIX FOLDER 1 – DOCUMENTS

- APPENDIX 1A – Project Description
- APPENDIX 1B – Analysis Document
- APPENDIX 1C – Logbook
- APPENDIX 1D – User Guide
- APPENDIX 1E – Installation Guide

APPENDIX FOLDER 2 – ASTAH

- APPENDIX SUBFOLDER 2A – ACTIVITY DIAGRAM
  - APPENDIX 2Aa – Schedule Session Activity Diagram
  - APPENDIX 2Ab – Reschedule Session Activity Diagram
  - APPENDIX 2Ac – Cancel Session Activity Diagram
  - APPENDIX 2Ad – Switch Student's Class Activity Diagram
  - APPENDIX 2Ae – View Own Timetable Activity Diagram
- APPENDIX SUBFOLDER 2B – CLASS DIAGRAM
  - APPENDIX 2Ba – Simplified Class Diagram
  - APPENDIX 2Bb – Complete Class Diagram
- APPENDIX SUBFOLDER 2C – DOMAIN MODEL
  - APPENDIX 2Ca – Domain Model
- APPENDIX SUBFOLDER 2D – USE CASE DIAGRAM
  - APPENDIX 2Da – Use Case Diagram
- APPENDIX SUBFOLDER 2E – SEQUENCE DIAGRAM
  - Appendix 2Ea – Sequence Diagram

APPENDIX FOLDER 3 – BALSAMIQ WIREFRAMES

- APPENDIX 3A – Website for Desktop Design
- APPENDIX 3B – Website for Mobile Design

- APPENDIX 3C – GUI Design

APPENDIX FOLDER 4 – SOURCE CODE

- EXTERNAL JAR FILES
- *SEP1Ygroup6* (IntelliJ Project Folder)
- *website* (Website directory)

APPENDIX FOLDER 5 – JAVADOC

- Open “index.html” to access Javadoc.