**Introduction - MATAS**

Hello and welcome to the presentation of our First Semester Project. We are group 6 of class Y, called Do Not Fail Course, composed of Daniel, Dragos, Laura and Matas. The case at hand is one that was presented to us: Bob, VIA’s timetable manager, did not have a proper tool to do his job. He needed a program that would be able to automatically check for time and classroom overlaps, meaning that no class could have two sessions at the same time, and no classroom could be booked for two sessions at the same time. Before this, his main tool for making timetables was Excel Sheets, meaning that he had to check for all of these overlaps by hand, which is simply unreasonable.

**Demonstration**

In order to aid him, we did exactly as he requested! We developed a program that checks for all of these overlaps and, on top of that, displays the schedules on a responsive website. We’ll show it to you right away:

* In Java application:
  + Schedule a class
  + Demonstrate that a class booked at the same time cannot book the same classroom
  + Demonstrate that the same class cannot book two sessions at the same time
  + Schedule at least three sessions of three different courses
* In website:
  + Demonstrate the scheduled sessions.
* Back to Java:
  + Cancel a session
* Back to website:
  + Show the cancelled session.

**Implementation**

For us to be able to come up with this final product, we vaguely followed a Waterfall methodology, as suggested by our SEP supervisors at the beginning of the semester. With that in mind, we started out by elaborating a Project Description, in which we dove deep into the problem without focusing on the solution: Bob’s current situation, what he was going through and what he was lacking.

After that, we could come up with an Analysis document: from our Project Description, we extracted a list of requirements and use cases in order to start thinking about how to solve the problem.

Then, with those on paper, it was possible to start working on Design:

* Show class diagram, say that it is the “skeleton” for the program
* Mention how there is a central class called VIA with access to a lot of container classes, which contain a lot of information for the current semester
* Balsamiq Wireframes: GUI and how it relates to use cases (each use case has a dedicated part of the GUI)
* Website: most of it was done inside of the Responsive Web Design assignments, except for the JavaScript part that directly relates to the project.

**ACTUAL implementation**

**Java - DRAGOS**

Then, as for the actual implementation, the first thing that we did was actually the GUI, and this is because in order for us to code the methods for the program, we needed something at hand, something real, we needed to see which buttons would be pressed and what information would be provided in each window so that we knew what methods to code and what arguments they could take, because otherwise, if we had done it “in the dark”, we believed that we might have done a lot of work in vain, because it might not have been used in the actual program, so with this, we could have a much more focused approach to coding.

After that, we implemented all of the classes provided in the Design, much like the Class Diagram shown just a while ago – and of course a lot of new ideas came up throughout the implementation phase, which made it so that, for a coherent hand-in, we had to go back and tweak Analysis and Design – but we believe that this only part of the process and that there is no harm in having new ideas on the go.

The main thing that the program needed to do was the check for time and classroom overlaps.

The check for overlaps for the same class was achieved in the following manner: Each class had a CourseList with Courses. Then, each course had a SessionList. What the program does is go through all of that class’s Sessions and check if there is one that overlaps timewise with the one trying to be scheduled. If there are no overlaps, it is scheduled. If not, the program will not allow it.

As for the classrooms, this one was a bit trickier, but consequently more interesting: the classrooms are something that all classes have in common – they belong to everyone. So it might have seemed more difficult to check for overlaps no matter the class. The way to achieve this was for each classroom to have an ArrayList of “times in which it is booked”. Then, when scheduling a session, the program will take the date and timeframe for this session and see which classrooms are not booked at any conflicting times, so: classrooms who do not have any “time booked” object that overlaps with the timeframe at hand.

Another thing worth mentioning is the maintenance of persistence for the program, which was done through files. A lot of files were used in this project. First of all, when the program is opened for the first time, it will look inside its directory to look for the files with information about the current semester: Students, Courses and Teachers. The information within them will be parsed and stored inside of the container classes shown in the class diagram. Then, the user can do whatever he wants inside of the program (schedule, cancel, etc.), and when he closes it, a .json file will be created with all of the information. We did not use XML because XML could not deal with empty or one-element ArrayLists, and there were plenty of these in our program. So, since the external parser that we used worked for both XML and JSON, we just switched into this other file structure out of convenience. Then, with the .json file created, every time the program is open again, it will just retrieve the information from it and go back to where the user left off. So, if Bob wants to start scheduling in the morning, then go have lunch and come back to work, he can do so with ease.

**HTML and JavaScript - DANIEL**

Moreover, about how the website works and how it relates to the program, it so happens that every time you do a change to a class’s schedule, the program will export a list of sessions for that class as an XML, and it does so inside of the website’s directory. This makes it so that whenever you make a choice with the dropdown menus (class and week), the desired information can be retrieved from the appropriate XML file through JavaScript. JavaScript will then grab all of the information for a session, so the start time, end time, course, classroom, etc., and put it inside of the desired table cell. And there is also one important element: “number of lessons” for a session. With this field, JavaScript makes use of the table cell “rowspan” attribute to make it so that a class that is, for example, 3 lessons long, will occupy 3 table cells. But with this solution comes an issue: whenever the rowspan is extended, the table cells that were “overwritten” are not automatically removed, they are shifted to the side – and this can hinder the integrity of the timetable, because classes that were, for example, on Thursday, can accidentally move to Friday, and this is not supposed to happen. So, to prevent this, we made a huge switch statement that grabs the start time and the number of lessons of a session to determine which table cells should be deleted to keep the timetable’s integrity. So if a session starting at 8:20 has more than 1 lesson, the table cell for 9:10 should be deleted. And this solves that problem, but another one arises. If a few table cells were deleted and the user decides to choose another item from the dropdown box, then the new timetable might need to make use of table cells that were previously deleted. They’re just gone. So, in order to prevent this, we made it so that whenever a new choice is picked from the dropdown menu, the table goes back to default: it regenerates. All of the table cells are back and ready to be used. And that was how we made our timetable page.

**Testing phase**

After having developed and implemented all of this, we had to do some testing. Of course, we also did test throughout the development phase, and it consisted mainly of producing a variety of different schedules, making use of all of the functionalities of the program – really pushing it to the limit to check for any possible bugs. So basically, just using it and abusing it to look for anything that might have gone wrong.

**Results & Discussion - LAURA**

Having done all of this, if we are to compare our predictions to the actual outcome of the project, we feel pretty good about it. The program can successfully schedule, reschedule and cancel sessions; there is a website that has smooth interaction with the program and can display timetables accurately; the system is overall very user friendly, and all of the critical requirements were met by it.

There are a couple of things we did not do, though: the program does not generate timetables or check for overlaps for teachers, and students with credits do not get personal timetables. With the timeframe at hand for the Semester Project, we felt that it was too ambitious for us to do individual timetables as opposed to timetables for whole classes, so we decided to give that up and focus on the more “elementary” priorities for the sake of having something functional and whole, instead of trying to do too much and risking ending up missing important functionalities. Either way, the program’s final outcome was very satisfactory.

**Learning outcome**

We are aware that the Semester Project relates directly to Problem Based Learning. Instead of us being given a step-by-step tutorial on how to get our project done, we were given a case and told to solve it by ourselves.

This made it possible for us to learn on a deeper level as opposed to a superficial one, due to the fact that we did stuff, got our hands dirty, instead of just memorizing stuff and dumping it on an exam.

By working by ourselves, depending on each other and actually creating something from scratch, we felt extremely engaged to developing the project, and also felt autonomous and self-fulfilled.

Of course, this does not come without its shortcomings: one issue that arose from time to time was that, without constant supervising, it became easier for us to get stuck in a task, because we were not being told what to do, we were figuring it out as we went. Then again, this risk was mitigated by the help we could get from our supervisors by reaching out to them, making us super grateful towards them.

And naturally, we are also grateful to each other. As a group, we worked very well, we consistently output good work, made progress each time we met and overall had a good time together. This made it possible for us to present this solid project and we look forward to working together again soon. Thank you for your time, and we hope you enjoyed our presentation. Thank you thank you!! :)