

Dissertation

**DEVELOPMENT OF AN AUTOMATED COURSE SCHEDULING  
TOOL USING CONSTRAINT PROGRAMMING**

**Sohaib Nassar**

**UNIVERSITY OF CYPRUS**



**DEPARTMENT OF COMPUTER SCIENCE**

**May 2023**

**UNIVERSITY OF CYPRUS**  
**DEPARTMENT OF COMPUTER SCIENCE**

**Development of an Automated Course Scheduling Tool using Constraint  
Satisfaction Programming**

**Sohaib Nassar**

Supervisor:  
Dr. Anna Philippou

The Individual Diploma Thesis was submitted for partial fulfillment of the requirements  
for obtaining the degree of Computer Science of the Department of Computer Science  
of the University of Cyprus

May 2023

## **Acknowledgment**

Foremost, I would like to express my sincere gratitude to my supervisor. Dr. Anna Philipou, for her unconditional guidance and support during my journey of writing my dissertation. Her insight, feedback and expertise have been invaluable in guiding me, and ensuring the quality of my work.

I am also deeply thankful to my family for their support, understanding and encouragement during my journey.

I would like to also extend my gratitude to my friends who have provided me with valuable insight. Their presence and support have kept me encouraged and motivated to excel in my work.

Lastly, I would like to express my appreciation to every professor, friend, colleague, and everyone who played a part in my journey. Their support means the world to me.

## **Abstract**

Course scheduling holds immense significance within a university department, demanding extensive coordination and substantial manpower to be crafted every semester. The department of Computer Science in the University of Cyprus produces its course schedules by utilizing a manual approach.

Course scheduling, being an NP-complete problem, is a very hard problem for computers to solve efficiently. Hence, most approaches that attempt to solve it use some form of Artificial Intelligence techniques.

This dissertation focuses on the development of a system that would replace the current time-consuming approach with a more efficient and effective automated approach. Our method employs the Minizinc tool, which is a modeling language used for modeling Constraint Satisfaction problems to be passed to independent solvers.

Our work starts with exploring the course scheduling problem. Forthwith, we follow the stages of the Software Development Process in the light of developing the automated tool. Our final system consists of a web-interface which is used for the communication of the requirements of the schedule to the system, a Minizinc model, used to describe the schedule problem in a solver-independent form, and back-end scripts which are responsible for the communication of the aforementioned parts.

Finally, we evaluate our approach and determine that the developed Minizinc model successfully generates solutions for previous semester's schedules within a reasonable timeframe.

# Contents

<b>1 Introduction .....</b>	<b>1</b>
1.1 General introduction .....	1
1.2 Motivation.....	2
1.2.1 The Scheduling problem and its time-complexity .....	2
1.2.2 Scheduling in the Department of Computer Science.....	2
1.2.3 Motivation summary .....	3
1.3 Objectives of the dissertation.....	3
1.4 Methodology .....	3
1.5 Outline of Dissertation.....	4
<b>2 Background.....</b>	<b>6</b>
2.1 Introduction.....	6
2.2 Existing tools .....	6
2.3 Related work .....	7
2.3.1 Techniques overview.....	8
2.3.2 Advanced ILP Techniques .....	9
2.3.3 Scheduling in MiniZinc .....	10
2.3.4 Particle Swarm Optimization with Local Search .....	11
2.3.5 Simulated Annealing.....	12
2.4 Chosen approach.....	13
2.4.1 Constraint Satisfaction Programming .....	13
2.4.2 Minizinc modeling language .....	14
<b>3 Software Development Process &amp; Technologies .....</b>	<b>15</b>
3.1 Introduction.....	15
3.2 The Incremental model .....	17
3.3 Technologies .....	17
3.3.1 Front-end Development.....	18
3.3.1.1 HTML: Hypertext Markup Language .....	18
3.3.1.2 CSS: Cascading Style Sheets.....	19
3.3.1.3 JavaScript Programming Language .....	19
3.3.2 Back-end Development.....	19
3.3.2.1 PHP Hypertext Preprocessor.....	20
3.3.2.1 Python Programming Language .....	20
3.3.2.3 MySQL Relational Database Management System.....	21
<b>4 Requirements .....</b>	<b>22</b>
4.1 Introduction.....	22

4.2 Requirements Specification .....	23
4.3 Schedule Specifications.....	23
4.3.1 Course types .....	23
4.3.2 Schedule requirements .....	24
4.4 User types.....	26
4.5 Interface requirements .....	26
4.5.1 Interface functional requirements for Instructor users .....	27
4.5.2 Interface functional requirements for Administrator users .....	28
4.5.3 Non-functional requirements.....	30
4.6 Use cases.....	31
<b>5 Design .....</b>	<b>33</b>
5.2 System architecture .....	33
5.3 Database design.....	34
5.3.1 Tables .....	34
5.3.2 Database Relational Schema.....	38
5.4 Front-end design .....	39
5.4.1 Navigation pattern .....	39
5.4.2 Common pages .....	40
5.4.3 Instructor pages .....	42
5.4.4 Administrator pages .....	48
<b>6 Implementation .....</b>	<b>55</b>
6.1 Front-end implementation .....	56
6.1.1 Development environment .....	56
6.2.2 Development .....	56
6.1.2.1 Bootstrap.....	56
6.1.2.2 PHP .....	56
6.1.2.3 Security .....	57
6.1.2.4 Error prevention and handling .....	57
6.2 Minizinc Implementation .....	59
6.2.1 Environment.....	59
6.2.2 Algorithmic logic.....	60
6.2.3 Minizinc syntax .....	61
6.2.4 Parameters and Variables.....	62
6.2.4.1 Timeslots parameters .....	62
6.2.4.2 Variables .....	62
6.2.4.3 Unavailability parameters.....	63

6.2.4.4 Course relationships parameters.....	63
6.2.4.5 Course details parameter .....	63
6.2.5 Constraints .....	64
6.2.6 Versions.....	72
6.2.7 Solver.....	73
6.3 Back-end implementation .....	74
6.3.1 Input script .....	74
6.3.2 Output scripts.....	74
<b>7 Testing and Evaluation .....</b>	<b>76</b>
7.1 Minizinc Model Testing.....	76
7.2 Website Evaluation .....	80
7.3 Limitations.....	85
<b>8 Conclusion.....</b>	<b>86</b>
8.1 Conclusion .....	86
8.2 Challenges.....	86
8.3 Future work.....	87
<b>Bibliography .....</b>	<b>88</b>
<b>Appendix 1 .....</b>	<b>91</b>
<b>Appendix 2 .....</b>	<b>93</b>
<b>Appendix 3 .....</b>	<b>95</b>

# Chapter 1

## 1 Introduction

---

### 1.1 General introduction

### 1.2 Motivation

#### 1.2.1 The problem with Scheduling

#### 1.2.2 Scheduling in the Department of Computer Science

#### 1.2.3 Motivation summary

### 1.3 Objectives of the dissertation

### 1.4 Methodology

### 1.5 Outline of dissertation

---

### 1.1 General introduction

In our fast-paced world, staying organized is essential for the seamless and dependable operation of any entity. A vital ingredient of an effective organization is the thoughtful utilization of schedules. In simpler terms, for an organization to function optimally, it must embrace customized schedules that align perfectly with its unique needs and available resources. By doing so, it can cultivate an effective and efficient environment. These schedules are used to organize the different tasks, activities, or events in an organization to several times, places, and resources. The Department of Computer Science is a great example of such an institution where schedules play a key role in its operation. As such, the department develops a new schedule for every semester, to keep up with the ever-changing needs and resources available to it. These schedules mainly assign the different courses to be taught in the department to different timeslots, based on varying criteria. This dissertation aims to explore the scheduling problem and develop a dynamic scheduling system for the Department, that uses advanced artificial intelligence techniques, such as constraint satisfaction programming to automate the scheduling process. By doing so, the system can reduce the time and effort required to create and



manage the department's schedules. Through this work, we hope to enhance the department's operations and contribute to its overall efficiency and effectiveness.

## **1.2 Motivation**

### **1.2.1 The Scheduling problem and its time-complexity**

Task scheduling for any type of organization can be an extremely challenging job. The problem with scheduling lies in the difficulty of coordinating the availability of **people**, **resources**, and **time**. For small entities, such as small schools, small university departments or small businesses, the task can be overseen using simple organization tools, such as AscTimetable and SaaSworthy, or simply be done manually. On the other hand, as these organizations get bigger and become more complex, there is a substantial increase in resources and dependencies between the various tasks and resources to be scheduled. These factors deem the job of developing a correct and efficient schedule more complex and time consuming, which may make it overwhelming if not impossible for a human to accomplish.

The university course scheduling problem was shown by Kingston et al. [18] to be **NP-Complete**, meaning it is as hard to solve as any other NP problem. Therefore, finding a polynomial solution to it would prove that all NP problems can be solved in polynomial time [18]. This fact discourages us from attempting to implement a brute force approach, where all combinations would be tested until a satisfactory one is found, for the simple reason that the search space is astronomically large. Therefore, a different approach must be taken, such as utilizing techniques from the field of Artificial Intelligence

### **1.2.2 Scheduling in the Department of Computer Science**

Currently, an iterative method is used by the department's secretary to create each semester's schedule. In this method, the corresponding schedule from the previous year is edited based on any new needs or requirements that appear. This, for example, can include the addition of a new teaching instructor, a new course, a change to an existing course, or the addition of a new laboratory. This approach, as expected, can be very time consuming and labor intensive. In addition, it can lead to errors and inaccuracies in the

final schedule, particularly when there are significant differences in the requirements compared to the previous version. Furthermore, even minor adjustments to the semester schedule can have implications for the resulting timetable when applying the manual approach. Such subtle modifications may necessitate significant reconstruction due to the intricate interdependencies and resource availabilities among the various entities involved. Subsequently, creating an optimal schedule with this method can be very challenging, hence the schedule that is usually produced can be furtherly optimized, which in the case of a manual approach cannot be easily achieved.

### **1.2.3 Motivation summary**

The inspiration for this dissertation is to solve the problem of manual scheduling, as it has been described above. An automated tool for the creation of a semester schedule for the Department of Computer Science, which would consider the factors and complexities of the schedule. Such a tool would make the job of the secretary easier and make the department less vulnerable to unanticipated consequences.

## **1.3 Objectives of the dissertation**

The main aim of this dissertation is to research and develop an automated tool for the creation of a timetable-schedule for the numerous professors and teaching assistants at the Department of Computer Science in the University of Cyprus. This tool would consider the various factors that need to be taken in mind when creating a timetable, such as the availability of instructors and teaching assistants, the availability of resources, and the dependencies between the different courses. In addition, another objective is to ensure that the tool is adaptable to changing requirements and needs. For example, it should be able to handle the addition or removal of courses, changes in the availability of instructors or resources, and other unpredicted changes.

## **1.4 Methodology**

With the goal of understanding the problem at hand, conducting research was one of the first steps that were taken. Firstly, the methods with which different organizations handled their scheduling tasks were investigated, this included the way they managed to create their schedules, the difficulties they faced, and the software that they used, if any.

Furthermore, after some in-person talks with some of the instructors at the Department of Computer Science, a general idea as to what the problem is, was developed, as well as for an idea as to what a solution to this problem would look like and how it would function. Moreover, existing implementations that solved the scheduling problem were investigated, alongside the different algorithms that were used to implement such tools. Subsequently, the incremental model of the Software Development Life Cycle (SDLS) was decided to be used, and the first design iterations for the database and the front-end interface were created. After the initial design, the Incremental method was applied to advance the interface, the basic functionalities were added one by one. Besides that, the Minizinc tool was chosen to be used for the development of the algorithm (model) that would generate the final schedules. The initial model was developed to match the existing functionality of the interface. Following that, both the Minizinc implementation and the interface were developed iteratively, by adding new constraints to the Minizinc model and testing them, then adding the new functionalities to the interface. In addition, the database was also incrementally developed, on top of the initial design, so that it can fulfill the growing needs and complexity of the tool. Finally, once the tool was complete, testing was conducted to find any weak points, and to examine its final abilities.

## **1.5 Outline of Dissertation**

The remainder of the dissertation is organized as follows.

Chapter 2 presents the conducted research which was orchestrated for the study of existing approaches that solve the scheduling problem. Moreover, the background work preceding the dissertation is also presented. In addition, the decisions for the approach used in our implementation are discussed.

In Chapter 3, the Software Development Process (SDP) is demonstrated. Specifically, the Incremental Model, which was used in our development process, is illustrated. Additionally, the various employed technologies are presented and discussed.

Chapter 4 documents the Requirements Analysis stage of the Software Development Process. This includes documenting and elaborating on the functional and the non-functional requirements for the system interface and the backend algorithm.

Chapter 5 illustrates the Design stage of the SDP where the design decisions for the Web interface and the Database are depicted and explained.

In Chapter 6, some key components of the implementation are showcased, providing explanations on some decisions that were made regarding the front-end interface, the back-end scripts and the Minizinc model.

Chapter 7 discusses the testing and evaluations that were conducted in the Evaluation stage of the SDP. In addition, the results of these evaluations are also presented and discussed.

Finally, Chapter 8 concludes this work by highlighting the encountered challenges that were faced during the development process and proposes some future improvements that can be implemented in the system as part of future work.

## Chapter 2

### 2 Background

---

#### 2.1 Introduction

#### 2.2 Existing tools

#### 2.3 Related work

##### 2.3.1 Techniques overview

##### 2.3.2 Advanced ILP Techniques

##### 2.3.3 Scheduling in MiniZinc

##### 2.3.4 Particle Swarm Optimization with Local Search

##### 2.3.5 Simulated Annealing

#### 2.4 Chosen approach

##### 2.4.1 Minizinc

##### 2.4.2 Constraint Satisfaction Programming

---

### 2.1 Introduction

In this chapter, we present the research which was conducted for this dissertation for the purpose of understanding and solving the scheduling problem. The main goal was to understand and identify the different methods that have already been employed and researched for this specific problem. The first section of this chapter discusses the related works, which include various techniques such as Advanced ILP Techniques, Scheduling in Minizinc, Particle Swarm Optimization with Local Search, and Simulated Annealing. The second section presents and reviews the technique of Constraint Satisfaction Programming from the field of Artificial Intelligence.

### 2.2 Existing tools

There exists a plethora of scheduling tools used for task scheduling. Ad Astra, College Scheduler and aSc Timetables are some of many proprietary scheduling tools. These tools

provide convenient solutions for small institutions to efficiently organize their courses. By automating the scheduling process, these tools can save significant time and effort for the institution. Moreover, they offer a diverse range of established functionalities that can be tailored to meet the specific requirements of the institution. However, it's important to note that these tools are proprietary, meaning they require a financial investment from the institution to utilize their benefits. In addition, they might not be suited for the unique requirements of an institution and therefore force the institution to change the way it operates.

Developing a specialized tool for the department, on the other hand, can offer many advantages in comparison to the use of the existing tools. Firstly, the department would be self-sufficient, meaning it would not rely on a third party for a crucial part of its operation. Furthermore, a specialized tool can offer long term scalability, where the tool can grow and adapt to evolving needs. Finally, a specialized tool could be integrated with existing systems in the department to facilitate seamless and integrated operation.

## **2.3 Related work**

The field of automated course scheduling has been subject to comprehensive research, incorporating diverse techniques and methodologies. The objective of this sub-chapter is to present and expound upon various approaches that have been established for the purpose of addressing this specific issue.

It is important to note that the scheduling problem can have a variety of instances and characteristics, based on the needs and requirements of each problem. In essence, the scheduling problems can look very different, depending on its specific application. Consequently, certain approaches may prove more suitable for specific problems compared to others.

The problem which this work is trying to solve is the problem of course scheduling in the Department of Computer Science as it was described in **Chapter 1**. Hence, the approach that we would choose for solving this problem must satisfy some criteria. It should be efficient, as in, find a valid solution in an acceptable timeframe. It should be

uncomplicated, so that it can be implemented in the timeframe available to us. It should be suitable for the department's schedule requirements. Finally, it should not require any proprietary software.

In light of this, in the next sub-chapters, an overview of the different methods used for solving the problem of course scheduling is presented. Consequently, some implementations that have been deployed for tackling scheduling problems are explored.

### **2.3.1 Techniques overview**

There are various techniques used in solving the scheduling problem, these can be grouped into three categories, single solution based meta-heuristics approaches, operational research-based approaches and population based meta-heuristics techniques [21]. Each one of these will be briefly introduced and evaluated to determine their suitability for addressing our problem.

Single solution-based approaches, sometimes referred to as local search algorithms start with a single solution and then try to find a better solution by searching the neighborhood space. They do this by iteratively improving the given solution, until an optimal solution is found. These techniques offer a high-quality solution while also being efficient. However, these approaches may be difficult to implement in large scale problems because of their scalability issues. In addition, they can be biased in the solution they find, as the scheduling problems are very complex and have a large search space [21]. For these reasons, they would not be our first choice for developing a scheduling algorithm from the department. Simulated Annealing is an example of a single solution based meta-heuristic approach.

Operational research-based approaches are usually derived from the Graph Coloring problem. These approaches try to develop an algorithm that finds one of the many valid solutions that satisfy all the requirements. From their nature, these approaches don't always find the optimal solution, but a solution that satisfies the given requirements. The problems in these techniques can also be presented as a constraint satisfaction problem [21]. These techniques could be suitable for our problem, as they are flexible for a variety

of complex problems. In addition, their adaptability to changing requirements is ideal for our implementation since the requirements of the department are not static. Examples of OR based approaches include Advanced ILP Techniques and Constraint Satisfaction Programming.

Population-based meta-heuristics approaches are similar to single solution-based approaches with the difference that they focus on a set of possible solutions rather than focusing on a single solution to be optimized. This is achieved by iteratively changing a population of solutions until a solution of high quality is found. An advantage of using these approaches includes their global search capability, whereby searching through multiple solutions, they are more likely to find a good solution. In addition, their ability to be parallelized can offer better utilization of computational resources and find solutions in less time. Nevertheless, these approaches come with some drawbacks. As they work on a set of solutions, their computational complexity is higher than single solution-based approaches, especially for large problems such as the scheduling problem. In addition, it can be difficult to model complex scheduling constraints in these approaches [21]. Examples of algorithms that follow the population-based meta-heuristics approaches include Particle Swarm Optimization with Local Search.

### **2.3.2 Advanced ILP Techniques**

In their paper Ahmed Wasfy and Fadi A. Aloul [12] describe an integer linear programming (ILP) based approach for creating university course schedules. ILP is a mathematical optimization programming method in which all the variables in a program are restricted to integers. The authors mention that most of the proposed solutions to the university class scheduling problem either cannot prove un-satisfiability or cannot guarantee that the found solution is the optimal one. For this reason, they decided to develop a new approach which is mathematically complete. In other words, their approach would be able to detect if a certain scheduling problem is unsatisfiable, or on the other hand, the solution if it exists. If the problem is satisfiable, then their approach would search for all possible solutions to find the optimal one. The authors first start by formulating the problem into a 0-1 ILP problem. To do this, they analyzed the requirements of the schedule and generated the equivalent mathematical constraints. They



also developed an easy-to-use GUI interface, which they used to gather user input. The tool then converts these inputs into a 0-1 ILP instance, which is passed to a backend ILP solver that tries to find the optimal solution. Finally, they tested their approach on different cases with variable sizes and concluded that it showed promising results.

### 2.3.3 Scheduling in MiniZinc

Rahman et al. [15] proposed a solution to the problem of course scheduling which employed Constraint Satisfaction Programming. Specifically, they utilized the Minizinc tool, which is a high-level declarative modeling language that allows for the development of solver-independent general models. This language allows for the creation of models that can be efficiently passed to a solver along with the required data, which are passed as parameters. In the Minizinc language, a satisfiable solution is achieved with the use of the relevant variables, parameters, and their relations in the form of constraints.

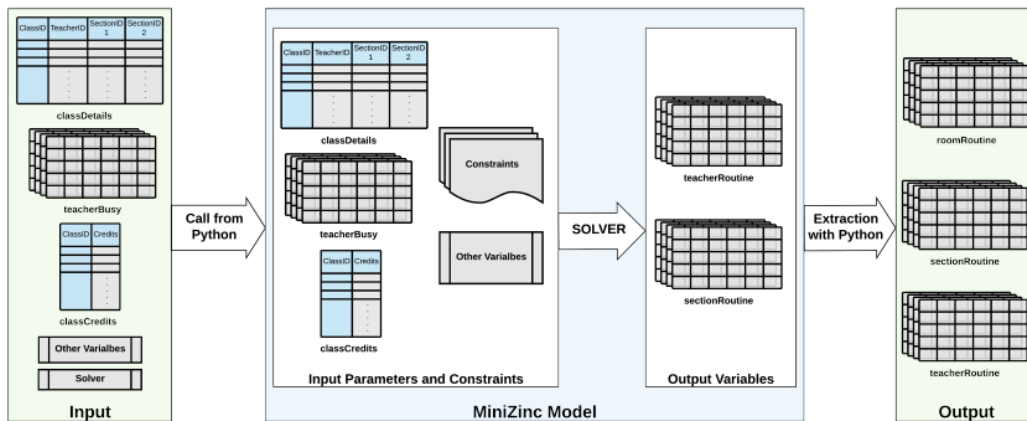


Figure 2.1 – Minizinc Implementation [15]

The above diagram depicts their implementation. On the left, the input can be seen as defined by the details of each course, a 3D data structure which represents the availability of the teachers, and some other variables which are needed for the specific problem. The model was then passed to the available off-the-shelf solvers, which Minizinc provides, and received the satisfiable schedules in the form of populated 3-D tables.

It is important to note that in their approach, the authors defined the days as having a fixed number of slots, therefore courses cannot be taught in time periods that are not included inside one of these slots.

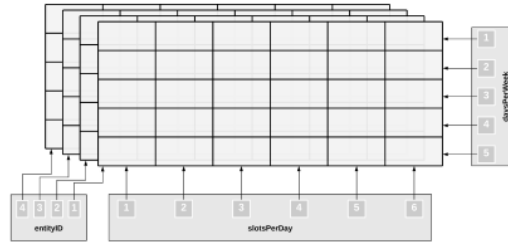


Figure 2.2 – Weekly Schedule [15]

The above diagram displays a 3-D data structure which is comprised of 2-D planes which they used to represent the weekly schedule of one individual instructor.

In their quest for finding the best solver that can satisfy a solution to the problem, the authors tested several sets of problems, on a varying number of variables, such as courses and teachers, and found the below results.

Routines	3	6	7	9	16	20	40
Teacher(s)	1	2	3	5	8	12	24
Section(s)	2	4	4	4	8	8	16
Courses(s)	1	4	6	10	16	24	48
Chuffed	0.56	0.69	0.80	1.08	2.17	5.04	47.3
Gecode	0.51	0.93	9362	$\infty$	$\infty$	$\infty$	$\infty$
Optimathsat	-	-	-	-	-	-	-
Opturion	-	-	-	-	-	-	-
Gurobi	-	-	-	-	-	-	-
CBC	0.88	2.6	16.7	36.9	911	1680	$\infty$
Oscalr CBLS	-	-	-	-	-	-	-
OR Tools	-	-	-	-	-	-	-

Figure 2.3 – Results [15]

They concluded that the performance of the Chuffed solver was outstanding in comparison to the other ones. This, they explained, was because of Chuffed’s ability to generate “no-goods” or sub-trees which would lead to failures. In addition, its ability to detects conflicts from an early stage and eliminate bad decisions, makes it the best available solver for the problem of course shedding.

### 2.3.4 Particle Swarm Optimization with Local Search

Ruey-Maw Chen and Hsiao-Fang Shih in their article titled “Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search” [13], experimented in a new approach that takes advantage of Particle swarm optimization to solve the NP-complete problem of course timetabling. They first evaluated the two types of PSO versions, these being the inertia weight version and the

constriction version. Then, the authors defined the course timetabling problem and differentiated the constraints into hard and soft constraints. Hard constraints are these which the final schedule must always satisfy, and the soft constraints are these which the program satisfies as much as possible. Furthermore, they explained how the PSO algorithm works and how it was implemented in their case. PSO, as the authors described, is a population-based optimization algorithm developed in 1995 by Kennedy and Eberhart. Each particle represents a bird in a flock, and the swarm is a group of particles. The position of each particle is considered as a candidate solution to an optimization problem, and each particle is assigned a fitness function that is designed based on the corresponding problem. As the PSO particles evolve, their movement is determined by the personal best of each particle and the global best. However, in this implementation, the particles are likely to be trapped on the local optimal solution. For this reason, they designed a disturb mechanism to the movement of the particles, to make them more likely to escape the local solution and discover the global one. This mechanism is called the interchange heuristic. Finally, the authors concluded that their approach was able to improve the satisfaction of the teachers and classes towards the final schedule, this they determined was a result of using the interchange heuristic mechanism.

### **2.3.5 Simulated Annealing**

Another solution to the course scheduling problem was developed by E. Ayca and T. Ayav in their paper titled “Solving the Course Scheduling Problem Using Simulated Annealing” [14]. In this article, Ayav et al. addressed the task of resolving the course scheduling challenge utilizing simulated annealing techniques. They implemented a solution based on their specific department’s needs, namely the department of computer engineering in the university of Izmir. Firstly, they presented the scheduling problem along with the constraints that their solution would need to satisfy. Then, the authors explained the simulated annealing method, which as they describe, is a probabilistic technique for approximating the global optimum. This technique is based on particles and system energy (temperature), the particles are replaced by elements and the energy is defined by the timetable cost. The particles are initially placed randomly, and the initial temperature and cost are calculated. Then, a neighborhood search is carried out to find the next possible solution set. For this, they utilized three algorithms, namely the Simple

Search Neighborhood (SSN), Swapping Neighborhoods (SWN), and Simple Searching and Swapping Neighborhoods (S3WN). The authors concentrated their efforts on the implementing and testing the merging of the three distinct algorithms, and subsequently evaluated and documented the results pertaining to their respective speeds, which are documented in the below table.

SSN		SWN		S <sup>3</sup> WN	
Cost	CPU(sec)	Cost	CPU(sec)	Cost	CPU(sec)
3900	29	9300	40	4300	34

Figure 2.4 – Simulated Annealing Results

From this, the authors determined that the SSN provided the best results. Forwith, they decided to combine the aforementioned algorithms and found that it achieved the best results, which can be seen in the below table.

Case A (sequentially)		Case B (in turn)	
Cost	CPU(sec)	Cost	CPU(sec)
4100	87	3600	28

Figure 2.5 – Simulated Annealing Combination Results

The authors concluded by stating that their approach obtained a timetable with a lower cost than the manual method which was used in creating the schedule for the 2007-2008 fall semester for their department.

## 2.4 Chosen approach

Based on the conducted research, particularly referencing [15], it appeared that the Minizinc constraint modeling language demonstrates exceptional potential in comparison to the other methods. Additionally, it was also found to be one of the most versatile and user-friendly approaches. For these reasons, it was decided that this tool would be the method to be explored in the development of the automated system.

### 2.4.1 Constraint Satisfaction Programming

Constraint satisfaction programming (CSP) is a problem-solving technique that uses a variety of approaches from artificial intelligence and operations research to solve combinatorial problems. In recent years, it received the attention of many researchers

because of its potential for solving real-world hard problems. CSP uses constraints, which are logical relations between several unknowns to express real world dependencies between different entities or physical objects. These unknowns are also referred to as variables, which must take a value from a given domain. Thus, the constraints limit the possible values of an object, in essence giving partial information about it. Constraints are non-directional, meaning a constraint on object A can be influenced by object B, but object B is not influenced by object A. In addition, these constraints are often not independent, meaning multiple constraints usually refer to the same object. An algorithm that solves such problems i.e., the satisfaction of the constraints, can give a single solution, all possible solutions, or the optimal solution based on an objective function. CSP has been adapted to many applications in recent years, computer graphics, natural language processing and timetabling to name a few [17].

#### **2.4.2 Minizinc modeling language**

Minizinc is a language used for modeling constraint satisfaction. It was developed with the intent of being a standard modeling language which is expressive and solver independent. In other words, most constraint satisfaction problems can be modeled using it, and then passed to many generic constraint satisfaction solvers to be solved. Minizinc can do this because of its nature, being sufficiently high-level which enables the easy expression of many combinatorial optimization problems, and sufficiently low-level that it can be easily graphed onto many solvers [16].

A Minizinc model has three components, parameters, variables, and constraints. Parameters can be either an input value or a fixed value defined in the model. Parameters are mainly used for passing the inputs for each problem into the model. Variables, on the other hand, are not given a value in the model, instead only their domain is defined, their value is determined by the solver. For instance, “**var 1..130: age**”, is a variable named “age” which can take values from 1 to 130. The solver tries to find values for these variables, while also respecting all the constraints that involve them. For example, “**constraint age > 17**” ensures that the value given to the “age” variable is always bigger than 17. Therefore, when the solver is called to find a satisfactory solution, we can be certain that the resulting age variable will be 18 or higher, up to 130.

## Chapter 3

### 3 Software Development Process & Technologies

---

#### 3.1 Introduction

#### 3.2 The Incremental model

#### 3.3 Technologies

##### 3.3.1 Front-end Development

###### 3.3.1.1 HTML: Hypertext Markup Language

###### 3.3.1.2 CSS: Cascading Style Sheets

###### 3.3.1.3 JavaScript Programming Language

##### 3.3.2 Back-end Development

###### 3.3.2.1 PHP Hypertext Preprocessor

###### 3.3.2.2 Python Programming Language

###### 3.3.2.3 MySQL Relational Database Management System

---

#### 3.1 Introduction

The software development life cycle defines the steps with which a software or system will be developed. This includes everything from initially defining the goal to the deployment of the software. The life cycle aims to simplify the process of developing a software system and lowering the cost while ensuring high-quality operations. Furthermore, it tries to minimize any errors that might occur from the development process, it also tries to detect and eliminate any gaps that might have been left unseen during the development process [1].

As mentioned above, the development life cycle consists of many stages. The first stage is the Requirement Analysis stage, which can be viewed as the foundation of the project. Creating a good Requirement Analysis stage can make the rest of the development simpler and more efficient [1]. In this stage the goal of the project is determined, the

resources are defined and allocated, the different functionalities that will be developed are defined and given priorities, the risks and the limiting factors are also defined [2].

The second stage is the Design stage. In this stage the gathered requirements are used to create the system architecture which will be used for implementing the system [3]. To be more precise, the database design is created in this stage, as well as for the design for the system overview, how the system will function and what the user interface will look like.

The third stage is the Development stage. During this stage the development of the system starts, this means that the programmers start developing the different parts of the system. This development is based on the design documents that have been gathered in the Design stage [3].

The fourth stage is the Testing stage. The goal of this stage is to find any defects which will be documented and assigned to the developers to resolve them. The system in this stage is tested thoroughly so that it fits the customer's standards [3].

The fifth and final stage is the deployment and Maintenance stage. After the system has been tested thoroughly and all defects have been fixed, the system is ready to be deployed. During the Maintenance part of this stage, the final system is monitored and any issues that may arise are fixed or any enhancements that need to be applied are also implemented [3].



Figure 3.1 – The Software Development Life Cycle (DSLFC) visualised.

### 3.2 The Incremental model

The Incremental model, also known as the iterative waterfall model, is one of the main models which are used for the development of software systems. As the name suggests, this model can be viewed as a series of waterfall models [4]. It is based on the idea of incremental development, where the software under development is divided into smaller parts. These parts are designed, developed, and tested incrementally until the final product is realized. This model offers many advantages compared to other software development models. Its main advantage is the fact that the feedback of previous iterations can be used in the development of a new iteration [4]. This allows the developers to better understand the problem, which may not be clear from the early stages. In addition, it pushes the stakeholders to be more involved in the development of the system [4].

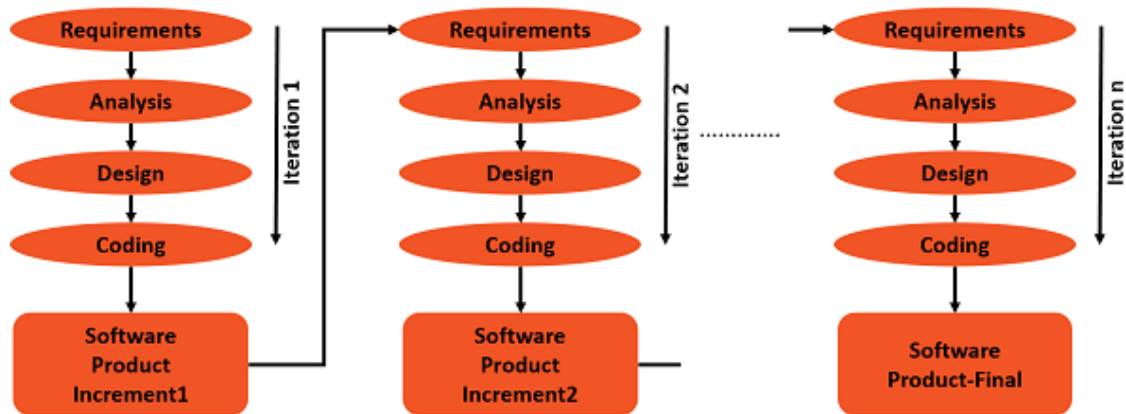


Figure 3.2 – The Incremental model visualized.

After some research and collaboration with the dissertation supervisor, this approach was deemed as the best to be used in the development of the automated tool. This decision was based on its seamless alignment with the specific needs and conditions. In addition, the abilities, and strengths of the Minizinc tool were not very clear from the beginning. Subsequently, the system was developed incrementally, where after each iteration, additional requirements were added and then designed and developed, after that, they were tested to insure proper functionality.

### 3.3 Technologies



In the process of developing the automated tool, many technologies and programming languages were used. These technologies can be differentiated into front-end technologies and back-end technologies. In the next sections, these technologies are briefly presented.

### 3.3.1 Front-end Development

Front-end Development focuses on the visible side of an application, this means the development of the part of the application which the user sees and interacts with. The goal of front-end developers is to create a visually appealing, intuitive, and user-friendly interface that enhances the user experience [6]. This includes the ease of use of the interface, the learnability of the interface, in other words how easy it is for a user to learn how to use the interface, the reliability of the interface and the responsiveness of the interface. Some of the most known and used languages that support the development and maintenance of such interfaces include HTML, JavaScript, and CSS. In the below figure, the roles of these languages in the overall interface can be visualized.

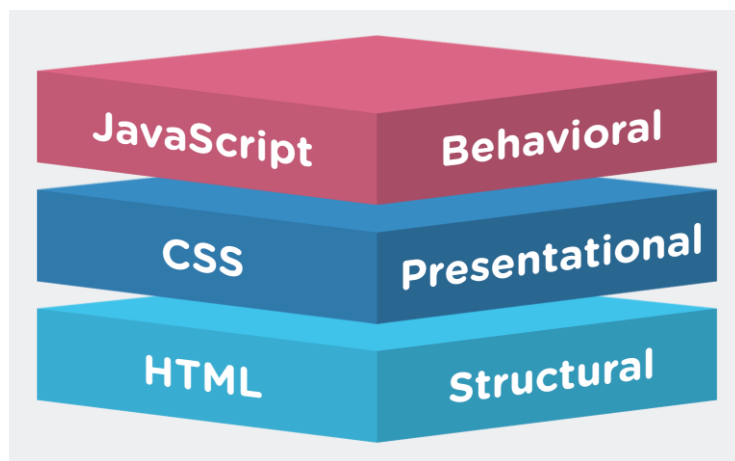


Figure 3.3 – Front-end languages stack

#### 3.3.1.1 HTML: Hypertext Markup Language

HTML stands for Hypertext Markup Language, a markup language which creates the structure of a webpage, this includes the page layout, paragraphs, links and more. It creates what can be viewed as the skeleton of the web page. It does this by utilizing tags and elements, where each tag and element has a specific function that contributes to the overall structure of the web page. An advantage of using HTML is its compatibility with other languages used for web-development, these include JavaScript and CSS, which when combined create a more robust and seamless web page [9].

### **3.3.1.2 CSS: Cascading Style Sheets**

As explained above, HTML mainly controls and organizes the structure of a webpage, without much of an ability to control the styling of the elements. Meanwhile CSS, which is a styling language, is responsible for the specification of fonts, colors and layouts of the elements that are to be displayed on the page [10]. This styling makes the webpage more visually appealing, easier to read and use. The CSS code can be included in the headers of the HTML tags, in the HTML document with the <style> tag, or simply be placed in a different file.

### **3.3.1.3 JavaScript Programming Language**

JavaScript is a type of scripting or programming language that enables the implementation of intricate functionalities on web pages. Whenever a web page is more than just a static display of information, JavaScript can be used to display dynamic content updates, interactive maps, animations, video players with scrolling capability, and many more features. In other words, it is commonly used for determining the behavior of web pages [11]. It runs on the client-side, which means it is executed by the user's web browser rather than the web server.

### **3.3.2 Back-end Development**

Back-end development is, as the name implies, the behind-the-scenes part of any application, whether that is a web application or a mobile application. It focuses on server-side programming but also on the databases and managing them. This process can be visualized in Figure 3.4. Languages that support backend development include PHP and Python, which will be analyzed in more detail in the subsequent sections [6].

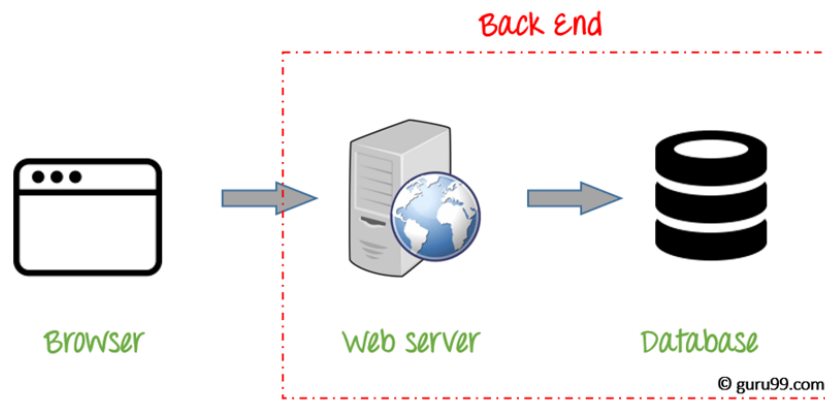


Figure 3.4 – Back-end visualized.

### 3.3.2.1 PHP Hypertext Preprocessor

PHP is a general-purpose server-side scripting language and interpreter which is used widely in the development of web applications. The PHP script can easily be included in any html file, without the need to include additional files. When a client requests a web page, the parser interprets the PHP code and generates the required html code. As a result, web pages become dynamic since the displayed content varies based on the input or other factors. As mentioned above, PHP is a server-side scripting language, this implies that the script is only available to the server, meaning the client does not receive any PHP code, instead it receives the complete html code. PHP is especially useful in applications where there is a need for a database, as it can communicate with the database and request any data that needs to be displayed or any updates to the data that need to be executed in the database [5].

### 3.3.2.1 Python Programming Language

Python is a high-level, object-oriented programming language that was developed in 1991 by Guido van Rossum and has since become a staple language in the world of programming. It is great for rapid application development because of its built-in high-level data structures alongside its abilities for dynamic typing and dynamic binding. Dynamic typing means that Python does not require variable type declarations, which can save time and make code easier to read. Dynamic binding means that Python can handle changes to an object's type at runtime, which enables programmers to write flexible and adaptable code. It emphasizes readability which characterizes it as simple to use and easy

to learn. Another great advantage of using Python is its support for a vast number of useful modules and packages which encourages code reuse [6].

### **3.3.2.3 MySQL Relational Database Management System**

MySQL is one of the most popular database management systems and has since been deemed as the number one choice for developers for a variety of reasons. It is an open-source relational database management system, which means that it is freely available for developers to download, use, and modify. It offers a variety of useful sets of functions which can make the process of database management and administration much more efficient. For instance, it offers built-in support for indexing, which can speed up database queries and improve performance. It also offers robust backup and recovery features, thus ensuring that data is not lost in the event of a system failure. Other advantages of using it include its ease of use, which is supported by its graphical user interface (GUI), its reliability, its scalability, its security, and its performance compared to other database management systems.

## Chapter 4

### 4 Requirements

---

#### 4.1 Introduction

#### 4.2 Requirements Specification

#### 4.3 Schedule requirements

#### 4.4 User types

#### 4.5 Interface requirements

##### 4.5.1 Interface functional requirements for Instructor users

##### 4.5.2 Interface functional requirements for Administrator users

##### 4.5.3 Non-functional requirements

#### 4.6 Use cases of the scheduling tool

---

### 4.1 Introduction

Requirements identification is extremely important as an initial step in the software development life cycle (SDLC), as elaborated in Chapter 3. It involves identifying the needs, expectations, and specifications of the stakeholders of a particular software application. In other words, the main goal is to ensure that the application fulfils the needs of the users and stakeholders. This process involves identifying and documenting both the functional and the non-functional requirements of the software. Functional requirements refer to the features and functionalities that the software must offer. These requirements are usually documented in terms of what the software “should do” and under which conditions it should be able to do so. Non-functional requirements refer to the reliability, performance, and usability of the application. Examples of Non-functional requirement include turnaround time, which is the length of time needed for completing a task, learnability which is defined as the time duration that is needed for a beginner user to learn and become efficient in using an application, and usability. In this chapter, the requirements for the automated tool are documented and described.

## **4.2 Requirements Specification**

The main source for identifying the requirements for the automated tool was my supervising professor Dr. Anna Philipou. With her extensive experience in the department, Dr. Philipou provided valuable insight into the needs and expectations of the department regarding the automated tool. Another source which offered great insight was the study of similar systems that are being used by similar institutions. Furthermore, studying the systems that have been developed in the field of schedule making, whether experimental or functional, has contributed to a greater understanding of the required essential features and capabilities. In addition, my own experience as a computer science student at the university provided me with valuable information, as I have a good understanding of the difficulties and needs that need to be satisfied in the semester schedule.

All these sources provided valuable insight into what the system should be able to do and how it should be designed. By leveraging these insights, a comprehensive set of possible functional and non-functional requirements that accurately reflected the needs and expectations of the stakeholders was created.

## **4.3 Schedule Specifications**

The automated tool will be designed primarily to fit the needs of the Department of Computer Science, for this reason, the requirements for the final schedule were specified based on the current scheduling model that is implemented in the department. With this in mind, the department offers a range of different lectures and laboratories, which all had to be identified and analyzed for the purpose of developing the schedule generator.

### **4.3.1 Course types**

As mentioned above, the university offers a variety of courses, for this reason The different combinations of lectures, tutorials and laboratories that are common in the department's schedule were identified and are listed below.

1. Courses that have one lecture per week only.
2. Courses that have two lectures per week only.

3. Courses that have one lecture plus a tutorial per week.
4. Courses that have two lectures plus a tutorial per week.
5. Courses that have two lectures plus a laboratory per week.
6. Courses that have one lecture plus a laboratory per week.
7. Courses that have one lecture, a tutorial and one laboratory per week.
8. Courses that have two lectures, a tutorial and one laboratory per week.
9. Courses that have two lectures, a tutorial and two laboratories per week.

#### **4.3.2 Schedule requirements**

The automated tool should produce two types of schedules. The first one being a schedule for each individual instructor. In this schedule, for each Instructor, all the courses, tutorials, and laboratories they teach, would be displayed on it with the time of teaching. The second type is a set of schedules for each lecture hall/laboratory room. In these schedules, for each room, the courses would be displayed along with the time of teaching.

In the instructors' schedule, the model would focus on the instructors and their availability to produce a valid schedule, while in the rooms' schedule, the model would focus on the number of rooms and on their availability. In our implementation, only the instructors' schedule was realized.

Generally, in the department, the lectures are scheduled on weekdays except for Wednesdays. The lectures of type "two lectures a week" are scheduled on Monday and Thursday or on Tuesday and Friday. Meanwhile, the tutorials are mostly reserved for Wednesdays. In addition, laboratories are scheduled to any day if they are of type "once a week" or follow the Monday and Thursday or Tuesday and Friday model if they are of type "twice a week". The schedule generator must respect this pattern in the final generated schedules.

The priorities assigned to the requirements are determined by the level of importance they hold in the final schedule implementation.

The requirements for the instructor's schedules are presented in more detail in the below table.

	Requirement description	Priority
1	Courses with two lectures per week should be taught at the same time on the two days of the week. Either on Monday-Thursday or Tuesday-Friday.	High
2	The instructor must not have any assigned courses during the days/ hours where they are busy.	High
3	The schedule should not have courses that are taught at the same time and belong to the “course conflicts” list.	High
4	The schedule should assign all the courses in the “parallel courses” list to be taught at the same time.	High
5	Tutorials should be scheduled on Wednesdays.	Moderate
6	The number of lectures/tutorials that are taught at the same time should not exceed the number of available lecture halls.	High
7	The number of laboratories that are taught at the same time should not exceed the number of available laboratories.	High
8	Each assigned room should not have a maximum capacity lower than the number of expected students for each lecture/tutorial.	High
9	In the case of lectures that require laboratories where the number of students exceeds the capacity of the laboratories, the schedule should have multiple iterations/sections of the same laboratory.	High
10	The instructor that instructs a lecture can be different from the instructor that instructs the laboratory of a course. Meaning, the instructor that registers a course, also registers its laboratories. These laboratories can be set to be instructed by a different instructor.	Moderate
11	For laboratories that have multiple sections, these sections should be scheduled in succession. Meaning they would be scheduled one after the other, on the same day.	Moderate

Table 4.1 – Schedule Requirements



#### **4.4 User types**

The users of the system should be differentiated into different types with their own set of functionalities in the system, depending on their role and position in the university. For the use of the automated tool, two types of users were identified to be necessary for the smooth functionality of the system. These two different user types will not be used in practice by an equal number of users.

The first user type is the instructors, which will cover most of the users. These users will have a somewhat limited set of functionalities, their main goal for using it will be to document their needs and demands in the semester schedule, including, amongst other things, their availability during the week and the courses they will instruct.

The second type is the administrators, which will consist of a selected few in the department's staff. These users will have many of the same functionalities as the instructors. However, they will have some additional functionalities which would allow them to initiate the system, manage the users, and manage any dependencies between the different courses.

#### **4.5 Interface requirements**

The interface should be a web-based application where the users can interact with the tool. It should offer a variety of functionalities based on the type and privileges of the user. The main goal of the interface is to be a simple, easy to use tool to supplement the required information about the schedule to the database, with which the backend algorithm would utilize to produce the schedule.

Below are the specified requirements for the interface, with their description and the priority that should be given to them in the implementation stage, categorized based on the user type.

#### 4.5.1 Interface functional requirements for Instructor users

Requirement		Description	Priority
1	Logging in	The instructors should easily be able to login to the system using their username and password.	High
2	Logging out	The instructors should have an easy way of logging out of the system.	High
3	Setting a password	The instructors should be able to set their password once they login into the system for the first time.	Moderate
4	Changing the password	The instructors should be able to change their current password.	Moderate
5	Registering a course	The instructors should be able to register courses that they will be instructing in the semester. These courses should include information such as their course id, number of expected students and the type of the course. The system should automatically register any needed laboratories and tutorials based on the type of the course.	High
6	Adding unavailability	The instructors should be able to add their unavailability, or in other words, when they are not able to teach in the university. These availabilities should be separated based on the day and hour of the week.	High
7	Updating unavailability	The instructors should be able to update the registered unavailable hours.	Moderate

<b>8</b>	Updating course Information	The instructors should be able to easily update the information for each course they have registered.	Moderate
<b>9</b>	Deleting a course	The instructors should be able to delete any course they registered amongst with any laboratories or tutorials that might be related to it.	High
<b>10</b>	Viewing added courses	The instructors should have the option to view all the courses, laboratories, and tutorials that they have registered, amongst with their information.	Moderate
<b>11</b>	Viewing final schedule	The instructors should be able to view their final weekly schedule that was automatically created by the tool.	High
<b>12</b>	Choosing a laboratory to instruct	The instructors should be able to choose any of the added laboratories so that they can instruct them.	High

Table 4.2 – Instructor Requirements

#### 4.5.2 Interface functional requirements for Administrator users

	Requirement	Description	Priority
<b>1</b>	Logging in	The administrators should be able to login into the system with their credentials.	High
<b>2</b>	Logging out	The administrators should be able to logout of the system from any page.	High
<b>3</b>	Changing the password	The administrators should be able to change their password.	Moderate

<b>4</b>	Registering Instructors	The administrators should be able to register any new instructors into the system.	High
<b>5</b>	Removing Instructors	The administrators should be able to remove any instructor from the system, doing so should remove their access to the system.	Moderate
<b>6</b>	Editing Instructor's information	The administrators should be able to edit any instructor's registered information.	Low
<b>7</b>	Registering more Administrators	Any administrators should be able to register additional administrators who would have the same privileges as them.	Moderate
<b>8</b>	Setting schedule's initial settings and variables	The administrators should be able to set the initial settings and variables of the semester schedule.	High
<b>9</b>	Viewing all courses	The administrators should be able to see all the courses that have been registered by the instructors, this should include all the course's details.	Moderate
<b>10</b>	Editing courses	The administrators should have the option to edit and change any information of any registered course.	Low
<b>11</b>	Adding course conflicts	The administrators should be able to set any pair of courses as "conflict courses", or in other words, courses that should not be taught at the same time of the week.	High

<b>12</b>	Removing course conflicts	They should also be able to remove any course conflicts that have already been added.	Moderate
<b>13</b>	Setting courses as parallel	The administrators should be able to set any pair of courses as “parallel courses”, courses that should be taught at the exact same time during the weekly schedule.	High
<b>14</b>	Removing parallel courses	They should also be able to remove any parallel courses that have already been added.	Moderate
<b>15</b>	Running scheduling tool	The administrators should have the option to run the automated scheduling tool to generate the weekly semester schedule.	High
<b>16</b>	Viewing all the instructors’ schedules	The administrators must have the option to view all the created schedules of all the instructors that are registered in the system.	High
<b>17</b>	Creating course combination schedules	The administrators should be able to create schedules that consist of a chosen set of courses.	High

Table 4.3 – Administrator Requirements

### 4.5.3 Non-functional requirements

	Requirement	Description	Priority
<b>1</b>	Performance	The tool should produce the schedules withing a reasonable amount of time (less than an hour).	High
<b>2</b>	Scalability	The tool should be able to at least handle the needs that are currently present in the department.	Moderate

3	Availability	The system should be available 24/7 with minimal downtime.	Moderate
4	Usability	The front-end interface must be easy to use and learn.	High
5	Reliability	The tool should be reliable with a low error rate.	High
6	Security	The system should not have any vulnerabilities and therefore be secure against any attack.	High
7	Error prevention	The system should be able to detect any inconsistencies or invalid inputs and handle them.	High

Table 4.4 – None-functional Requirements

## 4.6 Use cases

In the below diagrams, some use cases for the two types of users are presented.

### Instructor's use case diagram:

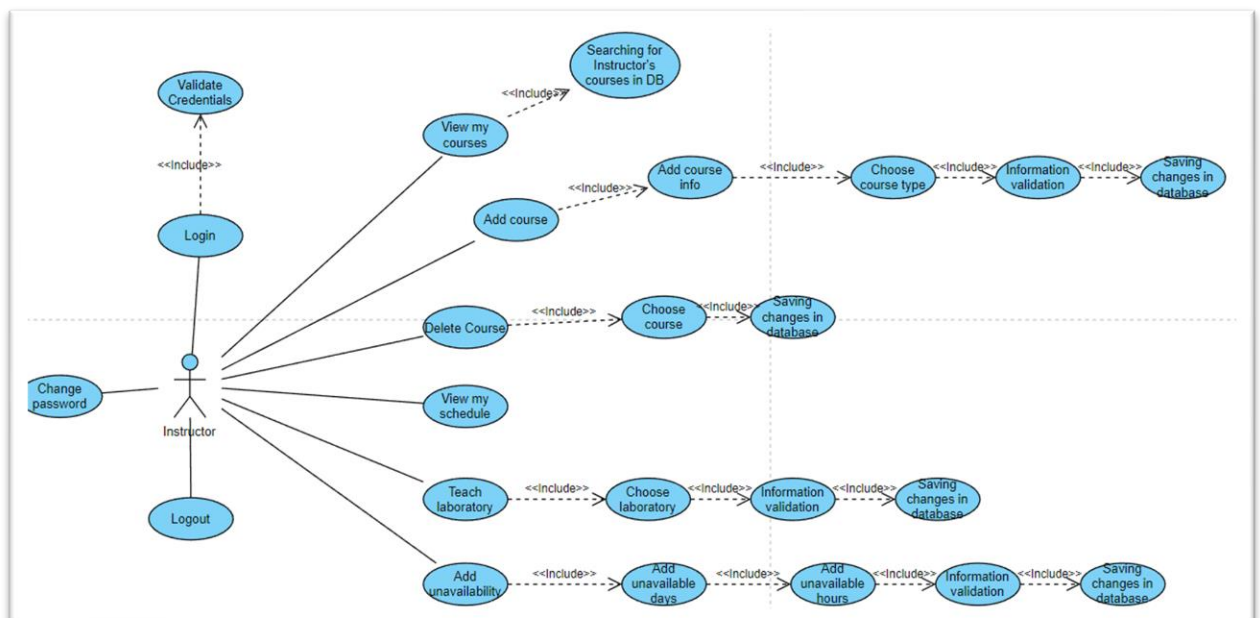


Figure 4.1 – Instructor use case diagram

This diagram displays the functionalities and the sub-functionalities that are available to the instructor users. For instance, the add course functionality includes adding the course's info like course id and course type, then the validation of the submitted information to insure proper input, and finally the storing procedure in the database.

### Administrator's use case diagram:

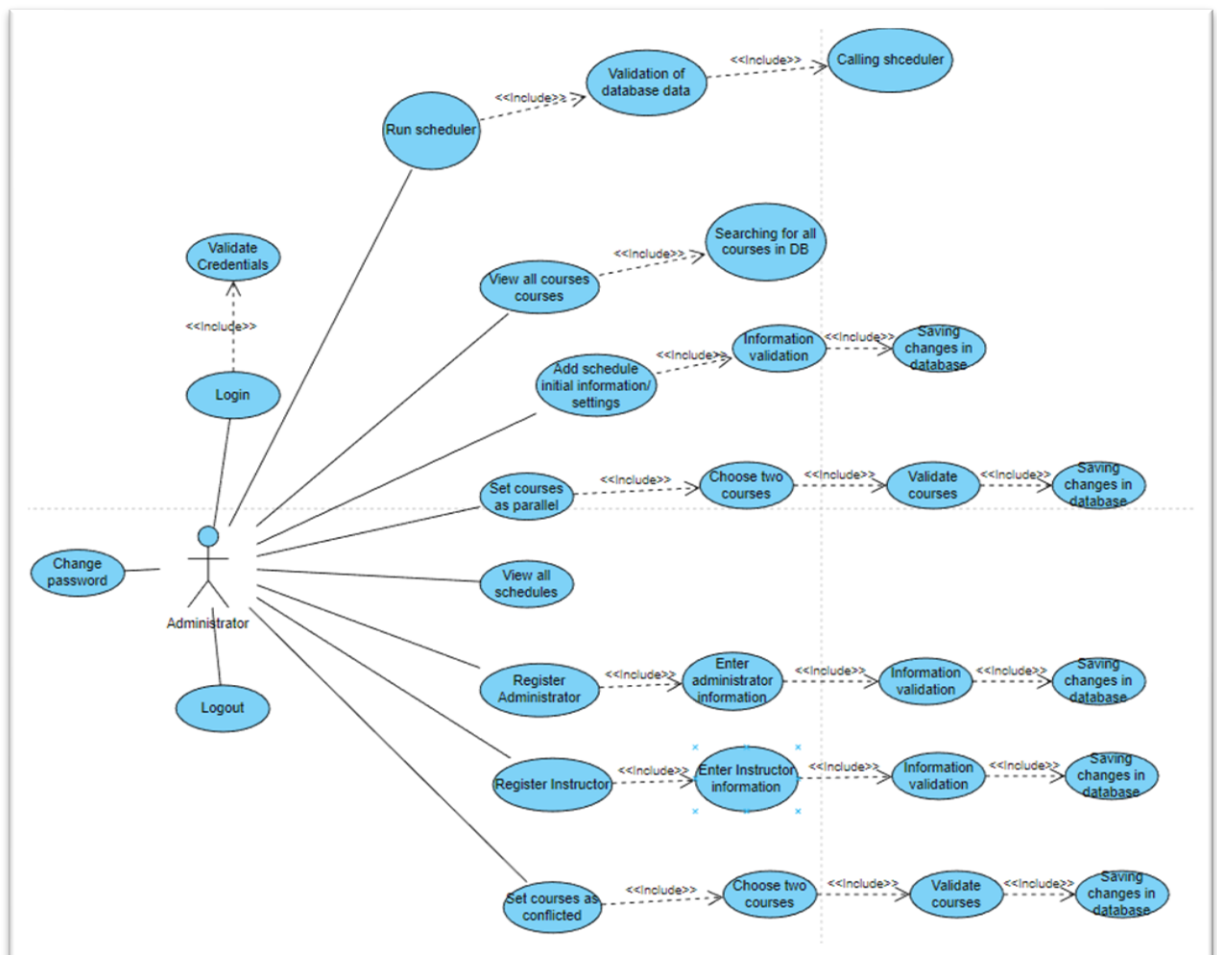


Figure 4.1 – Administrator use case diagram

The functionalities of the Administrator can be seen in the above diagram. For example, the user can set two courses as conflicting courses. To do this, they must first access the “Set courses as conflicted” functionality, which in our implementation is a web page. This action also includes choosing two courses from the registered courses list, subsequently the course combination is checked in the “Validate courses” phase and then the action is saved in the database automatically.

## Chapter 5

### 5 Design

---

#### 5.1 System architecture

#### 5.2 Database design

##### 5.2.1 Tables

##### 5.2.2 Relational Schema

#### 5.3 Front-end design

##### 5.3.1 Navigation pattern

##### 5.3.2 Common pages

##### 5.3.2 Instructor pages

##### 5.3.4 Administrator pages

---

### 5.2 System architecture

The Administrators and Instructors users interact only with the web-application of the tool, to manage their account and to specify the variables and requirements for the schedule. These variables are all communicated from the interface directly to the database without the need for the user's intervention. When the time comes for executing the Minizinc solver, a python script is called which reads the database and generates the required input files to be used by the solver. Then, the solver is called, and the output is managed by another python script, this final program produces easy to read schedules for the users to access. This entire process can be visualized in the below diagram.



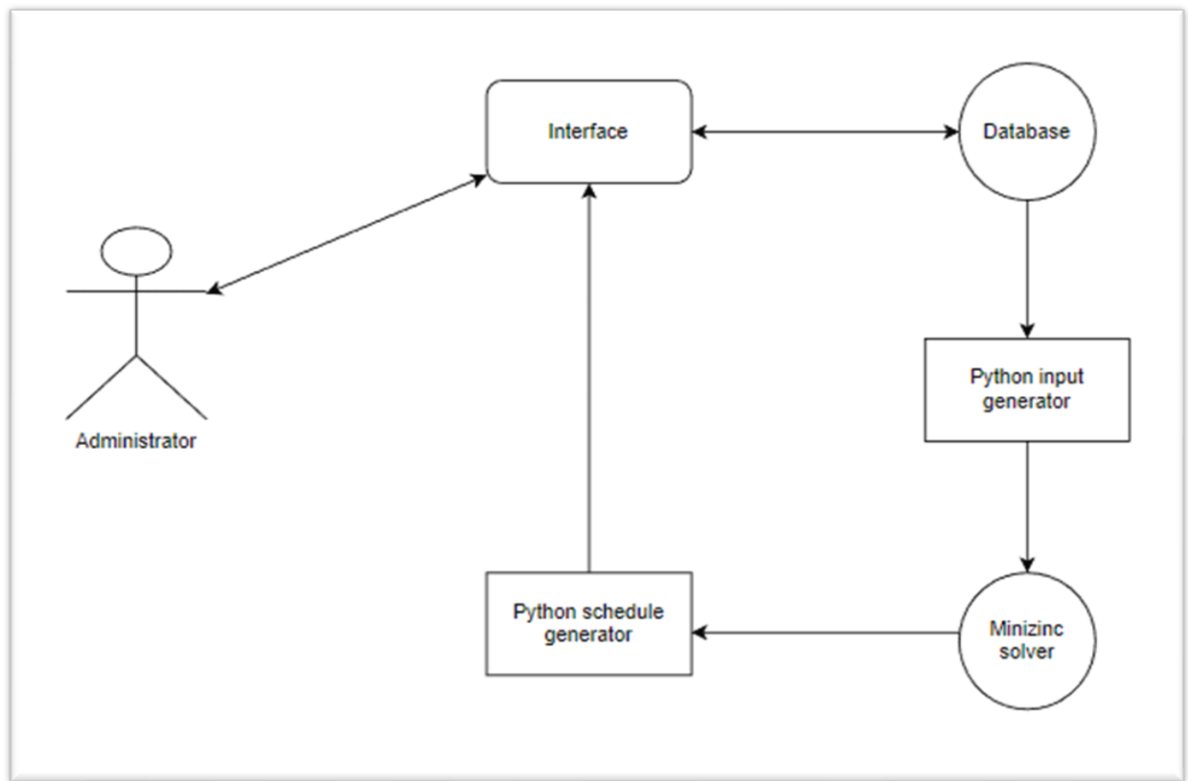


Figure 5.1 – System Overview

### 5.3 Database design

The intent of the designed database is to be used for storing the data needed for the front-end interface to function, and the data to be used by the Minizinc solver. This data includes the users of the system, the unavailability of the instructors, the courses and their details, and the dependencies between the different courses.

The chosen database is MySQL, because of its many advantages that have been discussed in **Chapter 3.3.2.3**, and especially its ease of use.

#### 5.3.1 Tables

The database consists of 8 tables. In the next section, the tables and their attributes are presented.

Table **admins**:

This table contains information about the administrators of the system, mainly their unique usernames and their hashed passwords.

Key: admin\_username

Attribute	Type	Comments
admin_username	string	A unique username.
<b>admin_password</b>	string	A hashed password.
<b>First_login</b>	integer	An int of value 1 or 0, to represent if the user needs to create a password or not.

Table 5.1 – Database Table admins

Table **instructors**:

Similarly, the instructors table contains information for each instructor as shown below. The first\_login attribute is used by the system to determine if the instructor has already set a password or not.

Key: instructor\_username

Attribute	Type	Comments
<b>instructor_username</b>	string	A unique username.
<b>instructor_password</b>	string	A hashed password.
<b>first_login</b>	integer	An integer of value 1 or 0, to represent if the user needs to create a password or not.
<b>name</b>	string	
<b>email</b>	string	A unique email.
<b>surname</b>	string	
<b>admin_username</b>	string	The username of the admin that registered the instructor.

Table 5.2 – Database Table instructors

Table **courses**:

This table is utilized to save the information of each course in the schedule.

Key: course\_code

Attribute	Type	Comments
<b>course_code</b>	string	A unique code for the course.
<b>course_type</b>	integer	An integer value which represents the type of the course (lecture, lab, tutorial etc.).
<b>max_students</b>	integer	The maximum number of students expected to enroll in the course.
<b>instructor_username</b>	string	The username of the instructor that will instruct this course.
<b>num_of_labs</b>	string	The number of sections/iterations that this course will have. (Will be determined by the system and not be given as user input).

Table 5.3 – Database Table courses

Table **conflicts**:

This table specifies the courses that should not be taught during the same time slot. This is achieved by adding two different courses that are not taught by the same instructor to the table.

Key: course\_code1 and course\_code2

Attribute	Type	Comments
<b>course_code1</b>	string	The course code of the first code.
<b>course_code2</b>	string	The course code for the second course.

Table 5.4 – Database Table conflicts

Table **parallel**:

Similarly, it defines the courses which should be taught at the same time slot.

Key: course\_code1 and course\_code2

Attribute	Type	Comments
<b>course_code1</b>	string	The course code of the first code.
<b>course_code2</b>	string	The course code for the second course.

Table 5.5 – Database Table parallel

Table **instructor\_unable\_hours**:

It contains the unavailability of the instructors based on the day and hour. The day attributes will take one of the below values.

- 1 – representing Mondays and Thursdays
- 2 – representing Tuesdays and Fridays

Key: instructor\_username, day, and hour

Attribute	Type	Comments
<b>instructor_username</b>	string	
<b>hour</b>	Float	The start hour in which the course is unavailable.
<b>day</b>	integer	The day in which this availability is valid.

Table 5.6 – Database Table instructor\_unable\_hours

Table **instructor\_unable\_hours\_Wednesday**:

Similarly, this table represents the unavailability of the instructors but on Wednesdays only.

Key: instructor\_username and hour

Attribute	Type	Comments
<b>instructor_username</b>	string	
<b>Hour</b>	float	The start hour in which the course is unavailable.

Table 5.7 – Database Table instructor\_unable\_hours\_Wednesday

Table **additional\_info**:

This table is used to store some additional information about the overall schedule.

Key: none

Attribute	Type	Comments
<b>Lectures_capacity</b>	integer	The capacity of the lecture in the department.
<b>laboratories_capacity</b>	integer	The capacity of the laboratories in the department.
<b>num_of_labs</b>	integer	The number of available laboratories.

<b>num_of_lecture_rooms</b>	integer	The number of available lecture halls in the department.
-----------------------------	---------	--

Table 5.7 – Database Table additional\_info

The above tables, except for tables admins and instructors, are used for the purpose of creating the semester schedule. The latter two are also used for the management of the interface's users.

### 5.3.2 Database Relational Schema

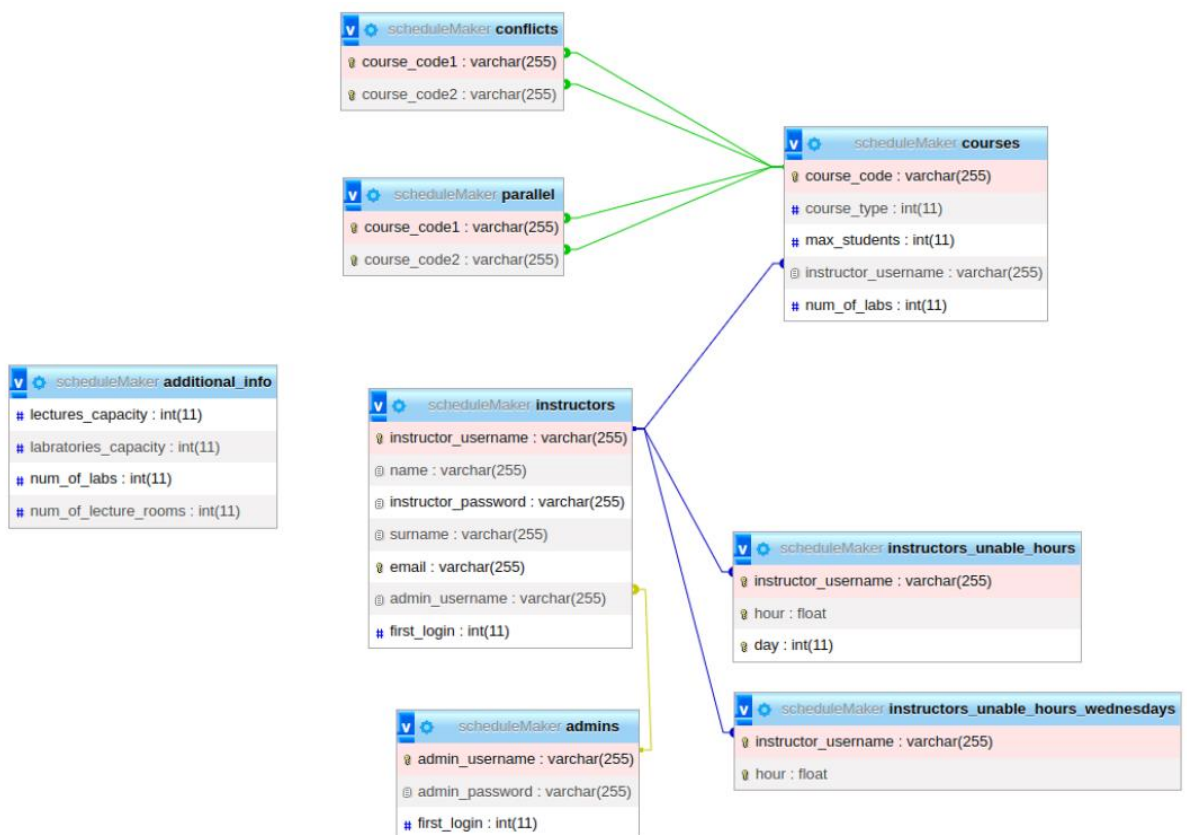


Figure 5.1 – Database Relational Schema

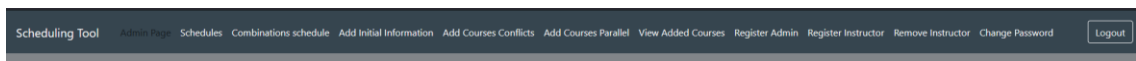
This diagram presents the relational shcema between the different tables. Each admin is connected to the instructor that they registered. Each instructor is connected to the courses that they teach, and to their unavailabilities tables. Table additional\_info is not connected to the rest of the database tables.

## 5.4 Front-end design

The interface, as per the requirements, was designed to be as simple to use and as efficient as possible. For this reason, a modern and minimalistic design was chosen. This design ensures that the users can get familiar with its functionality quickly, reducing the learning curve and enabling them to efficiently achieve their goals.

### 5.4.1 Navigation pattern

The Global navigation pattern was chosen to be used for the navigation between the different pages. With this pattern, a navigation bar, which is always present at the top of each page, displays **all** the pages that can be navigated to at any given point. This pattern is especially fitted for use in simple, small sized websites that do not have many pages. Its main advantages are its simplicity and usability.



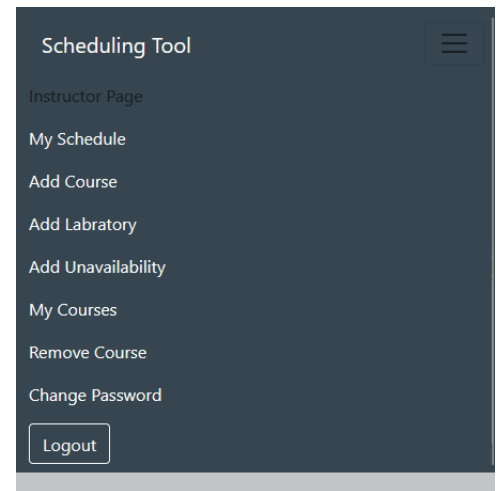
The background color of the navigation bar is dark blue, and the pages which can be navigated to are white. This is no coincidence, the colors were picked to have good contrast, to aid in better readability and attract the attention of the user. In addition, the color of the page which the user is situated on is black. This was designed so that the users can always know on which page they are located.

On the very left of the navigation bar, the main page of the interface is always displayed and on the very right, a logout button is always present. This button only appears when the user is logged in, and when pressed, the user is taken back to the login page.

Mobile view:



The navigation bar is suitable for a mobile view too. When there isn't enough space on a browser to fit all the pages, three lines appear on the right representing a menu. When clicked, the pages are shown in a dropdown list.



### 5.4.2 Common pages

Home page:



This is the main page which the user lands on when first accessing the interface. Its main purpose is to be an informatory page about the interface.

Login page:

The users can only login into the system after they have been registered by an administrator. This means that the users cannot register themselves, this task is left to the administrators.

On this page, the user can sign in using their unique username and password. If the user signs in for the first time, then they will be directed to a different page, where they will create their own password.

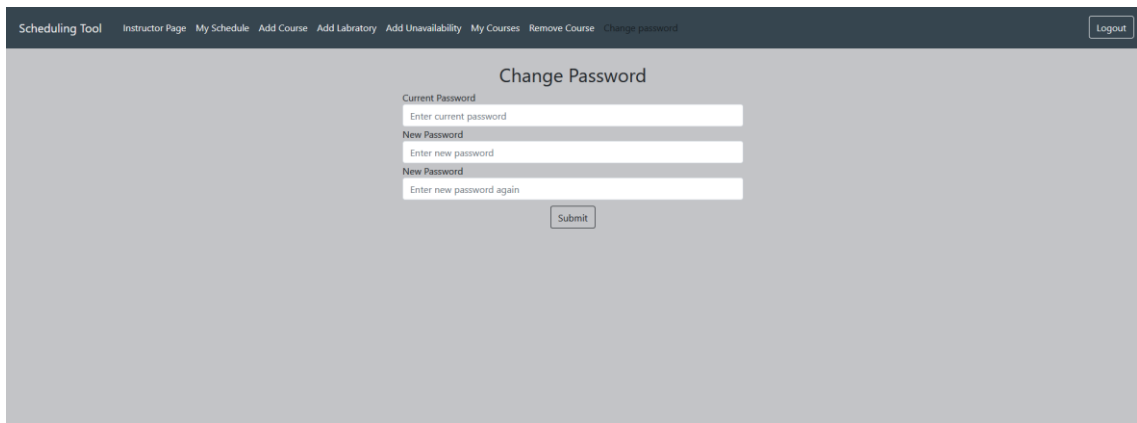
When the mouse is hovered over any button in the interface, it changes color, turning to white. This gives the website a responsive character.

### Set password page:

This page is only used once by the users, on their first login. It consists of a title and a form. Submitting the new password will direct the users to the Instructor or Administrator page.



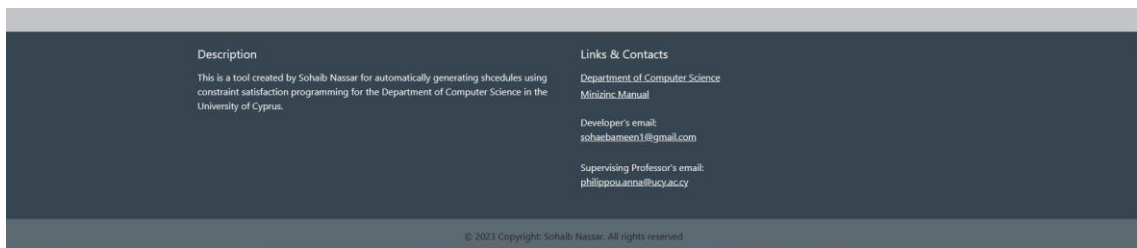
## Change password page:



The screenshot shows the 'Change Password' page of a web application. At the top, there is a dark navigation bar with links: 'Scheduling Tool', 'Instructor Page', 'My Schedule', 'Add Course', 'Add Laboratory', 'Add Unavailability', 'My Courses', 'Remove Course', and 'Change password'. A 'Logout' button is in the top right corner. The main content area has a light gray background. The title 'Change Password' is centered. Below it, there are three input fields: 'Current Password' (with placeholder 'Enter current password'), 'New Password' (with placeholder 'Enter new password'), and another 'New Password' field (with placeholder 'Enter new password again'). A 'Submit' button is located below the third input field.

Similarly, this can be used by any user to change their password for whatever reason. With the only difference being that the user needs to insert the current password as a requirement to ensure security.

## Footer:

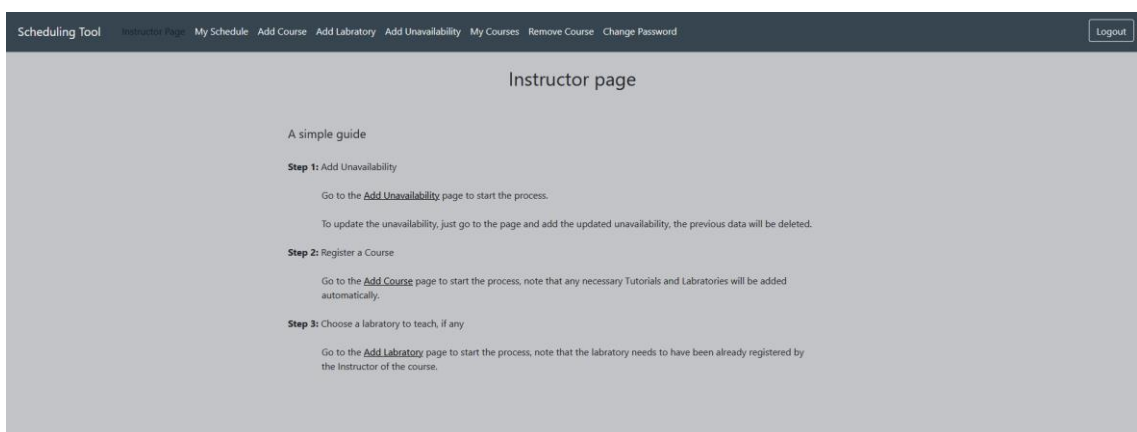


The screenshot shows the footer of the web application. It is divided into two columns. The left column is titled 'Description' and contains the text: 'This is a tool created by Sohaib Nasser for automatically generating schedules using constraint satisfaction programming for the Department of Computer Science in the University of Cyprus.' The right column is titled 'Links & Contacts' and contains links for 'Department of Computer Science', 'Minicourse Manual', 'Developer's email: sohaibnasser1@gmail.com', and 'Supervising Professor's email: philippou.anna@ucy.ac.cy'. At the bottom, there is a copyright notice: '© 2023 Copyright: Sohaib Nasser. All rights reserved.'

This footer exists on the bottom of each page. Its primary goal is to provide some useful links and contact information.

## 5.4.3 Instructor pages

### Instructor page:



The screenshot shows the 'Instructor page' of a web application. At the top, there is a dark navigation bar with links: 'Scheduling Tool', 'Instructor Page', 'My Schedule', 'Add Course', 'Add Laboratory', 'Add Unavailability', 'My Courses', 'Remove Course', and 'Change Password'. A 'Logout' button is in the top right corner. The main content area has a light gray background. The title 'Instructor page' is centered. Below it, there is a section titled 'A simple guide'. This section contains three steps: 'Step 1: Add Unavailability' (with instructions to go to the 'Add Unavailability' page and update the unavailability), 'Step 2: Register a Course' (with instructions to go to the 'Add Course' page), and 'Step 3: Choose a laboratory to teach, if any' (with instructions to go to the 'Add Laboratory' page).

On this page, the instructor is given a simple guide on how to use the interface.

My schedule page:

Time	Monday	Tuesday	Wednesday	Thursday	Friday
9:00					
9:30					
10:00					
10:30					
11:00					
11:30					
12:00					
12:30					
13:00					
13:30					
14:00					
14:30					
15:00					
15:30					
16:00			EPL211_T		
16:30					
17:00					
17:30					
18:00					
18:30		EPL211			EPL211
19:00					
19:30					
20:00					
20:30					

On this page, each Instructor can view their final shedule once it has been created.

The schedule is in a form of a ready to download pdf.

Add course page:

Course code  
EPL100

Type of course  
1 Lecture ONLY

Maximum number of students  
35

Submit and continue

On this page, the instructors can register a course that they are going to teach in the shcedule. For the type of course, a dropdown list includes all the course types that are available in the department.

A small arrow to the right is used as a metaphor to indicate the existence of the dropdown list.

### Add laboratory page:

The add laboratory page is used by the instructors to choose a laboratory of a course that has already been registered, so that they instruct it. This feature satisfies **requirement 10** of the schedule requirements. Similarly to the add course page, the available laboratories are shown in the form of a dropdown list.

### Add unavailability page:

This page is used to determine the unavailability of the instructor during the week. The functionality takes place on multiple pages. This separation was implemented so that it does not become overwhelming for the users, and to make the process more clear-cut and straightforward.

The first page, as shown above, asks the instructors to determine if they are unavailable for an **entire day**. This is done by utilizing a checklist, where multiple options can be checked. After clicking “Submit and continue”, the instructors will be taken to the below page. Option “I’m available every day”, directs the user to the next page.

Scheduling Tool   Instructor Page   My Schedule   Add Course   Add Laboratory   Add Unavailability   My Courses   Remove Course   Change Password   Logout

Specify hours and days where teaching is not possible

On which of the following hours on Monday/Thursday you cannot teach?

You can choose multiple

- ☐ 09:00-10:30
- ☐ 10:30-12:00
- ☐ 12:00-13:30
- ☐ 13:30-15:00
- ☐ 15:00-16:30
- ☐ 16:30-18:00
- ☐ 18:00-19:30
- ☐ 19:30-21:00
- ☐ I'm available the entire day

Submit and continue

On this page, the users are asked to choose the time slots on Mondays and Thursdays where they are busy.

- ☐ 09:00-10:30
- ☒ 10:30-12:00
- ☐ 12:00-13:30
- ☐ 13:30-15:00
- ☐ 15:00-16:30
- ☒ 16:30-18:00
- ☒ 18:00-19:30
- ☐ 19:30-21:00
- ☐ I'm available the entire day

Submit and continue

They have the option to choose multiple timeslots, or no timeslot at all. When clicking on the button, the process continues, and they are taken to a new page which will ask about the unavailability on Wednesday.

Scheduling Tool   Instructor Page   My Schedule   Add Course   Add Laboratory   Add Unavailability   My Courses   Remove Course   Change Password   Logout

### Specify hours and days where teaching is not possible

On which of the following hours on Wednesday you cannot teach?  
You can choose multiple

<input type="checkbox"/>	09:00-10:00
<input type="checkbox"/>	10:00-11:00
<input type="checkbox"/>	11:00-12:00
<input type="checkbox"/>	12:00-13:00
<input type="checkbox"/>	13:00-14:00
<input type="checkbox"/>	14:00-15:00
<input type="checkbox"/>	15:00-16:00
<input type="checkbox"/>	16:00-17:00
<input type="checkbox"/>	17:00-18:00
<input type="checkbox"/>	18:00-19:00
<input type="checkbox"/>	19:00-20:00
<input type="checkbox"/>	20:00-21:00
<input type="checkbox"/>	I'm available the entire day

Submit and continue

This page was deemed necessary to have different timeslots from the rest of the days because of the schedule's nature in the department.

Scheduling Tool   Instructor Page   My Schedule   Add Course   Add Laboratory   Add Unavailability   My Courses   Remove Course   Change Password   Logout

### Specify hours and days where teaching is not possible

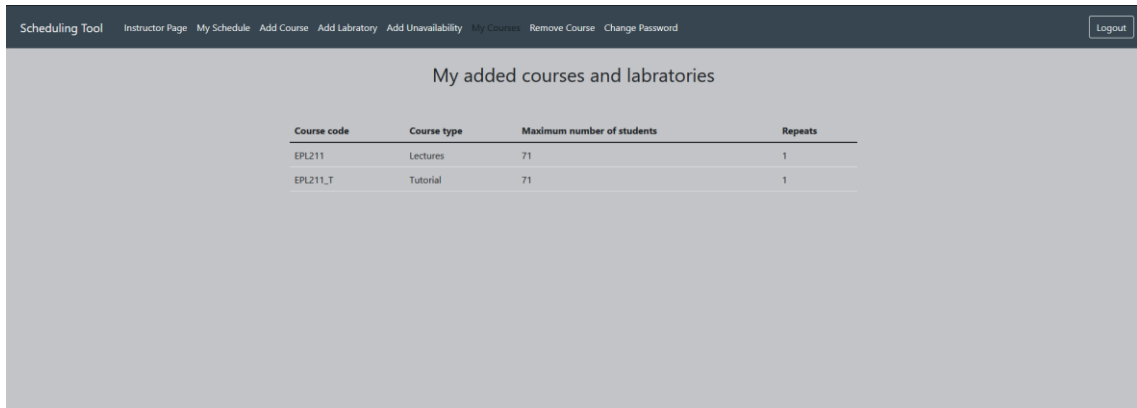
On which of the following hours on Tuesday/Friday you cannot teach?  
You can choose multiple

<input type="checkbox"/>	09:00-10:30
<input type="checkbox"/>	10:30-12:00
<input type="checkbox"/>	12:00-13:30
<input type="checkbox"/>	13:30-15:00
<input type="checkbox"/>	15:00-16:30
<input type="checkbox"/>	16:30-18:00
<input type="checkbox"/>	18:00-19:30
<input type="checkbox"/>	19:30-21:00
<input type="checkbox"/>	I'm available the entire day

Submit

Finally, the process is finished when the users inform about their unavailability on Tuesdays and Fridays.

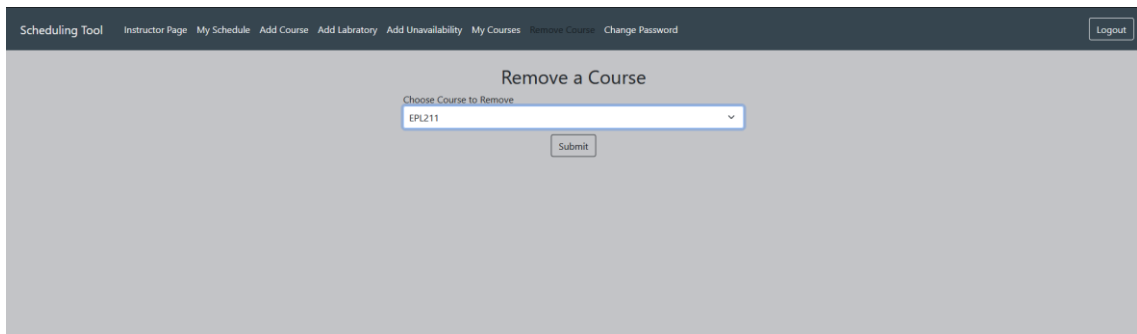
## My courses page:



Course code	Course type	Maximum number of students	Repeats
EPL211	Lectures	71	1
EPL211_T	Tutorial	71	1

On this page, the instructors can see the courses which they will teach, including some additional information, such as the type of the course, the maximum number of students to be registered and the number of sections of the course.

## Remove course page:



Remove a Course

Choose Course to Remove

EPL211

Submit

This is a very simple page with one drop down list, it is used by the instructors to remove any course. Removing a course also removed all of its registered laboratories and tutorials.

## 5.4.4 Administrator pages

Administrator page:

The screenshot shows the 'Select a solver version to create the Schedule' page. At the top is a navigation bar with links: 'Scheduling Tool', 'Admin Page', 'Schedules', 'Combinations schedule', 'Add Initial Information', 'Add Courses Conflicts', 'Add Courses Parallel', 'View Added Courses', 'Register Admin', 'Register Instructor', 'Remove Instructor', 'Change Password', and a 'Logout' button. The main heading is 'Select a solver version to create the Schedule'. Below it are three radio buttons: 'Version 1' (selected), 'Version 2', and 'Version 3'. Under each version is a brief description: 'Version 1: Generates a timetable that fulfills all constraints provided by instructors and administrators.', 'Version 2: Satisfies all given requirements but also ensures that iterations of a laboratory are taught on the same day.', and 'Version 3: Not only satisfies all given requirements but also ensures that iterations of a laboratory are taught on the same day in succession.' A 'Create Schedule' button is at the bottom, followed by the text 'All solvers utilize the [MiniZinc](#) tool.'

This is the page that the administrators land on once they sign in. Its main functionality is to initiate the shedule solver. The administrators are given the option to choose between different versions of the solver, which have slight differences in the way they handle the schedule's requirements. These verions are promptly explained on the page.

Schedules page:

The screenshot shows the 'View Instructor's shcedule' page. It has the same navigation bar as the previous page. The main heading is 'View Instructor's shcedule'. Below it is a 'Select Instructor' label and a white drop-down menu with a downward arrow.

This is a close-up of the 'Select Instructor' dropdown menu. It shows a list of instructor IDs: giorgos0, iakwvos0, ilias0, kakas0 (highlighted), keravnou0, maria0, marios0, markoullis0, mavronikolas0, mixalis0, mulonas0, mwusis0, pallis0, panagi0, pasxalidis0, pattixis0, paulos0, pieris0, and pitsillidis0. At the bottom of the list, there is a partial view of 'University of Cyprus'.

To view the created schedule, the administrator is given a drop-down list that contains all of the registered instructors. Once an instructor is selected, their schedule is displayed in the form of a pdf on the right side of the page. To change the schedule, all the administrator has to do is choose another instructor from the list. These PDFs can be easily downloaded.

Scheduling Tool Admin Page Schedules Combinations schedule Add Initial Information Add Courses Conflicts Add Courses Parallel View Added Courses Register Admin Register Instructor Remove Instructor Change Password Logout

### View Instructor's shcedule

Select Instructor

iliad0

Time	Monday	Tuesday	Wednesday	Thursday	Friday
8:00					
9:30					
10:00					
10:30					
11:00			EPL326, T		
11:30					
12:00					EPL451
12:30	EPL451				
13:00					
13:30			EPL451, T		
14:00					
14:30					
15:00					
15:30					
16:00					
16:30					
17:00					
17:30					
18:00					
18:30					
19:00					
19:30					
20:00	EPL326				EPL326
20:30					

## Courses combinations schedule page:

Scheduling Tool Admin Page Schedules Combinations schedule Add Initial Information Add Courses Conflicts Add Courses Parallel View Added Courses Register Admin Register Instructor Remove Instructor Change Password Logout

### Create a courses combination schedule

Select courses

Hold Cntrl + Select course

- EPL133
- EPL202
- EPL211
- EPL221
- EPL222
- EPL231
- EPL236
- EPL325
- EPL326
- EPL341
- EPL420
- EPL421
- EPL422
- EPL425
- EPL426
- EPL431
- EPL441
- EPL448
- EPL449
- EPL451
- EPL646
- EPL649
- EPL656
- EPL657

Generate PDF

Select courses

Hold Cntrl + Select course

- EPL032\_0
- EPL034\_1
- EPL034\_2
- EPL036
- EPL042
- EPL121
- EPL131
- EPL133
- EPL202
- EPL211
- EPL221
- EPL222
- EPL231
- EPL236
- EPL325
- EPL326
- EPL341
- EPL420
- EPL421
- EPL422
- EPL425
- EPL426
- EPL431
- EPL441

Generate PDF

On this page, the administrators can view a schedule that consists only of a set of courses. For example, to see if a set of courses that must be taken by a group of students do not overlap.

To do this, the administrator must select the courses from the given list. Then, once the Generate PDF button is pressed, the PDF schedule is created and is shown on the right side of the screen.

In this example we have courses EPL211, EPL325, EPL326 and EPL341.



Scheduling Tool Admin Page Schedules Combinations schedule Add Initial Information Add Courses Conflicts Add Courses Parallel View Added Courses Register Admin Register Instructor Remove Instructor Change Password Logout

### Create a courses combination schedule

#### Select courses

Hold Ctrl + Select course

- DSC511
- EPL002
- EPL032\_1
- EPL032\_2
- EPL032\_3
- EPL032\_5
- EPL032\_6
- EPL034\_1
- EPL034\_2
- EPL036
- EPL042
- EPL121
- EPL131
- EPL133
- EPL202
- EPL211
- EPL221
- EPL222
- EPL231
- EPL236
- EPL225
- EPL326
- EPL341
- EPL420

Generate PDF

Courses: EPL326, EPL341, EPL211, EPL326

Time	Monday	Tuesday	Wednesday	Thursday	Friday
8:00			EPL341_1		
8:30					
9:00			EPL341_2		
9:30					
10:00					
10:30					
11:00					
11:30					
12:00					
12:30	EPL341			EPL341	
13:00					
13:30			EPL341_2	EPL341	
14:00	EPL341				
14:30					
15:00					
15:30			EPL341_2		EPL341_1
16:00			EPL341_2		
16:30			EPL341_1		EPL341_1
17:00			EPL341_1		EPL341_1
17:30					
18:00					
18:30	EPL341		EPL341_2	EPL341	EPL341_2
19:00					
19:30			EPL341		EPL341_2
20:00			EPL341_2		EPL341_2
20:30					

The process can be repeated by selecting a new set of courses and a new PDF would be generated.

Add initial information page:

Scheduling Tool Admin Page Schedules Combinations schedule Add Initial Information Add Courses Conflicts Add Courses Parallel View Added Courses Register Admin Register Instructor Remove Instructor Change Password Logout

### Enter Initial Information

The number of available lecture halls/rooms

The number of available laboratories

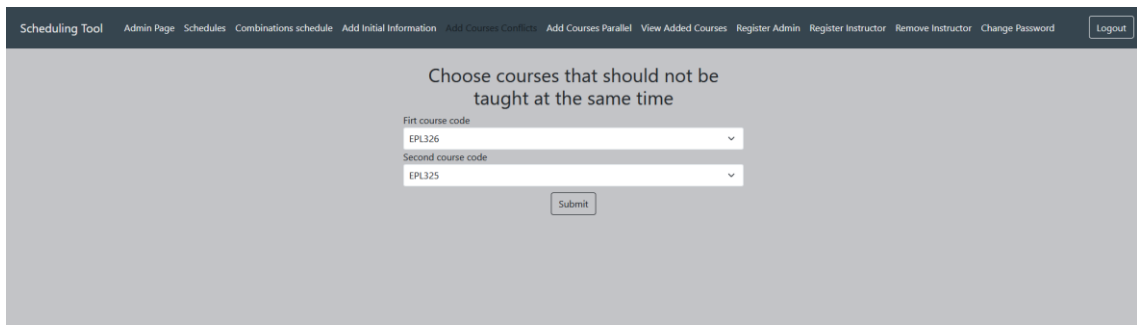
The maximum capacity of the lecture halls

The maximum capacity of the laboratories

Submit

The administrators must use this page to set some variables that are needed for the schedule generator to function properly and produce a valid schedule. The data that is submitted in this form is saved in the additional\_information table in the database, as explained in **Chapter 5.3.1**.

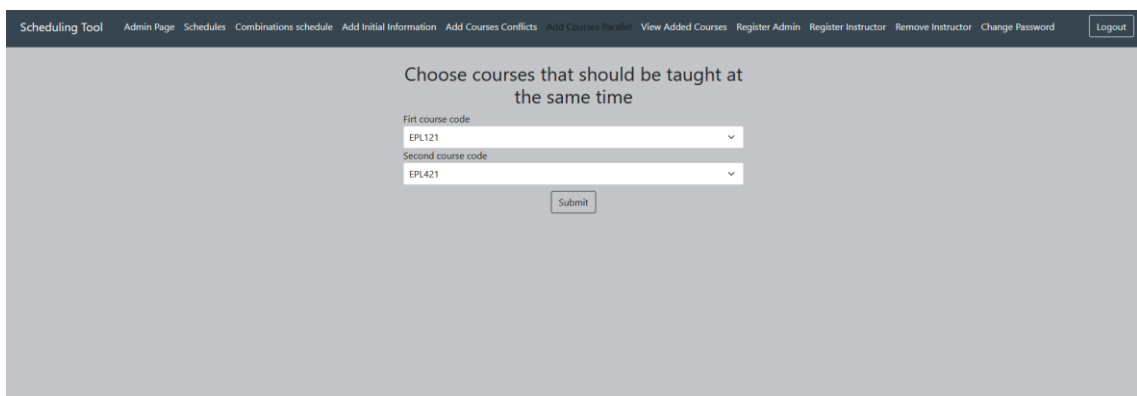
## Add courses conflicts:



The screenshot shows a web application interface for a scheduling tool. At the top, there is a dark navigation bar with the following links: 'Scheduling Tool', 'Admin Page', 'Schedules', 'Combinations schedule', 'Add Initial Information', 'Add Courses Conflicts', 'Add Courses Parallel', 'View Added Courses', 'Register Admin', 'Register Instructor', 'Remove Instructor', 'Change Password', and a 'Logout' button. The main content area has a light gray background and contains the heading 'Choose courses that should not be taught at the same time'. Below this heading are two dropdown menus. The first is labeled 'First course code' and has 'EPL326' selected. The second is labeled 'Second course code' and has 'EPL325' selected. A 'Submit' button is located below the second dropdown menu.

This page is only available to the administrators. It functions as a way for the administrators to define courses that should not be scheduled to be taught at the same time. This for example can be used for courses that a group of students must take in the same semester, therefore the courses should not have any conflicts.

## Add courses parallel page:



The screenshot shows a web application interface for a scheduling tool, similar to the one above. The navigation bar is identical. The main content area has a light gray background and contains the heading 'Choose courses that should be taught at the same time'. Below this heading are two dropdown menus. The first is labeled 'First course code' and has 'EPL121' selected. The second is labeled 'Second course code' and has 'EPL421' selected. A 'Submit' button is located below the second dropdown menu.

Similarly, this page is used to set combinations of courses that should be scheduled to be taught at the same day and hour. This is done by using a form which contains two dropdown lists that have all of the registered courses as options.

## View added courses page:

Scheduling Tool	Admin Page	Schedules	Combinations schedule	Add Initial Information	Add Courses Conflicts	Add Courses Parallel	View Added Courses	Register Admin	Register Instructor	Remove Instructor	Change Password	Logout
-----------------	------------	-----------	-----------------------	-------------------------	-----------------------	----------------------	--------------------	----------------	---------------------	-------------------	-----------------	--------

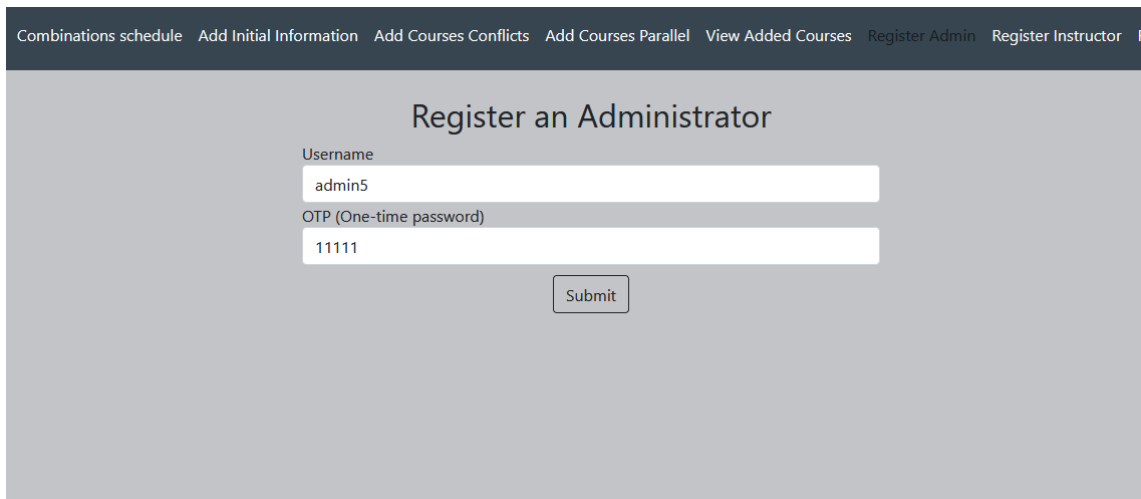
  

Added courses				
Course code	Course type	Maximum number of students	Taught by	Repeats
DSC511	Lecture	60	mwusis0	1
DSC511_T	Tutorial	60	mwusis0	1
EPL002	Lecture	80	iakwvos0	1
EPL002_L	Once a week Lab	80	xajipollas0	3
EPL002_T	Tutorial	80	iakwvos0	1
EPL032_1	Lecture	80	mixalis0	1
EPL032_1_L	Once a week Lab	80	mixalis0	3
EPL032_1_T	Tutorial	80	mixalis0	1
EPL032_2	Lecture	80	maria0	1
EPL032_2_L	Once a week Lab	80	maria0	3
EPL032_2_T	Tutorial	80	maria0	1
EPL032_3	Lecture	100	arguris0	1
EPL032_3_L	Once a week Lab	100	arguris0	3
EPL032_3_T	Tutorial	100	arguris0	1
EPL032_5	Lecture	64	maria0	1
EPL032_5_L	Once a week Lab	64	maria0	2

Course code	Course type	Maximum number of students	Taught by	Repeats
DSC511	Lecture	60	mwusis0	1
DSC511_T	Tutorial	60	mwusis0	1
EPL002	Lecture	80	iakwvos0	1
EPL002_L	Once a week Lab	80	xajipollas0	3
EPL002_T	Tutorial	80	iakwvos0	1
EPL032_1	Lecture	80	mixalis0	1
EPL032_1_L	Once a week Lab	80	mixalis0	3
EPL032_1_T	Tutorial	80	mixalis0	1
EPL032_2	Lecture	80	maria0	1
EPL032_2_L	Once a week Lab	80	maria0	3
EPL032_2_T	Tutorial	80	maria0	1
EPL032_3	Lecture	100	arguris0	1
EPL032_3_L	Once a week Lab	100	arguris0	3
EPL032_3_T	Tutorial	100	arguris0	1
EPL032_5	Lecture	64	maria0	1
EPL032_5_L	Once a week Lab	64	maria0	2

On this page the administrators can monitor the courses that have been registered in the system by the instructors. This includes seeing which instructors registered which courses. In addition, for the laboratories, the administrators have the ability of seeing which laboratories has yet to be taken by a laboratory instructor.

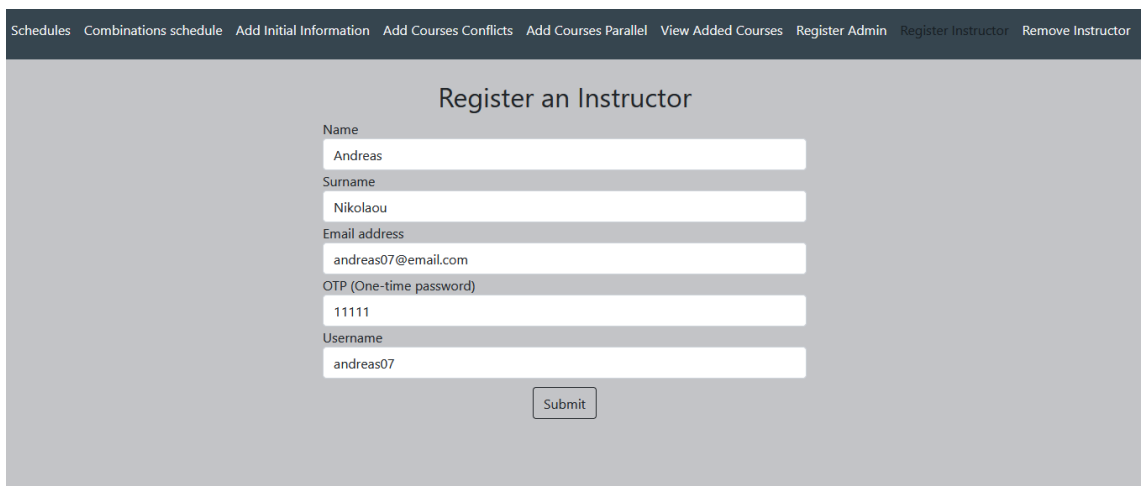
### Register admin page:



The screenshot shows a web application interface with a dark blue navigation bar at the top containing the following links: Combinations schedule, Add Initial Information, Add Courses Conflicts, Add Courses Parallel, View Added Courses, Register Admin, Register Instructor, and a partially visible 't'. The main content area has a light gray background and is titled 'Register an Administrator' in a large, bold, black font. Below the title, there are two input fields: 'Username' with the value 'admin5' and 'OTP (One-time password)' with the value '11111'. A 'Submit' button is positioned below these fields.

On this page, an administrator can register another administrator by giving them a username and an OTP. The OTP is a one time use password which the administrators will use to sign in for the first time.

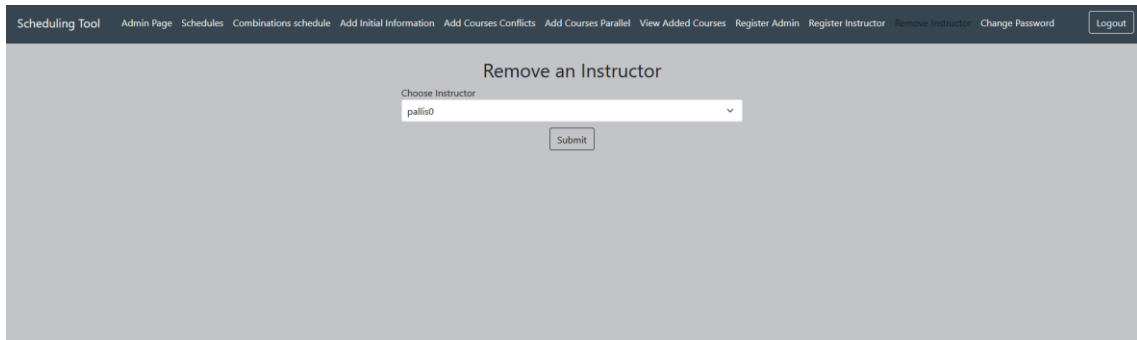
### Register instructor page:



The screenshot shows a web application interface with a dark blue navigation bar at the top containing the following links: Schedules, Combinations schedule, Add Initial Information, Add Courses Conflicts, Add Courses Parallel, View Added Courses, Register Admin, Register Instructor, and Remove Instructor. The main content area has a light gray background and is titled 'Register an Instructor' in a large, bold, black font. Below the title, there are six input fields: 'Name' (Andreas), 'Surname' (Nikolaou), 'Email address' (andreas07@email.com), 'OTP (One-time password)' (11111), 'Username' (andreas07), and a 'Submit' button.

Additional instructors can be registered in the system by the administrators by using this page. It consists of a form that contains some information about the instructor, such as their username, email address and OTP.

## Remove instructor page:



The screenshot shows a web application interface with a dark navigation bar at the top. The navigation bar contains the following links: [Scheduling Tool](#), [Admin Page](#), [Schedules](#), [Combinations schedule](#), [Add Initial Information](#), [Add Courses Conflicts](#), [Add Courses Parallel](#), [View Added Courses](#), [Register Admin](#), [Register Instructor](#), [Remove Instructor](#), [Change Password](#), and a [Logout](#) button. The main content area has a light gray background and is titled "Remove an Instructor". Below the title, there is a label "Choose Instructor" above a white dropdown menu. The dropdown menu currently displays "pallas0" and a downward arrow. Below the dropdown menu is a small "Submit" button.

This is a simple page that is meant to be used by the administrator to remove Instructors from the system. Removing an instructor from the system will also remove any courses that they have already registered.

# Chapter 6

## 6 Implementation

---

### 6.1 Front-end implementation

- 6.1.1 Development environment
- 6.1.2 Development
  - 6.1.2.1 Bootstrap
  - 6.1.2.2 PHP
  - 6.1.2.3 Security
  - 6.1.2.4 Error prevention and handling

### 6.2 Minizinc Implementation

- 6.2.1 Environments
- 6.2.2 Algorithmic logic
- 6.2.3 Minizinc Syntax
- 6.2.4 Parameters and Variables
  - 6.2.4.1 Timeslots parameters
  - 6.2.4.2 Variables
  - 6.2.4.3 Unavailability parameters
  - 6.2.4.4 Course relationships parameters
  - 6.2.4.5 Course details parameters
- 6.2.5 Constraints
- 6.2.6 Versions
- 6.2.7 Solver

### 6.3 Back-end implementation

- 6.3.1 Input
  - 6.3.2 Output
-

## **6.1 Front-end implementation**

### **6.1.1 Development environment**

Microsoft's Visual Studio Code was used as the integrated development environment (IDE) as it offers many useful functionalities while also being simple and easy to use. XAMPP, a local webserver environment, was also utilized for testing and evaluating the interface during the development process.

### **6.2.2 Development**

#### **6.1.2.1 Bootstrap**

Bootstrap is a web development framework which helps in the creation of responsive simple websites. Many of the elements used in the interface, such as buttons, tables, and forms, are sourced from it. An advantage of using Bootstrap is the fact that the elements are easily placed in the code, without the need for adding additional CSS files, something which makes the code easier to read and simpler.

#### **6.1.2.2 PHP**

Because of the application's nature, communication with the database for reading and uploading data was going to be a major part of its functionality. As such, choosing a web-server language which supported an efficient and easy way to connect and transfer information between the webserver and the database was of great importance. For this reason, PHP was the preferred choice for this task, along with its advantages which were discussed in **Chapter 3.3.2.1**.

Whenever a user enters the website, a new **PHP session** is started. This session establishes a connection with the database and generally manages the user's overall interaction with the website. For instance, when a user submits one of the many forms at their disposal, a PHP function is called at the time of submitting the form. This function uses the given connection with the database to delete, insert or update data. This process can be seen in **Appendix 1.3**.

### 6.1.2.3 Security

As it was specified in the requirements phase, the security of the system was to be taken in mind during the development stage.

For this reason, the passwords in the database are not stored in a plaintext format, something which could lead to catastrophic consequences in case of a data breach. Instead, they are stored in an encrypted format. This was achieved by leveraging a hash function which turns the real password of any length into a cypher text of a specified length. The cypher text cannot be used to login to the system in case of a data breach.

Algorithm **sha512** is used for hashing the password stored in the database. Whenever a user is registered or tries to login, the plaintext password is hashed and then the hashvalue is saved or used respectively. This can be seen in **Appendix 1.1**.

To protect against **brute force attacks**, the tool enforces strict password requirements, specifically, the password must have at least one capital character, one small character, one number, one symbol and the total length to be at least 8 characters. These password constraints are checked by employing a JavaScript function, which validates the given input before it is sent to the webserver. This can be seen in **Appendix 1.2**.

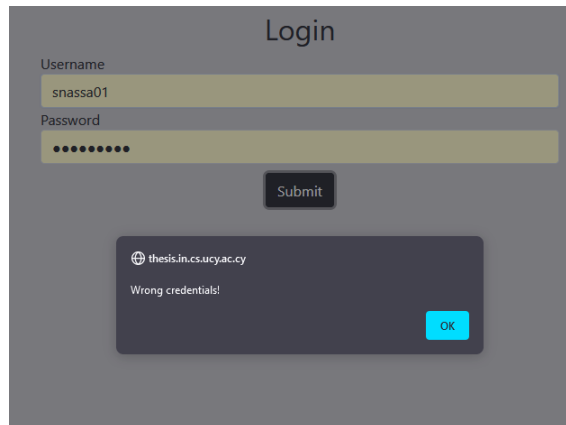
Furthermore, to ensure that pages are not available to users who are not logged in, every time a new page is accessed, the system checks if the user is logged in, and if so, if that user is allowed to view that specific page. For instance, if a user who is logged in as an instructor tries to change the URL into /admin.php, to access the admin page, an “ACCESS DENIED” message is displayed. An example of that can be seen on page admin.php which is provided in **Appendix 1.4**.

### 6.1.2.4 Error prevention and handling

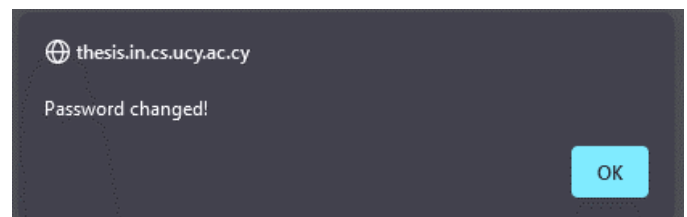
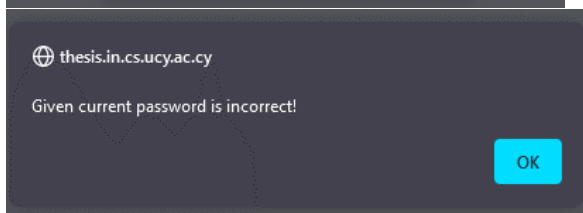
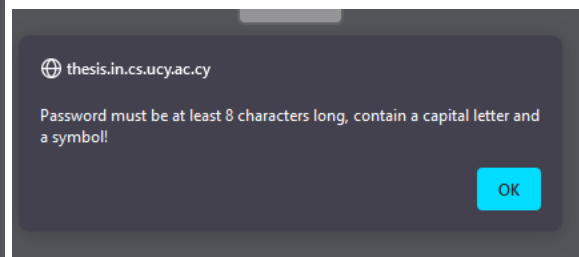
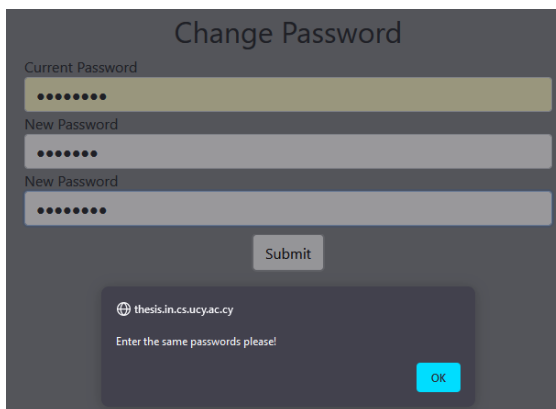
As seen in the Design chapter, the front-end web-application consists of many forms. For this reason, checking the validity of the form inputs is very important for the system’s correct functionality.



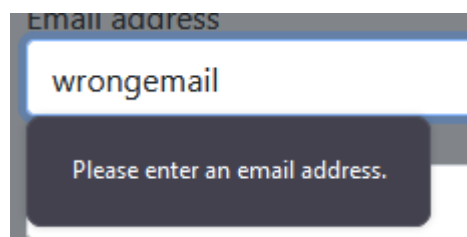
For instance, when the user enters the wrong username or password while trying to log into the application, a warning message is given.



Similarly, when the user sets or changes their password, if the given password does not fulfill the requirements described in chapter **6.1.2.3 Security**, a warning message is displayed, and the password is not accepted.



When an administrator registers instructors, the system checks if the entered email is a valid email and if the given username already exists in the system.



The screenshot shows a web interface for registering an instructor. At the top, there is a navigation bar with links: "Add Initial Information", "Add Courses Conflicts", "Add Courses Parallel", "View Added Courses", and "Register Admin". The main heading is "Register an Instructor". Below this, there are input fields for "Name" (containing "Andreas"), "Surname" (containing "Mariou"), "Email address" (containing "andreas.mariou@email.com"), "OTP (One-time password)" (containing "00000"), and "Username" (containing "marios0"). A modal error message is displayed in the foreground, stating "Instructor already exists!" with an "OK" button. The error message also includes a small icon and the URL "thesis.in.cs.ucy.ac.cy".

If an administrator tried to enter two courses that are taught by the same instructor as “parallel courses” then the system will recognise that and output a warning message.

The screenshot shows a web interface for choosing courses to be taught at the same time. The heading is "Choose courses that should be taught at the same time". Below this, there are two dropdown menus: "First course code" (containing "EPL451") and "Second course code" (containing "EPL326"). A "Submit" button is located below the dropdowns. A modal error message is displayed in the foreground, stating "Please choose courses taught by different instructors!" with an "OK" button. The error message also includes a small icon and the URL "thesis.in.cs.ucy.ac.cy".

Similar error prevention techniques can be found all over the system.

## 6.2 Minizinc Implementation

### 6.2.1 Environment

For the development of the Minizinc model, the Minizinc IDE, which is freely available for use, was utilized. For the purpose of integrating the Minizinc implementation into the final system, the solver had to be executed through command line. Therefore, it had to be installed on a Linux operating system. As Minizinc is not supported on Linux, a software

packaging and deployment system had to be utilized, in this case **Snap** [19]. With Snap installed, the Minizinc snap package was then also installed. Finally, with this configuration set up, the Minizinc solvers can be easily executed from the command line with the option of choosing any of the out of the box solvers.

```
“snap run minizinc --solver chuffed data/inputData.dzn iteration5.mzn -o data/result.txt”
```

The above command runs a Minizinc solver on a given model, “iteration5.mzn” in this case, with the option for the solver set as “Chuffed” and finally with the output saved in file “result.txt”.

### **6.2.2 Algorithmic logic**

As the scheduling problem was defined in **Chapter 4.3.2**, the solver must generate two types of schedules. In the initial iterations, the developed models tried to satisfy these two types of schedules in the same model. After some testing on small to medium data sets, it was obvious that this technique is not ideal. This conclusion came from the fact that it took the solver significantly more time to find a satisfactory solution. For this reason, it was decided that the two schedules would be split into two different models. In other words, there would be a model which would find a solution for the instructors’ schedules and another one that would find a solution for the Room’s schedules.

This dissertation focused on the implementation of the **first schedule type**, since it was deemed to be of higher importance to the department. The room’s schedule is left for future work and for improvement of the system.

In the next sections, the parameters, variables, and constraints that were used for the model are presented and explained.

### 6.2.3 Minizinc syntax

The Minizinc language stands out among other programming languages due to its unique syntax. For this reason, the following section is dedicated to introducing and briefly explaining the syntax.

Firstly, the input data is saved in a 1-D array called `CLASS_DETAILS_INPUT`, which is responsible for cataloguing the information about each course. This array has a length equal to the number of classes in the schedule multiplied by the number of parameters, 5.

```
array[1..((numOfClasses+1)*5)] of int: CLASS_DETAILS_INPUT;
```

The following code section defines the array `CLASS_DETAILS_INPUT` in the “.dzn” input file. For each class, the number of the class, the number of instructor, the number of expected students, the class type, and the number of sections are defined. In the following example, “C1”, “T19”, “60”, “1”, and “1” respectively.

```
CLASS_DETAILS_INPUT = [  
C1, T19, 60, 1, 1,  
C2, T19, 60, 3, 1]
```

Subsequently, the parameters of the model are parsed into the form of 2-D arrays. The following code parses the 1-D array, used for reading data from the input file, into a 2-D array called `CLASS_DETAILS`. The x dimension in this array has the same length as the number of courses in the schedule, `CLASSES` in this case, which is an array that includes all the courses in the schedule. This is achieved with the help of function `array2d`, which parses a 1-D array into a 2-D array.

```
array[CLASSES, 1..5] of int: CLASS_DETAILS = array2d(CLASSES, 1..5,  
CLASS_DETAILS_INPUT);
```

Constraints in Minizinc start with the word “constraint” and end with a semicolon. These constraints can have loops that cover a set of variables. For instance, the bellow constraint example has a for loop that includes all the classes in the schedule that are of type “1”, and are not class “1”, as seen in the “where” statement. Each one of these classes, in other words, all classes of type “Twice a week lectures”, are constrained into being placed exactly 2 times in the overall schedule. This is achieved with the help of function `sum`.

This function calculates the number of times that each class is present in the entire schedule array, as seen in line 2. The if statement in line 3, only checks if the class exists in a specific array slot, and if so, adds value “1” to the overall sum. The below example constraints the values of a 3D array I\_SCHEDULE, which is an array that is used to represent the schedules of all instructors.

```
1. constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 1)(
2.    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS)
3.    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) == 2);
```

## 6.2.4 Parameters and Variables

### 6.2.4.1 Timeslots parameters

The days in the schedule are divided into timeslots, each timeslot represents 60 or 90 minutes. On normal lecture days, these being Monday, Tuesday, Thursday, and Friday, the day is divided into **8 timeslots**, each representing a **90-minute period**. For Wednesdays the day is split into **12 timeslots**, representing **60-minute periods**.

The days were broken up in this way so that the schedule complies with requirement **5** in the **Schedule requirements**, which specifies that tutorials should be taught on Wednesdays, and since the length of tutorials in the department is 60 minutes, the separation was inevitable.

```
3 int: numOfSlots = 8;
4 int: numOfWedSlots = 12;
```

Figure 6.9 Timeslots Parameters

### 6.2.4.2 Variables

The variables in the model represent the schedules of the instructors. These schedules are represented by a 3D array that has a domain of type classes. The x-axis in the array represents the instructors, the y-axis represents the days (5 in total), and the z-axis represents the timeslots.

Similarly, the variables for the Wednesday shcedule are represented by a 2D array, with the only difference being that the days dimension is not present.

```

%THE INSTRUCTORS SCHEDULE OUTPUT FOR DAYS EXCEPT WEDNESDAYS
array[INSTRUCTORS, DAYS, SLOTS] of var CLASSES: I_SCHEDULE;
%THE INSTRUCTORS SCHEDULE OUTPUT FOR WEDNESDAYS
array[INSTRUCTORS, W_SLOTS] of var CLASSES: I_W_SCHEDULE;

```

Figure 6.10 – Schedules Variable Arrays

For representing the lack of a course, since the variable tables are defined to be of type classes, an independent class called E (represented by number 1 in the model) is needed. This class symbolizes an empty slot and is not involved in any of the constraints, making it placable in any time slot. Thus, empty slots in the final solution will be occupied by class E “1”.

#### 6.2.4.3 Unavailability parameters

The unavailability of the instructors is represented by a 3D array for the lecture days and by a 2D array for Wednesdays. These arrays are used to determine when instructors are unavailable for teaching. Value "1" in these arrays indicates that the instructor is unavailable, while the value "0" indicates that the instructor is available.

```

%INPUT FOR INSTRUCTORS UNAVAILABILITY
array [1..(numOfInstructors*5*numOfSlots)] of int: BUSY_INPUT;
array [INSTRUCTORS, SLOTS, DAYS] of int: BUSY = array3d(INSTRUCTORS, SLOTS, DAYS, BUSY_INPUT);
%INPUT FOR INSTRUCTORS UNAVAILABILITY WEDNESDAYS
array [1..(numOfInstructors*numOfWedSlots)] of int: W_BUSY_INPUT;
array [INSTRUCTORS, W_SLOTS] of int: W_BUSY = array2d(INSTRUCTORS, W_SLOTS, W_BUSY_INPUT);

```

Figure 6.11 – Busy Parameter Arrays

#### 6.2.4.4 Course relationships parameters

The DISJUNCT and PARALLEL\_LEC arrays are lists of sets of two courses, these course peers signify combinations of courses that shouldn't be taught at the same time or must be taught at the same time, respectively.

```

%INPUT FOR CLASS CONFLICTS
array[1..(numOfConflicts * 2)] of CLASSES: DISJUNCT_INPUT;
array[1..numOfConflicts, 1..2] of CLASSES: DISJUNCT = array2d(1..numOfConflicts, 1..2, DISJUNCT_INPUT);
%INPUT FOR PARALLEL LECTURES
array[1..(numOfParallelLectures * 2)] of CLASSES: PARALLEL_LEC_INPUT;
array[1..numOfParallelLectures, 1..2] of CLASSES: PARALLEL_LEC = array2d(1..numOfParallelLectures, 1..2, PARALLEL_LEC_INPUT);

```

Figure 6.12 – Course Dependencies Parameters

#### 6.2.4.5 Course details parameter

Array CLASS\_DETAILS contains information about each course, such as the class number, the instructor's number, the maximum number of students, and the type of the course as show below.

```
%INPUT FOR LINKING CLASSES TO INSTRUCTORS
%CLASS NUMBER, INSTRUCTOR NUMBER, NUMBER OF STUDENTS, TYPE: 1 FOR LECTURE, 2 FOR ONCE A WEEK LAB
%3 FOR TUTORIAL, 4 FOR TWICE A WEEK LAB, LAB NUMBER
array[1..(numOfClasses+1)*5] of int: CLASS_DETAILS_INPUT;
array[CLASSES, 1..5] of int: CLASS_DETAILS = array2d(CLASSES, 1..5, CLASS_DETAILS_INPUT);
```

Figure 6.13 – Course Details Parameter

## 6.2.5 Constraints

The constraints are separated into different groups in this section, depending on their functionality. For example, a set of 3 constraints could be needed to ensure that one of the requirements, that are shown in **Chapter 4.3.2**, is enforced.

### Constraints 1 – Unavailability

#### 1.1 - Instructors cannot teach when they are busy:

```
constraint forall(i in INSTRUCTORS, d in DAYS, s in SLOTS where d != WE ∧ BUSY[i,s,d] == 1)
    (I_SCHEDULE[i,d,s] == 1);
```

This constraint sets all the timeslots where the instructor is busy as having class 1, which is class E, representing an empty slot. This way, the instructor cannot be assigned any course during that timeslot.

#### 1.2 - Instructors cannot teach when they are busy on Wednesdays:

```
constraint forall(i in INSTRUCTORS where W_BUSY[i,1] == 1 ∨ W_BUSY[i,2] == 1)
    (I_SCHEDULE[i,WE,1] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,2] == 1 ∨ W_BUSY[i,3] == 1)
    (I_SCHEDULE[i,WE,2] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,4] == 1 ∨ W_BUSY[i,5] == 1)
    (I_SCHEDULE[i,WE,3] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,5] == 1 ∨ W_BUSY[i,6] == 1)
    (I_SCHEDULE[i,WE,4] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,7] == 1 ∨ W_BUSY[i,8] == 1)
    (I_SCHEDULE[i,WE,5] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,8] == 1 ∨ W_BUSY[i,9] == 1)
    (I_SCHEDULE[i,WE,6] == 1);
```

```

constraint forall(i in INSTRUCTORS where W_BUSY[i,10] == 1 ∨ W_BUSY[i,11] == 1)
    (I_SCHEDULE[i,WE,7] == 1);
constraint forall(i in INSTRUCTORS where W_BUSY[i,11] == 1 ∨ W_BUSY[i,12] == 1)
    (I_SCHEDULE[i,WE,8] == 1);

```

Since Wednesdays and the other days have a different number of timeslots, the unavailability of the instructors on Wednesdays are passed through a different array, W\_BUSY. Therefore, to determine the unavailability of the instructors in the schedule for courses that are not tutorials, the above constraints had to be used. For instance, the first one checks if the instructor is unavailable on either one of the first two 60-minute slots, if so, then they are also unavailable in the first slot of the 90-minute period, and so on.

### 1.3 - Instructor can't teach when they are busy on Wednesday for Wednesday Schedule:

```

constraint forall(i in INSTRUCTORS, s in W_SLOTS where W_BUSY[i,s] == 1)
    (I_W_SCHEDULE[i,s] == 1);

```

The tutorials, as previously shown in **6.2.3.2 Variables**, are scheduled in a different array, this being I\_W\_SCHEDULE.

This constraint sets the timeslots where the instructors are unavailable on Wednesdays as occupied by class E which represents an empty timeslot.

### Constraints 2 – Instructor and Course coherence

#### Each class can only be taught by its instructor:

```

constraint forall(c in CLASSES where c != 1)(
    forall(i in INSTRUCTORS, d in DAYS, s in SLOTS where i != CLASS_DETAILS[c,2])
        (I_SCHEDULE[i,d,s] != CLASSES[c]));

```

This constraint makes sure that each class that does not belong to an instructor is not present in their schedule. CLASS\_DETAILS[c,2] contains the instructor which teaches class c.

### Constraint 3 – Conflict courses

```

constraint forall(f in 1..numOfConflicts)(
    forall(i in INSTRUCTORS, d in DAYS, s in SLOTS)(
        forall(i2 in INSTRUCTORS, d2 in DAYS, s2 in SLOTS where d==d2 ∧ s==s2 ∧
            I_SCHEDULE[i,d,s] == DISJUNCT[f,1])(
                (I_SCHEDULE[i2,d2,s2] != DISJUNCT[f,2]) ));

```



Checks if a course is being taught in a timeslot, then no other instructor can have the second course in the DISJUNCT array in their schedule at the same timeslot. This constraint ensures that each pair of two courses defined in the DISJUNCT array are not taught on the same day and timeslot.

#### Constraint 4 - Parallel courses

```
constraint forall(f in 1..numOfParallelLectures)(
  forall(i in INSTRUCTORS, d in DAYS, s in SLOTS where d==MO ∨ d==TU)(
    forall (i2 in INSTRUCTORS, d2 in DAYS, s2 in SLOTS where (d2==MO ∨ d2==TU) ∧ (d!=d2
    ∨ s!=s2) ∧ (I_SCHEDULE[i,d,s] == PARALLEL_LEC[f,2]))
    (I_SCHEDULE[i2,d2,s2] != PARALLEL_LEC[f,1]));
```

Similarly, this constraint ensures that each of the two courses defined in the PARALLEL\_LEC array are taught at the same timeslot and day. This is done by ensuring that for all combinations of **different** timeslots, if the first course is taught, then the second course should not be taught. Therefore, constraining the courses into being taught at the same day and timeslot.

#### Constraints 5 – Twice a week Lectures

##### 5.1 - Each lecture can be taught a maximum of one time per day:

```
constraint forall (day in DAYS where day != 3)(
  forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 1)(
    sum (i in INSTRUCTORS, s in SLOTS where i == CLASS_DETAILS[c,2])
    (if I_SCHEDULE[i,day,s] == CLASSES[c] then 1 else 0 endif) <= 1));
```

##### 5.2 - Two Lectures for course of type twice a week lectures in total:

```
constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 1)(
  sum (i in INSTRUCTORS, d in DAYS, s in SLOTS)
  (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) == 2);
```

##### 5.3 - Lectures are taught at the same time slot on different days (MO-TH):

```
constraint forall(i in INSTRUCTORS, s in SLOTS)(
  forall(s2 in SLOTS where s != s2 ∧ I_SCHEDULE[i,MO,s] != 1 ∧ I_SCHEDULE[i,TH,s2] != 1 ∧
  CLASS_DETAILS[I_SCHEDULE[i,MO,s], 4] == 1 ∧ CLASS_DETAILS[I_SCHEDULE[i,TH,s2], 4]
  == 1)(
    I_SCHEDULE[i,MO,s] != I_SCHEDULE[i,TH,s2] );
```

#### 5.4 - Lectures are taught at the same time slot on different days (TU-FR):

```
constraint forall(i in INSTRUCTORS, s in SLOTS)(  
    forall(s2 in SLOTS where s != s2 ∧ I_SCHEDULE[i,TU,s] != 1 ∧ I_SCHEDULE[i,FR,s2] != 1  
    ∧ CLASS_DETAILS[I_SCHEDULE[i,TU,s], 4] == 1 ∧ CLASS_DETAILS[I_SCHEDULE[i,FR,s2], 4]  
    == 1)(  
        I_SCHEDULE[i,TU,s] != I_SCHEDULE[i,FR,s2]);
```

#### 5.5 - Lectures on MO-TH, TU-FR :

```
constraint forall(c in CLASSES where c != 1 ∧ (CLASS_DETAILS[c,4] == 1))(  
    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS where d == MO ∨ d == TH)  
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) == 2  
    ∨ sum (i in INSTRUCTORS, d in DAYS, s in SLOTS where d == TU ∨ d == FR)  
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) == 2);
```

Constraints 5.1, 5.2, 5.3, 5.4 and 5.5 ensure that courses of type twice a week lectures ( $CLASS\_DETAILS[C,4] == 1$ ) are taught on Monday and Thursday or Tuesday and Friday, once in each day and at the same timeslot. This is accomplished by limiting the lectures to being taught exactly two times per week in constraint 5.2, then forcing them into being scheduled at the exact same timeslot on the two different days, constant 5.3 & 5.4. Finally, constraint 5.5 ensures that one lecture is scheduled on Monday and the other one on Thursday, or Tuesday and Friday, respectively.

### Constraints 6 - Twice a week labs

#### 6.1 - Labs of type 4 (Twice a week) on MO-TH, TU-FR:

```
constraint forall(c in CLASSES where c != 1 ∧ (CLASS_DETAILS[c,4] == 4))(  
    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS where d == MO ∨ d == TH)  
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) ==  
    CLASS_DETAILS[CLASSES[c], 5] * 2  
    ∨ sum (i in INSTRUCTORS, d in DAYS, s in SLOTS where d == TU ∨ d == FR)  
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) ==  
    CLASS_DETAILS[CLASSES[c], 5] * 2);
```

#### 6.2 - Labs of type Twice a Week be taught at the same time slot on different days (MO-TH) or (TU-FR):

```
constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 4)(  
    sum (s in SLOTS)  
    (if I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c, 2]], TU, s] ==  
    I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c, 2]], FR, s] ∧
```

```

I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c, 2]],TU,s] == c then 1 else 0 endif) ==
CLASS_DETAILS[c, 5]
    √
    sum (s in SLOTS)
    (if I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c,2]],MO,s]==
I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c,2]],TH,s] ∧
I_SCHEDULE[INSTRUCTORS[CLASS_DETAILS[c, 2]],MO,s] == c then 1 else 0 endif) ==
CLASS_DETAILS[c, 5]);

```

### 6.3 - Labs of type (Twice a week) are taught in a week as many times as defined in CLASS\_DETAILS:

```

constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 4)(
    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS)
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) ==
(CLASS_DETAILS[CLASSES[c], 5] * 2));

```

Like constraints 5, the above constraints guarantee that the laboratories of type “twice a week” (CLASS\_DETAILS [c, 4] == 4) are taught in the schedule on Mondays and Thursdays or Tuesdays and Fridays, at the same timeslot. The only difference here is that the laboratories can be taught multiple times per day. This is defined in parameter CLASS\_DETAILS[CLASSES[c], 5].

### Constraint 7 - Once a week labs

#### Labs of type (Once a week) are taught in a week as many times as defined in CLASS\_DETAILS:

```

constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 2)(
    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS)
    (if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) ==
CLASS_DETAILS[CLASSES[c], 5]);

```

This constraint specifies that laboratories of type “Once a week” are taught as many times as the iterations are defined in CLASS\_DETAILS[CLASSES[c], 5]). It is important to note that in this case, these laboratories can be scheduled on any day of the week, without day constraints.

### Constraint 8 – Once a week Lectures

```

constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 5)(
    sum (i in INSTRUCTORS, d in DAYS, s in SLOTS)

```

```
(if I_SCHEDULE[i,d,s] == CLASSES[c] then 1 else 0 endif) == 1);
```

Lectures of type “once a week” (CLASS\_DETAILS[c, 4] == 5) are taught only once in the instructor’s schedule. This lecture can be placed anywhere in the schedule with no day or timeslot constraints, if all the other constraints are satisfied.

## Constraints 9 – Tutorials

### 9.1 - Tutorial conflicts:

```
constraint forall(f in 1..numOfConflicts)(
  forall(i in INSTRUCTORS, s in W_SLOTS)(
    forall(i2 in INSTRUCTORS, s2 in W_SLOTS where s==s2 ∧ I_W_SCHEDULE[i,s] ==
DISJUNCT[f,1])(
      (I_W_SCHEDULE[i2,s2] != DISJUNCT[f,2]) ));
```

### 9.2 - Tutorials taught in parallel:

```
constraint forall(f in 1..numOfParallelLectures)(
  forall(i in INSTRUCTORS, s in W_SLOTS)(
    forall (i2 in INSTRUCTORS, s2 in W_SLOTS where s!=s2 ∧ (I_W_SCHEDULE[i,s] ==
PARALLEL_LEC[f,2]))
      (I_W_SCHEDULE[i2,s2] != PARALLEL_LEC[f,1]) ));
```

### 9.3 - Each tutorial is only taught once in the I\_W\_TIMETABLE:

```
constraint forall (c in CLASSES where c != 1 ∧ CLASS_DETAILS[c, 4] == 3)(
  sum(i in INSTRUCTORS, s in W_SLOTS)
    (if I_W_SCHEDULE[i,s] == c then 1 else 0 endif) == 1 );
```

### 9.4 - Each tutorial can only be taught by its instructor:

```
constraint forall(c in CLASSES where c != 1 ∧ CLASS_DETAILS[c,4] == 3)(
  forall(i in INSTRUCTORS, s in W_SLOTS where i != CLASS_DETAILS[c,2])
    (I_W_SCHEDULE[i,s] != CLASSES[c]));
```

Like the other array, the array for the tutorials is constrained into following the same rules.

## Constraints 10 – Coherence between the two arrays.

### 10.1 - I\_SCHEDULE does not have tutorials:

```
constraint forall(i in INSTRUCTORS, d in DAYS, s in SLOTS)(
```

$(CLASS\_DETAILS[I\_SCHEDULE[i,d,s], 4] \neq 3));$

## 10.2 - Wednesday schedule can only have tutorials:

*constraint forall*(*i in INSTRUCTORS, s in W\_SLOTS*)(

$(CLASS\_DETAILS[I\_W\_SCHEDULE[i,s], 4] == 3 \vee I\_W\_SCHEDULE[i,s] == 1));$

## 10.3 - Wednesdays in I\_SCHEDULE only have labs (No lectures or tutorials):

*constraint forall*(*i in INSTRUCTORS, s in SLOTS*)(

$(CLASS\_DETAILS[I\_SCHEDULE[i,WE,s], 4] == 2 \vee I\_SCHEDULE[i,WE,s] == 1));$

## 10.4 - Instructors cannot have labs and tutorial at the same time in the two different schedules:

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,1] == 1 \vee (I\_W\_SCHEDULE[i,1] == 1 \wedge I\_W\_SCHEDULE[i,2] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,2] == 1 \vee (I\_W\_SCHEDULE[i,2] == 1 \wedge I\_W\_SCHEDULE[i,3] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,3] == 1 \vee (I\_W\_SCHEDULE[i,4] == 1 \wedge I\_W\_SCHEDULE[i,5] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,4] == 1 \vee (I\_W\_SCHEDULE[i,5] == 1 \wedge I\_W\_SCHEDULE[i,6] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,5] == 1 \vee (I\_W\_SCHEDULE[i,7] == 1 \wedge I\_W\_SCHEDULE[i,8] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,6] == 1 \vee (I\_W\_SCHEDULE[i,8] == 1 \wedge I\_W\_SCHEDULE[i,9] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,7] == 1 \vee (I\_W\_SCHEDULE[i,10] == 1 \wedge I\_W\_SCHEDULE[i,11] == 1));$

*constraint forall*(*i in INSTRUCTORS*)(

$(I\_SCHEDULE[i,WE,8] == 1 \vee (I\_W\_SCHEDULE[i,11] == 1 \wedge I\_W\_SCHEDULE[i,12] == 1));$

The above constraints ensure that there is consistency between the two variable arrays. For example, if an instructor has a tutorial scheduled on the first timeslot on Wednesday, then they shouldn't have a laboratory scheduled at the same time in the I\_SCHEDULES array.

## Constraints 11 – Iterations of laboratories

### 11.1 - Labs of type (Twice a week) are taught one after the other with no breaks in between on MO-TH:

$$\text{constraint forall}(c \text{ in } \text{CLASSES}, s1 \text{ in } \text{SLOTS}, s2 \text{ in } \text{SLOTS} \text{ where } c \neq 1 \wedge s1 \neq s2 \wedge \\
\text{CLASS\_DETAILS}[c, 4] == 4 \wedge \text{CLASS\_DETAILS}[c, 5] > 1 \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], MO, s1] == c \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], MO, s2] = c)( \\
( \text{abs}(s1-s2) < \text{CLASS\_DETAILS}[c, 5]) );$$

### 11.2 - Labs of type (Twice a Week) are taught one after the other with no breaks in between on TU-FR:

$$\text{constraint forall}(c \text{ in } \text{CLASSES}, s1 \text{ in } \text{SLOTS}, s2 \text{ in } \text{SLOTS} \text{ where } c \neq 1 \wedge s1 \neq s2 \wedge \\
\text{CLASS\_DETAILS}[c, 4] == 4 \wedge \text{CLASS\_DETAILS}[c, 5] > 1 \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], TU, s1] == c \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], TU, s2] = c)( \\
( \text{abs}(s1-s2) < \text{CLASS\_DETAILS}[c, 5]) );$$

### 11.3 - Labs of type (Once a week) are taught one after the other with no breaks in between:

$$\text{constraint forall}(c \text{ in } \text{CLASSES}, d \text{ in } \text{DAYS}, s1 \text{ in } \text{SLOTS}, s2 \text{ in } \text{SLOTS} \text{ where } c \neq 1 \wedge s1 \neq s2 \wedge \\
\text{CLASS\_DETAILS}[c, 4] == 2 \wedge \text{CLASS\_DETAILS}[c, 5] > 1 \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], d, s1] == c \wedge \\
I\_SCHEDULE[INSTRUCTORS[\text{CLASS\_DETAILS}[c, 2]], d, s2] = c)( \\
( \text{abs}(s1-s2) < \text{CLASS\_DETAILS}[c, 5]) );$$

### 11.4 - Labs of type (Once a week) are taught on the same day:

$$\text{constraint forall}(c \text{ in } \text{CLASSES} \text{ where } c \neq 1 \wedge (\text{CLASS\_DETAILS}[c, 4] == 2))( \\
\text{sum } (i \text{ in } \text{INSTRUCTORS}, s \text{ in } \text{SLOTS}) \\
(\text{if } I\_SCHEDULE[i, MO, s] == \text{CLASSES}[c] \text{ then } 1 \text{ else } 0 \text{ endif}) == \\
\text{CLASS\_DETAILS}[\text{CLASSES}[c], 5] \\
\vee \text{sum } (i \text{ in } \text{INSTRUCTORS}, s \text{ in } \text{SLOTS}) \\
(\text{if } I\_SCHEDULE[i, TU, s] == \text{CLASSES}[c] \text{ then } 1 \text{ else } 0 \text{ endif}) == \\
\text{CLASS\_DETAILS}[\text{CLASSES}[c], 5] \\
\vee \text{sum } (i \text{ in } \text{INSTRUCTORS}, s \text{ in } \text{SLOTS}) \\
(\text{if } I\_SCHEDULE[i, WE, s] == \text{CLASSES}[c] \text{ then } 1 \text{ else } 0 \text{ endif}) == \\
\text{CLASS\_DETAILS}[\text{CLASSES}[c], 5] \\
\vee \text{sum } (i \text{ in } \text{INSTRUCTORS}, s \text{ in } \text{SLOTS}) \\
(\text{if } I\_SCHEDULE[i, TH, s] == \text{CLASSES}[c] \text{ then } 1 \text{ else } 0 \text{ endif}) == \\
\text{CLASS\_DETAILS}[\text{CLASSES}[c], 5] \\
\vee \text{sum } (i \text{ in } \text{INSTRUCTORS}, s \text{ in } \text{SLOTS}) \\
(\text{if } I\_SCHEDULE[i, FR, s] == \text{CLASSES}[c] \text{ then } 1 \text{ else } 0 \text{ endif}) == \\
\text{CLASS\_DETAILS}[\text{CLASSES}[c], 5]);$$

These constraints force the iterations/sections of the laboratories to be taught consecutively one after the other on the same day.

## Constraints 12 – Room capacities

### 12.1 - Maximum capacity for how many lectures can be taught at the same time:

```
constraint forall(d in DAYS, s in SLOTS)(
    sum(i in INSTRUCTORS where CLASS_DETAILS[I_SCHEDULE[i,d,s],4] == 1)
    (if I_SCHEDULE[i,d,s] != 1 then 1 else 0 endif) <= num_of_lecture_rooms );
```

### 12.2 - Maximum capacity for how many labs can be taught at the same time:

```
constraint forall(d in DAYS, s in SLOTS)(
    sum(i in INSTRUCTORS where CLASS_DETAILS[I_SCHEDULE[i,d,s],4] == 2 ∨
    CLASS_DETAILS[I_SCHEDULE[i,d,s],4] == 4)
    (if I_SCHEDULE[i,d,s] != 1 then 1 else 0 endif) <= num_of_lab_rooms );
```

### 12.3 - Maximum capacity for how many tutorials can be taught at the same time:

```
constraint forall(s in W_SLOTS)(
    sum(i in INSTRUCTORS where CLASS_DETAILS[I_W_SCHEDULE[i,s],4] == 3)
    (if I_W_SCHEDULE[i,s] != 1 then 1 else 0 endif) <= num_of_lecture_rooms );
```

The above constraints limit the number of lectures, tutorials and laboratories that can be scheduled in each day and timeslot, between all the instructors. The numbers of rooms in the department are passed into the model through parameters num\_of\_lecture\_rooms and num\_of\_lab\_rooms.

## 6.2.6 Versions

The final system consists of three Minizinc models, these are referred to as versions in this dissertation. Each one of these versions has slight differences from the other two, mainly regarding how strict the requirements for the laboratory sections are.

The Department usually schedules the sections of a laboratory, which are created when the number of students registered in a laboratory is higher than the capacity of the laboratory rooms, to be taught consecutively, one after the other.

Version 1 is the simplest one, it created a valid schedule that satisfies all the requirements given to it. Meaning, in the case of laboratories that must be taught in many sections, this version will freely schedule them on different days and timeslots without consecutivity constraint.

Version 2, on the other hand, schedules the sections to be taught on the same day but without any constraint on the time of instructing. In other words, one laboratory could be taught in the morning and the other one in the evening, as long as these slots satisfy the rest of the constraints.

Lastly, version 3 is the strictest. It schedules all the sections of a laboratory to be taught, on the same day and consecutively. This means that each laboratory section will be taught immediately after the previous one, without any breaks in between.

### **6.2.7 Solver**

The Minizinc tool comes with several off-the-shelf solvers. These solvers have different characteristics and are optimized for different uses. The default solver in Minizinc is the Gecode solver. It is a general-purpose solver used for solving simple constraint problems. In addition, Minizinc also supports the OR-Tools solver, which has the advantage of being able to execute multi-threadedly to improve performance. [20]

The chosen solver for the model is the Chuffed solver. From the early stages of the model's development, it was obvious that this solver was going to be the solver of choice. This conclusion came from small tests that were conducted on early versions of the model. The results were very clear, while Chuffed solver could produce an answer in an acceptable timeframe, the other off-the shelf solvers could not produce an answer, even if left for hours. Similar results were found by Rahman et al. [15] as discussed in **Chapter 2.2.2**.

The Chuffed solver, supported by Minizinc, stands out as one of the top-performing solvers. Its proficiency in conflict clause learning, watched literal propagation, and activity-based search heuristics makes it significantly more efficient compared to other solvers. [20]



## 6.3 Back-end implementation

Minizinc, as it is a modeling language, can only read data in the form of parameters and find satisfactory values to a set of variables. Meaning, it is not able to read any parameters from a database or create easy to read graphical outputs. For these reasons, separate **Python scripts** were created to handle the input and output of data from the model.

### 6.3.1 Input script

The MiniZinc solvers read the input parameters of a model from a file in the ".dzn" format. This file contains all the parameters as defined in **Chapter 6.2.3**. An example of such a file can be seen in **Appendix 2.1**.

However, the user-defined input variables from the interface are stored in the database in the form of records. For this reason, a script which is able to transform the data from the database into the ".dzn" file format was required.

It is essential to point out that the model can only recognize inputs in the form of integer numbers, therefore the Python scripts must be able to associate the courses and instructor names/code to integer values. This association is very critical for the output scripts, which would associate the output integer values back to the course/instructor codes.

The script first establishes a connection to the database, then creates two files that associate the course codes and instructor usernames to integer values and saves them in ".txt" files. Afterwards, it builds the ".dzn" file by reading the database and creating the input for each needed array or parameter required by the model.

### 6.3.2 Output scripts

Once the solver has completed its task, it outputs the populated 3D schedule arrays in a 2D format, one 2D schedule for each instructor. This output operation can be seen in **Appendix 2.2**. The output results are then saved into a text file called "result.txt", this file is used by the Python scripts, along with the assassination files created by the input script to create graphical easy to read PDF schedules.

The first script splits the output results into separate files for each instructor, which then the second script uses to generate PDF schedules for each instructor. In addition, the second script uses the files “courses.txt” and “instructors.txt” to associate the given output integers to the courses and instructors.

**Instructor: sazeidis0**

Time	Monday	Tuesday	Wednesday	Thursday	Friday
9:00					
9:30					
10:00			EPL420_T		
10:30					
11:00	EPL131	EPL420		EPL131	EPL420
11:30					
12:00					
12:30					
13:00					
13:30					
14:00					
14:30					
15:00					
15:30					
16:00					
16:30					
17:00	EPL420_L		EPL131_T		
17:30					
18:00					
18:30					
19:00					
19:30					
20:00					
20:30					

Figure 6.14 – Instructor’s PDF Schedule

The generated PDF files visualise the schedule for each instructor. A different color is used for each lecture to emphasize the time of start and end. The course code for Lectures and Laboratories is shown in the middle of the teaching period, while for the tutorials, on the top portion.

## Chapter 7

### 7 Testing and Evaluation

---

#### 7.1 Minizinc Model Testing

#### 7.2 Website Evaluation

#### 7.3 Limitations

---

#### 7.1 Minizinc Model Testing

For the purposes of testing the developed algorithm with realistic data sets, the number of instructors and courses in the Fall semester of 2023 was used as a guide. Meaning, a ratio of instructors to courses was calculated and used in the testing process. The schedule consisted of 110 different courses, these being lectures, laboratories, and tutorials, and 35 instructors. Therefore, the ratio is  $110/35 = 3.14$  courses to instructors. Subsequently, the model was tested on data sets ranging from 30 courses and 10 instructors up to 150 courses and 47 instructors. These tests were conducted with the static 5 courses conflicts constraints.

It is important to note that these tests were conducted on a personal computer and not on a large-scale supercomputer. The computer is powered by an Intel Core (TM) i7-9750H CPU running on 2.60GHz with 16GB of RAM. Therefore, executing the solver on a high-performance computer most probably would result in better running speeds.

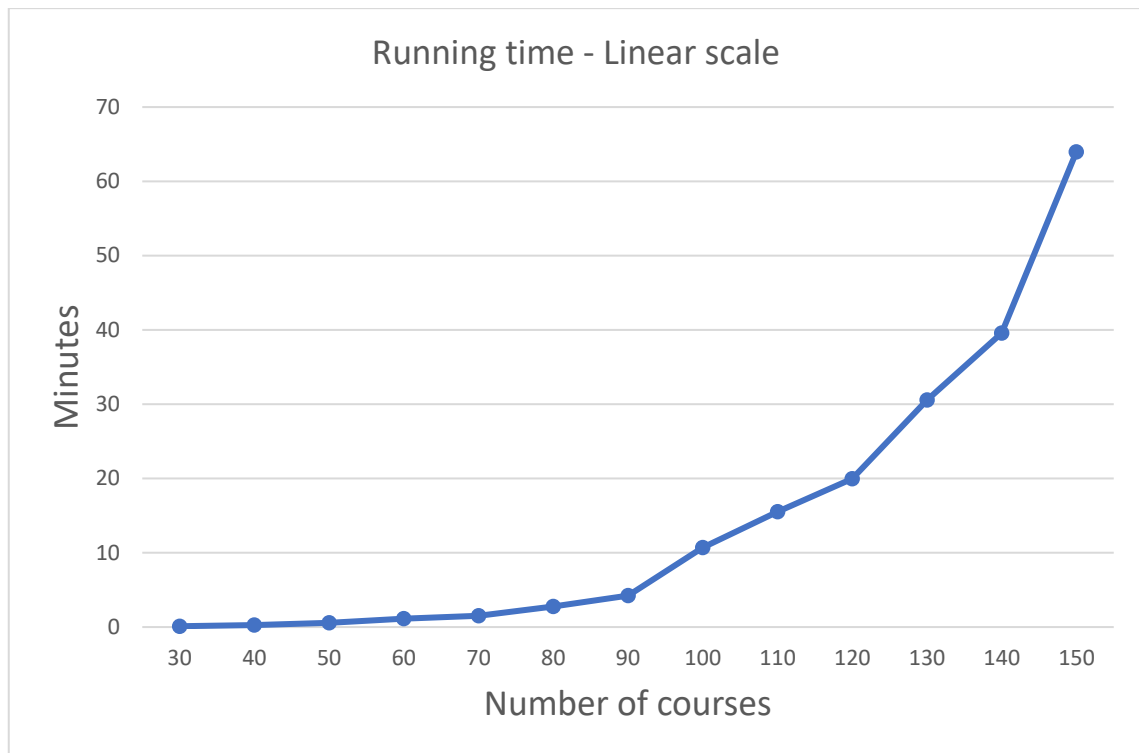


Figure 7.1 – Running Time Test Linear Scale

The above graph shows the running time of the **Chuffed** solver when applied to the most constraining version of the model (Version 3), since the goal of the testing is to verify that the model can solve the hardest problem in an acceptable timeframe. The time complexity of the model seems to be exponential. A satisfactory schedule can be found for problems with up to 90 courses in less than 5 minutes. However, only a 60% increase in size to 150 courses results in an astronomical 1517% increase in the observed running time. This exponential trend can be seen in the steep upward curve of the graph, showing a significant increase in running time as the size of the problem gets larger.

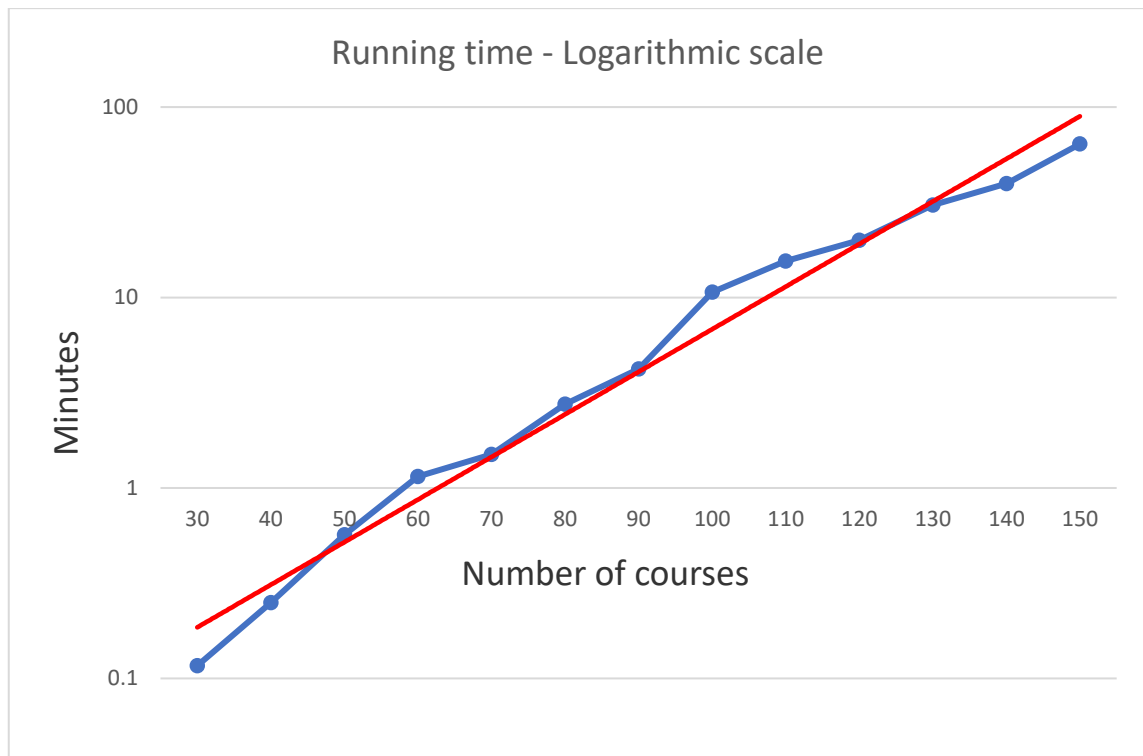
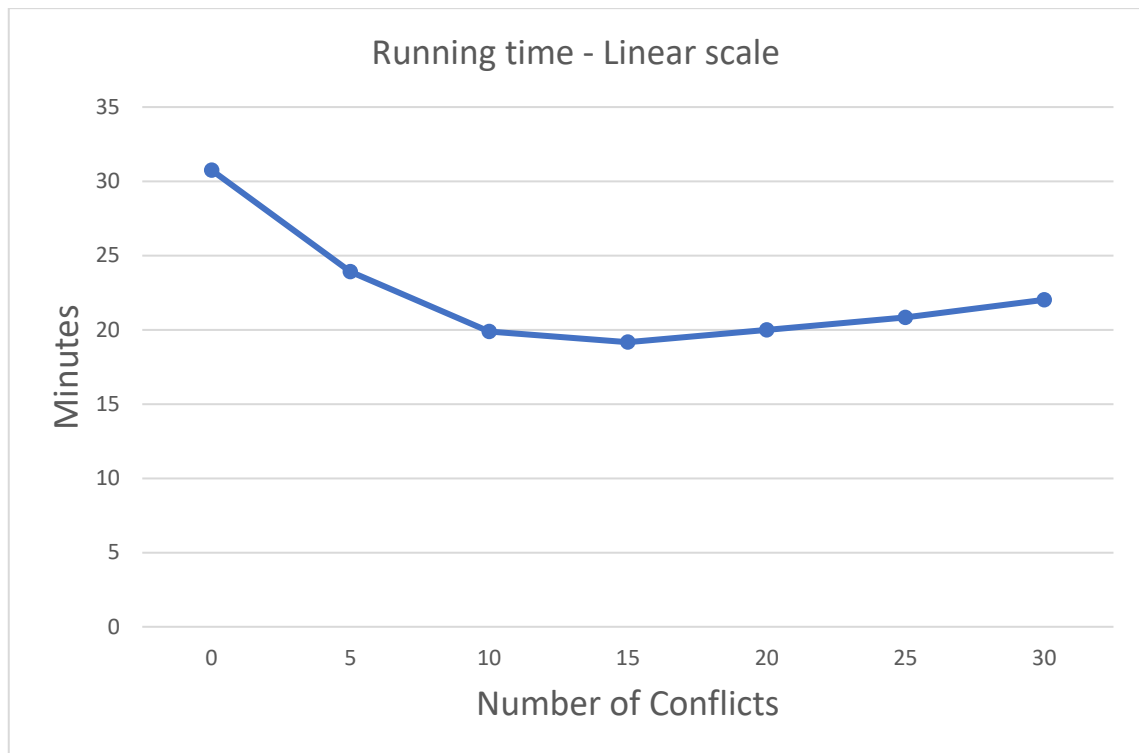


Figure 7.2 – Running Time Test Algorithmic Scale

This exponential trend can be seen more clearly in the above graph where the Y-axis is set to a logarithmic scale. The upward trendline is marked in red.

This information can be useful in determining the practicality and efficiency of using this approach. For the department of computer science, the model can find a satisfactory schedule in **about 17 minutes**, which is a very reasonable timeframe. Subsequently, even if the department scales up its operations, making the problem larger, this approach can still be useful as a schedule can be produced for 150 courses in about an hour.

Subsequently, the model was testing on different amounts of course conflicts with the same number of courses and instructors. Specifically, with 110 courses and 35 instructors. The tests were conducted with schedules with 0 course conflicts, up to 30 course conflicts. These conflicts were set to a maximum chain of 5 courses. In other words, only a maximum of 5 courses could be chained in the same conflicts chain. Conflicts chains exist when course A and course B are conflicting courses, and course B and C are conflicting courses, therefore course A and C also become conflicting courses.



The above graph depicts the results of these tests. The number of conflicts does not seem to affect the required running time of the solver by a substantial amount. In addition, it seems that a small number of conflicts improve the speed of the solver.

## 7.2 Website Evaluation

The goal of this part of the evaluation phase is to determine the usability of the we-interface along with the user experience. The system was given to multiple users to be used in a normal setting, without any guidance. Subsequently, a questionnaire, consisting of 11 questions about ease of use and general usability was given to the participants for collecting and documenting the feedback. The questionnaire can be seen in **Appendix 3**. The results of the questionnaire are presented below.

Do you feel that the website was easy to navigate?  
8 responses

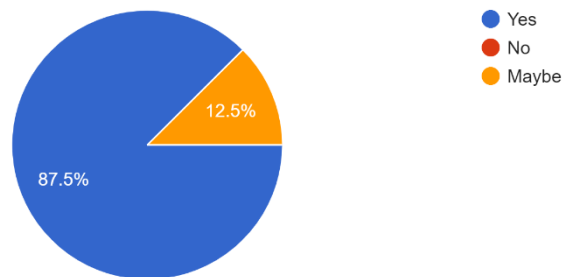


Figure 7.3 – Questionnaire Question 1

Is the content on the website clear and understandable?  
8 responses

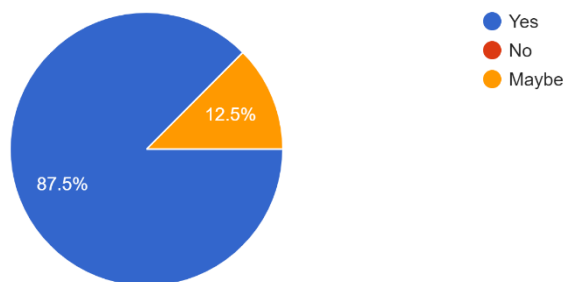


Figure 7.4 – Questionnaire Question 2

Do you find the fonts and colors used in the website to be easily recognizable and readable?  
8 responses

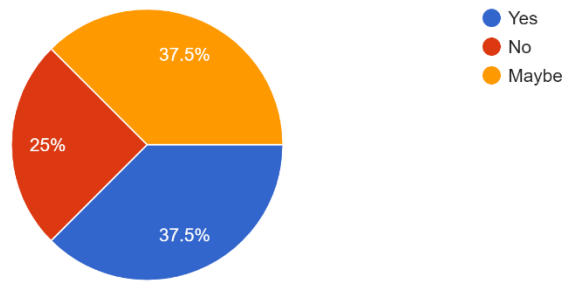


Figure 7.5 – Questionnaire Question 3

How satisfied are you by the graphical design on the website?  
8 responses

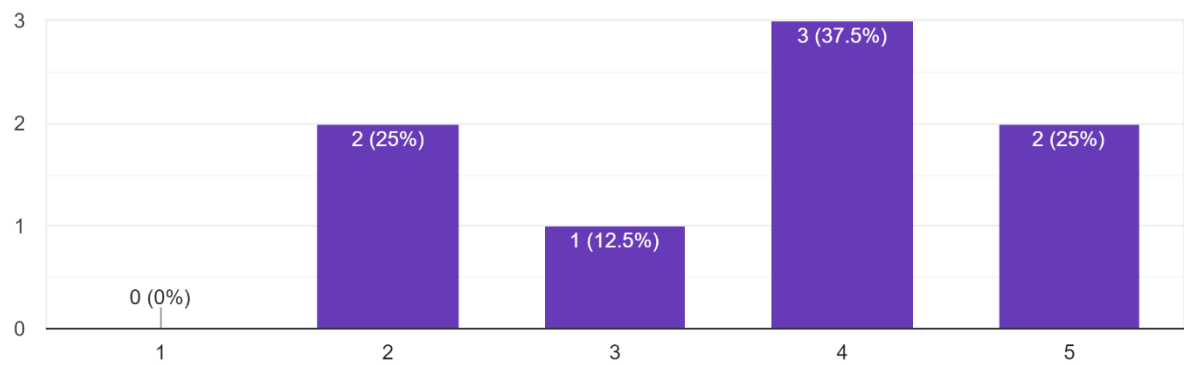


Figure 7.6 – Questionnaire Question 4



How did the website compare to other similar websites you've used in the past?  
8 responses

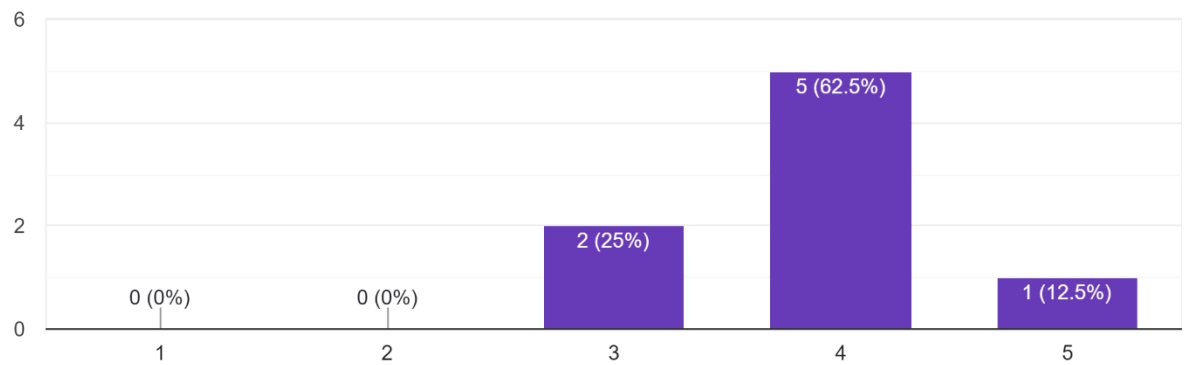


Figure 7.7 – Questionnaire Question 5

How easy did you find it to understand the purpose of each page on the website?  
8 responses

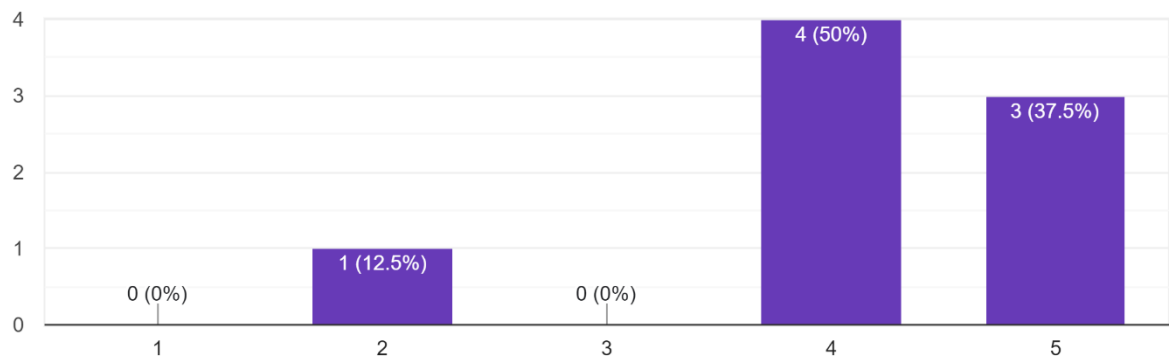


Figure 7.8 – Questionnaire Question 6

How satisfied are you by the way the schedules are presented on the website?

8 responses

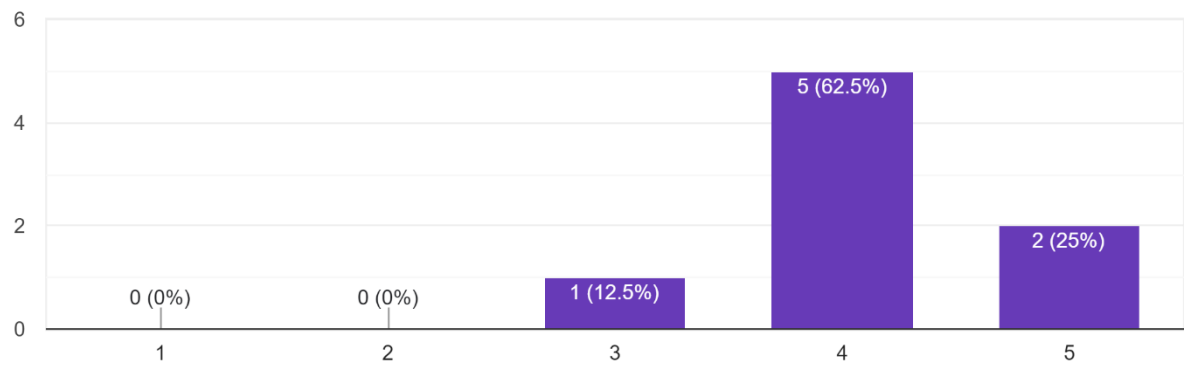


Figure 7.9 – Questionnaire Question 7

Did you encounter any issues while filling out any forms?

8 responses

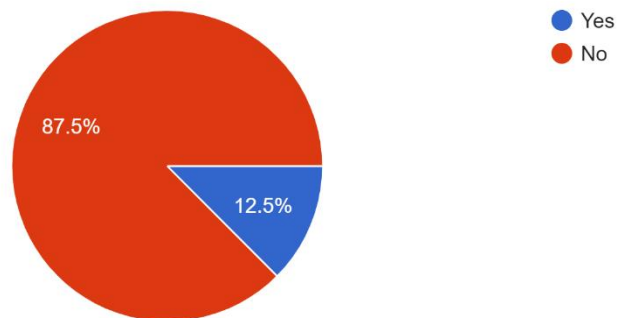


Figure 7.10 – Questionnaire Question 8

Please rate your overall experience while using the website  
8 responses

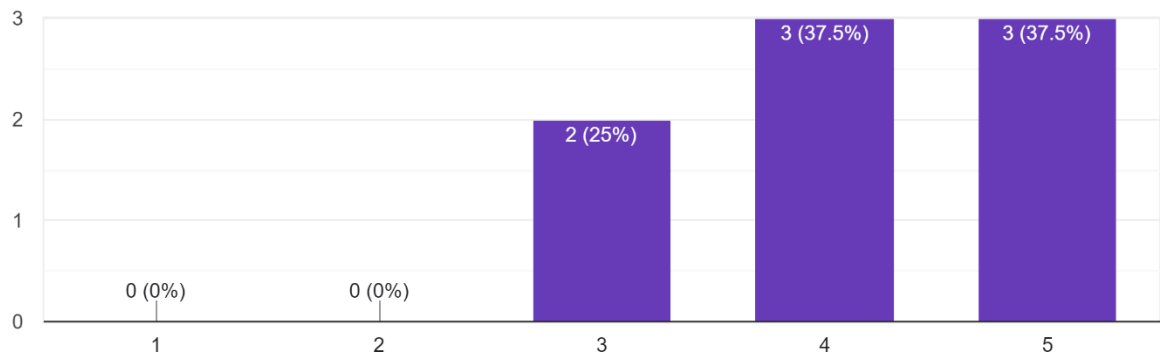


Figure 7.11 – Questionnaire Question 9

Firstly, the questionnaire revealed that most of the participants found the website to be easily navigable, with only 12.5% of the respondents having difficulties in navigating between the different pages. Similarly, most participants found the content on the web interface to be clear and easy to understand, along with the purpose of each page. On the other hand, the fonts and colors used on the website seemed to not satisfy the participants' expectations, as only 37.5% of participants found the colors and fonts to be easily recognizable and readable. In addition, most users found the graphical design to be satisfactory, with only 25% of participants answering with "Not satisfied" and none with "Not satisfied at all". These two areas, the colors and fonts, and the graphical design can be the focus of future improvements.

Most participants found the interface to be quite similar systems they have used in the past but not exactly the same. This is great because it suggests that the interface is familiar enough for users to navigate with ease, yet unique enough to stand out from the crowd. In addition, the participants seemed to be satisfied with the way the schedules are displayed.

Finally, when the participants were asked to suggest any changes to the interface, one participant noted that the Error message could be improved to be more specific in some cases.

### **7.3 Limitations**

The implemented system succeeds in managing and producing valid schedules for the instructors of the department in a reasonable timeframe. However, the system has some limitations.

For one, the system can produce a schedule only for the instructors of the department, classroom assignment was not realized in this implementation. As previously discussed, merging both problems into the same model was not successful, this is because of the produced time-complexity, which is much higher than the time-complexity of the problems individually. In other words, the merged model required astronomical timeframes to solve the problems, which is not realistic. Meanwhile, the instructor's schedule problem was prioritized as it posed a more challenging problem for the department's secretary than the room assignment problem. Although, the classroom assignment problem can be implemented by extending the system into including a second Minizinc model, which takes the generated schedule by the first model as an input and generates a similar schedule for the classrooms.

In addition, due to time constraints, the developed system does not schedule sections of a laboratory to different instructors, meaning that all sections of a laboratory must be taught by the same instructor. This feature can be implemented in future work.

Finally, the interface could benefit from additional functionalities, such as editing the information of instructors or resetting a forgotten password. These functionalities can be implemented in future versions of the system.

## **Chapter 8**

### **8 Conclusion**

---

8.1 Conclusion

8.2 Challenges

8.3 Future work

---

#### **8.1 Conclusion**

In this dissertation, the course scheduling problem was explored and analyzed. From its nature, the use of Artificial Intelligence techniques made perfect sense as the most viable solution to such an NP hard problem. Subsequently, existing approaches were presented and discussed, along with the Constraint Satisfaction Programming technique. Forthwith, the Software Development Life Cycle was introduced and followed during the development of the system. Starting with the Requirements analysis, where the requirements for the front-end interface and the schedule were defined and documented. Subsequently, the design of the database and the interface were presented and discussed. In addition, some key components from the implementation stage were showcased. Finally, the system was tested and evaluated. In this stage, we concluded that the developed system could go through further enhancements, mainly regarding the web-interface and the realization of a room course scheduler. In closing, the conducted testing revealed that the automated tool is capable of handling the required number of courses and instructors to produce a valid schedule for the Department in a reasonable timeframe.

#### **8.2 Challenges**

During the designing and developing of the tool, many challenges were encountered. Firstly, as previously discussed, the capabilities and weaknesses of the Minizinc tool were not clear from the early stages. This deemed the project as being somewhat experimental

because there weren't many existing similar implementations of such an approach that uses the Minizinc tool.

The Minizinc tool, although user-friendly, has the disadvantage of having a lack of widespread usage. Due to this, there is a limited set of documentation and materials available about it. This was one of the main challenges that we faced, as acquiring the necessary knowledge and skills from the very few available sources was challenging.

Additionally, the implementation of a Minizinc model that would create both the instructors' schedule as well as the rooms' schedule posed a significant challenge because of the time-complexity required, as shown from the testing. For this reason, only the model for the instructors' schedule was realized.

Lastly, the deployment of the back-end component of the implementation faced significant obstacles and was unsuccessful due to a cyberattack targeting the university's infrastructure, which posed numerous technological challenges.

### **8.3 Future work**

Due to time constraints, some of the features that were proposed in the Requirements Analysis stage were not realized. Firstly, as explained above, the implementation of a Minizinc model that would create schedules for the rooms and laboratories of the Department can be realized in future work.

In addition, the web interface of the tool could be formerly improved to better suit the needs of the users and enhance its usability. This can include the implementation of some of the features that were documented in the Requirements Analysis stage, such as the possibility for an instructor to update their unavailability or change the details of a course, since these requirements were not implemented in the developed system.

Furthermore, given that only the Interface of the system has been successfully deployed on the servers of the department, the back-end algorithm could also be deployed and tested on the servers.

## Bibliography

- [1] PagerDuty. (n.d.). What Is the Software Development Life Cycle?  
<https://www.pagerduty.com/resources/learn/software-development-life-cycle/>
- [2] Jevtic, G. (2019, May 15). What is SDLC? How the Software Development Life Cycle Works. PhoenixNAP Global IT Services.  
<https://phoenixnap.com/blog/software-development-life-cycle>
- [3] What Is SDLC (Software Development Life Cycle) Phases & Process. (n.d.).  
<https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc>
- [4] Ruparelia, N. B. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, 35(3), 8.
- [5] What is PHP (Hypertext Preprocessor)? - Definition from WhatIs.com. (n.d.). WhatIs.com. <https://www.techtarget.com/whatis/definition/PHP-Hypertext-Preprocessor>
- [6] Simmons, L. (2022, January 7). The Difference Between Front-End vs. Back-End. Code a New Career | ComputerScience.org.  
<https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/>
- [7] Python Software Foundation. (2019). What is Python? Executive Summary. Python.org. <https://www.python.org/doc/essays/blurb/>
- [8] Oracle. (2021). What is MySQL? Oracle.com.  
<https://www.oracle.com/mysql/what-is-mysql/>
- [9] What is HTML, and why should I learn it? (2020, February 17). Code Institute Global. <https://codeinstitute.net/global/blog/what-is-html-and-why-should-i-learn-it/>

- [10] O’Grady, B. (2020, February 25). What is CSS? The Ultimate Intro. Code Institute Global. <https://codeinstitute.net/global/blog/what-is-css-and-why-should-i-learn-it/>
  
- [11] Mozilla. (2019, July 2). What is JavaScript? MDN Web Docs. [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/What\\_is\\_JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript)
  
- [12] Wasfy, A., & Aloul, F. (n.d.). Solving the University Class Scheduling Problem Using Advanced ILP Techniques. [http://www.aloul.net/Papers/faloul\\_sch\\_gcc07.pdf](http://www.aloul.net/Papers/faloul_sch_gcc07.pdf)
  
- [13] Chen, R.-M., & Shih, H.-F. (2013). Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search. *Algorithms*, 6(2), 227–244. <https://doi.org/10.3390/a6020227>
  
- [14] Aycan, E., & Ayav, T. (2009, March 1). Solving the Course Scheduling Problem Using Simulated Annealing. *IEEE Xplore*. <https://doi.org/10.1109/IADCC.2009.4809055>
  
- [15] Rahman, M., Binte Noor, S., & Siddiqui, H. (n.d.). Automated Large-scale Class Scheduling in MiniZinc. <https://arxiv.org/pdf/2011.07507.pdf>
  
- [16] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., & Fischer, J. (2014). The MiniZinc Challenge 2008–2013. *AI Magazine*, 35(2), 55. <https://doi.org/10.1609/aimag.v35i2.2539>
  
- [17] Barták, R. (n.d.). Constraint Programming - What is behind? <https://kti.mff.cuni.cz/~bartak/NASSLLI2003/files/paper1.pdf>
  
- [18] Cooper, T., & Kingston, J. (n.d.). The Complexity of Timetable Construction Problems. [http://jeffreykingston.id.au/tt\\_papers/cooper\\_kingston\\_1996\\_complexity.pdf](http://jeffreykingston.id.au/tt_papers/cooper_kingston_1996_complexity.pdf)



- [19] Snapcraft - Snaps are universal Linux packages. (n.d.). Snapcraft  
<https://snapcraft.io/>
  
- [20] The MiniZinc Handbook. (n.d.). Solving Technologies and Solver Backends.  
<https://www.minizinc.org/doc-2.4.3/en/solvers.html?highlight=cbc%20solver>
  
- [21] Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., & Kendall, G. (2021). A  
Survey of University Course Timetabling Problem: Perspectives, Trends and  
Opportunities. *IEEE Access*, 9, 106515–106529.  
<https://doi.org/10.1109/access.2021.3100613>

## Appendix 1

This appendix includes some key parts of the implementation.

1:

```
// LOGIN USER
if (isset($_POST['login_user'])) {
    $username = mysqli_real_escape_string($db, $_POST['username']);
    $passwordReal = mysqli_real_escape_string($db, $_POST['password']);
    $password = hash("sha512", $passwordReal);
```

PHP function that is triggered by the user's login, it reads the user's credentials and hashes the password to compare it later to the saved password in the database.

2:

```
<script>
function validatePassword() {
    var password1 = document.getElementById("password1").value;
    var password2 = document.getElementById("password2").value;
    if (password1 != password2){
        alert("Enter the same passwords please!");
        return false;
    }
    var regex = /^(?=.*[A-Z])(?=.*[!@#$%^&*])(?={8,})/;
    if (!regex.test(password)) {
        alert("Password must be at least 8 characters long, contain a capital letter and a symbol!");
        return false;
    }
    return true;
}
</script>
```

JavaScript function for testing the validity of the passwords to ensure compliance with the password security requirements.

3:

```
process.php / ...
<?php
session_start();
// initializing variables
$name = "";
$surname = "";
$email = "";
$password = "";
$errors = array();

// connect to the database
$db = mysqli_connect('localhost', 'timefadetool', 'QJJHpr8qFrR1aCbK', 'timefadetool');

// REGISTER INSTRUCTOR
if (isset($_POST['reg_user'])) {
    $inst_username = mysqli_real_escape_string($db, $_POST['inst_username']);
    $user_check_query = "SELECT * FROM instructors WHERE instructor_username='$inst_username' LIMIT 1";
    $result = mysqli_query($db, $user_check_query);
    $user = mysqli_fetch_assoc($result);
    if ($user) {
        if ($user['instructor_username'] === $inst_username) {
            function_alert("Instructor already exists!");
        }
    } else {
        $inst_email = mysqli_real_escape_string($db, $_POST['email']);
        $user_check_query = "SELECT * FROM instructors WHERE instructor_username='$inst_email' LIMIT 1";
        $result = mysqli_query($db, $user_check_query);
        $user = mysqli_fetch_assoc($result);
        if ($user) {
            if ($user['instructor_username'] === $inst_username) {
                function_alert("Email already used!");
            }
        } else {
```

PHP function for registering a user.

4:

The code for the tool can be found in this GitHub link:

<https://github.com/sohanassa/Thesis.git>

## Appendix 2

This appendix presents a sample of the “.dzn” file used for the Minizinc model and the output functions in the model in part 2.

**1:**

```
numOfInstructors = 35;

INSTRUCTORS = {T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T32, T33, T34, T35};

num_of_lecture_rooms = 20;
num_of_lab_rooms = 20;

CLASSES = {E, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30, C31, C32, C33, C34, C35, C36, C37, C38, C39, C40, C41, C42, C43, C44, C45, C46, C47, C48, C49, C50, C51, C52, C53, C54, C55, C56, C57, C58, C59, C60, C61, C62, C63, C64, C65, C66, C67, C68, C69, C70, C71, C72, C73, C74, C75, C76, C77, C78, C79, C80, C81, C82, C83, C84, C85, C86, C87, C88, C89, C90, C91, C92, C93, C94, C95, C96, C97, C98, C99, C100, C101, C102, C103, C104, C105, C106, C107, C108, C109, C110};

numOfConflicts = 6;

DISJUNCT_INPUT = [C29, C32, C32, C35, C39, C41, C39, C44, C53, C56, C53, C59];

numOfParallelLectures = 0;

PARALLEL_LEC_INPUT = [];

W_BUSY_INPUT = [
0,
0,
0,|
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
```

```
#60  
#61 0,  
#62 0,  
#63 0,  
#64 0,  
#65 0,  
#66 0,  
#67 0,  
#68 0,  
#69 0,  
#70 0,  
#71 0,  
#72 0];  
#73  
#74  
#75 BUSY_INPUT = [  
#76 0, 0, 0, 0, 0, 0,  
#77 0, 0, 0, 0, 0, 0,  
#78 0, 0, 0, 0, 0, 0,  
#79 0, 0, 0, 0, 0, 0,  
#80 0, 0, 0, 0, 0, 0,  
#81 0, 0, 0, 0, 0, 0,  
#82 0, 0, 0, 0, 0, 0,  
#83 0, 0, 0, 0, 0, 0,  
#84  
#85 0, 0, 0, 0, 0, 0,
```

• • •

```

773 0, 0, 0, 0, 0,
774 0, 0, 0, 0, 0,
775 0, 0, 0, 0, 0,
776 0, 0, 0, 0, 0,
777 0, 0, 0, 0, 0,
778 0, 0, 0, 0, 0,
779 0, 0, 0, 0, 0,
780 0, 0, 0, 0, 0,
781
782 0, 0, 0, 0, 0,
783 0, 0, 0, 0, 0,
784 0, 0, 0, 0, 0,
785 0, 0, 0, 0, 0,
786 0, 0, 0, 0, 0,
787 0, 0, 0, 0, 0,
788 0, 0, 0, 0, 0,
789 0, 0, 0, 0, 0];
790
791 CLASS_DETAILS_INPUT = [
792 E, T1, 30, 1, 0,
793 C1, T19, 60, 1, 0,
794 C2, T19, 60, 3, 0,
795 C3, T9, 80, 1, 0,
796 C4, T31, 80, 2, 3,
797 C5, T9, 80, 3, 0,
798 C6, T17, 80, 1, 0,
799 C7, T17, 80, 2, 3,
800 C8, T17, 80, 3, 0,
801 C9, T13, 80, 1, 0,
802 C10, T13, 80, 2, 3,

```

2:

```

8 %-----PRINTING-OUTPUT-----
9 output ["INSTRUCTORS SCHEDULE:\n"];
10 output
11 [if z==1 /\ j==MO then "Instructor: " ++ show_int(1,i) ++ "\n" else "" endif ++ if show_int(1,I_SCHEDULE[i,j,z]) != "1" then
12 show_int(1,I_SCHEDULE[i,j,z] -1) else "-" endif ++ "," ++
13 if z == length(SLOTS) /\ j == length(DAYS) then "\n" else "" endif ++
14 if j == FR then "\n" endif |
15 i in INSTRUCTORS, z in SLOTS, j in DAYS ];
16
17 output ["WEDNESDAY SCHEDULE:\n"];
18 output
19 [if s==1 then "Instructor: " ++ show_int(1,i) ++ "\n" else "" endif ++ if show_int(1,I_W_SCHEDULE[i,s]) != "1" then
20 show_int(1,I_W_SCHEDULE[i,s] -1) else "-" endif ++ "\n" ++
21 if s == length(W_SLOTS) then "\n" else "" endif |
22 i in INSTRUCTORS, s in W_SLOTS];

```

## Appendix 3

This appendix includes the questions of the questionnaire that was used in the interface evaluation stage.

### Scheduling Tool Usability Evaluation

This questionnaire was designed for the evaluation phase of the Scheduling tool, developed by Sohaib Nassar for the Department of Computer Science.

Do you feel that the website was easy to navigate? \*

☐ Yes

☐ No

☐ Maybe

Is the content on the website clear and understandable? \*

☐ Yes

☐ No

☐ Maybe

Do you find the fonts and colors used in the website to be easily recognizable and readable? \*

☐ Yes

☐ No

☐ Maybe

How satisfied are you by the graphical design on the website? \*

	1	2	3	4	5	
Not satisfied at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very satisfied

How did the website compare to other similar websites you've used in the past? \*

	1	2	3	4	5	
Much worse	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Much better

How easy did you find it to understand the purpose of each page on the website? \*

Very hard      1      2      3      4      5      Very easy

☐   ☐   ☐   ☐   ☐

How satisfied are you by the way the schedules are presented on the website? \*

Not satisfied at all      1      2      3      4      5      Very satisfied

☐   ☐   ☐   ☐   ☐

Did you encounter any issues while filling out any forms? \*

☐ Yes

☐ No

If the answer to the previous question was Yes, please state what was the issue you faced

Long-answer text

Please rate your overall experience while using the website \*

Not satisfied at all      1      2      3      4      5      Very satisfied

☐   ☐   ☐   ☐   ☐

Is there anything you would change, add or remove?

Long-answer text