Individual Thesis

IMPLEMENTATION OF A STUDENT SELECTION SYSTEM FOR INDIVIDUAL THESIS PROJECTS USING CONSTRAINT SATISFACTION TECHNIQUES

Panagiotis Kourkoulis

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

December 2025

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Implementation of a student selection system for individual thesis projects using constraint satisfaction techniques

Panagiotis Kourkoulis

Supervising Professor
Dr Yiannis Dimopoulos

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

December 2025

Acknowledgments

I would like to, first of all, give a special thanks to my supervisor for my Individual Thesis, Dr Yiannis Dimopoulos, for the guidance and support he provided me with throughout the whole thesis, helping me to fully understand the objectives of the system and guiding me in the creation of my constraint satisfaction model for student assignment.

Additionally, I would like to thank my family, friends, and fellow students who pushed me every day and encouraged me to give my all and achieve my goals.

Again, I would like to thank all the aforementioned individuals, as without their contribution, my academic journey over the past 4 years would not have been the same.

Abstract

This thesis presents the design and implementation of a web-based system aimed at streamlining the assignment of supervisor teachers to students for their Thesis Project in the Computer Science Department of the University of Cyprus. The existing manual process, based on email exchanges and editable Word Documents, has proven to be error-prone, inefficient, and difficult to manage. To address these challenges, the proposed system automates the collection of preference lists from both students and teachers, provides a centralized interface for the Thesis Coordinator and department secretary, and integrates a matching algorithm to generate supervisor assignments based on the submitted preferences. The system improves transparency, reduces the risk of mismatches, and simplifies the overall workflow. User interfaces were developed based on my experience as a student who has been through this process, feedback from my supervisor, and feedback from administrative staff members, ensuring usability and alignment with the needs of each user group. The result is a more reliable, organized, and user-friendly solution for managing thesis supervision assignments.

Table of Contents

Chapter 1
Introduction

1.1 Background Information and Motivation

Currently, the process in place for the assignment of supervisor teachers to students consists of a lot of manual tasks, which makes the process more complex and less organized, while also, at the same time, increasing the likelihood of errors. The current procedure begins with the Thesis Coordinator, who is a selected academic member of the department, who works closely with the secretary of the department to prepare the preference forms for the students and the teachers. The forms have fields where the students rank four teachers that they would prefer to be supervised by (for the student forms), and the teachers select four students that they would prefer to supervise (for the teacher forms). The forms, after being prepared, are sent by the secretary of the department to the students and the teachers, who then have to complete them before sending them back to the secretary of the department. The secretary, after receiving all the forms, has to manually assign supervisors to the students based on all the preference lists they received. This process poses many risks for possible errors, such as a mismatch from the secretary when assigning the supervisors, or a form not being received by the secretary. After the secretary finishes the assignments, they send a new list containing all the assignments made, along with another list that contains the names of the students who were not assigned to a teacher in the last assignment round and are referred to a subsequent assignment round, and another list that contains the teachers who can receive assignments in the next round based on their remaining capacity. For the next assignment round, the aforementioned process has to be repeated with the students who remained unassigned and the teachers with remaining capacity.

With the implementation of this system, the students will submit their preferences through the website, the teachers will be able to view the students who have selected them, along with the preference rank they placed them in, and will also be able to submit their own preferences list. The Thesis Coordinator and the secretary of the department will be able to view an overview of all the preferences submitted by both students and teachers, and they will also

have the option to run the matching algorithm straight from the website. After successfully running the matching algorithm, they will view a table with the assignments made, along with the list of unmatched students and a list with the remaining capacities of the teachers.
The goal of this project was to implement this process in a web-based system in order to simplify the whole procedure for everyone involved. Firstly, the students and the teachers will no longer need to manually fill out their preference lists and send them to the secretary of the department. Moreover, the secretary will no longer need to track down all the submitted lists, which could lead to lost lists, and will not need to go through the process of manually assigning teachers to students, which introduces the risk of mismatching errors. Overall, the whole process will be easier and safer.

1.2 System Integration

The system will, at a later time, be integrated into the existing infrastructure of the Computer Science Department of the University of Cyprus. The users will be able to log into the system using their university credentials, username, and password, and then, based on their role, they will be redirected to the appropriate page (students' page, teachers' page, and admins' page). The table used to store the users' credentials is similar to the one already used by the university to allow for seamless integration. The table stores the internal database identifier (database ID) of the user, their full name, username, user role (Student, Faculty, Admin), and hashed password. Furthermore, the technology used for the application layer, which is Express on top of Node.js, is also compatible with the infrastructure of the university.

Chapter 2

Constraint Satisfaction Problems and Programming

2.1 Introduction

In this chapter, we introduce the Constraint Satisfaction Problems (CSPs) declarative programming paradigm, which was used in this thesis to model and solve the problem of assigning teachers to students.

The tool used for the model is Minizinc [10], a high-level modeling language for Constraint Satisfaction Problems. This chapter explains the concept of Constraint Satisfaction Problems and how it was used in the context of the teacher assignment.

2.2 Constraint Satisfaction and Minizinc Language

2.2.1 Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs) are a fundamental concept in computer science and Artificial Intelligence. A Constraint Satisfaction Problem is defined as a problem where the goal is to find values for a set of variables that satisfy the constraints defined in the problem. A typical Constraint Satisfaction Problem consists of: the variables (e.g., X_1, X_2, X_3, …, X_n) to which we want to assign values that satisfy the constraints, the domains (e.g., D_1, D_2, D_3, …, D_n) that define the possible values that a variable can have and each variable has its own domain, and the constraints which are rules that define the allowable combinations of values that can be assigned to variables (e.g., $X_1 \neq X_2$).

The objective is to assign values to all variables, from their respective domains, in a way that all the constraints are satisfied simultaneously.

Constraint Satisfaction Problems appear in a wide range of real-world applications, such as scheduling (e.g., university timetable and lecture room allocation), resource allocation, and optimization in logistics and manufacturing (e.g., route planning for delivery vehicles).

Solving Constraint Satisfaction Problems can be approached using brute-force methods by trying all the possible value assignments to variables, but more efficient techniques have been suggested. These techniques include: backtracking and constraint propagation.

Example of Solving a simple CSP using Backtracking and Arc Consistency techniques:
Problem:

- Variables: X_1, X_2
- Domains: $D(X_1) = \{1,2,3\}$, $D(X_2) = \{2,3\}$
- Constraints:
  1. $X_1 < X_2$
  2. $X_1 \neq X_2$

Step 1: Apply Arc Consistency – we check each arc and remove unsupported values from the domains.

Arc: $X_1 \rightarrow X_2$ ($X_1 < X_2$)

- For $X_1 = 1 \rightarrow X_2$ can be 2 or 3 $\Rightarrow$ OK – no values removed from $D(X_1)$
- For $X_1 = 2 \rightarrow X_2$ can be 3 $\Rightarrow$ OK – no values removed from $D(X_1)$
- For $X_1 = 3 \rightarrow X_2$ must be >3 – no values in $D(X_2) \Rightarrow$ remove 3 from $D(X_1)$

Now: $D(X_1) = \{1,2\}$

Arc: $X_2 \rightarrow X_1$

- For $X_2 = 2 \rightarrow X_1$ can be 1 $\Rightarrow$ OK – no values removed from $D(X_2)$
- For $X_2 = 3 \rightarrow X_1$ can be 1 or 2 $\Rightarrow$ OK – no values removed from $D(X_2)$

Now: $D(X_2) = \{2,3\}$ – no change

Step 2: Backtracking search – assigning values to variables one at a time and backtracking in case of unsatisfied constraints.

1. Assign $X_1 = 2 \rightarrow$ no constraints violated – OK

2. Assign $X_2 = 2 \rightarrow$ violates $X_1 \neq X_2$ – backtrack (unassign $X_2 = 2$ and assign other value from $D(X_2)$)
3. Assign $X_2 = 3 \rightarrow$ no constraints violated – OK

Solution: $X_1 = 2$, $X_2 = 3$

2.2.2 Introduction to Minizinc

Minizinc is a high-level, declarative modeling language designed for describing constraint satisfaction and optimization problems. The declarative syntax of the language allows users to describe what the problem is and not how it should be solved. This makes it easier to model Constraint Satisfaction Problems, as the syntax of Minizinc is also very straightforward and understandable. Moreover, Minizinc offers a wide range of built-in. arithmetic, logical, and global constraints such as alldifferent, which enforces that the values of specific variables should all be different from each other, and cumulative, which can be used in scheduling problems to ensure that resources over time do not exceed a given capacity. Additionally, it offers optimization for the minimization and maximization of selected parameters. Also, Minizinc offers modularity as it supports reusable components through modules and 'include' files. Finally, another very important feature of Minizinc is that it is solver-independent, meaning that problems modelled in Minizinc can be solved using different solvers such as Gecode, Chuffed, and COIN-BC. This flexibility is very important as, depending on the problem structure, different solvers can perform differently.

```
int: W; %Number of workers
int: T; %number of tasks
int: p_unassigned; %Penalty of leaving a task unassigned

set of int: Workers = 1..W; %Each value in the set corresponds to a worker
set of int: Tasks = 1..T; %Each value in the set corresponds to a task

%Cost of assigning worker w to task t
array[Workers, Tasks] of int: cost;

%Feasibility: 1 if worker w can do task t, else 0
array[Workers, Tasks] of 0..1: can_do;

%Maximum workload capacity of each worker
array[Workers] of int: max_tasks;

%Decision variable: x[w,t] = 1 if worker w is assigned to task t, else 0
array[Workers, Tasks] of var 0..1: x;

%Decision variable: u[t] = 1 if task t is left unassigned, else 0
array[Tasks] of var 0..1: u;

%Constraint: a task can be assigned to no worker or at most one worker
constraint forall(t in Tasks)(sum(w in Workers)(x[w,t]) + u[t] = 1);

%Constraint: only assign worker if worker can do the task
constraint forall(w in Workers, t in Tasks)(x[w,t] <= can_do[w,t]);

%Constraint: Workload limit per worker
constraint forall(w in Workers)(sum(t in Tasks)(x[w,t]) <= max_tasks[w]);

%Objective: Minimize total assignment cost and penalties for unassigned tasks
solve minimize sum(w in Workers, t in Tasks)(cost[w,t] * x[w,t]) + p_unassigned * sum(t in Tasks)(u[t]);
```

Figure 1: Simple Minizinc model example. Assign workers to tasks such that we minimize the assignment cost and the penalty for unassigned tasks.

2.2.3 Application in Thesis

The Minizinc model used in this thesis was written as a .mzn file and executed through Python using the official minizinc-python library [11]. More specifically, when the declared endpoint (/admin/run-matching) is called, JavaScript spawns a child process using the spawn module from the child_process library to run the aforementioned Python script. The Minizinc model included sections to define the students' and teachers' preferences data, assignment and capacity constraints, and optimization criteria. Once executed, Minizinc returned the solution that minimized the number of students that were left unassigned. This optimization statement allowed us to get the solution that matched the most students to teachers while satisfying the assignment constraints.

Minizinc was chosen for its expressiveness and its ability to work with a range of solvers, which made it possible to improve the performance by choosing the appropriate solver for the specific problem structure.

In summary, Minizinc proved to be a powerful tool for modeling and solving the teacher assignment problem. Its syntax, paired with its optimization capabilities and solver compatibility, allowed for concise and readable problem encodings and enabled the generation of fast and high-quality solutions.

2.2.4 Function and Constraint Explanations

For this project, built-in functions of Minizinc were used to model and solve the problem. Firstly, the array2d function was used. This function is called as follows: arra2d(<rows>, <columns>, <flat-list>), and it rearranges the flat list that it takes as input into a 2d array with rows and columns based on the values given as first and second parameters to the function respectively.

```
int: rows = 3;
int: cols = 4;

array[1..rows*cols] of int: flat_list = [5, 3, 8, 6, 2, 9, 1, 4, 7, 0, 3, 2];

array[1..rows, 1..cols] of int: grid = array2d(1..rows, 1..cols, flat_list);
```

Figure 2: Simple example of formatting a flat list into a 2D array in Minizinc, using the array2d built-in function.

The exists function was also used. The exists function is a quantifier used to check if at least one element in a set or array satisfies a condition. The formal call syntax for the function is: exists(i in <IndexSet>)(<Condition(i)>), where <IndexSet> is a set or range of values to iterate over, and <Condition(i)> is a boolean expression involving i. The function will return true if any value in the IndexSet makes Condition(i) true.

```
int: n = 5;
array[1..n] of int: ages = [25, 42, 67, 33, 59];

constraint (exists(i in 1..n)(ages[i] > 60));
```

Figure 3: Simple example of the exists function that checks if at least one of the values in the ages array is over 60.

Moreover, the let–in construct was used. This construct is a scoping expression used to introduce temporary local definitions that are only visible inside a single expression. The formal syntax for the construct is: let { <local declarations> } in <expression>, where the <local declarations> is one or more definitions (e.g., int: x = 5; set of int: A = 1..3;), and <expression> is any Minizinc expression that can use those local declarations/definitions.

```
set of int: S = 1..3;
array[S] of int: vals = [10, 20, 30];

array[S] of int: doubled =
  [ let { int: temp = vals[s] * 2 }
    in temp
    | s in S ];
```

Figure 4: Simple example of the let-in construct that is used to create a new array that stores the doubled values of an original array. The '|' symbol is used in Minizinc inside comprehensions (like arrays and sets), and it separates the expression being built from the iteration/filtering condition.

Additionally, the bool2int function was used. This function is used to convert a boolean value (true or false) into an integer. True becomes 1 and False becomes 0. The formal call syntax of the function is: bool2int(<boolean-value>).

```
int: n = 5;
array[1..n] of int: ages = [12, 18, 25, 17, 30];

array[1..n] of int: is_adult = [bool2int(ages[i] >= 18) | i in 1..n];
```

Figure 5: Example of the bool2int function in Minizinc that is used to create an array that holds 1 if a value is greater than or equal to 18 and 0 if otherwise.

Chapter 3

Problem Definition and Methodology

3.1 Problem Definition

3.1.1 Research Context and Motivation

Assigning teachers to supervise students for their Thesis Project is a time-consuming and error-prone process that is currently done manually. Also, the process that teachers and students follow to submit their preferences is outdated, and it itself poses risks for errors. At the same time, this is a mandatory process that has to be completed by all students of the department, so creating a modern solution that is simpler and eliminates the risk of mismatching errors was important.

The task of assigning teachers to students takes into account the preferences of the students and the teachers alike, as well as the capacity of the teachers, meaning how many students they can supervise. Satisfying the preferences of both aforementioned parties to the maximum degree while also respecting the teacher's capacities at the same time presents significant research challenges.

3.1.2 Formal Problem Statement

This study addresses the problem of mutual preference-based student-teacher assignment for their thesis, where both teachers and students express ranked preferences over each other. The objective is to assign teachers to students in a way that respects mutual preferences, capacity constraints, and prioritizes optimal satisfaction of student choices.

Let:

$S = \{s_1, s_2, \ldots, s_n\}$ be the set of students

and

$T = \{t_1, t_2, \ldots, t_m\}$ be the set of teachers

Each student s_i provides a ranked list of their four preferred teachers, denoted as:

$P_s(s_i) = [t_{i1}, t_{i2}, t_{i3}, t_{i4}]$

Each teacher t_j provides a ranked list of their four preferred students, denoted as:

$P_t(t_j) = [s_{j1}, s_{j2}, s_{j3}, s_{j4}]$

Each teacher t_j has supervision capacity $C(t_j)$, indicating the maximum number of students they can supervise.

The goal is to derive a mapping f: $S \rightarrow T \cup \{0\}$ where $f(s_i) = t_j$ if student s_i is assigned to teacher t_j, or $f(s_i) = 0$ if the student remains unassigned.

From f we can derive the mapping $g: T \rightarrow 2^S$ where $g(t_j) = \{ s_i \in S \mid f(s_i) = t_j \}$

The constraints defined for the problem are:

1. Capacity constraint: no teacher can be assigned more students than their capacity allows – how many students they can supervise.
   $|g(t_j)| \leq C(t_j) \; \forall \; t_j \in T$

2. Mutual Preference constraint: a student s_i can only be assigned to a teacher t_j if:
   $t_j \in P_s(s_i)$ and $s_i \in P_t(t_j)$

3. Coherence constraint: the mappings f and g must be consistent
   $f(s_i) = t_j \Rightarrow s_i \in g(t_j)$
   and
   $g(t_j) = \{ s_i \mid f(s_i) = t_j \}$
4. Preference Prioritization (Blocking Pairs): if a student s_i is assigned to a lower-ranked teacher t_a in their preferences, it must be because any higher-ranked teacher t_b in their list either:
   - Has not selected the student in their preferences
     $s_i \notin P_t(t_b)$
   - Has reached their capacity limit
     $|g(t_b)| = C(t_b)$

Example of a Student-Teacher Assignment Problem:

Let the set of students be:

$S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$

And the set of teachers be:

$T = \{t_1, t_2, t_3\}$

Each teacher has a supervision capacity of 2:

$C(t_1) = C(t_2) = C(t_3) = 2$

Each student submits a ranked list of preferred teachers:

$P_s(s_1) = [t_1,t_2,t_3]$

$P_s(s_2) = [t_1,t_3,t_2]$

$P_s(s_3) = [t_2,t_1,t_3]$

$P_s(s_4) = [t_2,t_3,t_1]$

$P_s(s_5) = [t_3,t_1,t_2]$

$P_s(s_6) = [t_3,t_2,t_1]$

Each teacher submits a list of preferred students:

$P_t(t_1) = [s_1,s_2,s_3]$

$P_t(t_2) = [s_3,s_4,s_1]$

$P_t(t_3) = [s_5,s_6,s_2]$

Decision Variable f where $f(s_i) = t_j$ if student s_i is assigned to teacher t_j, or $f(s_i) = 0$ if the student remains unassigned.

The assignment must satisfy the following constraints:

1. No teacher may supervise more students than their capacity.
2. A student may only be assigned to a teacher if both have selected each other.
3. Each student may be assigned to at most one teacher.

4. If a student is assigned to a lower-ranked teacher, then all higher-ranked teachers in their preference list either did not select the student or have reached their capacity.

The objective is to maximize student satisfaction while minimizing the number of unassigned students. This is achieved by prioritizing higher-ranked teachers in each student's preference list and minimizing the total number of students who are unassigned (assigned to 0)

Example of a Valid Assignment (Solution):

$f(s_1) = t_1$

$f(s_2) = t_1$

$f(s_3) = t_2$

$f(s_4) = t_2$

$f(s_5) = t_3$

$f(s_6) = t_3$

This assignment is valid because:

- All teachers supervise exactly two students, respecting capacity limits.
- All assigned pairs satisfy mutual preferences.
- No student is assigned to more than one teacher.
- No blocking pairs exist, as no student-teacher pair would prefer to deviate from the assignment.

Example of an Invalid Assignment (Not a Solution):

$f(s_1) = t_1$

$f(s_2) = t_1$

$f(s_3) = t_1$

$f(s_4) = t_2$

$f(s_5) = t_3$

$f(s_6) = t_3$

This assignment is not valid because teacher t_1 is assigned three students, exceeding its capacity constraint.

3.2 Similar Problems

3.2.1 Stable Marriage Problem

3.2.1.1 Problem Definition

The Stable Marriage Problem involves matching equal numbers of men and women, each with a ranked preference list for the other group, to find a stable matching where no man and woman would prefer each other to their assigned partners. A stable matching has no "blocking pairs" – individuals who are matched but would both prefer to be with each other than their current partners.

3.2.1.2 Problem Modeling as CSP

The stable marriage problem can be modelled as a Constraint Satisfaction Problem as follows:

Parameters:

n: number of women and men (number of women is equal to number of men)

$rank_m(m_i, w_j)$: The ranking list of the men – position of woman w_j in man m_i 's preferences

$rank_w(w_j, m_i)$: The ranking list of the women – position of man m_i in woman w_j 's preferences

Variables:

Let:

$M = \{m_1, m_2, \ldots, m_n\}$ be the set of Men

and

$W = \{w_1, w_2, \ldots, w_n\}$ be the set of Women

Define:

X_i : the partner/wife assigned to man m_i

Y_j : the partner/husband assigned to woman w_j

Each value X_i and Y_j represents a match and must be assigned a value from the opposite set

Domains:

$D(X_i) = W$: Each man can be matched to a woman

$D(Y_j) = M$: Each woman can be matched to a man

Constraints:

1. Unique assignments constraint: All men must be matched to a different woman, and all women must be matched to a different man – no two (or more) men can have the same wife, and no two (or more) women can have the same husband.
   Alldifferent$(X_1, X_{2, \ldots,} X_n)$: all men should have different wives from one another
   Alldifferent$(Y_1, Y_{2, \ldots,} Y_n)$: all women should have different husbands from one another

2. Coherence constraint: if a man m_i is matched to a woman w_j, then w_j is matched to m_i and vice versa.
   $X_i = w_j \Leftrightarrow Y_j = m_i$

3. Blocking Pairs constraint: prevent blocking pairs (m_i, w_j) that would prefer each other over their current partners. For each unmatched pair, should enforce:
   $\neg\, (rank_m(m_i, w_j) < rank_m(m_i, X_i) \ \wedge rank_w(w_j, m_i) < rank_w(w_j, Y_j))$

3.2.2 Hospitals – Residents Problem

3.2.2.1 Problem Definition

The Hospitals – Residents problem involves matching a set of residents to a set of hospitals, where each resident has a ranked preference list of hospitals, and each hospital has a ranked preference list of residents, along with a capacity indicating how many residents it can accept. The goal is to find a stable matching where no resident and hospital would both prefer to be

matched to each other over their current assignments. A stable matching has no blocking pairs – a resident and hospital who are not matched together but would both prefer each other to their current assignments, and where the hospital either has an unfilled position or prefers the resident over at least one of its current assignees.

3.2.2.2 Problem Modeling as CSP

The Hospitals – Residents problem can be modelled as a Constraint Satisfaction Problem as follows:

Parameters:

r: the number of residents

h: the number of hospitals

capacity[k]: capacity of hospital h_k

$rank_r(r_i, h_k)$: position of hospital h_k in resident r_i 's preference list

$rank_h(h_k, r_i)$: position of resident r_i in hospital h_k 's preference list

Variables:

Let:

$R = \{r_1, r_2, ..., r_r\}$ be the set of residents

$H = \{h_1, h_2, ..., h_h\}$ be the set of hospitals

Define:

X_i : the hospital assigned to resident r_i (0 if unassigned)

Y_k : the set of residents assigned to hospital h_k

Domains:

$D(X_i) = \{0\} \cup H$: each resident can remain unassigned (0) or be matched to a hospital

$D(Y_k) \subseteq R$: each hospital can have a subset of residents assigned to it

Constraints:

1. Capacity constraint: a hospital cannot be assigned more residents than its capacity allows.

   $|Y_k| \leq capacity[k] \; \forall \; h_k \in H$

2. Coherence constraint: if a resident is assigned to a hospital, then the resident should belong in the set of assigned residents to that hospital, and vice versa.

   $X_i = h_k \Leftrightarrow r_i \in Y_k$

3. Unique assignment constraint: each resident is assigned to at most one hospital

   $X_i \in \{0\} \cup H$

4. Blocking pairs constraint:

   $\neg \; (rank_r(r_i, \; h_k) < rank_r(r_i, \; X_i) \wedge (|Y_k| < capacity[k] \; \vee \exists r_j \in Y_k \; : \; rank_h(h_k, \; r_i \;) < rank_h(h_k, \; r_j \;)))$

3.2.3 Comparison with Related Matching Problems

Both the Stable Marriage Problem and the Hospitals-Residents Problem share fundamental characteristics with the Students-Teachers Assignment Problem addressed in this thesis:

1. Mutual Preference: All three problems involve two distinct sets of entities that express ranked preferences over members of the opposite set.
   - In the Stable Marriage Problem, men and women rank each other.
   - In the Hospitals-Residents Problem, residents rank hospitals, and hospitals rank residents.
   - In the Students-Teachers Problem, students rank teachers, and teachers select students.

2. Matching: Each problem requires finding a matching that satisfies specific constraints.
   - In the Stable Marriage Problem, the constraint is stability – no blocking pairs.

- In the Hospitals-Residents Problem, the constraints are stability and hospital capacity limits.
- In the Students-Teachers Problem: the constraints are teacher capacity limits, mutual selection, and avoidance of blocking pairs.

3. Goal: All three problems aim to produce assignments that are as fair and optimal as possible within the given constraints.
    - The Stable Marriage Problem seeks a stable matching where no pair would prefer each other over their current partners.
    - The Hospitals-Residents Problem seeks a stable matching between the residents and the hospitals, while respecting the capacities of the hospitals.
    - The Students-Teachers Problem seeks to maximize student satisfaction while respecting the teacher capacities and mutual preferences.

Chapter 4

Constraint Programming Solution

4.1 Constraint Satisfaction Problem

The objective was to assign teachers to students based on the preference lists submitted by all the teachers and the students. The solution should respect the preferences of everybody and should try to satisfy the rankings of the students. At the same time, the solution should respect the capacity limitations of the teachers so that no teacher is assigned more students than their capacity allows.

4.2 Minizinc Model Explanation

The model used in this project is defined in minizincModel.mzn, and consists of:

Parameters: (Figure 1)

- numStudents: the total number of students who have submitted preferences
- numTeachers: the total number of teachers with remaining capacity
- Students: the set with integer values 1 – numStudents
- Teachers: the set with integer values 1 – numTeachers
- [Array] studentsPreferences: 2D array that holds the preferences of each student. Each teacher selected by a student is denoted in the choices of the student with the index of the teacher in the set of Teachers as defined above.
- [Array] teacherPreferences: 2D array that holds the preferences for each teacher. Each student selected by a teacher is denoted in the choices of the teacher with the index of the student in the set of Students as defined above.
- [Array] teacherCapacity: 1D array that holds the capacity of each teacher. The index in the array is the index of the specific teacher from the Teachers set as defined above.

```
int: numStudents;
int: numTeachers;

set of int: Students = 1..numStudents;
set of int: Teachers = 1..numTeachers;

array[Students, 1..4] of int: studentsPreferences;

array[Teachers, 1..4] of int: teacherPreferences;

array[Teachers] of int: teacherCapacity;
```

Figure 1: Parameters of the Minizinc Model

Derived Parameters: (Figure 2)

- [Array] studentPrefers: 2D boolean array that stores whether a student prefers a teacher – studentPrefers[s,t] = true if student s has listed teacher t in their preferences list, = false if otherwise.
- [Array] teacherPrefers: 2D boolean array that stores whether a teacher prefers a student – teacherPrefers[t,s] = true if teacher t has listed student s in their preferences list, = false if otherwise.
- [Array] allowed: 2D boolean array that stores whether a student can be matched to a teacher based on the preferences of both. Uses the teacherPrefers and studentPrefers arrays to decide – allowed[s,t] = true iff studentPrefers[s,t] = true AND teacherPrefers[t,s] = true.
- [Array] rankS: 2D array that stores the ranking that each student selected for each teacher. If the student has selected a teacher more than once in their preferences list, then it stores the best ranking the student gave to the teacher. If the student has not selected the teacher, then it sets the ranking in the rankS array for that teacher by that student to 5 – rankS[s,t] = 5.

```
array[Students, Teachers] of bool: studentPrefers =
  array2d(Students, Teachers,
  [ exists(i in 1..4)(studentsPreferences[s,i] = t) | s in Students, t in Teachers ]);

array[Teachers, Students] of bool: teacherPrefers =
  array2d(Teachers, Students,
  [ exists(j in 1..4)(teacherPreferences[t,j] = s) | t in Teachers, s in Students ]);

array[Students, Teachers] of bool: allowed =
  array2d(Students, Teachers,
  [ studentPrefers[s,t] /\ teacherPrefers[t,s] | s in Students, t in Teachers ]);

array[Students, Teachers] of int: rankS =
  array2d(Students, Teachers,
    [ let {
        set of int: hits = { i | i in 1..4 where studentsPreferences[s,i] = t }
      } in if card(hits) > 0 then min(hits) else 5 endif
    | s in Students, t in Teachers ]);
```

Figure 2: Derived parameters of the Minizinc model

Decision Variables: (Figure 3)

- [Array] supervisor: 1D array that holds the assigned teacher/supervisor of each student. The length of the array is equal to the number of students, and the index of the array corresponds to the specific student from the Students set as defined above. The values that can be given to each position in the array are values from the Teachers set as defined above, and the value 0 if the student remains unassigned.
- [Array] x: 2D array that holds the assignment of each student. If a student s has been assigned to a teacher t, then x[s,t] = 1, otherwise x[s,t] = 0.
- [Array] unassigned_count: Integer value that represents the number of students who have remained unassigned.

```
array[Students, Teachers] of var 0..1: x;

array[Students] of var 0..numTeachers: supervisor;

var int: unassigned_count = sum(s in Students)(1 - sum(t in Teachers)(x[s,t]));
```

Figure 3: Variables of the Minizinc Model

Constraints: (Figure 4)

- Constraint 1: Coherence constraint – ensure that if a student s is assigned to a teacher t, then supervisor[s] = t or if unassigned then supervisor[s] = 0.
- Constraint 2: Ensures that a student can only be assigned to a teacher if both the teacher and the student have selected each other – uses the 'allowed' derived parameter.
- Constraint 3: Ensures that a student can be assigned to at most one teacher.
- Constraint 4: Ensures that the number of students assigned to a teacher does not exceed the capacity of the teacher.
- Constraint 5: Ensures that no blocking pairs occur. If a student s is assigned to a teacher t, but there exists a teacher a that s prefers to t, then that should be because either a has not selected s in their preference list or because a has reached their capacity.

```
%Constraint 1:
constraint forall(s in Students)(
  supervisor[s] = sum(t in Teachers)(t * x[s,t])
);
%Constraint 2:
constraint forall(s in Students, t in Teachers)(
  x[s,t] <= bool2int(allowed[s,t])
);
%Constraint 3:
constraint forall(s in Students)(
  sum(t in Teachers)(x[s,t]) <= 1
);
%Constraint 4:
constraint forall(t in Teachers)(
  sum(s in Students)(x[s,t]) <= teacherCapacity[t]
);
%Constraint 5:
constraint forall(s in Students, t in Teachers where rankS[s,t] <= 4)(
  (x[s,t] = 1) -> forall(b in Teachers where rankS[s,b] < rankS[s,t])(
        (not allowed[s,b]) \/ (sum(ss in Students)(x[ss,b]) >= teacherCapacity[b])
    )
  );
```

Figure 4: Constraints of the Minizinc Model

4.3 Integration of Minizinc Into the System

4.3.1 Python-Minizinc Library

Python offers an official Minizinc library that lets us run the model straight from Python. The library allows the definition of a model through the Model module, which takes as a parameter the .mzn file, the selection of a solver using the Solver module with the built-in lookup function, and the creation of an instance which allows us to give values to the parameters so the model can run on the specified data without having to create a .dzn file by parsing the data. Furthermore, through this approach, we can create a dataclass that stores the output of the model in a structured manner in order to work on the resulting data more easily and have the option to pass it to Node.js in order to manipulate it, and query the database to insert it into the table that stores the assignments.

4.3.2 Python-Minizinc Implementation

Firstly, the minizincModel.mzn file, which contained the model of the problem, was added to the project sub-folder dedicated to the backend of the website. The Model() module from the Minizinc library was then used to access the model from the Python script. Next, the lookup function of the Solver module was utilized to select the solver that would be used to solve the problem. Finally, the Instance module was used to define the values of the parameters expected by the model, instead of creating a separate .dzn data file. After the parameters were set up, the solve() function was called on the instance to run the model and obtain the result, which was stored in a variable called res, along with the solution extracted from the result, which was saved in a variable named sol.

```
here = Path(__file__).resolve().parent
model_path = here / "minizincModel.mzn"

model = Model(str(model_path))
model.output_type = MatchSolution
solver = Solver.lookup("gecode")
inst = Instance(solver, model)

inst["numStudents"] = payload["numStudents"]
inst["numTeachers"] = payload["numTeachers"]
inst["studentsPreferences"] = payload["student_prefs"]
inst["teacherPreferences"] = payload["teacher_prefs"]
inst["teacherCapacity"] = payload["teacher_capacity"]

res = inst.solve()
sol = res.solution
```

Figure 5: Model, Solver, Instance from Python script

The values that were given to the parameters, numStudents, numTeachers, studentsPreferences, teacherPreferences, and teacherCapacity, were fetched from the database by Node.js before calling the sub-process that spawned the Python script, and upon spawning the Python script, the data was passed to the child process by Node.js.

Then, the supervisor resulting list was extracted from the model's solution and passed to the parent process (Node.js) for handling. Only the supervisor list from the result was used, as it was the easiest to manipulate in order to extract the assignments, which would later be stored in the pairings table in the database.

For the extraction of the supervisor list from the result, a method given in the official Minizinc-Python documentation was used. A dataclass was created to store the output of the Minizinc model into variables, which could then be manipulated independently from one another.

```python
@dataclass
class MatchSolution:
    supervisor: List[int]
    x: Optional[Any] = None
    unassigned_count: Optional[int] = None
    objective: Optional[int] = None
    __output_item: InitVar[str] = None
```

Figure 6: Dataclass from Python script

4.3.3 Node.js Parent Process

In order to call the Python script from the endpoint '/admin/run-matching' in the admin.routes.js file, the spawn module from the child_process library, which comes pre-installed with Node.js, was used. This module allows spawning a child process within the Node.js code and passing arguments to the child process upon spawn. Firstly, a helper function was created to spawn the Python script process with the correct parameters.

```javascript
function runPython(pythonFile, payloadPython){
    return new Promise((resolve, reject) => {
        const process = spawn(PYTHON_BIN, [pythonFile], { stdio: ['pipe', 'pipe', 'pipe'], env: CHILD_ENV })
        let stdout = ''
        let stderr = ''

        process.stdout.on('data', (c) => (stdout += c.toString()))
        process.stderr.on('data', (c) => {
            const s = c.toString()
            console.error('[py]', s.trim())
            stderr += s
        })
        process.on('error', (err) => reject(err))
        process.on('close', (code) => {
            if( code !== 0 ){
                return reject(new Error(`Python exited with code ${code}. stderr: ${stderr || '(empty)'}`))
            }
            try{
                const json = JSON.parse(stdout)
                if (json && json.ok === false) {
                    json._stderr = stderr
                }
                resolve(json)
            }catch (err){
                reject(new Error(`Failed to parse Python JSON.\nstdout:\n${stdout}\n\nstderr:\n${stderr}`))
            }
        })
        process.stdin.write(JSON.stringify(payloadPython))
        process.stdin.end()
    })
}
```

Figure 7: Helper function to call the Python child process in Node.js

In the '/admin/run-matching' endpoint code, the database was first queried to retrieve the database IDs of the students who had submitted preferences and were not yet assigned to a teacher, the database IDs of the teachers who had remaining capacity, the preferences of the students, the preferences of the teachers, and the remaining capacities of the teachers.

```
const [studentIdsRows] = await pool.execute(
    `
    SELECT sp.student_id
    FROM student_preferences sp
    LEFT JOIN pairings p ON p.student_id = sp.student_id
    WHERE p.student_id IS NULL OR p.teacher_id IS NULL
    ORDER BY sp.student_id
    `
)
const studentIds = studentIdsRows.map(r => r.student_id)


const [teacherIdsRows] = await pool.execute(
    `
    SELECT u.id
    FROM usersTest u
    JOIN teacher_capacity tc ON tc.teacher_id = u.id
    WHERE u.user_role = 'Faculty' AND tc.capacity_remaining > 0
    ORDER BY u.id
    `
)
```

Figure 8: Node.js /run-matching queries (a)

```
const [studentPrefsRows] = await pool.execute(
    `
    SELECT student_id, preference1, preference2, preference3, preference4
    FROM student_preferences
    WHERE student_id IN (${studentIds.map(()=>'?').join(',')})
    ORDER BY student_id
    `,
    studentIds
)

const [teacherPrefsRows] = await pool.execute(
    `
    SELECT teacher_id, preference1, preference2, preference3, preference4
    FROM teacher_preferences
    WHERE teacher_id IN (${teacherIds.map(()=>'?').join(',')})
    ORDER BY teacher_id
    `,
    teacherIds
)

const [capacityRows] = await pool.execute(
    `
    SELECT teacher_id, capacity_remaining
    FROM teacher_capacity
    WHERE teacher_id IN (${teacherIds.map(() => '?').join(',')})
    ORDER BY teacher_id
    `,
    teacherIds
)
```

Figure 9: Node.js /run-matching queries (b)

The issue was that the database IDs of the teachers and the students, which were used in the table that stored the preferences, the table that stored the remaining capacities of the teachers, as well as in all other tables in the database, were not continuous indices, but Minizinc expects the students and the teachers to have continuous indices in the parameter lists. Take the example of the user table below:

id	name	username	user_role	password
1	Panagiotis Kourkoulis	pkourk02	Student	…
2	Yiannis Dimopoulos	yiannisdim	Faculty	…
3	Panagiotis Kolios	pkolios	Faculty	…
4	Georgia Papadopoulou	gpapad04	Student	…
5	Kyriakos Poyiadjis	kpoyia03	Student	…
6	Vassos Vasileiou	vasosvas	Faculty	…
7	Panagiotis Ioannides	pioann04	Student	…
8	Costas Pattichis	costaspatt	Faculty	…

Table 1: Users table example

And the preference tables based on example preferences submitted by students and teachers:

id	student_id	preference1	preference2	preference3	preference4	created_at
1	1	2	3	6	8	…
2	4	3	6	8	2	…
3	5	6	8	2	3	…
4	7	8	2	3	6	…

Table 2: Student Preferences table example

id	teacher_id	preference1	preference2	preference3	preference4	created_at
1	2	1	4	5	7	…
2	3	4	5	7	1	…
3	6	5	7	1	4	…
4	8	7	1	4	5	…

Table 3: Teacher Preferences table example

These preferences, if given to Minizinc unprocessed, would look like this:

studentsPreferences = [[2,3,6,8], [3,6,8,2], [6,8,2,3], [8,2,3,6]]

teacherPreferences = [[1,4,5,7], [4,5,7,1], [5,7,1,4], [7,1,4,5]]

But this creates indexing errors in Minizinc since the indices are based on the database IDs, which are not continuous, as Minizinc expects. For this reason, in the Node.js code, mappings for the teachers' and the students' database IDs were created, and new preference tables for each group were generated, where all database IDs were replaced with the corresponding mappings. So the mappings would look like this:

Key (Student ID)	Value (Index)
1	1
4	2
5	3
7	4

Table 4: Student mapping example

Key (Teacher ID)	Value (Index)
2	1
3	2
6	3
8	4

Table 5: Teacher mapping example

Now, using the mapped teacher and student indices, new preference arrays were created by replacing the database IDs with the new mapped indices:

id	student_id	preference1	preference2	preference3	preference4	created_at
1	1	1	2	3	4	…
2	2	2	3	4	1	…
3	3	3	4	1	2	…
4	4	4	1	2	3	…

Table 6: Student Preferences table with mapped indices example

id	teacher_id	preference1	preference2	preference3	preference4	created_at
1	1	1	2	3	4	…
2	2	2	3	4	1	…
3	3	3	4	1	2	…
4	4	4	1	2	3	…

Table 7: Teacher Preferences table with mapped indices

Now, when the preference lists are created based on the newly mapped preferences, the arrays produced match the format that minizinc expects with continuous indices:
studentsPreferences = [[1,2,3,4], [2,3,4,1], [3,4,1,2], [4,1,2,3]]
teacherPreferences = [[1,2,3,4], [2,3,4,1], [3,4,1,2], [4,1,2,3]]

The same mapping logic was used for the capacities of the teachers. The remaining capacity of each teacher was fetched from the database, and an array was created using the mapping of the teacher IDs, as expected from Minizinc, for the teacherCapacity parameter.

```
const studentMap = new Map(studentIds.map((id, i) => [id, i + 1]))
const teacherMap = new Map(teacherIds.map((id, i) => [id, i + 1]))

const teacherSet = new Set(teacherIds)
const badTeacherRefs = []
for (const row of studentPrefsRows) {
    for (const tId of [row.preference1, row.preference2, row.preference3, row.preference4]) {
        if (!teacherSet.has(tId)){
            badTeacherRefs.push(tId)
        }
    }
}

if (badTeacherRefs.length) {
    const missing = Array.from(new Set(badTeacherRefs)).sort((a,b)=>a-b)
    return res.status(400).json({
        error:
        `Student prefs reference teacher_ids without submissions: ${missing.join(', ')}. ` +
        `Either collect those teachers' prefs or change the teachers fethced.`,
    })
}

function toValidIndex(val, max) {
    return (Number.isInteger(val) && val >= 1 && val <= max) ? val : 1
}

const student_prefs = studentPrefsRows.map(row => {
    const prefs = [row.preference1, row.preference2, row.preference3, row.preference4]
    return prefs.map(tId => toValidIndex(teacherMap.get(tId), teacherIds.length))
})

const teacher_prefs = teacherPrefsRows.map(row => {
    const prefs = [row.preference1, row.preference2, row.preference3, row.preference4]
    return prefs.map(sId => toValidIndex(studentMap.get(sId), studentIds.length))
})
```

Figure 10: Node.js code to create the mappings for the IDs and create the appropriate preference arrays.

```
const capacityMap = new Map(capacityRows.map(r => [r.teacher_id, r.capacity_remaining]))

const teacher_capacity = teacherIds.map(id => {
    if (!capacityMap.has(id)) {
        throw new Error(`No capacity entry found for teacher_id ${id}`)
    }
    return capacityMap.get(id)
})
```

Figure 11: Node.js code to create the mapping for the capacity array.

Then, these arrays were added to the payload that would be passed as arguments to Python, so then, in the Python script, these arrays were used as the values for the studentsPreferences, teacherPreferences, and teacherCapacity parameters in the model. Along with the two preference lists and the capacity list, in the payload for the Python script, the number of teachers and the number of students were also given, as it was expected by the Minizinc model for the parameters: numStudents and numTeachers.

Then, after the model had completed running and the Python script had returned, the supervisor list was passed to the parent process (Node.js) by the Python script. The teacher mappings were then used to un-map the indices from the supervisor array to the correct database IDs. Based on the unmapped IDs, the pairs were inserted into the database table 'pairings', where the database IDs of the students and their assigned teacher were stored. The entries in the 'pairings' table based on the above example would be:

id	student_id	teacher_id	created_at
1	1	2	…
2	4	3	…
3	5	6	…
4	7	8	…

Table 8: Resulting Pairings table example.

```
const result = await runPython(PY_PATH, payloadPython)
if(!result || !result.ok || !Array.isArray(result.assignments)){
    return res.status(500).json({ error: "Solver error", details: result})
}
```

Figure 12: The code in Node.js to run the Python script using the helper function (Image 8)

```
for (let sIdx = 0; sIdx < studentIds.length; sIdx++) {
    const student_id = studentIds[sIdx]
    const teacherIndex = result.assignments[sIdx]
    const teacher_id = (Number.isInteger(teacherIndex) && teacherIndex >= 1 && teacherIndex <= teacherIds.length) ? teacherIds[teacherIndex - 1] : null

    await pool.execute(
        `
        INSERT INTO pairings (student_id, teacher_id)
        VALUES (?, ?)
        ON DUPLICATE KEY UPDATE teacher_id = VALUES(teacher_id)
        `,
        [student_id, teacher_id]
    )
    inserted++
}
```

Figure 13: The code in Node.js to unmap the indices of the teachers in the Minizinc result and insert the assigned pairs in the 'pairings' table in the database

Chapter 5

Software Technologies

5.1 Introduction

This chapter provides an overview of the different user groups that would have access to the system, the architecture of the system, and the different tools and frameworks used to develop the system, including front-end technologies for building an intuitive user interface, back-end technologies for handling the application logic and API endpoints, and database solutions for data storage.

5.2 User Categories

When designing and developing a system, the users who will interact with it must be accounted for. Each type of user has to complete different tasks and thus has to have access to different information.

There will be 3 types of users that will have access to the present system.

Faculty Members of the Computer Science Department of the University of Cyprus

Each teacher will have access to the data that is relevant to their tasks, which include selecting preferences for which students to supervise and managing their assigned students. More specifically, they will be able to view the students who have selected them in their preference lists and where they ranked them. Moreover, they will be able to select the students they would prefer to supervise from a list of all the available students. Additionally, they will be able to see the preference list they have submitted, along with the option to modify and update it, as well as the students whom they have been assigned so far.

Students of the Computer Science Department of the University of Cyprus

Each student will be able to select the four teachers, from the list of the department's teachers, who they would prefer to be supervised by. After submitting their preferences,

they will be able to view the preferences they have submitted, along with the option to modify and update them. After being assigned to a teacher, they will not be able to make new submissions or modifications to their submission, and they will view the teacher they were assigned to.

Secretary of the Department – Thesis Coordinator of the Department:

For presentation purposes, I will refer to this user as the 'secretary', who will potentially be using the system together with the Thesis Coordinator. The secretary will be able to view the preferences submitted so far by both the teachers and the students. Additionally, they will be able to view the assignment section, which, if the algorithm for the pairing has not run yet, will display the 'run' button, and in case the algorithm has ran and assignments have been made, they will see the assignments made so far, along with the button to re-run the algorithm. Moreover, they will be able to see a summary of the assignments, the students who have remained unassigned so far, and the remaining capacities of the teachers.

5.3 System Architecture

For the system, a 3-layer system architecture was used. This architecture separates the system into: Presentation Layer, Application Layer, and Data Access Layer. This architecture provides a clean approach for systems that require to have a front-end UI interface that interacts with a database to access and display data. This architecture provides easier maintenance and scalability.

Presentation Layer:

This layer consists of the front-end of the system (user interfaces). It handles the interfaces that the users use to interact with the system. The technologies used for this layer are: HTML, CSS, and JavaScript.

Application Layer:

This layer, logically, sits between the presentation and the data access layers. It processes the data, enforces rules, and coordinates the behavior of the application. In this layer, we

define the different API endpoints of the application and how the system should react when those endpoints are triggered by the front-end. Once the endpoint completes its task, it sends a response back to the front-end, which contains the requested data or an error message.

After the front-end receives the response from the application layer, it can display the data received to the user. If the call was successful, it will display the data in a formatted way, or if an error occurred, it will display the error to the user.

The technology used for this layer is the Express framework that works on Node.js, which is a JavaScript runtime environment that allows server-side execution of JavaScript.

Data Access Layer:

The database of the system. In the database, tables were created to store the credentials of the users, the preferences of the students, the preferences of the teachers, the capacities of the teachers, and the assignments of the students. For the management of the database, MySQL was used because it is supported by the university's infrastructure, along with the DBeaver database client, which supports MySQL databases.

5.4 Implementation Tools

This section describes in detail the technologies used to implement the system.

5.4.1 Front-end Web Development

Front-end web development refers to the creation of the interface through which users interact with the system. The first technology used to build the front-end was HyperText Markup Language (HTML) [1], specifically HTML5, which is the fifth and most recent version of the language. HyperText Markup Language is the foundational language used to create and structure content on the web. It is not a programming language, but rather a descriptive language that instructs the web browsers on how to display various elements (images, text, links, etc.) on a page. Every visible component of a webpage is defined using HTML. At its core, HTML uses tags to wrap content, the different elements, and assign

semantic meaning to them. For example, a paragraph is marked with <p>, an image with <img>, and a form with <form>. These tags allow web browsers to interpret the role of each element and render it accordingly.

To style the pages and control their visual appearance, Cascading Style Sheets (CSS) [2] was used. CSS is the language responsible for defining the appearance and layout of webpages. While HTML provides the structural framework, CSS enhances it by specifying design attributes such as colors, fonts, spacing, and positioning. CSS rules can be written within an HTML file using <style> tags, but for projects that require more complex styling, they are typically placed in separate .css files and linked to the HTML. When a browser loads a webpage, it first reads the HTML to determine what content to display, and then applies the CSS rules to style the content accordingly. In CSS, the different elements defined in the HTML file can be referenced using their HTML tags or certain attributes assigned to them in HTML.

The last technology used for the front-end web development is JavaScript [3]. JavaScript is used in front-end web development to add interactivity and dynamic behavior to websites. While HTML defines and structures the content, and CSS styles it, JavaScript makes it dynamic by allowing the page to respond to user actions in real time. For instance, when a user presses a button, fills out a form, or hovers over an element, JavaScript can detect these events and trigger changes such as showing a pop-up window, updating part of the page content without needing to reload the page, or even validating user input. JavaScript runs on the client side, meaning it executes directly in the user's browser rather than on the server, which allows for fast and responsive interactions. It also provides access to the Document Object Model (DOM) [12], which is the browser's internal representation of the page. By manipulating the DOM, JavaScript can dynamically make changes such as changing text, hiding or showing elements, etc., after the page has loaded and without needing to reload it. In this thesis project, JavaScript was extensively used to display user-specific information and enhance the interactivity of the interface, contributing to a more intuitive and engaging user experience. Additionally, JavaScript was chosen for its ability to handle asynchronous operations. Specifically, the Fetch API was utilized, which provides the fetch() function for

making HTTP requests directly from the browser. This function was used to communicate with the endpoints defined in the Application Layer, allowing the system to retrieve or store data in the database without requiring a full page reload.

5.4.2 Back-end Development

Back-end development refers to the creation of the server-side logic that powers the system, processes data, and communicates with the database. It operates behind the scenes to ensure that the front-end interface receives the correct information and behaves as expected in response to user actions.

The back-end of the system was implemented using the Express framework [5] on top of Node.js [4], forming the core of the Application Layer in the 3-layer architecture. Node.js provides the runtime environment that allows JavaScript to be executed on the server side, while Express offers a lightweight and flexible framework for building web applications and defining HTTP endpoints.

In this layer, a set of RESTful API endpoints [6] was created that handle requests from the front-end and interact with the database. Each endpoint is responsible for a specific operation, such as retrieving user preferences, submitting preferences, or validating credentials. These endpoints serve as the bridge between the user interface and the underlying data, ensuring the system responds appropriately to user actions.

The central file in the backend is server.js, which initializes the Express application, configures the middleware, and mounts the various route modules. It begins by importing dependencies, including Express, dotenv for environment variable management, and route handlers for different user roles (authRoutes, studentRoutes, teacherRoutes, etc.). The Express app is configured to parse incoming JSON [7] requests and serve static files (HTML files) from the public directory.

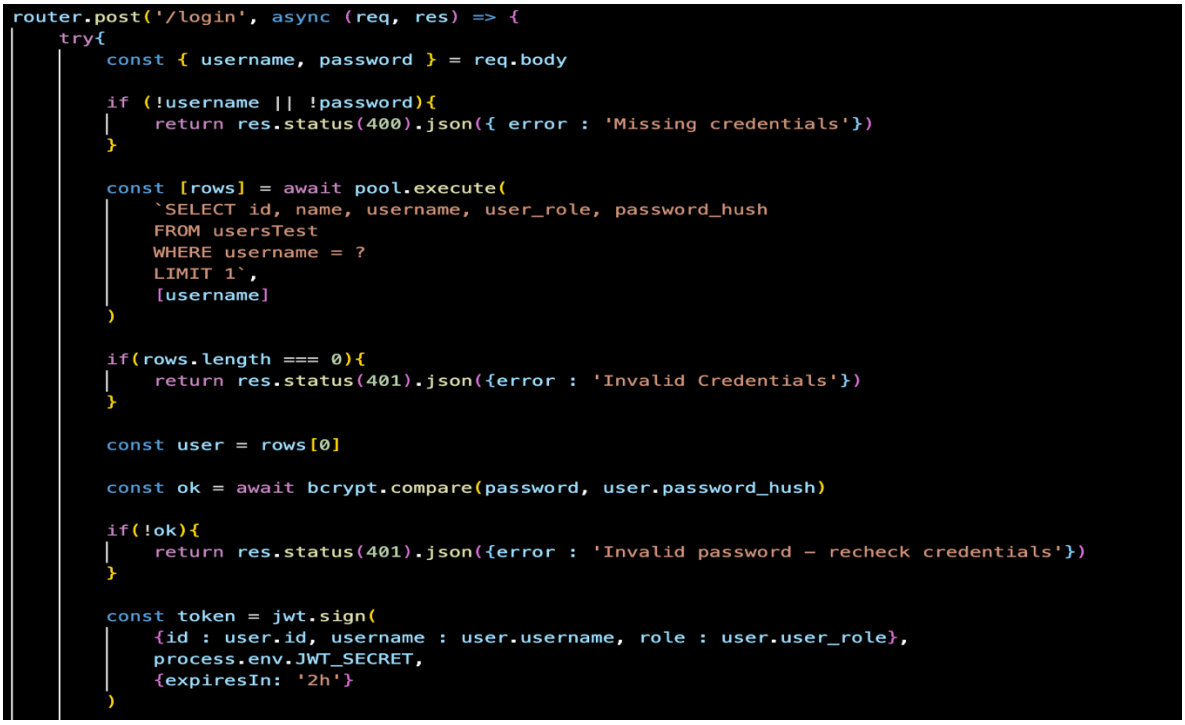
The database connection is managed in a separate file, database.js, which uses the mysql2 library to create a connection pool to the MySQL database. This pool is configured using the database credentials to gain access to the database, which are stored in environment variables and exported for use across the backend. This modular approach ensures the database access is centralized and reusable.

To protect sensitive routes, a middleware function in auth.middleware.js was implemented called requireAuth. This middleware verifies the presence of a valid JSON Web Token (JWT) [8] in the request's Authorization header. If the token is valid, the decoded user information (id, username, role) is attached to the request object, allowing downstream route handlers to personalize or restrict access based on user roles. If the token is missing or invalid, the middleware responds with a 401 [9] Unauthorized error.

In many cases, the logic within the endpoints depends on information provided in the requests, such as credentials, identifiers, or form data. For example, when a user logs in or submits their preferences form, the system extracts relevant data from the request body or headers and uses it to query the database.

Once the necessary operations are performed (querying the database, validating input), the endpoint sends a structured response back to the front-end. This response may contain requested data, confirmation of a successful operation, or an error message if the request could not be fulfilled. The front-end then uses this response to update the user interface accordingly (e.g., display the preferences submitted by the logged-in user).

This modular and event-driven approach allowed for a clean separation of concerns, efficient handling of asynchronous operations, and scalable integration with the database. The use of Express on top of Node.js makes the back-end adaptable to future enhancements.



```
router.post('/login', async (req, res) => {
    try{
        const { username, password } = req.body

        if (!username || !password){
            return res.status(400).json({ error : 'Missing credentials'})
        }

        const [rows] = await pool.execute(
            `SELECT id, name, username, user_role, password_hush
            FROM usersTest
            WHERE username = ?
            LIMIT 1`,
            [username]
        )

        if(rows.length === 0){
            return res.status(401).json({error : 'Invalid Credentials'})
        }

        const user = rows[0]

        const ok = await bcrypt.compare(password, user.password_hush)

        if(!ok){
            return res.status(401).json({error : 'Invalid password - recheck credentials'})
        }

        const token = jwt.sign(
            {id : user.id, username : user.username, role : user.user_role},
            process.env.JWT_SECRET,
            {expiresIn: '2h'}
        )
```

Figure 1: '/auth/login' endpoint that is called when a user clicks the login button in the login page, in order to log in.

```
    return res.json({
        ok : true,
        token,
        user : {
            id : user.id,
            name : user.name,
            username : user.username,
            role : user.user_role
        }
    })
catch (err) {
    console.error("Login Error: ", err)
    return res.status(500).json({error : 'Server Error'})
```

Figure 2: What the '/auth/login' endpoint returns as a response in case of success and in case of error.

5.5 Database Section

This presents the tables created and used in the database to store the relevant data for the system.

Users Table

The department already has a table to store the credentials of the users, so the user table that was created for this thesis had to match the schema of the existing table to allow for seamless integration.

Column Name	#	Data Type	Not Null	Auto Increment	Key
123 id	1	int	[v]	[v]	PRI
A-Z name	2	varchar(200)	[v]	[ ]	
A-Z username	3	varchar(50)	[v]	[ ]	UNI
A-Z user_role	4	enum('Student','Faculty','Admin')	[v]	[ ]	
A-Z password_hush	5	varchar(255)	[v]	[ ]	

Figure 3: Users table schema ([v] means true)

This table stores the database ID of each user, which is a unique identifier for each user in the database, their full name, username, their user role, and their hashed password. To populate this table, I used fake data for teachers (faculty), students, and the admin.

	id	name	username	user_role	password_hush
1	1	Panos Kourkoulis	pkourk02	Student	$2b$12$0YyH6mbRFQj1Nh.F6NPdG.vbzfPj9dLHfwdbJGLV3S
2	3	Yiannis Dimopoulos	yiannisdim	Faculty	$2b$12$dOmPNQCT85FO2YbirbO9sO4iLkeKSCp0aGnPZAC
3	4	Panagiotis Kolios	pkolios	Faculty	$2b$12$UY4t8GXnlKvPwpY5Fw2DjeOMs0nY4f2fK0b9aEEps/
4	5	Yiannos Sazeidis	yiannossaze	Faculty	$2b$12$nNuhcmB72ki9X5fwuZ4tbe8gC8aptgxtA5s9FYhJ5f
5	6	Vasos Vasileiou	vasosvas	Faculty	$2b$12$mcGDOrTrti.TGW3D6GgB9uBVuUoHxpzeKmZ4fNvu
6	7	Demitriana Georgiou	dgeorg04	Student	$2b$12$RvDBtrHCRvHPolraA3JseOUjELZ2x95JQj.xn0.PRmX
7	8	Alexis Andreou	aandre06	Student	$2b$12$TO8oKNupMBKlOEx282AOCuPbgk8R5xAvHqamZ0
8	9	Kyriakos Poyiadjis	kpoyia01	Student	$2b$12$9ogoM.7j6YuY5O/L9gCoaOm26s.2Tejmu3LzqEFL0sl
9	10	Dora Georgiou	addora	Admin	$2b$12$ugot9mx7kiqjO7pBCEhNJu4f7aEqgvmulKyOk4A7Lu
10	11	Georgia Eirini Papadopoulou	gpapad01	Student	$2b$12$6V6xp3MSpUqqC.qT6Z7Sh.dm/o0N.EPSvw0LRN/u9
11	12	Georgia Kourkouli	gkourk01	Student	$2b$12$TXrjPNGkZp4IT5GyZPkrdOY4P9jF/l7mmdg/KxxiMk9

Figure 4: Subset of the users created to populate the users table

Student Preferences Table

Currently, the department does not use a table to store the preferences of the students, as they are sent in forms, and then the matching is completed manually.

Column Name	#	Data Type	Not Null	Auto Increment	Key
123 id	1	int	[v]	[v]	PRI
123 student_id	2	int	[v]	[ ]	UNI
123 preference1	3	int	[v]	[ ]	
123 preference2	4	int	[v]	[ ]	
123 preference3	5	int	[v]	[ ]	
123 preference4	6	int	[v]	[ ]	
created_at	7	timestamp	[ ]	[ ]	

Figure 5: student_preferences table schema ([v] means true)

This table stores the database ID of the submitted preference, which is just an identifier of the entry, the student_id, which is the database ID from the users table of the student who has submitted the preferences, along with the 4 database IDs of the teachers they selected, and the timestamp of when the preferences were submitted.

	id	student_id	preference1	preference2	preference3	preference4	created_at
1	9	1	3	4	6	28	2025-10-02 17:49
2	10	7	27	32	6	40	2025-10-02 17:50
3	11	8	30	36	5	31	2025-10-02 17:52
4	12	9	4	27	5	32	2025-10-02 17:53
5	13	11	6	29	36	27	2025-10-02 17:56
6	14	12	3	31	40	26	2025-10-02 17:58
7	15	13	3	32	38	34	2025-10-02 17:59
8	16	14	35	29	26	41	2025-10-02 18:00
9	17	15	28	5	36	26	2025-10-02 18:01

Figure 6: Subset of data stored in the student_preferences table, based on preferences created for a subset of the student users.

Teacher Preferences Table

Similar to the student preferences table, a table to store the teacher preferences is not currently used by the department. The teacher_preferences table was created to store the

preferences submitted by the teachers, with a similar structure to the student_preferences table, as the logic is the same.

Column Name	#	Data Type	Not Null	Auto Increment	Key
123 id	1	int	[v]	[v]	PRI
123 teacher_id	2	int	[v]	[ ]	UNI
123 preference1	3	int	[v]	[ ]	
123 preference2	4	int	[v]	[ ]	
123 preference3	5	int	[v]	[ ]	
123 preference4	6	int	[v]	[ ]	
created_at	7	timestamp	[ ]	[ ]	

Figure 7: teacher_preferences table schema ([v] means true)

This table stores the database ID of the submitted preference, which is just an identifier for the entry, the teacher_id, which is the database ID from the users table of the teacher who submitted the preferences, along with the 4 database IDs of the 4 students they selected, and the timestamp of when the preferences were submitted.

	id	teacher_id	preference1	preference2	preference3	preference4	created_at
1	13	3	1	12	13	24	2025-10-03 11:49
2	14	4	23	22	9	1	2025-10-03 11:53
3	15	5	15	18	25	8	2025-10-03 11:55
4	16	6	11	43	19	7	2025-10-03 11:57
5	17	26	20	48	47	14	2025-10-03 11:59
6	18	27	7	20	21	9	2025-10-03 12:05
7	19	28	15	51	53	19	2025-10-03 12:07
8	20	29	52	11	14	46	2025-10-03 12:09
9	21	30	8	49	22	16	2025-10-03 12:11

Figure 8: Subset of data stored in the teacher_preferences table, based on preferences created for a subset of the teacher/faculty users.

Teacher Capacity Table

No table exists to store the capacity of the teachers. A table was created to store the initial capacity of the teachers, which was four for all teachers, along with the remaining capacity of each teacher, which depended on the number of students they were assigned (remaining_capacity = total_capacity - # of assigned students).

Column Name	#	Data Type	Not Null	Auto Increment	Key
123 teacher_id	1	int	[v]	[ ]	PRI
123 capacity_total	2	int	[v]	[ ]	
123 capacity_remaining	3	int	[v]	[ ]	

Figure 9: teacher_capacity table schema ([v] means true)

This table stores the teacher_id, which is the database ID of the teacher, along with their total capacity (capacity_total), and their remaining capacity (capacity_remaining).

Assignments Table

No table exists to store the assigned teachers to students. The pairings table was created to store the assigned teacher for each student after the matching algorithm produces its results.

Column Name	#	Data Type	Not Null	Auto Increment	Key
123 id	1	int	[v]	[v]	PRI
123 student_id	2	int	[v]	[ ]	UNI
123 teacher_id	3	int	[ ]	[ ]	
created_at	4	timestamp	[ ]	[ ]	

Figure 10: pairings table schema ([v] means true)

This table stores the database ID of the assignment, which is just an identifier for the entry, the database ID of the student (student_id), along with the database ID of the teacher that the student was assigned to (teacher_id), and the timestamp of when the assignment was made. If a student is assigned to a teacher, then the teacher_id column for the row that concerns the student will be populated with the database ID of the teacher they were assigned to. But if the student remained unassigned, the value of the teacher_id will be NULL.

	id	student_id	teacher_id	created_at
1	33	1	3	2025-10-03 19:41:10
2	34	7	27	2025-10-03 19:41:10
3	35	8	30	2025-10-03 19:41:10
4	36	9	4	2025-10-03 19:41:10
5	37	11	6	2025-10-03 19:41:10
6	38	12	3	2025-10-03 19:41:10
7	39	13	3	2025-10-03 19:41:10
8	40	14	35	2025-10-03 19:41:10
9	41	15	28	2025-10-03 19:41:10
10	42	16	[NULL]	2025-10-03 19:41:10
11	43	17	33	2025-10-03 19:41:10

Figure 11: Subset of data stored in the pairings table after running the matching algorithm.

Chapter 6
User Interfaces

6.1 Introduction
This chapter presents the user interfaces designed for the different user groups within the system. It provides a detailed overview of all the pages implemented for each role, illustrating how these interfaces support the specific functions required to complete the assignment process efficiently and intuitively.

6.2 Student Interface
This section will present all the pages implemented for the student interface. These pages cover all the functions needed for the students to submit their preferences, along with summaries to help them visualize and track their submitted preferences and their assignment.

6.2.1 Teacher Selection Page
On this page, the student can see the form where they will select the four teachers they would prefer to be supervised by. The section is made using dropdowns that list all the teachers who are available, meaning they have remaining capacity and can take on more students. This page is visible if the student has not submitted any preferences yet.

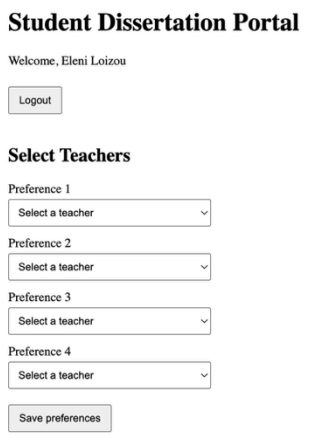


Figure 1: The teacher selection page in the student interface (before they have submitted any preferences).

Figure 2: The dropdown to select a teacher from the list of teachers with remaining capacity.

After the student has selected teachers in all four fields, they submit their preferences by pressing the ‘Save Preferences’ button. After successful submission, the student will see a pop-up message from the browser noting the successful submission.

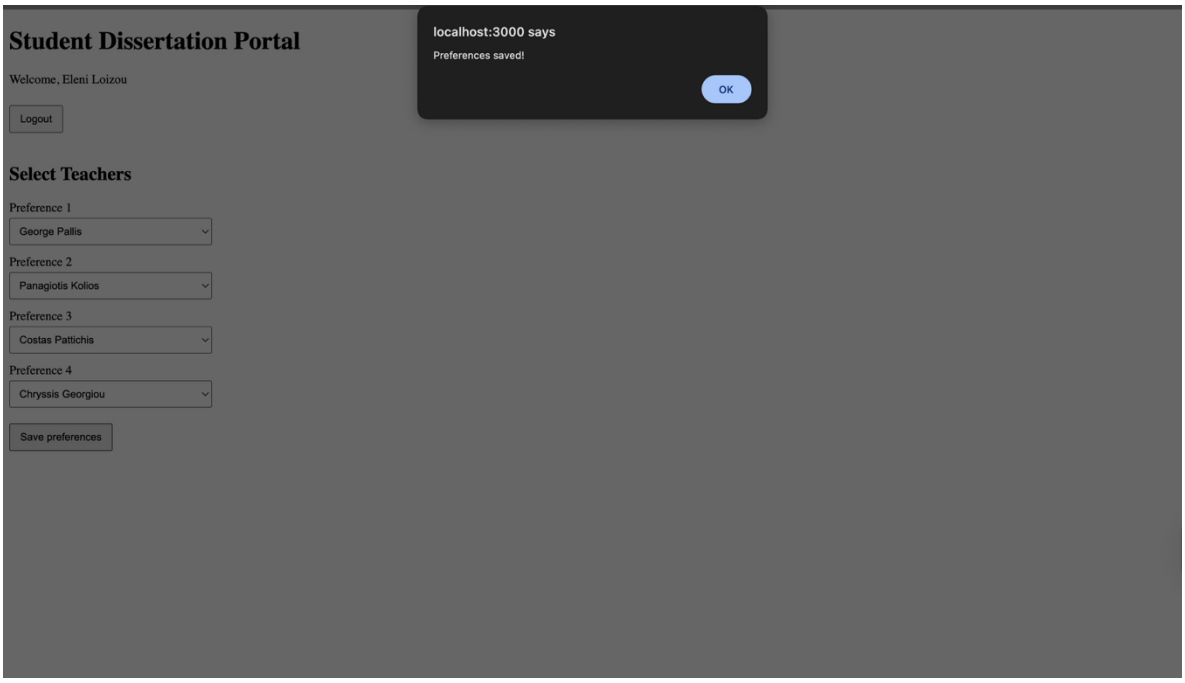


Figure 3: Pop-up message to confirm successful submission after selecting four teachers and pressing the ‘Save Preferences’ button.

6.2.2 Submitted Preferences Summary Page

Once the student submits their preferences, the form is hidden, and they can view a summary of their submitted preferences, which includes the name of each choice along with its rank.

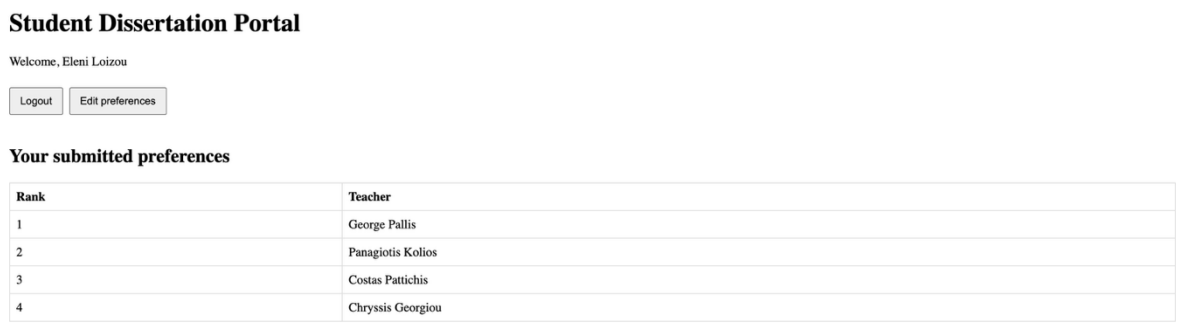


Figure 4: The summary of the submitted preferences of the student.

In this page, the user has the option to edit their preferences by pressing the 'Edit Preferences' button found on the page. Once pressed, the form for submitting preferences reappears, but in each selection field, the name of the existing preference is shown instead of the 'Select a Teacher' text placeholder.

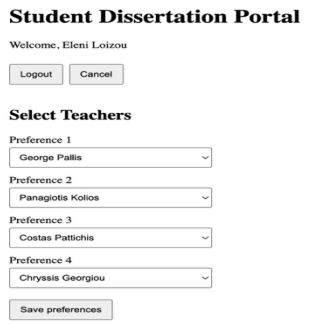


Figure 5: The form with the pre-filled choices, which the user sees when pressing the 'Edit Preferences' button to edit their preferences.

If the user presses cancel, then the summary table of the submitted preferences reappears unchanged, but if the user makes changes and presses the ‘Save Preferences’ button, the confirmation pop-up reappears before showing again the table of the submitted preferences, which reflects the changes made in the last submission.

6.2.3 Student Assignment Page

After the matching algorithm has been run by the administrator, if the student was assigned to a teacher, they will see the teacher they were assigned to, but if the student remained unassigned, they will see the pages mentioned above (page to select teachers, page to view submitted preferences)

Figure 6: The page that the student sees after being assigned to a teacher.

6.3 Teacher Interface

This section will present all the pages implemented for the teacher interface. These pages cover all the functions needed for the teachers to view the students who have selected them, submit their preferences, and view summaries to help them visualize and track their submitted preferences and their assignments.

6.3.1 Student Preferences Page

On this page, the teacher can view the names and usernames of all the students who have selected them in their submitted preferences, along with the rank that each of those students selected the teacher in.

Figure 7: The page with the summary of the students who have selected the teacher.

The students on this page are sorted based on the ranking they gave the teacher. The students who ranked the teacher higher in their preferences appear first.

6.3.2 Student Selection Page

On this page, the teacher can see the form where they will select the four students they would prefer to supervise. The section is made using dropdowns that list all the students who are available, meaning they have not been assigned to a teacher yet, and who have submitted preferences.

Figure 8: Student selection page in the teacher interface (before they have submitted any references).

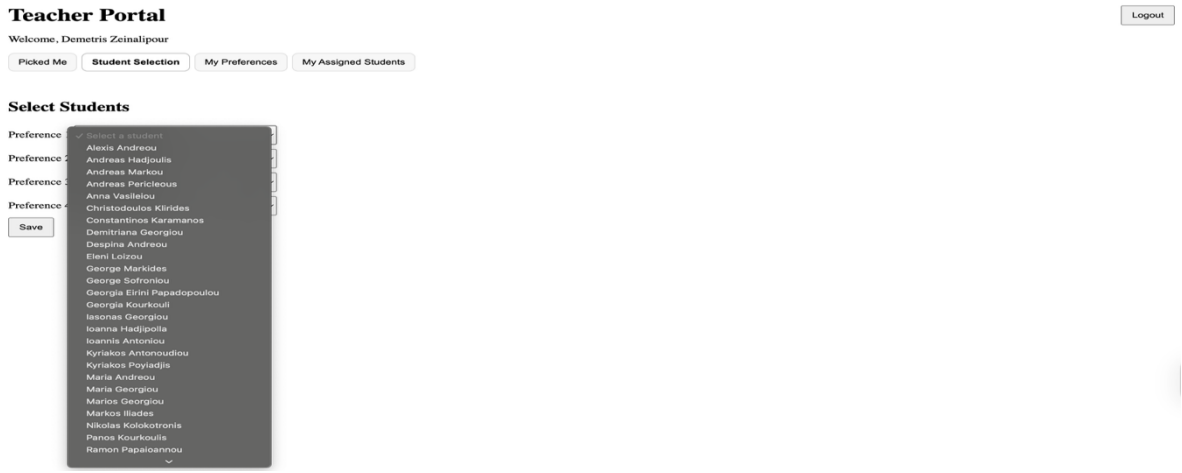


Figure 9: The dropdown to select a student from the students who have not been assigned to a teacher yet and who have submitted preferences.

After the teacher has selected students in all four fields, they submit their preferences by pressing the ‘Save’ button. After successful submission, the teacher will see a pop-up message from the browser noting the successful submission.

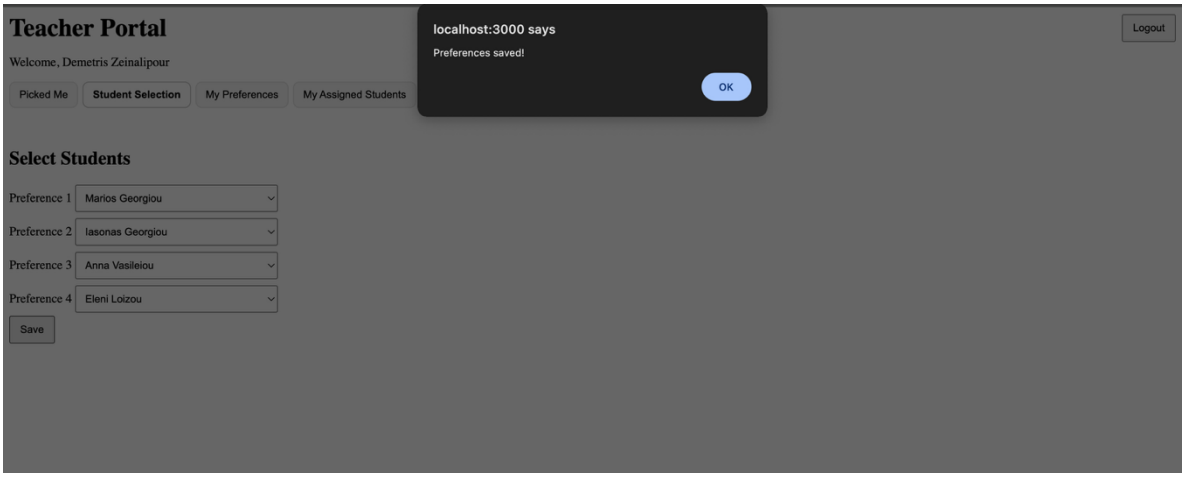


Figure 10: Pop-up message to confirm successful submission, after choosing 4 students and pressing the ‘Save’ button.

After the teacher has submitted preferences, the form will be visible, but instead of the ‘Select a Student’ placeholder text in each field, they will view the student they selected in their last submission in that field. They can then make changes and press the ‘Save’ button again to resubmit and update their preferences. On successful resubmission, the confirmation pop-up message will show again.

Figure 11: The form for the teacher to resubmit preferences, after they have already submitted preferences.

6.3.3 Submitted Preferences Summary Page

On this page, the teacher can see the students who they have selected in their latest preferences submission.

If the teacher has not submitted any preferences yet, nothing shows on this page.

Figure 12: The submitted preferences summary page, before the teacher submits preferences for the first time.

Once the teacher submits their preferences, they can see the students they selected.

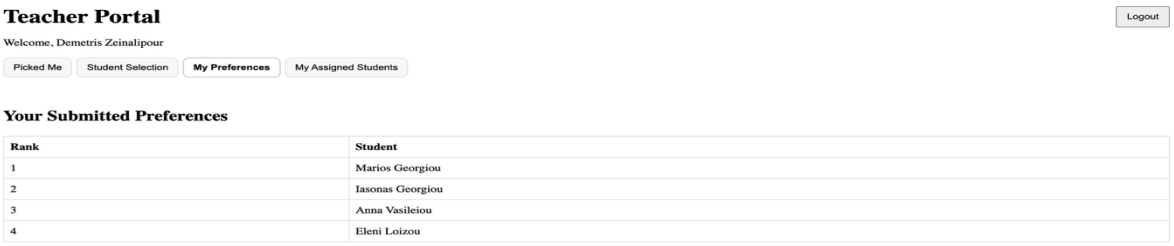


Figure 13: The submitted preferences summary page, after the teacher has submitted preferences.

6.3.4 Assigned Students Summary Page

On this page, the teacher can view all the students they were assigned.

If the algorithm to match the students with the teachers has not run yet, or if the teacher has not been assigned any students, this page will display no assigned students, along with the appropriate message.

Figure 14: The Assigned Students Summary Page if the teacher has not been assigned any students.

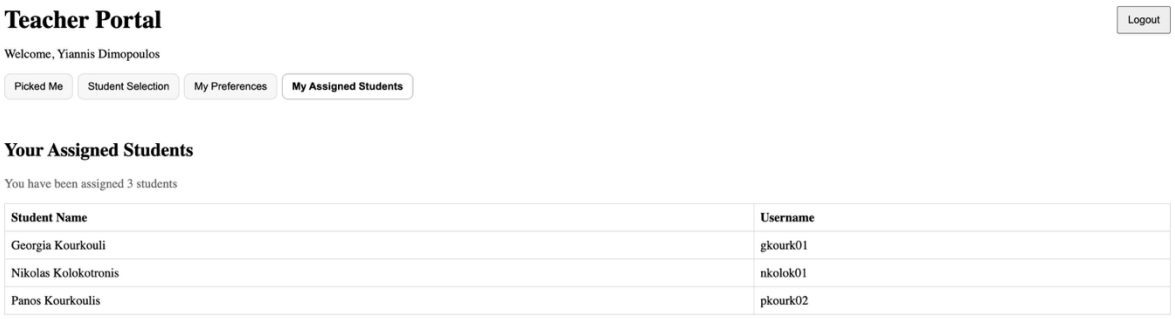


Figure 15: The Assigned Students Summary Page after the teacher has been assigned students.

6.4 Administrator Interface

This section presents the administrator interface of the system. It describes all the pages implemented for administrative use, covering all the functions required to manage the matching process. Specifically, the interface enables the administrator to run the matching algorithm, review the preferences submitted by both the teachers and the students, and view comprehensive summaries of the results. These summaries include the final assignments, the students who remain unassigned, and the remaining capacity of each teacher. Moreover, it allows the user to edit the remaining capacities of the teachers and also reset the system after the process has been completed.

6.4.1 Submitted Preferences Overview Page

On this page, the administrator can view an overview of all the preferences submitted by the students and the teachers. The submissions are separated into two tables, one for the teacher preferences and one for the student preferences, to make reviewing easier.

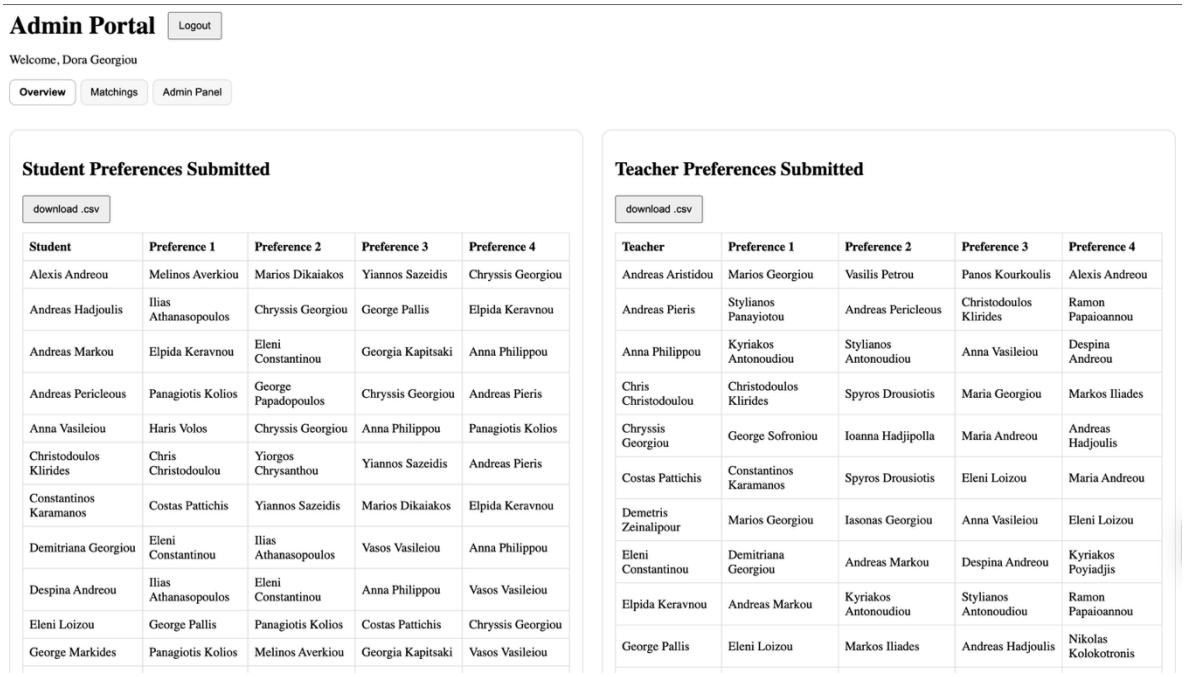


Figure 16: The tables (parts of them) in the submitted preferences overview page, where the administrator can review all the submitted preferences and download the tables as .csv files.

The administrator has the ability to download the data as a CSV file by pressing the ‘download .csv’ button to download each table individually.

6.4.2 Matching Algorithm Page

On this page, before the matching algorithm has been run, the administrator will see a message saying that the algorithm has not run yet, along with the button to run the algorithm to create the assignments.

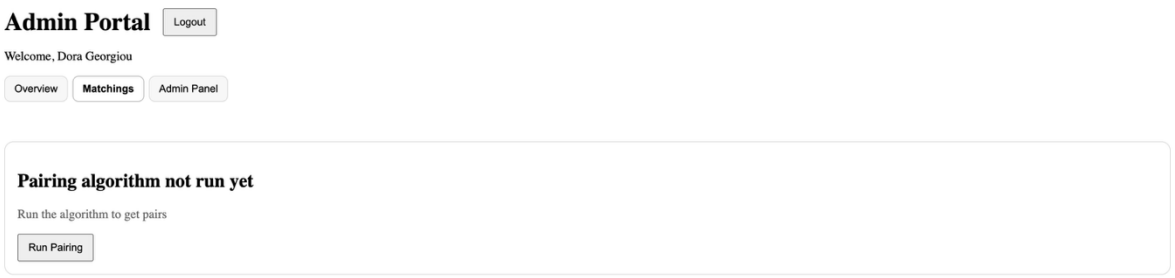


Figure 17: Matching algorithm page before the algorithm has been run.

After pressing the ‘Run Pairing’ button, the system will run the algorithm to create the assignments. If the algorithm runs successfully, a success message will appear in green color, indicating that the matching has completed, along with the number of rows that were inserted in the pairings table in the database. In case of an error, a message in red will appear stating the error.

6.4.3 Assignments Summary Page

On this page, the administrator will be able to view all the assignments made so far in a table. Each row on the table depicts one assignment. The rows where the name of the teacher is missing indicate that the student whose name is on that row has remained unassigned.

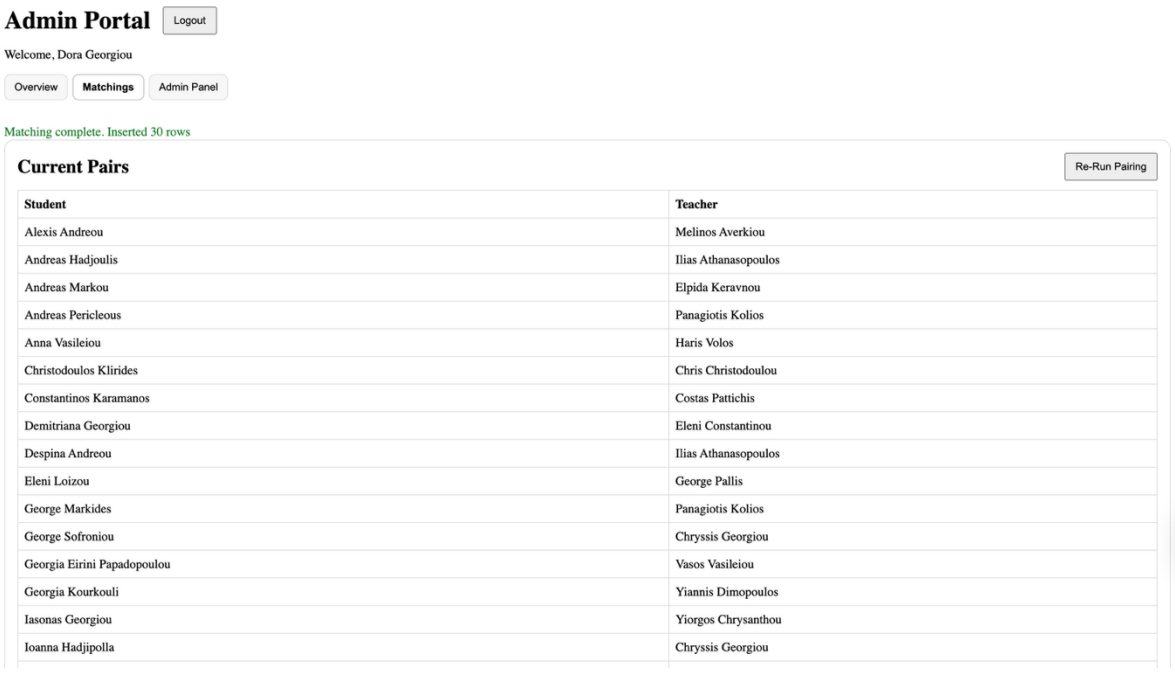


Figure 18: Assignment Summary Page after the algorithm has ran and produced assignments. Also, the text to indicate a successful run of the algorithm is present because the screenshot was taken immediately after running the algorithm; if the page refreshes, the text will disappear.

Also, the administrator has the option to re-run the pairing algorithm, which can be used for the second assignment round to assign the students who have remained unassigned.

6.4.4 Administrator Control Page

On this page, the administrator will be able to view, in separate tables, the students who have been assigned to teachers, along with the teacher they have been assigned to, the students who have remained unassigned, and the remaining capacity of all the teachers. Moreover, the administrator will be able to (a) clear all the assignments made and reset the capacities of the teachers to four (4) by pressing the “Clear Pairings” button and (b) change the capacity of each teacher individually by pressing the “Edit Capacities” button. Upon clicking the “Clear Pairings” button or the “Edit Capacities” button, a confirmation message will appear. If the user presses cancel in the confirmation pop-up, then nothing will happen, but if the user presses “confirm,” then the action triggered by the button will be completed.

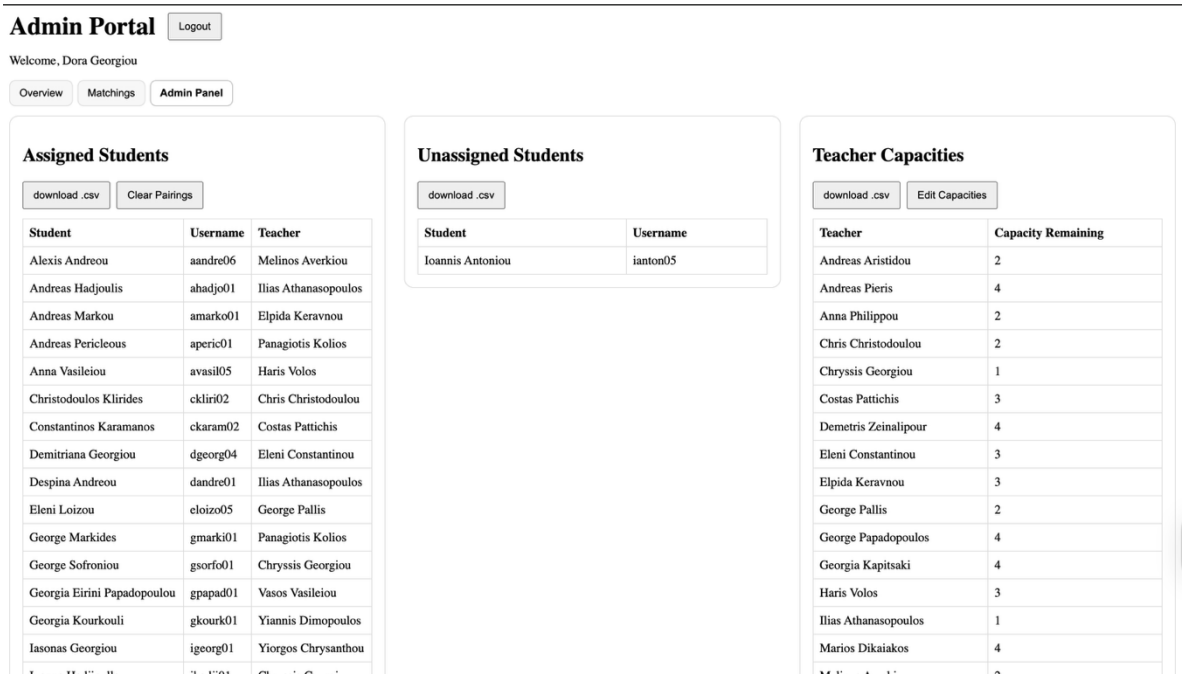


Figure 19: Administrator Control Page, where the admin can see the assigned students, the unassigned students, and the remaining capacities of the teachers, and download the tables as .csv files.

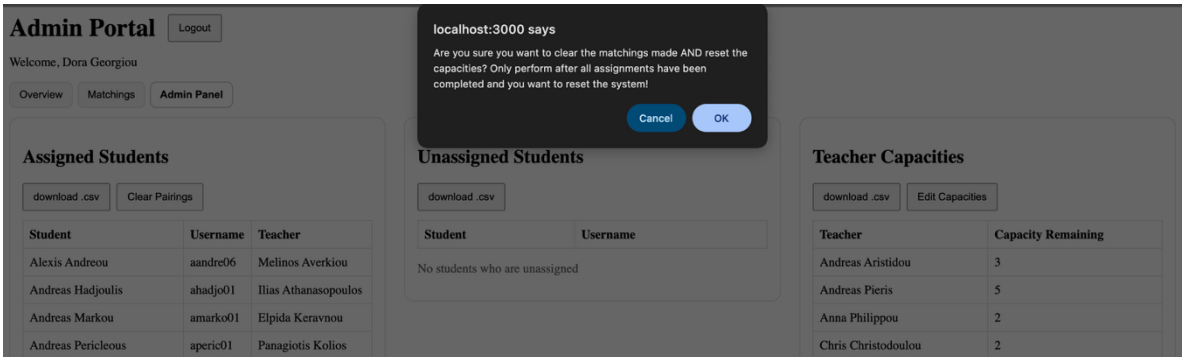


Figure 20: The confirmation pop-up message when the user presses the “Clear Pairings” button.

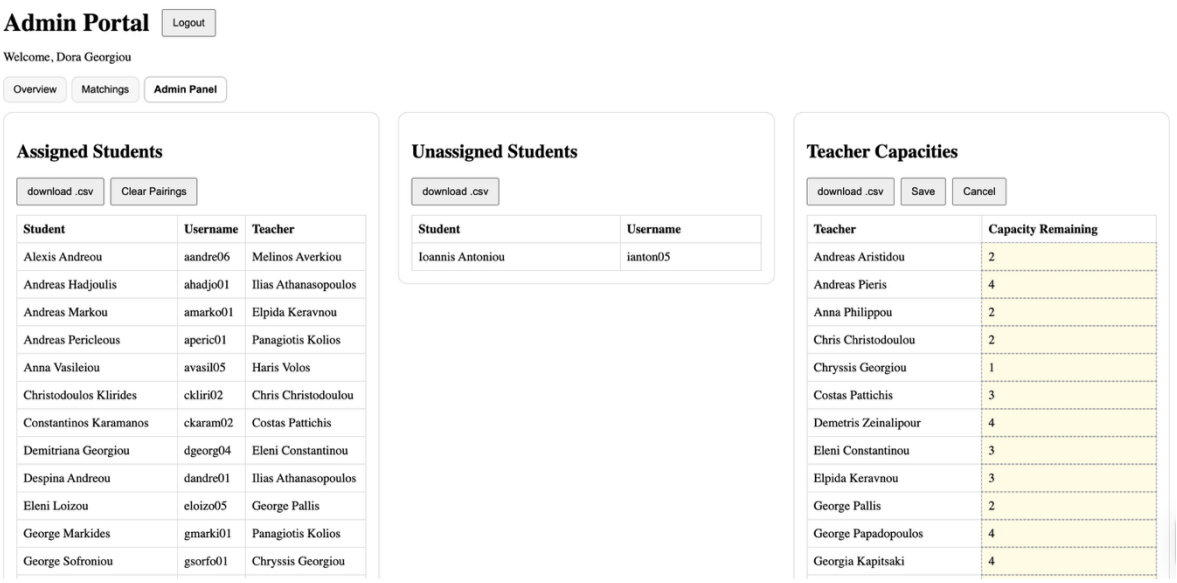


Figure 21: When the user presses the “Edit Capacities” button. The fields of the capacities change background color to signify they are editable. If the user presses the “Cancel” button, then nothing happens and the view is restored. If the user presses “Save”, then a confirmation message appears.

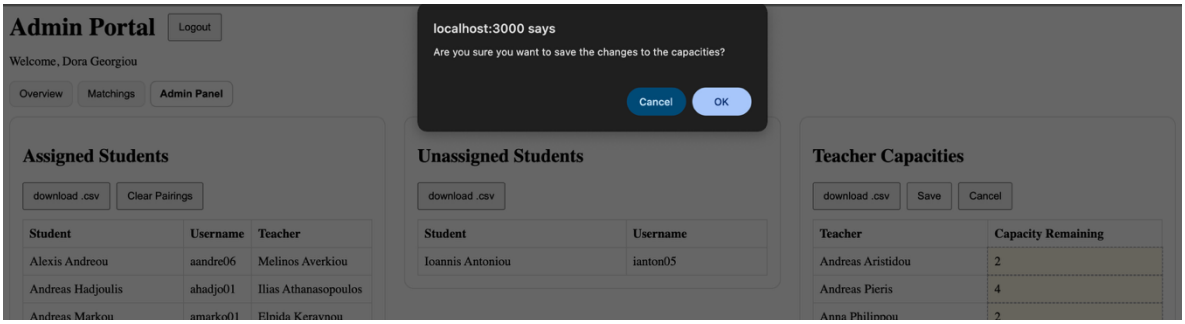


Figure 22: Confirmation pop-up message after the user presses the “Save” button to make changes to the capacities of the teachers.

The user can download the tables as separate .csv files by pressing the ‘download .csv’ button on each table.

Chapter 7
Conclusions

7.1 System Overview

The current procedure followed to assign supervisor teachers to students for their Individual Thesis Project is done manually by the secretary of the department. Currently, the secretary of the department, after receiving all the preference lists of the students and the teachers, has to manually match each student to a teacher in a way that satisfies the preferences of both parties, and simultaneously maximizes the satisfaction of the student rankings. This is a very time-consuming and error-prone process that can lead to mismatches or unfair assignments. Moreover, the lists are transferred via email, which makes it harder for the teachers and the students to fill out their lists, and also makes it more complex for the secretary to track all the replies in their inbox, considering that they also get other emails. Finally, after the secretary completes the manual assignment process, they have to then manually calculate the remaining capacity of each teacher as well as find which students have remained unassigned, which again poses risks for errors and is also time-consuming.

With the implementation of this system, the students and the teachers will be able to submit their preferences from the platform without having to fill out forms manually and send them via email. Also, they will have the option to make changes to their preferences if they so wish, which they are currently not able to do since they can only send the form once. Moreover, the teachers will be able to view, in real time, the students who have selected them in order to make more informed decisions about their preferences. Then, the secretary will be able to see all the submitted preferences and run the matching algorithm to make the assignments by just clicking a button. After the algorithm runs successfully, the secretary will be able to view all the students who have been assigned and to which teacher, the students who have remained unassigned, and the remaining capacity of each teacher without needing to perform any further actions or calculations. It is also worth mentioning that the system is able to support not just the first assignment round, but also any additional rounds that may follow.

In conclusion, this system significantly simplifies, streamlines, and automates the assignment process. By leveraging the constraint satisfaction algorithm, it eliminates the risk of human error and ensures fair and optimal matches. The entire procedure is completed within the system, removing the need for email communication and manual tasks.

The system has undergone extensive testing and has consistently produced correct results. The constraint satisfaction algorithm was validated using manually solved cases, confirming its accuracy and correctness. In a benchmark test where the dataset involved 80 students and 30 teachers, the algorithm completed the matching process in just 446 milliseconds, demonstrating both efficiency and scalability.

7.2 Instructions

In order for the system to work, Node.js should first be installed. Then the Express framework should be installed. This installation for the framework can be performed using npm, the built-in package manager of Node.js. Additionally, Minizinc has to be installed. Also, Python has to be installed along with the minizinc-python library.

For the database, assuming that the table which stores the user credentials exists, tables should also be created to store the following data: student preferences, teacher preferences, teacher capacities, assignments.

7.3 Restrictions

The system redirects the user, upon successful login, to the appropriate page by taking into account the role of the user found in the table which stores the user credentials. In order for a user to access the administrator portal, their role should be set to ‘Admin’ in the aforementioned table. The Thesis Coordinator, being a selected academic member of the department, will not have access to this portal since their role is set to ‘Faculty’. A possible solution to bypass this would be to create an additional user for the selected teacher, whose role will be set to ‘Admin’, and keep the existing ‘Faculty’ account of the teacher as is. Also, in order for the system to run correctly, all teachers who have remaining capacity (>0) should submit preferences. Additionally, when the administrator changes the remaining capacity of a teacher (thereby logically modifying the default total capacity of four), the change only

applies to the current matching round. In the following round, the capacities are automatically recalculated based on the existing assignments, again assuming each teacher's total capacity is four. Therefore, if the administrator intends to maintain a changed capacity for a teacher in future rounds, they must manually adjust the teacher's capacity again before executing the next matching algorithm.

7.4 Future System Enhancements

In the future, the user interfaces could be enhanced with improved styling to increase usability, making them more intuitive, visually appealing, and engaging for all user groups.

Bibliography

[1] MDN Web Docs. “HTML: HyperText Markup Language,” Mozilla Developer Network [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Accessed: 25-Oct-2025]

[2] MDN Web Docs, “CSS: Cascading Style Sheets,” Mozilla Developer Network [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Accessed: 25-Oct-2025]

[3] MDN Web Docs, “JavaScript,” Mozilla Developer Network [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed: 25-Oct-2025]

[4] Node.js, “Node.js v25.0.0 documentation,” Node.js [Online]. Available: https://nodejs.org/api/all.html. [Accessed: 25-Oct-2025]

[5] Express.js, “Express – Node.js web application framework,” Express.js [Online]. Available: https://expressjs.com. [Accessed: 25-Oct-2025]

[6] Lokesh Gupta, “REST API Tutorial,” RESTfulAPI.net [Online]. Available: https://restfulapi.net. [Accessed: 25-Oct-2025]

[7] MDN Web Docs, “JSON,” Mozilla Developer Network [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON. Accessed [25-Oct-2025]

[8] Auth0 Contributors, “Introduction to JSON Web Tokens,” JWT.io [Online]. Available: https://www.jwt.io/introduction#what-is-json-web-token. [Accessed: 25-Oct-2025]

[9] MDN Web Docs, “HTTP response status codes,” Mozilla Developer Network [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status [Accessed: 25-Oct-2025].

[10] MiniZinc, “The MiniZinc Handbook 2.9.4,” MiniZinc [Online]. Available: https://docs.minizinc.dev/en/stable/index.html. [Accessed: 25-Oct-2025].

[11] MiniZinc, “MiniZinc Python 0.10.0 documentation,” MiniZinc Python [Online]. Available: https://python.minizinc.dev/en/latest/getting_started.html. [Accessed: 25-Oct-2025].

[12] MDN Web Docs, “Document Object Model (DOM),” Mozilla Developer Network [Online].
Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model [Accessed: 25-Oct-2025].

Appendix

Appendix A

Constraint Satisfaction Algorithm written in MiniZinc that assigns supervisor teachers to students for their Individual Thesis.

```
int: numStudents; %The number of students
int: numTeachers; %The number of teachers

set of int: Students = 1..numStudents;
set of int: Teachers = 1..numTeachers;

%Array that holds the preferences of each student
array[Students, 1..4] of int: studentsPreferences;

%Array that holds the preferences of each teacher
array[Teachers, 1..4] of int: teacherPreferences;

%Array that holds the capacity of each teacher
array[Teachers] of int: teacherCapacity;

%Array that stores wether a students has selected a
teacher in their list
%studentPrefers[s,t] = true if student s has selected
teacher t in their preferences, false if otherwise
array[Students, Teachers] of bool: studentPrefers =
  array2d(Students, Teachers,
  [ exists(i in 1..4)(studentsPreferences[s,i] = t) | s
in Students, t in Teachers ]);

%Array that stores wether a teacher has selected a
student in their list
%teacherPrefers[t,s] = true is teacher t has selected
student s in their preferences, false if otherwise
array[Teachers, Students] of bool: teacherPrefers =
```

```
  array2d(Teachers, Students,
  [ exists(j in 1..4)(teacherPreferences[t,j] = s) | t
in Teachers, s in Students ]);

%Array that stores wether a pair is allowed
%allowed[s,t[ = true is student s selected teacher t and
teacher t selected student s. otherwise false
array[Students, Teachers] of bool: allowed =
  array2d(Students, Teachers,
  [ studentPrefers[s,t] /\ teacherPrefers[t,s] | s in
Students, t in Teachers ]);

%Array that stores in what rank studentt has chosen a
teacher
%rankS[s,t] = x means that student s ranked teacher t as
x (1-4) in their preferences
%if the student s didn't select the teacher t in their 4
preferences then rankS[s,t] = 5
array[Students, Teachers] of int: rankS =
  array2d(Students, Teachers,
    [ let {
        set of int: hits = { i | i in 1..4 where
studentsPreferences[s,i] = t }
      } in if card(hits) > 0 then min(hits) else 5 endif
    | s in Students, t in Teachers ]);


%Decision Variable 1: [Array] stores the assignments -
x[s,t] = 1 if student s is assigned to teacher t. x[s,t]
= 0 if otherwise
array[Students, Teachers] of var 0..1: x;
%Decision Variable 2: [Array] stores the supervisor of
each student
%supervisor[s] = t means that student s is
supervised/assigned to teacher t
%supervisor[s] = 0 if the student s is unassigned
array[Students] of var 0..numTeachers: supervisor;

%Constraint 1: coherence constraint supervisor[s] = 0 or
supervisor[s] = t since x[s,t] = 0 || 1
constraint forall(s in Students)(
  supervisor[s] = sum(t in Teachers)(t * x[s,t])
```

```
);

%Constraint 2:
%if allowed[s,t] = false then bool2int(false) = 0 =>
x[s,t]<=0 => x[s,t] = 0 otherwise x[s,t] can be 1
%so only allowed assignment of student s to teacher t if
allowed
constraint forall(s in Students, t in Teachers)(
  x[s,t] <= bool2int(allowed[s,t])
);

%Constraint 3: a student can only be assigned to at most
one teacher
%they are either assigned to one teacher or no teacher
constraint forall(s in Students)(
  sum(t in Teachers)(x[s,t]) <= 1
);

%Constraint 4: the number of assigned students to a
teacher cannot exceed the capacity of the teacher
constraint forall(t in Teachers)(
  sum(s in Students)(x[s,t]) <= teacherCapacity[t]
);

%Constraint 5: assigned each student to their most
preferred teacher if allowed and if the teacher has
capacity
constraint forall(s in Students, t in Teachers where
rankS[s,t] <= 4)(
  (x[s,t] = 1) -> forall(b in Teachers where rankS[s,b]
< rankS[s,t])(
        (not allowed[s,b]) \/ (sum(ss in
Students)(x[ss,b]) >= teacherCapacity[b])
    )
  );

%Decision Variable 3: tthe number of unassigned students
- used to minimize the unassigned students => maximize
the assignments
%otherwise many will be left unassigned and the solution
would still fill the constraints
```

```
var int: unassigned_count = sum(s in Students)(1 - sum(t
in Teachers)(x[s,t]));

solve minimize unassigned_count;
```

Appendix B

Python code that runs the MiniZinc model and returns the *supervisor* array.

```
from dataclasses import dataclass, InitVar
from typing import List, Optional, Any
from minizinc import Model, Instance, Solver
import json, sys
from pathlib import Path

@dataclass
class MatchSolution:
    supervisor: List[int]
    x: Optional[Any] = None
    unassigned_count: Optional[int] = None
    objective: Optional[int] = None
    __output_item: InitVar[str] = None

def main():
    raw = sys.stdin.read()
    if not raw.strip():
        print(json.dumps({"ok": False, "error": "No input"}))
        return

    payload = json.loads(raw)

    here = Path(__file__).resolve().parent
    model_path = here / "minizincModel.mzn"

    model = Model(str(model_path))
    model.output_type = MatchSolution
    solver = Solver.lookup("gecode")
    inst = Instance(solver, model)

    inst["numStudents"] = payload["numStudents"]
    inst["numTeachers"] = payload["numTeachers"]
    inst["studentsPreferences"] = payload["student_prefs"]
    inst["teacherPreferences"] = payload["teacher_prefs"]
    inst["teacherCapacity"] = payload["teacher_capacity"]

    res = inst.solve()
    sol = res.solution

    if sol is None:
```

```
        print(json.dumps({"ok": False, "error": f"No solution: {res.status}"}))
        return

    assignments = list(sol.supervisor)
    print(json.dumps({"ok": True, "assignments": assignments}))

if __name__ == "__main__":
    main()
```

Appendix C

Express endpoint in the admin routes module (admin.routes.js) that creates the mappings and calls the Python script for Appendix B in order to create the assignments. When the Python program returns, it takes the array returned (supervisor), unmaps the IDs, and updates the relevant tables.

```
//Function to call python as a child process
function runPython(pythonFile, payloadPython){
    return new Promise((resolve, reject) => {
        const process = spawn(PYTHON_BIN, [pythonFile], { stdio: ['pipe', 'pipe',
'pipe'], env: CHILD_ENV })
        let stdout = ''
        let stderr = ''

        process.stdout.on('data', (c) => (stdout += c.toString()))
        process.stderr.on('data', (c) => {
            const s = c.toString()
            console.error('[py]', s.trim())
            stderr += s
        })
        process.on('error', (err) => reject(err))
        process.on('close', (code) => {
            if( code !== 0 ){
                return reject(new Error(`Python exited with code ${code}. stderr:
${stderr || '(empty)'}`))
            }
            try{
                const json = JSON.parse(stdout)
                if (json && json.ok === false) {
                    json._stderr = stderr
                }
                resolve(json)
            }catch (err){
                reject(new Error(`Failed to parse Python
JSON.\nstdout:\n${stdout}\n\nstderr:\n${stderr}`))
            }
        })
        //Send the payload to the child process (the python script)
        process.stdin.write(JSON.stringify(payloadPython))
        process.stdin.end()
    })
```

```
}

/** POST /admin/run-matching
 * fetches the data: ids of the students and the teachers and the preferences
list of both
 * creates mappings to have continuous indexes for the students and the teachers.
Maps the user ids
 * calls the python code as a child process which in turn runs the minizinc model
 */
router.post('/run-matching', requireAuth, async (req, res) => {
    try {
        if (req.user.role !== 'Admin') {
            return res.status(403).json({ error: 'Forbidden' });
        }

        // Fetch the students who have submitted preferences and are not assigned
to a teacher
        const [studentIdsRows] = await pool.execute(
            `
            SELECT sp.student_id
            FROM student_preferences sp
            LEFT JOIN pairings p ON p.student_id = sp.student_id
            WHERE p.student_id IS NULL OR p.teacher_id IS NULL
            ORDER BY sp.student_id
            `
        )
        const studentIds = studentIdsRows.map(r => r.student_id)

        //Fetch the teachers
        const [teacherIdsRows] = await pool.execute(
            `
            SELECT u.id
            FROM usersTest u
            JOIN teacher_capacity tc ON tc.teacher_id = u.id
            WHERE u.user_role = 'Faculty' AND tc.capacity_remaining > 0
            ORDER BY u.id
            `
        )

        const teacherIds = teacherIdsRows.map(r => r.id)

        if (studentIds.length === 0) {
            return res.status(400).json({ error: 'Could not fecth students from
student_preferences who are unassigned' })
        }
        if (teacherIds.length === 0) {
```

```
            return res.status(400).json({ error: 'Could not fetch teachers with
left capacity' })
        }

        //Fetch the preferences of the teachers and the students
        const [studentPrefsRows] = await pool.execute(
            `
            SELECT student_id, preference1, preference2, preference3, preference4
            FROM student_preferences
            WHERE student_id IN (${studentIds.map(()=>'?').join(',')})
            ORDER BY student_id
            `,
            studentIds
        )

        const [teacherPrefsRows] = await pool.execute(
            `
            SELECT teacher_id, preference1, preference2, preference3, preference4
            FROM teacher_preferences
            WHERE teacher_id IN (${teacherIds.map(()=>'?').join(',')})
            ORDER BY teacher_id
            `,
            teacherIds
        )

        //Fetch the capacities of the teachers from the teacher_capacity table
        const [capacityRows] = await pool.execute(
            `
            SELECT teacher_id, capacity_remaining
            FROM teacher_capacity
            WHERE teacher_id IN (${teacherIds.map(() => '?').join(',')})
            ORDER BY teacher_id
            `,
            teacherIds
        )

        //Build the map: teacher_id -> capacity_remaining
        const capacityMap = new Map(capacityRows.map(r => [r.teacher_id,
r.capacity_remaining]))

        //Capacity array that will give to minizinc (Ordered based on teacherIds)
        const teacher_capacity = teacherIds.map(id => {
            if (!capacityMap.has(id)) {
                throw new Error(`No capacity entry found for teacher_id ${id}`)
            }
            return capacityMap.get(id)
        })
```

```
        //Create the mappings for the teachers and the students (starting from 1)
- key = DB id, value = index 1..N
        const studentMap = new Map(studentIds.map((id, i) => [id, i + 1]))
        const teacherMap = new Map(teacherIds.map((id, i) => [id, i + 1]))

        //Check that all users have submitted their preferences so no error will
occur with minizinc model
        const teacherSet = new Set(teacherIds)
        const badTeacherRefs = []
        for (const row of studentPrefsRows) {
            for (const tId of [row.preference1, row.preference2, row.preference3,
row.preference4]) {
                //if a student has picked a teacher that does not exist then put
the id of the teacher in the bad teachers list
                if (!teacherSet.has(tId)){
                    badTeacherRefs.push(tId)
                }
            }
        }
        //If missing teachers exist return an error
        if (badTeacherRefs.length) {
            const missing = Array.from(new Set(badTeacherRefs)).sort((a,b)=>a-b)
            return res.status(400).json({
                error:
                `Student prefs reference teacher_ids without submissions:
${missing.join(', ')}. ` +
                `Either collect those teachers' prefs or change the teachers
fethced.`,
            })
        }

        //Make sure the index given is correct based on the filtering
        function toValidIndex(val, max) {
            return (Number.isInteger(val) && val >= 1 && val <= max) ? val : 1
        }

        //Building the preference arrays with the mappings
        const student_prefs = studentPrefsRows.map(row => {
            const prefs = [row.preference1, row.preference2, row.preference3,
row.preference4]
            return prefs.map(tId => toValidIndex(teacherMap.get(tId),
teacherIds.length))
        })

        const teacher_prefs = teacherPrefsRows.map(row => {
```

```
            const prefs = [row.preference1, row.preference2, row.preference3,
row.preference4]
            return prefs.map(sId => toValidIndex(studentMap.get(sId),
studentIds.length))
        })

        const payloadPython = {
            numStudents: studentIds.length,
            numTeachers: teacherIds.length,
            student_prefs,
            teacher_prefs,
            teacher_capacity
        }

        //Call the function to run the python code
        const result = await runPython(PY_PATH, payloadPython)
        console.timeEnd('phase:python-solve') //DEBUGGING
        console.log('python result ok?', !!result?.ok, 'assignments len:',
result?.assignments?.length) //DEBUGGING
        if(!result || !result.ok || !Array.isArray(result.assignments)){
            return res.status(500).json({ error: "Solver error", details:
result})
        }
        //-------------------------------------------------------------------
---------
        //-------------------------------------------------------------------
---------
        //-------------------------------------------------------------------
---------

        //Clear all the data in pairings table - this is just for the tests can
delete later
        //await pool.execute(`DELETE FROM pairings`)

        let inserted = 0

        // studentIds are sorted by student_id; assignments are in that same
order
        for (let sIdx = 0; sIdx < studentIds.length; sIdx++) {
            const student_id = studentIds[sIdx]
            const teacherIndex = result.assignments[sIdx] // 0..T
            const teacher_id = (Number.isInteger(teacherIndex) && teacherIndex >=
1 && teacherIndex <= teacherIds.length) ? teacherIds[teacherIndex - 1] : null

            await pool.execute(
                `
                INSERT INTO pairings (student_id, teacher_id)
```

```
                VALUES (?, ?)
                ON DUPLICATE KEY UPDATE teacher_id = VALUES(teacher_id)
                `,
                [student_id, teacher_id]
            )
            inserted++
        }

        //Recompute capacities of teachers
        await pool.execute(
            `
            UPDATE teacher_capacity tc
            LEFT JOIN(
                SELECT p.teacher_id, COUNT(*) AS assigned
                FROM pairings p
                WHERE p.teacher_id is not null
                GROUP BY p.teacher_id
            ) x ON x.teacher_id = tc.teacher_id
            SET tc.capacity_remaining = GREATEST(tc.capacity_total -
IFNULL(x.assigned, 0), 0)
            `
        )

        //Clear the preferences arrays (both) to prepare for the second draw
        await pool.execute(`DELETE FROM student_preferences`)
        await pool.execute(`DELETE FROM teacher_preferences`)

        return res.json({
            ok: true,
            numStudents: studentIds.length,
            numTeachers: teacherIds.length,
            rowsInserted: inserted
        })

    }catch (err){
        console.error('Error running the pairing algorithm', err)
        return res.status(500).json({error: 'Server error', details:
String(err)})
    }
})
```