Smart Invigilator Allocation System

Project Report

National University of Singapore – ISS

Master of Technology

Intelligent Reasoning Systems – Group Project

Term: Jan’2021 to May’2021

SIAS-GRP Project Team Members

Narendernath Baskar	A0230120J
Yusuf Pranggonoh	A0229966J
Neoh Shi Kang	A0229965L
Tan Wee Han	A0125244N

Table of Contents

1. Executive Summary

A Polytechnic in Singapore is manually allocating invigilators duty to test venue. Multiple data points should be considered during the allocation process, which is not only challenging and time-consuming task but also prone to human errors. Though this problem was surfaced by the Polytechnic in Singapore, it is a wider challenge faced by many educational institutions. Therefore, there is an immense need to build an intelligent system to manage the allocation of invigilators.

2. Background

Currently, in one of the Polytechnic in Singapore, the allocation of the invigilator duty to each test venue is performed and planned manually by the examination officer. There are many data points required in order for the examination officer to do the right allocation without errors. It is challenging and time-consuming task.

The data consists of the lecturers, examination dates and time, class group, test venue and test module. Given the complexity of the manual allocation, the examination officer is unable to accommodate other vital information such as invigilators calendar, preference, etc., during the allocation.

There are situations, where the assigned invigilator is not able to perform the duty in the last minute before the commencement of the test. This situation will make the planned allocation even more challenging. To mitigate such a situation, a standby invigilator is assigned to replace the duty, leading to excess allocation.

3. Objective

To develop and implement an intelligent system that provides optimum allocation of invigilator duty to test venues based on a set of defined rules. Build the functionality to handle exceptional situations by reallocation of invigilators with minimal impact to overall planned allocation.

4. Current State: Manual Process

Manual process is being presently followed by the Polytechnic to allocate the invigilator duty to test venues. Last minute reallocation due to invigilator absence on the test day is also handled manually.

Advantages: Continue the process without any additional investment on cost / time / effort.

Disadvantages: All the challenges faced by the polytechnic remains unchanged. The process has to continue with complexity, errors and over allocation. Face the challenges to retain individuals with the process knowledge.

Issues and Risks: Continuing the manual process with erroneous allocations may even lead the polytechnic with reputational risk. Very high keyman risk, as the knowledge is kept within the individuals.

Cost: As this is completely manual process, calculating the manday effort involved for every examination term, $30K is spend on an annual basis.

5. Alternative Solution: Off-the-shelf product

Description: Buy the products that are available in the market and implement them.

Advantages: Allocation process can be automated with minimal human interference. Key challenges can be resolved. Keyman risk can be reduced.

Disadvantages: All the challenges may not be addressed as the off-the-shelf products may not suit specific needs or challenges faced by the polytechnic. These exceptional cases that are not handled by the product needs to be managed by human manually. No visibility to code and logic used by the software.

Issues and Risks: These off-the-shelf software are generally global products with no in-country presence, leading to minimal / nil product support. Implementation and usage are on the own risk of the institution.

Cost: Single user license costing about $250 per year.

6. Proposed Solution: Smart Invigilator Allocation System

Develop and implement Smart Invigilator Allocation System (SIAS). SIAS is an intelligent system that provides optimal allocation of invigilator duty to test venues based on a set of defined rules and capable of reallocation with minimal impact to overall plan.

Advantages: Completely automated solution built with intelligence to allocate invigilator duty based on customized rules and to handle reallocation during exceptional situations. Availability of required documents and source code for any further enhancement or changes.

Disadvantages: As SIAS is group project, planning to be developed as part of course curriculum, there won't be any ongoing support for this solution.

Issues and Risks: Risk for the polytechnic staff in understanding the source code or lack of knowledge in the packages used to build SIAS. This risk may turn out to be higher when the polytechnic wants to enhance or change the product after the implementation.

Cost: As this is academic project which will be deployed on an existing infrastructure, there is zero implementation cost.

Business Benefits:

- SIAS provided intelligent and completely automated solution to meet the customized requirements of the polytechnic.
- SIAS is capable of handling all the defined rules while allocation and reallocation process.
- SIAS takes care of reducing the impact to the overall plan, whenever there is a reallocation due to unexpected situations.
- Availability of source code and detailed documentation for future enhancements and changes.
- Zero development and implementation cost.

7. Smart Invigilator Allocation System

7.1 Approach

Allocation and reallocation of invigilators can be performed at three stages in SIAS.

Stage-1: Initial allocation based on the invigilator available calendar and preference
Stage-2: Reallocation based on invigilator acceptance / changes
Stage-3: Reallocation on the day of examination based on invigilator attendance

The below Figure-1 outlines various data that are required for SIAS to perform the allocation and generate the required information through the three-stage approach.

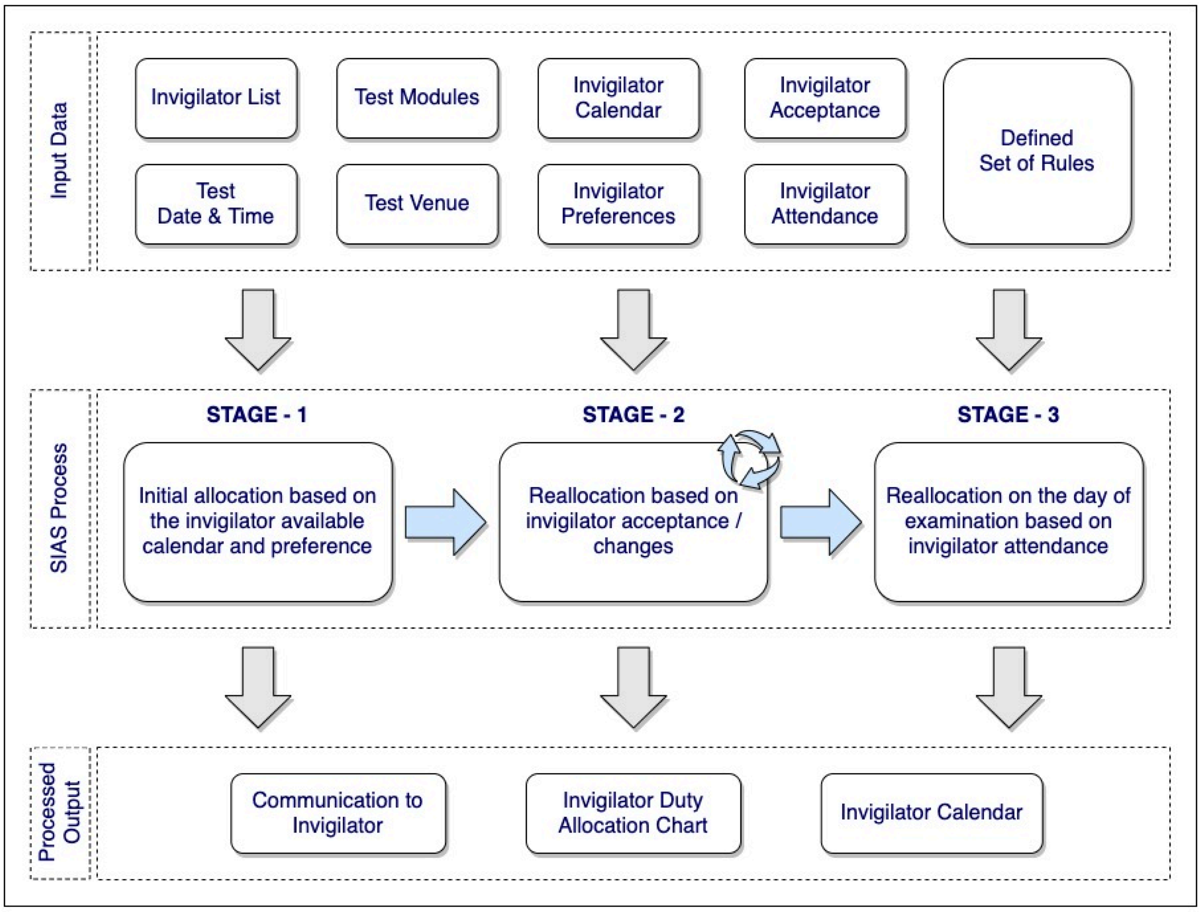

Input Data
Invigilator List
Test Modules
Invigilator Calendar
Invigilator Acceptance
Defined Set of Rules
Test Date & Time
Test Venue
Invigilator Preferences
Invigilator Attendance
SIAS Process
STAGE - 1
STAGE - 2
STAGE - 3
Initial allocation based on the invigilator available calendar and preference
Reallocation based on invigilator acceptance / changes
Reallocation on the day of examination based on invigilator attendance
Processed Output
Communication to Invigilator
Invigilator Duty Allocation Chart
Invigilator Calendar

Figure-1: Operation Diagram

7.2 System Architecture

Smart Invigilator Allocation System (SIAS) is implemented with Python programming language with Flask having web-based user interface for users interact with the application. Java script is used for scheduler and SQLite to store the data. Figure-2 shows the system architecture diagram of SIAS, illustrating the interaction of front-end to back-end to scheduler and database.

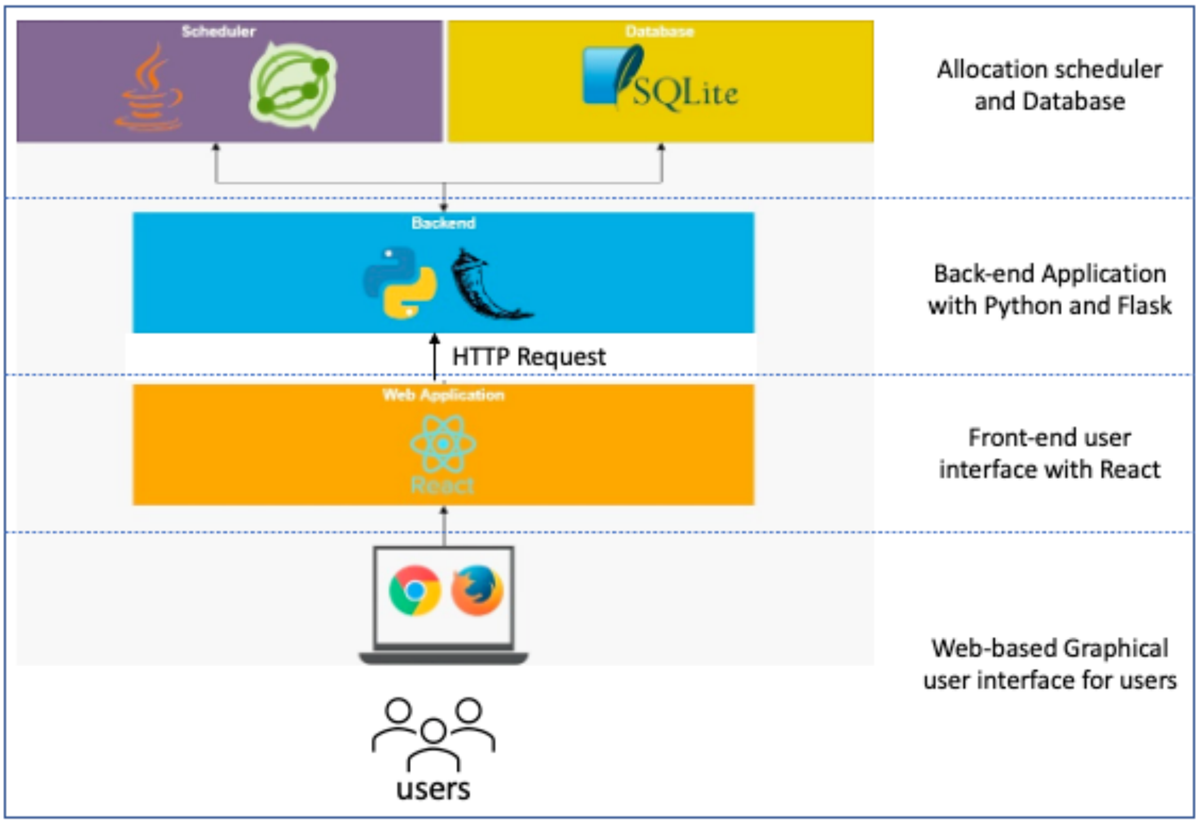

Scheduler
Database
SQLite
Allocation scheduler and Database
Backend
Back-end Application with Python and Flask
HTTP Request
Web Application
React
Front-end user interface with React
Web-based Graphical user interface for users
users

Figure-2: System Architecture Diagram

The below table shows the list of tools and packages used in SIAS application:

#	Tool / Package	Tier	Purpose
1	Python	Backend	Backend Application
2	Flask	Backend	Web apps built to interact with backend
3	Python NumPy	Backend	To perform numerical computation in Python application
4	Python Request	Backend	To make HTTP requests with scheduler
5	Axios React	Frontend	To make HTTP request with Python API
6	Shards-UI	Frontend	To design the frontend web-based UI
7	Material-UI	Frontend	To design the frontend web-based UI
8	SQLite	Database	To store the data and processed allocations
9	Opta Planner	Scheduler	To optimize the allocation schedule

7.3 Key Features

- Accept various data points, such as invigilators list and their calendar, exam dates and time, test venues and test modules.
- Accept invigilators preferences exam dates and time.
- Use the data points and defined rules to form an optimized allocation schedule for the invigilators.
- Communicate with the invigilators on their respective allocation details.
- Enable the invigilators to accept or alter the schedule based on their preferences.
- Reallocate and communicate to impacted invigilators for their confirmation
- Retrieve the attendance of the invigilators on the day of exam.
- If attendance is not marked within a stipulated time, reallocate to available invigilators and communicate to impacted invigilators.

7.4 Input Data

#	Input data	Description
1	Invigilator List	List of Invigilator names to be allocated
2	Test Modules	List of various Test modules for the term
3	Test Date & Time	Date and time of the test conducted in the term
4	Test Venue	List of available venues for the term
5	Invigilator Calendar	Calendar to know the availability of invigilator
6	Invigilator Preference	Allocation preferences provided by the invigilators
7	Invigilator Acceptance	Acceptance / confirmation of allocation by the respective invigilators
8	Invigilator Attendance	Attendance marked by the invigilators on the day of test to show their presence

7.5 Roles

#	Role	Responsibilities
1	CI – Chief Invigilator	• To ensure all the test papers are collected for distribution to SI at least half an hour before the commencement of the test. • If the SI appointed for the test fails to report for duty, CI will appoint the Inv to act as SI and the Stdby will in turn become the Inv. CI is to complete an authorization form to the Inv to inform him/her of his/her appointment as SI and the standby is taking over his role as Inv.
2	SI – Senior Invigilator	• To collect the test papers from CI at least 20 minutes before the test and proceed to the test venue. • To report to CI any Inv who is absent. The CI will then assign a Stdby invigilator to cover the duties.
3	Inv – Invigilator	• To be at the test venue at least 15 minutes before the commencement of the test.

		• To collect answer booklets for the classes you are invigilating and bring them with you to the test venues when invigilating.
4	Stdby – Standby	• To report to CI at school office at least half an hour before the commencement of the test. • To carry out duties as assigned by the CI

7.6 Allocation Rules

1. Each Staff will be allocated 3 schedules during the Examination with the following rules:
    a. None of the Staff will assigned as SI for 3 schedules.
    b. When Staff has an SI role, Then Staff must have Inv. role but not Stdby role. Format example: SI, SI, Inv. OR SI, Inv., Inv.
    c. When Staff has Inv. role, Then Staff will have Stdby role OR Inv. role for 3 schedules but cannot have Stdby role more than 2 schedules. Format example: Inv., Inv., Inv. OR Inv., Inv., Stdby.
2. The schedule assigned must not fall on the same day or back-to-back timing unless the staffs key in their own preferences.
3. Staffs cannot have less than 2 schedules during Examination unless there is an approval from School Director. The schedule will be added on the next Examination dates (3+1).
4. Staffs can have 2 schedules during Examination with approval from School Director. The schedule will not be added on the next Examination dates (3).
5. All preferences / exchange / changes are based on first come first serve basis.
6. Staff cannot exchange preference to Stdby, only between SI or Inv. In case that Stdby wants to exchange, it is between the Stdby.

7.7 Constraints in Allocation Rules

1. A Staff can have at most the number of Duties equal to the Staff's Required Duties which is 3 by default.
2. A Staff cannot be assigned to Duties on the same Day unless the Duty IDs are in the Staff's Preferences.
3. A Staff cannot be assigned to 3 Duties with SI Role.
4. A Staff who has been assigned a Duty with SI Role cannot be assigned a Duty with Stdby Role.
5. A Staff who has been assigned a Duty with Stdby Role cannot be assigned another Duty with Stdby Role.
6. A Staff shall be assigned duties in the Staff's Preferences as much as possible.

7.8 Processed Output

- Communication to Invigilator
- Invigilator Duty Allocation Chart
- Invigilator Calendar

8. Research on Optimization Tools

8.1 Selection Criteria

The goal of our project is staff scheduling, based on complex rules involving number of shifts, combinations of shift types for each staff, shift times, as well as preferences where earlier preference submission times are given higher priority. Given 390 duties required to be filled by more than a 138 staff, this is a combinatorial optimization problem with an extremely large search space. With the complex hard constraints, verification of valid solutions requires significant computing effort. The addition of soft constraints means that the valid solutions can be improved and thus computing time is required to compare them to find an optimal or near-optimal solution. Since computing time and computing power are limited, there is a need for an efficient optimization algorithm to provide the best, if not optimal, solution given a certain time frame.

In addition to computing efficiency, our group also understands that our coding effort and time are limited. With limited expertise in optimization tools and a deadline to meet, the learning curve, availability of learning resources, size of user community, and development effort required are also major considerations in our selection of optimization tools. Also, since no budget is provided for the project, only free-to-use tools were considered. In addition, the availability of sample usage code is highly advantageous for us to save coding time.

In summary, the following criteria were used to select an optimization tool:
1. Ability to model customized hard and soft constraints,
2. High computing efficiency,
3. Flat learning curve,
4. High availability of learning resources,
5. Large size of user community,
6. Low development effort required,
7. Free-to-use,
8. Availability of sample code

8.2 Sourcing and Selection

For ease of development, as well as to seek availability of existing solutions, we first searched for open-sourced scheduling software that can be easily implemented. However, after looking at a few software, we realised that all of them are not customizable to our complex requirements. In addition, we want our solution to be unique for the purpose of the coding project. Hence, we decided to look at schedule optimization solvers.

A quick search on the internet gave us the top 3 optimization solvers which are Google OR-Tools, Optaplanner, and Microsoft Excel Solver. We did not search further because if the solver is hard to find, the support and learning resources for the solver will likely be hardly available. Since Microsoft Excel is not a free-to-use software and requires an effort to be integrated with web applications, it is eliminated from our list of choices. Thus, we are left with OR-Tools and Optaplanner.

The choice between these two is a difficult one, since both meet all the criteria as stated above. We were leaning toward OR-Tools initially, since it is available in Python code, which all of us are familiar with. On the other hand, Optaplanner is only available in Java. OR-Tools is also easier to learn, with more documentation and a larger user community evidenced by the ease of finding discussion on forums and coding discussion sites. However, we decided on Optaplanner upon finding an end-to-end tutorial code implementing a scheduling solution which we can easily follow, customize, and deploy. The tutorial is available in this link: https://quarkus.io/guides/optaplanner

9. Modelling using Optaplanner

9.1 Summary of Optaplanner Operation and Modelling Requirements

To model the invigilator allocation problem into Optaplanner, we first have to understand how Optaplanner works. Optaplanner is an AI constraint solver based around an object with the "@PlanningSolution" annotation.

This object contains
- the input lists of objects annotated "@PlanningVariable" which are problem facts which do not change during solving,
- the input list of objects annotated "@PlanningEntity" that change during solving and becomes an output list of objects after solving, and
- a score object annotated "@PlanningScore" which is calculated during solving.

The constraints are modelled in a "ConstraintProvider" object, which uses the annotations to select instances of objects during solving.

In simple terms, the "SolverManager"
1. takes in the "@PlanningSolution", together with input data containing the "@PlanningVariable"(s) and "@PlanningEntity"(s),
2. calculates the "@PlanningScore" based on the constraints in the "ConstraintProvider",
3. generates a solution output "@PlanningEntity"(s) and the output "@PlanningScore".

Behind the scenes, various algorithms are used in the solving process. The solver can either run till a final best solution is determined, or be stopped in the middle of the solving process to output the best solution at the point of stoppage.

Since this report focuses on the implementation of Optaplanner, the short description above is sufficient to identify the objects that are required to be modelled based on the invigilator allocation problem. More detailed descriptions of Optaplanner can be found on the official Optaplanner guides.
https://docs.optaplanner.org/8.6.0.Final/optaplanner-docs/html_single/index.html

The objects that are required to be modelled are:
- "@PlanningVariable",
- "@PlanningEntity",
- and "ConstraintProvider"

Notes:
- The "@PlanningSolution" is an integration of the "@PlanningVariable", "@PlanningEntity" and "@PlanningScore", thus it does not have to be specially modelled.
- The "@PlanningScore" can be modelled if more complexity is required but it is not required for this problem. The built-in "HardSoftScore" will be used to provide hard and soft scoring for the hard and soft constraints in our problem.

9.2 Understanding the Scheduling Problem

Teaching staff are required to take up invigilation duties during a week of examinations. The Chief Invigilation duties are pre-assigned while the roles of Senior Invigilator (SI), Invigilator (Inv.), and Standby (Stdby) are required to be allocated to staff. Each staff is given a choice of three preferred duties which they will submit on a webpage where the submission timestamps will be recorded. All duties require a staff assigned.

The allocation rules for the scheduling problem are given as follows:

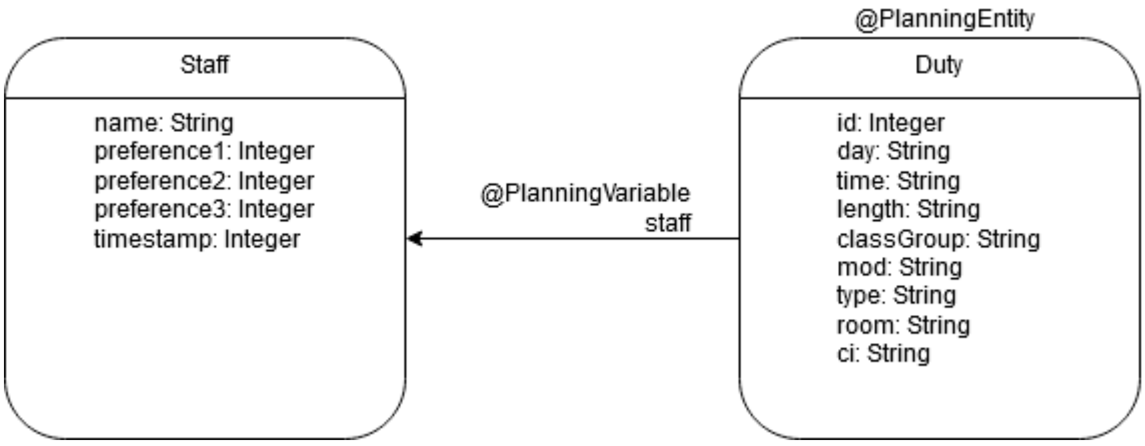
1. Each Staff will be allocated up to 3 schedules during the Examination with the following rules:
    a. None of the Staff will assigned as SI for 3 schedules.
    b. When Staff has an SI role, Then Staff must have Inv. role but NOT Stdby role.
    Format example: SI, SI, Inv. OR SI, Inv., Inv.
    c. When Staff has Inv. role, Then Staff will have Stdby role OR Inv. role for 3 schedules, BUT cannot have Stdby role for 2 or more schedules.
    Format example: Inv., Inv., Inv. OR Inv., Inv., Stdby.
2. The schedule assigned MUST not fall on back-to-back timing, UNLESS the staffs key in their own preferences.
3. Staffs CANNOT have less than 2 schedules during Examination UNLESS there is an approval from School Director. The schedule will be added on the next Examination dates (3+1).
4. Staffs CAN have 2 schedule during Examination WITH approval from School Director. The schedule will NOT be added on the next Examination dates (3).
5. All preferences / exchange / changes are based on first come first served basis.
6. Staff CANNOT exchange preference to Stdby, ONLY between SI or Inv. In case that Stdby wants to exchange, it is between the Stdby.

9.3 Modelling of Optaplanner Objects

From the problem described above, it is clear that there are 2 domain objects that are required to be modelled, the Staff and the Duty (or Duties). In addition, the constraints have to be modelled in the ConstraintProvider object.

A key decision in the modelling process is the choice of “@PlanningVariable” and “@PlanningEntity”. Given that we can assign Staff to Duties, or alternatively Duties to Staff, either Staff or Duties can be the “@PlanningVariable”. However, since all Duties are required to be assigned a Staff, while a Staff can be assigned no Duty if there are insufficient Duties available, Duties will be modelled as the “@PlanningEntity”. Staff will consequently be modelled as the “@PlanningVariable”. The “@PlanningEntity” will be assigned with a “@PlanningVariable”, which means that the Duties will be assigned Staff, which is easier to implement.

The below diagram illustrates the "@PlanningVariable" Staff object, and the "@PlanningEntity" Duty object.



The data available includes a database of Staff and their details, as well as a database of Duties and their details. Since not all data presented are used in the allocation rules, only relevant details are selected to be modelled in the objects. For example, the subject taught by each Staff is not relevant in the allocation process, hence it does not have to be modelled in the object. On the other hand, the role of each Duty is used in the first allocation rule above, thus it is necessary to include the role in the modelled object.

Note: More information is modelled in the Duty object than required because of plans to utilize this information to more accurately represent the preferences of the teaching staff. However, this was not implemented due to time constraint, but the additional information remained in the Duty object.

The Staff object contains the following variables for the following reasons:

- name (String) – this is used as an identifier for the staff. It is represented as a 4-letter string to avoid using real names in this project.
- preference1, preference2, preference3 (Integer) – these are the preferences submitted by the teaching staff through the SIAS platform. The staff select preferences from a list of duties, and these preferences are stored as Integers representing the Duty IDs (more on that below in Duty object). The preferences are used to improve the allocated schedule by trying to allocate the staff to their preferred duties.
- timestamp (Integer) – The dates and times which the staff submitted their preferences are recorded as a Unix timestamp and converted into a normalized reverse timestamp. The purpose of this conversion will be discussed in the modelling of constraints in "ConstraintProvider". This timestamp variable is used to prioritize earlier submissions over later ones in a proportional manner (meaning that other than ranking based on time submitted, the time difference between submissions matters as well).

The Duty object contains the following variables for the following reasons:

- id (Integer) – this is used as an identifier for the duty. Each of the duties provided are assigned an id. This id is selected by the staff during preference selection, as mentioned above.

- day (String) – this is the day of the examination. The day variable is used together with time variable to determine clashes and back-to-back duties to validate solutions based on the allocation rules.
- time (String) – this is the start time of the examination. Used together with day variable to determine clashes and back-to-back duties.
- length (String) – this is the length of the examination. This variable was not used in the solver but has the potential to allow better scheduling solutions.
- classGroup (String) – this is the class group which is taking the examination. This variable was not used in the solver but has the potential to allow better scheduling solutions.
- mod (String) – this is the subject module of the examination. This variable was not used in the solver but has the potential to allow better scheduling solutions.
- type (String) – this refers to the role the staff takes for the duty. This is an important variable which is used to validate solutions in which the allocation rules regarding roles are followed.
- room (String) – this is the room which the examination takes place in. This variable was not used in the solver but has the potential to allow better scheduling solutions.
- ci (String) – this is the chief invigilator assigned to the examination(s). This variable was not used in the solver but has the potential to allow better scheduling solutions.

To better understand why the domain objects are modelled this way, we can look at how the allocation rules are modelled as constraints. In Optaplanner, constraints are modelled as sets of domain objects with conditions that are either desired or undesired. If desired, the constraint will add a reward to the score and if undesired, the constraint will penalize the score. The score that is rewarded or penalized can be the hard score or soft score (or medium score if implemented). The quantity to reward or penalize can also be defined in the constraint.

The allocation rules above are converted into constraints by defining the set of domain objects and the conditions.

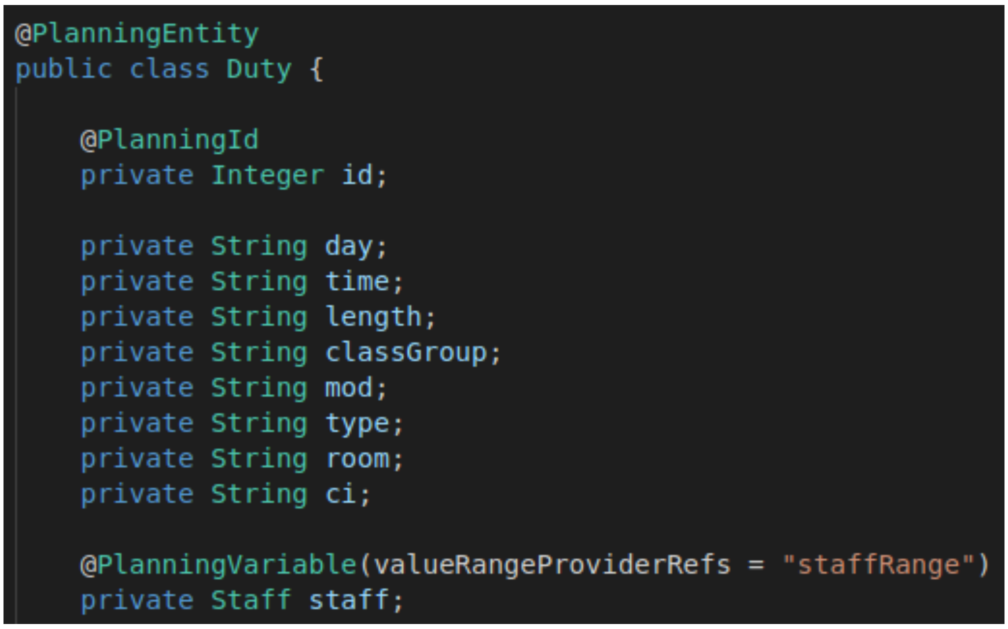
Ref.	**Allocation Rules**	**Constraints**
1	Each Staff will be allocated up to 3 schedules	Number conflict – penalize 1 hard score if more than 3 duties are assigned to a staff
1a	None of the Staff will assigned as SI for 3 schedules.	Three SI conflict – penalize 1 hard score if 3 duties with type “SI” are assigned to a staff
1b	When Staff has an SI role, Then Staff must have Inv. role but NOT Stdby role. Format example: SI, SI, Inv. OR SI, Inv., Inv	SI Stdby conflict – penalize 1 hard score if “SI” and “Stdby” duties are assigned to a staff
1c	When Staff has Inv. role, Then Staff will have Stdby role OR Inv. role for 3 schedules BUT cannot have Stdby role for 2 or more schedules. Format example: Inv., Inv., Inv. OR Inv., Inv., Stdby.	Stdby conflict – penalize 1 hard score if 2 “Stdby” duties are assigned to a staff

2	The schedule assigned MUST not fall on back-to-back timing UNLESS the staffs key in their own preferences.	Day conflict – penalize 1 hard score if duties assigned to a staff are back-to-back. (The preference part of this allocation rule was not implemented but can be implemented by checking if the back-to-back duties are in the staff's preferences.)
3	Staffs CANNOT have less than 2 schedules during Examination UNLESS there is an approval from School Director. The schedule will be added on the next Examination dates (3+1).	This was not implemented but can be implemented by adding a "schedule" variable representing the number of schedules required for the staff and combined with Number conflict. The number of schedules required by each staff can be provided as the input data.
4	Staffs CAN have 2 schedules during Examination WITH approval from School Director. The schedule will NOT be added on the next Examination dates (3).	
5	All preferences / exchange / changes are based on first come first served basis.	Preference – reward soft score proportional to how early the preference is submitted if a duty is allocated to a staff with a preference of that duty (changes and exchanges were not implemented)
6	Staff CANNOT exchange preference to Stdby, ONLY between SI or Inv. In case that Stdby wants to exchange, it is between the Stdby.	Exchanges were not implemented.

10. Implementation of domain objects and constraints in Optaplanner

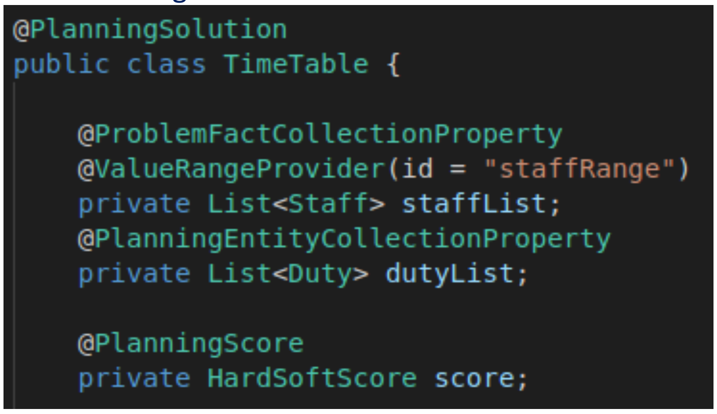
10.1 Domain Objects

The Duty and Staff domain objects are defined as classes in Java. The Staff class is a typical Java class with constructor to initialize object instances and get functions to retrieve the variables. The Duty class on the other hand requires "@" annotations in addition a typical Java class. The Duty class itself is annotated "@PlanningEntity". The id variable within the Duty class is annotated "PlanningId", and the staff variable within the Duty class is annotated "@PlanningVariable". These annotations allow the solver to identify the domain objects and variables. On top of that, the Duty class has a set function for the staff variable to change. This allows the solver to assign Staff objects to Duty objects.



```java
@PlanningEntity
public class Duty {

    @PlanningId
    private Integer id;

    private String day;
    private String time;
    private String length;
    private String classGroup;
    private String mod;
    private String type;
    private String room;
    private String ci;

    @PlanningVariable(valueRangeProviderRefs = "staffRange")
    private Staff staff;
```

The "TimeTable" class is the "@PlanningSolution" which integrates the "@PlanningEntity", "@PlanningVariable", and "@PlanningScore". It defines the lists of Staff objects and Duty objects, so that Optaplanner knows where to look for the input data and generate output data. The "@PlanningScore" used is the "HardSoftScore" available in Optaplanner which contains Integer hard and soft scores.

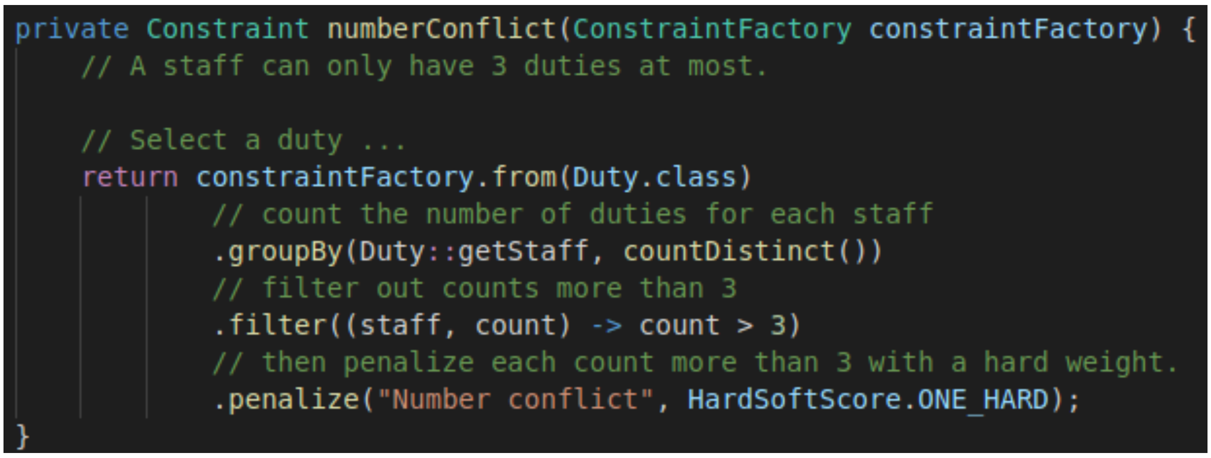


```java
@PlanningSolution
public class TimeTable {

    @ProblemFactCollectionProperty
    @ValueRangeProvider(id = "staffRange")
    private List<Staff> staffList;
    @PlanningEntityCollectionProperty
    private List<Duty> dutyList;

    @PlanningScore
    private HardSoftScore score;
```

10.2 Constraints

The class "TimeTableConstraintProvider" is defined as an implementation of "ConstraintProvider" from Optaplanner library. The Constraint objects are defined with an input Optaplanner's constraintFactory which allows instances of the domain objects to be selected and filtered according to the condition required.

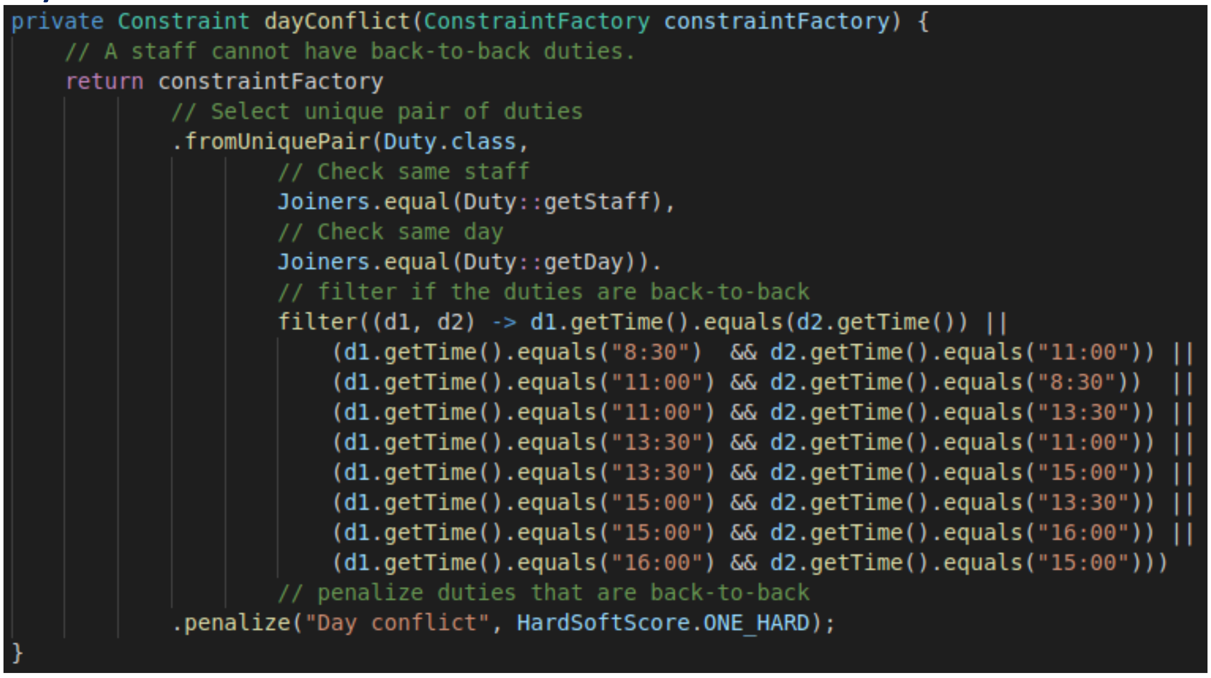
Number Conflict:



```java
private Constraint numberConflict(ConstraintFactory constraintFactory) {
    // A staff can only have 3 duties at most.

    // Select a duty ...
    return constraintFactory.from(Duty.class)
            // count the number of duties for each staff
            .groupBy(Duty::getStaff, countDistinct())
            // filter out counts more than 3
            .filter((staff, count) -> count > 3)
            // then penalize each count more than 3 with a hard weight.
            .penalize("Number conflict", HardSoftScore.ONE_HARD);
}
```

The list of duties is grouped by staff. The groups are then filtered, keeping the groups with more than 3 duties. 1 hard score is penalized for each remaining group.

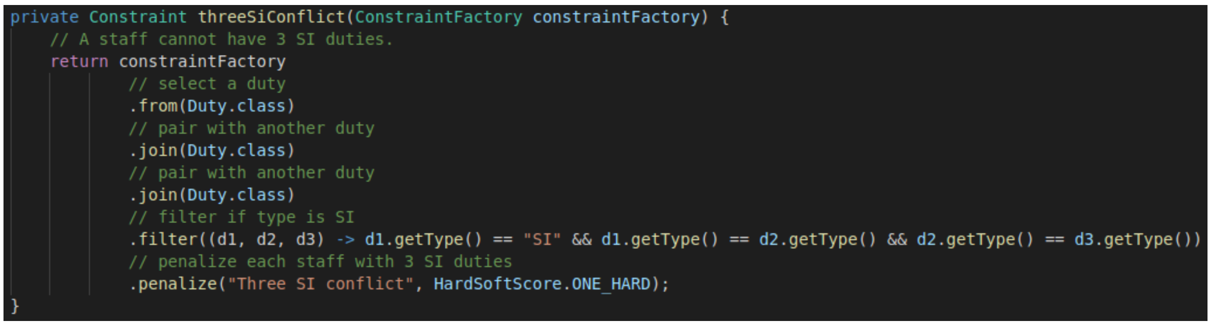
Day Conflict:



```java
private Constraint dayConflict(ConstraintFactory constraintFactory) {
    // A staff cannot have back-to-back duties.
    return constraintFactory
            // Select unique pair of duties
            .fromUniquePair(Duty.class,
                    // Check same staff
                    Joiners.equal(Duty::getStaff),
                    // Check same day
                    Joiners.equal(Duty::getDay)).
                    // filter if the duties are back-to-back
                    filter((d1, d2) -> d1.getTime().equals(d2.getTime()) ||
                        (d1.getTime().equals("8:30")  && d2.getTime().equals("11:00")) ||
                        (d1.getTime().equals("11:00") && d2.getTime().equals("8:30"))  ||
                        (d1.getTime().equals("11:00") && d2.getTime().equals("13:30")) ||
                        (d1.getTime().equals("13:30") && d2.getTime().equals("11:00")) ||
                        (d1.getTime().equals("13:30") && d2.getTime().equals("15:00")) ||
                        (d1.getTime().equals("15:00") && d2.getTime().equals("13:30")) ||
                        (d1.getTime().equals("15:00") && d2.getTime().equals("16:00")) ||
                        (d1.getTime().equals("16:00") && d2.getTime().equals("15:00")))
                    // penalize duties that are back-to-back
            .penalize("Day conflict", HardSoftScore.ONE_HARD);
}
```

Unique pairs of duties are selected, keeping those with same staff and same day. The time of the first duty and the time of the second duty are compared. If the times are the same or back-to-back, they are kept, while the rest are filtered off. For each pair of duties remaining, 1 hard score is penalized.

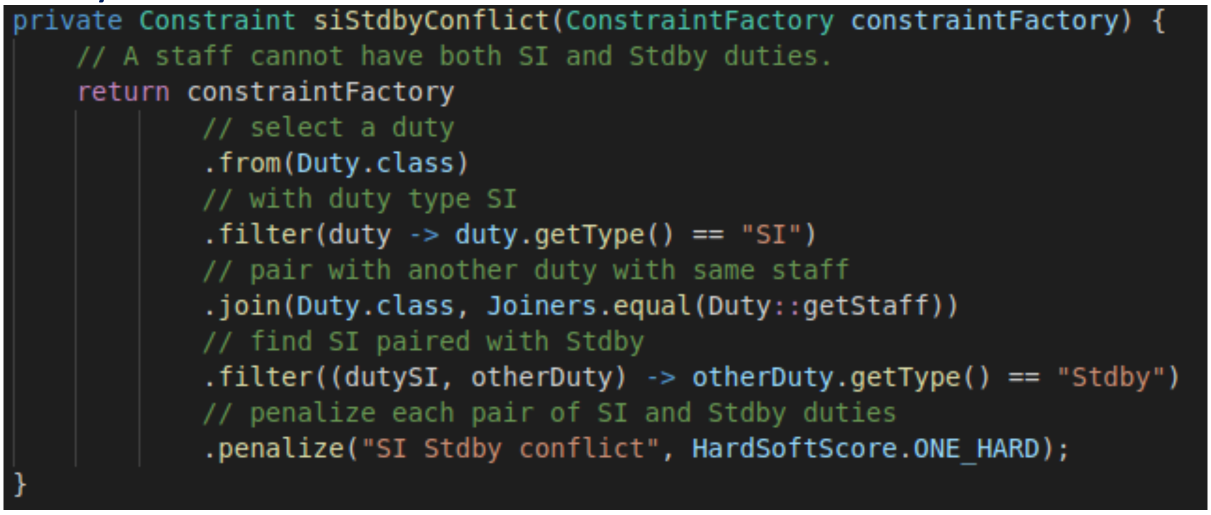
Three SI Conflict:



```
private Constraint threeSiConflict(ConstraintFactory constraintFactory) {
    // A staff cannot have 3 SI duties.
    return constraintFactory
            // select a duty
            .from(Duty.class)
            // pair with another duty
            .join(Duty.class)
            // pair with another duty
            .join(Duty.class)
            // filter if type is SI
            .filter((d1, d2, d3) -> d1.getType() == "SI" && d1.getType() == d2.getType() && d2.getType() == d3.getType())
            // penalize each staff with 3 SI duties
            .penalize("Three SI conflict", HardSoftScore.ONE_HARD);
}
```

Groups of 3 duties are selected, keeping those with all 3 “SI” duties. For each of the groups assigned to staff, 1 hard score is penalized.

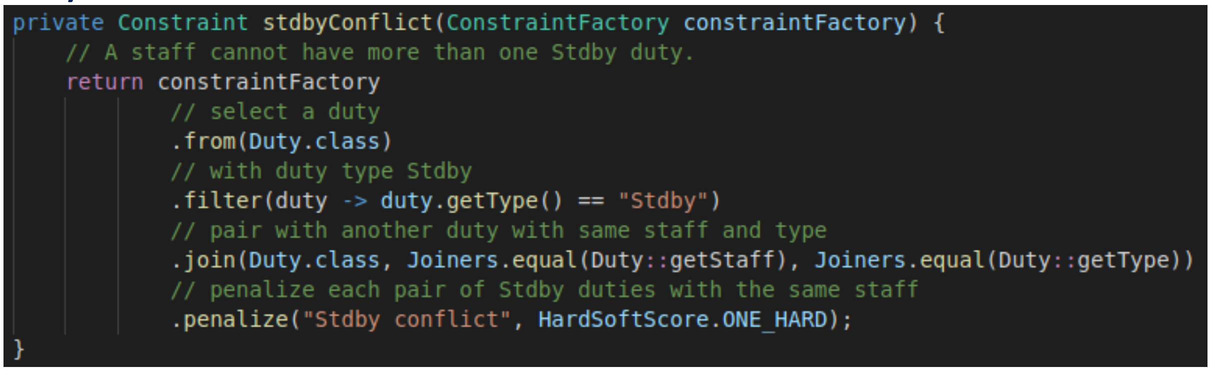
SI Stdby Conflict:



```
private Constraint siStdbyConflict(ConstraintFactory constraintFactory) {
    // A staff cannot have both SI and Stdby duties.
    return constraintFactory
            // select a duty
            .from(Duty.class)
            // with duty type SI
            .filter(duty -> duty.getType() == "SI")
            // pair with another duty with same staff
            .join(Duty.class, Joiners.equal(Duty::getStaff))
            // find SI paired with Stdby
            .filter((dutySI, otherDuty) -> otherDuty.getType() == "Stdby")
            // penalize each pair of SI and Stdby duties
            .penalize("SI Stdby conflict", HardSoftScore.ONE_HARD);
}
```

Duties with type “SI” are selected and paired with another duty of with the same staff. For each pair with “SI” paired to “Stdby”, 1 hard score is penalized.

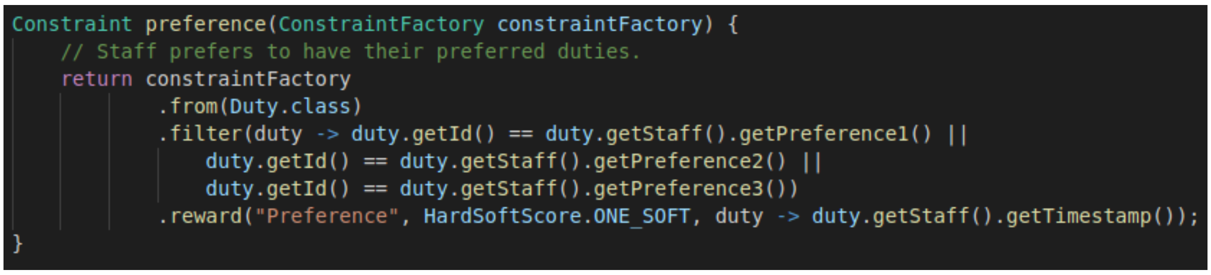
Stdby Conflict:



```
private Constraint stdbyConflict(ConstraintFactory constraintFactory) {
    // A staff cannot have more than one Stdby duty.
    return constraintFactory
            // select a duty
            .from(Duty.class)
            // with duty type Stdby
            .filter(duty -> duty.getType() == "Stdby")
            // pair with another duty with same staff and type
            .join(Duty.class, Joiners.equal(Duty::getStaff), Joiners.equal(Duty::getType))
            // penalize each pair of Stdby duties with the same staff
            .penalize("Stdby conflict", HardSoftScore.ONE_HARD);
}
```

Duties with type “Stdby” are selected and paired with another duty with same staff and same type. For each pair, penalize 1 hard score.

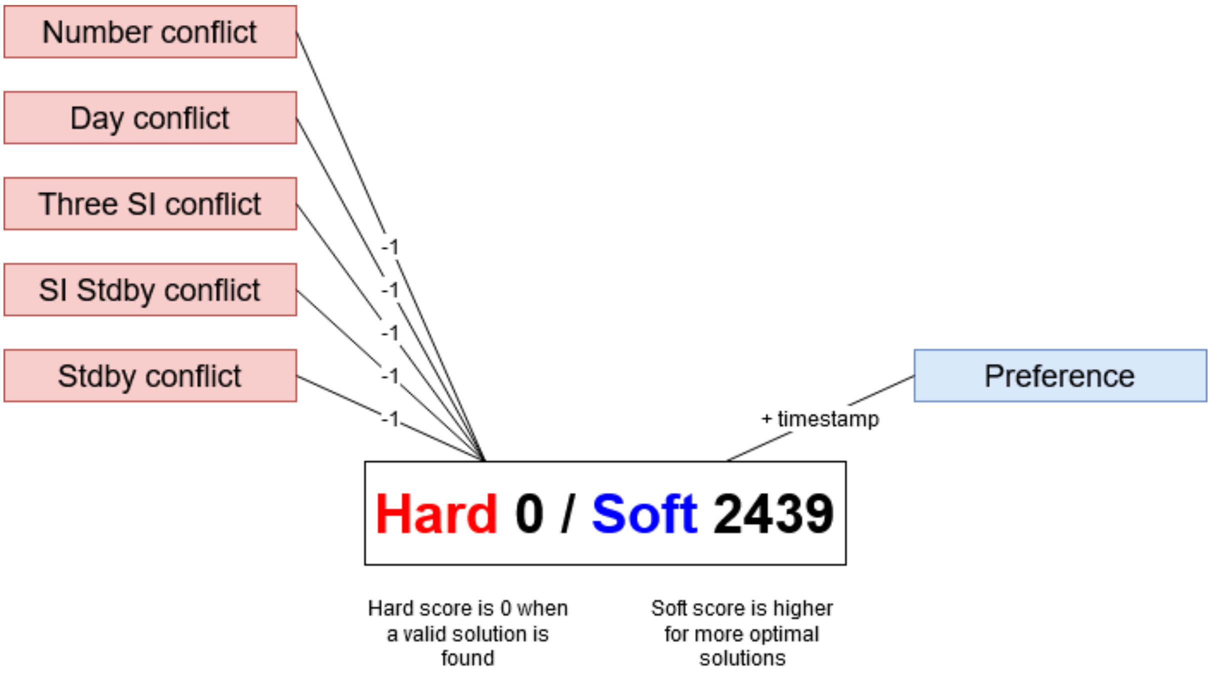
Preference:



```
Constraint preference(ConstraintFactory constraintFactory) {
    // Staff prefers to have their preferred duties.
    return constraintFactory
            .from(Duty.class)
            .filter(duty -> duty.getId() == duty.getStaff().getPreference1() ||
                duty.getId() == duty.getStaff().getPreference2() ||
                duty.getId() == duty.getStaff().getPreference3())
            .reward("Preference", HardSoftScore.ONE_SOFT, duty -> duty.getStaff().getTimestamp());
}
```

Select duties where the duty id is equal to either preference1, preference2, or preference3 of the assigned staff. For each duty, reward soft score equal to the timestamp. Since the timestamp is normalized and reversed, the higher the number, the earlier the preference was submitted. Thus, earlier submissions that are matched are given higher soft scores.

The resulting output score will be 2 numbers, a hard score, and a soft score. This is illustrated in the diagram below. For valid solutions, the hard score is 0, meaning no hard constraints are broken. For optimal solutions, the soft score will be as high as possible, meaning that the preferences are met as much as possible.



11. Assumptions and Constraints

11.1 Assumptions

- The schedule is focusing only on the staff allocation (there is a part where student schedule will need to be consider, as for our system we assume that the student allocation had already been completed).
- The test venue allocation, we assume that it had already been completed.

11.2 Constraints

- To change the allocation method to the staffs, as most of them are comfortable with the existing method.
- Ability to adopt with the proposed system over the existing process.

12. Testing Approach

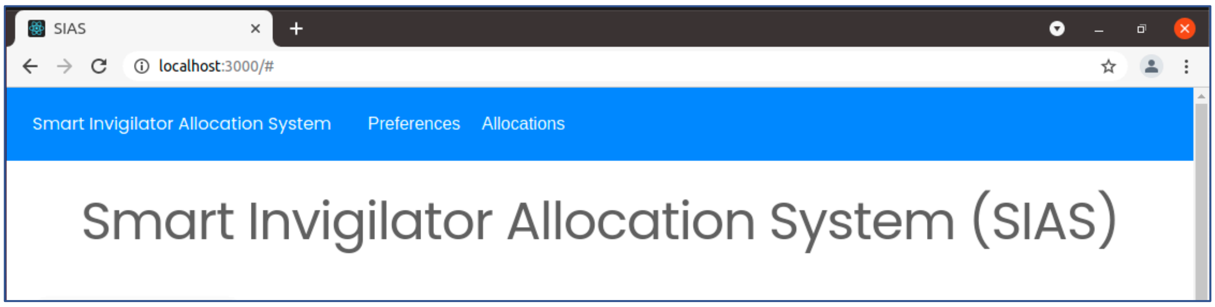
12.1 Test Environment

Test environment was built on Virtual Box with Ubuntu 20.04, Linux operating system on a laptop. The minimum expected hardware requirement is 4GB of RAM and 20GB of hard disk storage space.

12.2 Test Data

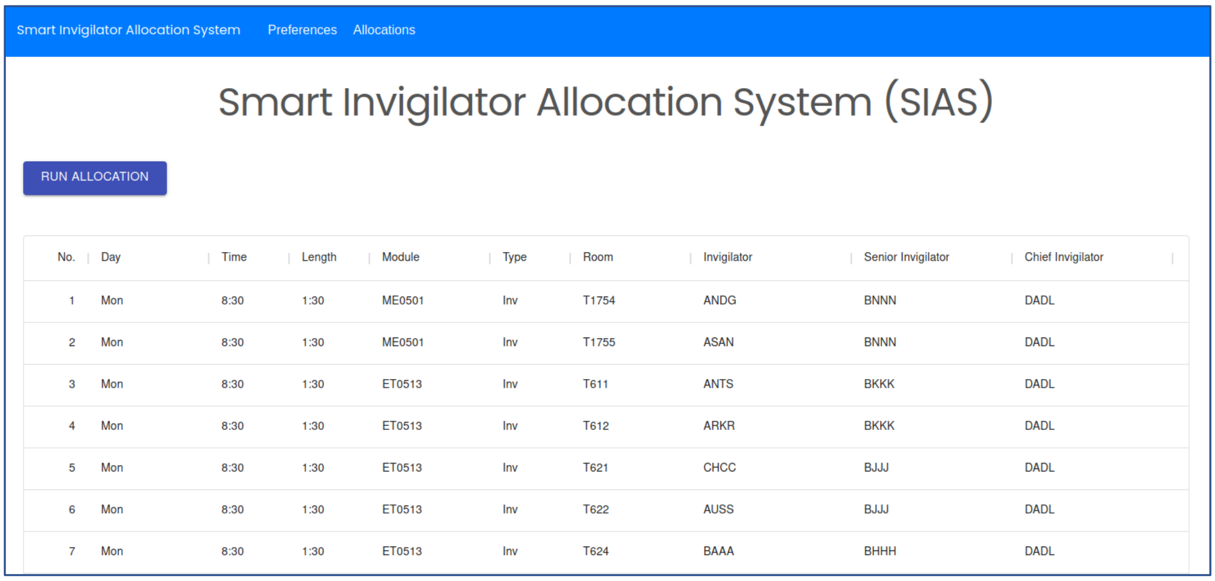
Original production data was taken, and the names are masked to ensure data privacy is maintained. However, text time and number of test modules are retained to test the real-life scenarios. Invigilator preferences were chosen on random and performed the reallocation of schedule. Unit Testing was performed along with the code development. Finally, System Integration Testing was performed to ensure all the components of SIAS application are interacting with each and producing the expected results.

Test Case-1: Invigilator Schedule before allocation should reflect blank sheet as shown below.



Smart Invigilator Allocation System (SIAS)

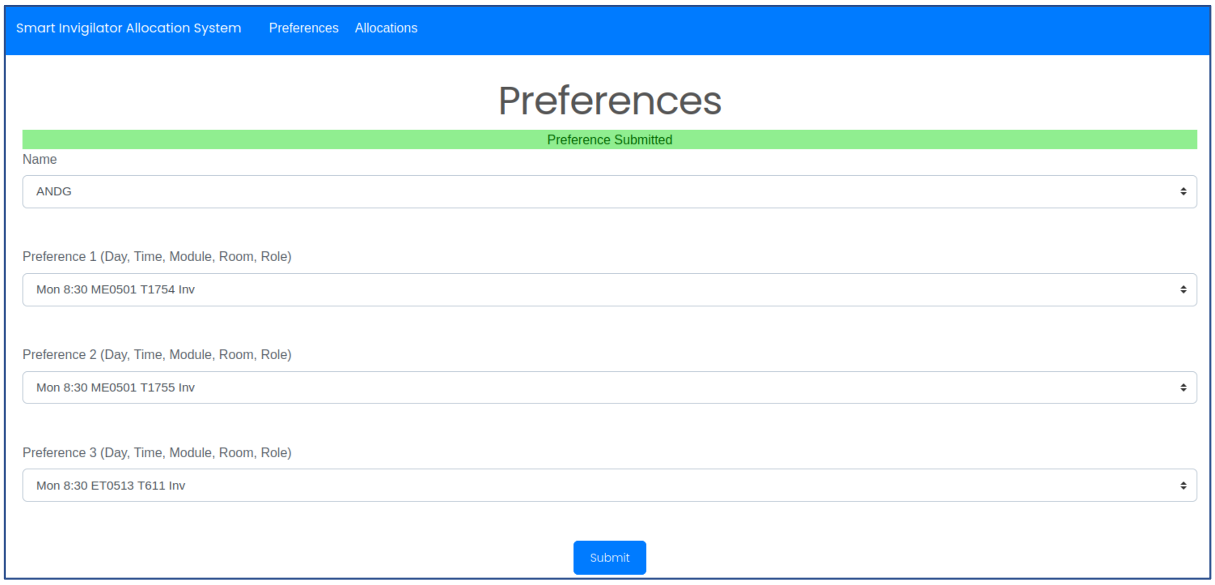
Test Case-2: Invigilator Schedule should be populated after initial allocation



Smart Invigilator Allocation System (SIAS)

No.	Day	Time	Length	Module	Type	Room	Invigilator	Senior Invigilator	Chief Invigilator
1	Mon	8:30	1:30	ME0501	Inv	T1754	ANDG	BNNN	DADL
2	Mon	8:30	1:30	ME0501	Inv	T1755	ASAN	BNNN	DADL
3	Mon	8:30	1:30	ET0513	Inv	T611	ANTS	BKKK	DADL
4	Mon	8:30	1:30	ET0513	Inv	T612	ARKR	BKKK	DADL
5	Mon	8:30	1:30	ET0513	Inv	T621	CHCC	BJJJ	DADL
6	Mon	8:30	1:30	ET0513	Inv	T622	AUSS	BJJJ	DADL
7	Mon	8:30	1:30	ET0513	Inv	T624	BAAA	BHHH	DADL

Test Case-3: Apply the Invigilator Preference and running the allocation, should reallocate the schedule based on the provided preferences.



Preferences

Preference Submitted

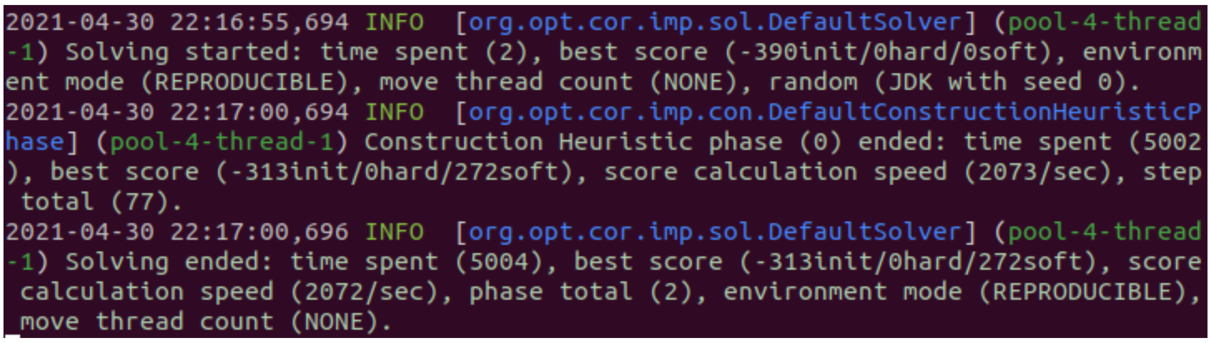
Name: ANDG

Preference 1 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1754 Inv

Preference 2 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1755 Inv

Preference 3 (Day, Time, Module, Room, Role): Mon 8:30 ET0513 T611 Inv

After running the scheduler with preferences the soft score has increased, showing that some preferences have been met.



```
2021-04-30 22:16:55,694 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving started: time spent (2), best score (-390init/0hard/0soft), environm
ent mode (REPRODUCIBLE), move thread count (NONE), random (JDK with seed 0).
2021-04-30 22:17:00,694 INFO  [org.opt.cor.imp.con.DefaultConstructionHeuristicP
hase] (pool-4-thread-1) Construction Heuristic phase (0) ended: time spent (5002
), best score (-313init/0hard/272soft), score calculation speed (2073/sec), step
 total (77).
2021-04-30 22:17:00,696 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving ended: time spent (5004), best score (-313init/0hard/272soft), score
 calculation speed (2072/sec), phase total (2), environment mode (REPRODUCIBLE),
 move thread count (NONE).
```

13. Conclusion and Future expansion

Based on the scope, Smart Invigilator Allocation System was developed to meet the needs of allocation of test schedules for invigilators, however the methods, approach and the code is scalable to include various other dimension and can be expanded to meet any scheduling / allocation needs.

Improvements can be done on various areas; some are listed below:

- Inclusion of user profiles
- Enhancement to graphical user interfaces
- Optimization of scheduler
- Performance improvement in allocation runtime
- Interfacing with Outlook calendar of Invigilators

14. Appendix:

14.1 Project Proposal

Date of proposal: 1 May 2021
Project Title: Smart Invigilator Allocation System (SIAS)
Sponsor/Client: (Name, Address, Telephone No. and Contact Name) Institute of Systems Science (ISS) at 25 Heng Mui Keng Terrace, Singapore NATIONAL UNIVERSITY OF SINGAPORE (NUS) Contact: Mr. GU ZHAN / Lecturer & Consultant Telephone No.: 65-6516 8021 Email: zhan.gu@nus.edu.sg
Background/Aims/Objectives: Currently, in one of the Polytechnic in Singapore, the allocation of the invigilator duty to each test venue is performed and planned manually by the examination officer. There are many data points required for the examination officer to do the right allocation without errors. It is challenging and time-consuming task. To develop and implement an intelligent system that provides optimum allocation of invigilator duty to test venues based on a set of defined rules. Build the functionality to handle exceptional situations by reallocation of invigilators with minimal impact to overall planned allocation.
Requirements Overview: • Research ability • Programming ability • System integration ability
Resource Requirements (please list Hardware, Software, and any other resources) Hardware proposed for consideration: • Minimum of 4GB RAM, 20GB Hard disk storage Software proposed for consideration: • Reasoning systems: OptaPlanner • Backend application: Python, Flask, NumPy, Request • Frontend application: React, Shards-UI, Material-UI, JavaScript • Database: SQLite

Number of Learner Interns required: (Please specify their tasks if possible)

A team of 4 members.

Methods and Standards:

Procedures	**Objective**	**Key Activities**
Requirement Gathering and Analysis	The team should meet with ISS to scope the details of project and ensure the achievement of business objectives.	1. Gather & Analyze Requirements 2. Define internal and External Design 3. Prioritize & Consolidate Requirements 4. Establish Functional Baseline
Technical Construction	· To develop the source code in accordance to the design. · To perform unit testing to ensure the quality before the components are integrated as a whole project	1. Setup Development Environment 2. Understand the System Context, Design 3. Perform Coding 4. Conduct Unit Testing
Integration Testing and acceptance testing	To ensure interface compatibility and confirm that the integrated system hardware and system software meets requirements and is ready for acceptance testing.	1. Prepare System Test Specifications 2. Prepare for Test Execution 3. Conduct System Integration Testing 4. Evaluate Testing 5. Establish Product Baseline
Acceptance Testing	To obtain ISS user acceptance that the system meets the requirements.	1. Plan for Acceptance Testing 2. Conduct Training for Acceptance Testing 3. Prepare for Acceptance Test Execution 4. ISS Evaluate Testing 5. Obtain Customer Acceptance Sign-off
Delivery	To deploy the system into production (ISS standalone server) environment.	1. Software must be packed by following ISS's standard 2. Deployment guideline must be provided in ISS production (ISS standalone server) format 3. Production (ISS standalone server) support and troubleshooting process must be defined.

Team Formation & Registration

Team Name: **SIAS-GRP**
Project Title (repeated): **Smart Invigilator Allocation System (SIAS)**
System Name (if decided): **SIAS**
Team Member 1 Name: **NARENDERNATH BASKAR**
Team Member 1 Matriculation Number: **A0230120J**
Team Member 1 Contact (Mobile/Email): Mobile: **93832433** Email id: **"Narendernath Baskar" <e0690417@u.nus.edu>**
Team Member 2 Name: **YUSUF PRANGGONOH**
Team Member 2 Matriculation Number: **A0229966J**
Team Member 2 Contact (Mobile/Email): Mobile: **81639510** Email Id: **"Yusuf Pranggonoh" <e0687374@u.nus.edu>**
Team Member 3 Name: **NEOH SHI KANG**
Team Member 3 Matriculation Number: **A0229965L**
Team Member 3 Contact (Mobile/Email): Mobile: **91784983** Email Id: **"Neoh Shi Kang" <e0687373@u.nus.edu>**
Team Member 4 Name: **TAN WEE HAN**
Team Member 4 Matriculation Number: **A0125244N**
Team Member 4 Contact (Mobile/Email): Mobile: **85962515** Email Id: **"Tan Wee Han" <e0689794@u.nus.edu>**

14.2 Mapped System Functionalities against knowledge, techniques, and skills

Functions	Knowledge, Techniques, and skills
Knowledge Acquisition	Allocation rules and constraints are used in the knowledge acquisition process. Knowledge base was enhanced with acquired knowledge of allocation and reallocation of invigilators to test venues.
Knowledge Representation	Enhanced knowledge on allocation was used during the reallocation of schedule for invigilators with increase in rules, venues, time etc.
Knowledge Based Reasoning	Used Generative Reasoning techniques to form an optimized allocation schedule. Regenerated the schedule after applying the schedule preferences from invigilators, which is considered as new set of rules.

14.3 Installation Guide

14.3.1 Installation of Pre-Requisites

The SIAS project comprises of 3 components: Frontend, Backend, and Scheduler. All the commands will be run in terminal, unless otherwise stated.

Step-1: For Frontend, install Node Package Manager using the below terminal commands

```
sudo apt update
sudo apt install npm
```

Step-2: For Backend, install Anaconda, then create a conda environment.

a. Install “curl” if not available

```
sudo apt install curl
```

b. Download and install Anaconda for Linux

```
cd /tmp
curl -O https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh
bash Anaconda3-2020.02-Linux-x86_64.sh
```

c. Use "Enter" key to review the license agreement and type "yes" at the bottom to agree the terms.

d. Press "Enter" key to confirm the installation location.

e. Press "yes" when prompted for "Do you wish the installer to initialize Anaconda3 by running conda init? [yes|no]"

f. Activate the installation by typing following command

```
source ~/.bashrc
```

g. Create clean "sias" conda environment (First time only)

```
conda create -n sias python=3.7
```

Step-3: For Scheduler, install JDK 1.8+

```
sudo apt install default-jdk
```

14.3.2 Download the SIAS Package

One of the below options can be followed to download and copy the application code into local drive.

14.3.2.1 Option-1: Download from GitHub using terminal command

This step explains how to download and copy the code from GitHub project repository.

Step-1: Extract the Package from GitHub and move into "/sias" directory using the below terminal commands

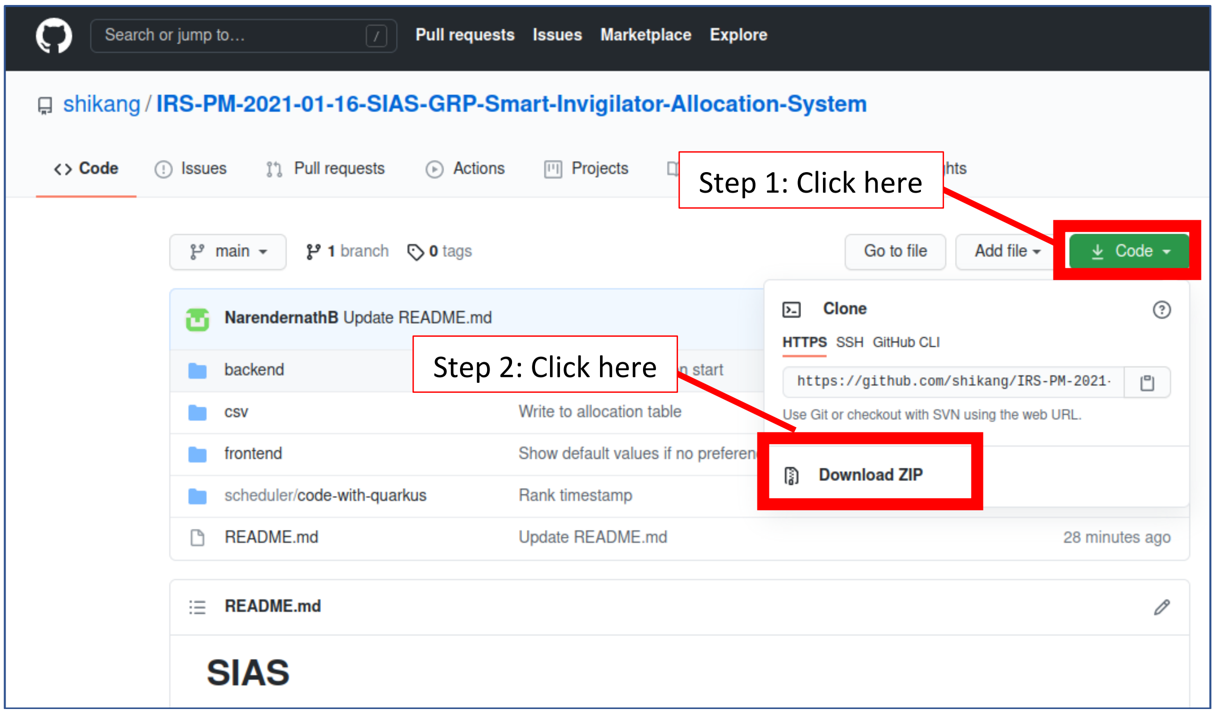
```
git clone https://github.com/shikang/IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-
Allocation-System
sudo mv IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System /sias
cd /sias
```

14.3.2.2 Option-2: Download from GitHub using web browser

SIAS project is located on GitHub in the following link:

https://github.com/shikang/IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System

Step-1: Visit the link and download the zip package to local "Downloads" folder



Step-2: Create "/sias" folder using the below terminal command

```
sudo mkdir /sias
```

Step-3: Extract all the files and folders under “IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System” folder from the zip file “IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System.zip” that was downloaded in “Downloads” folder (Step-1) into “/sias” directory.

14.3.2.3 Option-3: Copy from submitted project package

This step explains how to copy the code from project package submitted in “LumiNUS” portal under “IRS-PM: Practice Module”.

Step-1: Download “IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System.zip” from “LumiNUS” portal under “Files → IRS-PM: Practice Module” into “Downloads” folder.

Step-2: Create “sias” folder using the below terminal command

```
sudo mkdir /sias
```

Step-3: Extract all the files and folders under “IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System” folder from the zip file “IRS-PM-2021-01-16-SIAS-GRP-Smart-Invigilator-Allocation-System.zip” that was downloaded in “Downloads” folder (Step-1) into “/sias” directory.

14.3.3 Installation of SIAS Application

The SIAS project comprises of 3 components: Frontend, Backend, and Scheduler. All the commands will be run in terminal, unless otherwise stated.

Step-1: For Frontend, install dependencies

```
cd /sias/frontend
npm install
```

Step-2: For Backend, activate “sias” conda environment and install dependencies

```
cd /sias/backend
conda activate sias
pip install -r requirements.txt
```

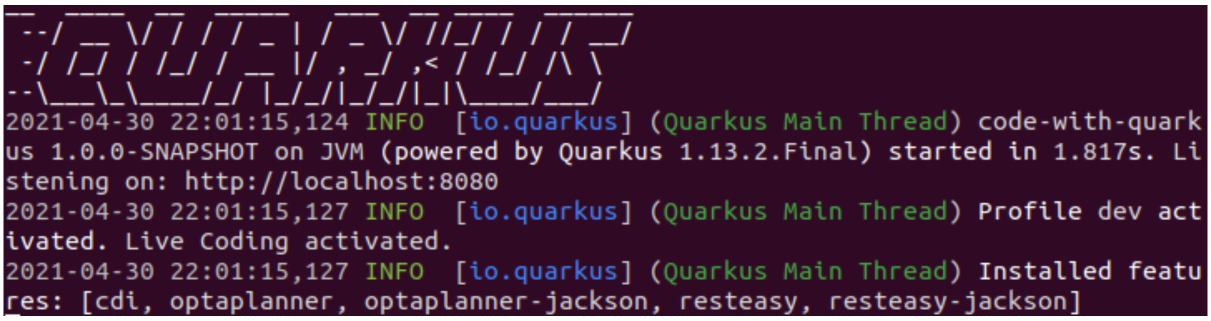
Note: In case if “flask-cors” is not installed from the above command, run the below command

```
pip install flask_cors
```

Step-3: Install and start the Scheduler

```
cd /sias/scheduler/code-with-quarkus
./mvnw compile quarkus:dev
```

The installation may take a few minutes. After installation, you should see the following in the terminal:



```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2021-04-30 22:01:15,124 INFO  [io.quarkus] (Quarkus Main Thread) code-with-quark
us 1.0.0-SNAPSHOT on JVM (powered by Quarkus 1.13.2.Final) started in 1.817s. Li
stening on: http://localhost:8080
2021-04-30 22:01:15,127 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev act
ivated. Live Coding activated.
2021-04-30 22:01:15,127 INFO  [io.quarkus] (Quarkus Main Thread) Installed featu
res: [cdi, optaplanner, optaplanner-jackson, resteasy, resteasy-jackson]
```

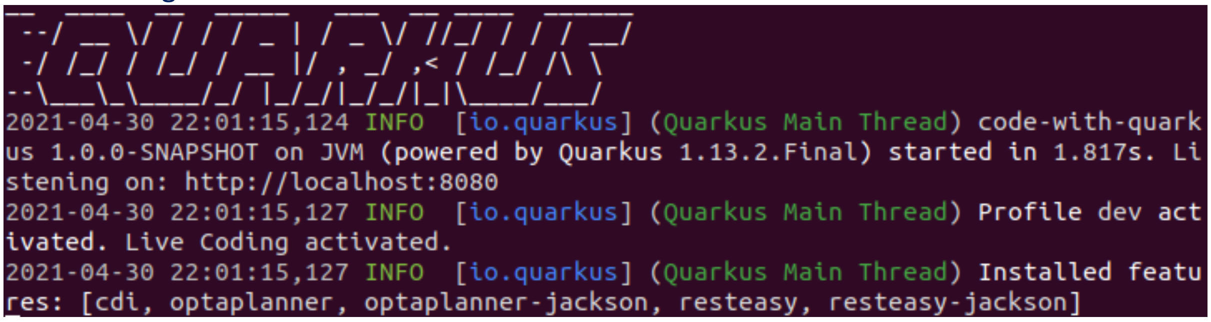
14.3.4 Start Services for SIAS Application

Services for SIAS application should be started in this sequence: Scheduler -> Backend -> Frontend

Step-1: Start the Scheduler
Open a new Terminal and execute the following commands

```
cd /sias/scheduler/code-with-quarkus
./mvnw compile quarkus:dev
```

After starting the Scheduler the text similar to below screenshot will be seen in the terminal.

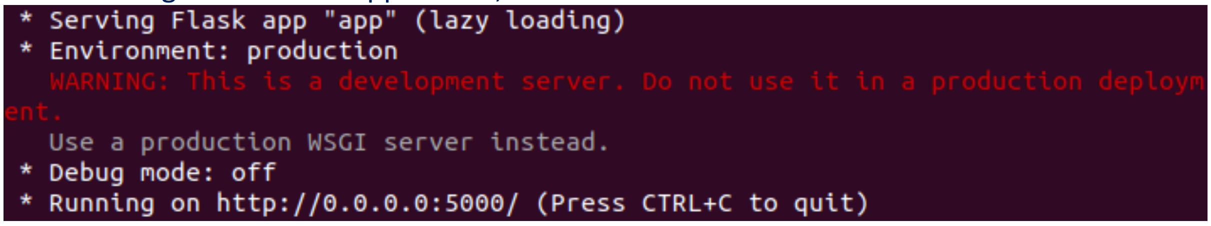


```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2021-04-30 22:01:15,124 INFO  [io.quarkus] (Quarkus Main Thread) code-with-quark
us 1.0.0-SNAPSHOT on JVM (powered by Quarkus 1.13.2.Final) started in 1.817s. Li
stening on: http://localhost:8080
2021-04-30 22:01:15,127 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev act
ivated. Live Coding activated.
2021-04-30 22:01:15,127 INFO  [io.quarkus] (Quarkus Main Thread) Installed featu
res: [cdi, optaplanner, optaplanner-jackson, resteasy, resteasy-jackson]
```

Step-2: Start the Backend
Open a new Terminal and execute the following commands

```
cd /sias/backend
conda activate sias
python app.py
```

After starting the backend application, the below text will be seen in the terminal.

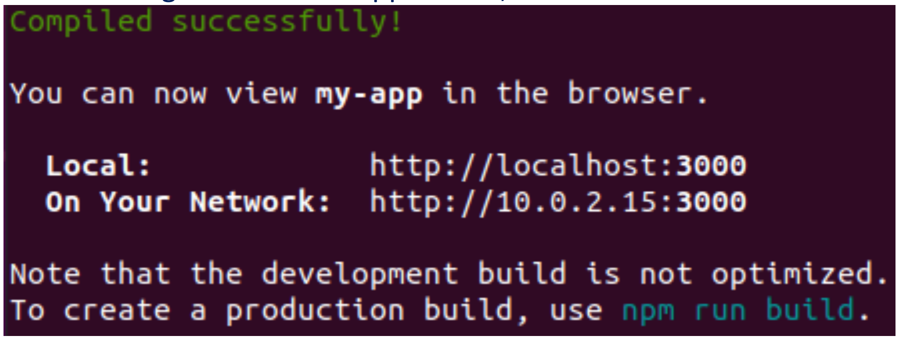


```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deploym
ent.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Step-3: Start the Frontend
Open a new Terminal and execute the following commands

```
cd /sias/frontend
npm start
```

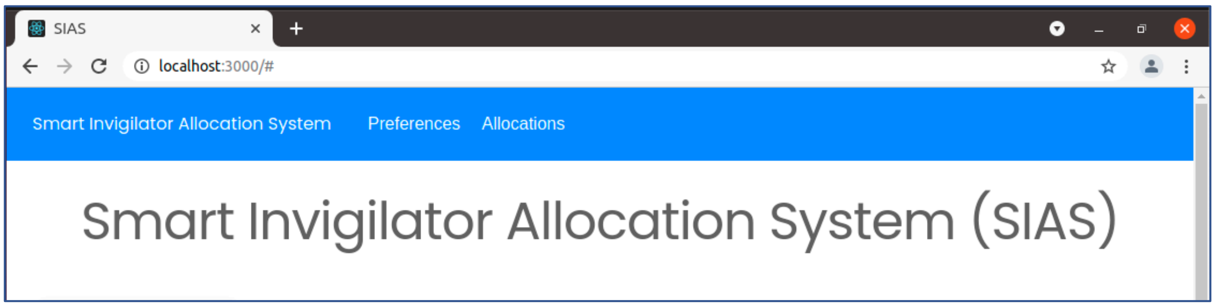
After starting the frontend application, the below text will be seen in the terminal.



```
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.2.15:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Upon starting the Frontend, the default browser will open up the SIAS application. If not opened, you can open your browser and go to the link: http://localhost:3000

The below page will be displayed.



Smart Invigilator Allocation System (SIAS)

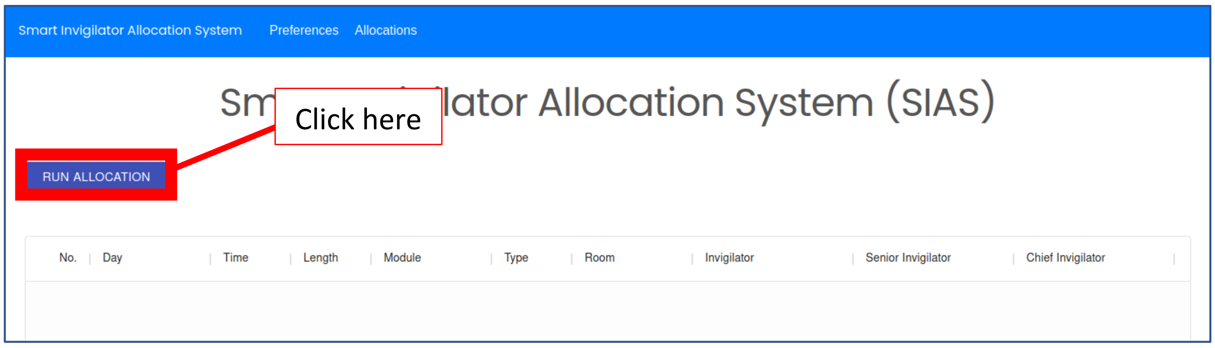
14.4 User Manual

Keynote:

- Ensure all the services are started for SIAS application as specified in section 11.3.4
- Open your web browser and go to the link: http://localhost:3000

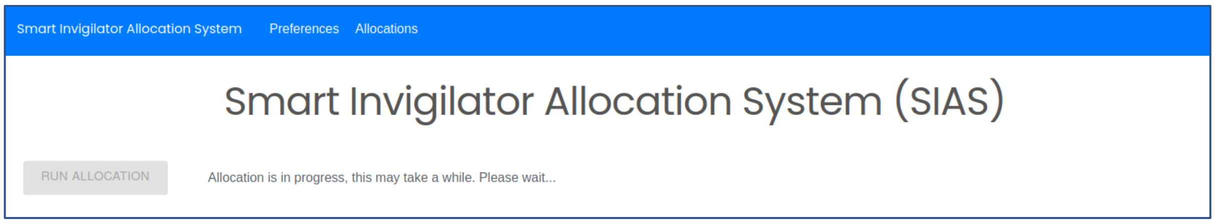
14.4.1 Allocation of Invigilation Schedule

1. On the browser, the SIAS "Allocation" page is displayed showing a blank table. This is because the scheduling is not done. To run the SIAS scheduler and obtain a schedule, click on the "RUN ALLOCATION" button.


Smart Invigilator Allocation System
Preferences
Allocations
Smart Invigilator Allocation System (SIAS)
Click here
RUN ALLOCATION

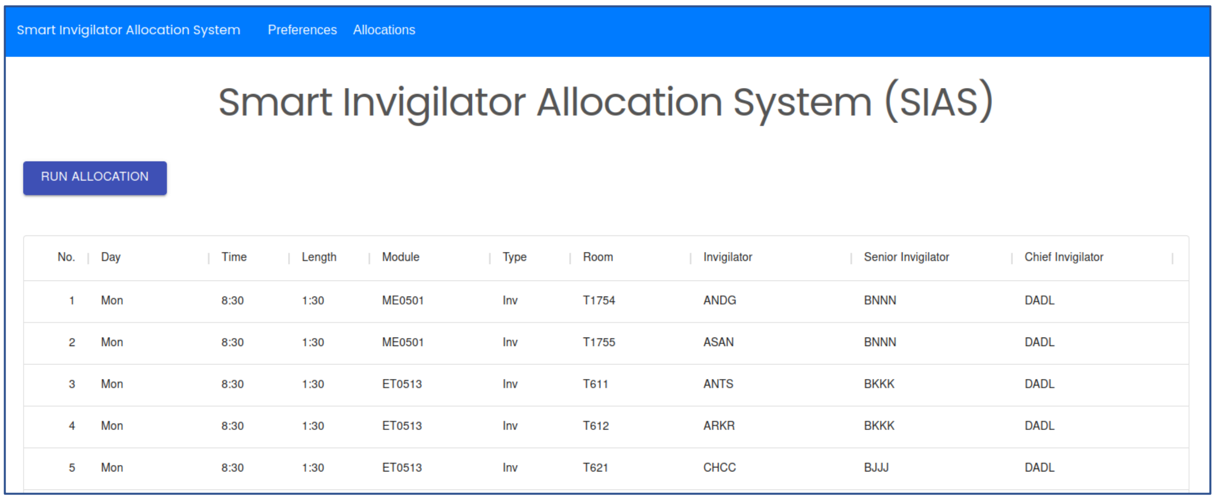
No.	Day	Time	Length	Module	Type	Room	Invigilator	Senior Invigilator	Chief Invigilator

2. Upon clicking the button, you should see the following message in the browser: "Allocation is in progress, this may take a while. Please wait..."


Smart Invigilator Allocation System
Preferences
Allocations
Smart Invigilator Allocation System (SIAS)
RUN ALLOCATION

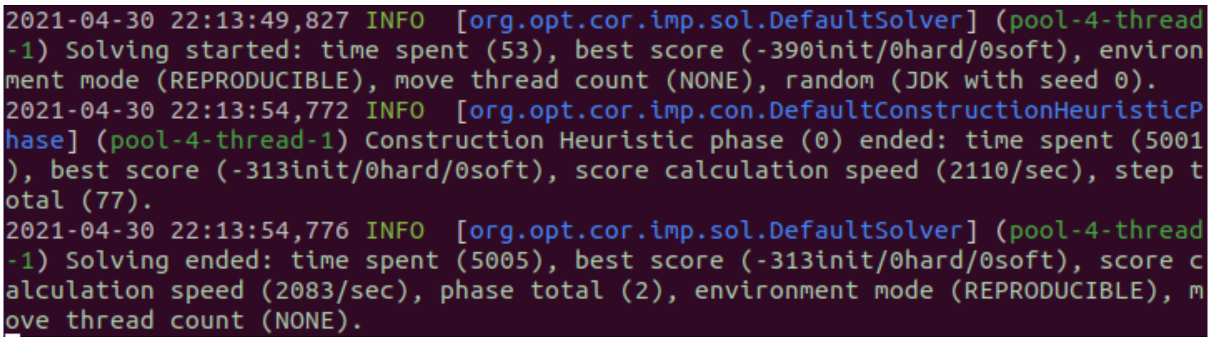
Allocation is in progress, this may take a while. Please wait...

3. When the allocation is complete, you will see that the table that was previously blank is now populated.


Smart Invigilator Allocation System
Preferences
Allocations
Smart Invigilator Allocation System (SIAS)
RUN ALLOCATION

No.	Day	Time	Length	Module	Type	Room	Invigilator	Senior Invigilator	Chief Invigilator
1	Mon	8:30	1:30	ME0501	Inv	T1754	ANDG	BNNN	DADL
2	Mon	8:30	1:30	ME0501	Inv	T1755	ASAN	BNNN	DADL
3	Mon	8:30	1:30	ET0513	Inv	T611	ANTS	BKKK	DADL
4	Mon	8:30	1:30	ET0513	Inv	T612	ARKR	BKKK	DADL
5	Mon	8:30	1:30	ET0513	Inv	T621	CHCC	BJJJ	DADL

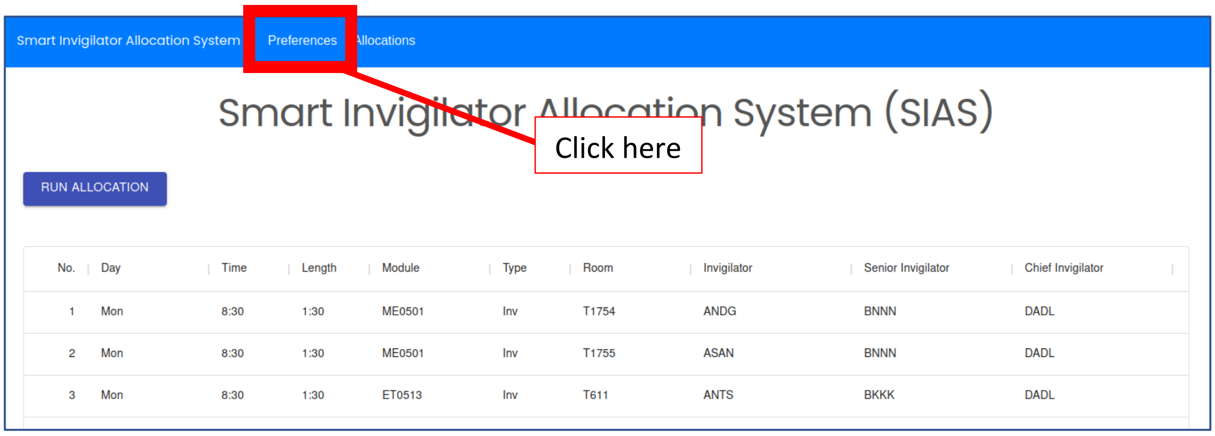
4. Looking at the terminal running the Scheduler, you can see the scheduler output such as time spent and score.



```
2021-04-30 22:13:49,827 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving started: time spent (53), best score (-390init/0hard/0soft), environ
ment mode (REPRODUCIBLE), move thread count (NONE), random (JDK with seed 0).
2021-04-30 22:13:54,772 INFO  [org.opt.cor.imp.con.DefaultConstructionHeuristicP
hase] (pool-4-thread-1) Construction Heuristic phase (0) ended: time spent (5001
), best score (-313init/0hard/0soft), score calculation speed (2110/sec), step t
otal (77).
2021-04-30 22:13:54,776 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving ended: time spent (5005), best score (-313init/0hard/0soft), score c
alculation speed (2083/sec), phase total (2), environment mode (REPRODUCIBLE), m
ove thread count (NONE).
```

14.4.2 Adding Invigilator Preferences to the Schedule

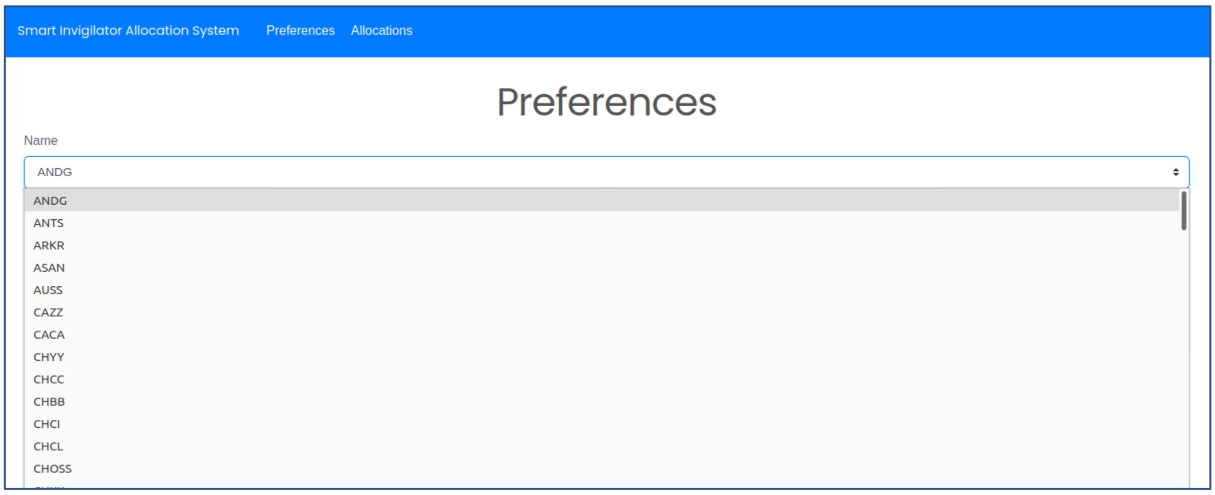
1. Other than the Duty schedule and Staff list, SIAS also considers for each Staff's preferences for the invigilation duties. To add in preferences, first, click on "Preferences" in the header bar.



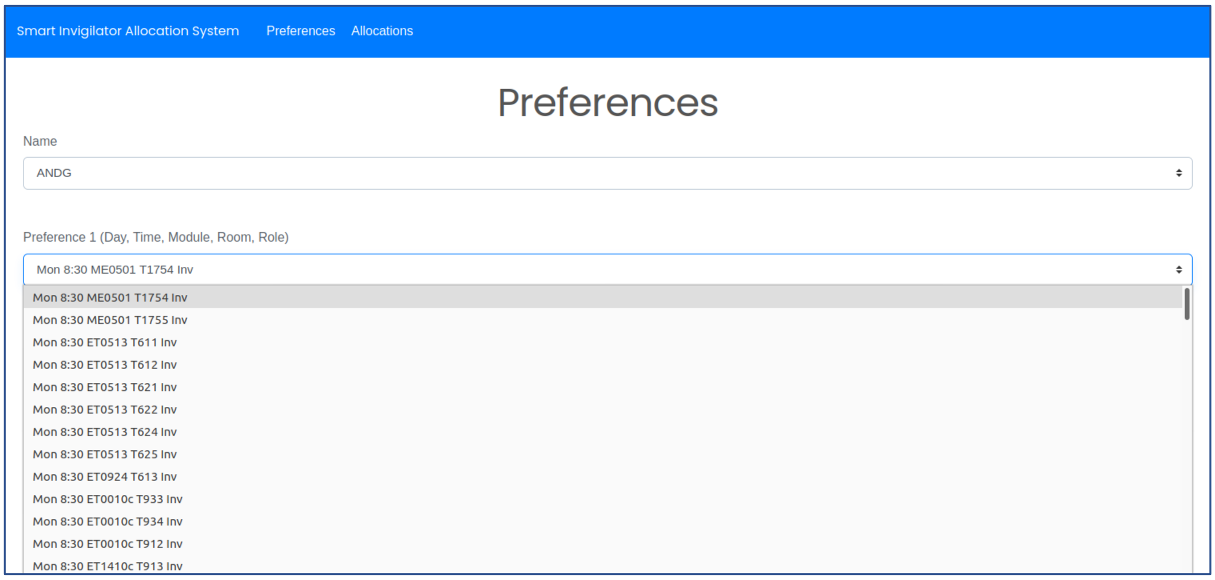
Smart Invigilator Allocation System
Preferences
Allocations
Smart Invigilator Allocation System (SIAS)
Click here
RUN ALLOCATION

No.	Day	Time	Length	Module	Type	Room	Invigilator	Senior Invigilator	Chief Invigilator
1	Mon	8:30	1:30	ME0501	Inv	T1754	ANDG	BNNN	DADL
2	Mon	8:30	1:30	ME0501	Inv	T1755	ASAN	BNNN	DADL
3	Mon	8:30	1:30	ET0513	Inv	T611	ANTS	BKKK	DADL

2. In the "Preferences" page, select the Staff in the "Name" section.



3. Then, select the 3 preferred duties for the Staff selected. The duties are labeled by "Day, Time, Module, Room, Role". (Note that all 3 preferences are given the same priority. Earlier submissions are given higher priorities.)



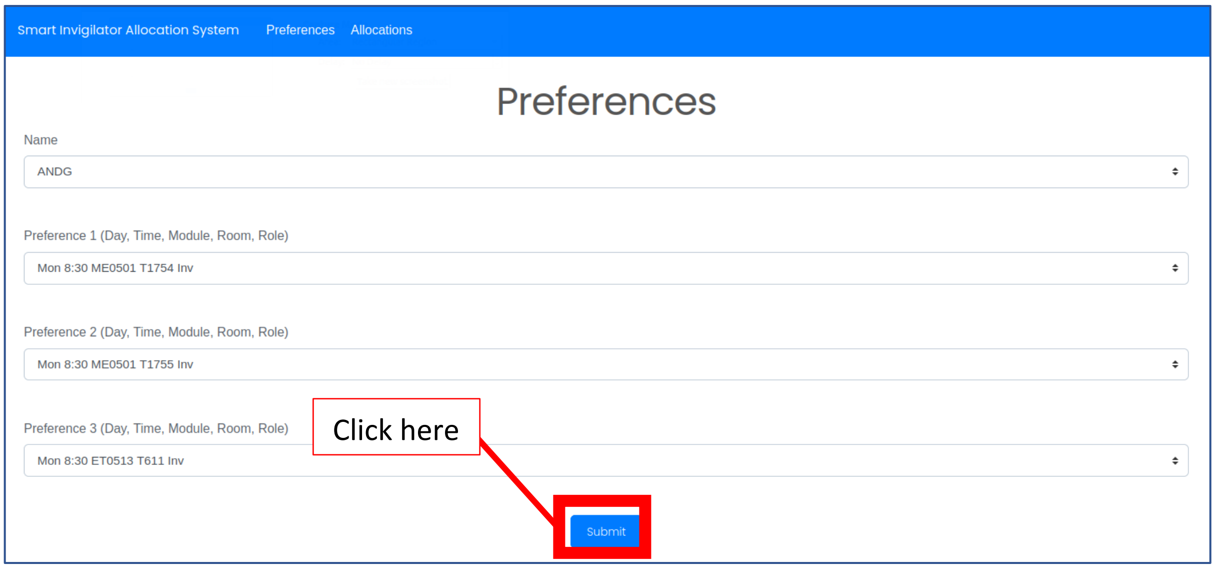
Preferences

Name: ANDG

Preference 1 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1754 Inv

Mon 8:30 ME0501 T1754 Inv
Mon 8:30 ME0501 T1755 Inv
Mon 8:30 ET0513 T611 Inv
Mon 8:30 ET0513 T612 Inv
Mon 8:30 ET0513 T621 Inv
Mon 8:30 ET0513 T622 Inv
Mon 8:30 ET0513 T624 Inv
Mon 8:30 ET0513 T625 Inv
Mon 8:30 ET0924 T613 Inv
Mon 8:30 ET0010c T933 Inv
Mon 8:30 ET0010c T934 Inv
Mon 8:30 ET0010c T912 Inv
Mon 8:30 ET1410c T913 Inv

4. Then click "Submit" to add the Staff's preferences into the database. You should see a message "Preference Submitted".



Preferences

Name: ANDG

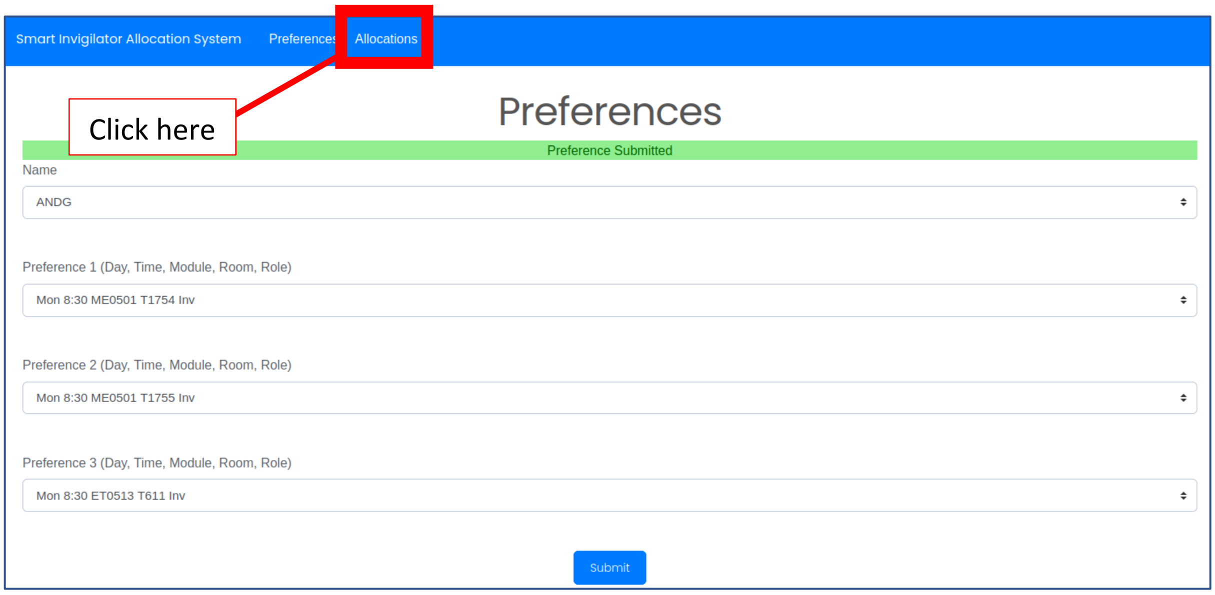
Preference 1 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1754 Inv

Preference 2 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1755 Inv

Preference 3 (Day, Time, Module, Room, Role): Mon 8:30 ET0513 T611 Inv

14.4.3 Allocation of Invigilator Schedule with their Preferences

1. After preferences for all Staff are added into the database, the invigilation schedule can be obtained again with Staff's preferences taken into account. Click on "Allocations" in the header bar to return to the "Allocations" page.



Preferences

Preference Submitted

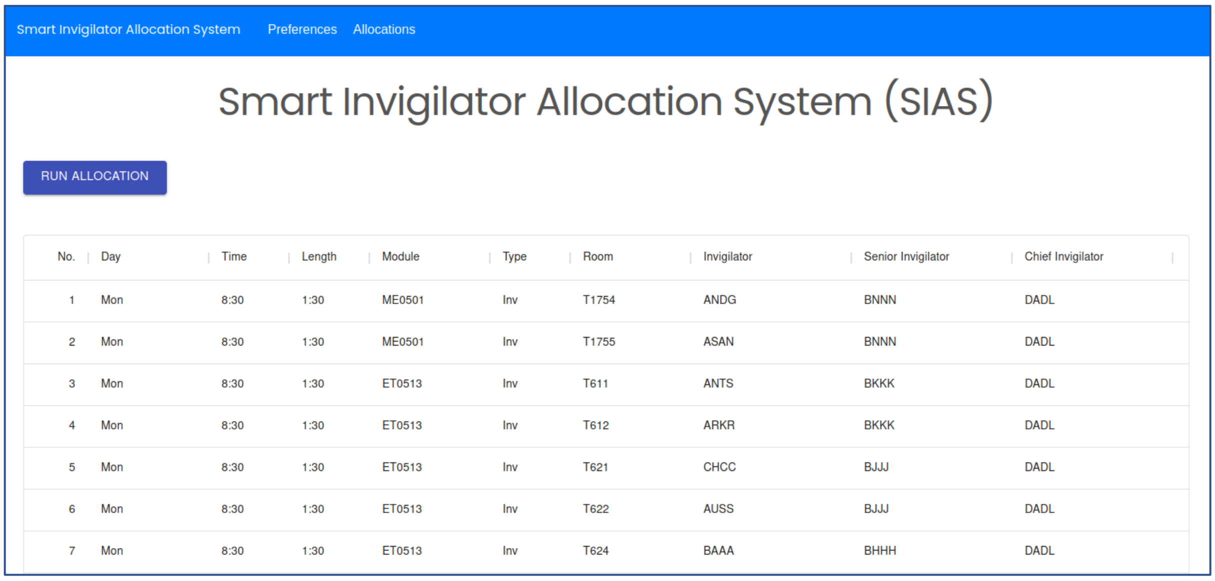
Name: ANDG

Preference 1 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1754 Inv

Preference 2 (Day, Time, Module, Room, Role): Mon 8:30 ME0501 T1755 Inv

Preference 3 (Day, Time, Module, Room, Role): Mon 8:30 ET0513 T611 Inv

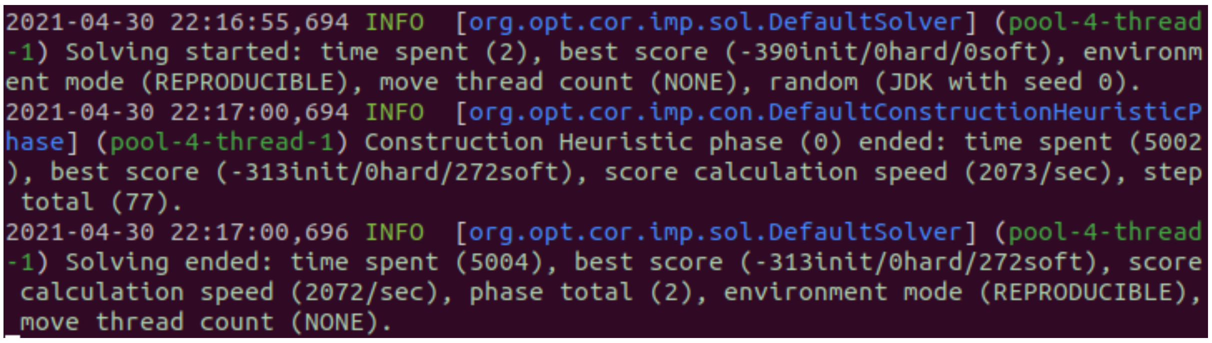
2. When the reallocation is complete, you will see the table populated with the inclusion of invigilator preferences.



Smart Invigilator Allocation System (SIAS)

No.	Day	Time	Length	Module	Type	Room	Invigilator	Senior Invigilator	Chief Invigilator
1	Mon	8:30	1:30	ME0501	Inv	T1754	ANDG	BNNN	DADL
2	Mon	8:30	1:30	ME0501	Inv	T1755	ASAN	BNNN	DADL
3	Mon	8:30	1:30	ET0513	Inv	T611	ANTS	BKKK	DADL
4	Mon	8:30	1:30	ET0513	Inv	T612	ARKR	BKKK	DADL
5	Mon	8:30	1:30	ET0513	Inv	T621	CHCC	BJJJ	DADL
6	Mon	8:30	1:30	ET0513	Inv	T622	AUSS	BJJJ	DADL
7	Mon	8:30	1:30	ET0513	Inv	T624	BAAA	BHHH	DADL

3. After running the scheduler with preferences added, you will see that the soft score has increased, meaning that some preferences have been met.



```
2021-04-30 22:16:55,694 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving started: time spent (2), best score (-390init/0hard/0soft), environm
ent mode (REPRODUCIBLE), move thread count (NONE), random (JDK with seed 0).
2021-04-30 22:17:00,694 INFO  [org.opt.cor.imp.con.DefaultConstructionHeuristicP
hase] (pool-4-thread-1) Construction Heuristic phase (0) ended: time spent (5002
), best score (-313init/0hard/272soft), score calculation speed (2073/sec), step
 total (77).
2021-04-30 22:17:00,696 INFO  [org.opt.cor.imp.sol.DefaultSolver] (pool-4-thread
-1) Solving ended: time spent (5004), best score (-313init/0hard/272soft), score
 calculation speed (2072/sec), phase total (2), environment mode (REPRODUCIBLE),
 move thread count (NONE).
```

14.5 Individual Project Report

Team Member 1: **NARENDERNATH BASKAR (A0230120J)**
Personal Contribution to the group project: - Governance of the project planning and execution - Analyze the requirements and define the project scope - Translate requirements into workable function points and features - Assessment of alternative solutions that can address the objective of the project - Define the operational flow and approach for the application - Define and document the Business Case, Architecture and Testing approach - Outline the script for product promotion video and demonstration video - Perform System Integration Testing - Validate the accuracy and sequence of Installation Guide and User Manual - Collate and document the Project Report
Useful Lessons Learnt: - Understood how to apply the theoretical learnings from reasoning systems to solve practical problems - Learnt many tools and packages available as open source that can solve many real-life problems in the workplace - Gaining confidence to move forward in AI concepts and implementation approaches
Applying the knowledge and skills in workplace: - Application of reasoning systems used in this project will definitely help me in my workplace to implement them at various aspects - I am sure this is not only going to improve the performance in the workplace, but also going to increase the quality in my data analytics and MI reports

Team Member 2: **YUSUF PRANGGONOH (A0229966J)**
Personal Contribution to the group project: - Act as project owner to conceptualize business requirements and prepare requirements specification. - Interview the person-in-charge (PIC) from the institution (Singapore Polytechnic) and the PIC provide the dataset for the project. - Produce a promotional video and demonstration video of the project. - Prepare the source data, such as staffs and duties. - Implement frontend, i.e. View Allocation page to display and control the solver execution. - Overall project testing, ensuring that it meets the business requirements.

Useful Lessons Learnt:

- As the implementation of the project is using OptaPlanner, I am able to understand how to solve optimization problem in scheduling more efficiently using AI optimization algorithms, which include:
    - ✓ Reasoning system architecture using data-driven / reactive reasoning systems.
    - ✓ Knowledge Representation rules in duties allocation.
- As software development is new to me, I am able to pick up the frontend development of this project using REACT and source code control and management using github.
- I am able to optimize the database design for this project.
- Team coordination and communication.

Applying the knowledge and skills in workplace:

- Be able to provide the demonstration and consultation to the institution as part of a proposal solution to their problem statements.
- Designing the database is an important knowledge to acquire in this project as it will affect the efficiency of the process and algorithm.
- This project can be applied to different application. Especially in education sector, not only for exam allocation duty, there are many application such us time-tabling, technical staff laboratory allocation, etc.

Team Member 3: **NEOH SHI KANG (A0229965L)**

Personal Contribution to the group project:

- Setup software architecture (Backend, Frontend, Database).
- Implement whole backend, preference page.
- Integration with solver
- Feature engineering (scaling of time stamp)

Useful Lessons Learnt:

- Time management
- Focus on the right things and tasks
- Optaplanner library

Applying the knowledge and skills in workplace:

- Technical skills picked up and learnt through this project can be applied at work.
- Exposure to more tech stack and libraries helps me to make a better decision at work when I’m deciding on technical solutions

Team Member 4: **TAN WEE HAN (A0125244N)**
Personal Contribution to the group project: - Researched on optimization tools to identify a suitable tool for the project - Modelled the invigilator allocation problem into Optaplanner scheduling problem with domain objects and hard and soft constraints - Implemented domain objects and constraints in a Optaplanner solver service application - Tested installation of project on a blank virtual machine - Wrote installation and user guide
Useful Lessons Learnt: The most useful thing I learnt is how to model real life scheduling problems into optimization problems with constraints and use Optaplanner to develop solutions for optimization problems. The tool is simple to use but can solve complex problems even beyond scheduling.
Applying the knowledge and skills in workspace: I can use Optaplanner to develop solutions for scheduling problems in operations, to take into account of various constraints such as availability, preference, overloading/underloading, suitability and cost.