

Executive Report

PEDRO FERNANDES (1060503), ALEXANDRA LEITE (1170541), VITOR COSTA (1180511),
VITOR PEREIRA (1191244)

Índice

| | | |
|-------|---|----|
| 1 | Introduction | 4 |
| 1.1 | Project Overview | 4 |
| 1.2 | Objectives | 4 |
| 1.3 | Background | 5 |
| 1.4 | Methodology | 5 |
| 2 | Domain Model | 6 |
| 2.1 | Viva | 7 |
| 2.2 | Role | 7 |
| 2.3 | ResourceId | 7 |
| 2.4 | Resource | 7 |
| 2.5 | Availability..... | 8 |
| 2.6 | ScheduledViva..... | 8 |
| 2.7 | CompleteSchedule..... | 8 |
| 3 | Key Decisions & Justifications..... | 9 |
| 3.1 | Viva Model | 9 |
| 3.2 | Role Operations | 10 |
| 4 | Challenges & Solutions | 12 |
| 4.1 | Availability Constraints | 12 |
| 4.1.1 | intersectable..... | 13 |
| 4.2 | Preference Maximization..... | 13 |
| 4.3 | Conflict Minimization | 14 |
| 4.4 | Scalability..... | 14 |
| 5 | Milestones | 15 |
| 5.1 | Milestone 1: First Come First Served (FCFS) | 15 |
| 5.1.1 | Objective | 15 |
| 5.1.2 | Implementation Details | 15 |
| 5.1.3 | Tests..... | 16 |
| 5.1.4 | Challenges..... | 20 |
| 5.1.5 | Results and Validation..... | 20 |
| 5.2 | Milestone 2: Property-based Tests..... | 21 |
| 5.2.1 | Objective | 21 |
| 5.2.2 | Implementation Details | 21 |
| 5.2.3 | Property Based Tests | 21 |

| | | |
|-------|--|----|
| 5.2.4 | Bugs Found | 23 |
| 5.2.5 | The First-Come First Served (FCFS) Problem | 24 |
| 5.2.6 | Challenges..... | 25 |
| 5.2.7 | Results and Validation..... | 26 |
| 5.3 | Milestone 3: Preference Maximization | 26 |
| 5.3.1 | Objective | 26 |
| 5.3.2 | Implementation Details | 26 |
| 5.3.3 | Tests..... | 27 |
| 5.3.4 | Challenges..... | 29 |
| 5.3.5 | Results and Validation..... | 29 |
| 6 | Conclusion | 29 |

1 Introduction

1.1 Project Overview

The project involves the scheduling of MSc dissertation defenses, also known as viva. The objective is to create a scheduling system that can efficiently allocate time slots for viva based on various constraints and restrictions. This system aims to streamline the scheduling process, saving time and effort for students, teachers, and external participants, while ensuring optimal scheduling that considers availability and preferences.

The significance of this project lies in its potential to greatly improve the scheduling process for MSc dissertation defenses. By automating and optimizing the scheduling process, the system can reduce the administrative burden on academic staff and ensure that vivas are scheduled at times that are convenient for all participants. This not only enhances the efficiency of the scheduling process but also improves the overall experience for students and faculty members.

Overall, the project aims to deliver a comprehensive scheduling solution that addresses the complexities and constraints of scheduling MSc dissertation defenses, making the process smoother and more effective for all parties involved.

1.2 Objectives

The main objectives of this project are:

1. **Efficiency:** To streamline the scheduling process for MSc dissertation defenses, saving time and effort for all entities involved.
2. **Optimization:** To schedule viva at the most favorable times by considering the availability and preferences of the participants.
3. **Conflict Minimization:** To ensure that scheduling conflicts are minimized, particularly when the same resources (teachers or examiners) are required for multiple vivas.

4. **Scalability:** To handle the scheduling of multiple vivas within a given period, accommodating any overlapping requirements and constraints.
5. **Robustness:** To develop a reliable and flexible scheduling system that can adapt to various changes in availability and preferences.

1.3 Background

Scheduling dissertation defenses is a complex problem due to the multiple constraints and entities involved. Each viva needs to be scheduled at a time that suits all participants, which includes the student, their advisor, co-advisors, supervisors, and the president of the jury. Additionally, the problem is compounded when multiple viva need to be scheduled within a short timeframe, requiring careful consideration of overlapping availabilities.

1.4 Methodology

To address this problem, the project is divided into three main milestones:

1. Milestone 1: First Come First Served (FCFS):

- Develop a basic scheduling algorithm that assigns time slots to viva in a sequential manner based on the first available common slot for all required participants.

2. Milestone 2: Property-based Tests:

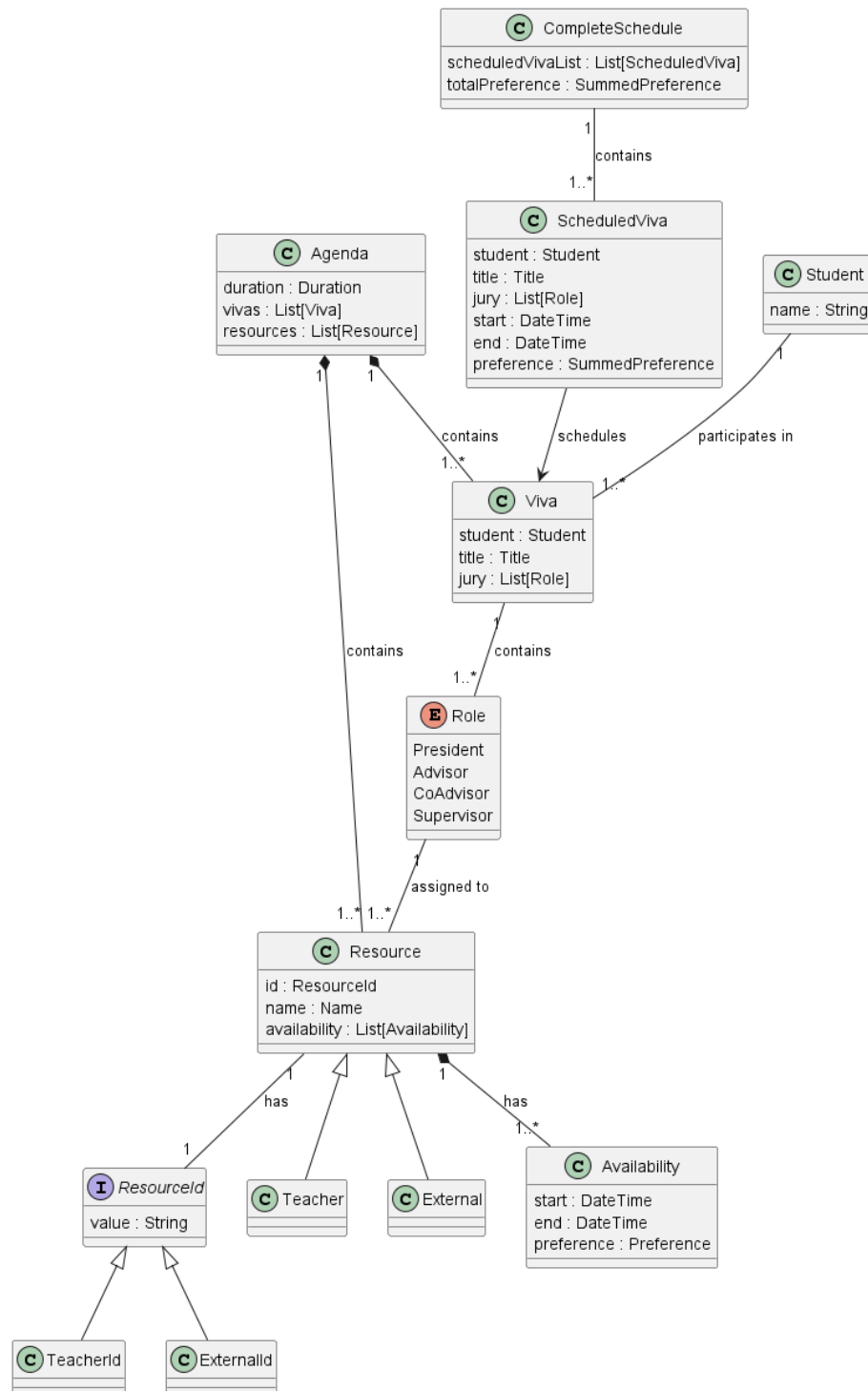
- Enhance the system by introducing property-based testing to ensure the correctness and robustness of the scheduling algorithm. This involves defining key properties that the scheduling system must satisfy and validating the system against these properties.

3. Milestone 3: Preference Maximization:

- Optimize the scheduling algorithm to maximize the preferences of all participants. This involves considering the preference scores of different time slots and selecting the schedule that offers the highest overall preference.

2 Domain Model

The domain model for the Viva scheduling system is designed around the concept of roles and their operations. The key entities in this model are:



2.1 Viva

A Viva instance represents an individual dissertation defense session. It contains information about the student presenting their dissertation, the title of the dissertation, and the members of the jury who will evaluate the defense. The jury members are represented by a list of Role instances.

2.2 Role

The Role enum defines the different roles that individuals can have during a dissertation defense. These roles include:

- **President:** Typically, the head of the examination panel.
- **Advisor:** The academic advisor who has guided the student throughout their dissertation.
- **CoAdvisor:** An additional advisor who may have provided support during the dissertation process.
- **Supervisor:** A supervisory role overseeing the dissertation defense process. Each role has an associated resource, which could be a teacher or an external examiner.

2.3 ResourceId

The ResourceId trait serves as the identifier for resources associated with roles. It is a common interface implemented by specific resource identifiers, such as TeacherId and ExternalId. These identifiers uniquely identify resources within the scheduling system.

2.4 Resource

The Resource trait represents the resources available for scheduling dissertation defenses. It includes information such as the resource identifier (ResourceId), the name of the resource (e.g., the name of a teacher or an external examiner), and the availability of the resource for scheduling (Availability). Resources are categorized

into subclasses Teacher and External, depending on whether they are internal faculty members or external examiners.

2.5 Availability

The Availability class represents the availability of resources for scheduling dissertation defenses. It contains details such as the start and end times during which the resource is available and the preference level associated with scheduling the resource for a particular defense. This allows the scheduling system to optimize the allocation of resources based on their availability and preferences.

2.6 ScheduledViva

A ScheduledViva instance represents a dissertation defense session that has been scheduled within the agenda. It includes details such as the student presenting their dissertation, the title of the dissertation, the members of the jury, the start and end times of the defense session, and the preference level associated with scheduling the defense.

2.7 CompleteSchedule

The CompleteSchedule class encapsulates the entire schedule of dissertation defenses, including all scheduled defense sessions (ScheduledViva instances) and the total preference level for the entire schedule. It provides a comprehensive view of the scheduled defenses and their associated preferences, allowing stakeholders to assess the overall scheduling efficiency and effectiveness.

3 Key Decisions & Justifications

3.1 Viva Model

- **Context and Problem Statement:** In the viva model should we use a field with a set of roles or have a president and advisor field.
- **Considered Options:**
 - Option 1

```
final case class Viva(student: Student,
                      title: Title,
                      president: Role.President,
                      advisor: Role.Advisor,
                      coAdvisors: List[Role.CoAdvisor],
                      superVisors: List[Role.Supervisor])
```

- Option 2

```
final case class Viva(
    student: Student,
    title: Title,
    jury: List[Role]
)
```

- **Constraints:**
 - Each teacher can have only one role.
 - The roles associated with the viva are:
 - one jury President (mandatory)
 - one Advisor (mandatory)
 - zero or more Co-advisors (optional)
 - zero or more Supervisors (optional)
- **Pros and Cons of the Options:**
 - Option 1

- Good, because there is no need to validate the rules for roles in a viva.
 - Bad, because there is need to validate that president != advisor != all CoAdvisors != all Supervisors
- Option 2
 - Good, because there is no need to validate that president != advisor != all CoAdvisors != all Supervisors
 - Bad, because there is need to validate if the Set has at least one element with role president and advisor
- **Decision Outcome:** Chosen option 2, because it's more flexible, and it's easier to check if a set has certain elements than validate equality across 4 fields.
- **Positive Consequences:**
 - Easier to implement.
 - Better for maintainability (in case new roles are required in the jury)
- **Negative Consequences:**
 - The viva model does not express the domain as much.

3.2 Role Operations

- **Context and Problem Statement:**
 - Where should the role operations be kept?
 - Companion object?
 - Module?
- **Considered Options:**
 - Option 1 - Module

```

trait RoleOps[Role]:
  def isPresident(role: Role): Boolean
  def isAdvisor(role: Role): Boolean

object RoleService extends RoleOps[Role]:
  def isPresident(role: Role): Boolean = role match
    case President(_) => true
    case _ => false
  def isAdvisor(role: Role): Boolean = role match
    case Advisor(_) => true
    case _ => false

```

- Option 2 - Companion Object

```

object Role:
  def isPresident(role: Role): Boolean = role match
    case President(_) => true
    case _ => false
  def isAdvisor(role: Role): Boolean = role match
    case Advisor(_) => true
    case _ => false

```

- **Pros and Cons of the Options:**

- Option 1
 - Good, because there is a clear separation between data model and its operations/behaviors.
 - Arguably Bad, because when creating a Viva, it depends on Role Service/Ops to validate the viva. This creates a coupling between the Viva model and the Role Operations. (Opposing argument: there is no problem with using operations from another "entity" in a given entity's operations.)
- Option 2
 - Bad, because it couples the data model with operations/behaviors.














- Good, because it's easier to find and work with.
- Arguably Good, because when creating a Viva, it depends on Role instead of RoleService. (Opposing argument: at the end of the day it does not matter because its importing operations/behaviors.)
- **Decision Outcome:** Chosen option 1, because we are aiming for clear separation between data model and its operations/behaviors.
- **Positive Consequences:** clear separation between data model and its operations/behaviours which is better for maintainability, testing and possible parallelization.
- **Negative Consequences:** n/a.

4 Challenges & Solutions

During the project, we encountered several challenges that required careful consideration and problem-solving. Here are some of the key challenges and their corresponding solutions:

4.1 Availability Constraints

Allen's Interval Algebra is a mathematical framework used to reason about and manipulate intervals of time. It provides a set of rules and operations that allow us to perform various operations on intervals, such as determining their intersection, checking if one interval is contained within another, and more.

| precedes | meets | overlaps | finished by | contains | starts | equals | started by | during | finishes | overlap-ped by | met by | preceded by |
|---|---|---|---|---|---|---|---|--|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| p | m | o | F | D | s | e | S | d | f | O | M | P |

In our project, Allen's Interval Algebra was a crucial tool for solving the challenge of determining the intersection of availabilities for scheduling dissertation defenses. By applying the principles of Allen's Interval Algebra, we were able to identify common

time slots where all required resources were available. This helped us streamline the scheduling process and ensure that all necessary resources were allocated efficiently.

4.1.1 intersectable

```
def intersectable(a: Availability, b: Availability): Boolean =  
    overlaps(a,b)    || overlaps(b, a)    ||  
    finishedBy(a, b) || finishedBy(b,a) ||  
    contains(a, b)   || contains(b, a)   ||  
    starts(a, b)     || starts(b, a)     ||  
    equals(a, b)
```

This method takes two *Availability* objects and a *Duration*. It checks if the two availabilities intersect and if the intersection is long enough to accommodate the given duration. It returns *true* if they do, *false* otherwise.

This method is associated with the "overlaps", "finishedBy", "contains", "starts" and "equals" relations, as it involves checking if one interval starts before and finishes after the start of another interval.

4.2 Preference Maximization

Preference maximization is a key challenge in the scheduling of viva, as it involves selecting the best possible time slots for all participants based on their preferences. The goal is to maximize the overall satisfaction of all involved parties by considering their preferred time slots.

1. Preference Calculation:

- Each availability period has a numeric preference value.
- The preference for a scheduled viva is the sum of the preference values of all involved resources for the chosen time slot.

2. Optimization Algorithm:

- Develop an algorithm that evaluates all possible time slots for each viva.
- Calculate the total preference score for each possible schedule.
- Select the schedule that maximizes the total preference score while ensuring no conflicts in resource allocation.

4.3 Conflict Minimization

Conflict minimization is essential to ensure that the scheduling system avoids double-booking resources and handles overlapping availabilities effectively.

1. Conflict Detection:

- Implement a method to check for conflicts by verifying if any resource is scheduled for more than one viva at the same time.
- Use interval algebra to detect overlapping intervals where a resource is already booked.

2. Conflict Resolution:

- Develop a conflict resolution strategy that adjusts the schedule to eliminate overlaps.
- Prioritize resolving conflicts by rescheduling viva with the least overall preference impact.

4.4 Scalability

Scalability is a crucial aspect of the scheduling system to ensure that it can handle an increasing number of viva and resources without a significant drop in performance or efficiency.

1. Efficient Algorithms:

- Develop efficient algorithms for scheduling, preference calculation, and conflict resolution that can handle large datasets.
- Utilize data structures that support quick lookups and updates to manage the schedule and availability of resources.

2. Incremental Scheduling:

- Develop an incremental scheduling approach that allows the system to handle new scheduling requests dynamically without re-computing the entire schedule.
- Use lazy evaluation techniques to defer computations until necessary, improving overall efficiency.

5 Milestones

The project is divided into three key milestones, each focusing on progressively enhancing the scheduling system's functionality, robustness, and optimization. This chapter outlines the objectives, implementation details, challenges, and results for each milestone.

5.1 Milestone 1: First Come First Served (FCFS)

5.1.1 Objective

Develop a minimum viable product (MVP) that schedules viva in a First Come First Served manner. The primary goal is to schedule each viva at the first available time slot where all required participants are free.

5.1.2 Implementation Details

- **Scheduling Algorithm:** Implemented a basic algorithm that iterates through the list of vivas and schedules them sequentially based on the earliest common availability of required participants.
- **Data Structures:** Utilized lists and sets to manage viva and resource availabilities.
- **Validation:** Ensured that all vivas are scheduled without conflicts in resource availability.

5.1.3 Tests

Domain Tests

Validate the core domain entities and operations, such as viva, role, resource, and availability. They ensure that the domain model is correctly implemented and that the entities behave as expected.

- **Viva Tests:** These tests would validate the creation and manipulation of viva instances. This could involve creating viva objects with different attributes, updating viva attributes, and verifying that the viva objects are correctly constructed.
- **Role Tests:** These tests would validate the creation and manipulation of role instances. This could involve creating role objects with different attributes, updating role attributes, and verifying that the role objects are correctly constructed.
- **Resource Tests:** These tests would validate the creation and manipulation of resource instances. This could involve creating resource objects with different attributes, updating resource attributes, and verifying that the resource objects are correctly constructed.
- **Availability Tests:** These tests would validate the creation and manipulation of availability instances. This could involve creating availability objects with different attributes, updating availability attributes, and verifying that the availability objects are correctly constructed.
- **Scheduled Viva Tests:** These tests would validate the creation and manipulation of scheduled viva instances. This could involve creating scheduled viva objects with different attributes, updating scheduled viva attributes, and verifying that the scheduled viva objects are correctly constructed.
- **Complete Schedule Tests:** These tests would validate the creation and manipulation of complete schedule instances. This could involve creating complete schedule objects with different attributes, updating complete

schedule attributes, and verifying that the complete schedule objects are correctly constructed.

- **Teacher Tests:** These tests would validate the creation and manipulation of teacher instances. This could involve creating teacher objects with different attributes, updating teacher attributes, and verifying that the teacher objects are correctly constructed.

SimpleType Tests

The simple type tests validate the simple types used in the domain model, such as *DateTime* and *Duration*. They ensure that these simple types are correctly implemented and behave as expected.

- **DateTime Tests:** These tests would validate the creation and manipulation of *DateTime* instances. This could involve creating *DateTime* objects with different attributes, updating *DateTime* attributes, and verifying that the *DateTime* objects are correctly constructed.
- **Duration Tests:** These tests would validate the creation and manipulation of *Duration* instances. This could involve creating *Duration* objects with different attributes, updating *Duration* attributes, and verifying that the *Duration* objects are correctly constructed.
- **Preference Tests:** These tests would validate the creation and manipulation of *Preference* instances. This could involve creating *Preference* objects with different attributes, updating *Preference* attributes, and verifying that the *Preference* objects are correctly constructed.
- **SummedPreference Tests:** These tests would validate the creation and manipulation of *SummedPreference* instances. This could involve creating *SummedPreference* objects with different attributes, updating *SummedPreference* attributes, and verifying that the *SummedPreference* objects are correctly constructed.
- **Student Tests:** These tests would validate the creation and manipulation of *Student* instances. This could involve creating *Student* objects with different

attributes, updating *Student* attributes, and verifying that the *Student* objects are correctly constructed.

- **Title Tests:** These tests would validate the creation and manipulation of *Title* instances. This could involve creating *Title* objects with different attributes, updating *Title* attributes, and verifying that the *Title* objects are correctly constructed.
- **Name Tests:** These tests would validate the creation and manipulation of *Name* instances. This could involve creating *Name* objects with different attributes, updating *Name* attributes, and verifying that the *Name* objects are correctly constructed.

Operation Tests

The operations tests validate the scheduling operations, such as scheduling a viva, updating resource availabilities, and calculating preferences. They test the scheduling algorithms and functions to ensure that they allocate time slots correctly and handle various constraints and restrictions.

- **Scheduling a Viva:** These tests would validate that the system can correctly schedule a viva. This could involve creating a viva with specific resources and availabilities, running the scheduling operation, and then verifying that the viva has been scheduled at the correct time with the correct resources.
- **Updating Resource Availabilities:** These tests would check that the system can correctly update the availability of resources. This could involve creating a resource with a specific availability, running the update operation with a new availability, and then verifying that the resource's availability has been updated correctly.
- **Calculating Preferences:** These tests would validate that the system can correctly calculate preferences for scheduling a viva. This could involve creating a viva with specific resources and preferences, running the preference calculation operation, and then verifying that the calculated preferences match the expected values.

Functional Tests

These tests validate the overall functionality of the scheduling system by testing the integration of various components and operations. They ensure that the system behaves as expected.

These tests utilize files of input and output created to validate the that the system is working correctly.

By running the tests in the file *ScheduleMS01Test* it shows the tests that pass.

```
File: simple01_in.xml Result: OK
File: simple02_in.xml Result: OK
File: simple03_in.xml Result: OK
File: simple04_in.xml Result: OK
File: simple05_in.xml Result: OK
Final score of Milestone 1: 5 / 5 = 100
```

Assessment Tests

These are tests given by the professor to assess the correctness of the scheduling system. It was given input files for us to tests the scheduling system and then compare to the given output files.

By running the tests in *AssessmentTestMS01* we can verify that they all pass, assuring that the scheduling system is working fine.

```
File: invalid_agenda_01_in.xml Result: OK
File: invalid_agenda_02_in.xml Result: OK
File: invalid_agenda_03_in.xml Result: OK
File: invalid_agenda_04_in.xml Result: OK
File: invalid_agenda_21_in.xml Result: OK
File: invalid_preference_in.xml Result: OK
File: missing_advisor_in.xml Result: OK
File: missing_president_in.xml Result: OK
File: valid_agenda_01_in.xml Result: OK
File: valid_agenda_02_in.xml Result: OK
File: valid_agenda_03_in.xml Result: OK
File: valid_agenda_04_in.xml Result: OK
File: valid_agenda_05_in.xml Result: OK
File: valid_agenda_06_in.xml Result: OK
File: valid_agenda_22_in.xml Result: OK
File: valid_agenda_23_in.xml Result: OK
File: valid_agenda_24_in.xml Result: OK
File: valid_agenda_25_in.xml Result: OK
File: valid_agenda_26_in.xml Result: OK
File: valid_agenda_27_in.xml Result: OK
Final score of Milestone 1: 20 / 20 = 100
```

5.1.4 Challenges

- Handling the constraints of resource availability.
- Ensuring that no two vivas overlap in terms of resource allocation.

5.1.5 Results and Validation

- Successfully scheduled all viva in a First Come First Served manner.
- Created unit and functional tests to validate the correctness of the scheduling algorithm.

5.2 Milestone 2: Property-based Tests

5.2.1 Objective

Enhance the scheduling system by introducing property-based testing. These tests validate the scheduling algorithm against a set of predefined properties to ensure robustness and correctness.

5.2.2 Implementation Details

- **Property Definitions:** Defined key properties that the scheduling system must satisfy, such as ensuring no resource is double-booked and all vivas are scheduled within available time slots.
- **Testing Framework:** Utilized ScalaCheck to automatically generate test cases and validate the scheduling algorithm.
- **Integration:** Integrated property-based tests with the existing scheduling algorithm to catch edge cases and potential failures.

5.2.3 Property Based Tests

Property-based testing is a testing methodology that involves defining properties that the system should satisfy and then generating random inputs to test these properties. This approach can help uncover edge cases and corner cases that may not be covered by traditional unit tests.

In our project we created generators for the domain entities and properties to test the scheduling system.

We have generators and property tests for the SimpleTypes, such as DateTime and Duration, and for the domain entities, such as Viva, Role, Resource, and Availability.

Here's a brief explanation of each property test:

- **DateTime Properties:** These properties test the behavior of the *DateTime* class, such as checking that the addition and subtraction of durations is consistent and that the ordering of *DateTime* instances is correct.
- **Duration Properties:** These properties test the behavior of the *Duration* class, such as checking that the addition and subtraction of durations is consistent and that the ordering of *Duration* instances is correct.
- **Viva Properties:** These properties test the behavior of the *Viva* class, such as checking that the creation and manipulation of viva instances is consistent and that the properties of viva instances are correctly constructed.
- **Role Properties:** These properties test the behavior of the *Role* class, such as checking that the creation and manipulation of role instances is consistent and that the properties of role instances are correctly constructed.
- **Resource Properties:** These properties test the behavior of the *Resource* class, such as checking that the creation and manipulation of resource instances is consistent and that the properties of resource instances are correctly constructed.
- **Availability Properties:** These properties test the behavior of the *Availability* class, such as checking that the creation and manipulation of availability instances is consistent and that the properties of availability instances are correctly constructed.

Then we have in *ScheduleVivaServiceProperties* the property tests for the scheduling system. The class contains several generator functions that generate random instances of various domain entities. These generators are used to provide random inputs for the property tests.

The *property("Schedule agenda")* function is a property test that validates the *scheduleVivaFromAgenda* function in the *ScheduleVivaService* class. It uses the *schedulableAgendaGen* generator to generate random Agenda instances and checks that the *scheduleVivaFromAgenda* function can successfully schedule all vivas in the agenda.

In summary, the *ScheduleVivaServiceProperties* class provides a set of property-based tests for the *ScheduleVivaService* class. It helps ensure that the scheduling system can correctly schedule viva sessions under various conditions and constraints.

Below is an image showing the results of the test execution:

```
[info] + SimpleTypesProperties.All duration must have an hour between 0 and 23 and minutes between 0 and 59: OK, passed 100 tests.
[info] + DomainProperties.TeacherIdGen always generates valid teacherId: OK, passed 100 tests.
[info] + SimpleTypesProperties.Duration hour and minute must always be two digits: OK, passed 100 tests.
[info] + DomainProperties.ExternalIdGen always generate valid externalId: OK, passed 100 tests.
[info] + DomainProperties.All availabilities must be valid: OK, passed 100 tests.
[info] + PreferencesGenerators.PreferencesService.sumPreferences: OK, passed 100 tests.
[info] + PreferencesGenerators.PreferencesService.sumSummedPreferences: OK, passed 100 tests.
[info] + AvailabilityServiceProperties.removeInterval: OK, passed 100 tests.
[info] + SimpleTypesProperties.SummedPreference must be greater or equal than 1: OK, passed 100 tests.
[info] + SimpleTypesProperties.Name must be characters non empty: OK, passed 100 tests.
[info] + SimpleTypesProperties.Student must be Student + numbers: OK, passed 100 tests.
[info] + SimpleTypesProperties.All date times must have valid dates and times: OK, passed 100 tests.
[info] + SimpleTypesProperties.Preference must be between 1 and 5: OK, passed 100 tests.
[info] + SimpleTypesProperties.Title must be characters + numbers: OK, passed 100 tests.
[info] + DomainProperties.All externals must have a id, name and at least one availability.: OK, passed 100 tests.
[info] + DomainProperties.All teachers must have a id, name and at least one availability.: OK, passed 100 tests.
[info] + ScheduleVivaServiceProperties.Schedule agenda: OK, passed 100 tests.
[info] + DomainProperties.Resources are unique and always have at least two teachers: OK, passed 100 tests.
[info] + DomainProperties.RoleGen must generate valid jury: OK, passed 100 tests.
[info] + DomainProperties.VivaGen must generate valid vivas: OK, passed 100 tests.
[info] + DomainProperties.AgendaGen must generate valid agendas: OK, passed 100 tests.
[info] + PreferencesGenerators.PreferencesService.sumPreferencesOfScheduledVivas: OK, passed 100 tests.
[info] Passed: Total 22, Failed 0, Errors 0, Passed 22
[success] Total time: 57 s, completed 19/05/2024, 12:04:51
[IJ]
```

5.2.4 Bugs Found

The calculation of the duration between 2 date times produced wrong output when the days of the date times are different.

```
def fromBetween(start: DateTime, end: DateTime): Result[Duration] =
  val startMinutes = start.getHour * 60 + start.getMinute
  val endMinutes = end.getHour * 60 + end.getMinute
  val durationMinutes = endMinutes - startMinutes
  from(durationMinutes / 60, durationMinutes % 60)
```

The faulty function above was replaced with the following:

```
def fromBetween(start: DateTime, end: DateTime): Result[Duration] =
  Try(java.time.Duration.between(start, end))
    .fold(
      error => Left(InvalidDuration(start.toString + "between" + end.toString)),
      duration =>
        Try(LocalTime.of(duration.toHours.toInt, (duration.toMinutes % 60).toInt))
          .fold(
            error => Left(InvalidDuration(duration.toString)),
            success => Right(success)
          )
    )
)
```

5.2.5 The First-Come First Served (FCFS) Problem

The property-based test for the scheduling algorithm fails because it finds an example where the scheduling fails because of the FCFS nature of the algorithm.

For instance, assume an example agenda with a 09:40 duration requirement and with the following vivas:

Viva 1: Student999

Participants and their availabilities:

- Supervisor (ExternalId E001)
 - 9999-01-17T00:24:38 to 9999-01-17T10:04:38
 - 9999-12-01T01:44:41 to 9999-12-01T11:24:41
- President (TeacherId T000)
 - 9999-01-17T00:24:38 to 9999-01-17T10:04:38
 - 9999-12-01T01:44:41 to 9999-12-01T11:24:41
- Advisor (TeacherId T001)
 - 9999-01-17T00:24:38 to 9999-01-17T10:04:38
 - 9999-12-01T01:44:41 to 9999-12-01T11:24:41

Viva 2: Student254

Participants and their availabilities:

- Supervisor (ExternalId E001)
 - 9999-01-17T00:24:38 to 9999-01-17T10:04:38
 - 9999-12-01T01:44:41 to 9999-12-01T11:24:41
- CoAdvisor (ExternalId E000)
 - 9999 -01-17T00:24:38 to 9999-01-17T10:04:38
- President (TeacherId T000)
 - 9999-01- 17T00:24:38 to 9999-01-17T10:04:38
 - 9999- 12-01T01:44:41 to 9999-12-01T11:24:41
- Advisor (TeacherId T001)
 - 9999 -01-17T00:24:38 to 9999-01-17T10:04:38
 - 9999 -12-01T01:44:41 to 9999-12-01T11:24:41

Availability intersections

- Viva 1
 - **Participants:** Supervisor (E001), President (T000), Advisor (T001)
 - **FCFS Common Time Slot chosen:** 9999-01-17T00:24:38 to 9999-01-17T10:04:38
- Viva 2
 - **Participants:** Supervisor (E001), CoAdvisor (E000), President (T000), Advisor (T001)
 - **FCFS Common Time Slot:** FAILS

The schedule fails because viva 1 chose the first intersection it found (9999-01-17T00:24:38 to 9999-01-17T10:04:38), which is the only time that would work for viva 2. If viva 1 chose the second time slot that works instead of the first one (9999-12-01T01:44:41 to 9999-12-01T11:24:41), the agenda would be schedulable.

5.2.6 Challenges

- Defining comprehensive properties that cover various scenarios and constraints.

- Ensuring the generated test cases are diverse and representative of real-world scheduling challenges.

5.2.7 Results and Validation

- Successfully implemented property-based tests to validate the scheduling system.
- Ensured that the system adheres to the specified constraints and performs reliably under various conditions.

5.3 Milestone 3: Preference Maximization

5.3.1 Objective

Optimize the scheduling algorithm to maximize the preferences of all participants. This involves considering the preferences of resources when allocating time slots to ensure the most favorable schedule for all parties involved.

5.3.2 Implementation Details

- **Preference Calculation:** Developed methods to calculate the preference scores for different time slots based on resource availabilities and preferences.
- **Optimization Algorithm:** Tried two different approaches to maximize preferences. After evaluating both approaches, we opted for a kind of Brute Force approach due to its ability to guarantee the highest possible preference score.

Graph Approach

- **Graph Representation:** Each node is represented as a viva and candidate availability pair, and edges between nodes indicate that there are no scheduling constraints or conflicts.

```

object ScheduleVivaServiceMS036Graph:

  final case class Node(viva: Viva, candidateAvailability: Availability, summedPreference: SummedPreference)
  final case class Edge(node: Node, preference: SummedPreference)

  def scheduleConflictingVivas(vivas: List[Viva], resources: List[Resource], duration: Duration): List[ScheduledViva] =
    val candidates: List[(Viva, List[(Availability, SummedPreference)])] =
      | vivas.map(viva => (viva, findAllCandidateSchedules(viva, resources, duration)))

    val graph = constructGraph(candidates, resources, duration)
    val sortedNodes = topologicalSort(graph)
    longestPath(graph, sortedNodes)

```

- **Graph Traversal:** Implemented a longest path algorithm to traverse the graph and find an optimal scheduling path that maximizes preferences while minimizing conflicts.
- **Preference Maximization:** Calculated preference scores during graph traversal to select the best scheduling path.

Brute Force Heuristic Approach

- **Exhaustive Search:** Implemented a brute force algorithm to explore a configurable number of possible scheduling combinations.
- **Preference Calculation:** Evaluated each combination based on the total preference score, selecting the combination with the highest score.
- **Optimization and Efficiency:** Although computationally intensive, this approach ensures the best possible schedule is found by considering a configurable number of potential solutions.

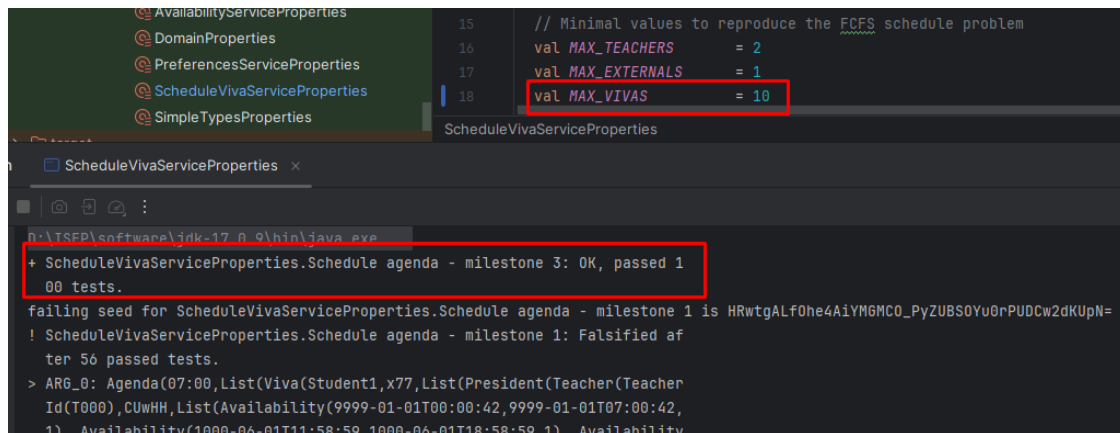
5.3.3 Tests

The test suite for Milestone 3 included 50 distinct XML files (47 files given by the teachers and 3 created by us) representing different scheduling scenarios. These files contained a variety of viva configurations, resource availabilities, and preference settings to thoroughly test the algorithm's ability to maximize preferences and avoid conflicts.

```
File: group_valid_agenda_01_in.xml Result: OK
File: group_valid_agenda_02_in.xml Result: OK
File: group_valid_agenda_03_in.xml Result: OK
Final score of Milestone 3: 3 / 3 = 100
```

```
File: valid_agenda_01_in.xml Result: OK
File: valid_agenda_02_in.xml Result: OK
File: valid_agenda_03_in.xml Result: OK
File: valid_agenda_04b_in.xml Result: OK
File: valid_agenda_04_in.xml Result: OK
File: valid_agenda_05_in.xml Result: OK
File: valid_agenda_06_in.xml Result: OK
File: valid_agenda_07_in.xml Result: OK
File: valid_agenda_08_in.xml Result: OK
File: valid_agenda_09_in.xml Result: OK
File: valid_agenda_10_in.xml Result: OK
File: valid_agenda_11_in.xml Result: OK
File: valid_agenda_12_in.xml Result: OK
File: valid_agenda_13_in.xml Result: OK
File: valid_agenda_14_in.xml Result: OK
File: valid_agenda_15_in.xml Result: OK
File: valid_agenda_16_in.xml Result: OK
File: valid_agenda_17_in.xml Result: OK
File: valid_agenda_18_in.xml Result: OK
File: valid_agenda_19b_in.xml Result: OK
File: valid_agenda_19_in.xml Result: OK
File: valid_agenda_20_in.xml Result: OK
File: valid_agenda_21_in.xml Result: OK
File: valid_agenda_22_in.xml Result: OK
File: valid_agenda_23_in.xml Result: OK
File: valid_agenda_24b_in.xml Result: OK
File: valid_agenda_24_in.xml Result: OK
File: valid_agenda_25_in.xml Result: OK
File: valid_agenda_26_in.xml Result: OK
File: valid_agenda_27_in.xml Result: OK
File: valid_agenda_28b_in.xml Result: OK
File: valid_agenda_28_in.xml Result: OK
File: valid_agenda_29_in.xml Result: OK
File: valid_agenda_30_in.xml Result: OK
File: valid_agenda_31_in.xml Result: OK
File: valid_agenda_32_in.xml Result: OK
File: valid_agenda_33b_in.xml Result: OK
File: valid_agenda_33_in.xml Result: OK
File: valid_agenda_34_in.xml Result: OK
File: valid_agenda_35_in.xml Result: OK
File: valid_agenda_36b_in.xml Result: OK
File: valid_agenda_36_in.xml Result: OK
File: valid_agenda_37_in.xml Result: OK
File: valid_agenda_38b_in.xml Result: OK
File: valid_agenda_38_in.xml Result: OK
File: valid_agenda_39_in.xml Result: OK
File: valid_agenda_40_in.xml Result: OK
Final score of Milestone 3: 47 / 47 = 100
```

A new property-based test was also created for Milestone 3 to validate the schedule's adherence to preferences and conflict resolution. This test, defined in *ScheduleVivaServiceProperties*, ensures that the scheduling algorithm meets the predefined properties for preference maximization and conflict avoidance.



```
AvailabilityServiceProperties
DomainProperties
PreferencesServiceProperties
ScheduleVivaServiceProperties
SimpleTypesProperties

15 // Minimal values to reproduce the FCFS schedule problem
16 val MAX_TEACHERS = 2
17 val MAX_EXTERNALS = 1
18 val MAX_VIVAS = 10

ScheduleVivaServiceProperties

ScheduleVivaServiceProperties x
+ ScheduleVivaServiceProperties.Schedule agenda - milestone 3: OK, passed 100 tests.
failing seed for ScheduleVivaServiceProperties.Schedule agenda - milestone 1 is HRwtgALf0he4AiYMGMC0_PyZUBSOYU0rPUDCw2dKUPN=
! ScheduleVivaServiceProperties.Schedule agenda - milestone 1: Falsified after 56 passed tests.
> ARG_0: Agenda(07:00,List(Viva(Student1,x77,List(President(Teacher(TeacherId(T000),CUWHH,List(Availability(9999-01-01T00:00:42,9999-01-01T07:00:42,1) Availability(1000-06-01T11:58:59,1000-06-01T18:58:59,1) Availability
```

5.3.4 Challenges

- Balancing individual viva preferences with the overall schedule optimization.
- Ensuring that the preference-based scheduling does not introduce conflicts or unresolvable constraints.

5.3.5 Results and Validation

- Successfully optimized the scheduling algorithm to maximize participant preferences.
- Developed additional tests to validate the preference maximization logic.
- Improved the overall satisfaction of participants by considering their preferred time slots.

6 Conclusion

The project has successfully achieved its objectives through the completion of three key milestones, each enhancing the scheduling system's functionality, robustness, and optimization for MSc dissertation defenses.

The following summarizes the journey and accomplishments of the project:

Milestone 1: First Come First Served (FCFS)

The first milestone established a foundational scheduling algorithm based on the First Come First Served principle. This initial implementation effectively handled the basic requirements of scheduling viva, ensuring that all sessions were allocated without conflicts in resource availability. The development of unit and functional tests provided a robust validation framework to confirm the algorithm's correctness.

Milestone 2: Property-based Tests

Building on the initial implementation, the second milestone introduced property-based testing to enhance the reliability and robustness of the scheduling system. By defining comprehensive properties and using automated test case generation, the project was able to uncover edge cases and ensure that the scheduling algorithm adhered to all constraints. This milestone highlighted the importance of rigorous testing in developing a reliable scheduling system.

Milestone 3: Preference Maximization

The third milestone focused on optimizing the scheduling algorithm to maximize participant preferences. Two approaches were explored: a graph-based approach and a brute force approach. After careful evaluation, the brute force approach was selected for its ability to guarantee the highest possible preference scores. This milestone faced significant challenges in balancing individual preferences and avoiding conflicts, but ultimately succeeded in creating a highly optimized scheduling solution. The comprehensive testing with 50 distinct scenarios validated the effectiveness of the algorithm, resulting in a perfect score of 47/47 in the test cases.