POLITECNICO
MILANO 1863

Automation and Control Engineering

Design Document

Project of Software Engineering (for Automation)
Professors: Rossi Matteo Giovanni, Lestingi Livia

SchedulEx

26/07/2023

Authors:
Petulicchio Lorenzo (10639923) – Talacci Mattia (10647582)

Summary

1 - INTRODUCTION

This document constitutes the Design Document (DD). Its purpose is to analyse the design of the software.

It is divided in 5 sections:

- 'Introduction' to give a generic document overview.
- 'Components Breakdown' where each single component of the entire system is described.
- 'Components Tasks' in which for every component are discussed their own operations.
- 'Components sequence diagrams', in this chapter are presented the sequence diagrams of the component interactions.
- 'Simplification for code' to list all the simplifications adopted to have easy code to execute.

Design overview

The system adopted is a distributed one, the choice is due to the interest in creating a project that can be as similar as possible to a real case. Is in fact probably to have a high-performance computer that host the back-end optimization algorithm, due to high computation demanding, while the secretary office has access to low budget PCs. Moreover, it is considered to have an existing database to interface with provided by PoliMi, in which are stored all the exam for each course.

The components of the entire system are:

- User interface (front-end)
- Back-end + optimization algorithm
- Databases

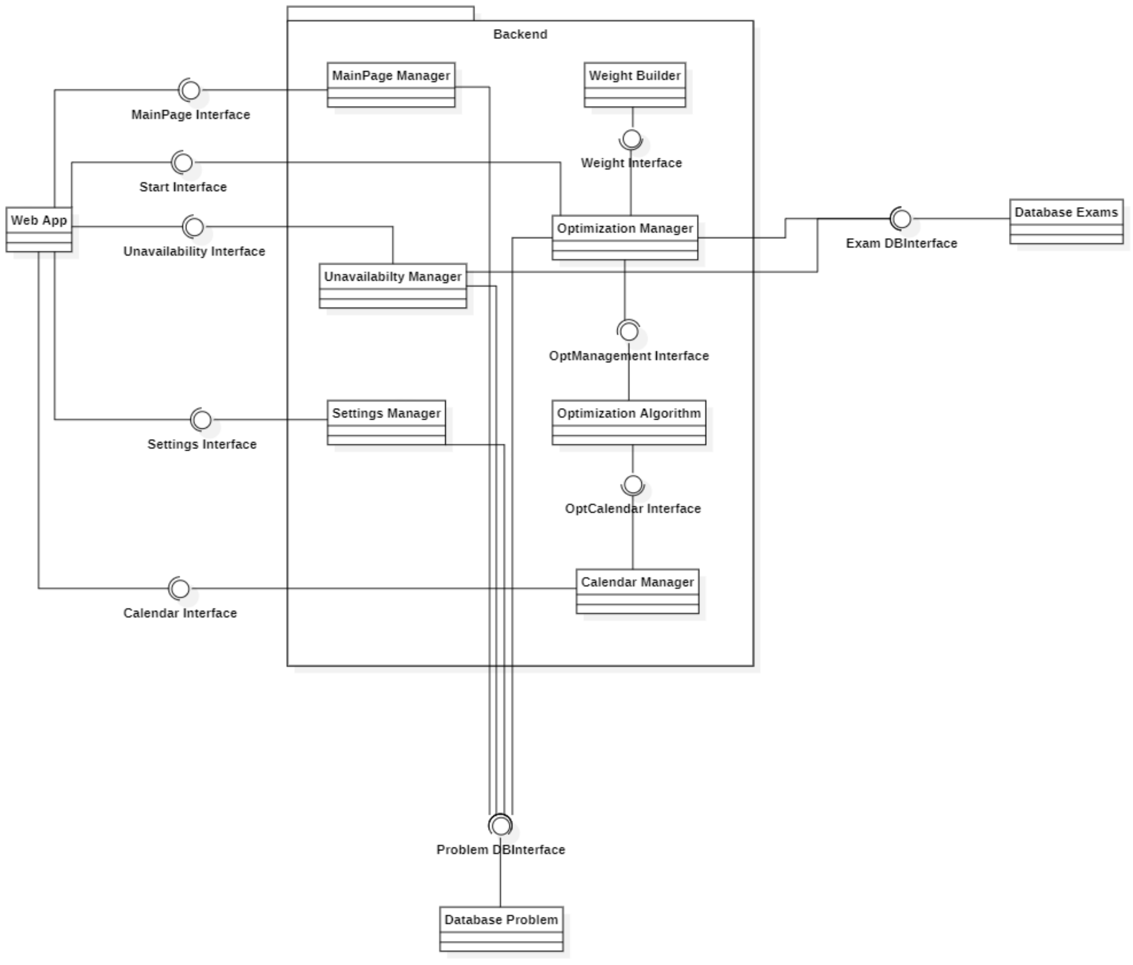
2 - COMPONENTS BREAKDOWN

In this chapter are explained all the components divided in packages (front-end, back-end and databases), how they are structured, which interface each component provides and use (with a description of it) and the object with which the component interacts.

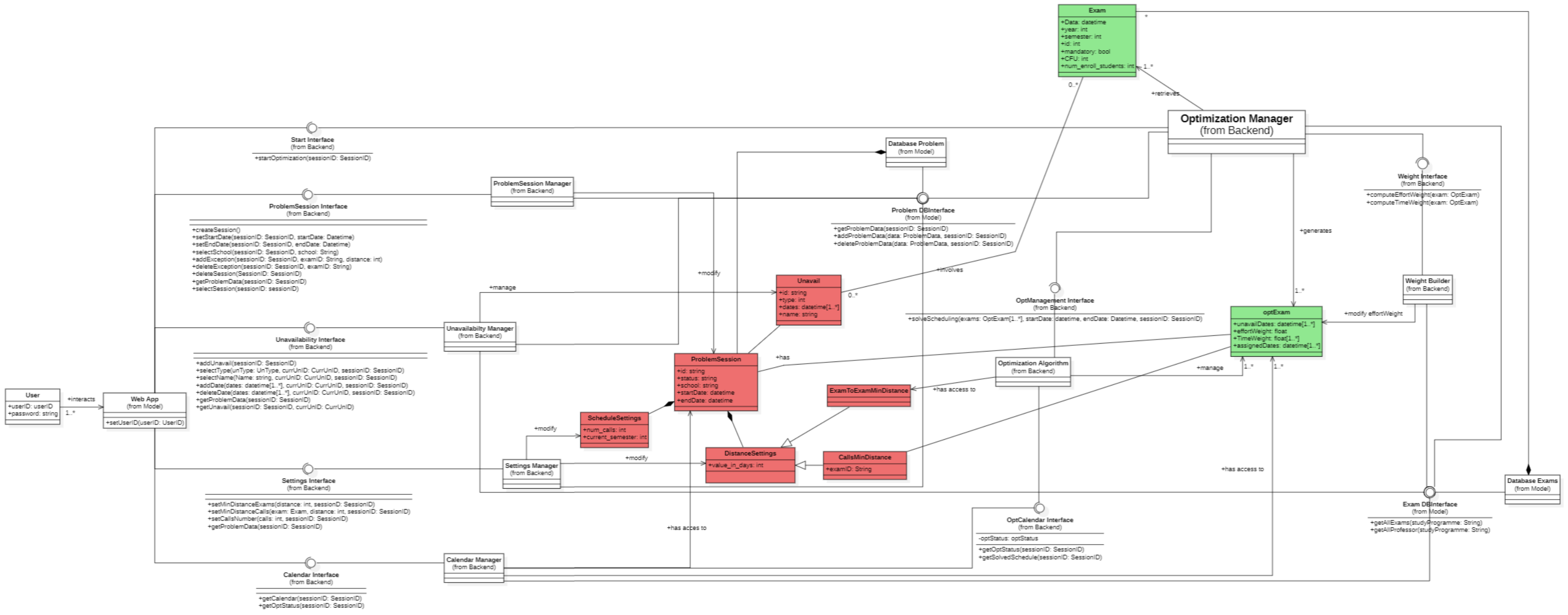
Overview

Now, it is presented the entire components diagram in which it is possible to find on the left the webapp component that communicates with the backend, in particular with its component on the left. These last ones communicate with a database that represents the link with the back-end components on the right of the Backend package. Instead, the other database is connected only with these ones. Moreover, there is only a connection between the webapp and the right components of the backend, this is realized with the main of the interfaces that allow the system to start the optimization.

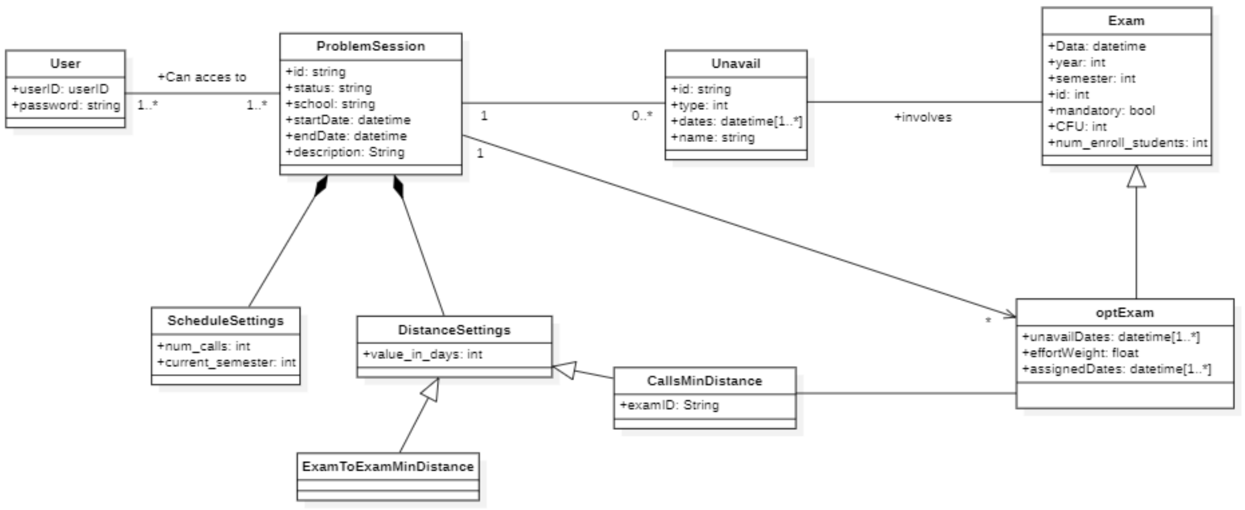
Inside the Backend package, the division in left and right component is done to visualize better the different purpose of the components, on the left there are components that work directly with the backend separately from each other (the common components are the webapp and the Problem Database). While the other components work for the exam scheduling and its optimization, and they are linked among them.



General component scheme


User
+userID: userID
+password: string
+interacts
1..*
Web App
(from Model)
+setUserID(userID: UserID)
Start Interface
(from Backend)
+startOptimization(sessionID: SessionID)
ProblemSession Interface
(from Backend)
+createSession()
+setStartDate(sessionID: SessionID, startDate: Datetime)
+setEndDate(sessionID: SessionID, endDate: Datetime)
+selectSchool(sessionID: SessionID, school: String)
+addException(sessionID: SessionID, examID: String, distance: int)
+deleteException(sessionID: SessionID, examID: String)
+deleteSession(SessionID: SessionID)
+getProblemData(sessionID: SessionID)
+selectSession(sessionID: sessionID)
Unavailability Interface
(from Backend)
+addUnavail(sessionID: SessionID)
+selectType(unType: UnType, currUnID: CurrUnID, sessionID: SessionID)
+selectName(Name: string, currUnID: CurrUnID, sessionID: SessionID)
+addDate(dates: datetime[1..*], currUnID: CurrUnID, sessionID: SessionID)
+deleteDate(dates: datetime[1..*], currUnID: CurrUnID, sessionID: SessionID)
+getProblemData(sessionID: SessionID)
+getUnavail(sessionID: SessionID, currUnID: CurrUnID)
Settings Interface
(from Backend)
+setMinDistanceExams(distance: int, sessionD: SessionID)
+setMinDistanceCalls(exam: Exam, distance: int, sessionID: SessionID)
+setCallsNumber(calls: int, sessionID: SessionID)
+getProblemData(sessionID: SessionID)
Calendar Interface
(from Backend)
+getCalendar(sessionID: SessionID)
+getOptStatus(sessionID: SessionID)
ProblemSession Manager
(from Backend)
Unavailabilty Manager
(from Backend)
Settings Manager
(from Backend)
Calendar Manager
(from Backend)
+manage
+modify
+modify
+modify
+has acces to
ScheduleSettings
+num_calls: int
+current_semester: int
ProblemSession
+id: string
+status: string
+school: string
+startDate: datetime
+endDate: datetime
Unavail
+id: string
+type: int
+dates: datetime[1..*]
+name: string
0..*
+has
DistanceSettings
+value_in_days: int
ExamToExamMinDistance
CallsMinDistance
+examID: String
Database Problem
(from Model)
Problem DBInterface
(from Model)
+getProblemData(sessionID: SessionID)
+addProblemData(data: ProblemData, sessionID: SessionID)
+deleteProblemData(data: ProblemData, sessionID: SessionID)
+involves
Exam
+Data: datetime
+year: int
+semester: int
+id: int
+mandatory: bool
+CFU: int
+num_enroll_students: int
0..*
1..*
*
+retrieves
Optimization Manager
(from Backend)
OptManagement Interface
(from Backend)
+solveScheduling(exams: OptExam[1..*], startDate: datetime, endDate: Datetime, sessionID: SessionID)
Optimization Algorithm
(from Backend)
+has access to
+manage
OptCalendar Interface
(from Backend)
-optStatus: optStatus
+getOptStatus(sessionID: SessionID)
+getSolvedSchedule(sessionID: SessionID)
+generates
1..*
optExam
+unavailDates: datetime[1..*]
+effortWeight: float
+TimeWeight: float[1..*]
+assignedDates: datetime[1..*]
1..*
1..*
+has access to
Weight Interface
(from Backend)
+computeEffortWeight(exam: OptExam)
+computeTimeWeight(exam: OptExam)
Weight Builder
(from Backend)
+modify effortWeight
Exam DBInterface
(from Model)
+getAllExams(studyProgramme: String)
+getAllProfessor(studyProgramme: String)
Database Exams
(from Model)

This UML diagram could be divided in two section, user input related functionality, and the scheduling related one's. This is highlight by the colours of classes involved in this functionality. Red classes are the one to which the user has access to it through the respective components while in green are depicted the classes that are managed by scheduling process components.



Front end

The front-end package contains only the webapp component because it just provides a connection with all logics stored in the back end.

It is structured in five pages, the first one is a login page in which the user must insert his username, after this he can enter in the second page where he has the possibility to choose the session to modify or use, alternatively he can create a new session. The term session refers to the set of entries/settings that can be entered by the user to perform optimisation, a more detailed explanation of the session object is given after ('ProblemSession'). The third page is the one in which can be entered, in sequence, the start and the end date of the exam session, the unavailability of the professors or university and the settings that are the minimum distance between calls of the same exam or the minimum distance between the calls of different exams. When the user inserts the unavailability, he accesses to the fourth page in which he can selects if the unavailability regards the university or a professor (choosing him) and the unavailability dates. This selection can be done in three different methods: selecting a single day, a day for a period or a period. The last page regards the settings, here the user should specify the settings regarding the minimum distances between calls of a specific exam and the minimum distance between exams. After all these actions and savings, the user can start the optimization process.

The webapp component doesn't provide any interface but uses the following interfaces that will be explained in the next chapter:

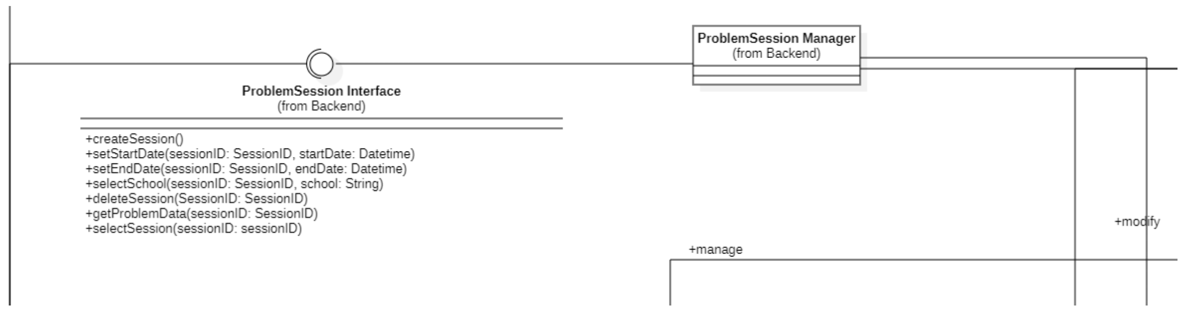
- Start Interface
- ProblemSession Interface
- Unavailability Interface
- Settings Interface
- Calendar Interface

Moreover, the webapp doesn't work with any object calls some back-end components that will modify them with the data inserted by the user.

Back end

The back-end package is the core of the project because its components manage the entire system. They are ProblemSession Manager, Unavailability Manager, Settings Manager, Weight Builder, Optimization Manager, Optimization Algorithm and Calendar Manager.

The following part provides a description for each component and show the interface that it provides.


ProblemSession Manager
(from Backend)
ProblemSession Interface
(from Backend)
+createSession()
+setStartDate(sessionID: SessionID, startDate: Datetime)
+setEndDate(sessionID: SessionID, endDate: Datetime)
+selectSchool(sessionID: SessionID, school: String)
+deleteSession(SessionID: SessionID)
+getProblemData(sessionID: SessionID)
+selectSession(sessionID: sessionID)
+modify
+manage

`ProblemSession Manager`

This component collect data regarding the start and end date from webapp, then save them in the proper database.

- `ProblemSession Interface`
  - `createSession()`
    This function creates the session ID to understand in which session the other input value (start date) must be saved in the Database Problem.
  - `setStartDate(sessionID: SessionID, startDate: Datetime)`
    This function takes as input the session ID to understand in which session the other input value (start date) must be saved in the Database Problem.
  - `setEndDate(sessionID: SessionID, endDate: Datetime)`
    This function takes as input the session ID to understand in which session the other input value (end date) must be saved in the Database Problem.
  - `selectSchool(sessionID: SessionID, school: String,)`

    This function takes as input the session ID to understand in which session the other input value (school) must be saved in the Database Problem.
  - `deleteSession(sessionID: SessionID)`

    This function let to delete the session specified as input.
  - `getProblemData(sessionID: SessionID)`

    With this function it is possible to obtain all the data regarding the session specified as input.
  - `selectSession(sessionID: SessionID)`

    This function let to select the session to work with.

`Unavailability Manager`

This component collect data regarding the unavailability of professors and Politecnico from webapp, then save them in the proper database.

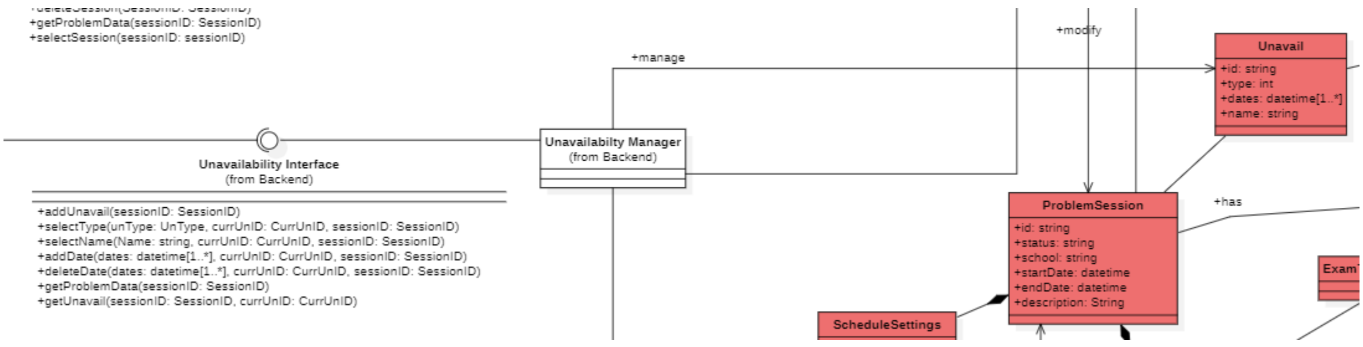
- `Unavailability Interface`
  - `addUnavail(sessionID: SessionID)`
    This function takes as input the session ID to understand in which session will be save the unavailability.

- `selectType(unType: UnType, currUnID: CurrUnID, sessionID: SessionID)`
  This function let to select the unavailability type between professor or university.
- `selectName(name: String, currUnID: CurrUnID, sessionID: SessionID)`
  This function let to select the name related to the type previously selected, it can be a name of a professor if the type is professor or the name of a classroom if the type is university.
- `addDate(dates:datetime[1..*], currUnID:CurrUnID, sessionID: SessionID)`
  This function let to add dates of unavailability.
- `deleteDate(dates:datetime[1..*], currUnID:CurrUnID, sessionID: SessionID)`

  This function let to delete dates of unavailability that are already present.
- `getProblemData(sessionID: SessionID)`

  With this function it is possible to obtain all the data regarding the session specified as input.
- `getUnavail(sessionID: SessionID, currUnID: CurrUnID)`

  With this function it is possible to select the unavailability to work with from a list.


+getProblemData(sessionID: SessionID)
+selectSession(sessionID: sessionID)
+manage
+modify
Unavailability Interface
(from Backend)
+addUnavail(sessionID: SessionID)
+selectType(unType: UnType, currUnID: CurrUnID, sessionID: SessionID)
+selectName(Name: string, currUnID: CurrUnID, sessionID: SessionID)
+addDate(dates: datetime[1..*], currUnID: CurrUnID, sessionID: SessionID)
+deleteDate(dates: datetime[1..*], currUnID: CurrUnID, sessionID: SessionID)
+getProblemData(sessionID: SessionID)
+getUnavail(sessionID: SessionID, currUnID: CurrUnID)
Unavailabilty Manager
(from Backend)
Unavail
+id: string
+type: int
+dates: datetime[1..*]
+name: string
ProblemSession
+id: string
+status: string
+school: string
+startDate: datetime
+endDate: datetime
+description: String
+has
ScheduleSettings

Settings Manager

This component collect data regarding the minimum distance between calls of the same exam and different exams from webapp, then save them in the proper database.

- Settings Interface
  - `setMinDistanceExams(distance: Int, sessionID: SessionID)`
    This function let to set the minimum distance between different exams in the specified session.
  - `setMinDistanceCalls(exam: Exam, distance: Int, in sessionID: SessionID)`

    This function let to set the distance between calls of the same exam in the related session. The distance between calls can be different among exams.
  - `addException(sessionID: SessionID,examID: String, distance: int)`

    This function takes as input the session ID to understand in which session the other input values must be saved in the Database Problem. They are the custom distance between calls and the exam ID of the exam for which is inserted the exception.

- `deleteException(sessionID: SessionID, examID: String)`

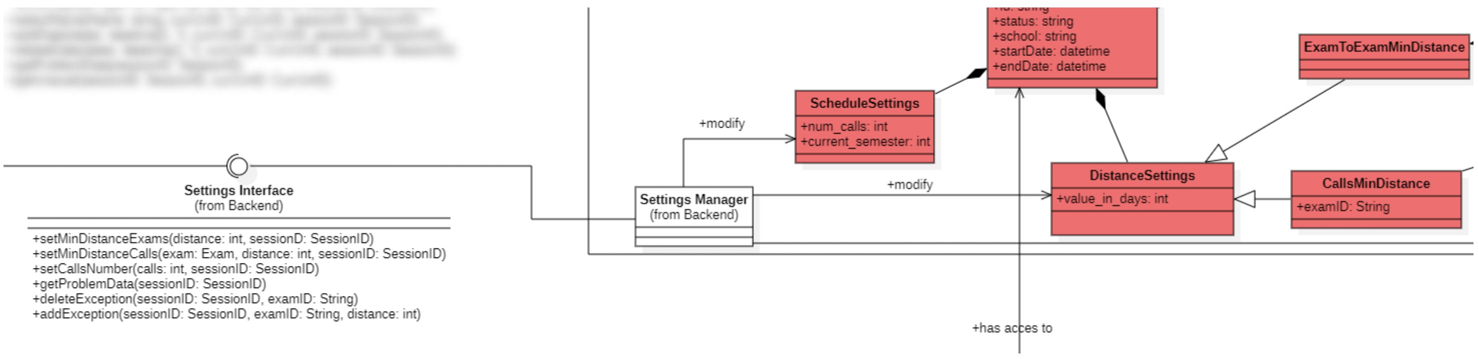
  This function takes as input the session ID to understand in which session of the Database Problem there is the comment to delete. The deleted exception is the one identified by the exam ID.

- `setCallsNumber(calls: Int, sessionID: SessionID)`

  This function let to specify the number of calls for all the exam, in the specified work session.

- `getProblemData(sessionID: SessionID)`

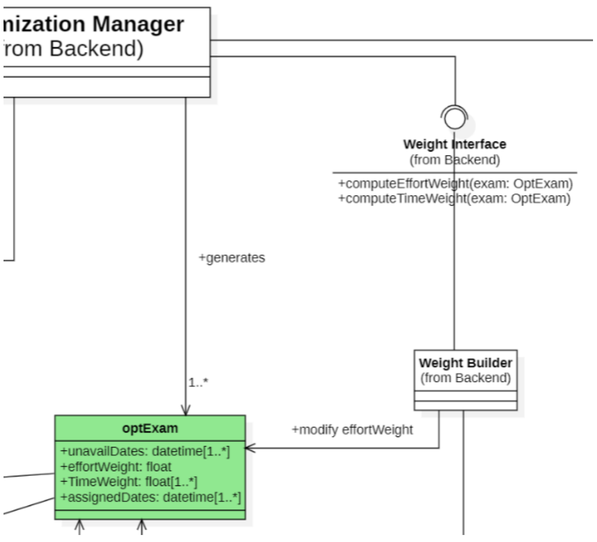
  With this function it is possible to obtain all the data regarding the session specified as input.



+status: string
+school: string
+startDate: datetime
+endDate: datetime
ExamToExamMinDistance
ScheduleSettings
+num_calls: int
+current_semester: int
+modify
Settings Interface
(from Backend)
+setMinDistanceExams(distance: int, sessionD: SessionID)
+setMinDistanceCalls(exam: Exam, distance: int, sessionID: SessionID)
+setCallsNumber(calls: int, sessionID: SessionID)
+getProblemData(sessionID: SessionID)
+deleteException(sessionID: SessionID, examID: String)
+addException(sessionID: SessionID, examID: String, distance: int)
Settings Manager
(from Backend)
+modify
DistanceSettings
+value_in_days: int
CallsMinDistance
+examID: String
+has acces to

Weight Builder

This component builds and associates each exam with its respective weights calculated from the data retrieved from the Exams database, there is one weight regarding the effort and one regarding the time (the is seen as the distance between the semester of an exam and the semester for which the optimization is done).



- `Weight Interface`
  - `computeEffortWeight(exam: OptExam)`
    This function let to computes the weight for each exam using the parameters of the input parameter and return the value calculated.
  - `computeTimeWeight(exam: OptExam)`

    This function let to compute the weight regarding the time, so if the exam is in the current exam session or in one of the previous. It returns the value compute for each exam.

    Weight Builder modify values of optExam object.

Optimization Manager

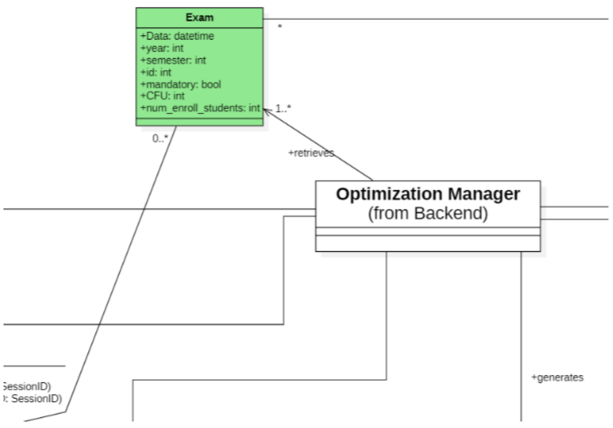
This component provides the interface to start the optimization sequence from the webapp and manage all the requested to be made in order to effectively start the problem.

- `Start Interface`
  - `startOptimization(sessionID: SessionID)`

This function let to start the whole optimization process.

It uses the functions provided by the: weightBuilder Interface (to obtain the values of weights), OptManagement Interface, Exam DBInterface (to collects data regarding the exam to schedule), Problem DBInterface (to collects data regarding the work session).

Moreover, this component works with two objects, it retrives data from Exam object and generates optExam object.

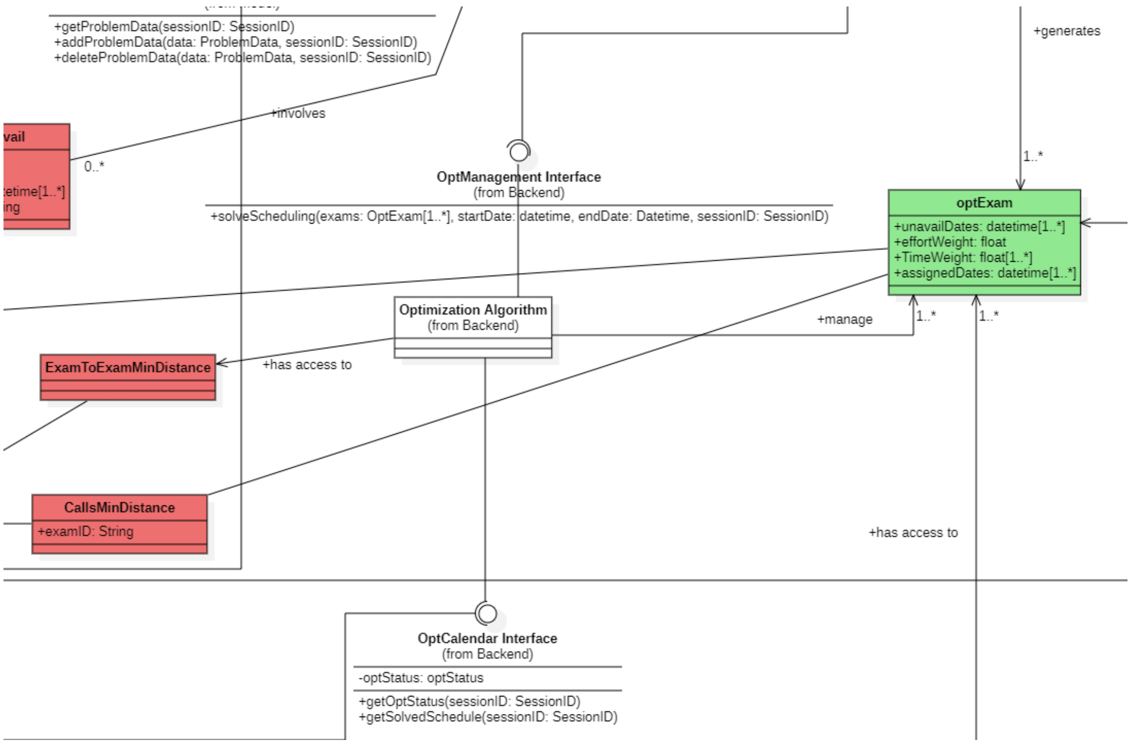


Optimization Algorithm

Returns the solution to the optimization scheduling problem that takes as input.

- `OptManagement Interface`
  - `solveScheduling(exams: OptExam[1..*], startDate: Datetime, endDate: Datetime, sessionID: SessionID): OptExam[1..*])`
    This function let to run the optimization scheduling process using the input parameters.
- `OptCalendar Interface`
  - `getOptStatus(sessionID: SessionID)`
    It returns the status of the optimization process for the selected work session
  - `getSolvedSchedule(sessionID: SessionID): OptExam[1..*])` It It returns the exam with their assigned dates after the optimization algorithm has been computed.

Further explanation on how this component is structured is developed in the 3.4 - Optimization Algorithm.

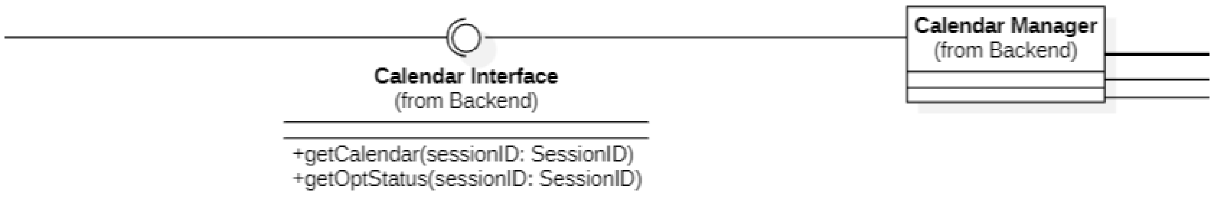

+getProblemData(sessionID: SessionID)
+addProblemData(data: ProblemData, sessionID: SessionID)
+deleteProblemData(data: ProblemData, sessionID: SessionID)
+generates
+involves
0..*
1..*
OptManagement Interface
(from Backend)
+solveScheduling(exams: OptExam[1..*], startDate: datetime, endDate: Datetime, sessionID: SessionID)
optExam
+unavailDates: datetime[1..*]
+effortWeight: float
+TimeWeight: float[1..*]
+assignedDates: datetime[1..*]
Optimization Algorithm
(from Backend)
+manage
1..*
1..*
ExamToExamMinDistance
+has access to
CallsMinDistance
+examID: String
+has access to
OptCalendar Interface
(from Backend)
-optStatus: optStatus
+getOptStatus(sessionID: SessionID)
+getSolvedSchedule(sessionID: SessionID)

`Calendar Manager`

This component collect data from optimization and databases to create the exam calendar then, provides it to the webapp.

- `Calendar Interface`
  - `getCalendar(sessionID: SessionID)`
    It gives as Output the solved scheduling problem in a format that can be used to export the results as Excel or graphic calendar GUI.
  - `getOptStatus(sessionID: SessionID)`
    It allows to know the status of the process for the session specified as input.

The Calendar Manager component uses the functions of the OptCalendar Interface to obtain the scheduled calendar.


Calendar Interface
(from Backend)
+getCalendar(sessionID: SessionID)
+getOptStatus(sessionID: SessionID)
Calendar Manager
(from Backend)

Database

The database package contains two databases, one (Database Exam) to store all the information about each study course, for example course name, professor associated with it but also the data for weight calculations, the study courses are grouped in relation with their study programme to which they belong, some study course can be present in more group. While the other database (Database Problem) contains all the information that the user can insert from the webapp plus the userID, this information is divided into sessions. Database Exam provides these data to the backend only. If there is a change in some fields, it is necessary to perform it by accessing directly. While the data contained in Database Problem are inserted by the user through the webapp

Now are presented and described all the data that can be store in them, the object with which they interact and the interfaces that they provide.

Database Exam, for each study course there are:

- Study Programme (School)
- Course Code
- Exam type: mandatory or elective (M/E)
- Course Name
- Semester
- Year
- Semester (SEM)
- Location
- Exam Head
- Professor
- Section
- Enrolled student number (Enrolled number)

- CFU
- Passed percentage (Passed %)
- Average Mark

Study Programme (School)

The study programme contains more study courses, for each study course (exam) can there be more study programme associated, this means that the study course is shared among other study programmes.

Course Code

The course code is a 6-digit number unique for each course as the name it identifies the course.

Exam type: mandatory or elective (M/E)

This field identifies if the exam is a mandatory one or an elective one. The mandatory exams must be taken by the students, the elective exam can be chosen in a poll of exam following specified rules.

The exam type could be an useful information for the scheduling process because the mandatory course must be optimized with a higher priority than the elective one, because these last group is chosen by the student and most of the time these courses are shared among other course of study so the scheduling are more difficult and the student can not to choose it while they can't not to attend a mandatory course so the must be facilitated in taking the exam. (This part is not considered in the implemented software)

Course Name

The course name is the unique name of the course.

Semester

The semester is the semester in which the course is taught, it can be first semester (form mid-September to end of December) and second semester (from end of February to beginning of June), this filed indicate only in which semester there are the lecturers of the course not when it is possible to take and exam because, each exam as 5 calls per year (for the 3I school for example) independently spread from the semester of lessons.

Year

The year as for the semester identify the year of the degree in which the course is taught but it is not binding with respect to how much you can take the exam.

Semester (SEM)

This field is a mix of the previous two, in the sense that it relates the semester and the year following this table.

	January-February	June-July
1st year	1	2
2nd year	3	4
3rd year	5	6

Location

Location indicates if the course is taught in Bovisa (BV) or Leonardo (MI) Campus.

Exam Head

The name present here is the name of the professor responsible for the exam.

Professor

In this field are listed all the professors that hold the study course, they can be one or more, they are more especially in the case in which the study course is divided in sections.

Section

The section is the batch of student to whom the course is reserved, students are divided in batch in the case in which they are too much to attend together a course, so they are divide depending on their surname. If a course is divided in sections, it remains the same but probably the professor change.

Enrolled student number (Enrolled number)

Student number is the number of students enrolled in a course this, for sure, is essential in the classroom association (but this issue is not addressed in this project). However, this data can be used also for exam scheduling considering that schedule badly a very attended course will result in the discontent of more people.

CFU

CFU is the number of university educational credits, it a number that generally is 5 or 10, in some other cases 8 for example. It represents the theoretical commitment required by and exam and it is also linked to the teaching our if the course.

Passed percentage (Passed %)

This value (in a percentage form) indicates the number of the students that pass the exam over all student that attend the exam. It is an average number. It does not consider situation in which student reject the mark.

Average Mark

This is the average value ok the mark that the students take if they pass the exam, it is an average value and it does not consider the case in which the students reject the mark.

For further development, for example the possibility to assign the classroom other two fields can be considered in this database. They are Exam mode and Pc flag.

Exam mode

The exam mode regards the modality with which an exam is delivered, it could be a written test, an oral test, or a written test plus an oral one. This information could be also used in the scheduling because for example, a written test plus an oral one could require more effort, or for example sometimes the date of oral exam is proposed by the student and confirmed by the professor, so it do not require a specific date in the scheduling. Or rather, also the oral test requires a date in the exam calendar due to its verbalization, but it have a very low weight in the scheduling process since the student can chose the date in which take it.

Pc flag

PC flag is used to know if the student must bring their laptop so if a cabled room is necessary.

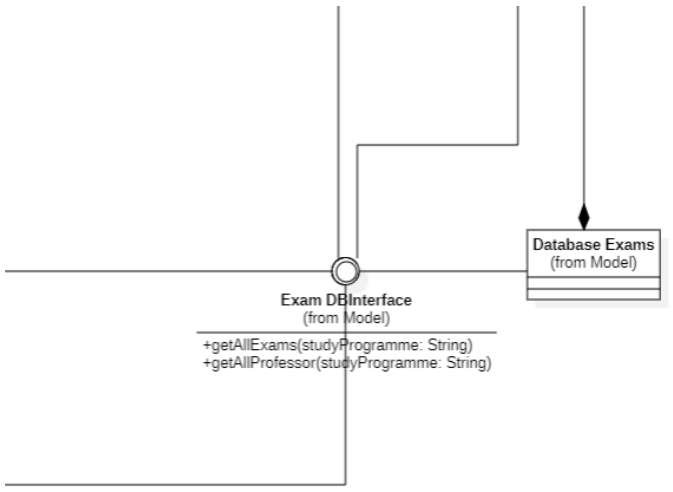
The Interface (and its functions) that Exam Database provides is:

- `Exam DBinterface`
  - o `getAllExams(studyProgramme:string)`

    This function takes as input the study programme and thanks to it, it is selected all the exams that belong to the study programme.
  - o `getAllProfessor(studyProgramme:string)`

    This function takes as input the study programme and return a list of all the professors that belongs to entered study programme.

Its data are necessary to build the Exam object.


Database Exams
(from Model)
Exam DBInterface
(from Model)
+getAllExams(studyProgramme: String)
+getAllProfessor(studyProgramme: String)

In the Database Problem, for each sessionID there are the following data.

- UserID
- Description
- StartDate
- EndDate
- School
- UnavailList
- Status
- Settings

SessionID

This is a unique alphanumeric code that identifies the work session.

UserID

This is the ID of the user that he enters in the first page of the webapp.

Description

Here are stored the comments that an user can write regarding the session.

StartDate

In this field is present the start date of the exam session.

EndDate

In this field is present the end date of the exam session.

School

This identifies the school for which the optimization will be done (for example school 3I).

UnavailList

This filed contains an unique id which in turn contains other elements: type, dates, professors/classroom name). The type indicates if the unavailability regards professors or university, and the name is the name of the professors or the classroom. While the dates are the dates of unavailability.

Status

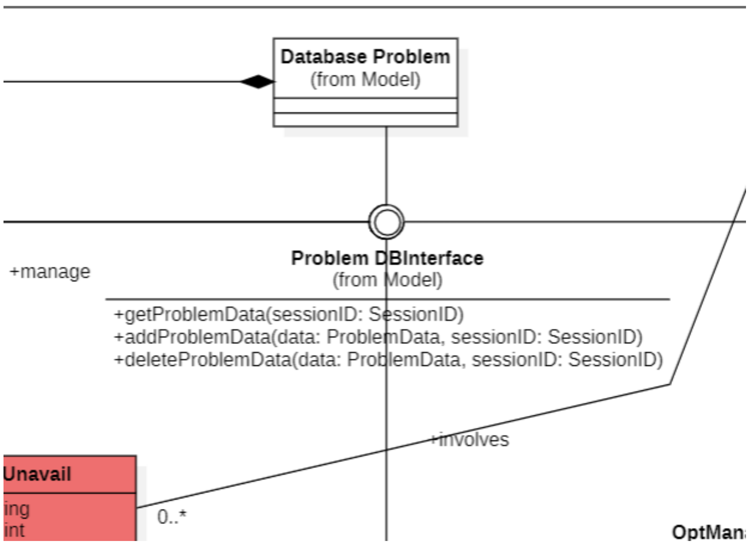
This field regards the status of the optimization process.

Settings

Here are present the settings, they are the minimum distance between calls (there is a default value and eventually an exception for a specific exam), minimum distance between exams, number of calls and semester.

The Interface (and its functions) that Database Problem provides is:

- `Problem DBinterface`
  - `getProblemData(sessionID: SessionID)`
    This function takes as input the SessionID and comparing it the related data are selected.
  - `addProblemData(data: ProblemData, sessionID: SessionID)`
    This function add data in a specific session thank the input parameters.
  - `deleteProblemData(data: ProblemData, sessionID: SessionID)`
    This function deletes an entire problem session stored in the database.



Database Problem
(from Model)
+manage
Problem DBInterface
(from Model)
+getProblemData(sessionID: SessionID)
+addProblemData(data: ProblemData, sessionID: SessionID)
+deleteProblemData(data: ProblemData, sessionID: SessionID)
+involves
Unavail
ing
int
0..*
OptMan

3 - COMPONENTS TASKS

Each component has a specific task indeed it is developed in a custom way. In this chapter for each component, divided in package (front-end, back-end, database) are listed the operations that it must do and in the next chapter they are analysed in detail with sequence diagrams.

Front end

The front end (user interface) lets the registrar's office personnel insert input data in the database and, once the settings are inserted, it calls the optimization algorithm with custom data. Being the front end divided in four pages, each of them has a specific task, but they are represented as a single component, the webapp.

In brackets are listed the satisfied requirements, referring to RASD for them.

It is structured in four pages, the first one is a login page in which the user has to insert his username, after this he can enter in the second page where he has the possibility to choose the session to modify or use, alternatively he can create a new session. The term session refers to the set of entries/settings that can be entered by the user to perform optimisation, a more detailed explanation of the session object is given after ('ProblemSession'). The third page is the one in which can be entered, in sequence, the start and the end date of the exam session, the unavailability of the professors or university and the settings that are the minimum distance between calls of the same exam or the minimum distance between the calls of different exams. When the user inserts the unavailability, he accesses to the fourth page in which he can selects if the unavailability regards the university or a professor (choosing him) and the unavailability dates. This selection can be done in three different methods: selecting a single day, a day for a period or a period. After all these actions and clicking on the respective save button the user can press the start bottom to start the optimization process.

The first page allows user to insert the User ID, log in.

The second page allows user to view the sessions created, view the session status, select the session to work with, delete a specific session, create a new session of work.

The third page allows user to select the school of reference to work with, select the start date of the exam session, select the end date of the exam session, view unavailability, insert unavailability entering in the fourth page, insert settings regarding the distance between different exams or calls of a single exam entering in the fifth page, check the status of the optimization process or Download the excel file of the calendar of the scheduled exam session.

The fourth page allows user to select the type of unavailability, select the professor or the classroom, add unavailability modify unavailability or delete unavailability.

The fifth page allows user to select the type of settings, elect the exam, insert the distance between calls of a specific exam or insert the distance between different exams.

All inputs are used to the respective component in the backend thank to the interfaces.

ProblemSession Manager for the start date, end date, school, description. Unavailability Manager for unavailability and Settings Manager for the distances.

After the user has entered data, it can start the optimization and scheduling process. This is one of the tasks of the webapp (present in the Main Page), so the webapp calls a specific function in the back end. Lastly thanks to the Calendar Manager it is possible to download the calendar.

Back-end

The back-end contains all the logic of the system and as previously explained it is composed by different components, some for the interaction with the webapp issues and others for the optimization and scheduling process. The first group is composed of ProblemSession Manager, Unavailability Manager, Setting Manager and Calendar Manager. The first three components gather data from webapp and store them in Database Problem.

ProblemSession Manager stores start exam session date, end exam session date, description, sessionID, userID, school.

Unavailability Manager stores unavailability of university or professors’ unavailability.

Settings Manager stores distance between different exams or distance between calls of the same exam.

Moreover, Unavailability Manager let to delete unavailability, ProblemSession Manager let to delete session.

The other component, Calendar Manager let to check the status of the optimization and obtain the scheduled calendar.

Regarding the other tasks of the back-end they are done by other components that don’t interact with the webapp.

Optization Manager is the component that manager all the process before the optimization.

It has the task to collect data from the two databases, respectively:

From Database Problem start exam session date, end exam session date, unavailability of university, professors’ unavailability, distance between calls of the same exam and distance between calls of different exams.
From Database Exam, for each exam cfu, exam type, exam mode, commitment required, course name and programme.
The **Optimization Manager** has also other task to do. After having taken data from Database Exam, invokes the Weight Builder components that from it takes parameters as cfu, exam type, exam mode and commitment required to create a weight for each exam necessary for the optimization algorithm. This output parameter is given to the Optimization Manager that will provide it to the optimization algorithm.
The Optimization Manager merges the professor and university unavailability in the exam unavailability too.

Then, it calls the optimization algorithm and provides it with the necessary parameters.

This call is done iteratively for each study programme, and it can update the capacity of each day, after each iteration of the algorithm it stores the result data. If a course is common in more than a programme, and it’s been assigned in the previous iteration, the associated date variable will be set.

At the end of each iteration the backend receives first call date and second call date (only for winter and summer exam session, no for autumn session)

With them and these other listed items (taken from Database Exam): programme, course code, course name, semester, year, location, professor and section.

the calendar, divided by the programme, is made. To create the calendar there is a specific component called Calendar Manager, it creates the calendar saving it as an excel file, but it also provides the calendar to the webapp to let its visualization from the webapp.

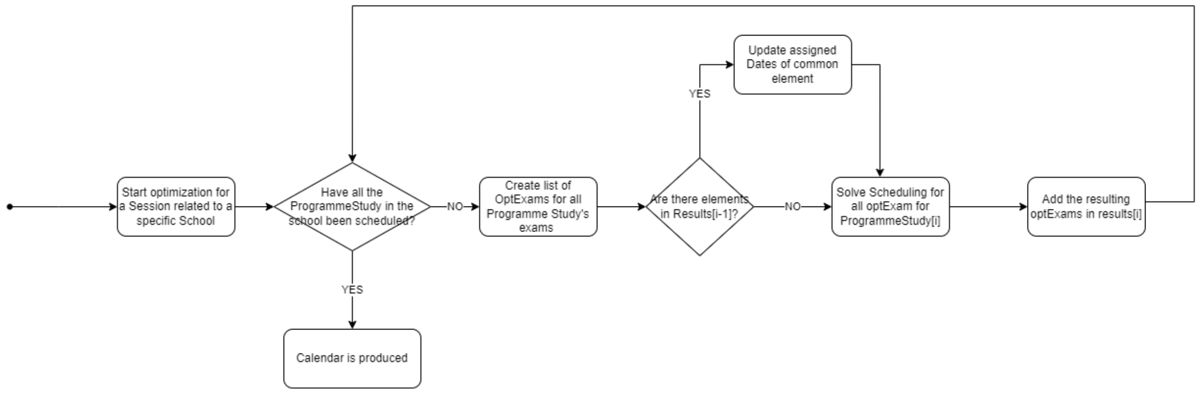
The other tasks made by other components are creating the weight for the effort and time done by the weight builder.

Database

The two databases let the system to store data and take them when necessary.

Optimization Algorithm

The core of SchedulEx is represented by this component because it solves the main problem of exam scheduling for a programme study. In this section we analyse the workflow by which the Optimization Algorithm component is executed and its related logic.



Start optimization for a Session related to a specific School
Have all the ProgrammeStudy in the school been scheduled?
NO
Create list of OptExams for all Programme Study's exams
Are there elements in Results[i-1]?
YES
Update assigned Dates of common element
NO
Solve Scheduling for all optExam for ProgrammeStudy[i]
Add the resulting optExams in results[i]
YES
Calendar is produced

The optimization algorithm is executed for each Programme Study of the school related to the Problem Session that is started by a User. To works properly and to be called successfully from Optimization Manager, which is the component that manage the loop for each Programme Study, it requires the following data:

- **startDate**, a Datetime object.
- **endDate**, Datetime object.
- List of **optExam**, with each instance properly populated.
- **SessionID**, id of the Problem Session to be processed.

After generating a list of optExam corresponding to all the exams that belongs to the programme study, it must be checked whether all common exams that have been scheduled so far belongs to the current list that is taken into consideration. If so, the already assigned dates are set in the corresponding common exam. This must be done to satisfies the rules logic requirements *(R27)*, to not schedule again the common exam during the iteration. The logic will be:

$$\forall i,j\big(e_j \in PS(e_i) \land e_i \in PS(e_j)\big) \Rightarrow d_{i,k} = d_{j,t}\big)$$

After the exam schedule processing and all iteration is completed the Optimization Algorithm returns as output a Set of unique optExam. Each element of this set will have *assignedDates* populated with as much number of element corresponding of the number of calls.

Now, we address to the constraint's redefinition for the problem, considering the design described so far and on which objective function this optimization problem is based on. To this aim we will introduce this abbreviation:

- $d_{i,k} = e_i.assignedDates[k]$
- $w_{i,j} = e_i.timeWeight[j]$
- $v_{i,j} = e_i.effortWeight * e_j.effortWeight$
- $dist(d_1; d_2)$ returns the distance in days between d_1 and d_2
- D is the number of calls.
- N is the length of the list of optExam given in input.

The objective function is:

$$max \sum_{i}^{N} \sum_{j}^{N} \sum_{k}^{D} \sum_{k}^{D} v_{i,j} * w_{i,j} * dist(d_{i,k}, d_{j,t})$$

(1)

This objective function maximizes the distance between every pair of exams included in the scheduling problem. So, in this way we achieve to maximizes distance between different exam or between calls of the same exam.

The costraints will be formulated as following:

- *(R24)* The constrained is now focus on each element of the assigned dates of the single exam

$$\forall i, j, k, t \left(e_j \neq e_i \wedge \ e_i.semester = e_j.semester \Rightarrow \left(dist\left(\mathrm{d_{i,k}} \ ; \ d_{j,t}\right) \geq 2 \right) \right)$$

- *(R25)* For each exam it must be verified that elements of assignedDates distance from each other the number of days specified in input by the user for that exam or the standard value of 14 days.

$$\forall i, k, t \ \left(dist\left(\mathrm{d_{i,k}} \ ; d_{i,t}\right) \geq minCallDistance(e_i) \right)$$

- *(R26)* For each professor who reports his or her unavailability, we need to avoid assigning exam that involve him or her, in those dates. The function *involves_professor(u,e)* determines whether the unavailability *u* involves professor of exam *e.*

$$\neg \exists i, k \left(\ d_{i,k} \in \ e_i.unavailDates \right)$$

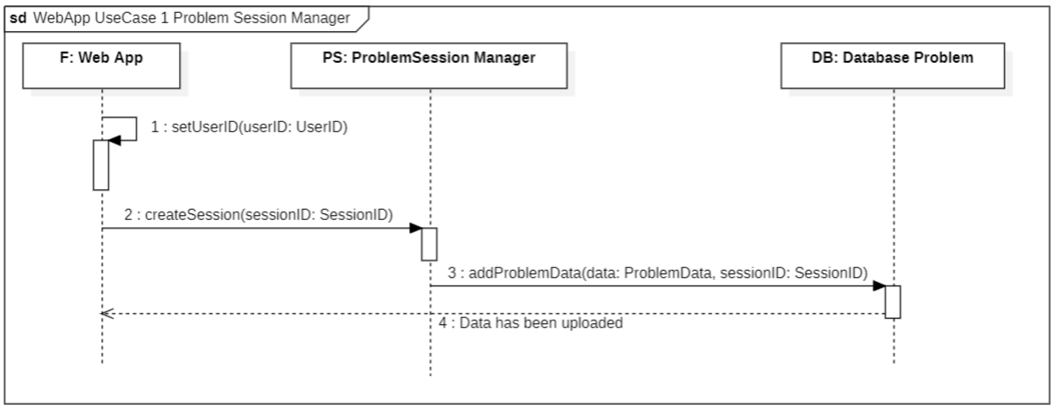
The problem to be faced in the implementation is related to Linear Programming Optimization. Since we will use tools that can solve only linear problem, we need to develop a dist(d,d) as a linear function.

4 - COMPONENTS SEQUENCE DIAGRAM

In this section are present the sequence diagram that

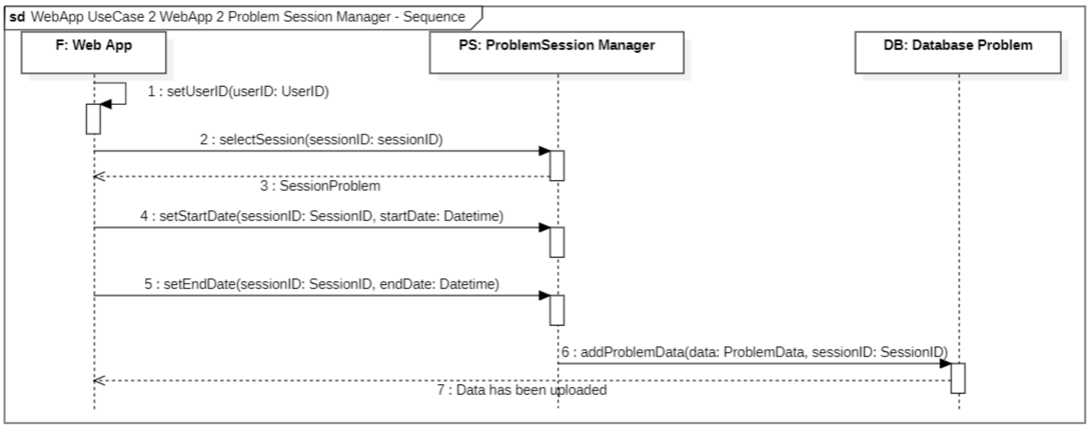
Use Case 1. Create ProblemSession:

Participating actors	• Secretary Users
Entry Condition	Usually, an exam session calendar must be created some months before the actual start
Flow of events	• Secretary User activates the "Create ProblemSession" function. • SchedulEx respond by presenting the GUI to insert all the needed information. • Secretary Users, once they logged in, can open the same page created before to add any information to it
Exit Condition	


sd WebApp UseCase 1 Problem Session Manager
F: Web App
PS: ProblemSession Manager
DB: Database Problem
1 : setUserID(userID: UserID)
2 : createSession(sessionID: SessionID)
3 : addProblemData(data: ProblemData, sessionID: SessionID)
4 : Data has been uploaded

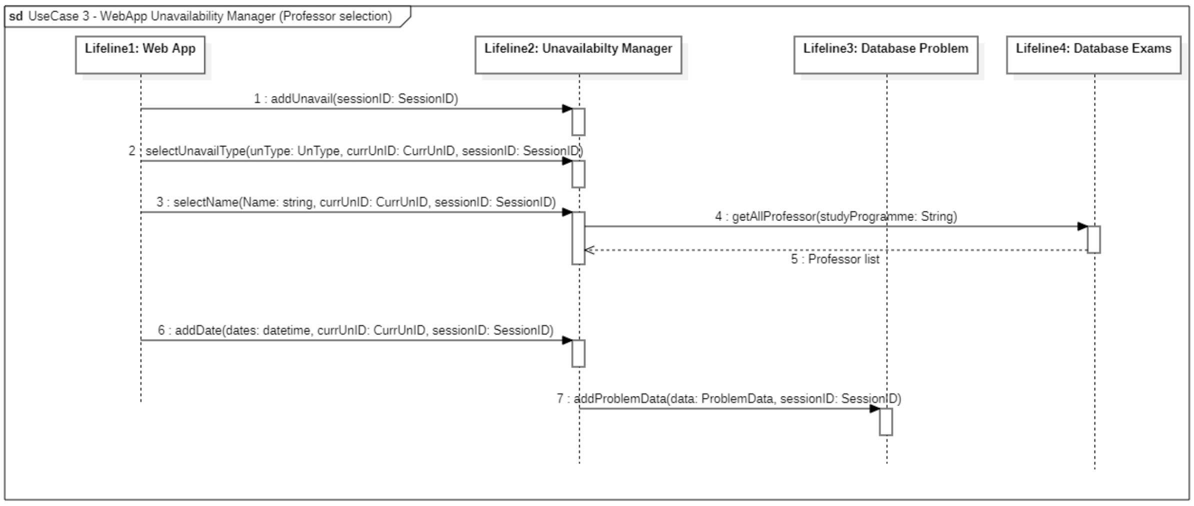
Use Case 2. Set StartEndDate

Participating actors	• Secretary User • School Dean (*indirect user)*
Entry Condition	A ProblemSession must exist
Flow of events	• Secretary User activates the "Open ProblemSession" and SchedulEx return the GUI for the relative ProblemSession. • Here he clicks on the button and activates "set StartEndDate", is returned a calendar in which he can easily select the start and the end of the exam session. • After Prof Svelto confirmation, the selected dates are stored by SchedulEx
Exit Condition	An acknowledgement of the data saving that occurred is visualized


sd WebApp UseCase 2 WebApp 2 Problem Session Manager - Sequence
F: Web App
PS: ProblemSession Manager
DB: Database Problem
1 : setUserID(userID: UserID)
2 : selectSession(sessionID: sessionID)
3 : SessionProblem
4 : setStartDate(sessionID: SessionID, startDate: Datetime)
5 : setEndDate(sessionID: SessionID, endDate: Datetime)
6 : addProblemData(data: ProblemData, sessionID: SessionID)
7 : Data has been uploaded

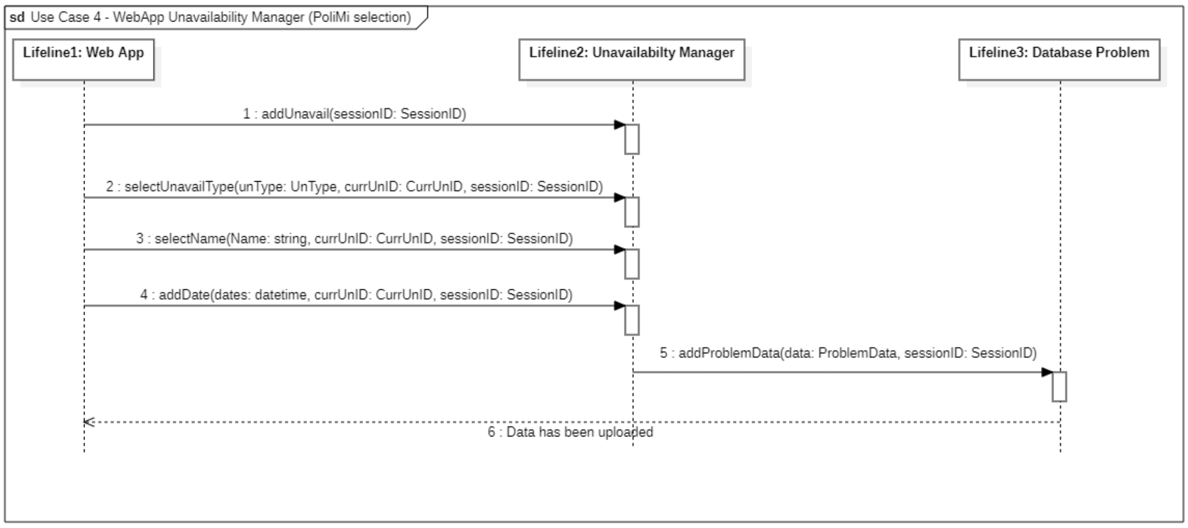
Use Case 3. Add Unavailability

Participating actors	• Secretary User • Professor (*indirect user)*
Entry Condition	A ProblemSession must exist
Flow of events	• *Professor* communicates to the secretary his unavailability. • **Secretary User** activates the "Open ProblemSession" that involves **Professor** and **SchedulEx** return the GUI for the relative ProblemSession. • Here he clicks on button and activates the "Add Unavailability". • **Schedulex** return a form in which **Secretary User** can defined the type (professor), the name (Franchi), and dates in which professor is unavailable. • Thanks to a GUI, **Secretary User** specifies the dates of unavailability (Single day/Period/Recurring) and add to the Unavailability form. • After **Secretary User** confirmation, the overall dates are computed and stored by **SchedulEx**
Exit Condition	An acknowledgement of the data saving that occurred is visualized


sd UseCase 3 - WebApp Unavailability Manager (Professor selection)
Lifeline1: Web App
Lifeline2: Unavailabilty Manager
Lifeline3: Database Problem
Lifeline4: Database Exams
1 : addUnavail(sessionID: SessionID)
2 : selectUnavailType(unType: UnType, currUnID: CurrUnID, sessionID: SessionID)
3 : selectName(Name: string, currUnID: CurrUnID, sessionID: SessionID)
4 : getAllProfessor(studyProgramme: String)
5 : Professor list
6 : addDate(dates: datetime, currUnID: CurrUnID, sessionID: SessionID)
7 : addProblemData(data: ProblemData, sessionID: SessionID)

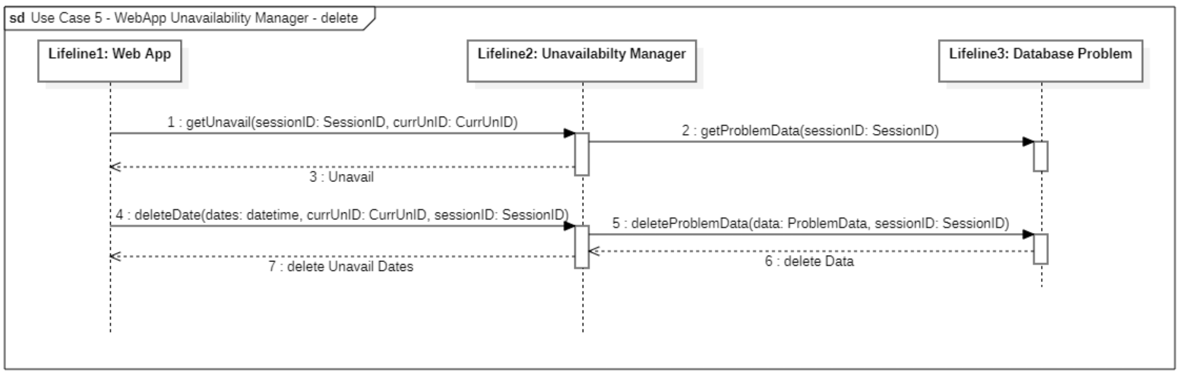
Use Case 4. Add Unavailability – PoliMi Unavailability

Participating actors	• Secretary User • Politecnico (*indirect user)*
Entry Condition	A ProblemSession must exist
Flow of events	• *Politecnico* communicates to the secretary that some class will not be able to use during certain dates. • Secretary User activates the "Open ProblemSession" that involves the rooms that will be used by the School involved and SchedulEx return the GUI for the relative ProblemSession. • Here he clicks on button and activates the "Add Unavailability". • Schedulex return a form in which Secretary User can defined the type (University), the identifier of the rooms (e.g. "3.1.3"), and dates in which those rooms are not available. • Thanks to a GUI, Secretary User specifies the day of this rooms unavailability, and add to the Unavailability form. • After Secretary User confirmation, the overall dates are computed and stored by SchedulEx
Exit Condition	An acknowledgement of the data saving that occurred is visualized


sd Use Case 4 - WebApp Unavailability Manager (PoliMi selection)
Lifeline1: Web App
Lifeline2: Unavailabilty Manager
Lifeline3: Database Problem
1 : addUnavail(sessionID: SessionID)
2 : selectUnavailType(unType: UnType, currUnID: CurrUnID, sessionID: SessionID)
3 : selectName(Name: string, currUnID: CurrUnID, sessionID: SessionID)
4 : addDate(dates: datetime, currUnID: CurrUnID, sessionID: SessionID)
5 : addProblemData(data: ProblemData, sessionID: SessionID)
6 : Data has been uploaded

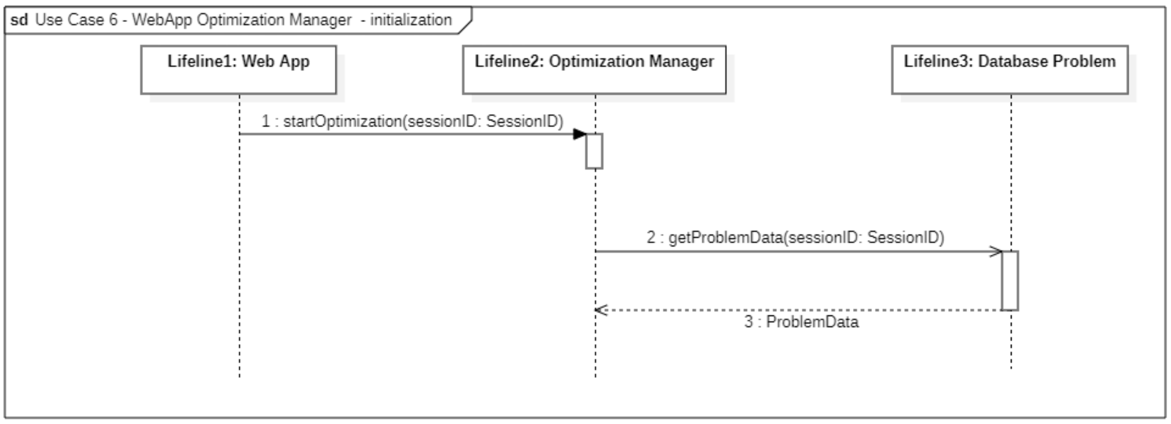
Use Case 5. Modify/Delete Unavailability

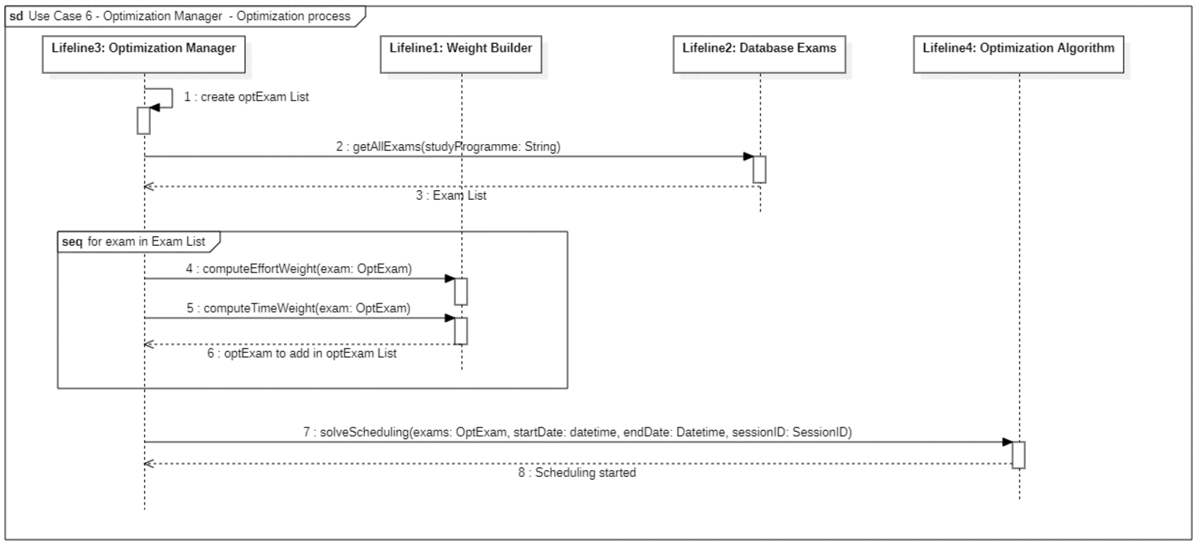
	• Secretary User • Professor (*indirect user)*
Entry Condition	A ProblemSession must exist
Flow of events	• *Professor* communicates to the secretary his change about his unavailability. • Secretary User activates the "Open ProblemSession" that involves Professor and SchedulEx return the GUI for the relative ProblemSession. • Here he searches for the unavailability associated to Professor and select it. • SchedulEx return a resume of that unavailability in which Secretary User can performs some action to modify the input field or delete it. • Secretary User activates "Delete unavailability" function and SchedulEx perform the action. • After this modification, Secretary User insert his comment to keep track of this editing in the description.
Exit Condition	An acknowledgement of the data deleting that occurred is visualized


sd Use Case 5 - WebApp Unavailability Manager - delete
Lifeline1: Web App
Lifeline2: Unavailabilty Manager
Lifeline3: Database Problem
1 : getUnavail(sessionID: SessionID, currUnID: CurrUnID)
2 : getProblemData(sessionID: SessionID)
3 : Unavail
4 : deleteDate(dates: datetime, currUnID: CurrUnID, sessionID: SessionID)
5 : deleteProblemData(data: ProblemData, sessionID: SessionID)
6 : delete Data
7 : delete Unavail Dates

Use Case 6. Start Optimization Process

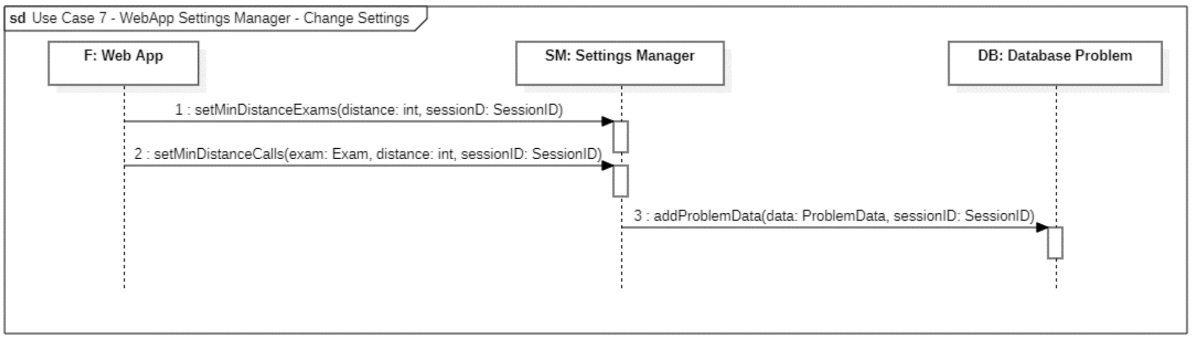
Participating actors	• Secretary User
Entry Condition	A ProblemSession must exist and all information has been entered
Flow of events	• Secretary User needs to compute the overall calendar considering all information entered before. To do so, he/she activates the "Start Optimization". • SchedulEx start to compute the relative problem session and provide information on its status. • Secretary User cannot modify no more the Problem Session
Exit Condition	An acknowledgement of the starting process that occurred is visualized.


sd Use Case 6 - WebApp Optimization Manager - initialization
Lifeline1: Web App
Lifeline2: Optimization Manager
Lifeline3: Database Problem
1 : startOptimization(sessionID: SessionID)
2 : getProblemData(sessionID: SessionID)
3 : ProblemData


sd Use Case 6 - Optimization Manager - Optimization process
Lifeline3: Optimization Manager
Lifeline1: Weight Builder
Lifeline2: Database Exams
Lifeline4: Optimization Algorithm
1 : create optExam List
2 : getAllExams(studyProgramme: String)
3 : Exam List
seq for exam in Exam List
4 : computeEffortWeight(exam: OptExam)
5 : computeTimeWeight(exam: OptExam)
6 : optExam to add in optExam List
7 : solveScheduling(exams: OptExam, startDate: datetime, endDate: Datetime, sessionID: SessionID)
8 : Scheduling started

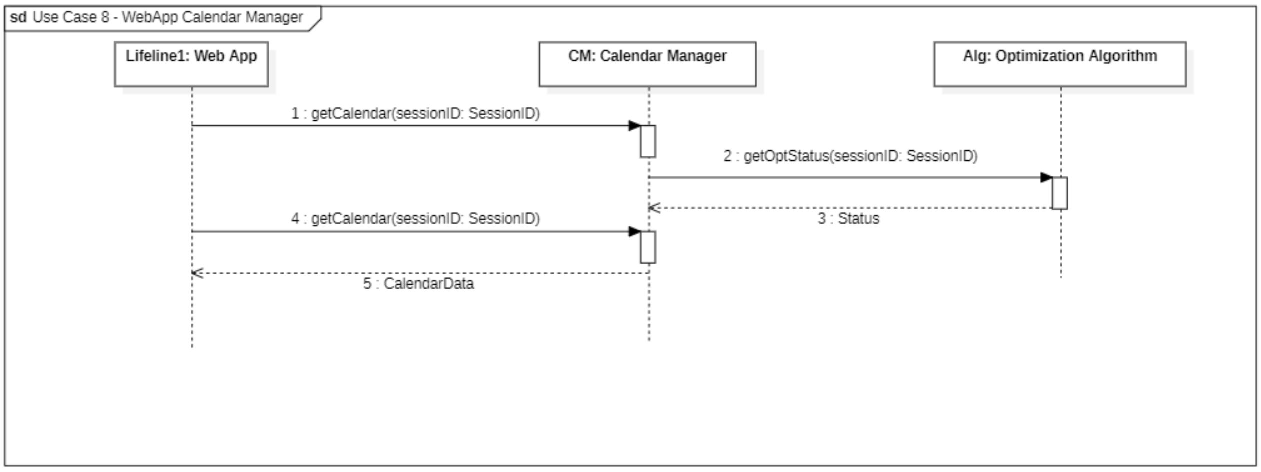
Use Case 7. Change Settings

Participating actors	• Secretary User
Entry Condition	A problem session must exist
Flow of events	• Student Secretary receives the new settings. • **Secretary User** activates the "Open ProblemSession" that involves the rooms that will be used by the School involved and **SchedulEx** return the GUI for the relative ProblemSession. • Here he clicks on button and activates the "Change Settings". • The secretary user makes the changes • After **Secretary User** confirmation, the overall dates are computed and stored by **SchedulEx.**
Exit Condition	An acknowledgement of the data saving that occurred is visualized


sd Use Case 7 - WebApp Settings Manager - Change Settings
F: Web App
SM: Settings Manager
DB: Database Problem
1 : setMinDistanceExams(distance: int, sessionD: SessionID)
2 : setMinDistanceCalls(exam: Exam, distance: int, sessionID: SessionID)
3 : addProblemData(data: ProblemData, sessionID: SessionID)

Use Case 8. Calendar publishing

Participating actors	• Secretary User
Entry Condition	Scheduling procession has completed
Flow of events	• **Secretary User** activates the "view calendar" that opens and display the calendar. • **Secretary User** can download a excel format of the calendar that can be publish on the PoliMi website.
Exit Condition	An acknowledgement of the fact that the calendar is online


sd Use Case 8 - WebApp Calendar Manager
Lifeline1: Web App
CM: Calendar Manager
Alg: Optimization Algorithm
1 : getCalendar(sessionID: SessionID)
2 : getOptStatus(sessionID: SessionID)
3 : Status
4 : getCalendar(sessionID: SessionID)
5 : CalendarData