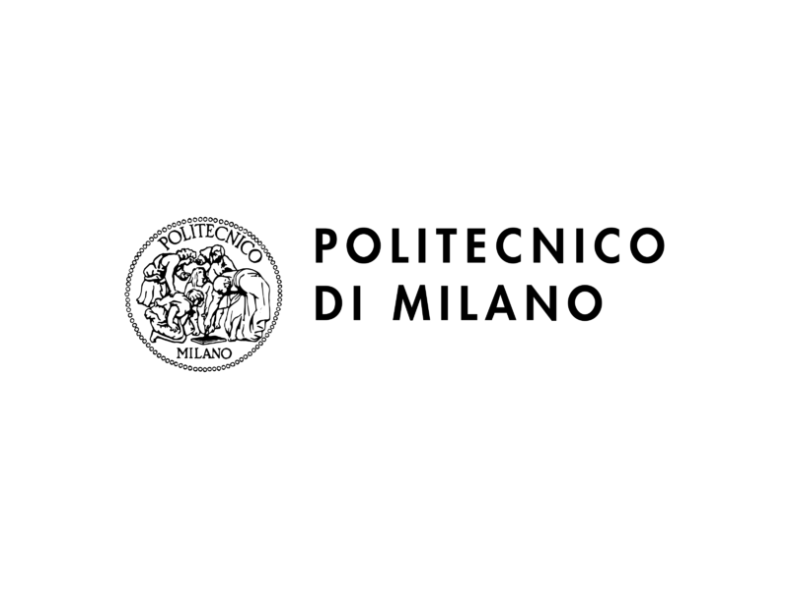
|  |
| --- |
| Petulicchio Lorenzo – Talacci Mattia  [Data] |
| Automation and Control Engineering |
| Design Document |
| Software Engineering Project |

Immagine che contiene Carattere, testo, Elementi grafici, design

Descrizione generata automaticamente

[**1 - INTRODUCTION**](#_heading=) **2**

[**2 - COMPONENTS BREAKDOWN**](#_heading=h.gjdgxs) **3**

[2.1 - Front end](#_heading=h.30j0zll) 3

[2.2 - Back end](#_heading=h.1fob9te) 3

[2.3 - Database](#_heading=h.3znysh7) 3

[2.4 - Optimization Algorithm](#_heading=h.2et92p0) 3

[**3 - COMPONENTS TASKS**](#_heading=h.tyjcwt) **3**

[3.1 - Front end](#_heading=h.3dy6vkm) 3

[3.2 - Back end](#_heading=h.1t3h5sf) 4

[3.3 - Database](#_heading=h.4d34og8) 5

[3.4 - Optimization Algorithm](#_heading=h.2s8eyo1) 5

[**4 - COMPONENTS SEQUENCE DIAGRAM**](#_heading=h.17dp8vu) **6**

[4.1 - Front end](#_heading=h.3rdcrjn) 6

[4.2 - Back end](#_heading=h.26in1rg) 6

[4.3 - Database](#_heading=h.lnxbz9) 6

[4.4 - Optimization Algorithm](#_heading=h.35nkun2) 6

[**5 - SIMPLIFICATION FOR CODE**](#_heading=h.1ksv4uv) **6**

# 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.

1.1 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, we consider 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

# 

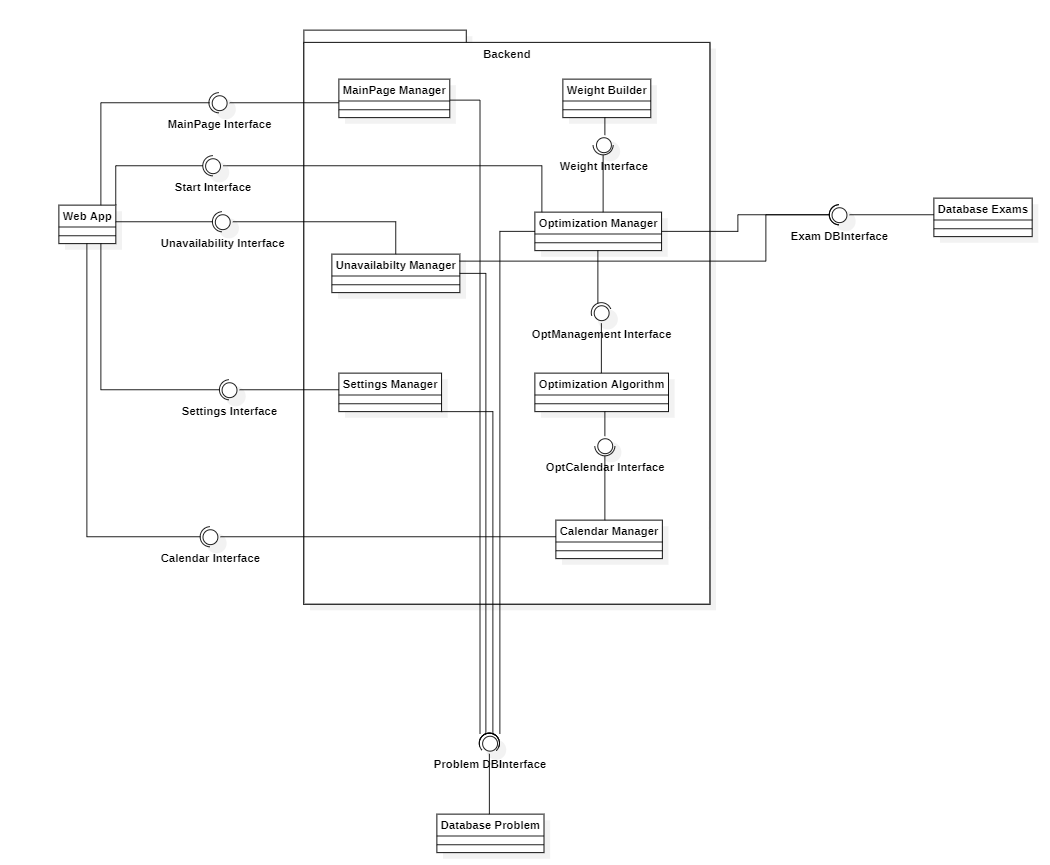
# 2 - COMPONENTS BREAKDOWN

Here, for every component it is described the motivation of the choice to have it and the programming language (if needed).

## Overview

Now, it is presented the entire components diagram in which we can 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 it optimization and they linked among them.



## 2.1 - Front end

The front end (user interface) is a web app developed with Flutter framework, the choice is done because it is a very popular framework and in addition web apps are extremely diffuse nowadays, so this choice is done in order to use tools useful in the world of work.

The front end is composed of three pages. The first one is the main page and appears when the user accesses to the web app, it lets you to insert the start and end date there is also a box to see the professor unavailability and the possibility to delete them. There are three buttons in the main page: one to start the exam scheduling, one to access to another page for other unavailability and the last one to change some settings.

The secondary page deals with professor unavailability and let the user to insert them in two different ways: selecting a period (a single day is considered as a period) or a day (same day in every week)

While the settings changing accessible form the main page let the user to change settings as: distance between calls of the same exam and distance between calls of different exams.

They have default values because generally they are standard.

## 2.2 - Back end

The back end is written with the programming language python, since the optimizer algorithm uses libraries that are very popular in optimization. The backend works before and after the exam scheduling processing because before it collects data and starts command from the frontend and performs the optimization. After it collects the optimization algorithm output and creates the calendar. Because the instant the input data is entered and the actual result is produced can differ greatly in time, each function is associated with a UserID that identifies the user who is using the webapp. This allows the user to be able to manage over multiple days the input of unavailability and track down all possible modification. Moreover, the optimal scheduled calendar will be send to the webapp according to the user that requires for it.

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

MainPage Interface

* setStartDate(in userID:UserID, in StartDate:datetime)
* setEndDate(in userID:UserID, in EndDate:datetime)
* selectStudyProgramme(in studyProgramme:string, in userID:UserID)
* **Unavailability Manager -** This component collect data regarding the unavailability of professors and Politecnico from webapp, then save them in the proper database.

Unavailability Manager

* addUnavail(in userID:UserID)
* selectUnavailType(in unType:UnType, in currUnID:CurrUnID, in userID:UserID)
* selectProfessor(in professor:string, in currUnID:CurrUnID, in userID:UserID)
* addDate(in dates:datetime[1...\*], in currUnID:CurrUnID, in userID:UserID)
* deleteDate(in dates:datetime[1...\*], in currUnID:CurrUnID, in userID:UserID)
* **Settings Manager -** This component collect data regarding the the minimum distance between calls of the same exam and different exams from webapp, then save them in the proper database.

Settings Interface

* setMinDistanceExams(in distance:int, in userID:UserID)
* setMinDistanceCalls(in exam:Exam, in distance:int, in userID:UserID)
* **Weight Builder -** This component builds and associates each exam with its respective weight *w\_i* calculated from the data retrieved from the Exams database

Weight Interface

* computeEffortWeight(Exam: Exam): float Returns the computed weight for each exam. It takes into consideration parameter such as CFU, hour of study needed, average percentage of failed students and other data
* computeEffortWeight(in exam:Exam)
* **Optimization Manager -** This components provide 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(in UserID)
* **Optimization Algorithm** Returns the solution to the optimization scheduling problem that takes as input.

OptManagement Interface

* solveScheduling(exams: optExam[1…\*], startDate: Datetime, endDate: Datetime, userID: UserID)

OptCalendar Interface

* getOptStatus(in userID:UserID)Returns the status of the optimization problem
* getSolvedSchedule(in userID:UserID): Exams[1…\*] Returns the exam with their assigned dates after the optimization algorithm has been computed.
* **Calendar Manager -** This component collect data from optimization and databases to create the exam calendar then, provides it to the webapp.

Calendar Interface

* getCalendar(in userID:userId)Output the solved scheduling problem in a format that can be used to export the results as Excel or graphic calendar GUI.

## 2.3 - Database

The database is a relational one, in particular the adopted one is MySQL, the choice is due to the interest in learning how to use a relational database and being MySQL one of the most used it could be a good tool to learn.

* **Database Exams – In this database are stored all the information about each study course, for example course name, professor associated with it but also the data for weight calculations.**

Exam DBinterface

- getAllExams(in studyProgramme:string)

- getAllProfessor(in studyProgramme:string)

* **Database Problem – In this database are stored all data coming from webapp, so the ones inserted by the e user.**

Problem DBinterface

* getProblemData(in userID:UserID)
* addProblemData(in data:ProblemData, in userID:UserID)

## 2.4 - Optimization Algorithm

The optimization algorithm is written with the coding language Python, the choice is done because Python offers a large number of functions to solve optimization problems.

The following libraries are used to solve the problem

# 

# 3 - COMPONENTS TASKS

Each component has a specific task indeed it is developed in a custom way. In this chapter for each component are listed the operations that it has to do and in the next chapter they are analysed in detail with sequence diagrams.

## 3.1 - 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 three pages, each of them has a specific task, but they are represented as a single component, the webapp.

The first page, called Main Page has to:

* Allow users to insert the starting and ending date for the exam session.

The second page, called Unavailability Page has to:

* Allow users to insert unavailability for a specific professor or for Politecnico
* Allow users to delete unavailability previously inserted

The third page, called Settings Page has to:

* Allows user to define other specification for the exam session such as:
  + Minimum distance between exams
  + Minimum distance between calls of single exam

All these inputs will be part of the custom data settings and are sent to the respective component in the backend.

Main Manager for the starting and ending date, 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 task of the webapp (present in the Main Page), so the webapp calls a specific function in the back end.

At the end of the process, an output will be visible on the main page, this is the calendar with optimized exam scheduling.

## - 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 Main Manager, Unavailability Manager and Setting Manager, these components gather data from webapp and they are:

* start exam session date (date)
* end exam session date (date)

for the Main Manager

* unavailability of university (array of dates)
* professors’ unavailability (array of dates for each professor)
* unavailability data to delete (date)

for the Unavailability Manager

* distance between different exams
* distance between calls of the same exam

for the Settings Manager.

Moreover, the back end, with **Unavailability Manager** component, provides a list of professors to the webapp in order to let the user insert the unavailability for a specific professor. This list is taken from the Exam Database. It is important to notice that in the database some professors are present more times due to the fact that a professor could have more than one course, so it is important to select all the professors just one time.

Regarding the other tasks of the backend they are done by other components that start when a specific button on the webapp is clicked and after establishing a connection with the databases and gathering all the data makes some computation to build the right parameters in order to call the optimization algorithm.

In particular the **Optimization Manage**r has the task to collect data from the two databases, respectively:

* from Database Problem:
  + start exam session date (date)
  + end exam session date (date)
  + unavailability of university (array of dates)
  + professors’ unavailability (array of dates for each professor)
  + distance between calls of the same exam
  + distance between calls of different exams
* from Database Exam, for each exam:
  + cfu (int)
  + exam type (int)
  + exam mode (int)
  + commitment required (int)
  + course name
  + 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
* 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
* section

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

An important aspect to notice is that in the calendar there is also a field regarding the time slot that is not treated because it is assign in a second phase also in the current process.

## 3.3 - Database

The system is composed of two separated databases because they are different and are different the contained values

The Database Exam (a MySQL database) is used as data storage for parameters that are not entered by the user and are always the same except in the case where there are some modifications to the study course, for example a change in the number of cfu of an exam or in the association professor-exam .

For each PoliMi course the following data are stored:

* programme
* course code
* course name
* semester
* year
* location
* professor
* section
* CFU
* exam mode: written/oral/written+oral (oral includes also project discussion)
* exam type: mandatory or elective
* student number enrolled to the course
* pc flag

### Programme

The programme is the study programme that contains more study courses, the course code is a 6-digit number unique for each course as the name the identify the course. 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 independently spread from the semester of lessons. The year as for the semester identify the 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.

### Location

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

### Section

The section is the batch of student to whom the course is reserved, student 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 section it remain the same but probably the professor change.

### 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 represent the theoretical commitment required by and exam and it is also linked to the teaching our if the course.

### 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 is essential also 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 are 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 require a date in the exam calendar due to its verbalization but it have a very low weight in the scheduling process due to the fact that the student can chose the date in which take it.

### Exam type

The exam type is another important 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 o the time these courses are shared among other course of study so the scheduling are more difficult and the student can not to chose it while they can’t not to attend a mandatory course so the must be facilitated in taking the exam.

### Student number

Student number is the number of student 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

### Pc flag

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

Database provides these data to the backend only. If there is a change in some fields, it is necessary to perform it by accessing directly.

## 3.4 - Optimization Algorithm

The optimization algorithm solves the main problem that is the exam scheduling of the session for each programme. It takes as input from backend when it is called the following data:

* start exam session date (date)
* end exam session date (date)
* exam unavailability
* exam weight

The optimization is done in this way…………………………………………………

After the exam schedule processing the outputs given by the optimization algorithm to the backend are for each exam of each programme:

* first call date
* second call date (only for winter and summer exam session, no for autumn session)

# 

# 4 - COMPONENTS SEQUENCE DIAGRAM

In this section are present the sequence diagram for interactions of each component

## 4.1 - Front end

It is a passive components, it is divided in tree pages: *Home, Unavailability Manager, Session Settings* and each time a specific button is clicked, after that values are inserted in the appropriate spaces the respective backend function to store data in database is called.

* *Home* features and functions:

It let the user to set the reference period of the exam session (start and end date), select the study programme and obtain the calendar after optimization.

* + setStartDate(in userID:UserID, in StartDate:datetime)
  + setEndDate(in userID:UserID, in EndDate:datetime)
  + selectStudyProgramme(in studyProgramme:string, in userID:UserID)
  + getCalendar(in userID:userId)
* *UnavailabilityManager* features and functions:

It let the user to insert unavailability of professors or Politecnico selection one or more days. In case in some days there is not an availability previously insert, it is possible to delete it.

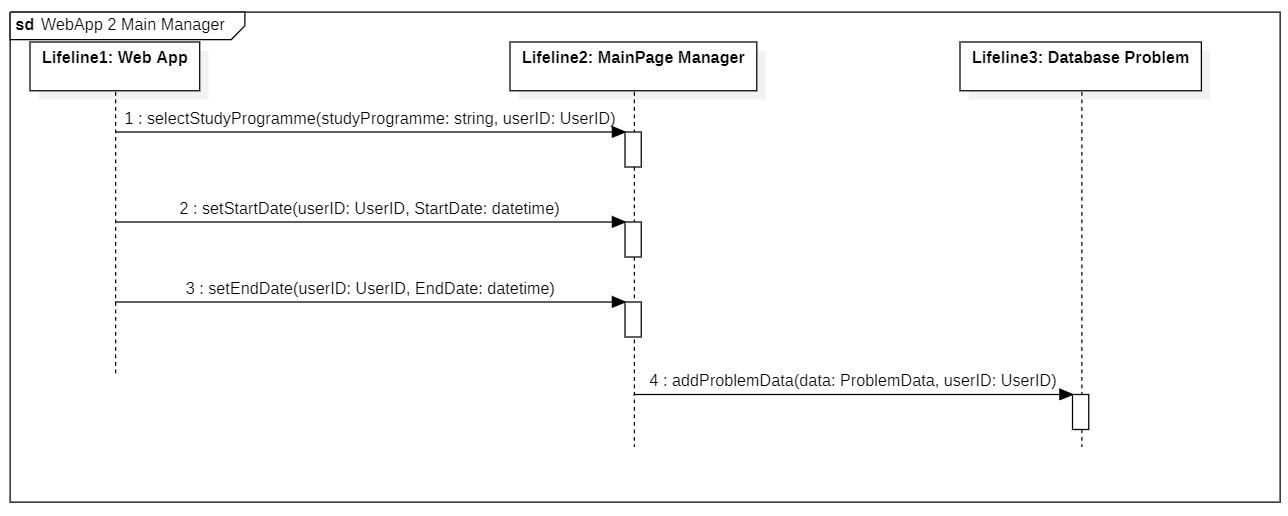
* + addUnavail(in userID:UserID)
  + selectUnavailType(in unType:UnType, in currUnID:CurrUnID, in userID:UserID)
  + selectProfessor(in professor:string, in currUnID:CurrUnID, in userID:UserID)
  + addDate(in dates:datetime[1...\*], in currUnID:CurrUnID, in userID:UserID)
  + deleteDate(in dates:datetime[1...\*], in currUnID:CurrUnID, in userID:UserID)
* *SessionSettings* features and functions:

It let the user to change the distance between two different exams or two calls of the same exam.

* + setMinDistanceExams(in distance:int, in userID:UserID)
  + setMinDistanceCalls(in exam:Exam, in distance:int, in userID:UserID)

### Entering start and end date and storing

In the diagram below it is presented the workflow of the first action by the user, the entering of the start and end date. For each of them the webapp involve the back-end, in particular the Main Manager that process the inserted data. after the user inserts the start and end dates they are saved in the database.



### Entering PoliMi unavailability and storing

In the diagram below it is presented the workflow of one of the two possibilities about the entering of unavailability. These regard the unavailability of Politecnico di Milano, that could be degree days or other closures for events for example. After the user inserts the unavailability dates they are saved in the database.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

### Entering professors’ unavailability

In the diagram below it is presented the workflow of the second of the two possible types of unavailability. This regard the professors that can express preferences to don’t have a specific day of the week or a set of days in which as day of exams. After the user inserts the unavailability dates they are saved in the database.

**Immagine che contiene testo, schermata, linea, numero

Descrizione generata automaticamente**

### Unavailability removal

At every time, selecting the desired unavailability from the list it is possible to delete it, independently if it regards a professor or Politecnico.

SECONDO ME MANCA QUALCHE FUNZIONE PER SELEZIONARE L’INDISPONIBILITA’



low it is shown that after the user inserts the unavailability dates they are saved in the database.

### Setting changing and storing

In the diagram below it is presented the situation in which it is necessary to change the minimum distance between two calls of the same study course or the minimum distance between two different exams, by default they are respectively 14 days and 2 days (2 night). After the user changes the settings the new values are saved in the database.

**Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente**

### Calendar visualization

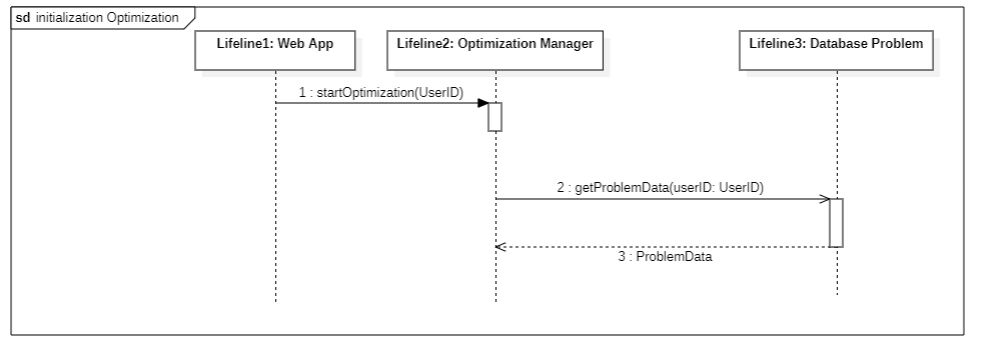
In the diagram below it is presented the interaction between the webapp and the component that provides the calendar, in this wat it possible to give provide it as a file to the user.

**Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente**

### Initialization of optimization

When user had inserted all the data needed to build the custom schedule, the optimization manager retrieves from the problem database those data, to process them.



## 4.2 - Back end

It is divided into different components; each one has a specific task to perform. They are:

*Home Manager*, *Unavailability Manager*, *Settings Manager* and *Start Manager* for the interactions with front end when a user insert data.

* *Home Manager* features and functions:

It adds to the Problem Database the values entered in the Home Page of the webapp.

* + addProblemData(in data:ProblemData, in userID:UserID)
* *Unavailability Manager* features and functions:

It adds to the Problem Database the values entered in the Unavailabilty Page of the webapp, but it also takes the professors’ list from the Exam Database

* + addProblemData(in data:ProblemData, in userID:UserID)
  + getAllProfessor(in studyProgramme:string)
* *Settings Manager* features and functions:

It adds to the Problem Database the values entered in the Settings Page of the webapp.

* + addProblemData(in data:ProblemData, in userID:UserID)

*Calendar Manager* for the interaction with front end when the optimization algorithm finishes and the back and creates the calendar.

* *Calendar Manager* features and functions:

It provides the calendar to the webapp when it is asked if it is ready. It takes it from the Optimization Algorithm.

* + getCalendar(in userID:userId)The sequence diagrams about these components are present in the previous section because they are linked with the front end.

*Weight Builder*, *Optimization Manager*, Optimization Algorithm and Calendar Builder are the components that work about the optimal calendar scheduling.

* *Weight Builder* features and functions:

This component is in charge of create the weight for each exam to use in the optimization algorithm. It is invoked by the Optmization Manager that gives to it the parameters for each exam (taken form the Database Exam) and the Weight Builder give back the computed weight.

* *Optimization Manager* features and functions:

The Optimization Manager is the main components of the backend, it is invoked by the webapp when the start button on it is clicked. The Optimization Manager takes exam values from the Database Exam and general settings from the Database Problem, then it invokes the Weight Builder Component for the weight calculation. After that it invokes the optimization algorithm giving to it the exam values taken from the Database Exam, the weight computed by the Weight Builder and the general settings provided by the Database Problem.

* + getProblemData(in userID:UserID)
  + computeEffortWeight(in exam:Exam)
  + solveScheduling(in exams:optExam[1...\*], in startDate:datetime, in endDate:datetime, in userID:UserID)
* Optimization Algorithm

Its task is to find the optimal scheduling of the exam session for each study course, it is invoked by the Optimization Manager that provide alle the necessary parameters for the optimization. After the process it returns the first and the second date of the calls for each exam of study programme. The optimization is performed sequentially for each study programme.

* *Calendar Builder*

The Calendar Builder component is the one that create the final calendar taking the dates scheduled from the Optimization Algorithm. Moreover, it does not take only the datesbut before receiving it, the Calendar Builder asks also the optimization state to understand if the optimized scheduling ready.

* + getOptStatus(in userID:UserID)
  + getSolvedSchedule(in userID:UserID)

### Optimization process

To first step done by the Optimization Manager, when receive the start command, is to take the data relative to the exam from the Database Exam

To second step done by the Optimization Manager, when receive the start command, is to call the Weight Builder components to create the weight for each exam.

……….

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

### Calendar Creation

The Calendar

Immagine che contiene testo, schermata, linea, ricevuta

Descrizione generata automaticamente

## 4.3 - Database

Only the backend components communicate with databases, they are two and are used to store data and provide them only when backend calls them. So, database is a passive component. There is one database (Database Problem) in which backend components can read and write data. The data stored in it are the general settings entered by the user through the webapp (for this reason it is necessary the write function)

The second database called Database Problem is the one that contains all the information about the exam.

## 4.4 - Optimization Algorithm

The optimization algorithm is composed as follows:

# 5 - SIMPLIFICATION FOR CODE

In order to develop a software that can work in a small amount of time and can find a feasible solution some simplifications are done.

In particular, it has been considered only *“x number”* of days available to schedule exams, *“y number”* of exams to schedule and an infinite classroom capacity. In this way the it is possible to find a feasible solution and it takes *“z”* minutes

The algorithm can be implemented with

# 