1. **INTRODUCTION**

Purpose

This software design document describes the architecture, and design of the system. which performs a fair division of elective courses for the students, according to their preferences over the courses and according to the maximum capacity of each course.

The target audience is university departments and their students.

Scope Provide

Today in most higher education institutions, the mode of division of elective courses is in the "first come, first served” (FCFS) method, which means that registration opens at the same time for a large number of students. In our opinion, this method causes an unfair division of the courses between the students. Thus, we would like to develop a new method of fair division among the courses.

Overview

## Our product will integrate with the current system of the university, we gonna add to the registration procedure another level-“Ranking level”, such that the level would be between the enrollment period and change period.

At the enrollment period student would enroll only in the mandatory courses.

At the ranking level, the students will rank the elective courses that are not overlapping with mandatory courses.

And for the change period, the procedure will stay the same.

**2. SYSTEM OVERVIEW**

Functionality:

Client-side:

**Student side:**

* input\_validation(rank): @return boolean Making sure that the student ranked all the course groups, and every pair of course groups have different rank.
* redefine\_rank(rank): @return boolean, rank

If the student has been press on the redefine the rank, the system will provide his/her latest rank and will let redefine the rank itself.

* time\_remaining(): @return DATE

The system will provide the time remaining for the student to rank.

* start\_period\_of\_rank(settings): @return boolean

The option to rank will open for the students from the date of registration until the deadline, that has been defined by the coordinator.

* Show\_result(student\_id): @return Result

The option to rank will open for the students from the date of registration until the deadline, that has been defined by the coordinator.

**Coordinator side:**

* receive\_input(String StudentJson , StringCourseJson):

@return Boolean

The function receives names of Json files which conclude students and courses. It will convert each one of them to an object.

Example of a Json input :

Student

{

"student": [

{

"id":"01",

"name": "Tom",

"Email": "tom@gmail.com",

"amount\_electiv":5,

"courses":["1","9","14"]

},

{

"id":"02",

"name": "Tami",

"Email": "tami@gmail.com",

"amount\_electiv":5,

""courses":["1","5","9","14"]

},

{

"id":"03",

"name": "Som",

"Email": "Som@gmail.com",

"amount\_electiv":5,

"courses":["5","9","14"]

},

]

}

Courses

{

"course": [

{

"id":"01",

"name": "course1",

"lecturer": "avi ron",

"capcity": 100,

"is\_elective": False,

"day":"C",

"semester":"A",

"start\_time":"10:00",

"end\_time":"13:00",

"link":"fsafasfa"

},

{

"id":"02",

"name": "course2",

"lecturer": "roni ron",

"capcity": 100,

"is\_elective": True,

"day":"D",

"semester":"B",

"start\_time":"10:00 B",

"end\_time":"13:00 B",

"link":"fsafasfa"

},

{

"id":"03",

"name": "course3",

"lecturer": "avi ron",

"capcity": 100,

"is\_elective": True,

"day":"A",

"semester":"A",

"start\_time":"10:00 A",

"end\_time":"13:00 A",

"link":"fsafasfa"

},

]

}

* Insert\_setting(start\_date, end\_date): @return Boolean

The coordinator will insert the starting time and deadline for ranking, the function will check validation.

* time\_remaining(): @return DATE

The system will provide the time remaining for the student to rank.

* approve\_results(list(result)): @return boolean

The system will send the results only after the approval of the coordinator.

**Server-side:**

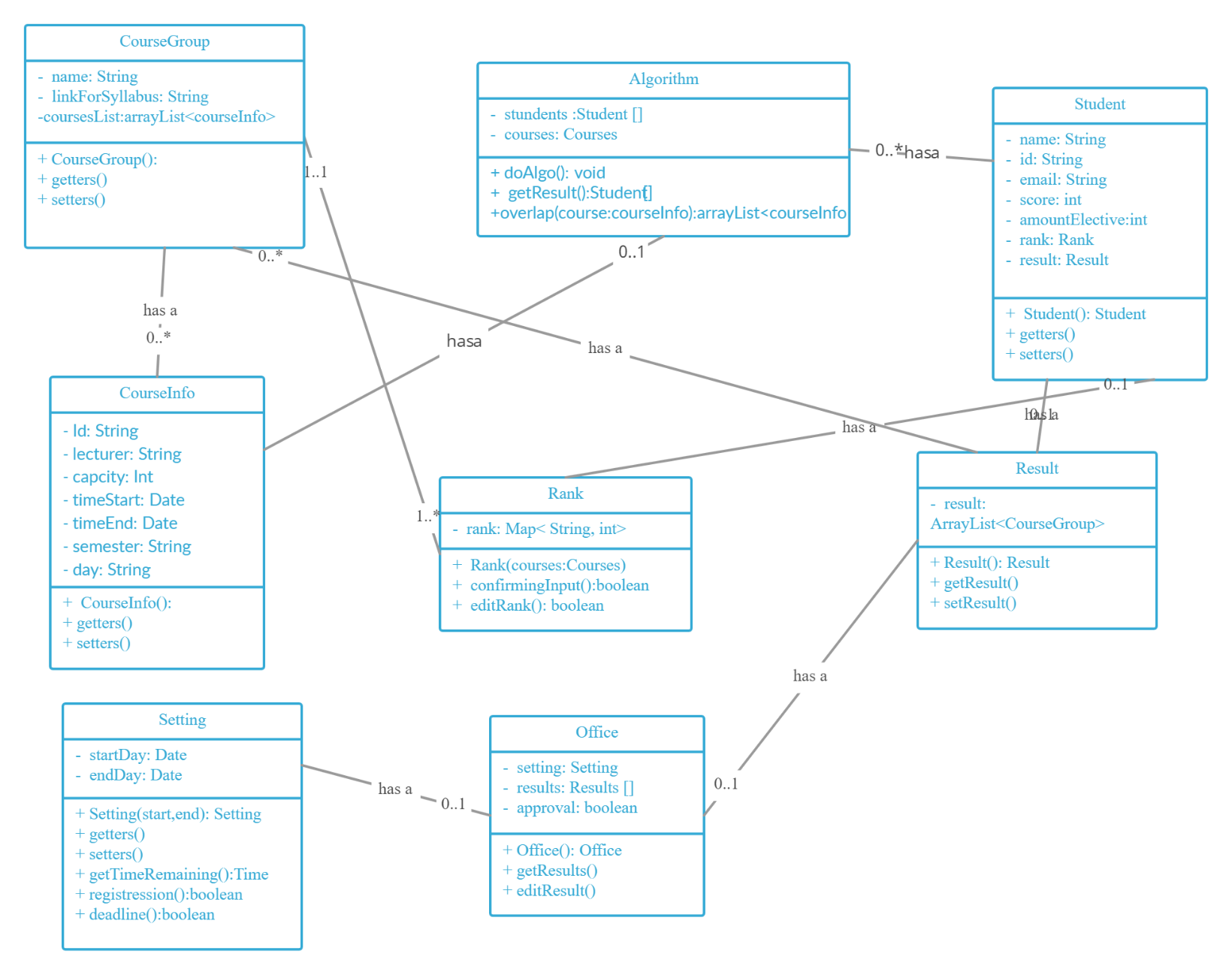
* over\_lap(courses): @return list(pair<course\_id,course\_id>)

To deﬁne forbade course overlap, the system will define binary parameter Ojj’ equal one if and only if the course section j and j’ are overlapping course sections in time or they are different sections of the same course(all the courses include required courses).

* output\_according\_to\_rank(list(course), list(rank)): @return list(result)

After the deadline, the system will process the input by the algorithm and return the output for each student according to his rank.

**Object Diagram:**



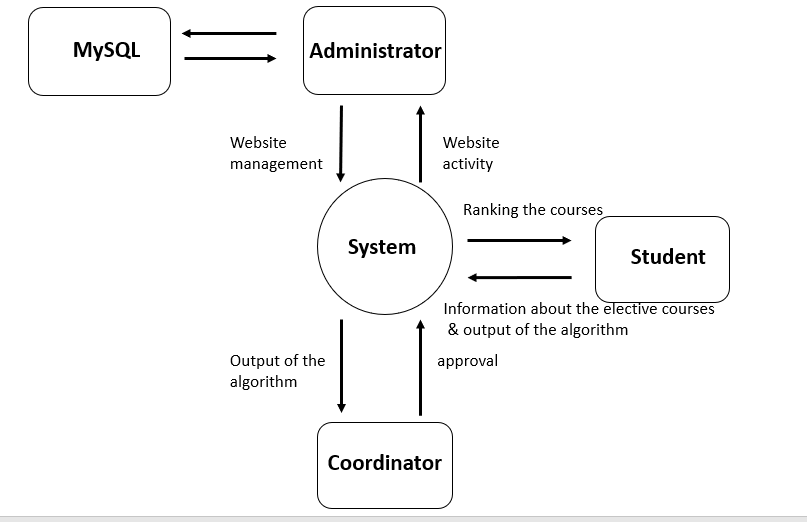
3. SYSTEM ARCHITECTURE

User Interface (=UI) Data & Processing

**Back-endd**

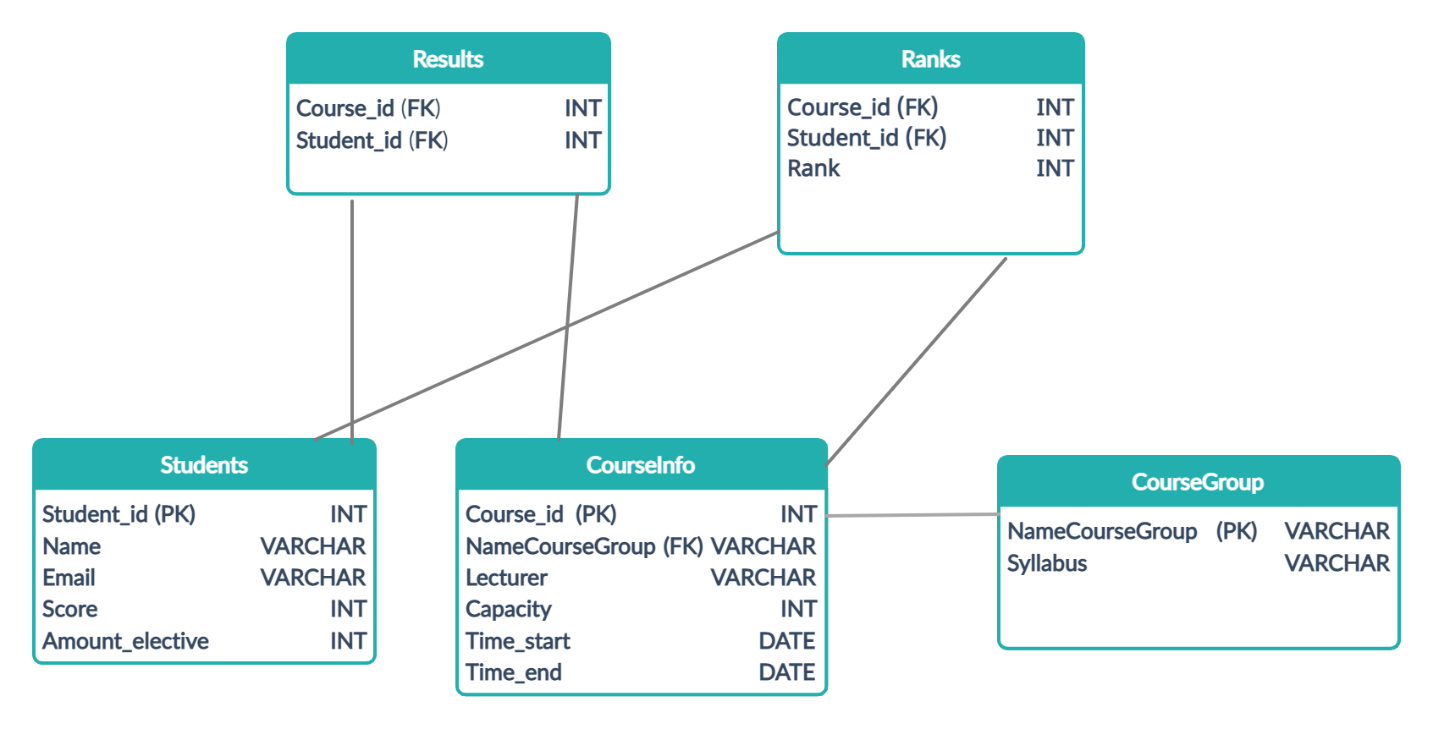
**Front-endd**

3.2 Decomposition Description



**Django**

4. DATA DESIGN



5. HUMAN INTERFACE DESIGN

Overview of User Interface

There will be two types of users with our software:

Student:

First, the student will log in by entering the ID and the password that has been given by the University. Second, the student considers the course information (=such that capacity, schedule, syllabus, lecturer, etc. ) given by the student coordinator and ranks the elective courses as he/she sees fit, and this rank will be editable until the deadline the students coordinator will provide. Third, after the algorithm will enroll the student in courses, he/she will see the enrollment outcome.

Students Coordinator:

First, the coordinator will define registration opening dates and deadlines.Second, the student coordinator will provide the course information as noted above. Third, after the algorithm will enroll the student in courses, the coordinator will approve the enrollment outcome.

Screen Images

Has been in the PowerPoint file it sends with the SRS file

6. Division Of Work

Lihi: will be responsible for the FrontEnd.

Itay: will work on the database and will define the models.

Joseph: will work on the algorithm.