Jorge Blanco Rey
Ángel Casanova Bienzobas
S.Xiao Fernández Marín

jorge.blancor@estudiante.uam.es
angel.casanova@estudiante.uam.es
sofiax.fernandez@estudiante.uam.es

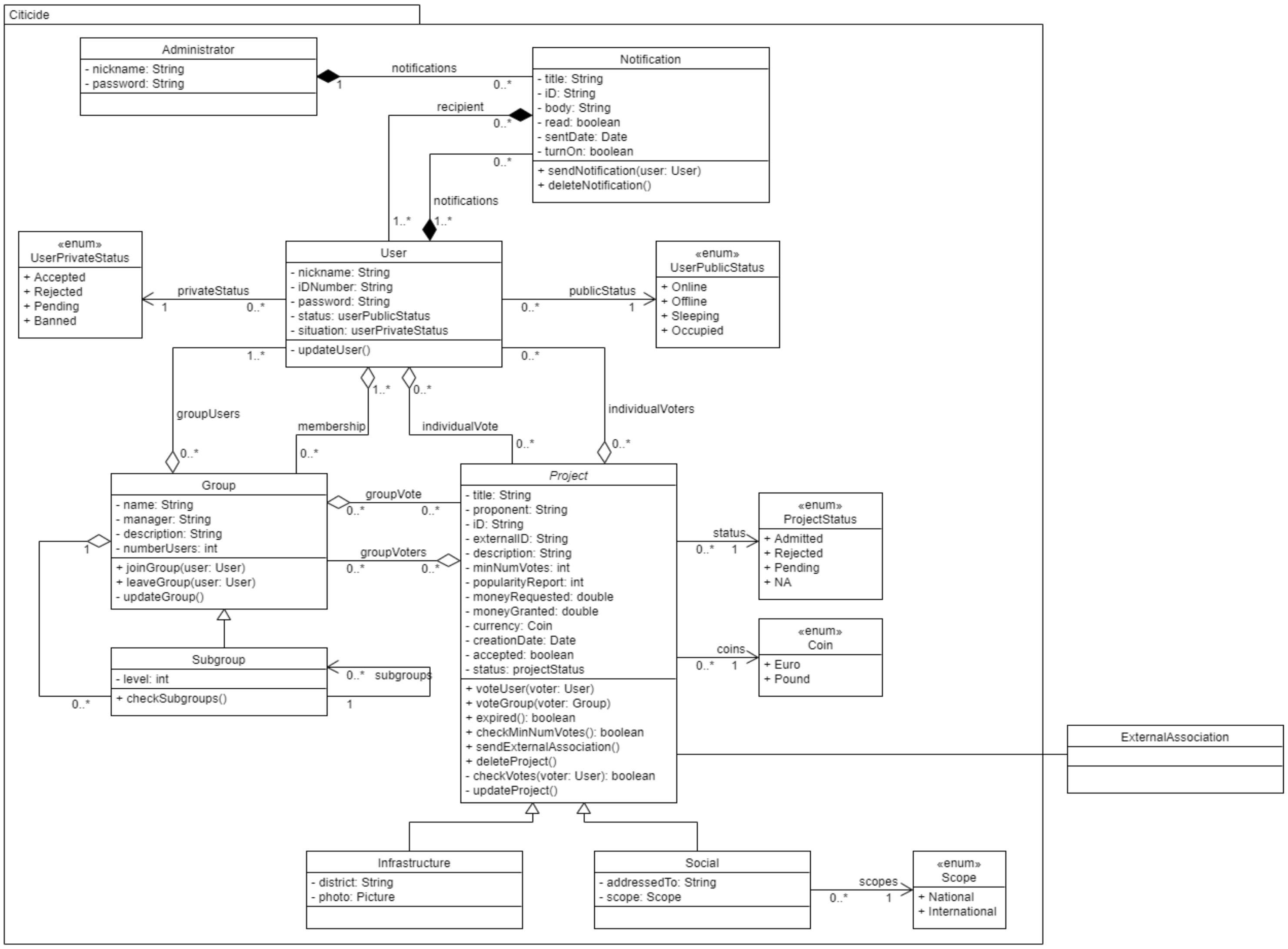
Object Oriented Design Document

Application: Citicide

Date: 03/03/2020

Index

1. Class diagram



This class diagram has several classes.
The Administrator, once that he has as attributes a string for the name, the password and an array of notifications of the users he sended the message. The Notification class has a String of characters for the title and the ID the body. It has a boolean attribute to check if it has been read and another one to let the users set the notifications on or off. It also has a date area and an array that contains all the users that received that notification.

User is another class that is composed of a string for the name, the id and the password. They also have to enumerations that are relationed to them, one is to set the private status they have (if they are accepted, rejected, pending or banned) and another one to set their public status (online, offline,..)

The user also has an array of the groups he belongs to and another where all the projects he voted for, are stored.

The Group class has a String of characters for saving the name of the group, the manager and the description. It also has a counter to know how many people there are in that group and an array to know in which project it has voted as a group. In the methods we find that we have one to create or leave the group just as one to update the group.
The Group class also has an array to set all the subgroups that group has. We created a Subgroup class too to do this. This class inherits from Group all its attributes and methods, and has a new one called level, because the users can only be members of siblings groups. The subgroups can have as many subgroups as they want but a subgroup can only have one parent.

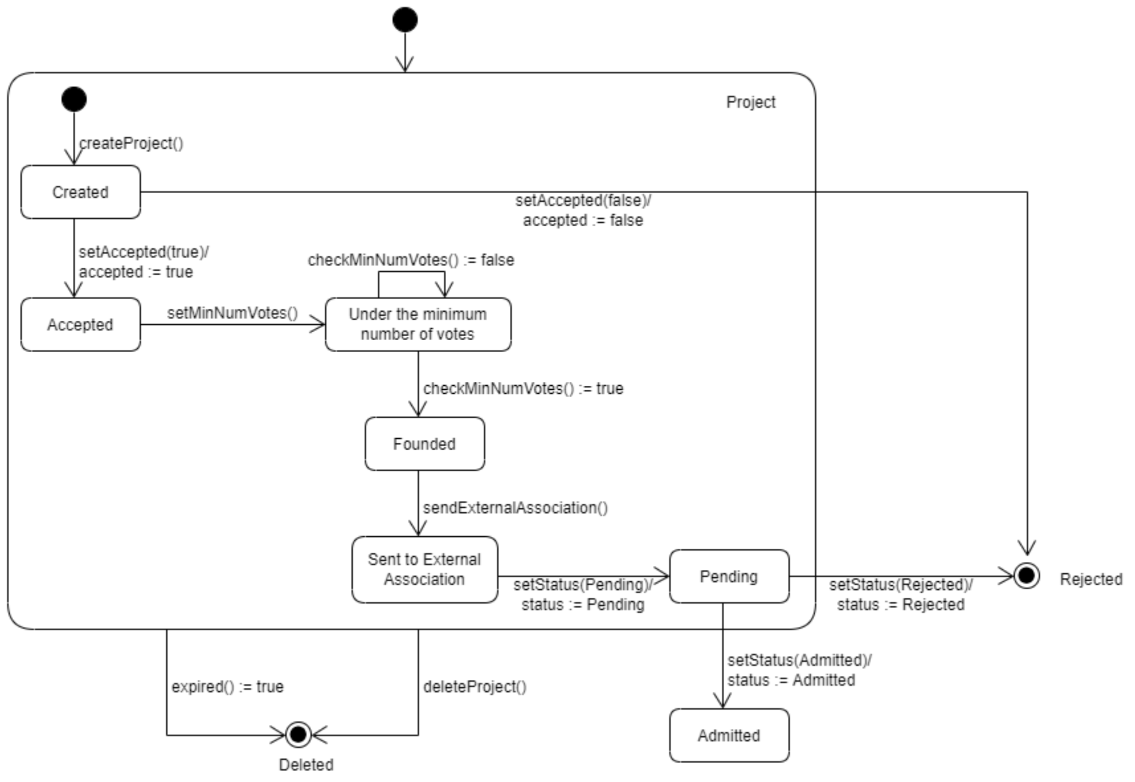
Project is the class created to store all the information that is relative to the projects. This class has a string for saving the title, proponent, ID, external ID that is returned by the external association and the description. There are integers to save the minimum number of votes that the administrator of the app will set if he approves the project, and to store the number of votes the project has if it is approved. It also has two double variables, one to store the money requested and another one to save the money granted. We put an enumeration to change the coin the amount of money (Euro or Pound) is and a creation date. If it is accepted or not it is stored in a boolean variable and the status of the project is saved on a enumeration (Admitted, rejected,...). It also has two arrays, to store the individual voters and the group voters.

The project class has inherited the Infrastructure and Social class. The Infrastructure class has a string to store the district we want to perform the project and a matrix that will be a picture of the project. The Social one has a string to whom it is addressed to and an enumeration to set if it is a national or international project.

In this whole implementation, we are going to duplicate information among the different clauses, by generating doubles associations between them. We know that this could appear tautological, but we reckon that this design decision is worthy, because we generate a quite important improvement of the system performance

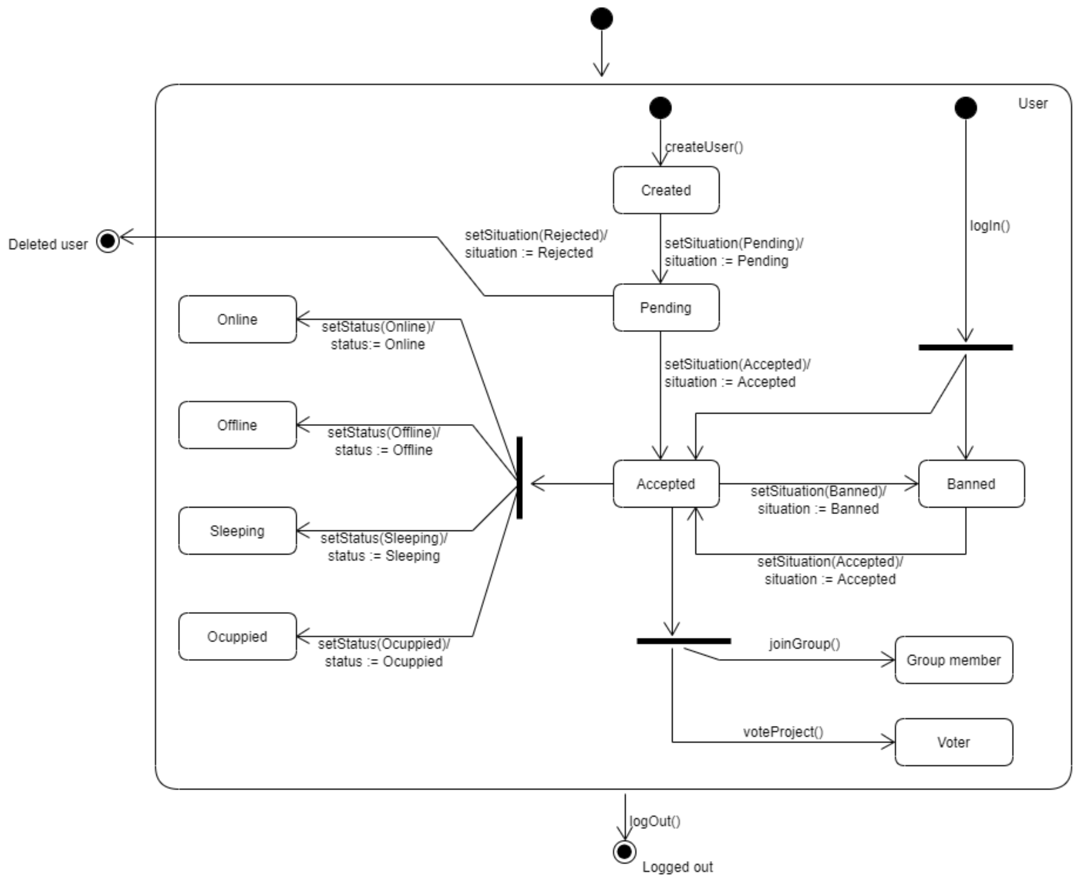
2. State transition diagrams

2.1 Project



In this diagram we can observe all the paths related to a project. The first step, is to create a project, from this point, the project can be rejected or accepted by the admin, in this case, he should establish a minimum number of votes for each particular project.
In this state of the diagram, the project is visible to anyone that wants to vote for it, and will stay in “Under the minimum number of votes” until enough users would support it. The next step on this procedure, is to wait for the proponent to send the project to an external association, once there, this association can reject it or admit it.
Besides, if the time threshold is over, independent of the state of the project at that time, it will be Deleted.

2.2 User



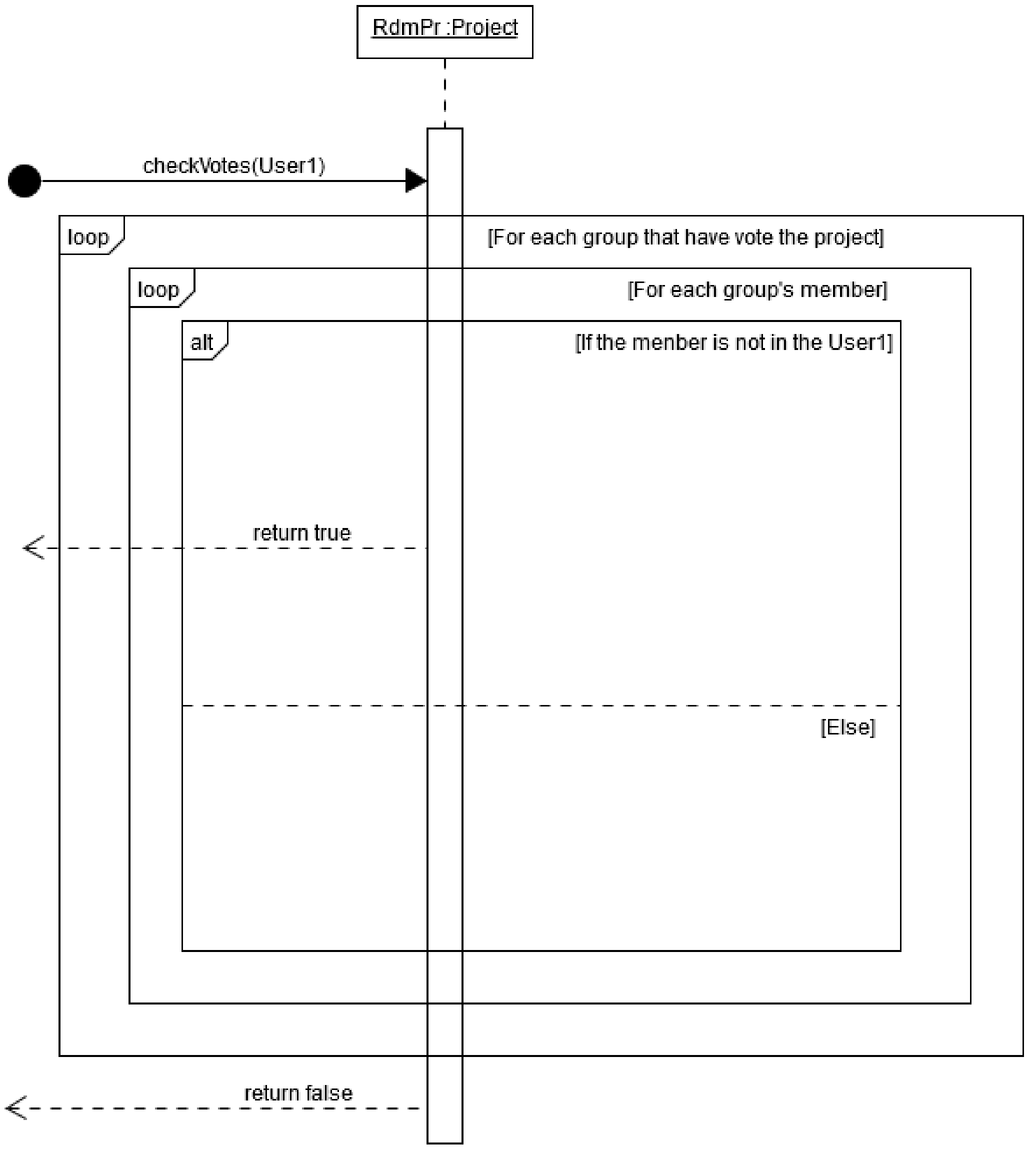
In this diagram we can see the different states that a user can have. Once a user is registered, "Created" state, he is in the "Pending" state waiting for the approval or rejection of the administrator, then he passes to one of these states. After that, if the administrator considers it, he can ban the user, so he passes to the "Banned" state, because of this, when a user logs in he can be "Banned" or "Accepted". It is important to remark that a user in the "Banned" state cannot do anything.
Then, when the user is in the "Accepted" state, he can change his status to the four different ones and can also become a "Group member" when joining a group or "Voter" when voting for a project as an individual. We should not forget that although an "Accepted" user can change his state to the ones mentioned in this paragraph, he does not leave the "Accepted" state. Also, when a user is "Banned", his states will be left as they were the last time he was in the "Accepted" state and will not be changed until he returns to this state.
Finally, from all the possible states previously mentioned the user can log out.

Jorge Blanco Rey
Ángel Casanova Bienzobas
S.Xiao Fernández Marín

jorge.blancor@estudiante.uam.es
angel.casanova@estudiante.uam.es
sofiax.fernandez@estudiante.uam.es

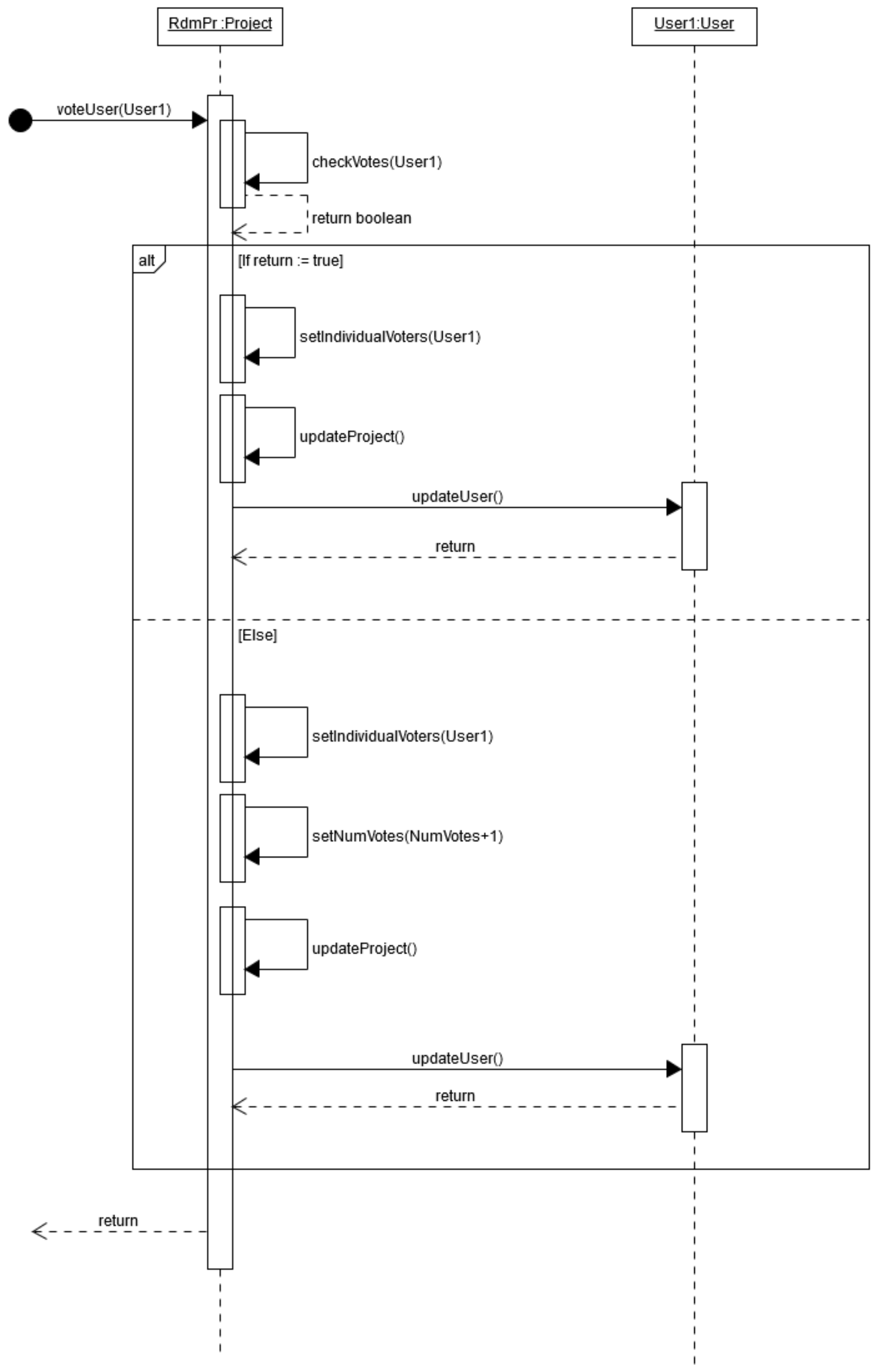
3. Sequence diagrams

3.1 checkVotes


RdmPr :Project
checkVotes(User1)
loop
[For each group that have vote the project]
loop
[For each group's member]
alt
[If the menber is not in the User1]
return true
[Else]
return false

This sequence diagram develop the scenario where the system has to determine if a user has already vote to a project since he is a group's member.
First, the method checkVotes() iterates the collection of groups that have supported the project, then for each group, it iterates again in each container of members in order to compare each of them with the "User1", who is the one that is passed as argument(the new voter). If the "User1" belongs to any of the group, then the function returns true, otherwise it will return false.

3.2 userVote


RdmPr :Project
User1:User
voteUser(User1)
checkVotes(User1)
return boolean
alt
[If return := true]
setIndividualVoters(User1)
updateProject()
updateUser()
return
[Else]
setIndividualVoters(User1)
setNumVotes(NumVotes+1)
updateProject()
updateUser()
return
return

This other sequence diagram, shows the scenario where a user wants to vote to a project as an individual, then, the method userVote() first make a call to checkVotes() and depending on the return of this subroutine, it update the number of votes of the project or not.
Both paths in the "alt"(conditional) square, first, includes the "User1"(the new voter), to the array of individual voters of the group and at the end of the method, they both update the collections of the project and the one in user.

As we are duplicating information by duplicating information among the classes, in the aim of gaining performance, it is highly important to call the update functions due to, those functions are ones that syncronis the system.

4. Traceability matrix

In this matrix we can see all the requirements of our project and all the class methods we created. We marked with an X all the methods that have to do with the requirement of the queue.
There are some requirements that do not have any mark, that is because we use as methods setters and getters, that were not asked to be added on the diagram.
The first requirement is that "only the administrator can ban or unban a user" has updateUser and sendNotification because you need to change the user class and you need to send a notification saying why he was banned. The requirement "only the administrator can reject a petition to join the app" as well as "Groups are created by users" have the updateUser method.
"Only the manager of a group can create a subgroup" needs updateGroup to change the number of people and the subgroups there are in the main group. The users are also allowed to be in several groups if they are siblings so we need a method called checkSubgroups that compares the level of each subgroup the user wants to join.
The users can unsubscribe from a group and for that we will need the methods upadateUser (for updating the numbers of groups he belong to), leaveGroup (that will call updateGroup) and updateGroup that will change the array of the persons that are in that group.
All users can receive notifications if they want to. That notification will be sended by the administrator of the application using the sendNotification method, so they all have the option to turn the notifications on or off (except from the proponent of a project, that will always have the notifications on for the project he created). The users can also delete the notifications with the deleteNotification method.
A project is done by a user, so if we want to implement that, we need updateUser to update the array.
Only the administrator of the application can approve or reject a project.
If the project is accepted, the user becomes a proponent and we will need to change the array of the projects he is proponent, as well as if there was a group that will want to create a project. The votes of all the persons of that group will be set to the project, and in

case that there was the minimum number of votes for the project, the project will be sended to the external association.
The administrator is also in charge of setting the minimum number of votes when he accepts the project. If he accepts the project, he will need to send a notification to the proponent.
If a project has not reached the minimum number of votes in 30 days the project gets expired and it gets deleted. If it has, the proponent of the project is in charge of sending it to an external association.

When voting, you can do it as an individual or as a group, in any case, the vote or votes will be added to the project and the array of users for the projects he has voted will be updated.
If you vote as a group, you will need to specify which group you are voting for to update the group votes and add the votes to the project.
And if you vote as an individual, the application will need to check if you have voted as a group using checkVotes, if that is the case you can not vote again.

Then, if a user gets banned, we would have to update all the projects in order to remove all his votes, but all the information about the user is stored in the system, since it can be the case that the user is unban and then all the votes will be returned to the proper projects. In a similar way, if an user leaves a group, all the projects that are supported by this group will lose one vote.
Finally, we show in the matrix that we also have implemented the popularity and the affinitive reports, the first one, return the number of votes of the project if the user who asked for it has previously voted as an individual for the project and the last one just computed a coefficient related to the “relativity association” between two groups.