Dario LORENZONI

Raffele NOCERINO

Marco PETTORALI

Riccardo XEFRAJ

Workgroup Manager

Task 3

Index

[Description of the system 2](#_Toc42017582)

[System requirements 2](#_Toc42017583)

[Functional requirements: 2](#_Toc42017584)

[Non-functional requirements: 3](#_Toc42017585)

[Main actors 3](#_Toc42017586)

[Use Cases 3](#_Toc42017587)

[Analysis Class Diagram 4](#_Toc42017588)

[Data model 4](#_Toc42017589)

[Classes 6](#_Toc42017590)

[UML Diagram 8](#_Toc42017591)

[Neo4j: 9](#_Toc42017592)

[Suggested Working Group: 9](#_Toc42017593)

[Leaders Ranking: 10](#_Toc42017594)

[User manual 13](#_Toc42017595)

[Login 13](#_Toc42017596)

[Working with the application 13](#_Toc42017597)

## Description of the system

The main purpose of our project is to provide a useful system for creating, managing and coordinating some working groups that consist of some users that have to realize a project. The administrator of the system or rather the boss of the company has his own administrator panel where he can add new user, delete a user or create a new working group specifying also his dimension in terms of working user.

The administrator will also specify the leader of the new working group selecting it from all the users of the system. The system will help the administrator on this choice giving a rank to the users, in order to do so the system will use some information about the past group headed by the user. The leader of each group will receive some application from the users for the team previously created and he can accept or discard them based on the number of works on which the users are already working or other information.

Instead, the user accessing its own panel can visualize on which job he/she is already working, the suggested working groups (with at least one position available) and some information about the completed job. Furthermore, the user can visualize all the groups that he/she lead (separated from the ones in which he/she works but that he/she doesn’t lead). A user can report the completion of his work for a team. If a user wants to join a new working group, he/she can send an application to the leader of the group and wait for his/her acceptance.

## System requirements

### Functional requirements:

#### USER:

* A user can login to the system using a login with the credential given by his/her;
* A user can see in the application all its works in progress with the information about the working groups, its passed works and the working group suggested;
* A user can mark his/her part of the work for a specific working group as complete;
* A user can send an application for being part of a suggested working group;
* A user can see the applications from other users to the working groups that he/she leads;
* A user that is the leader of a working group can accept the applications from other users to the working group that he/she leads.

#### ADMINISTRATOR:

* The administrator can login to the service;
* The administrator can view the list of all users;
* The administrator can add a new user putting the personal information in a sign-up service obtaining the username that will allow them to use the application;
* The administrator can create a new team setting its name and its dimension (number of users allowed) and select the leader of the new group;
* An administrator can delete a user from the system;
* The administrator can visualize the users’ rank

### Non-functional requirements:

* Quality: every time an error occurs the system will show an error message which explain the problem's reason;
* Response Time: the database chosen and used by the application is developed in order to get a low response time;

## Main actors

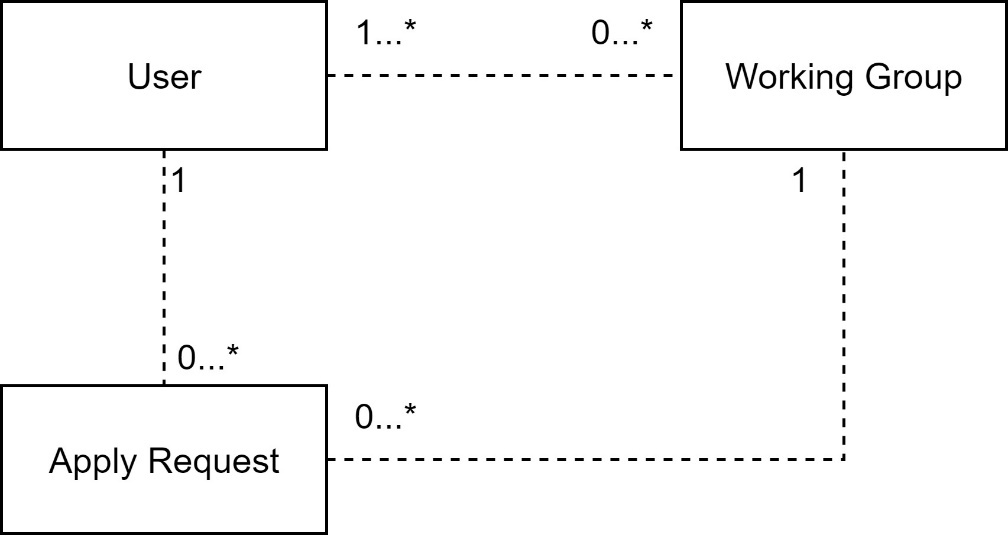
* User
* Administrator

## Use Cases

Immagine che contiene testo, mappa

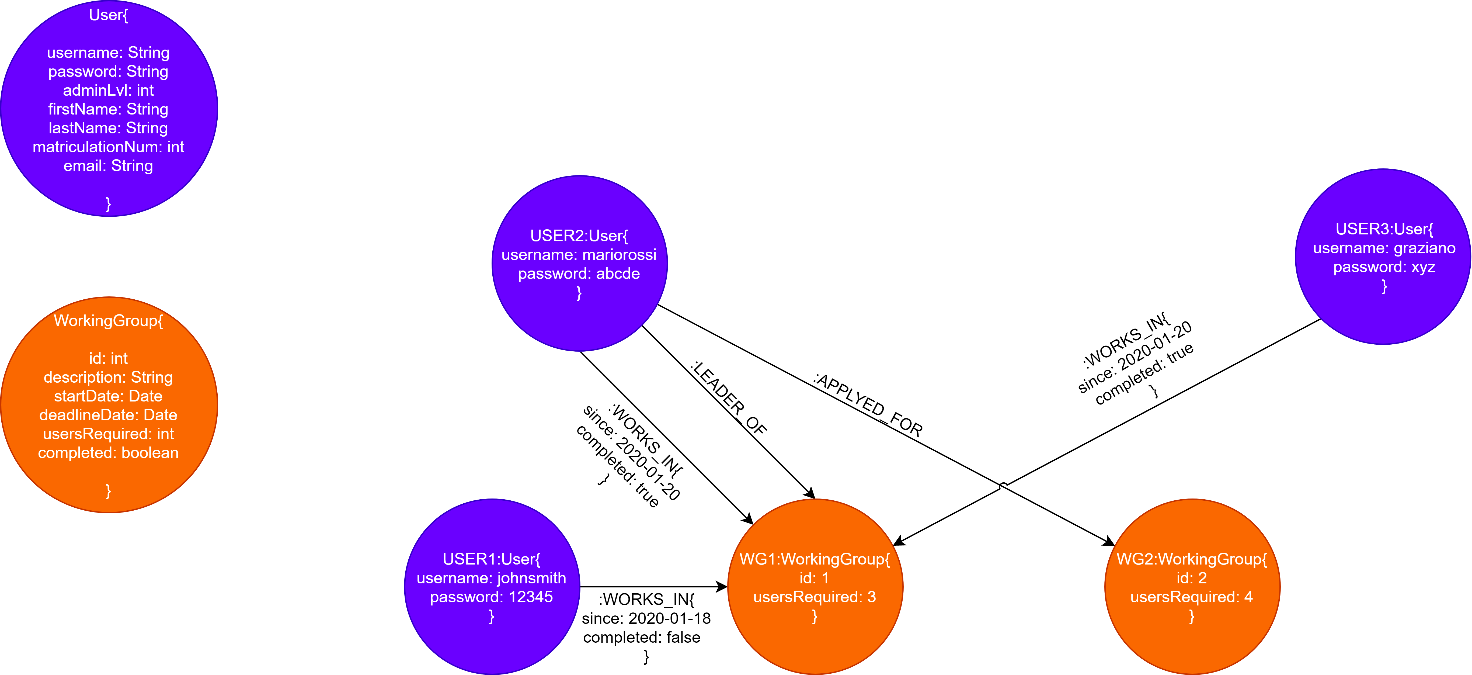
Descrizione generata automaticamente

## Analysis Class Diagram



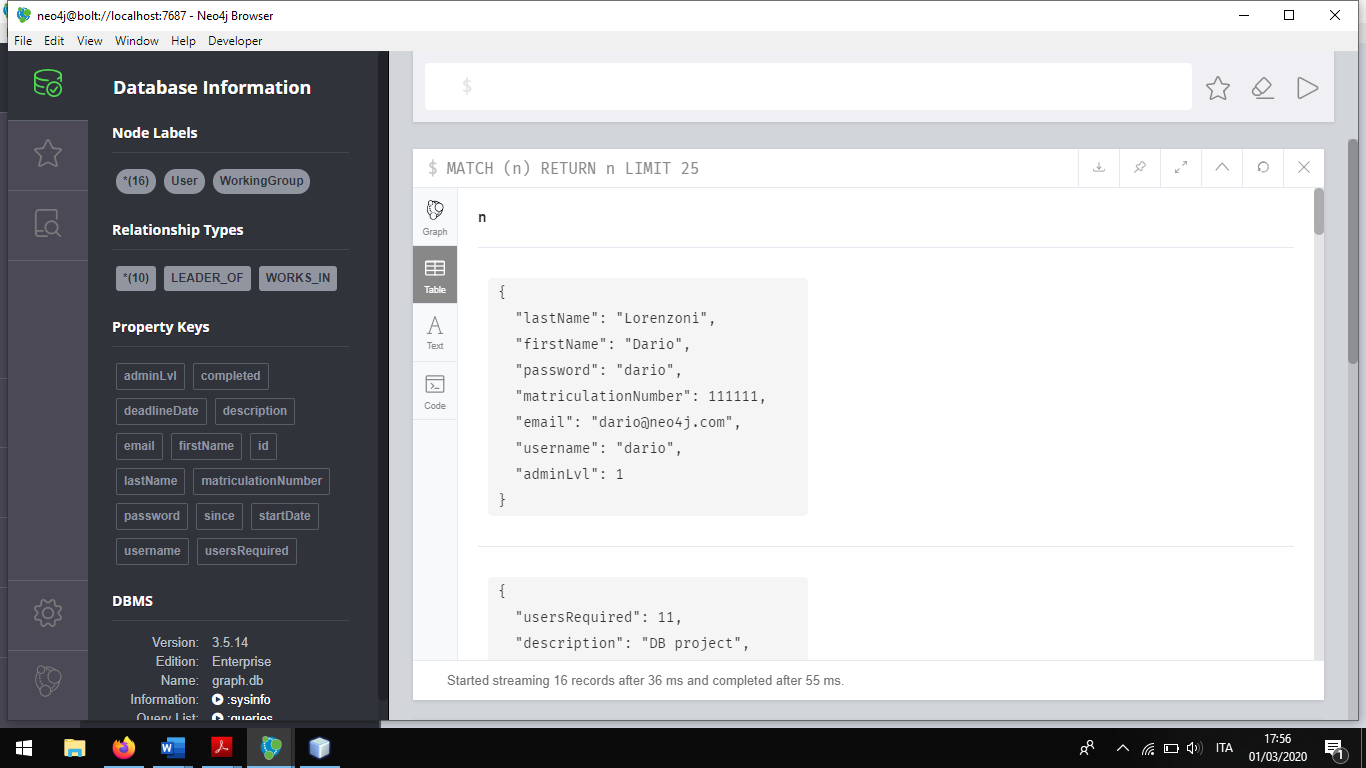
## Data model

In our application we have used a graph database, called Neo4j, that allow us to obtain some important advantages. We have created a database that store data in the best suitable manner for our proposal obtaining high performance representing our domain as a network of connected entities and storing some information about the relationship between different entities. An example of a part of our DB represented as a graph is the following:

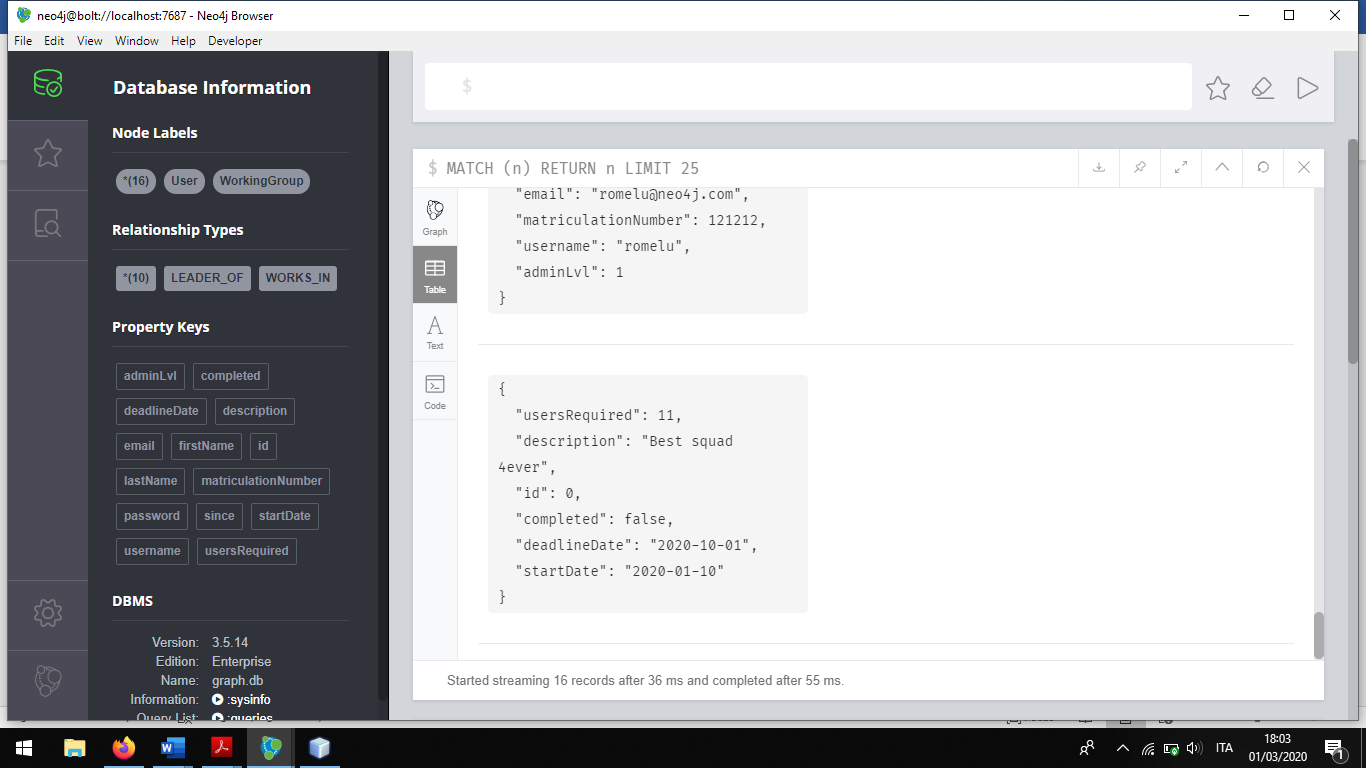


In our database, called Graph, we store 2 types of nodes :

* **User**: in a node of this type we store the information about the users of the system. A tabular example of the information stored in this node is the following:



* **Working Group**: in this node we store the information about the working groups created by the administrator. Another tabular example of the information stored in this type of nodes is the following:



We also have 3 types of relationship between users and working groups:

* **WORKS\_IN:** this relationship collects the information about the actual state of the part of the work assigned to a user in a certain group. This relationship has two attributes: “since”, representing the date on which the user joined the team and “completed”, that is set to true as soon as the user marks his part of the work in that group as completed.
* **APPLIED\_FOR:** this relationship represents the request of a user for working in a certain working group. This relationship has the “since” attribute, representing the date on which the user decided to apply for a certain working group.
* **LEADER\_OF:** this relationship is established between a working group and its leader. This relationship has no attributes.

## Classes

|  |  |  |
| --- | --- | --- |
| Front-end |  |  |
| **AdminPaneGUI** | This class implements the admin control panel in which the administrator can add or remove a user, can add a new group with its leader and visualize the user’s rank. |
| **LoginPaneGUI** | That class implements the panel that can be used to perform the login |
| **UserPaneGUI** | This class implements the user panel where the user can see all the information about all its works in progress, its passed works and the not full working group suggested. Moreover, the user can mark a work as completed, send an application for a working group and eventually visualize and accept the request for joining an application that (s)he leads. |
| **ApplicationWorkingGroupTable** | This class implements the table present in the user panel used to visualize all the application for the groups that the user leads. |
| **Ranking Table** | This class implements the table that the administrator visualizes in which each user and its associated rank value is reported. |
| **SuggestedWorkingGroupTable** | This class implements the table present in the user panel in which are reported the list of the suggested working groups and their information. |
| **UserOverviewTable** | This class implements the table present both in the user panel and in the administrator panel in which the users’ information are shown. |
| **WorkingGroupTable** | This class implements the table present both in the users’ panel and in the administrator panel that is used in order to visualize the information about the working groups. |
| Middleware | **MainApp** | MainApp is the main class of the application. It contains the main method and the methods for the initialization of the application. |
| Back-End | **ApplicationWorkingGroup** | This class represent an application for a working group with its related information. |
| **Neo4JManager** | That class contains all the method used to communicate with the database and used to perform the queries. |
| **SuggestedWorkingGroups** | This class represent a suggested working group with all its information. |
| **User** | This class represents a user with all his/her information. |
| **UserRanking** | This class represents the ranking of a user and the associated username. |
| **WorkingGroup** | This class represents the working group and all its information |

## UML Diagram

Immagine che contiene testo

Descrizione generata automaticamente

## Neo4j:

The database used in our application is Neo4j, we have used this graph database since it is suitable to represent our data as a network of connected entities that are linked with relationships. Moreover, some specific operation performed in our application require to rapidly traverse the path between users and working groups. This application was designed having in mind a small/medium enterprise where the number of users, project carried out and the dimension of the working group is limited and this help us to overcome the limitation of the graph database that in general are difficult to scale up. The main queries that benefits from the usage of a graph database are the following:

### Suggested Working Group:

As mentioned before one of the operations that our application need to perform is show to a specific user the list of the suggest working group, in order to identify those working group we perform the following query on the database:

String query = ""+ "MATCH (u1:User)-[:WORKS\_IN]->(w1:WorkingGroup)<-[:WORKS\_IN]-(u2:User)-[:WORKS\_IN]->(w2:WorkingGroup) "

+ "WHERE u1.username = $name "

+ "AND NOT (w1 = w2) "

+ "RETURN DISTINCT w2.id, w2.description, w2.startDate, w2.deadlineDate, w2.usersRequired, w2.completed, count(w2.id)";

In this case the $name will be properly set afterwards to the username of the specific user. So, in order to retrieve the suggested working group, the query searches for the groups with at least a member that shares a group with our user and returns it. Moreover, in order to give an affinity degree, the query counts for each working group how many of its members have been “working group mates” with our user. If instead we implemented our database as a relational database, the part interested by this query would be the following:

Immagine che contiene screenshot

Descrizione generata automaticamente

And the query just exposed above can be translated in SQL as follow:

SELECT WG.\*,COUNT(\*) as AffinityDegree

FROM working\_group\_member as W1

**JOIN** working\_group\_member as W2

ON W1.id\_working\_group = W2.id\_working\_group /\*I need the first join in order to get the members of the user's working groups\*/

**JOIN** working\_group\_member as W3

ON W2.username = W3.username /\*I need the second join in order to get the working group into which the user find above work\*/

**JOIN** working\_groups as WG

ON WG.id\_working\_groups = W3.id\_working\_group

WHERE W1.username = 'username' AND

NOT W2.username = 'username' AND

W3.id\_working\_group NOT IN(

/\*I need in order to find the group in which our user dosen't work\*/

SELECT W4.id\_working\_group

FROM working\_group\_member as W4

WHERE W4.username = 'username'

)

GROUP BY W3.id\_working\_group;

As we can see the same query that in Neo4j was very simple become instead more complex. In particular we have to use 3 different join operations and one nested table and this justifies our choice of using a graph database.

### Leaders Ranking:

The system will also help the administrator to choose among all the possible users the ones that are more suited for leading a group, this is done by computing a ranking degree for each user that at least once was elected leader of a group. The ranking degree is computed as the average of the ratio between the number of the members of the working group that (s)he leads and the number of requested users of that group; we measure how much users are encouraged to join a working group with the current user as leader. This results in a complex function that can be divided into three parts:

1. For each user we first find the list of working groups that (s)he leads; in particular we create a map that associate each username to the list of those working groups, and this is stored inside the variable *user\_group:*

public static Map<User, Double> loadLeadersRanking() {

try (Session session = driver.session()) {

Map<User, Double> ret = new HashMap<>();

Map<User, List<WorkingGroup>> user\_group = new HashMap<>();

Map<Integer, Double> group\_percentage = new HashMap<>();

Map<String, Object> params = new HashMap<>();

session.readTransaction((Transaction tx) -> {

String query = "MATCH (u:User)-[:LEADER\_OF]->(w:WorkingGroup) "

+ "RETURN DISTINCT u.username,u.password,u.adminLvl,u.firstName,u.lastName, "

+ " u.email "

+ ",w.id,w.description,w.startDate,w.deadlineDate,w.usersRequired,w.completed ";

StatementResult sr = tx.run(query);

while (sr.hasNext()) {

Record rec = sr.next();

int id = rec.get(6).asInt();

String descr = rec.get(7).asString();

String d1 = rec.get(8).asLocalDate().toString();

String d2 = rec.get(9).asLocalDate().toString();

int userReq = rec.get(10).asInt();

boolean compl = rec.get(11).asBoolean();

WorkingGroup temp = new WorkingGroup(id, descr, d1, d2, userReq, compl);

String usern = rec.get(0).asString();

String passw = rec.get(1).asString();

int adminLvl = rec.get(2).asInt();

String first = rec.get(3).asString();

String last = rec.get(4).asString();

String email = rec.get(5).asString();

User u = new User(usern, passw, adminLvl, first, last, email);

boolean b = false;

for (Map.Entry<User, List<WorkingGroup>> entry : user\_group.entrySet()) {

if (entry.getKey().getUsername().equals(u.getUsername())) {

b = true;

entry.getValue().add(temp);

break;

}

}

if (!b) {

user\_group.put(u, new ArrayList<WorkingGroup>());

user\_group.get(u).add(temp);

}

}

1. Then for each user we take into consideration all its working groups, and we first compute the ratio between the number of actual members in the group and the members needed. Then we compute the average of this ratios. And then, after this second point we have computed the leader rank for each user.

String query2 = "MATCH (u:User)-[wi:WORKS\_IN]->(w:WorkingGroup) "

+ "RETURN DISTINCT w.id,w.usersRequired,count(wi) ";

StatementResult sr1 = tx.run(query2);

while (sr1.hasNext()) {

Record rec = sr1.next();

int id = rec.get(0).asInt();

double req = rec.get(1).asDouble();

double count = rec.get(2).asDouble();

double percentage = (count / req) \* 100;

group\_percentage.put(id, percentage);

}

return 1;

});

for (Map.Entry<User, List<WorkingGroup>> entry : user\_group.entrySet()) {

double somma = 0;

List<WorkingGroup> appoggio = entry.getValue();

for (int i = 0; i < appoggio.size(); i++) {

somma += group\_percentage.get(appoggio.get(i).getId());

}

somma = somma / appoggio.size();

if (somma != 100) {

ret.put(entry.getKey(), somma);

}

}

1. Finally, we order the list of users belonging to the system in the leader ranking table, by descending order and return it to show this list to the administrator’s panel.

List<Entry<User, Double>> list = new ArrayList<>(ret.entrySet());

list.sort(Entry.comparingByValue());

Map<User, Double> result = new LinkedHashMap<>();

for (Entry<User, Double> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

return result;

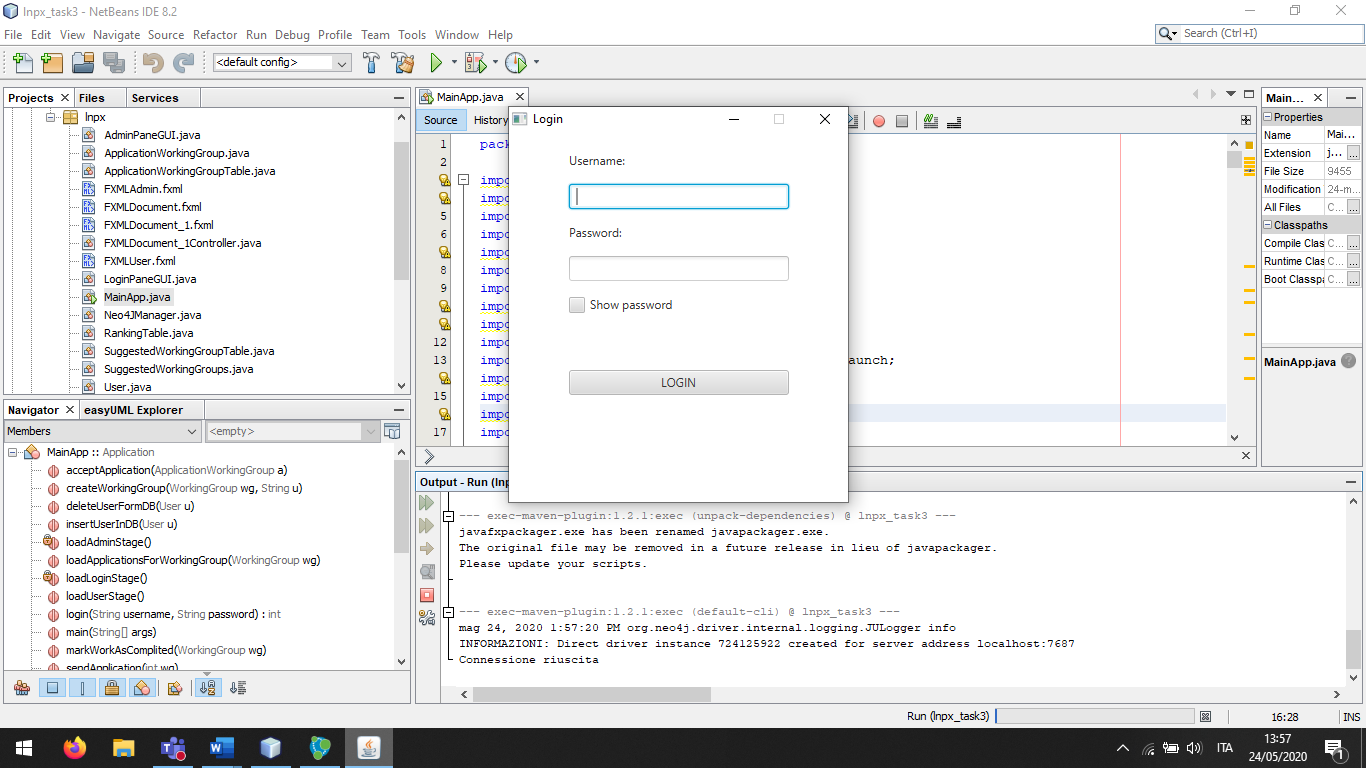
}

}

## User manual

### Login

When the application is launched, the system will display a login form. The user has to put in its username and password and then to click on “LOGIN” button. If desired, the user can also see its password as plain text by clicking on the “Show/Hide Password” radio button.



### Working with the application

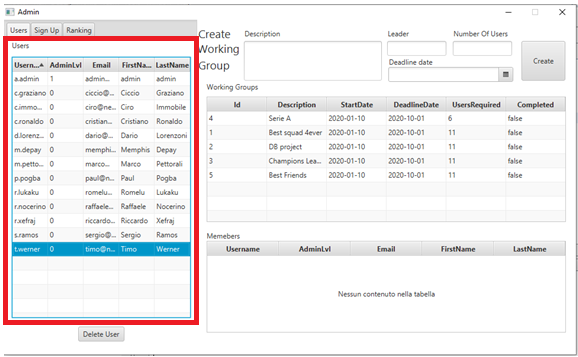
#### Administrator case

If the user is the administrator of the application, the main page showed will be the administrator control panel in which the administrator can:

* Visualize the list of the user currently present in the database
* Delete a user selecting it from the list
* Add a new user through the appropriate from
* Visualize the ranking of the users
* Add a new working group
* Visualize the list of the working groups and their members.

#### Visualize the list of the user:

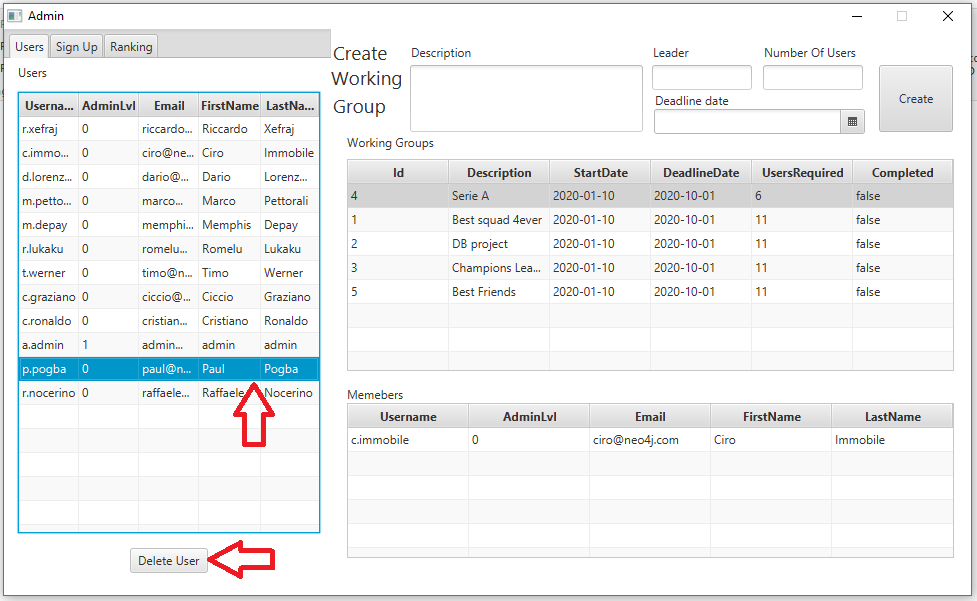
The list of the users in the system is shown on the left side of the administrator panel as highlighted in the following figure:



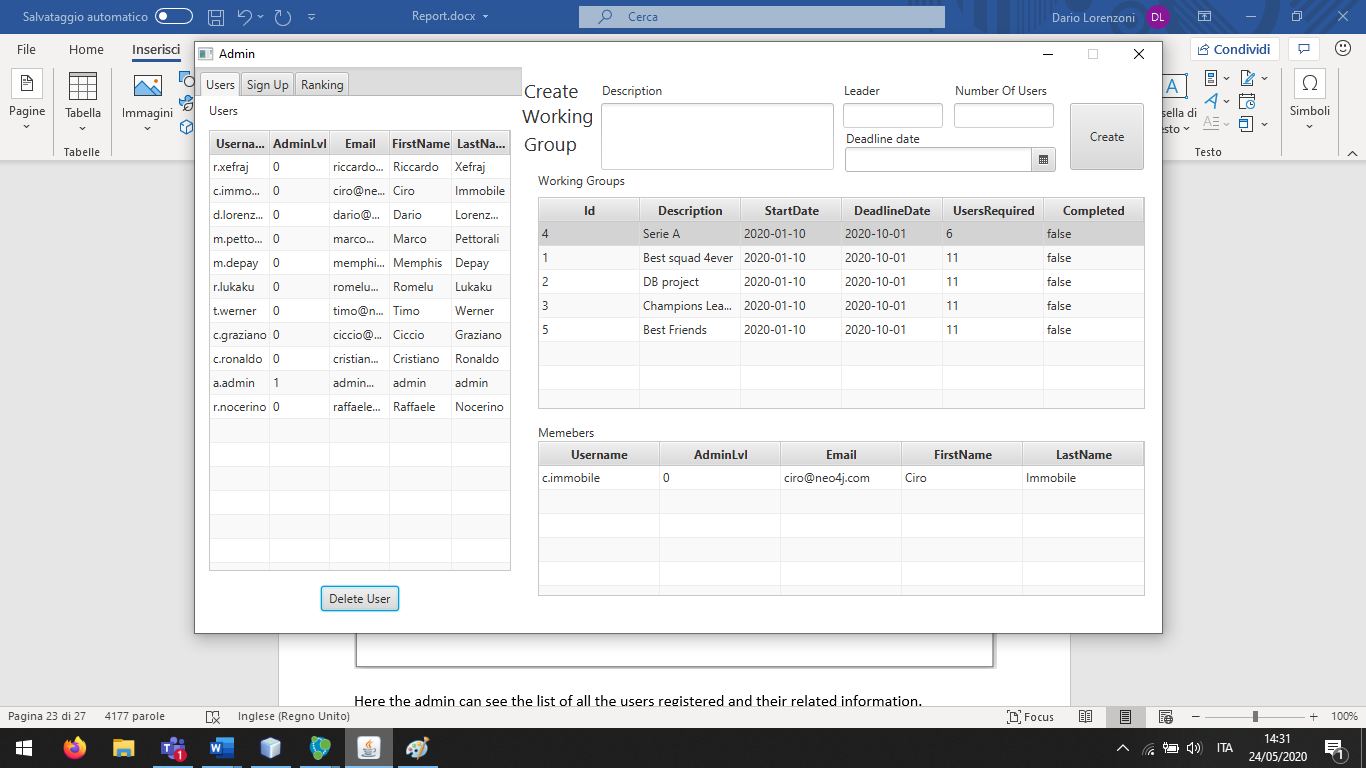
If the above table with the information of the users is not shown, the administrator can click in the label Users in order to show it.

#### Delete a user:

The administrator can also delete a user selecting it from the list of the users just shown and clicking on the delete button as it is shown in the following picture:



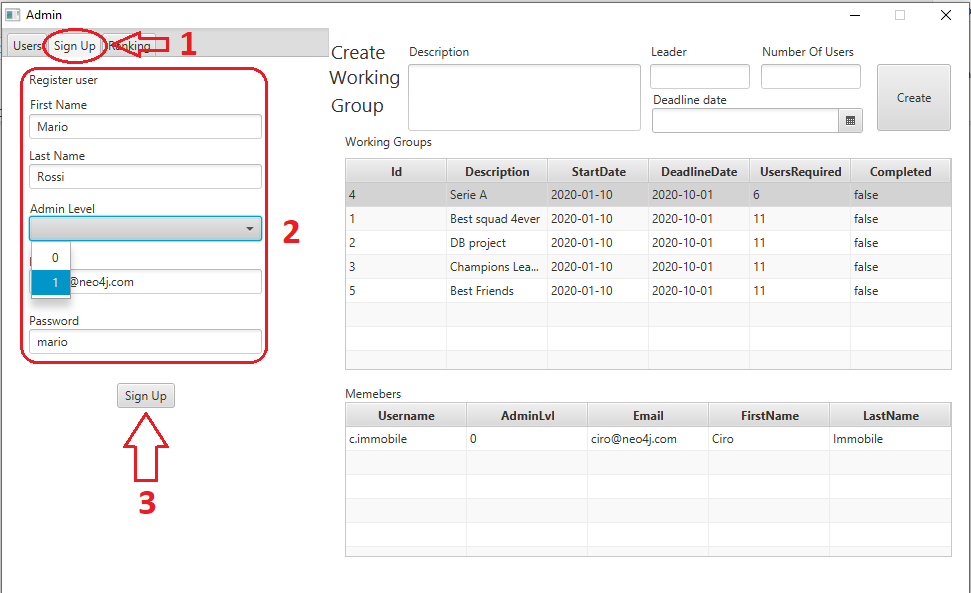
The list of the user before the deletion of a user



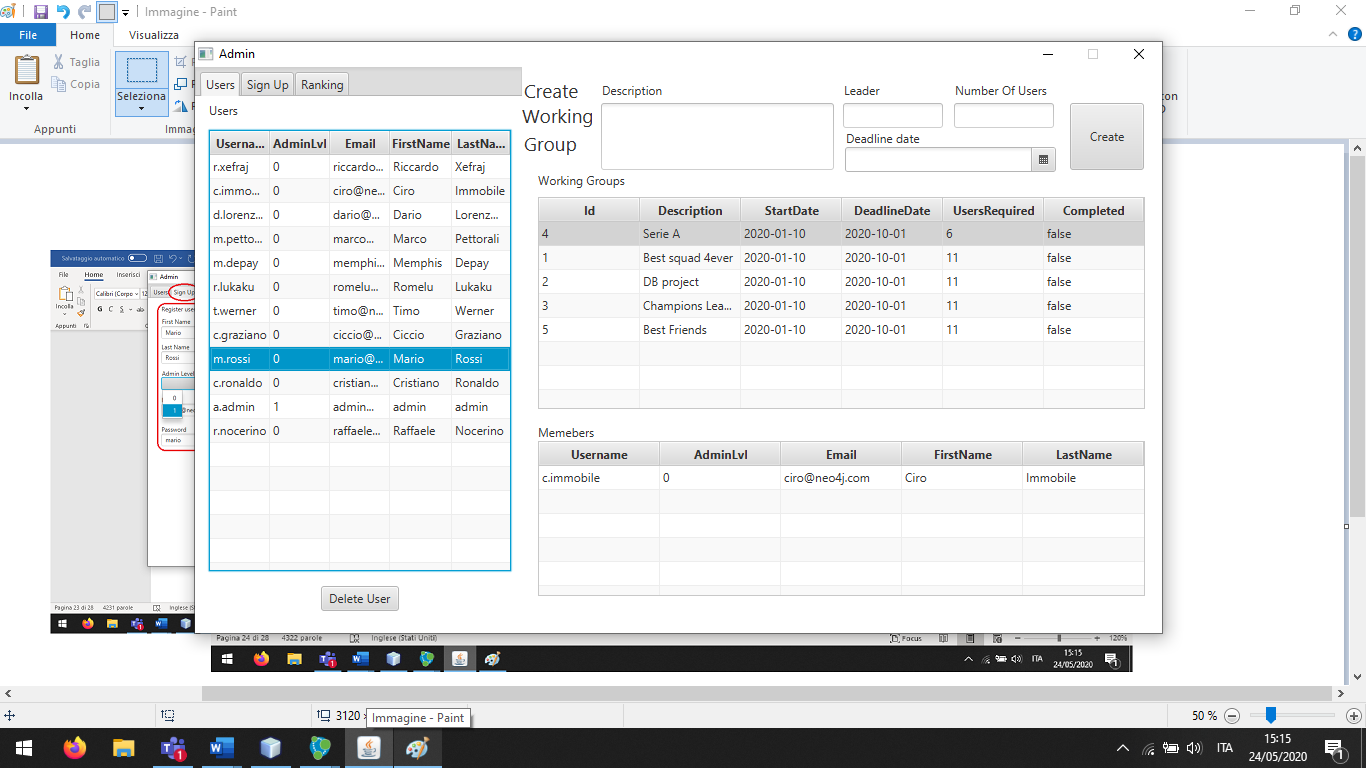
The list of the users after the deletion of the selected user

#### Add a new user:

The administrator can add a new user; in order to do so, it first has to click on the Sign Up in order to show the form that can be used in order to add a new user, then it has to fill all the fields with the information of the new user, in particular the Admin Level (as shown in picture) can be set to 1 if the new user is an administrator, or 0 otherwise. Finally, the administrator can click on the sign up button in order to add the new user. If something is missing an error message will be shown.



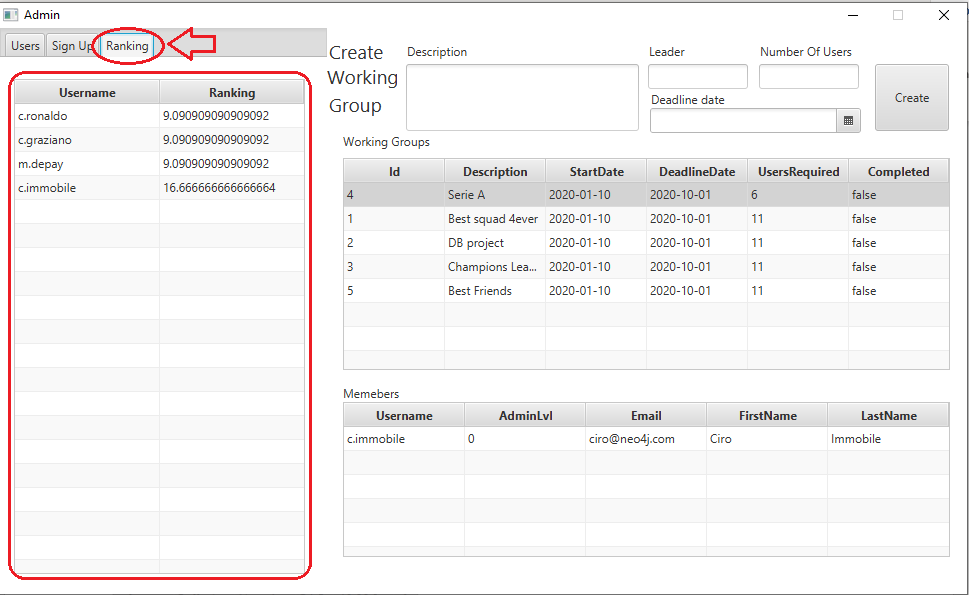
If the creation of the new user is correctly performed, the system will show the message: “User saved!”.



The list of the user after that the user is added

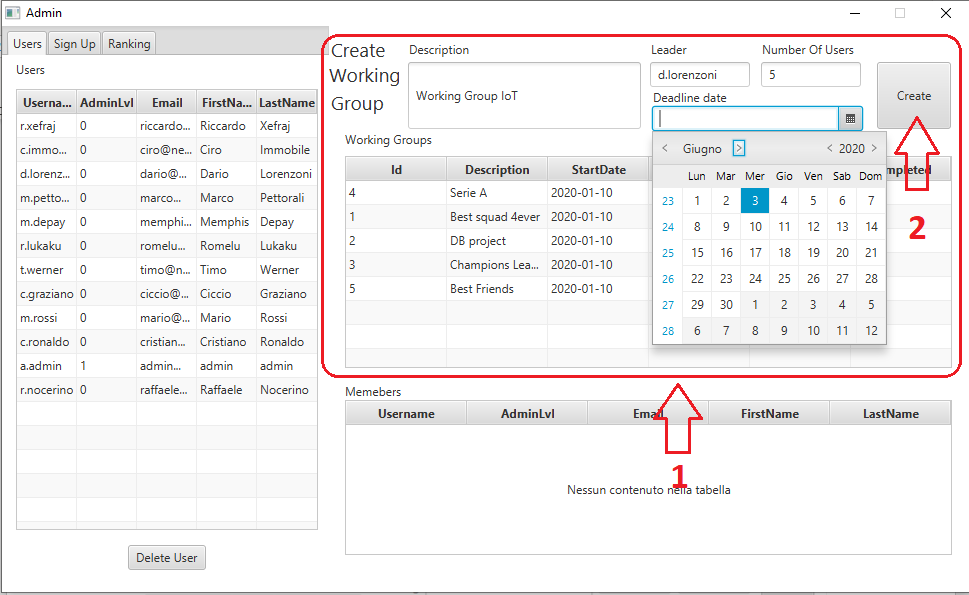
#### Visualize the ranking of the leaders:

In order to visualize the ranking of the users, the administrator can click on the ranking button in order to visualize the users and the associated ranking. The users that have never been administrator of a group are omitted from this table:

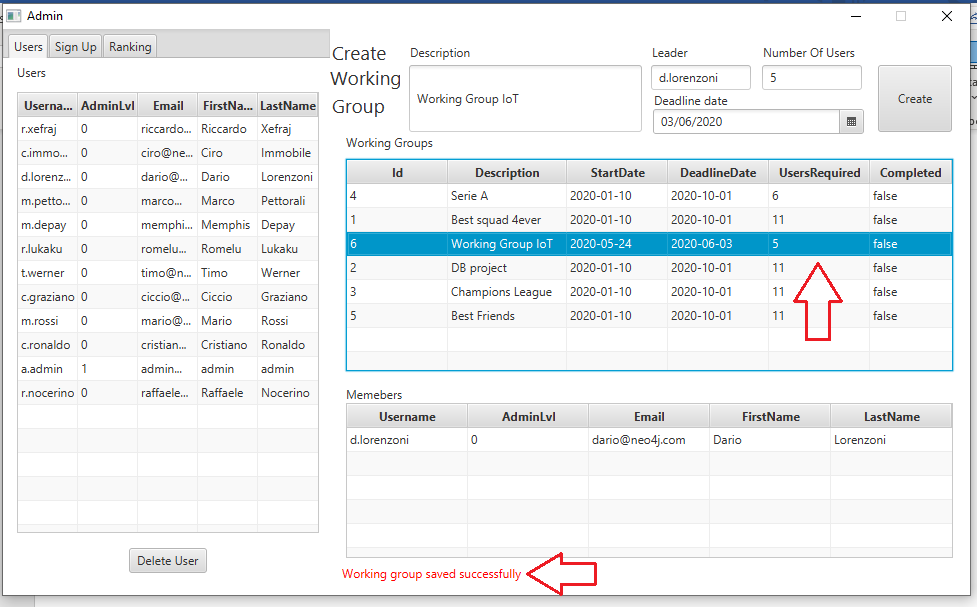


#### Add a new working group:

Using the form highlighted in the following picture the user is able to add a new working group filling the various fields:



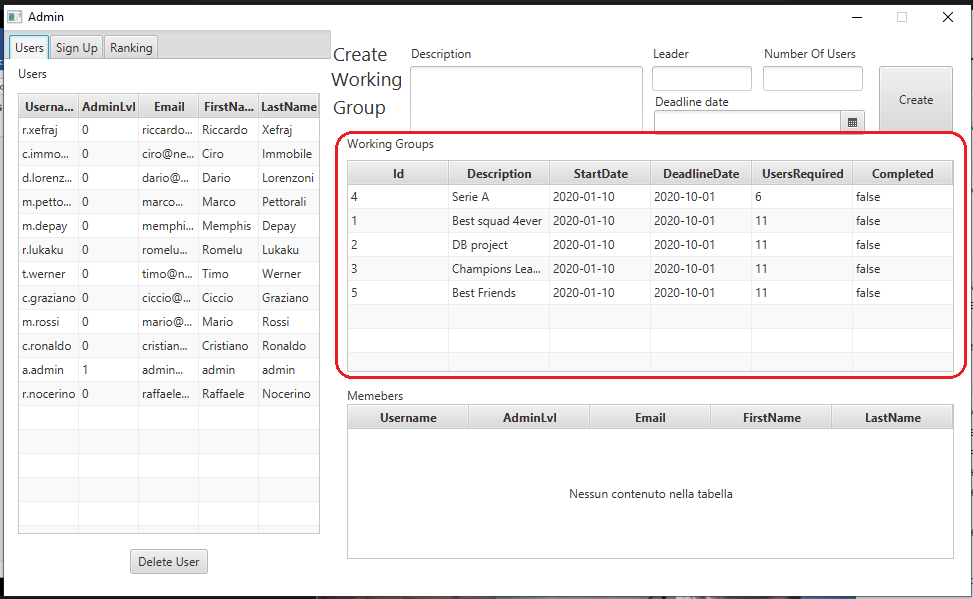
If everything goes well, the message “Working group saved successfully” will appear in the bottom of the panel and the new working group will appear in the list of the working groups (the groupID is automatically computed in an incremental manner) otherwise an error message will be displayed.



The list of working groups after the insertion of the new one

#### Visualize the working groups and their information:

The list of the working groups that were created by the administrator are always shown in the rigth bottom side of the administrator panel like in the following picture:



The administrator can click on a working group in order to display in the table below the information about the members of the group.

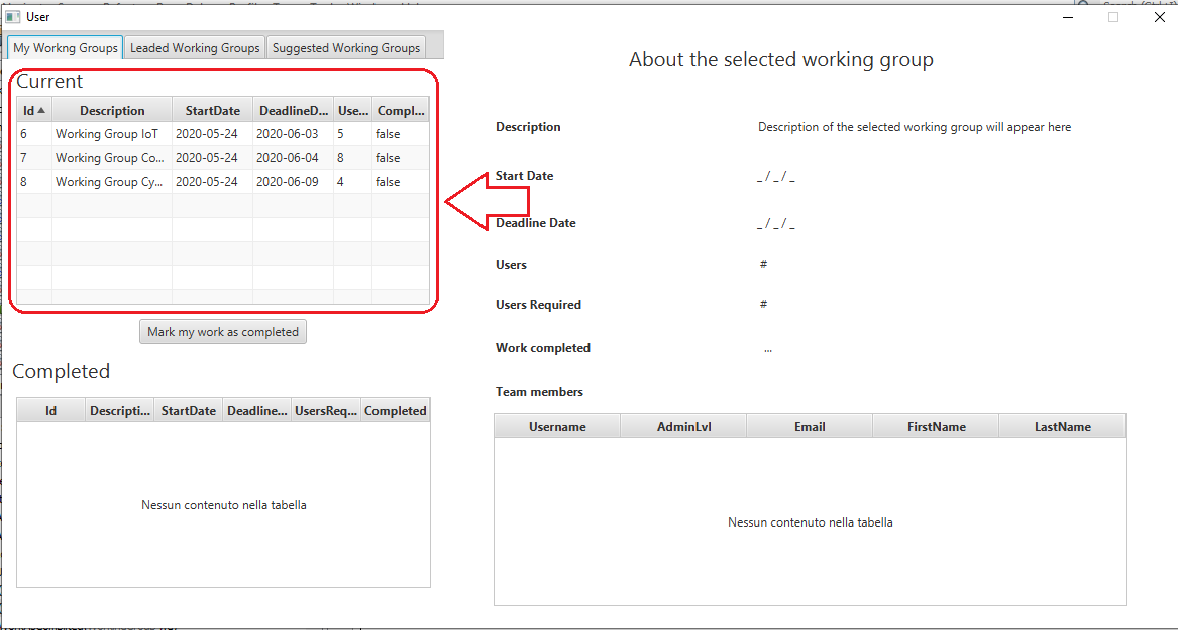
#### User case

If the user is a not an administrator, the main page showed will be the user control panel in which the (s)he can:

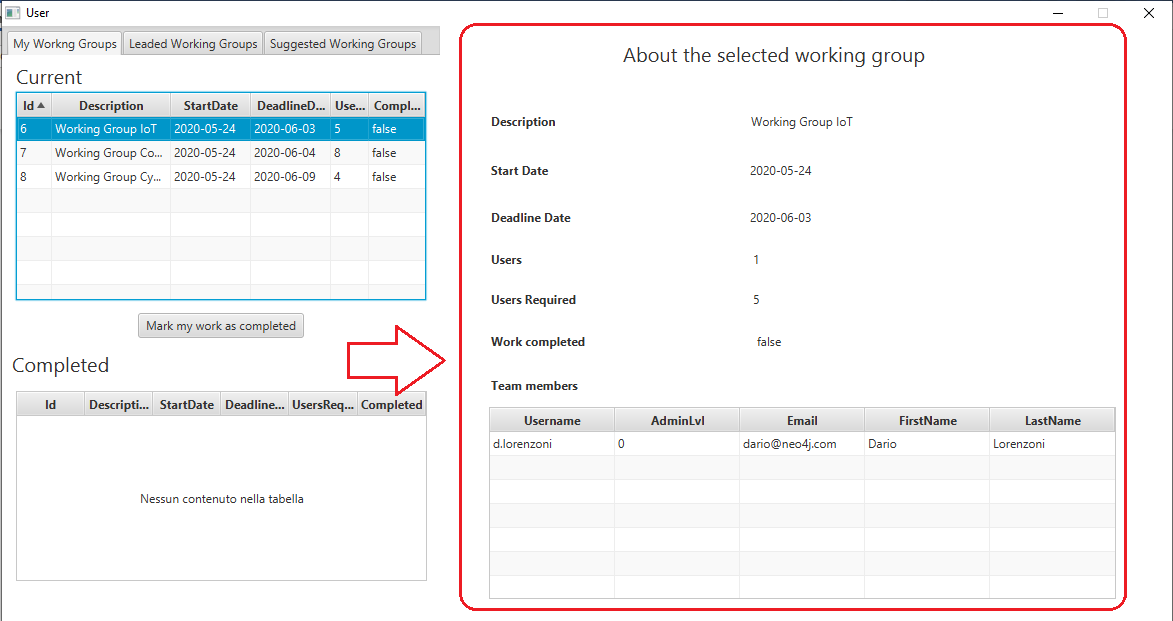
* Visualize the works in progress with their information
* Mark his(her) part of the work for a specific working group as completed
* Visualize the past works
* Visualize the information about the groups that (s)he leads and the application from the other users to this working group and accept them
* Visualize the not full working groups suggested and eventually send an application for them

#### Visualize the eventually work in progress with their information:

At the startup the user panel will show the list of the current working groups to which the user belongs to with all their general information.

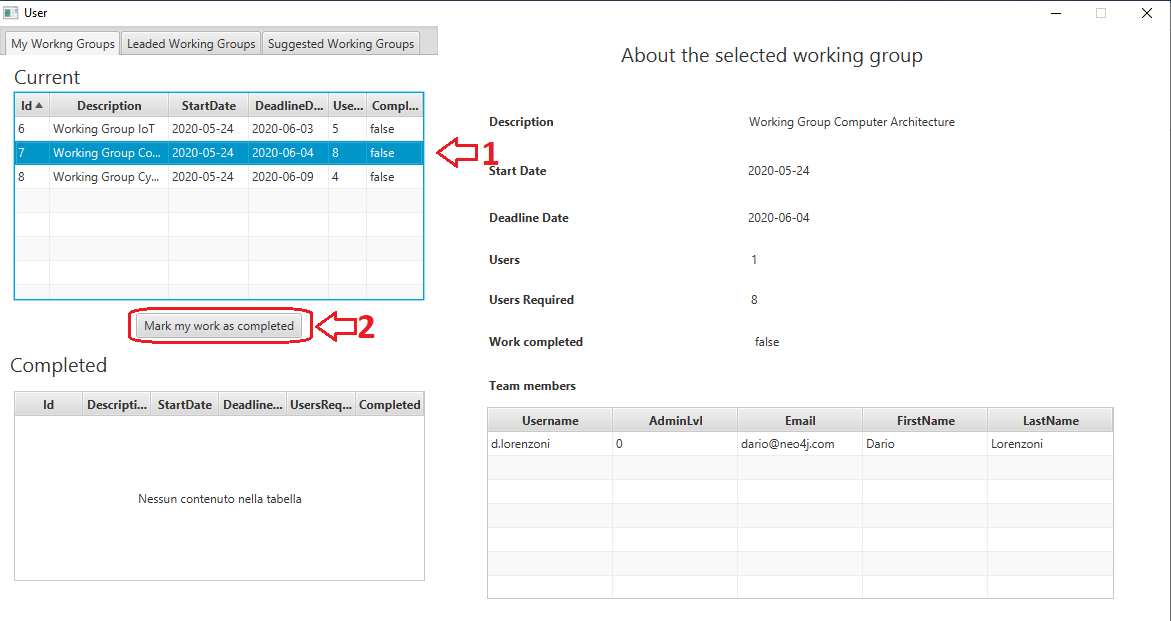


The user can select a working group from the list in order to have more detailed information in the right side of the panel. In particular, a list of the current members of the working group is reported in the table in the bottom.

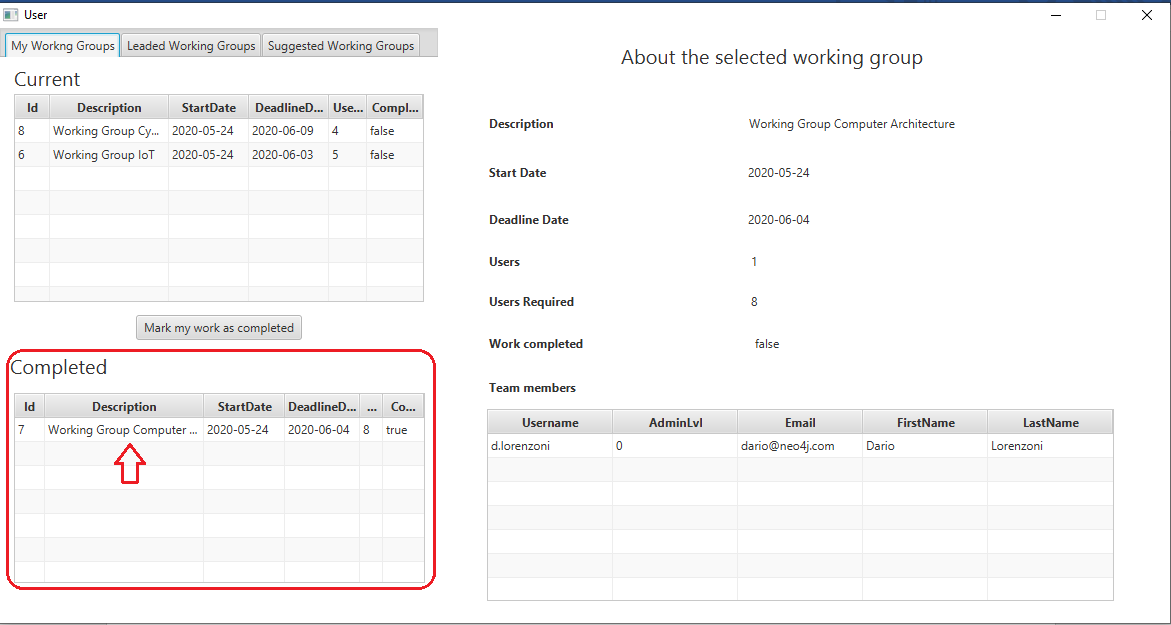


#### Mark his(her) part of the work for a specific working group as complete and visualize the past group:

From this panel, the user can mark its part of the working group as completed, it can perform this operation using the button “Mark my work as completed” like in the following picture:

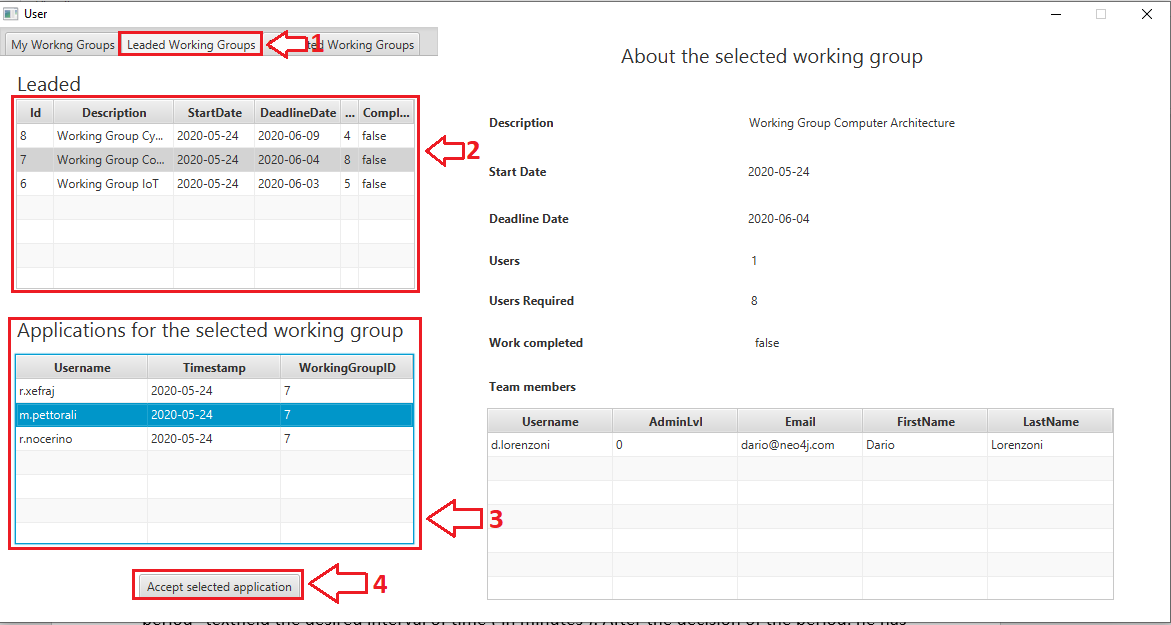


This operation will add this working group in the list of the working groups completed:

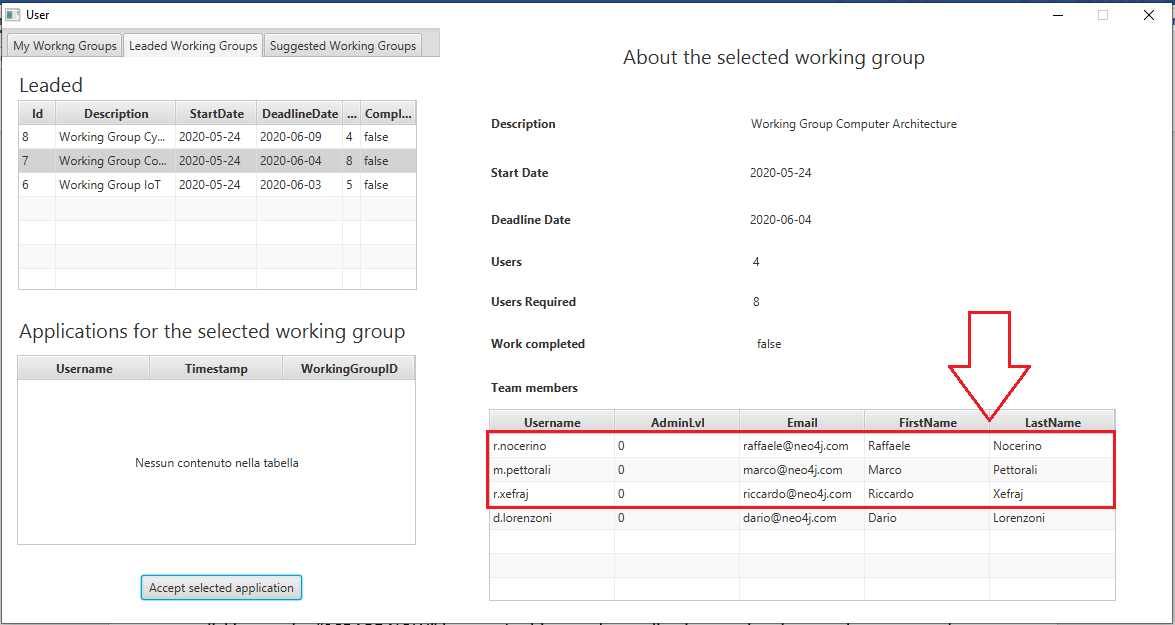


#### Manage the leaded working groups:

First of all in order to manage the working group that the user leads, the user has to click on the button “Leaded working groups”: this will change the content of the right side of the panel displaying the list of the working groups that the user leads. From this table the user can select one of the working groups in order to display the list of applications for the selected group. After that (s)he can select one of this applications and click the “Accept selected application button” in order to add the selected user to the members of the working group.



The list of members of the working group that is present on the right side of the panel will be update.



#### Visualization of the suggested working group and the application sending:

In order to display the suggested groups for a user, the latter has first to click on the button “Suggested Working Groups”; this will, one more time, change the content of the left side of the panel, but this time it will show the list of the suggested working groups, ordered by the affinity grade and give the possibility to the user to send an application for the selected working group clicking on the button “Send application for the selected working group”.

