My Fundraising Project

Code Documentation

Daria Maggi, Guido Gagliardi, Lucia Zinelli, Matteo Dessì

University of Pisa, 2019
Artificial Intelligence and Data Engineering
Department of Computer Engineering

Contents

Chapter 1

Use Case

1.1 Introduction

The main scope of this application is to allow a network of companies to create, handle and contribute to a fundraising scheme for jointly financed projects.

By logging in with its credentials, each company is able to create a new project, to update its stake or to withdraw one of the projects that the company has proposed for fundraising.

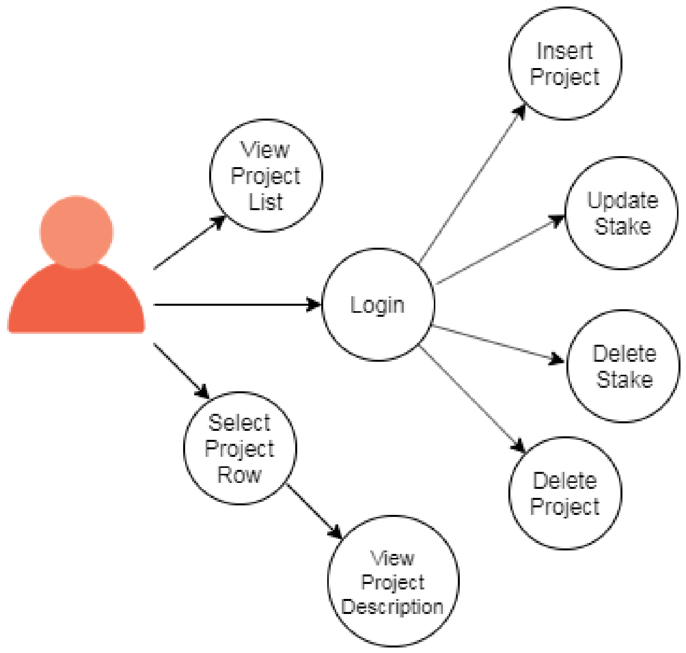


Figure 1.1: use case diagram

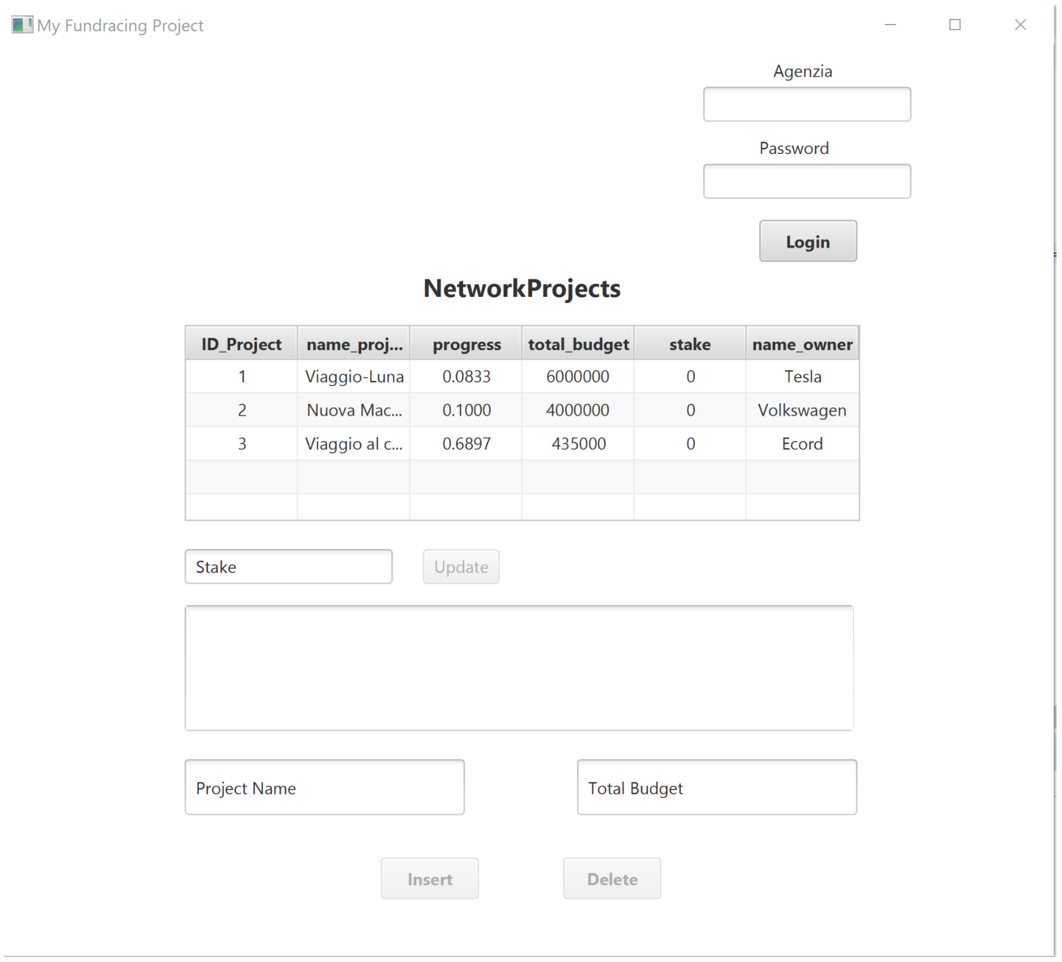
The main actor of the application is the company manager. When the app is launched, the user can see the existing projects listed in the the NetworkProjects table. The table has six columns. The first one is the project identifying code; the second is the project name; then we see the columns of progress, expressed in percentage with respect to the prefixed budget; total budget; stake, undefined if you are not logged in, and at the end, the name of the project's owner.

By clicking a table row, the user visualizes the project description. Then he can modify the project stake by adding it on its field and press the Update button. If the user want to delete one of its own projects, he can by pressing the delete Delete button. Otherwise if he's not the project's owner, its stake is set to zero. If it's already zero, an alert will appear on the screen.

If the user want to insert new projects in the network, it can by pressing the Insert button, after have filled the Description, Project Name and Total Budget.

1.2 Use Case

At the opening, the application show the interface rapresented in figure 1.2.



NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	0	Tesla
2	Nuova Mac...	0.1000	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord

Figure 1.2: The picture represents the interface tha will appear to the user at the opening of the application

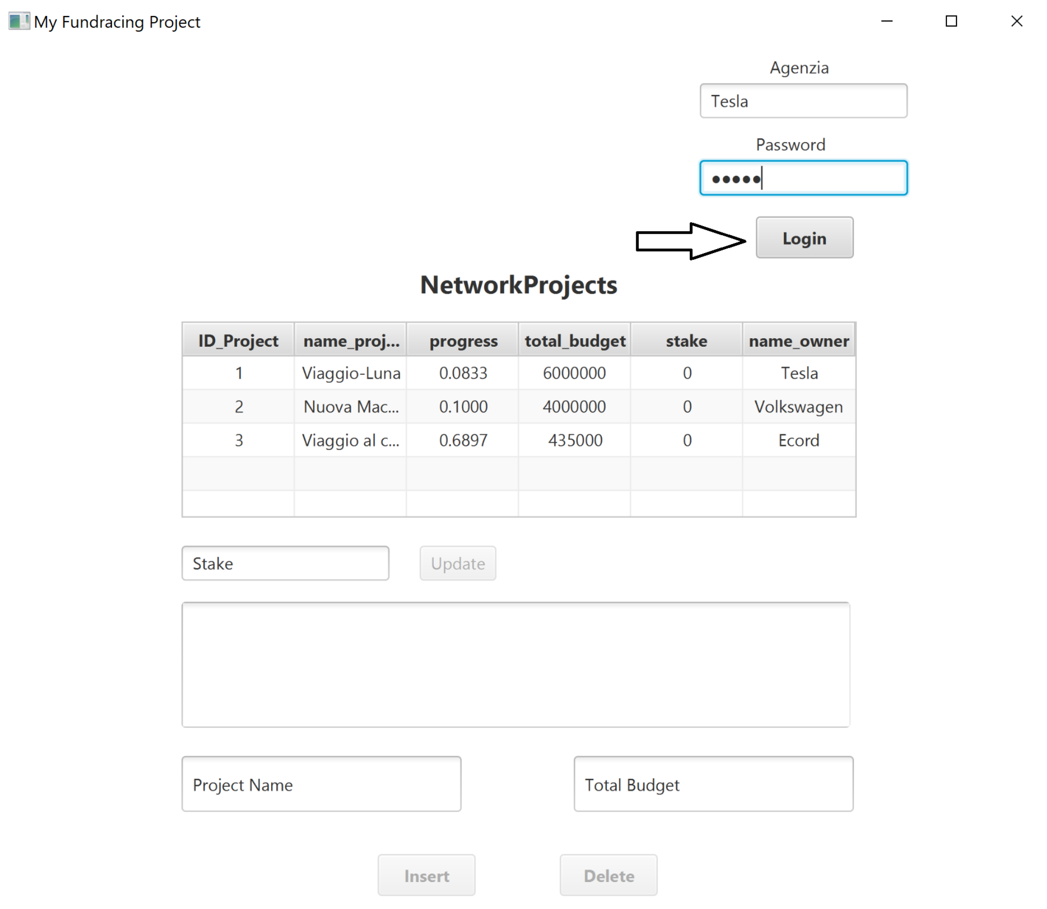
The user must then log in by inserting its credentials (company name and password) and clicking on the Login button (fig 1.3).

After a successfull login, the private informations in the table will appear and also the buttons and text fields will be activated as in the fig. 1.4.

If the user wants to insert a new project, it must specify the description, project name and total budget and then click the button (fig. 1.5.

After the insertion the interface will be as shown in fig 1.6.

When the user wants to delete one of its own project, it has to click on the corre-



ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	0	Tesla
2	Nuova Mac...	0.1000	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord

Figure 1.3: The picture represents an example of login, using the credentials of the Tesla agency

sponding row of the table and then click the Delete button (fig. 1.7) .

If the application deletes the selected project,in the case the user is also the project's owner, the corresponding row will disappear from the table (fig. 1.8).

Instead, if the user has selected a project whose it's not the owner, only the stake will reset (fig. 1.9 - 1.10).

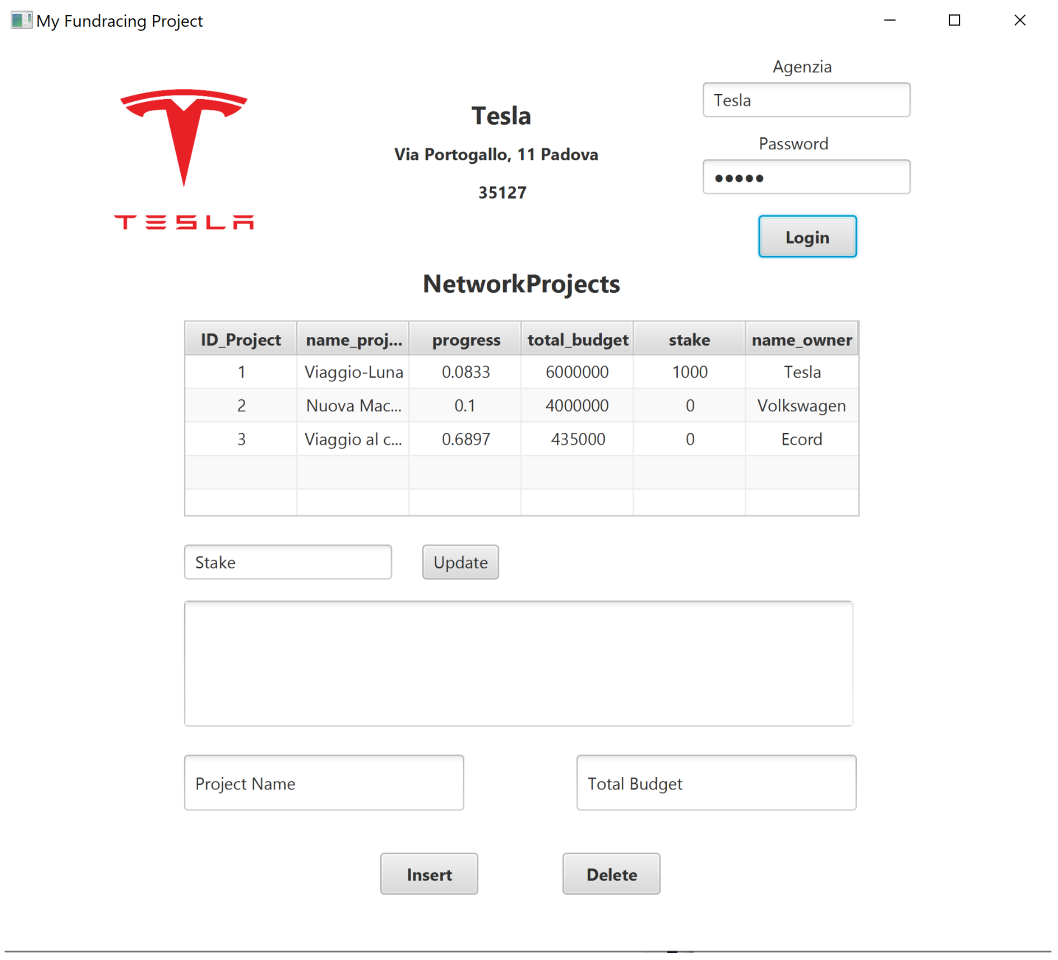
If the user wants to update its stake for a project, it has to click on the corresponding row of the table, inserts the new stake in its field, and then clicks the Update button (fig. 1.11).

The fig. 1.12 shows how the interface will appear after updating the stake.

1.3 Statechart

The statechart of the application is shown in fig 1.13, and explained below.

At the beginning, the application persists in a *Idle* state; when the system catch a LOGIN request, it controls that the username and password are filled, and perform a login authentication. If the authentication failed the system returns the Idle state, after sending an error message. If the authentication success the agency fields are updated and the stakes field are displayed in the table.



Tesla
Via Portogallo, 11 Padova
35127

Agenzia: Tesla
Password: •••••

NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	1000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord

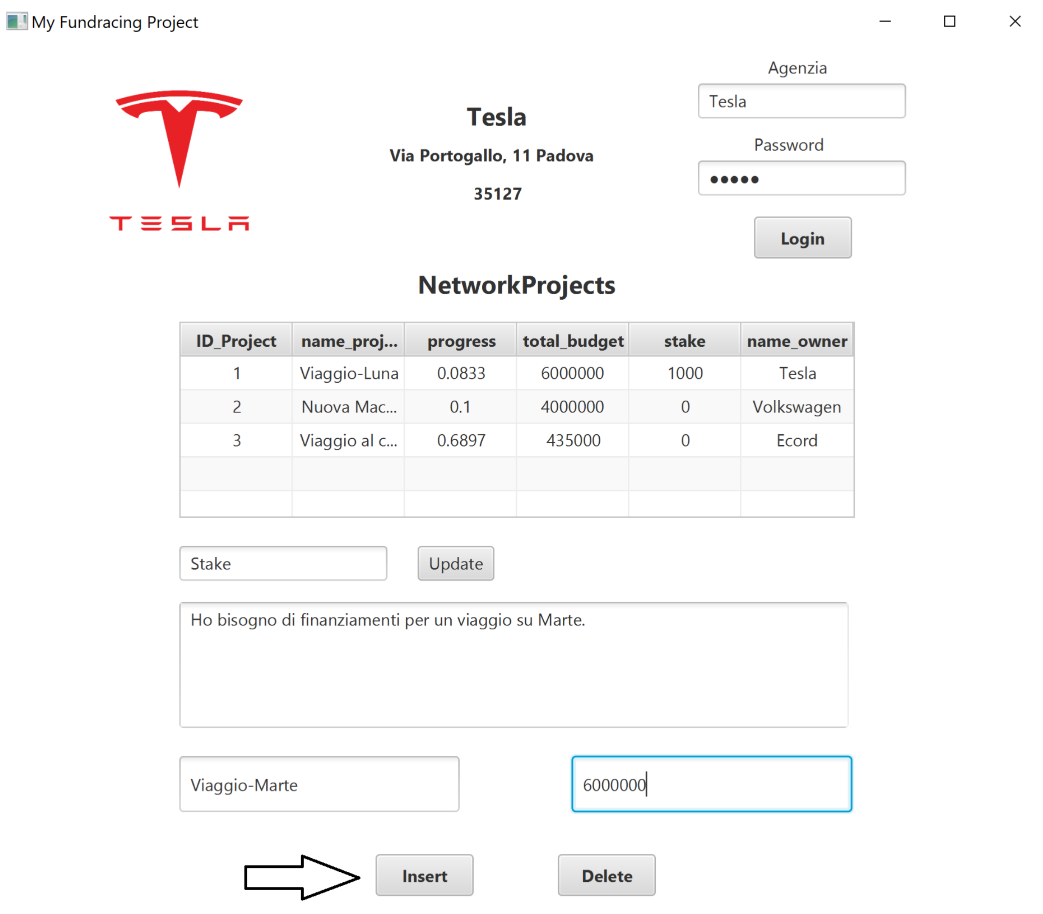
Figure 1.4: The picture represents an example of how the interface will appear after a successfull login

From here, the application waits for an input of LOGIN, INSERT or SELECT PROJECT nature: if the system catch a LOGIN request the application perform again a new LOGIN action, persisting on the waiting state in both sistuations of authentification failed or success; if the system catch an INSERT request, it controls the completition of thehttps://it.overleaf.com/project/5da70230fa08b20001998902 name_project, total_budget and description fields and than the system store a new project in the network and remains to the wait state; if it catchs a SELECT PROJECT event, the description and the stake fields of the corresponding project are edited with the pertinent informations and the application switches to a state of Project Selected.

From there, there are two possible request that the system could catches: DELETE and EDIT STAKE; after both of that the system returns to the Wait state.

When it catches a DELETE request it provides to control if the login information, username, is the same of the username associated with the project. If it is, the system provides to eliminate the project to the network. If it is not the system control if the username is associated with a stake on this project, and, if it is delete it, if it is not signalize an error message.

When it catches a EDIT STAKE request, the system provides to update the stake with the value in the corresponding field. Doing this, the system control if the new total budget of the project, with the last stake, exceeds the limit of the old project total budget.



Tesla
Via Portogallo, 11 Padova
35127

Agenzia: Tesla
Password: •••••

NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	1000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord

Stake

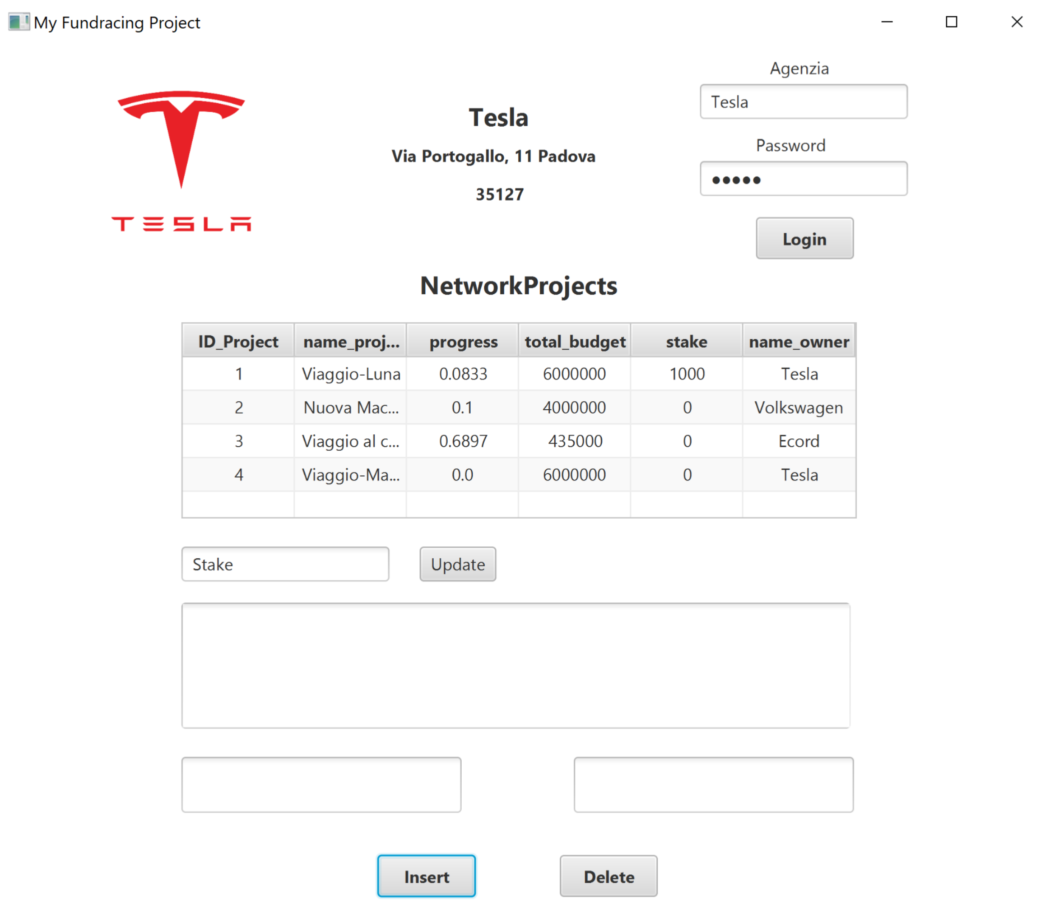
Ho bisogno di finanziamenti per un viaggio su Marte.

Viaggio-Marte

6000000

Figure 1.5: The picture represents an example of insertion of a new project

In that case the system provides to limit the amount of the updated stake to respect the constraint.



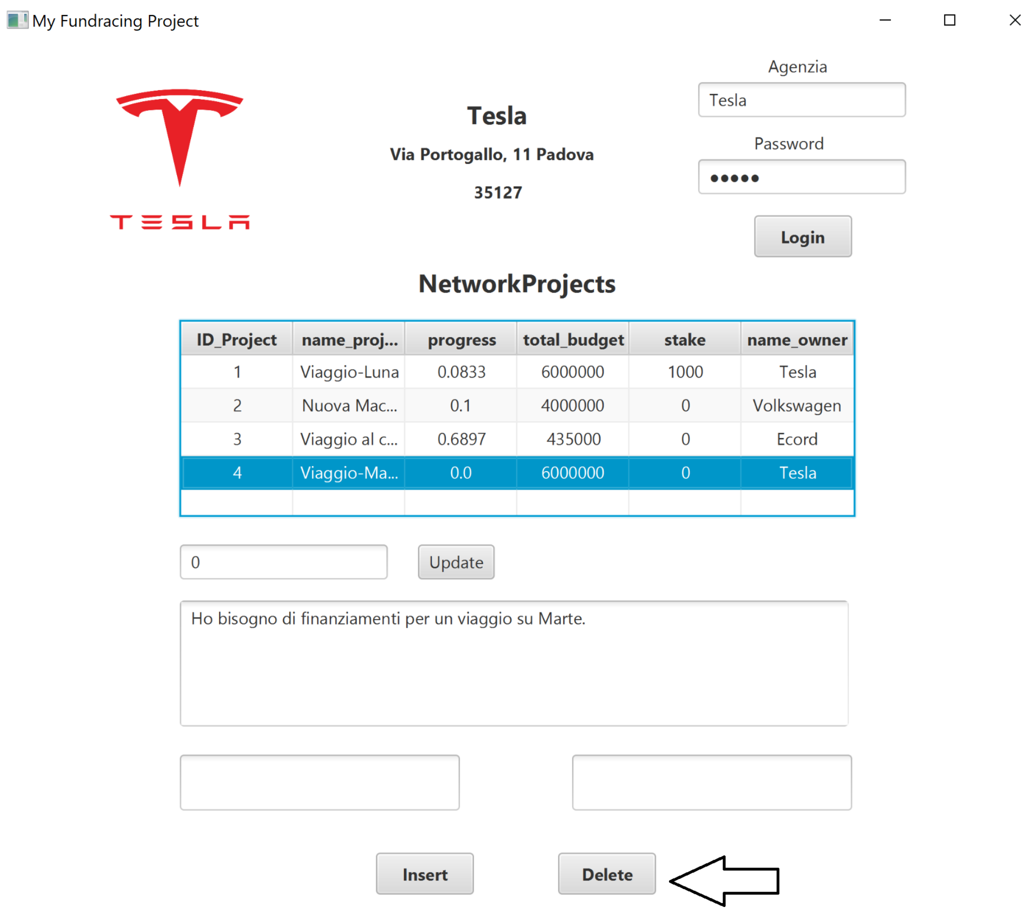
Agenzia: Tesla

Password: ●●●●●

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	1000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord
4	Viaggio-Ma...	0.0	6000000	0	Tesla

Stake

Figure 1.6: The picture represents how the interface will appear after inserting a new project



Tesla

Via Portogallo, 11 Padova

35127

Agenzia: Tesla

Password: ●●●●●

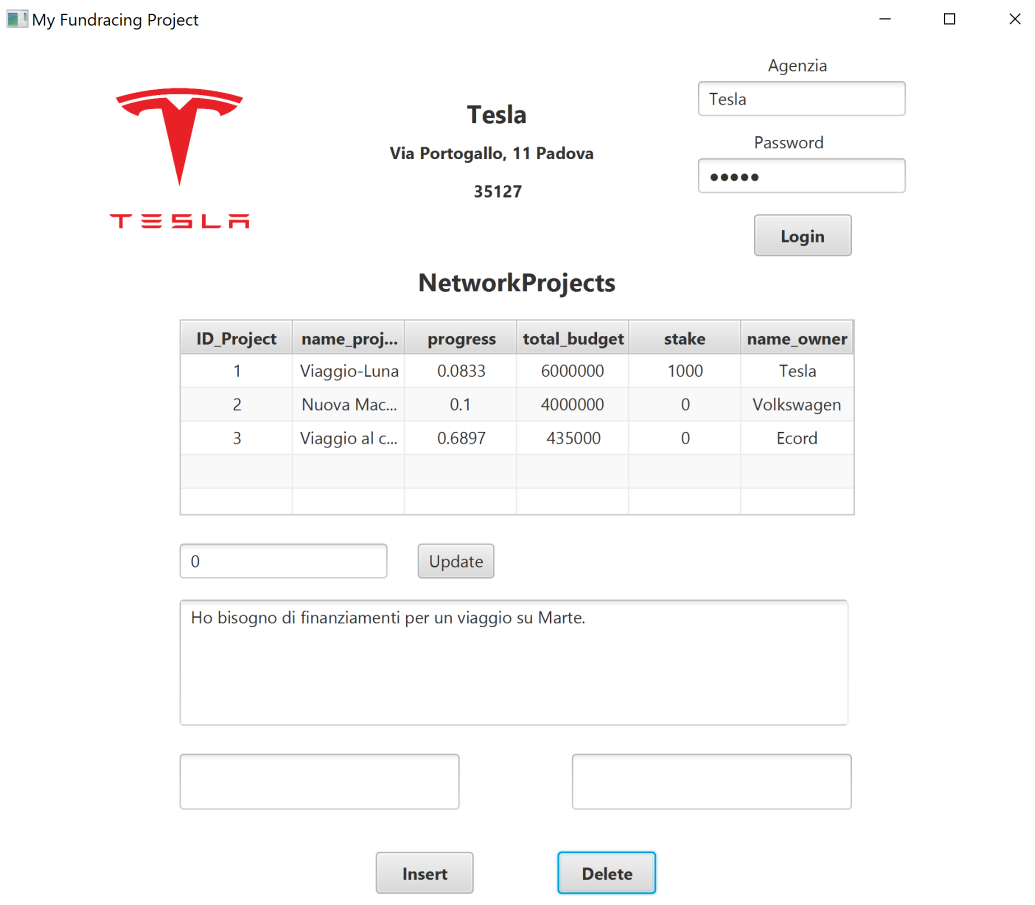
NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	1000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord
4	Viaggio-Ma...	0.0	6000000	0	Tesla

0

Ho bisogno di finanziamenti per un viaggio su Marte.

Figure 1.7: The picture represents how the interface will appear before deleting a project



Tesla

Via Portogallo, 11 Padova

35127

Agenzia: Tesla

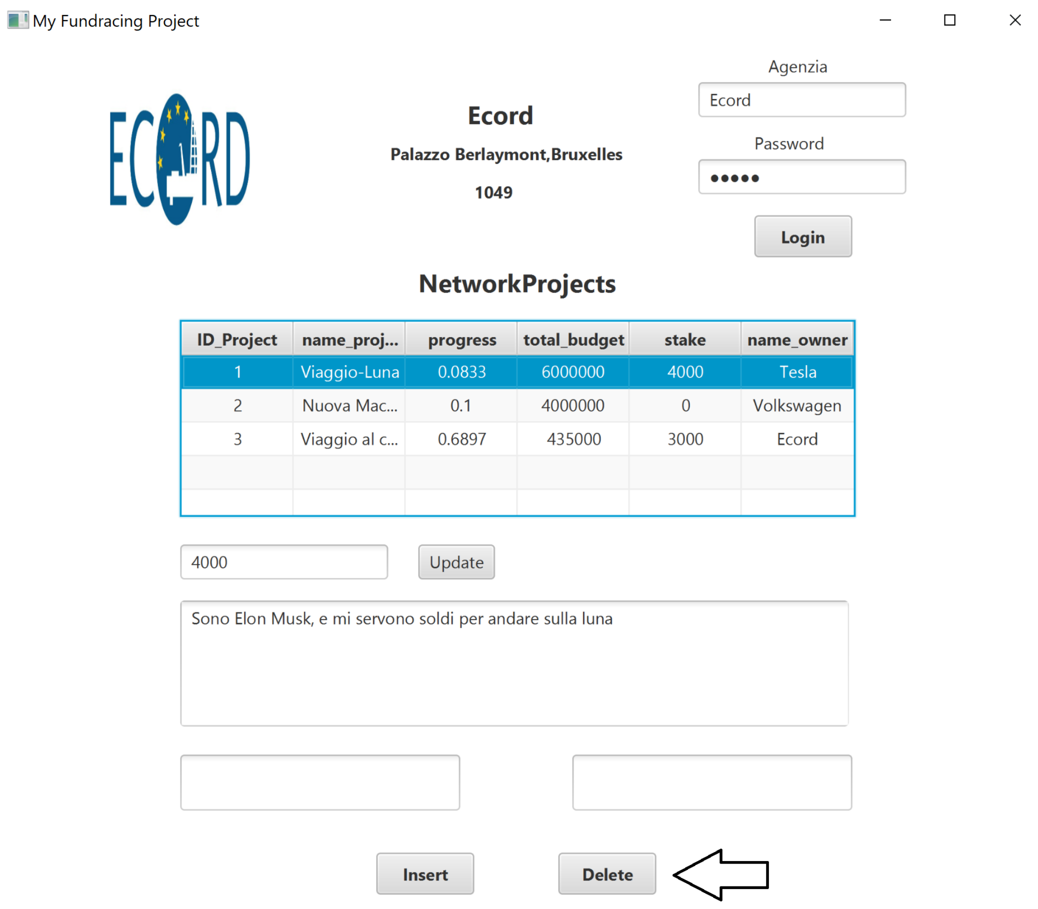
NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	1000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	0	Ecord

0

Ho bisogno di finanziamenti per un viaggio su Marte.

Figure 1.8: The picture represents how the interface will appear after deleting a project


My Fundracing Project

Ecord
Palazzo Berlaymont,Bruxelles
1049

Agenzia: Ecord
Password: •••••

Login

NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	4000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	3000	Ecord

4000

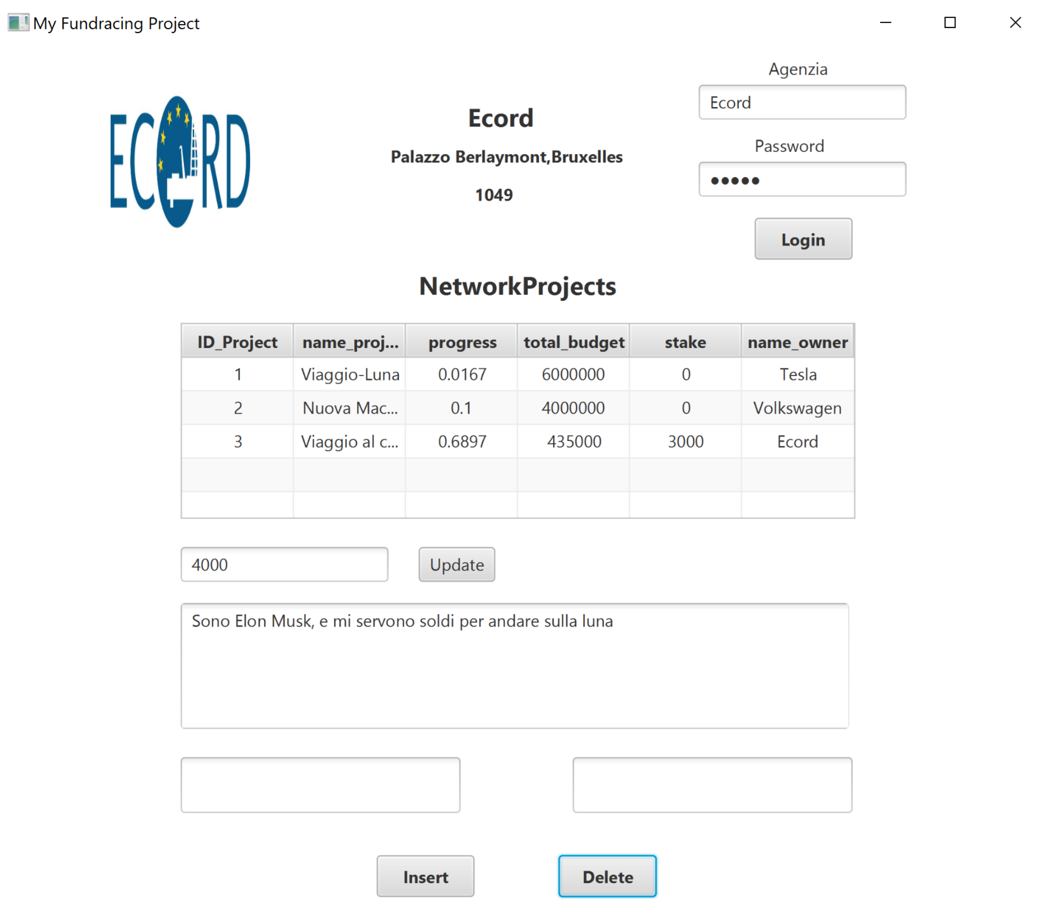
Update

Sono Elon Musk, e mi servono soldi per andare sulla luna

Insert

Delete

Figure 1.9: The picture represents how the interface will appear before deleting a stake



Ecord

Palazzo Berlaymont,Bruxelles

1049

Agenzia: Ecord

Password: •••••

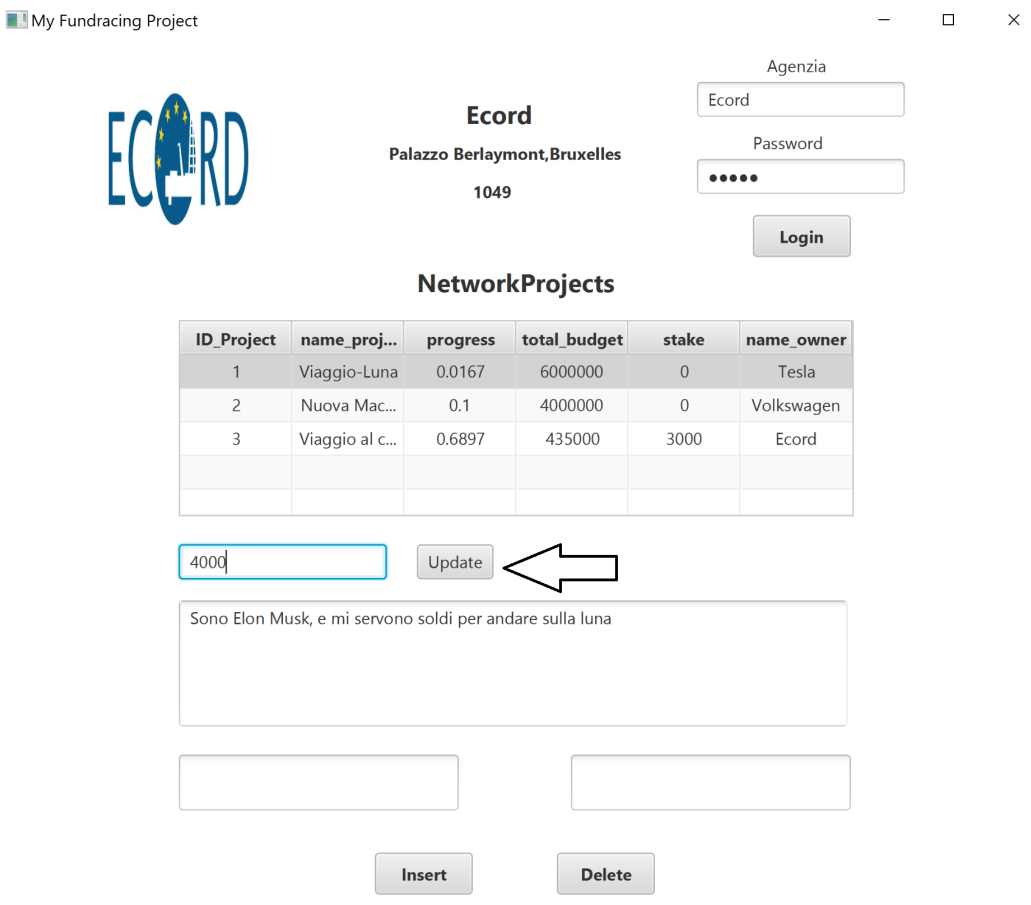
NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0167	6000000	0	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	3000	Ecord

4000

Sono Elon Musk, e mi servono soldi per andare sulla luna

Figure 1.10: The picture represents how the interface will appear after deleting a stake



My Fundracing Project

Ecord

Palazzo Berlaymont,Bruxelles

1049

Agenzia: Ecord

Password: ●●●●●

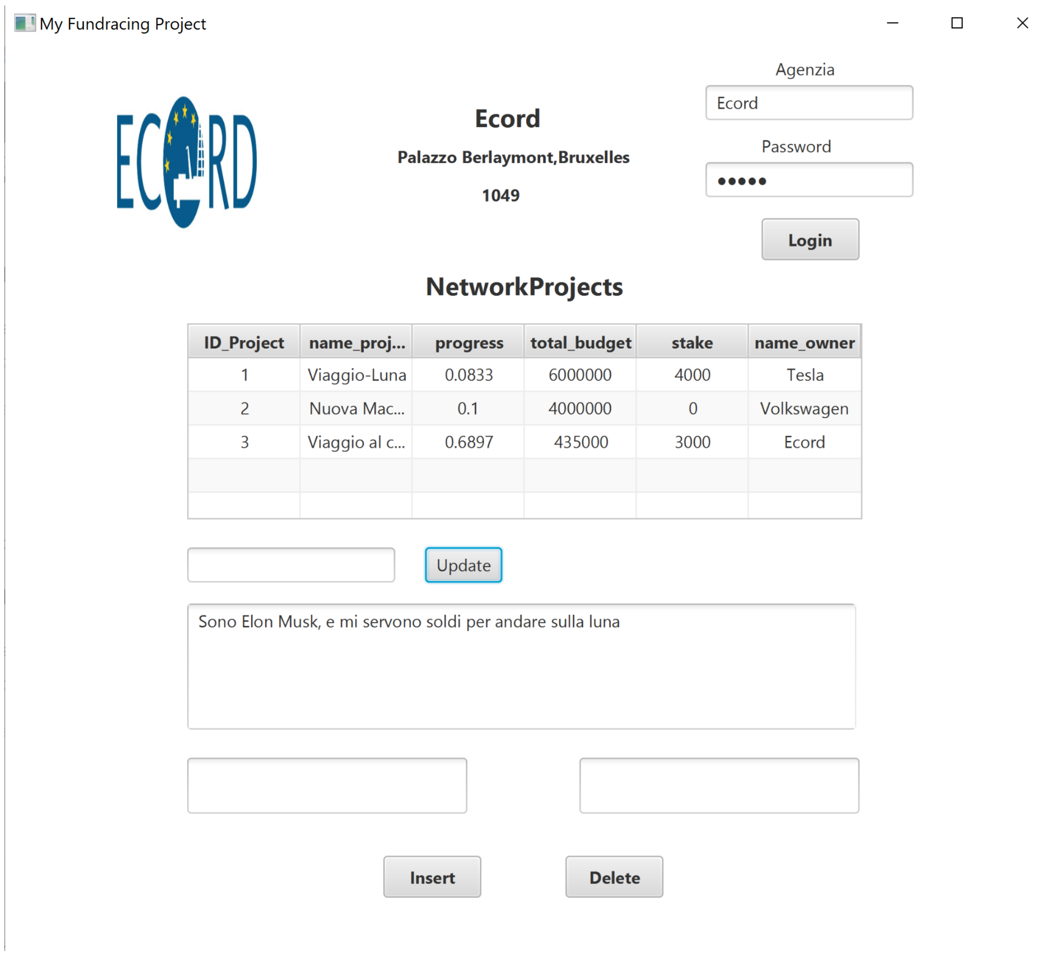
NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0167	6000000	0	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	3000	Ecord

4000

Sono Elon Musk, e mi servono soldi per andare sulla luna

Figure 1.11: The picture represents how the interface will appear before updating a stake



Ecord

Palazzo Berlaymont,Bruxelles

1049

Ecord

●●●●●

NetworkProjects

ID_Project	name_proj...	progress	total_budget	stake	name_owner
1	Viaggio-Luna	0.0833	6000000	4000	Tesla
2	Nuova Mac...	0.1	4000000	0	Volkswagen
3	Viaggio al c...	0.6897	435000	3000	Ecord

Sono Elon Musk, e mi servono soldi per andare sulla luna

Figure 1.12: The picture represents how the interface will appear after updating a stake

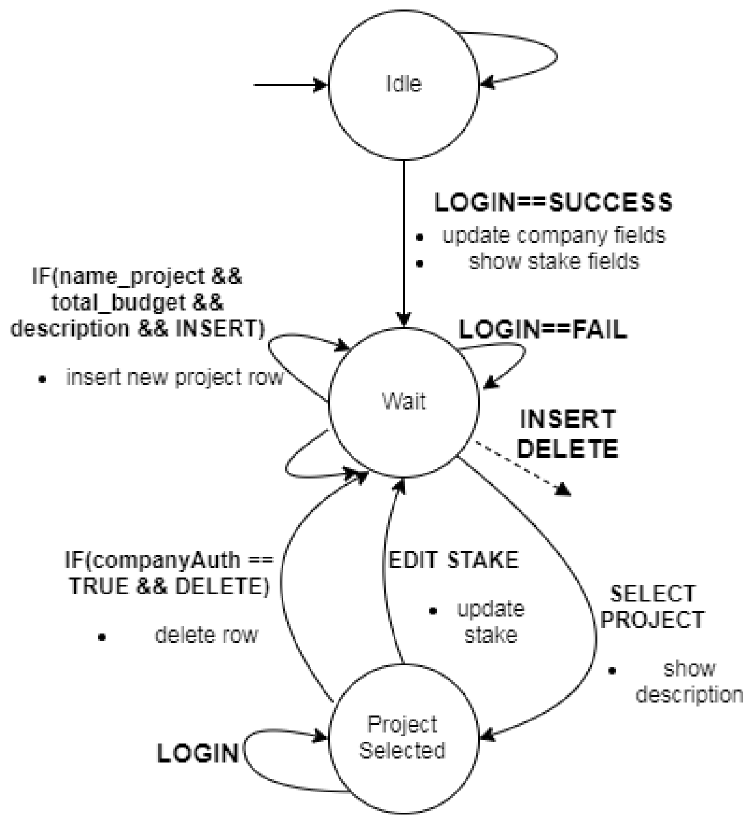


Figure 1.13: NFSA representation of the application's cycle of life, with the expected behaviour for each given event as input

Chapter 2

Requirements

The system needs to provide:

- to all users, even the unauthenticated ones, an update overall vision of the presented projects, even those to which the company is not currently contributing;
- a login mechanism, so that at all times it is possible to check your identity and then receive all the private informations about your projects and previous fundings;
- a way to each user, upon authentication, to create and to delete its own projects;
- a mechanism to increase the stake invested in its own projects or in other ones;

In order to do so, the system is required to implement a consistent way to:

- perform a login authentication
- upload new projects and stakes
- download the table and the agency informations

Chapter 3

Implementations

3.1 Database

My Fundraising Project is connected to a relational database consisting of three tables, as it can be deduced by the following ER diagram:

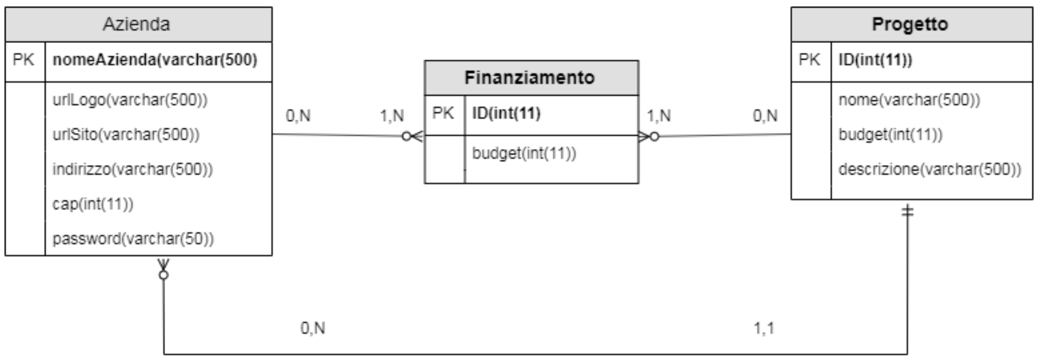


Figure 3.1: ER diagram

The relational database was created using SQL language and it is managed through MySQL 8.0.17.

3.1.1 Tables Explanation

The Azienda table contains all the informations related to an agency part of the network that will be visualized by the user after the login.

Each Azienda is uniquely identified by its business name, contained in the PK nomeAzienda(Varchar(500) not null). Each company is characterized by its own logo (“url-Logo”, varchar(500)), website (“urlSito”, varchar(500)), address (“indirizzo”, varchar(500)), the cap(“cap”, int(11)), password(“password”, varchar(50)).

The Finanziamento table contains the informations about the finances invested by the companies.

The Finanziamento table has a primary key called ID (int (11)), associated to a budget (“budget”, int(11)), which corresponds to the stake presented by the company to a particular project. The table has also the external key Azienda (“azienda”, varchar(500)), that corresponds to the primary key of the Azienda table (“nomeAzienda”) which represents the company that finances the project, and the external key Progetto(int(11)), that corresponds to the primary key of the Progetto table ("ID").

The Progetto table contains the informations related to each project in the network.

Each Project has an primary key ID (int(11)). Also it is characterized by the name ("nome", varchar(500)), the budget ("budget", int(11)), the description ("description", varchar(500)).

3.1.2 Relationship Explanation

There is a relationship zero to many from the Azienda table to the Finanziamento table, that rapresent the fact that a company could finances a lot of projects. Instead the relationship between the Finanziamento table and the Azienda table is one to one: it can not exits any stake without an agency that provide it.

A relationship zero to many is from the Progetto table to the Finanziamento table, saying that one project can have more than one stakes linked to him, but could also exists a project that doesn't have stakes. The relationship from the Finanaziamento table to the Progetto table, is instead one to one.

Finally there is a relationship zero to many from the Azienda table to the Progetto table, saying that an agency could be the owner of one or more projects and a relationship one to one from the Progetto table to the Azienda table, saying that can not exists a project without owner.

It is possible to insert a new financing only after having inserted in the database the information related to the financed project and to the company that wants to carry out the financing.

A new project can be inserted if, within the Azienda table, there are the informations related to the company that is making the insertion

3.2 Class Diagram

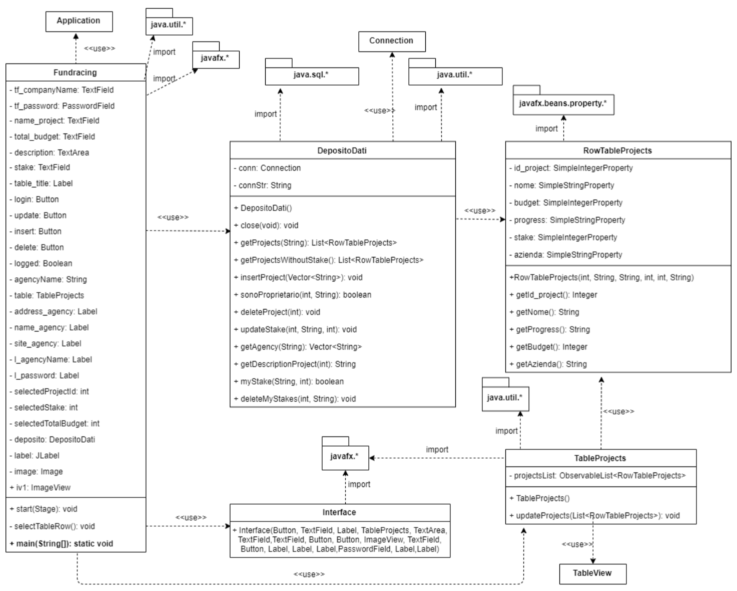

Application
java.util.*
javafx.*
<<use>>
import
import
Fundracing
- tf_companyName: TextField
- tf_password: PasswordField
- name_project: TextField
- total_budget: TextField
- description: TextArea
- stake: TextField
- table_title: Label
- login: Button
- update: Button
- insert: Button
- delete: Button
- logged: Boolean
- agencyName: String
- table: TableProjects
- address_agency: Label
- name_agency: Label
- site_agency: Label
- l_agencyName: Label
- l_password: Label
- selectedProjectId: int
- selectedStake: int
- selectedTotalBudget: int
- deposito: DepositoDati
- label: JLabel
- image: Image
+ iv1: ImageView
+ start(Stage): void
- selectTableRow(): void
+ main(String[]): static void
<<use>>
<<use>>
<<use>>
java.sql.*
Connection
java.util.*
import
<<use>>
import
DepositoDati
- conn: Connection
- connStr: String
+ DepositoDati()
+ close(void): void
+ getProjects(String): List<RowTableProjects>
+ getProjectsWithoutStake(): List<RowTableProjects>
+ insertProject(Vector<String>): void
+ sonoProprietario(int, String): boolean
+ deleteProject(int): void
+ updateStake(int, String, int): void
+ getAgency(String): Vector<String>
+ getDescriptionProject(int): String
+ myStake(String, int): boolean
+ deleteMyStakes(int, String): void
<<use>>
javafx.beans.property.*
import
RowTableProjects
- id_project: SimpleIntegerProperty
- nome: SimpleStringProperty
- budget: SimpleIntegerProperty
- progress: SimpleStringProperty
- stake: SimpleIntegerProperty
- azienda: SimpleStringProperty
+RowTableProjects(int, String, String, int, int, String)
+ getId_project(): Integer
+ getNome(): String
+ getProgress(): String
+ getBudget(): Integer
+ getAzienda(): String
java.util.*
import
<<use>>
javafx.*
import
import
TableProjects
- projectsList: ObservableList<RowTableProjects>
+ TableProjects()
+ updateProjects(List<RowTableProjects>): void
<<use>>
TableView
Interface
+ Interface(Button, TextField, Label, TableProjects, TextArea, TextField,TextField, Button, Button, ImageView, TextField, Button, Label, Label, Label,PasswordField, Label,Label)

Figure 3.2: UML class diagram

The application is written in Java language and structured in 5 classes: *DepositoDati, Fundracing, Interface, RowTableProjects, TableProjects.*

3.2.1 Back-end

The class **DepositoDati** provides the back-end functionality of the application, doing the upload and download of the data from and to the database.

DepositoDati

Private attributes:

- **conn: Connection**
- **connStr: String**

This attributes are required to create, open and close a mysql connection; specifically *connStr* is used to store the information about the connection with the mysql server.

Pubblic methods:

- **DepositoDati()** - class constructor, open a connection to the mysql-server described from connStr
- **close():void** - close the mysql connection opened in the constructor
- **getProjects(String agencyName):List<RowTableProjects>** - get all the projects from database and, for each project, calculate the stake of the agency passed from argument. Returns the list of projects with the stakes associated.
- **getProjectsWithoutStake():List<RowTableProjects>** - get all the projects from database and, for each project, calculate the progress state. Returns the list of projects with the progress associated.
- **insertProject(Vector<String> val):void** - insert a new project in the database using the data passed as arguments.
- **deleteProject(int projectId):void** - delete from database the project identified by the value projectId passed by argument.
- **ImOwner(int projectId, String agencyName): boolean** - returns true if the agency identified by agencyName in the database is the owner of the project identified by the identification value, projectId.
- **updateStake(int stakeBudget,String agencyName,int idProgetto):void** - update the stake made from the agency *agencyName* to the project identified by the ID *idProgetto* with the value *stakeBudget*. If the stake doesn't exists provides to create it.
- **getAgency(String agencyName, String password):Vector<String>** - perform the login for the agency *agencyName* evaluating if the password passed by argument is correct or not. If it is correct return the agency data as elements of array (vector).

- **deleteMyStakes(int projectId,String agencyName):void** - delete the stakes made by the agency rapresented by *agencyName* for the project rapresented by *projectId*.
- **getDescriptionProject(int id_project):String** - returns the description of the project rapresented by the ID, id_project, passed by argument.

3.2.2 Middle Layer

The Middle Layer is formed by the classes **TableProjects** and **RowTableProjects**.

TableProjects

Private attributes:

- **projectsList: ObservableList<RowTableProjects>**

This attribute represents a list containing all the rows present in the table.

Public methods:

- **TableProjects()** - class constructor, inizializes the columns of the table and then adds them to it.
- **updateProjects(List<RowTableProjects> projects):void** - This function clears all the content of the table and replace it with the informations stored in the List passed as argument.

RowTableProjects

This class is the data representation of the rows in the table.

Private attributes:

- **id_project: SimpleIntegerProperty**
- **nome: : SimpleStringProperty**
- **progress: : SimpleStringProperty**
- **budget: : SimpleIntegerProperty**
- **stake: : SimpleStringProperty**
- **azienda: : SimpleIntegerProperty**

Each attribute is a column in the table.

Public methods:

- **RowTableProjects(int ID_Project, String name_project, String progress, int total_budget, int stake, String name_owner)** -class constructor, it converts the attribute passed as argument in SimpleItsProperty, where Its is the type of the argument

- **getId_project():Integer** - returns the private field id_project of the class
- **getNome():String** - returns the private field nome of the class
- **getProgress():String** - returns the private field progress of the class
- **getBudget():Integer** - returns the private field budget of the class
- **getStake():Integer** - returns the private field stake of the class
- **getAzienda():String** - returns the private azienda stake of the class

3.2.3 Front-end

Class Fundracing extends the Application class from the JavaFX package, thus providing the *front-end* functionality of the application and managing the look-and-feel side from the point of view of the user.

Class Fundracing

The class has the following *private* attributes:

- **table:TableProjects** is the table containing the projects that will be shown in the interface.
- **logged:Boolean** actually determines what will be shown and which functionalities will be enabled, or disabled, in the interface. If set to *true*, the class will be responsible for showing the company identification data and its logo). Its *default* value is *false.*
- **deposito: DepositoDati** is used for retrieving data from the database.

The *protected* and *private* attributes of the class that are related to the technical side of the interface are not listed; their usage shall be described in detail below, in the documentation of the *start* method, but for now they only will be referred to in the class diagram.

It has the following *public* methods:

- **start(Stage stage): void** is responsible for retrieving data from the database, using its attribute *table.* It then handles all the events that may occur during the usage of the interface company side through dedicated *lambda functions*:
  - if a *login* events occur, the company business name (stored in the attribute **l_agencyName:Label**) and the password inserted by the user in the corresponding fields are checked through *deposito.* If the authentication parameters are correct, variables **iv1:Imageview site_agency address_agency name_agency** are set to the actual values of the company. If the authentication request is denied, an error message appears.
  - if an *insert* event occurs, via *deposito* all the information necessary is transmitted to the database in order to add a new row to it. Then, a new row is added to the interface table project via the *table* reference.

  - if a *delete* event occurs, it is first checked whether the company is the owner or not of the selected project, via the boolean-returning sonoProprietario method invoked through *deposito*; if so, the project is deleted both from the database and the interface project table. If not, but the company still has stakes in the project, they are deleted and the total current funding to the project is recalculated.
  - if an *update* event occurs, it is checked whether the proposed value of the stake is greater than the actual value. If so, the stake and the funding are updated both in the database and the interface project table; otherwise, nothing changes.

- **main(String[] args):void** is the static method launching the application.

The Fundracing class also presents a private method:

- **selectTableRow(): void** retrives the data from the database in order to update the project table information.

Class Interface

The class positions the various elements on the interface.

It has the following *public* method:

Interface(...) sets the correct position of all the elements passed as argument of the function.