DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Project Report LAPR2

Payments Management

Team 43 _ Class DKL
1190778 _ José Silva
1190780 _ José Forno
1190986_ Raúl Coelho
1191051_ Sandro Moura
1191091_ Tiago Magalhães

Teachers/Advisors
Teresa Araújo (TPA)
Álvaro Teixeira (ACT)
Ana Barata (ABT)
Rosa Reis (RMR)

Client
Carlos Ferreira (CGF)

Course Unit
Laboratório/Projeto II

June 2020

isep Instituto Superior de Engenharia do Porto

ABSTRACT

We have been approached by a client requesting us to produce an application able to fulfil the necessities of their company, which is making and managing organizations payments to freelancers as well as overall statistics about the payment values and the task execution delays.

Working with precise results is very important because we are dealing with hours of work and payment transactions which involve various aspects such as people, money, etc. An error can cause many problems to both the organization and the freelancer.

To solve this problem, we implemented a functional program capable to realize their needs, with the capacity to import information from documents, add organizations, collaborators, managers, freelancers, tasks and payments transactions manually, define a date to make automatic payments and qualified to send information to text files or by email. To keep the most precise values, the application contains tests to all calculating methods so there are no invalid results.

Table of Contents

List of figures

1. Introduction

This report has the purpose of describing the development process of the new application for T4J, developed within the course unit of LAPR2. This application has the objective of making and managing organizations payments to freelancers. The main objective of this project was for us to correctly apply the various concepts learned in this semester on our different course units and give a perspective about what is like to work in the job market.

In this report it will be shown the problem statement, in which the problem is described and analysed, the planning and the methodology used to develop this application and the proposed solution (explained and justified in detail).

2. Problem statement

We were addressed by the T4J administrator, requesting the development of an application that allows T4J organizations to make and manage payments to freelancers. The application should also enable the T4J administrator to monitor the payments made by the organizations as well as the performance of the freelancers.

2.1. Problem description (detailed)

The client solicited us an application able to fulfil the follow request:

- The T4J administrator should be capable to add new organizations to the system manually and specify one manager and one collaborator for the organization, and see statistics describing the performance of the freelancers
- The collaborator of the organization should be allowed to create tasks and freelancers when that information is not available in the system, have access to a graphical interface used to create payment transactions.
- The manager of the organization should be able to define a date when all the tasks/transactions in the system will be paid automatically. After this process, the freelancer will receive one e-mail with an invoice describing the amount to pay for each task and the overall payment value and the application will register all payments in a single txt file
- The manager of each organization should be allowed to load a file (txt or csv) containing the information about transactions and describing the execution information of the tasks
- The manager and the collaborator of the organization should be capable to see overall statistics about task execution times and freelancer payments. The application must allow to sort the freelancers by name or by payment value.
- The system should automatically send an e-mail, in the last day of every year, to all the freelancers who have a mean task delay time (during the current year) that is higher than 3 hours and have a percentage of delays (during the current year) that is higher than the overall percentage of delays. This e-mail can also be sent by the administrator at any given moment.

3. Work Organisation, Planning and Methodology

In the beginning we analysed the assignment to gather all the required information. We read it multiple times, so we really understood what we needed to do. Then we split the work into "use cases", using the principles we learned in Software Engineering, and distributed two per each team member according to the difficulty of each "UC" (I.e. One with higher difficulty and other with lower difficulty). We divided the assignment into the following "UC"'s:

- UC1 -Register Organization
- UC2 - Register Freelancer
- UC3 - Create Task
- UC4 - Create Payment transaction
- UC5 - Define automatic payment time
- UC6 - Load File
- UC7 - Make an Automatic Payment
- UC8 - Show Freelancer statistics
- UC9 - Show general statistics
- UC10 - Notify delay to Freelancer

We developed the project incrementally, using the software development process we learned in ESOFT[1]. We did a sketch of the Domain Model to help us in the initial steps and figuring out what classes we needed to connect. Then each member was in charge of developing the "UC"'s that were assigned to them. Before coding, we gathered all the work that had been done and corrected any incongruities that had happened. When coding we were in constant contact with each other's because we shared many classes and functionalities and, for the application to work, we had to coordinate efforts.

The tools we used in the development of our work were:

- Trello, where we make a daily registry of our work and registered what had already been done, what we were doing and what was left to do. We also assigned deadlines to each task to organize the time we had.
- Discord, our main communication platform once we already used it. We also found it easier to be in contact with each other while working in our "UC"'s.
- Bitbucket, where we had a repository with all our work.
- SourceTree, which we used to "upload" our work to the repository and to synchronize with each other's work.

[1] ESOFT – Abbreviation for "Engenharia de Software".

4. Proposed Solution

4.1. Login

To keep the program safe, we needed to make a login for security measures, for that we resorted on the program given by ESOFT which already had the login code implemented.

In order to login, we made a window in the application with two text fields to insert the email of the user and his password. Then the application will check if there is a user registered with the email inserted before. If there is then it will verify if the password is the same, if so, it will change the user session to the correspondent user, if the password isn't the same or the email doesn't correlate with any user then a message will appear in a label below the text fields as you can see in this image.

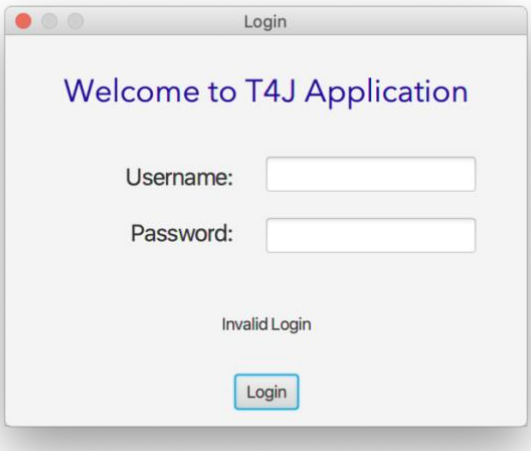

FIGURE 1

If the user closes the login window then the application will automatically close, saving all files beforehand. For the application to look modern we decided to include a welcoming message.

4.2. Main Window

In the main window is where we load all our scenes and it also regulates to the user who is currently logged in, which means that if a manager is logged in he won't see the same thing as an administrator. The window also shows the email and the name of the user. This are the three types of windows currently available:

FIGURE 2

FIGURE 3

FIGURE 4

We also included a logout button, so the user doesn't need to close the application in order to change accounts.

4.3. UC1 -Register Organization

To register the organization, we made a window in the application (that can be seen below), which only the administrator has access, where this can be made. There the administrator must insert the name of the Organization and it's NIF. He also has to insert one manager and one collaborator as well as theirs's name and email. This information is shown on the screen and after the administrator confirms the data the manager and collaborator's passwords are generated. To generate the manager and the collaborator's password we implemented an API, using protected variation and an adapter, that generates an alphanumeric random password. These passwords are then sent by email to the respective user(collaborator/manager).

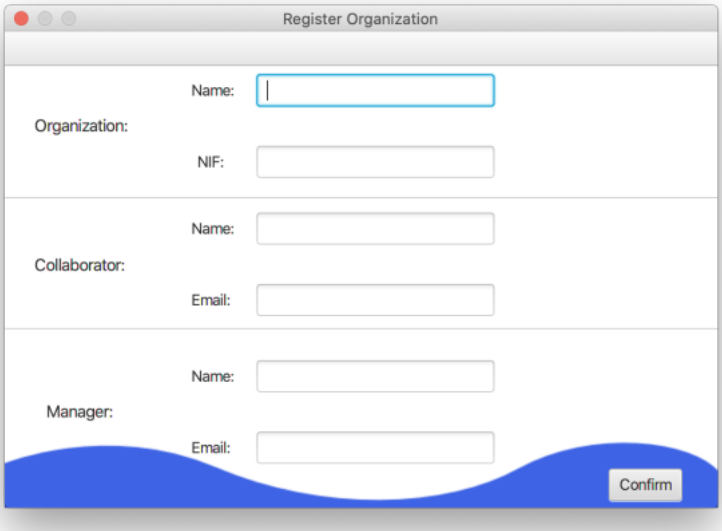

FIGURE 5

4.4. UC2 - Register Freelancer

To register a freelancer the collaborator has access to this window:

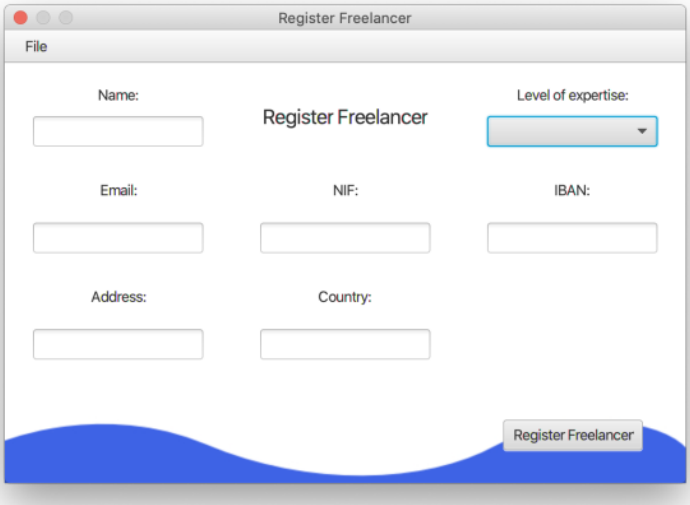

FIGURE 6

In this window, the collaborator has access to one "combo box" and six "text fields" where the collaborator can select the level of expertise of the freelancer from this two levels (Junior, Senior), and also being able to insert the name, email, NIF, IBAN, address and country, of the freelancer you want to register.

If the user wants to register a freelancer, all he needs to do is fill all the parameters and click the button “Register Freelancer”. It will be shown an alert for the user to confirm all the inserted data, similar to this:

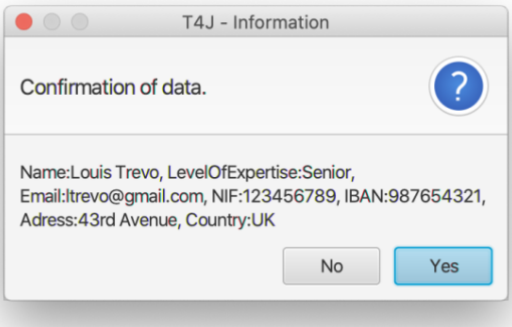

FIGURE 7

If the collaborator confirms all the information, the freelancer will be validated and registered in the system, if your information is valid. If your information is invalid, the collaborator has the option to change the data and reconfirm the freelancer register.

4.5. UC3 - Create Task

This functionality is just able when the user log in the applications as a collaborator. In the main window, there are the possibility to choose the option “Create task”, in the menu bar, that forwards the user to a new window, where will be showed the necessary information to the user be able to create an acceptable task.

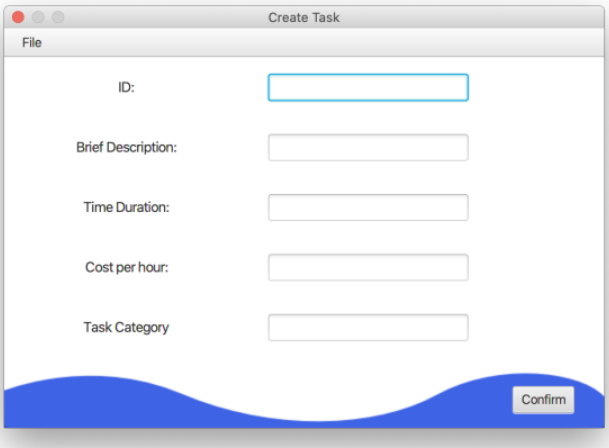

FIGURE 8

In this information is required an ID (inserted by the collaborator, that need to be unique for all the tasks), a brief description of the task, his duration, cost per hour and the task category. After completing the text fields, the collaborator needs to click on the “Create” button to register the task, will appear a confirmation message and, for least, all the data will pass by a validation process.

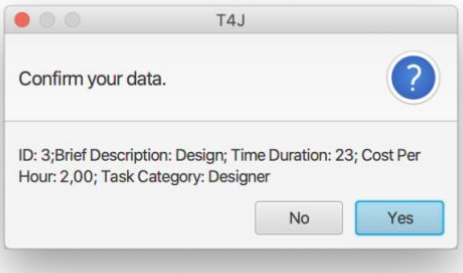

FIGURE 9

If all the data is valid, will appear a successful message, else will appear a message advertising the user for the happened.

4.6. UC4 - Create Payment transaction

To create the payment transaction the collaborator has access to this window:

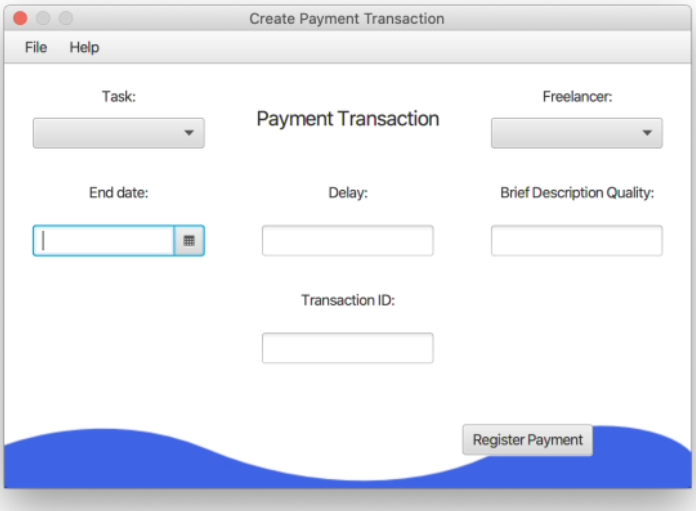

FIGURE 10

In this window, the collaborator has access to two "combo boxes", one "date picker" and three "text fields" where the collaborator can select the task and the freelancer for whom the payment is to be made, in addition to also being able to

entering the date, the delay, a brief description of the quality and the transaction id of the transaction he want to create.

To fill all this data we first need to select a task from a list that is in the “combo box”, and for that we need to get all tasks of the organization of the collaborator currently logged in. We gather the collaborator organization through his email and then we get the list of tasks of that organization. The next step is to select a freelancer also in a “combo box”, and to get the list of freelancers we gather the list of all freelancers registered in the platform and put them all in the “combo box”. After that we need to select a date from the end of the task execution, and to do that we choose the date of a calendar through a "date picker". Afterwards we just need to insert the delay of an execution task and its brief description of a quality of the work and finally the id of the transaction. After all the information is entered the collaborator needs to press the button "register payment" and all the information about the payment transaction is validated by the system and is shown to the collaborator the amount to be payed in an alert similar to this one:

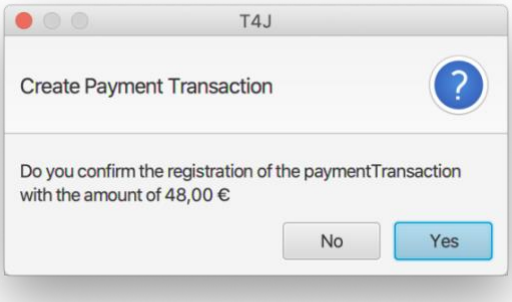

FIGURE 11

If the collaborator confirms the payment transaction amount, the transaction is registered in the system and a payment transaction is created. The collaborator can change the data and reconfirm the transaction if the data inserted isn’t valid.

Software Design Patterns:

For the development of this use case, we had to make its planning documentation as we learned in ESOFT classes. To make the diagrams pf this use case we had to use some software patterns that we learned also in ESOFT classes, being the following:

Type of Grasp patterns:

Pure Fabrication: Use of an artificial class (UI) that does not represent a concept in the domain of the problem, specially made to achieve low coupling as you can see in these image of the sequence diagram of this use case.

FIGURE 12

Controller: non-user interface object, responsible for handling the System event, as you can see in these images of the sequence diagram of this use case.

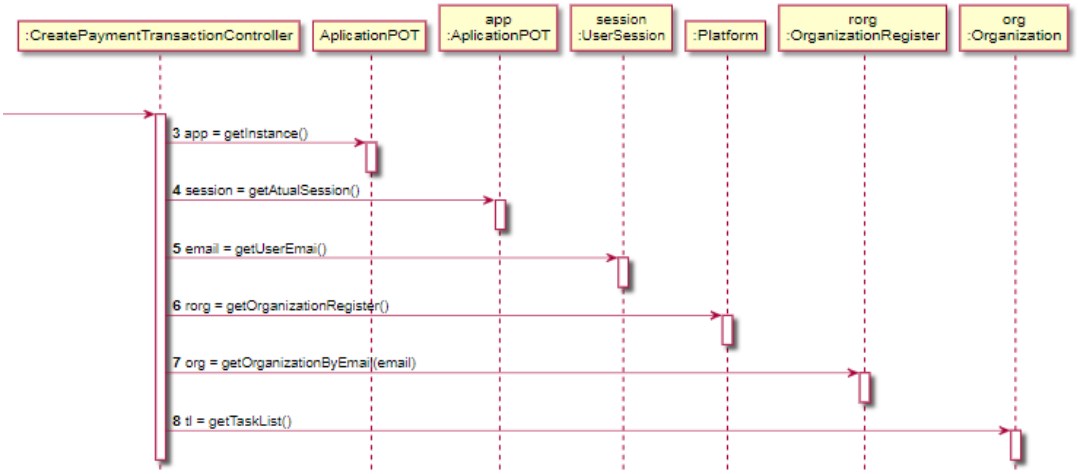

FIGURE 13

Creator: Responsible class for creating instances, in this case payment Transaction objects, as you can see in these images of the sequence diagram of this use case.

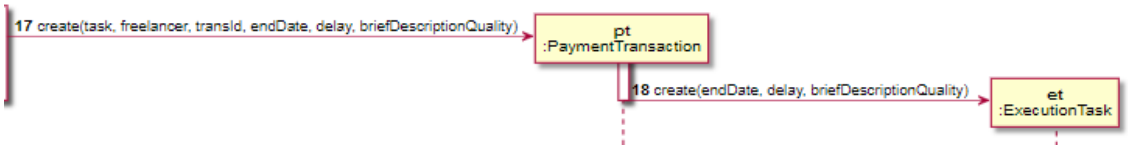

FIGURE 14

Information Expert: Principle used to determinate where to delegate responsibilities, one of its criteria is who has the data to carry out the necessary methods, as you can see in these images of the Rational of this use case.

Organization	Creator (Rule 1): on MD the Organization has Payment Transactions
PaymentTransactionList	IE: in MD the organization has payment Transactions. By application of HC + LC delegates this function to paymentTransactionList

FIGURE 15

High Cohesion + Low Coupling (HC + LC): Is an evaluative standard that tries to keep objects properly focused, manageable and organized, as you can see in these images of the class diagram of this use case, where the class platform delegates is functions to the class OrganizationRegister.

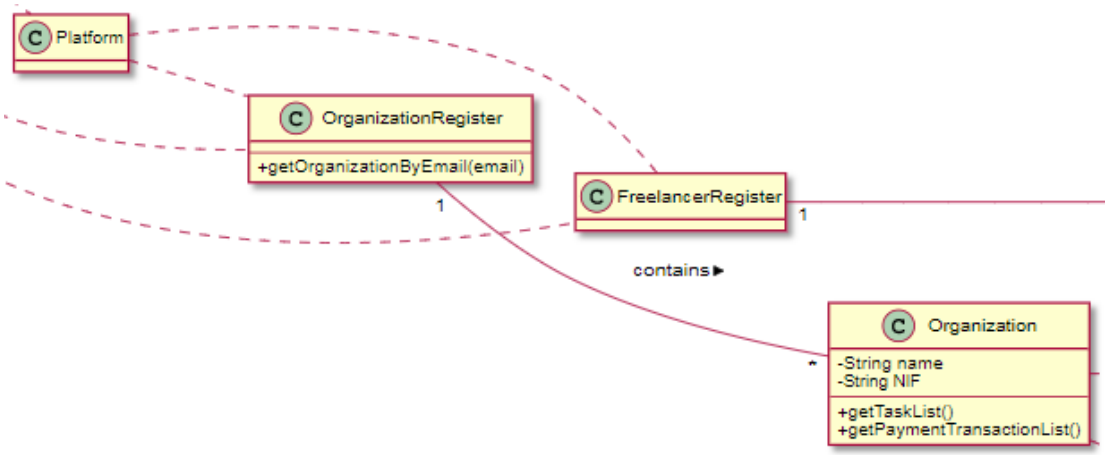

FIGURE 16

These are just small examples of situations where these patterns were used. To see all the patterns used in this use case click here.

4.7. UC5 - Define automatic payment time

In order to make the manager to be able to define when the payment transactions needed to be made we built a window in the application where the manager was able to select the date when he wanted to make the payments and insert at hour of the day the payment will occur as can be seen in the image below.

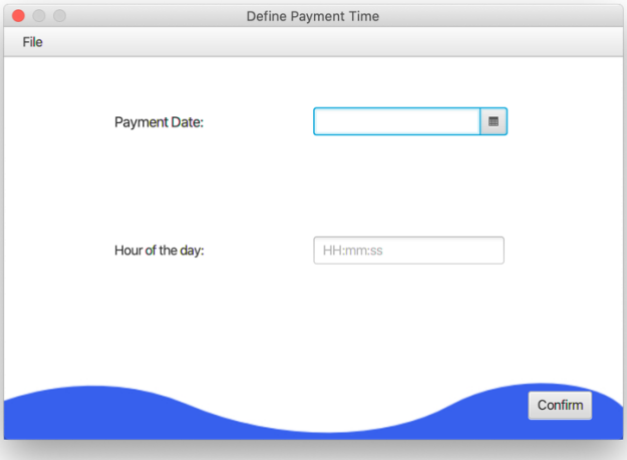

FIGURE 17

If, for some reason, either the date was not picked or the hour is inserted incorrectly (not in the format hour/minute/second or empty), the application will give the user an alert so he can correct that mistake.

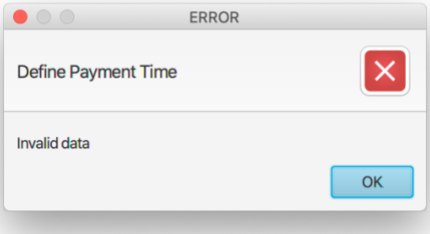

FIGURE 18

This data will then be used to schedule the payment with the task created in UC7.

4.8. UC6 - Load File

When loading a file, the user has the option to select a .txt or a .csv file as shown in the image below.

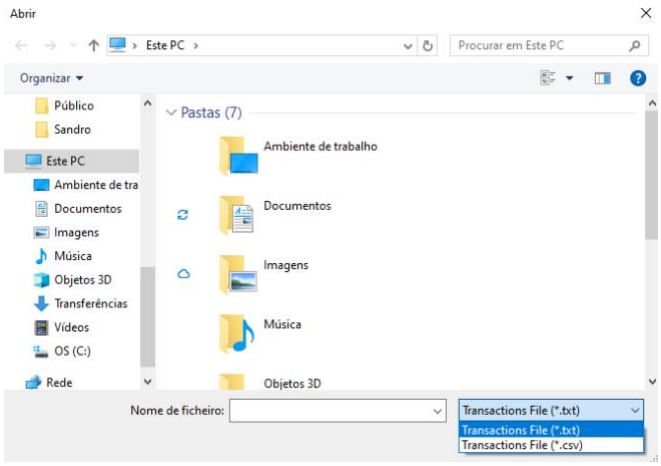

FIGURE 19

We studied and used the strategy pattern as asked. Quoting DZone “Defines a set of encapsulated algorithms that can be swapped to carry out a specific behaviour”. In this particular case it helped us to differentiate whether the user selected a .txt or a .csv file, making it easier to work from there.

After the file has been selected we split its content as we learned in APROG. We then create Freelancer instances, Task instances and Payment Transactions instances, gathering all the file content and registering it on the Platform.

4.9. UC7 - Make an Automatic Payment

To make the automatic payment we first need to get all freelancers that work the organization of the manager currently logged in, for that we gather the list of freelancers registered in the platform and then verify what tasks made by them were the same as the ones saved in the organization. To do that we pick a task of the freelancer at a time and then cover all the tasks of the organization and verify if the tasks are equal. After that we need to convert the amount to pay to the freelancer's country currency, to do that we decided to implement an API using protected variation and an adapter that received the amount in euros and the country where the freelancer resides and converts the price to that country currency by multiplying the amount by the value already registered on the application. Then we write the payment transaction and the amount to pay to the freelancer for that task, then after all tasks of that freelancer had been covered, we send the e-mail with the receipt to the freelancer which is saved in the file "e-mail.txt". This process is repeated until all freelancers are cleared. We also save each payment transaction in a specific file so that the T4J administrator can check all transactions made by all organizations.

4.10. UC8 - Show Freelancer statistics

We created two scenes to show overall statistics, one about task execution times and another one to freelancer payments. Only the manager and the collaborator have access to these statistics, as asked.

The task execution times statistics scene shows us the mean of the delays, the standard deviation of the delays (calculated with the Figure2 formula), and two histograms. The application developed gives the user the possibility to choose a freelancer and make a histogram to analyse its delays, but the user also has the option to analyse all the freelancers delays working to the organization in a single histogram.

$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

σ = standard deviation

$\sum$ = sum of

x = each value in the data set

$\bar{x}$ = mean of all values in the data set

n = number of value in the data set

FIGURE 20

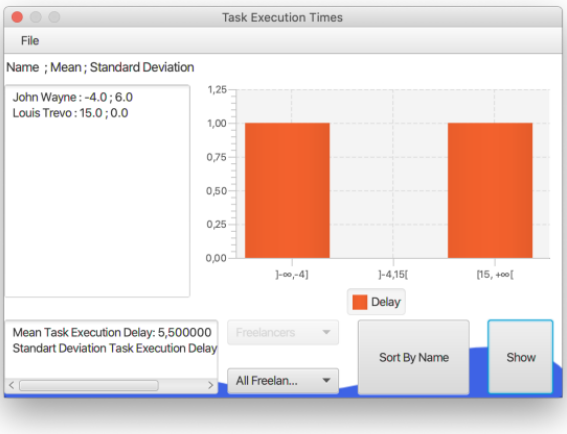

FIGURE 21

In the other hand, the payment transaction scene only shows the mean payments made to each one of the freelancers and the standard deviation of the payments, also calculated in accordance to the Figure2 formula.

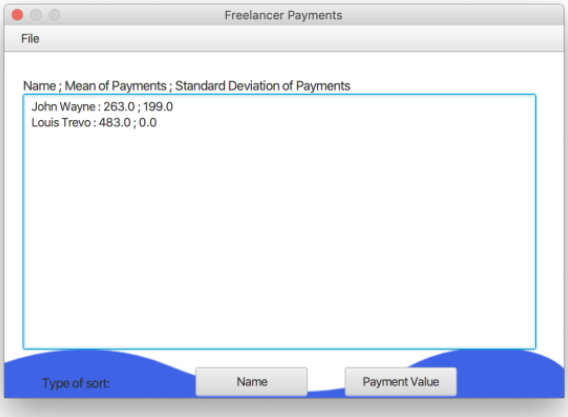

FIGURE 22

4.11. UC9 - Show general statistics

This is an option only able to use by the T4J administrator. At any time, he is apt to see and analyse the general statistics about freelancer's performance, including the mean and the standard deviation of the delays and payments and histograms to analyse these parameters for all the freelancers that exist in the system.

This window has, on the left, a text area that will contain the information about each freelancer, that is the freelancer name, the mean and standard deviation for his delay and the mean and standard deviation for his payments.

The values of the mean delay are calculated by running all the tasks associated to a freelancer, adding the delay of each one and dividing by the number of task that the freelancer has done. To calculate the standard deviation, we use the method as a

parameter to obtain his mean delay and a freelancer. The system will run the list of task associated to that freelancer and get his delay again, this value will be subtracted whit the value of the mean delay passed by parameter and the result will be squared and added to the previous results from another tasks. To finalize, we used the native method *Math.sqrt()* to execute the square root of the addiction of all the deviations by the number of tasks performed.

The methods to calculate the mean and the standard deviation of payments follow the same procedure, just changing getting the delay to getting payments values.

This is followed by two histograms, where their data will be defined whit the options on the combo boxes under them. In one combo box we have the possibility to choose if the administrator wants to see the histograms whit the data about each freelancer registered in the system or about all the freelancers. When the administrator chooses the option “Each Freelancer”, a second combo box will be able to use, showing all the freelancers registered in the system. After choosing one of them and click on the button “show”, the freelancer statistics selected will appear on the histograms.

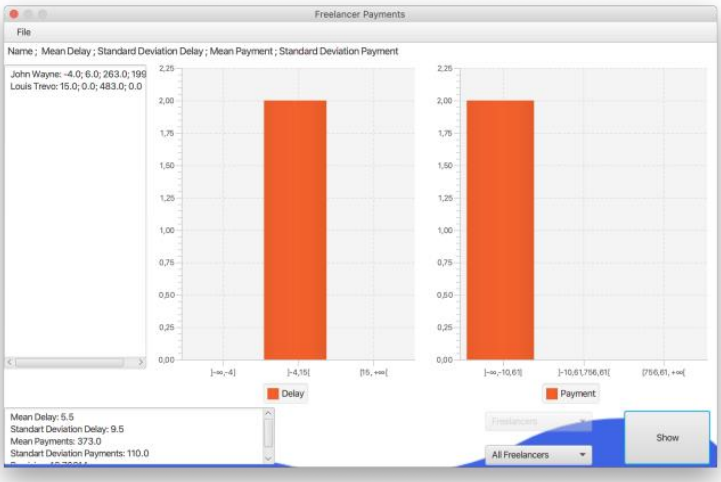

FIGURE 23

Another text area located under the first one, to present the statistic of all freelancers together. This information contains the mean and the standard deviation delay of all freelancers and the mean and the standard deviation payments of all freelancers. It also shows the probability that the sample mean is higher than 3 hours, considering the task delay is normally distributed with a mean of 2 hours and a standard deviation of 1.5 hours.

This function uses the native method *NormalDistributionImpl*() of java, using as parameters the mean delay given in the assignment and the standard deviation calculated previously.

Constructor Detail

NormalDistributionImpl

```
public NormalDistributionImpl(double mean,
                              double sd)
```

Create a normal distribution using the given mean and standard deviation.

Parameters:

`mean` - mean for this distribution
`sd` - standard deviation for this distribution

FIGURE 24

4.12. UC10 - Notify delay to Freelancer

This task is executed on the last day of each year and it is scheduled on the start of the program. This notification should be send to all freelancers that had a mean task delay time(during the year in which this operation occurs) higher than 3 hours and have a percentage of delays that is higher than the overall percentage of delays. In order to send the notification(email) about the delays first we need to get a list of all the freelancers registered in the platform. Then, we calculate the general percentage of delays.

For this we get the tasks of each freelancer(for each task that was executed during the current year we add 1 to the variable “numberTasks” to get the overall number of tasks) and get it´s delay(we access the task’s payment transaction and the payments transaction’s execution task for this). If this delay value is higher than 0 we add 1 to a variable called “numberDelays” which we will use later. After this we divide the variable “numberDelays” by the variable “numberTasks” and multiply it by 100 and therefore, we get the overall percentage of delays.

Then we need to calculate the mean delay and the percentage of delays of each freelancer. In order to do this, we go through the list of freelancers and get its tasks. We then add 1 to the variable “nTsk” for each task that was executed by a certain freelancer during the current year. We also get each task’s delay and add it to the variable “sumDl”. If the delay value is higher than 0 we also add 1 to “nDl” (the variables nDl, nTsk and sumDl are initially reset to 0 for each freelancer). Afterwards we divide “sumDl” by “nTsk” to get the mean delay, and divide “nDl” by “nTsk” and multiply by 100 to get the percentage of delays. Finally, if this freelancer’s mean delays is higher than 3 and it´s percentage of delays is higher than the overall percentage of delays an email is sent to the freelancer regarding this information. This will occur until all freelancers are cleared.

5. Conclusion

With this project we got our first glimpse of what real programming is and its difficulties, even though the situation we are living made the work harder, our group did not struggle when it came to hard-work, communication and commitment. With this project we developed our group communication and our programming skills.

We tried to make a simple but eye-catching application following all the requests given.

Dividing our work by use cases according to the methods learned in ESOFT made our work easier because we were able to analyse the requirements piece by piece instead of one big block of work. This gave us a perspective of how the classes were connected in Java.

The Model–View–Controller and the GRASP (General Responsibility Assignment Software Patterns) pattern as well as the object-oriented programming were the core of this project and it made us all realise how vast the programming market can get.

Our first steps on the application-developing were made.

References

[1] “Commons Apache,” 03 01 2011. [Online]. Available: http://commons.apache.org/proper/commons-math/javadocs/api-2.2/org/apache/commons/math/distribution/NormalDistributionImpl.html. [Accessed 09 06 2020].

[2] ISEP, *Laboratory/Project 2 - Project Assignment Payments Management,* ISEP, 2019/2020.

[3] ZZT32, “Wikipédia,” 14 04 2007. [Online]. Available: https://pt.wikipedia.org/wiki/Ficheiro:ASCII-Table.svg. [Accessed 05 06 2020].

[4] J. Sugrue, “DZone,” Devada Media, 01 03 2010. [Online]. Available: https://dzone.com/articles/design-patterns-strategy. [Accessed 08 06 2020].

[5] A. Redko, “Oracle,” 01 2014. [Online]. Available: https://docs.oracle.com/javafx/2/charts/bar-chart.htm. [Accessed 09 06 2020].

[6] “Moodle,” [Online]. Available: https://moodle.isep.ipp.pt/. [Accessed 06 2020].

ANNEXES