LABORATORY CASE

Cuenta de gastos

Interfaces Persona Computador

IPC – DSIC

UPV

Curso 2023-2024

Index

1. Description and usage scenarios

We want to develop a desktop application that allows registered users to control and monitoring their expenses. The application must render well on any device and ensure usability and satisfaction.

To use the application, the user must be registered. Once registered and logged in, the user will be able to access the different functionalities of the application. The user should be able to logout without the application exiting.

The user will be able to add expense notes to their history at any time, as well as view the history of all their notes. Information such as the date of the expense, the product, the units and the price will be saved for each note. Optionally, an image of the ticket or invoice will also be saved to be used to claim the product warranty.

The use scenarios obtained after the system requirements analysis are detailed below. These should be considered to properly design and implement the required application.

1.1. User register

Alberto has seen that in the IPC subject they have developed an application to monitor personal accounts, since he has to save money and does not know how he spends it, it seems like a good idea to use this application. After downloading it, Alberto accesses the registration option. From the option a form opens where Alberto can enter his contact information:

- Name
- Nickname (used to access the application and to be shown to other users. Cannot contain spaces. Cannot be repeated in the application)
- Password (any combination of letters and numbers with more than 6 characters)
- Email
- Profile image (optional)

After entering all contact details and choosing a squirrel as your profile image, the system checks that the information is correct and no field contains an incorrect data format. The system also saves the date on which Alberto is registering. All the information entered by Alberto is correct, the system displays a message indicating he has been successfully signed up and he should log in to use the application.

1.2. Authenticate

Pilar has just remembered that she filled the gas tank yesterday and decided to write down the expense before she forgets. As soon as she opens the application, she accesses the authenticate option, where a form appears. In this form she enters her nickname and password. After accessing, the system checks if the user is registered in the system.

Pilar made a mistake when entering the password, the application cannot match the data and displays a message to the user. Pilar is advised to try again, and she enters the nickname and password again. This time the data is correct, the system authenticates and allows the user to access the rest of functionalities.

1.3. Add an expense or note to the account.

Alberto wants to write down in his expense account what today's meal cost him. After authenticating, Alberto accesses the option to add expense. As he is adding the expense, realizes that it would be interesting to know how much is spending in restaurants, so he decides to create a new expense category. Alberto accesses the create category option and add the "Restaurant" category. Alberto continues with his task, adds the cost, the date of the expense, a title or name, a short description and selects "Restaurant" as the category.

Alberto also remembered that yesterday he also bought some Bluetooth headphones, so he adds this other expense. After entering all the data, he also adds a screenshot of the invoice so he can have it available in case he needs a warranty claim.

1.4. View expense account

The end of the month arrives, and Pilar does not understand how her bank account is so in bad shape. She wants to see the expenses she has made. After authenticating, Pilar uses the available options to view the monthly expense, in total and grouped by her different categories.

Since Pilar has been using the application for some time, she also wants to see how much she has spent this month compared to the rest of the months of the year. Besides, she wants it compared to the same month of the previous year.

This scenario is open for you to design it as you consider most appropriate. You can use filters of any type and graphics. Javafx incorporates graphs that you can use because they only require data observable lists.

1.5. Print expense account statement.

Alberto wants to show his partner the reasons why he has not saved anything this year. After authenticating and browsing his account, he generates a PDF report that shows the evolution of his annual expenses.

1.6. Update user data.

Pilar wants to update her profile data. After authenticating, she accesses the option that allows her to update her profile data. After making the changes Pilar intends to make (except her nickName, which cannot be modified), she tells the system that wants to save the changes made. After verifying that the new values meet the necessary requirements, the system saves the information and says goodbye to Pilar.

1.7. Update expense.

Pilar has just found the receipt for last Monday's meal that she has already entered as an expense, so she wants to add the scanned image of the receipt. After logging in, Pilar accesses the option to modify an expense, and proceeds to upload the scanned image and close the application.

1.8. Delete a charge.

Alberto has just received an email from the company confirming that they are going to pay for the gas for his last trip. Since he has already recorded the expense in his account, he decides to access sand delete it. After authenticating and searching for the note, he proceeds to delete it.

1.9. Delete a category.

Alberto defined several categories that he has not used after some time. He has decided to eliminate them so that they do not appear in the status display of his expenses. After authenticating, it searches for the categories and deletes them.

2. Data model

The data persistence should be implemented using an SQLite1 database. This is completely transparent for the developer. The access library acountIPC.jar is provided. It enables you to store and retrieve all the information that needs to be persisted. Also, it defines the data model required. To get the library working properly, it is mandatory to include in the project the sqlite-jdbc-3.41.2.1.jar. Both files are available for download at Poliformat.

It is recommended that these libraries get unzipped in a new folder "lib" inside the NetBeans project directory. See Figure 1.

The projects created by NetBeans for Java applications brings in a `Libraries` package that we can use to add third-party libraries that are necessary for the project. We can do right-click on the `Libraries` package and from the context menu click on Add JAR/Folder, see Figure 2.

In the Add Jar/Folder dialog, select the lib folder we created earlier. Then, select both .jar files and click on the Open button. NetBeans will add the libraries to the project and as you can see in Figure 3.

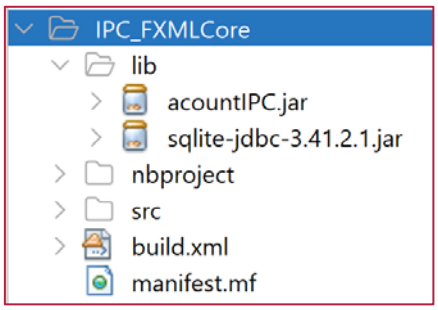

Figura 1. Carpeta lib

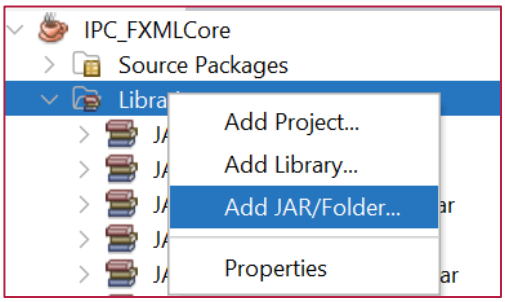

Figura 1 Seleccionar la opción de incluir la librería

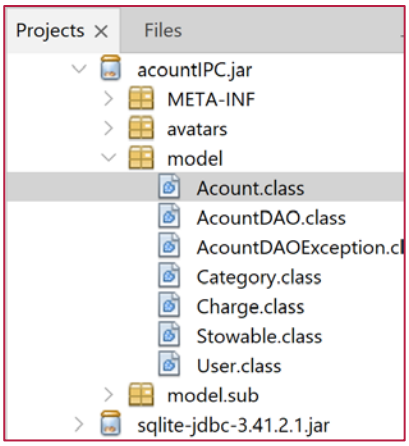

Figura 3 Clases en acountIPC.jar

2.1. Model classes.

As we have seen in class, the primary objects of the problem are user, category and charge (**User**, **Category Charge**). The data model defines multiple classes. The ones that you will need to develop the application are User, Category, Charge and Account.

The model has been structured in such a way that when objects of these classes are created, or when their fields are modified, they are saved in an SQLite database- Thus, every time we start the application, we will have the same objects that we had when we closed it.

To create and recover the objects we have created the **Acount class** in which the necessary methods have been defined to be able to register objects and to be able to recover them when starting the application.

This java class implements the Singleton pattern; it means that only one object of this class can be created in the application at the same time. This is the reason why the constructor has been defined as private, and there is a static method in the class that only creates this object and returns it. This method is **getInstance()**. For practical purposes it will reduce the complexity of the code since it will avoid passing objects between the different controller classes.

To facilitate the development of the practice, an object of the User type has been defined in the Account class that serves to store the user logged into the application.

User

The User class stores all the information related to the users of the application, the attributes of the class are:

- String **name:** username.
- String **surname:** user's last name.
- String **email:** string containing the user's email.
- String **nickName:** username used to authenticate in the system.
- String **password:** password used to authenticate in the system.
- LocalDate **registerDate**: date of registration of the user in the application.

- Image **image:** member's profile picture. If it is not used by the constructor (null), a default one gets automatically added.

Every attribute can be updated except nickName. To do so, the class should contain different setters.

The constructor of this class is not defined as public. To create a new user, it is necessary to invoke the Account class method, registerUser(). This method will create a new user and register it in the account. The method returns a boolean, true if it has been created.

This class has static methods to validate the nickName, email, and password fields.

Category

The class attributes are:

- String **name:** name of the category.
- String **description** – description of the category

The constructor of this class is not defined as public. To create a new category, it is necessary to invoke the Account class method, registerCategory(). This method will create a new Category attached to the logged user. Therefore, there must exists a logged in user.

Charge

The class attributes are:

- Int **Id:** it is an automatic field that is used to identify the object in the database, it is generated at the time of recording the expense.
- String **name:**A title of the expense.
- String **description** - String containing the description.
- Category **category**: expense category
- Double **cost:** cost of the expense.
- Int **units:** units
- LocalDate **date:** date of the expense.
- Image **scanImage** - scan of the invoice (optional).

Every attribute can be updated except **id**. To do so, the class should contain different setters.

The constructor of this class is not defined as public. To create a new object, it is necessary to invoke the Account class method, registerCharge().

Account

This object enables access to all stored data using public methods. Also, the associated setter and getter methods have been already implemented to facilitate the development of the project. Table 1 describes the functionality for all these methods.

Table 1. API for Account class

public static Acount getInstance()	Creates an object of the Account class, if it was not previously instantiated. If it has already been instantiated, it returns the same object. When it creates it for the first time, it checks if the DB exists and loads its information. If the DB does not exist, it creates one with all tables that are required as well as populate them with the default data. This DB file created can be edited with database tools such as DB Browser that you can download at https://sqlitebrowser.org/
public boolean registerUser(String name, String surname, String email, String login, String password, Image image, LocalDate date)	Registers a new user. This new user it is s NOT logged into account. Returns true if the user has been successfully registered
public boolean logInUserByCredentials(String login, String password)	If there is a match credentials entered, this user is saved within the Account class as a logged in user. Subsequent queries about categories and expenses are made on this user. If there is no match with these credentials, it returns false.
public boolean logOutUser()	The user logged in to Account is removed. Subsequent queries about categories and expenses will return null. If there is no logged in user, it returns false.
public boolean registerCategory(String name, String description)	Registers a new Category for the logged in user. If it has been successfully registered, it returns true.
public boolean removeCategory(Category category)	An existing Category for the logged in user is removed. It also removes all the Charges of this Category. If it has been successfully removed, it returns true.
public List<Category> getUserCategories()	Returns a List with all categories of the logged in user. If it is an empty list or there is not logged in user returns null

public boolean registerCharge(String name, String description, double cost, int units, Image scanImage, LocalDate date, Category category)	Registers a Charge on the logged in user. the ID of this Charge is generated internally and added to the charge created. Returns true if successfully registered.
public boolean removeCharge(Charge charge)	Deletes a charge based on its ID. Returns true if it has been successfully deleted.
public List<Charge> getUserCharges()	Returns a List with all Charges of the logged in user. If it is an empty list or there is not logged in user returns null

2.2. Using the library from the project

To access the library methods, it is necessary to first instantiate an object of the *Account class,* using the static *getInstance* () method. The registered information can then be obtained or modified through the provided API.

```
boolean isOK = Acount.getInstance().logInUserByCredentials("IPCuser", "123456");
```

3. Programming aids

3.1. Loading images from hard drive

There are several ways to load an image from the hard drive and display it in an *ImageView* :

1. The image is in a subdirectory of the project's src directory, for example, called images:

```
String url = File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

2. The image is anywhere on the hard drive and we have the complete path of the image::

```
String url = "c:"+File.separator+"images"+File.separator+"woman.PNG";
Image avatar = new Image(new FileInputStream(url));
myImageView.imageProperty().setValue(avatar);
```

3.2. Date and time management

The application should be able to manage attributes type LocalDateTime, LocalDate and DateTime. In this section, you can find some methods that might be useful.

Getting the week number for a given Date

The following code gets the current week number, as well as the number of the day within the week (1 for Monday, 2 for Tuesday, etc.)

```
WeekFields weekFields = WeekFields.of(Locale.getDefault());
int currentWeek = LocalDate.now().get(weekFields.weekOfWeekBasedYear());
int numDayNow=LocalDate.now().get(weekFields.dayOfWeek());
```

Creating a date or time field

You should check out the LocalDate and LocalTime API to learn all methods that are available for creating an object of this class. Among them, these two might be useful:

```
LocalDate sanJose = LocalDate.of(2020, 3.19);
LocalTime mascleta = LocalTime.of(14,0);
```

Updating a date

It is possible to increment or decrement a LocalTime, LocalDate, o LocalDateTime by days, months, years, minutes, hours, etc. using their own API. For example, the following code increments by 7 days the current date and updates a new variable with the result. Also, it increments by one month the current date, and updates another variable with the result. Finally, it increments the current time by 90 minutes, and updates a third variable, endedTime, with the result.

```
LocalDateTime nextWeekDay= LocalDateTime.now().plusDays(7);
LocalDateTime nextMontDay = LocalDateTime.now().plusMonths(1);
LocalTime endedTime = LocalTime.now().plusMinutes(90);
```

3.3. Configure DatePicker

The DatePicker components allow the user to select a date, being able to access the selected value through its valueProperty() property. It is possible to configure the way in which each day of the calendar is displayed by default, being possible to disable some days, change the background color, etc. The way this display is configured is like the way we use to configure a ListView or a TableView. The difference is that in this case we must extend the DateCell class and use the setDayCellFactory method. For example, the following code configures a DatePicker to be disabled the days before March 1, 2020 (see Figure 4).

```
dpBookingDay.setDayCellFactory((DatePicker picker) -> {
return new DateCell() {
@Override
public void updateItem(LocalDate date, boolean empty) {
super.updateItem(date, empty);
LocalDate today = LocalDate.now();
setDisable(empty || date.compareTo(today) < 0 );
                }
};
});
```

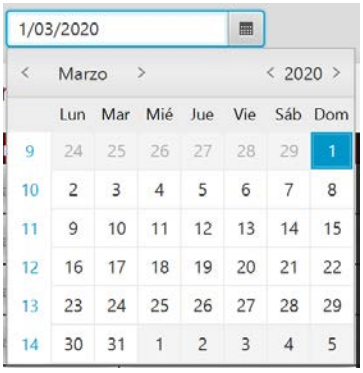

Figure 2. DataPicker configured to disable days in the past

4. Delivery Instructions

The project must be implemented taking into consideration every scenario from Case Study (page 2, 3 and 4). The project should follow the principles and patterns that have been explained during the course lectures.

Although documentation generated during the development process is not required for delivery, it is requested that the process must conforms a user-centered development seen in the subject. This imply that task analysis, conceptual design, and physical design with prototypes must be done.

Please read the following bullet-points carefully before sending the final project delivery:

Regarding delivery:

- Export the Netbeans project to a zip file (`File > Export Project > To Zip option`). As an alternative, you can zip the project folder. If you want to reduce the size of the file, the folder dist can be deleted from the ZIP file.
- Only one of the working group members should upload the zip file to Poliformat. As part of the process, he/she will fill out the comments text field with the name.
- The deadline for every group is the same one **May 26, 2024**

5. Evaluation

- Projects that do not compile or do not show the main screen at startup will be scored with a zero.
- Confirmation dialogues, errors, etc. should be included as deemed necessary.
- To evaluate the design of the application interface, **the guidelines studied in theory class will be taken into consideration.**
- It should be possible to resize the main screen, whose controls will be properly adjusted to use the available space (use the containers seen in class: VBox, HBox, BorderPane, GridPane, etc.)
- The UPV Academic Integrity Regulations and the ETSInf Academic Honesty Regulations will be applied. Anti-plagiarism tools are available in the course.