December 16, 2020

Beatrice Antoniu

w1688177

enterprise application development

coursework 2

# Design changes and reasons

I have not made many changes to my design as I tried to stick to the plan, I have made in Coursework 1. Starting with the *Functional Requirements*, I have fulfilled most of the requirements I have listed. I have created a login system that requires the user to register and then login to have access to the application and even retrieve their account. The changes I have made in this section was only the way of retrieving a lost password, I have initially suggested that the user would be prompted for their secret question but for security reasons I thought prompting for the email, which is a primary key in the database table I have created, would be more efficient because the secret question could be the same for two accounts. Although I have made this change, in the registration section I still have the secret question as an input. Another feature I have modified was the *recurring expenses*, my system can only add one-off expenses that only give details of who made the transaction, the date they have made it and the amount they have sent or received. This change makes the application a bit vague as it is left to the user interpretation because as long as they are logged into the system they can add or remove transactions and contacts. I do not keep track of the user that logged in and so I see the login system as a way for the manager of the software to access its features. Basically, the way I have made this application does not focus on who is using the software and who’s expenses are going to be predicted. I have also modified a little the way I produce a report, in the requirements I have said the user could select a payer or payee in order to produce a report but because of the way the login system works this feature would not be relevant and so the user can select a date and it would produce a report of all the expenses of that date. *Pre-set contact list* was another feature I proposed but ended up not adding to the final product for the same reason as *producing a report*, the user can still add or remove contacts, but they cannot modify any information and the contacts list is not linked to the expenses list in any way. *Logout and Exit buttons* were two things I have removed completely because I saw no point in having an Exit button since I could not remove completely the window option to exit the program. As for the Logout button, as mentioned before, I do not keep logs on who is logging in so the logging out option would not be necessary unless I would keep this data. Now the *Non-functional Requirements* I have completely kept, the program can run only on Windows, it was implemented with C# and Windows Forms and it does require the user to register and login in order to use the software.

# Evaluation

I have kept my design simple and straight-forward, the software starts with the login window which consists of a button, two textboxes, one for the username and one for the password, and two links to go to the register window and to prompt for password recovery. The fields were validated to match the database entries and to never be left empty and so the user would get a message if the details do not match or the fields are not filled. The register window consists of five textboxes, a combo box and two buttons. The username and password fields are validated to have at least eight characters, the email field has a regex validation to match the standard email format and altogether the field cannot be empty. There is a return button for the user to go back to the login window and a register button that saves all the details to the database. The user will be prompted with message boxes for every validation and for a successful or unsuccessful registration. The recovery link consists of an input box where the user can enter their email in order to retrieve their password, a message box will appear with the password if the details are correct and another message box will appear if they are not. Once they logged in a message would also appear and the user would be redirected to the main page. I have not made any encryption or decryption for the passwords and that would be a security issue because the way of recovering the account is simple. The design of the main page is matching the login system, consists of an empty window with a dock panel at the top that has four buttons. Once the use double-clicks a button in the empty part of the window a page would be loaded. I have chosen this style in order to act as a dashboard where you can have the menu at all times.

Starting with the *Transactions* button that leads to a page that has a data grid and three other buttons, this page will display the list of transactions added to the database with the name, date, and the amount of the transaction. There is a button to add a new expense or transaction and a button to remove those. In order to update the list after the data has been changed, I have created a button that refreshes this list to the current state; loading the page also refreshes the list. The *Add* feature has a validation to format the date according to the way the database stores are in order to work with it later on. The *Update* option also has a validation if it cannot display anything or if there is a database error, although not fully tested, a known issue in the beginning was that if the date would not have been formatted then the transaction would add to the database as a 0000-00-00 type of date and then the *Update* button could not display the list anymore unless the entry would have been deleted straight from the database. I have succeeded in fixing this issue later on with the formatting. The *Remove* function has also one validation for “inexistent transaction”. At the bottom of this page, I have added a label in which I return the sum of the whole expenses list. An efficiency problem in this section would be that I have used the same method twice, one for the button click and one that I instantiate in the constructor.

Going to the next page, *Predictions*, this contains two date pickers, a button and labels that return the prediction algorithm. On double-click of the button the user gets a prediction of their expenses over a year and six months, it also displays the sum of the expenses of the selected period and the period in days. I have formatted the dates as well to fit the database format in order to work and not create any errors. Initially there was a problem with displaying the algorithm as I was not using a float and it was breaking the code if the division was not exact, but I have fixed it by converting the values to floats with two decimals.

Next page would be, *Contacts,* which is really similar to the *Transactions* page. Works in the same way, you can Add, Remove and Update the list. It has validations for email input made with regex and for “inexistent contact” and it also has the same efficiency problem as the other page.

Last page is the *Reports* page where the user can produce reports of their expenses at a chosen date. Consists of a date picker, a button, and a data grid where the entries of the selected date are displayed. As well the *Prediction* page, the date from the date picker is formatted to fit the database format.

Altogether the code is clean and simple with well-defined names so it can be easily known what we are working with. I have not added comments as the code is similar in the forms. I created a database class separate from everything to initialise the database and have a few methods that can be easily called from it. The database server and connection I have created with Heroku online and I have managed it with MySQL Workbench. I tried to make everything as simple and efficient as possible in order to keep it maintainable and readable.