The Expense Tracker

Problem and solution description

The scope of the Expenses Tracker is to aid the users in tracking their daily expenses based on chosen categories such as: food expenses, basic utilities expenses, transportation expenses and so on.

All the different types of expenses are then added to a total which signals the total amount of money spent in a single day. On top of letting the user add daily expenses, the application also has a history of all the expenses in the past years and the user can check anytime the amount of money spent in any day over the year. It can also see the expenses made during a certain period of time or monthly basis on a calendar. To make the application more appealing, it features different types of graphs that showcase expenses in a set amount of time. Moreover, it also has a budgeting function that lets the user set a monthly budget and a budget based on the categories of different expenses.

The application supports multiple user accounts, allowing each user to have their own login and thus their own information about their spendings.

The application was made with Java to support the back-end and with Java FX and Java FXML for the interface. On top of that, a database made in PostgreSQL holds all the information about users accounts, passwords, expenses, type of expenses, dates when the expenses were made and so on.

The project follows the Model-View-Controller Pattern for an easier reading of the project’s code. In this project, the model represents all the computations on different data regarding the expenses and any operations done using the database. The View represents the user interface, the components of it, more specifically, and how they interact with each other and is made of the FXML files. The Controller, as the name suggests, represents the class which connects the View Class and the Master Class and all the interactions between the back end of the program and the front end.

The different “pages” in the application are made using distinct models, views and controllers.

Classes and interactions explained

First, the majority of the Model Classes (the Managers) inherit from a superclass called **CreditentialsManager** that has the soul purpose of hiding the connection details to the used database. It has a main method initialize that provides the connection details from a file. A Properties Object is used to read the data from an InputStream that is the file read and the information is collected in several String variables.

The main class that provides the current’s date expenses is called the **MainManager.** It has multiple variables that hold the sum spent on the current date, a LocalDate variable that holds the current date of the year and two static methods that hold the current user id that is being used and its total budget. It has several functions that work with the database:

* -init() -> connects to the database and checks if the combination current user and current date is present, if not, then it is inserted with the new index gotten from the function getMaxId
* getMaxId, getMaxId2() ->functions that get the maximum ids in different database tables and return them
* add\_expense(int value, String type, String desc) ->principal function in adding a new entry in the database, the input for the function is gotten from the MainController, where the user inputted a cost and a description of an expense.
* calc\_sums(LocalDate date) ->function that works again with the database and has the purpose of getting the total amount of money spent based on a category in the date given by the input date in the function’s header. The total sum is also then computed based on the categorized sums.
* getdatabaseBudget() ->simple function that has the purpose of extracting the budget of the user

The **HistoryManager** is the Model used for the History Page of the project. The

page features the expenses made during the last week and last month prior to the current date and a pie-chart with those expenses is made, as well as a list with the dates, details and costs of the expenses done in that time. In order to do that, a vector of Strings is needed for all the details, as well as its index. LocalDates are used to get the dates for the last week’s and last month’s start date and finish date. Integer variables are also used to hold the total sum during the time required based on the categories. The main functions used are:

* getweekdates(), getmonthdates() ->self-explanatory names, an object ZonedDateTime is used to get the dates as it features the methods to substract weeks and months from a date in order to get the desired ones
* getsumweek(), getsumonth() ->functions that access the database and fetch from it all the rows of data, depending on the start and end dates, based on a category and the current user. Then, the data is saved in the vector of strings and later used by the HistoryController.

The **FilterManager** is the model that behaves like the HistoryManager in the idea that it needs to capture data between two dates, but this time, the dates are more randomized, the user being able to choose any month of the year, or any day or even a period of time for the data that they want to see. The fetch on the database is done similar to the one in the HistoryManager, the information being gotten categorized, but on top of that, indexes for the different categories are needed to store differently the data, thus this time 5 vector of strings and 5 indexes are needed and the information is differently inserted into them, but the querry is identical to the one before.

The **CalendarManager** is a model linked to the filter manager in the idea that it gets information from the database in a certain day, but more transformations are needed in order for the fetch as the day, month, year are given consecutively and thus those need to be added together to form the correct input for the database querry. The information is given sequentially as this is the way part of the calendar page was implemented.

Now, the **Controllers** for the different pages will be presented to show the way the back end interacts with the front-end. The controllers have a common function and methods for going from page to page and a common method that does the initializations for the new page’s load.

The method setStage() is the method used to initialize info on the screen. It is a function that is given by the interface InitStage and is used by almost all of the controllers. Going from page to page is done by:

-loading a new FXML file by using a FXLoader object, with the page we want to go to

-creating a new Scene with the loaded fxml

-creating a controller, corresponding to that fxml file and setting the stage of the fxml file with the current stage that is being used

-setting the scene of the current stage with the newly loaded scene

The **MainController** has some initializing functions for the graphs and the labels, but the main function is: ‘addExpense’ that lets the user input a type of expense, a description of it and the cost. After pressing add expense, a dialog pop up with a drop-down menu appears to choose a type of expense, then, if OK is pressed, the user needs to input a whole, positive number for the price and finally a last pop-up appears so the user inputs a description linked to that expense. The function for adding an expense, in the MainManagers is called to update the database.

The **SecController** is the controller of a secondary page that has the sole purpose of displaying to the user more detailed information about the expenses they have made in that day. The page loads based on 5 buttons which all control the same scene, but with changed labels and different data from the database, corresponding to the 5 different type of expenses.

The **LoginController** has the functions necessary for the user to connect to an account. The passwords entered by the user when they create the account are hashed and when checking for the correct password during the login, the entered password is hashed using the same encrypting algorithm and compared to the one stored in the database. If they match, the user can further go to the main page.

The **CreateAccController** is used to register new encrypted accounts in the database. The user inputs an username and a password and if they met the conditions, the username and the encrypted password are stored in the database.

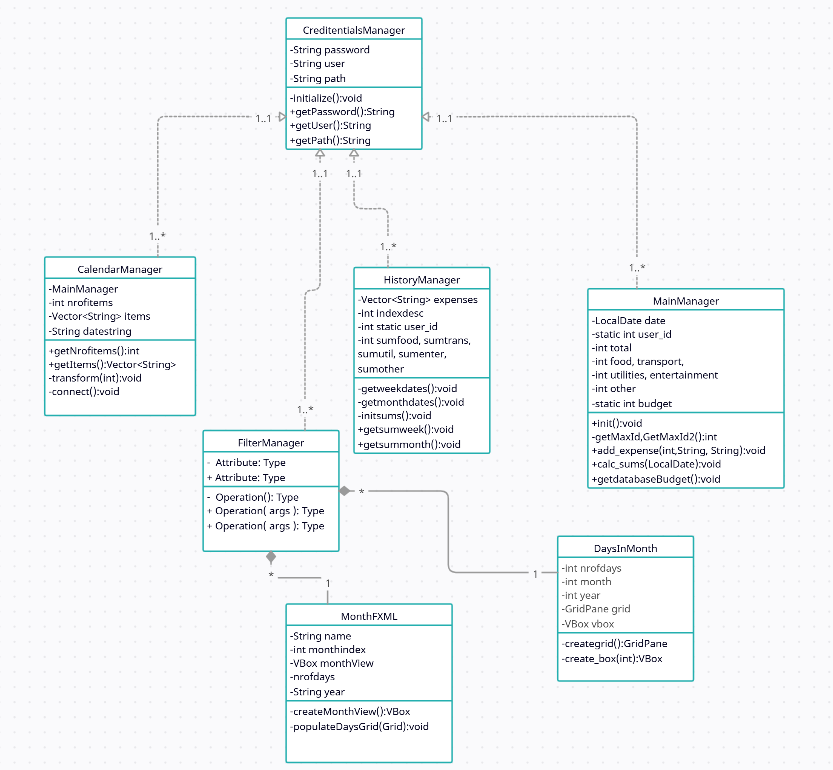
The **HistoryController** controls the history page. The information of the page shown is based on 2 buttons: “last week” and “last month”. The information of the pie-chart is fetched from the model of the history page and displayed in a vbox. On top of that, there is a Button for going to the Calendar page, based on the input, and what does the user want to check:expenses in a certain date, expenses in a certain month or in a period of time. These are all inputed based on dialog boxes with drop down menus and calendar menus.

The FilterController is the controller used for the page in which the expenses are displayed based on the category of expense and in a certain period of time. It gets the information from the Filter Model and just displays it on the interface.

The **CalendarController** uses two more Java Classes: DaysInMonth.java and MonthFXML that are ‘custom’ components made of simple FX components that are then used to make a unitary component. The MonthFXML is the component that is used 12 times in a GridPane to display the months of the year and DaysInMonth is used to display the days of the selected month. So the way the page works: after the user enters filter by day, the calendar fxml is loaded with a grid pane of the months of the year and each month represents an instance of the MonthFXML class. A listener is binded to each of these components and if the user presses on one of the months, the DaysInMonth instance of that month is created with the days of that month, each representing a button that if pressed will show the expenses in that day.

The **BudgetController**, which acts as a combination of the model+controller is used to control the budgeting page. The class has a lot of variables that hold the information from the database about budgeting categorized and remaining money to spend, based on the budget. When wanting to change a budget, either the general or a category one, a dialog box appears with a text input in which the user needs to enter a positive number. If the entered number meets this condition, the database is updated and different labels on the page are further updated. Then, functions that access the database are called to update the information in the database

Classes diagram



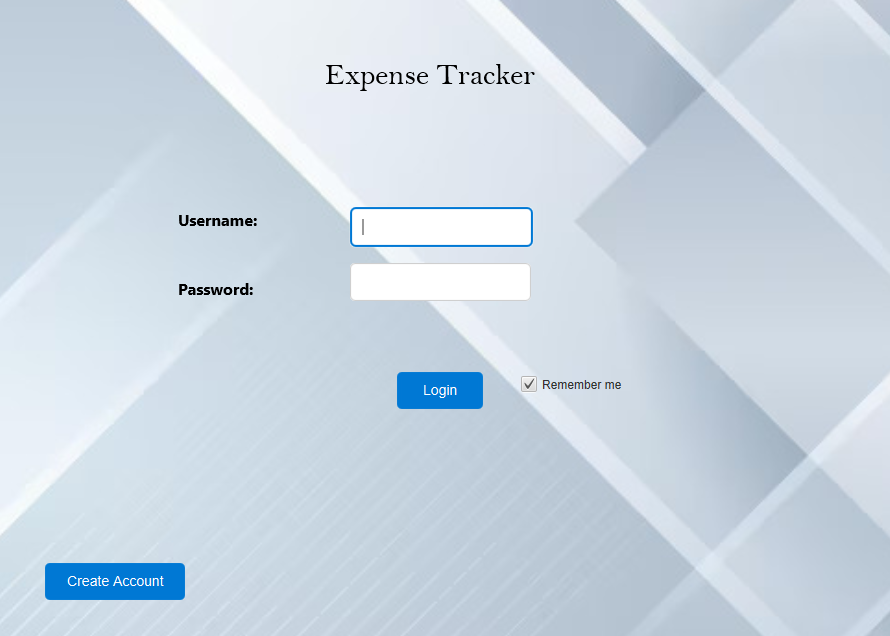
A computer screen shot of a diagram

Description automatically generated

Use cases

* user can create an account and log in
* user can add an expense and set its category, description and cost
* user can set a monthly expenses budget, or a budget by category
* user has access to interactive charts that showcase its expenses
* user can filter by day, month, or a period to see past spendings

User Guide

The first time the user opens the application, they are on the login page, in which they need to input their username and password. 

If they haven’t set up an account, they can opt for pressing the Create Account button to set up one.

After pressing the button, they are on the Create Account page.

On this page, the User must enter a username that’s more than 3 character long and a password that’s more than 5 characters long. To create the account, just press Create Account then just login with the entered credentials.

A screenshot of a login form

Description automatically generated

After successfully logging into the account, the user is on the main page where he has access to multiple features.

A screenshot of a computer

Description automatically generated

Go to History Page

Switch current user

Buttons to check detailed descriptions Go to Budget Page

There are several buttons to go to different pages, or check detailed information about the expenses made, but the main action is the ‘Add expense button, which after pressed, lets the user input information about an expense:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Choose a category Set a price

A screenshot of a computer

Description automatically generated Add a description

A screenshot of a computer

Description automatically generatedIf the User presses ‘Budget’ he is now on the budgeting page.

Set a general Budget

Change categorized

budgets

A screenshot of a computer

Description automatically generatedThe History Page, features expenses done in the last week or last month, based on the pressed button. The user can see the dates, costs and descriptions of the expenses but also the total amount spent in that time.

Money spent on the different categories

From this page, the user can filter by an exact date to check past expenses, via pressing the Calendar button. Then to choose the type of filter

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedIf chosen Filter by month:

Or, if selected Filter by exact date: The user then is on the Page that shows the expenses:

A screenshot of a computer

Description automatically generatedA screenshot of a calendar

Description automatically generated

Here, total spent and categorized spent can be seen.

A screenshot of a calendar

Description automatically generatedIf the user selected Filter by day, the further page appears:

A screenshot of a calendar

Description automatically generatedHere, the user can select a month, then a day in the month to see the expenses:

A screenshot of a computer

Description automatically generated