Josh Mkhari

Student number: 20104681

PATHWAY: PROG6211

Lecturer: Rodney Musinga

19 July 2021

Programming POE

TABLE OF CONTENTS

TABLES. DIAGRAMS AND FIGURES

2. Class Diagram

Figure 1 below is a class diagram of my solution.

Figure 1: Class Diagram of My Program

As displayed by the class diagram the **HomeLoan**, **Rent** and **Vehicle** classes are children of the **Expense** class, as such they inherit all the methods and variables listed in the **Expense** class (Nishadha, 2020). The **MainClass** controls the entire program and incorporates the **Expense** class and its children (Athuraliya, 2020).

3. Program Logic

MVVM (Source)

A **Logic** folder contains all the relevant classes from the prior tasks 1 & 2.

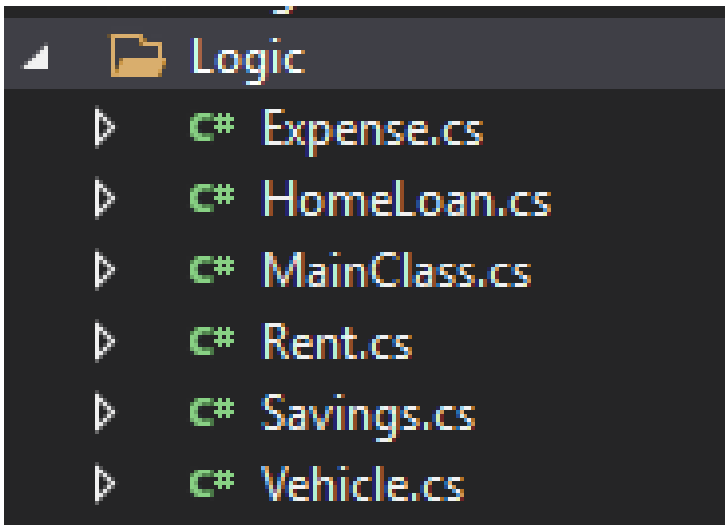


Figure 2: Logic Classes used

Changes made to task 1 and 2 code

Home Loan Method

```
1 reference
public static void HomeLoan(decimal Monthly, decimal tax, decimal[] expenses)//Used to calculate homeloan class
{
    HomeLoan h1 = new HomeLoan();
    if (BudgetPlannerModel.getBudgetPlanner())
    {
        PopulateMonthlyExpenses(h1, Monthly, tax, expenses);//Populates Monthly Gross income, Tax, Grocceries, Water...

    }
    h1.SetLoanExpensesList(propertyValuesArray);//Recieves Property Values Array from HomeLoan xaml
    h1.CalculateExpenses();//Calculates the home loan expenses
    livingType = "Home Loan";//Sets living type
    LivingMonthlyExpense = Convert.ToDouble(Math.Round(h1.MonthlyExpense));//Stores expense
    HomeLoanExpense = "" + Math.Round(h1.MonthlyExpense);
    availableBalance = Convert.ToDouble(Math.Round(h1.AvailBalValue, 2));//Updates avaialable Balance
    userTax = Convert.ToDouble(tax);//Stores user tax
}
```

Figure 3: Home Loan Method

This method is meant to simply populate the **HomeLoan** Class with the relevant values such as the Property Price, Deposit, Interest Rate and Number of Months. This method like the setRent, Vehicles, and Savings Call all the serve the purpose of receiving data and passing them on to the relevant classes for calculations

Vehicles is also meant to do the same as the Home Loan method above it and pass on data to the Vehicle Class

```
1 reference
public static void Vehicles(decimal[] expenses)//Used to calculate Vehicle class
{
    Vehicle v1 = new Vehicle();
    v1.SetUserExpensesList(expenses);//Set expenses such as Purchase price, interest, number of months
    v1.CalculateExpenses();//Calculates the vehicle expenses
    vehicleInsurance = Convert.ToDouble(expenses[3]);// stores vehicle insurance
    VehicleExpense = Convert.ToDouble(v1.TotalExpensesValue);// stores vehicle expense
}
```

Figure 4: Vehicles Method call

Method used to store data into the savings class just as the above Home Loan and Vehicles methods

```
1 reference
public static void SavingsCall(decimal[] expenses)//Used to calculate savings class
{

    Savings s1 = new Savings();
    s1.SetUserExpensesList(expenses);//sets expenses such as goal, interest,  starting balance and years
    s1.CalculateExpenses();//Calculates the savings expenses
    monthlySavings = Convert.ToDouble(Math.Round(s1.MonthlyExpense, 2));// stores savings expense
}
```

Figure 5: Savings Call Method

Method used to store data into the savings class just as the above Home Loan and Vehicles methods

```
1 reference
public static void setRent(double rent)//Used to calculate rent class
{
    livingType = "Rent";//stores the type of living
    rentExpense = rent;//stores the expense
    LivingMonthlyExpense = Math.Round(rent);//used in display
}
```

Figure 6: setRent Method

Method used to store data into the Rent class just as the above Home Loan, Vehicles and Rent methods

```
1 reference
static void PopulateMonthlyExpenses(Expense e1, decimal MonthlyIncone, decimal MonthlyTax, decimal[] expenses)//Used to populate monthly expenses
{

    e1.GrossMonthlyIncomeValue = MonthlyIncone;//Stores the validated gross monthly income.
    e1.MonthlyTaxValue = MonthlyTax;//Stores the validated monthly tax value.
    e1.SetUserExpensesList(expenses);//stores the groceries, water lights, cell phone etc
    ListOfExpenses.Clear();
    ListOfExpenses.AddRange(expenses);//updates the list of expenses
}
```

Figure 6: PopulateMonthlyExpenses Call Method

The Populate Monthly Expenses method serves the purpose of letting the Logic folder classes have access to all the monthly expenses sent in by the user such as Groceries, Water and Lights, Phone Telephone, Other Expenses, Tax and Gross Monthly income.

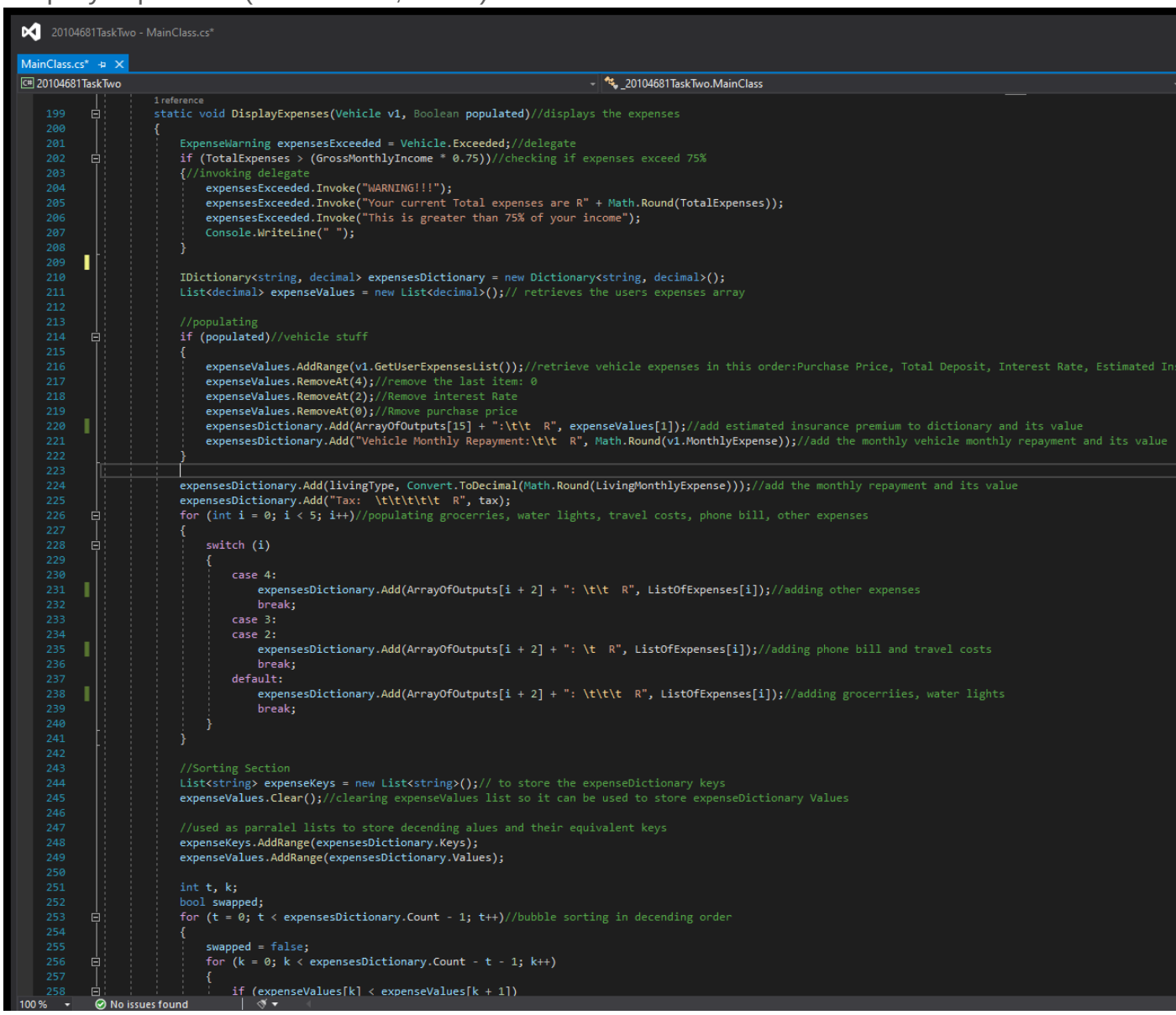
```
19 references
public static bool DecimalCheck(String value)// Checks if user input is a valid Decimal, Parameter takes a location of an output.
{
    /*
     * ListOfOutputs
     * Stores a list of potential outputs for any input the user may enter.
     * If a user is meant to enter travel costs, the relevant output is stored in listOfOutputs[4].
     */
    decimal converted;//stores the user input that has been converted to a decimal.
    bool successfullyParsed;//stores either true or false based on whether or not a conversion of user input was successfull.
    successfullyParsed = decimal.TryParse(value, out converted);//Stores either true or false based on whether or not a conversion of user input was successfull.
    if (!successfullyParsed)//checks if the conversion was unsuccessful or if the user input is empty.
    {
        MessageBox.Show("Incorrect input","Warning", MessageBoxButton.OK, MessageBoxImage.Warning);//warn user.
        MessageBox.Show("Enter numerical values only (example 20,5 or 20)","Warning", MessageBoxButton.OK, MessageBoxImage.Warning);
    }//alert user.

    return successfullyParsed;// return whether the input is
}
```

Figure 7: DecimalCheck Call Method

Unlike the previous tasks **DecimalCheck**, this method returns with slight alterations as now a while loop forcing the user to enter the correct values is no longer necessary but a check each time a user enters a character is more useful.

4. Main Class

DisplayExpenses (w3schools, 2018)



```
1 reference
static void DisplayExpenses(Vehicle v1, Boolean populated)//displays the expenses
{
    ExpenseWarning expensesExceeded = Vehicle.Exceeded;//delegate
    if (TotalExpenses > (GrossMonthlyIncome * 0.75))//checking if expenses exceed 75%
    {//invoking delegate
        expensesExceeded.Invoke("WARNING!!!");
        expensesExceeded.Invoke("Your current Total expenses are R" + Math.Round(TotalExpenses));
        expensesExceeded.Invoke("This is greater than 75% of your income");
        Console.WriteLine(" ");
    }

    IDictionary<string, decimal> expensesDictionary = new Dictionary<string, decimal>();
    List<decimal> expenseValues = new List<decimal>();// retrieves the users expenses array

    //populating
    if (populated)//vehicle stuff
    {
        expenseValues.AddRange(v1.GetUserExpensesList());//retrieve vehicle expenses in this order:Purchase Price, Total Deposit, Interest Rate, Estimated In
        expenseValues.RemoveAt(4);//remove the last item: 0
        expenseValues.RemoveAt(2);//Remove interest Rate
        expenseValues.RemoveAt(0);//Rmove purchase price
        expensesDictionary.Add(ArrayOfOutputs[15] + ":\t\t  R", expenseValues[1]);//add estimated insurance premium to dictionary and its value
        expensesDictionary.Add("Vehicle Monthly Repayment:\t\t  R", Math.Round(v1.MonthlyExpense));//add the monthly vehicle monthly repayment and its value
    }

    expensesDictionary.Add(livingType, Convert.ToDecimal(Math.Round(LivingMonthlyExpense)));//add the monthly repayment and its value
    expensesDictionary.Add("Tax:  \t\t\t\t\t  R", tax);
    for (int i = 0; i < 5; i++)//populating grocerries, water lights, travel costs, phone bill, other expenses
    {
        switch (i)
        {
            case 4:
                expensesDictionary.Add(ArrayOfOutputs[i + 2] + ": \t\t  R", ListOfExpenses[i]);//adding other expenses
                break;
            case 3:
            case 2:
                expensesDictionary.Add(ArrayOfOutputs[i + 2] + ": \t  R", ListOfExpenses[i]);//adding phone bill and travel costs
                break;
            default:
                expensesDictionary.Add(ArrayOfOutputs[i + 2] + ": \t\t\t  R", ListOfExpenses[i]);//adding grocerriies, water lights
                break;
        }
    }

    //Sorting Section
    List<string> expenseKeys = new List<string>();// to store the expenseDictionary keys
    expenseValues.Clear();//clearing expenseValues list so it can be used to store expenseDictionary Values

    //used as parralel lists to store decending alues and their equivalent keys
    expenseKeys.AddRange(expensesDictionary.Keys);
    expenseValues.AddRange(expensesDictionary.Values);

    int t, k;
    bool swapped;
    for (t = 0; t < expensesDictionary.Count - 1; t++)//bubble sorting in decending order
    {
        swapped = false;
        for (k = 0; k < expensesDictionary.Count - t - 1; k++)
        {
            if (expenseValues[k] < expenseValues[k + 1])
```

Figure 8: DisplayExpenses

- This method is used to display all expenses at the end of the program.
- Uses a delegate called **expensesExceeded** to warn the user of their expenses once they surpass 75% (Wagner, 2021)
- Keeps the relevant expenses within a dictionary called **expenseDictionary** storing their names (key) and their values (values) as Key value pairs
- First an **expenseValues** List is created which stores the user expenses
- If the user did choose to buy a vehicle the Expense values list will look like this at first.

ExpenseValues List

Index	Value (decimal)
0	Purchase Price value
1	Total Deposit value
2	Interest Rate value
3	Estimated Insurance Premium value
4	0

Table 3: ExpenseValues if vehicle chosen

- We then remove the values that do not need to be displayed

ExpenseValues update 1

Index	Value (decimal)
0	Total Deposit value
1	Estimated Insurance Premium value

Table 3: ExpenseValues update 1

- We then populate the **expenseDictionary**

Key (string)	Value (decimal)
“Estimated Insurance Premium: R”	Estimated Insurance Premium value
“Vehicle Monthly Repayment: R”	Vehicle Loan Value

Table 4: expenseDcitonary

- The program will add the **livingType** (and monthly repayment value for that living type and the tax the user expects to pay.
    - Living type options:
        - Monthly Home Loan Repayment:
        - Monthly Rent: R

Key (string)	Value (decimal)
“Estimated Insurance Premium: R”	Estimated Insurance Premium value
“Vehicle Monthly Repayment: R”	Vehicle Loan Value
livingType	livingType value
Tax	Tax value

Table 5: expenseDcitonary update 1

- The program then runs a for loop to retrieve the monthly expenses the user provided at the start of the program

Key (string)	Value (decimal)
Estimated Insurance Premium: R	Estimated Insurance Premium value
Vehicle Monthly Repayment: R	Vehicle Loan Value
livingType	livingType value
Tax	Tax value
Groceries: R	Groceries value
Water and lights: R	Water and lights value
Travel: R	Travel value
Other Expenses: R	Other Expenses value

Table 5: expenseDcitonary update 2

- Now that all the expense values and their keys have been stored the program will then store the keys and the values in 2 separate lists.

Index	Value (decimal)
0	Estimated Insurance Premium value
1	Vehicle Loan Value
2	livingType value
3	Tax value
4	Groceries value
5	Water and lights value
6	Travel value
7	Other Expenses value

Table 6: ExpenseValues update 3

Index	Value (String)
0	Estimated Insurance Premium: R
1	Vehicle Monthly Repayment: R
2	livingType
3	tax
4	Groceries: R
5	Water and lights: R
6	Travel: R
7	Other Expenses: R

Table 7: ExpenseKeys

- Now using a bubble sort (sparknotes, n.d.) on the **ExpenseValues** list the program will sort the **ExpenseValues** list in descending order and perform the same changes in the **ExpenseKeys** list which will result in the parallel lists both being sorted.
- Once the sort is complete the data is displayed.

5. Vehicle class

Delegate update.

```
1 reference
public static void Exceeded(string message)//delegate call
{
    MessageBox.Show(message,"Warning Vehicle Exceeded", MessageBoxButton.OK, MessageBoxImage.Warning);
}
```

Figure 9: Exceeded Delegate call

As the structure of my Delegate allowed me to change the Console.WriteLine to a MessageBox.Show, the Delegate still works the same was as it did in task 2.

6. Savings class

```
2 references
class Savings : Expense
{
    private List<decimal> savingsValueList = new List<decimal>();
    2 references
    public override void CalculateExpenses()//Used to calculate the total expenses and calculate the available balance.
    {
        //goal,years,initialDeposit,interestrate
        this.savingsValueList.AddRange(this.GetUserExpensesList()); //Populate savings list

        double goal = Decimal.ToDouble(this.savingsValueList[0]);// populate goal
        double initial = Decimal.ToDouble(this.savingsValueList[2]);//populate initial starting amount
        double interrestRate = 1 + Decimal.ToDouble(this.savingsValueList[3] / 100 / 12);//set interest rate
        double years = Decimal.ToDouble(this.savingsValueList[1]) * 12;//set years
        double interestRatePow = Math.Pow(interrestRate, years);//set interest rate to the power of years
        double variableA = (interrestRate - 1) * (goal - initial * (interestRatePow)); // Line A (Explained in Word Doc)
        double variableB = interestRatePow - 1;//Line B (Explained in word Doc)..
        this.MonthlyExpense = Convert.ToDecimal(variableA / variableB);// Final answer
    }..
}
```

Figure 10: CalculateExpenses Savings (WealthMeta, n.d.)

$$A = \frac{r}{12} \times (Goal - Principal\left(1 + \frac{r}{12}\right)^{Y \times 12})$$

$$B = (\left(1 + \frac{r}{12}\right)^{Y \times 12} - 1)$$

$$Monthly\ Savings\ Goal = \frac{A}{B}$$

7. Views Structure

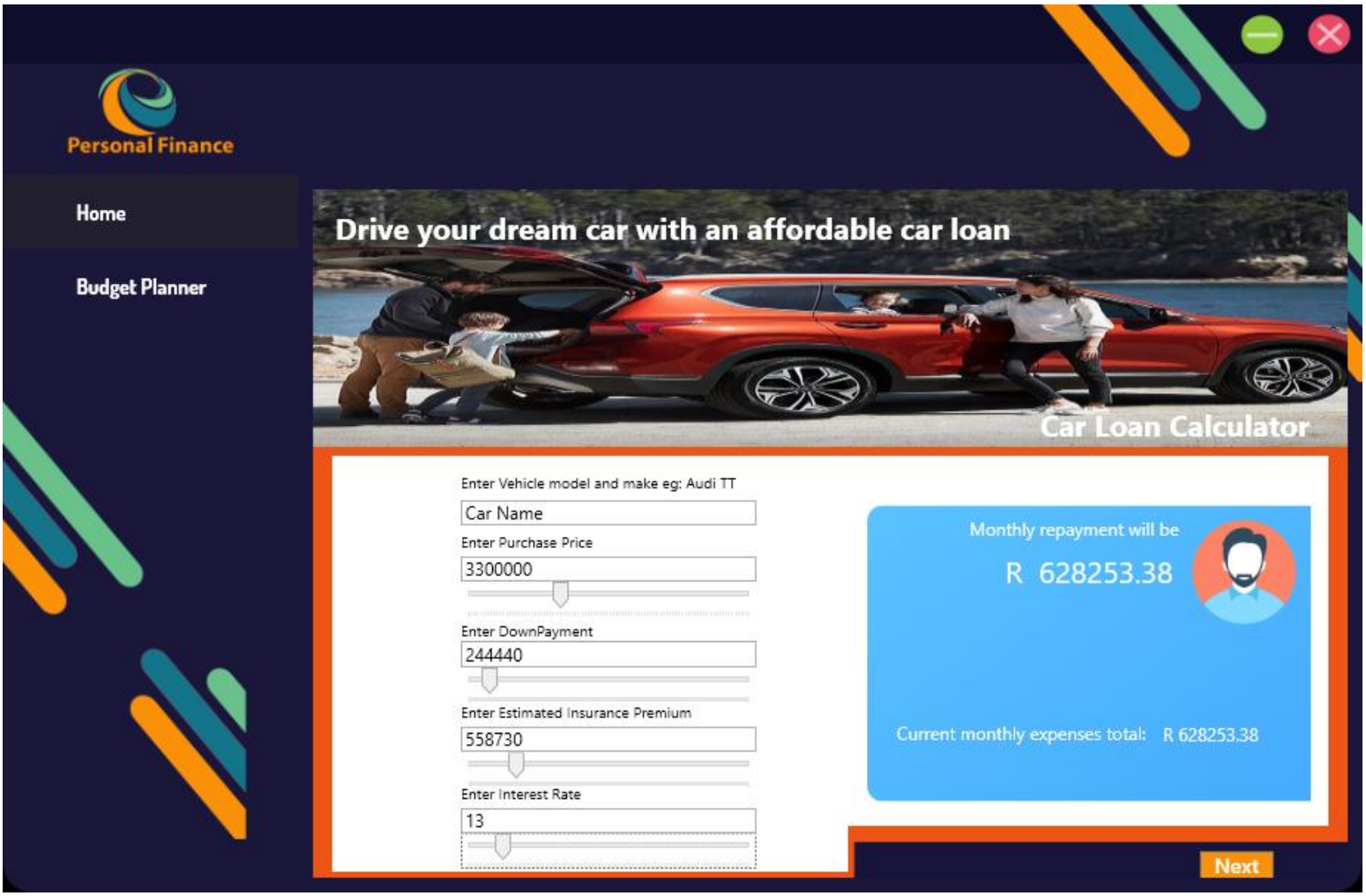


Figure 11: Vehicle View Calculator

Input views use a combination of sliders (source) and text boxes (source) to input data. The Monthly repayment and current monthly expense are auto updated based on what the user inputs.

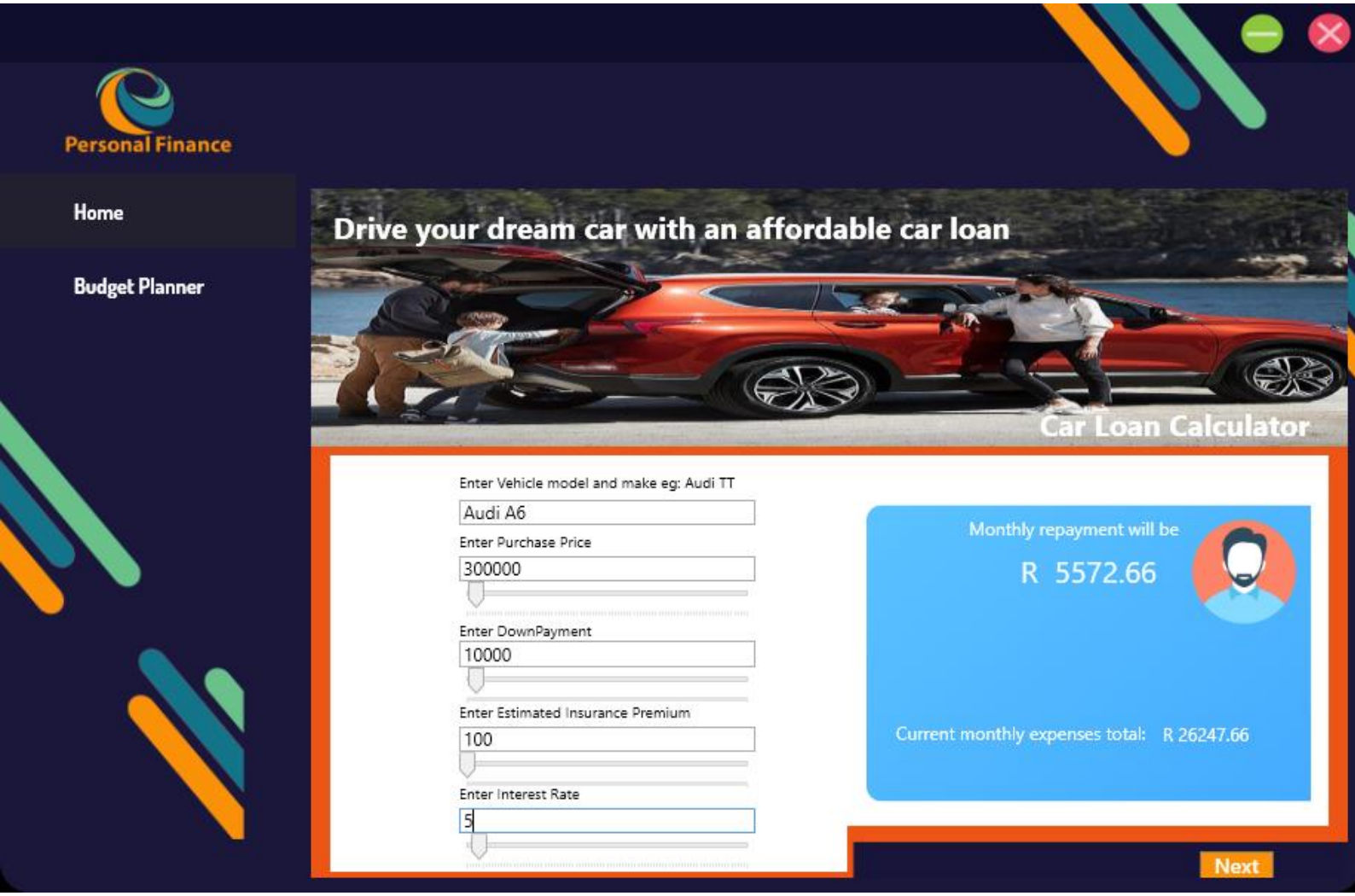


Figure 12: Vehicle View Budget Planner

If the user is in budget planner mode the current monthly expenses total will represent the then total based on data from previous views.

View Xaml

```
0 references
public VehicleView()
{
    InitializeComponent();
    this.DataContext = new VehicleViewModel();
    if (BudgetPlannerModel.getBudgetPlanner())
    {
        currentTotal.Value = MonthlyExpenseModel.getCurrentExpenses();
        updater.Value = 4;//"ChooseSavings or not"
    }
    else
    {
        updater.Value = 1;
    }
}
```

Figure 13: Vehicle View Constructor

Upon initialization the Data context is set to the **VehicleViewModel** which has all the relevant commands this view employs.

In **BudgetPlanner** Mode, the views are linked to present in a specific order, as such if the user has chosen a budget plan the **updater** must change as it is responsible for how the program changes between views.

8. View Model Structure

```
5 references
public class VehicleViewModel : BaseViewModel
{
    1 reference
    public ICommand UpdateViewCommand { get; set; }
    2 references
    public VehicleViewModel()
    {
        UpdateViewCommand = new UpdateViewCommand(this);
    }

}
```

Figure 14: VehicleViewModel

The **updateViewCommand** is used to switch from any view to another based on the command parameter being passed in. This line of code exists on every single view throughout the program.

9. Conclusion

It may never be possible to have a fully secure system that is immune to any form of attack. As long as a system needs to be logged into a backdoor, phishing technique or even malware can be invented to combat its defences. It is in every cloud analysts' best interest to safeguard their systems to a standard so high it automatically discourages bad actors from attempting to hack their systems.

REFERENCE LIST

Athuraliya, A., 2020. *The Easy Guide to UML Class Diagrams | Class Diagram Tutorial.* [Online]
Available at: https://creately.com/blog/diagrams/class-diagram-tutorial/
[Accessed 17 April 2021].

Investopedia, n.d. *Auto Loan Calculator.* [Online]
Available at: https://www.investopedia.com/car-loan-calculator-5084761
[Accessed 4 June 2021].

Liam, 2018. *Difference between decimal, float and double in .NET?.* [Online]
Available at: https://stackoverflow.com/questions/618535/difference-between-decimal-float-and-double-in-net#:~:text=(%20float%20is%20short%20for%20%22floating,point%20something%20on%20the%20end.)&text=float%20is%20a%2032%2Dbit,to%20get%20at%20the%20code
[Accessed 13 April 2021].

Nedbank, 2021. *Loan Repayment Calculator.* [Online]
Available at: https://www.nedbank.co.za/content/nedbank/desktop/gt/en/personal/tools-and-guidance/calculators/loan-repayment-calculator.html
[Accessed 15 April 2021].

Nishadha, 2020. *UML Class Diagram Relationships Explained with Examples.* [Online]
Available at: https://creately.com/blog/diagrams/class-diagram-relationships/
[Accessed 17 April 2021].

sparknotes, n.d. *The Bubble Sort Algorithm.* [Online]
Available at: https://www.sparknotes.com/cs/sorting/bubble/section1/#:~:text=The%20total%20number%20of%20comparisons,since%20no%20swaps%20were%20made.
[Accessed 2 June 2021].

w3schools, 2018. *C# Exceptions - Try..Catch.* [Online]
Available at: https://www.w3schools.com/cs/cs_exceptions.asp
[Accessed 14 April 2021].

Wagner, B., 2021. *Delegates (C# Programming Guide).* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/#:~:text=A%20delegate%20is%20a%20type,method%20through%20the%20delegate%20instance.
[Accessed 2 June 2021].