To run this application, open the .html file in the ‘Loan Calculator’ folder in a browser of your choice.

**Assignment 2: Report**

In this report, I will be discussing my application code design, the assumptions I made, and an explanation of the source code files I created.

**User Guide**

When a user first opens the calculator, they will see the image below, where they will need the enter all the correct information.

Graphical user interface, application

Description automatically generated

If the user has entered the correct information and they press the calculation button, they will be shown a table of their payment plan, shown in the image below. If the user presses the back button, they will be redirected to the pre-existing form.

Graphical user interface

Description automatically generated with medium confidence

**Code Walkthrough**

For my code, everything is contained in the loan.js file. The action of showing the table is a simple toggle method which is called ‘showHide()’ and shown in the below image, which is called when either the ‘calculate’ or ‘back’ buttons are pressed. This way I keep all my code in the same file to simplify things.

**A screenshot of a computer

Description automatically generated with medium confidence**

When a user first hits the calculate button, it will trigger the ‘validateForm()’ method, which will check if the user has entered the correct information such as it being a number, not being blank etc, if the validation checks out, the ‘calculate()’ method will be called.

The calculate method is the main method of the web app and ties everything together. Firstly, the clearTable() method is called to remove any pre-existing table that the user may have created and stops the new table from appending to the previous table, then the showHide() method is called.

Text

Description automatically generated

Then, the ‘getRows()’ method is called to calculate the number of months the user will have to pay back for, depending on the dropdown list value. If the value is months it will use the number the user has entered, else it will use times the number by 12, as this will convert the number of years the user has entered into months. This variable is used to calculate how many rows need adding to the table.

The totalsRows, interestRate and paymentAmount variables are calculated outside the loop as these are fixed. Inside the for loop, the formulas for the different payments for each specific row are calculated as these are dynamic and will need to be calculated again every loop.

After these are calculated, they are then correctly formatted by adding the pound sign, adding commas in the correct places to improved readability and correctly rounding the numbers to two decimal places. These values are then added to the table with the ‘addRow()’ method.

**Code Design**

For my code design, I decided to keep the form and table on the same page as this would allow me to keep all the code on the same page as well as the fact that there is not enough code written that would warrant the decision to split up the form and table into two pages.

I have also validated to ensure that everything runs smoothly and reduces the chance of the application crashing. I have also followed certain programming principles such as DRY by using for loops. I have also made sure to create different methods to contain certain sections of code that have a specific purpose, such as adding rows.

**Assumptions**

The main assumption I made was that interest will be added monthly rather than yearly because, from most sources I have seen, payments occur monthly. I also made some assumptions about what a user would enter for the form. For example, ensuring that the annual interest rate is no more than one hundred and the user could only enter a whole number for the loan period.