Fin-tastic! Calculator

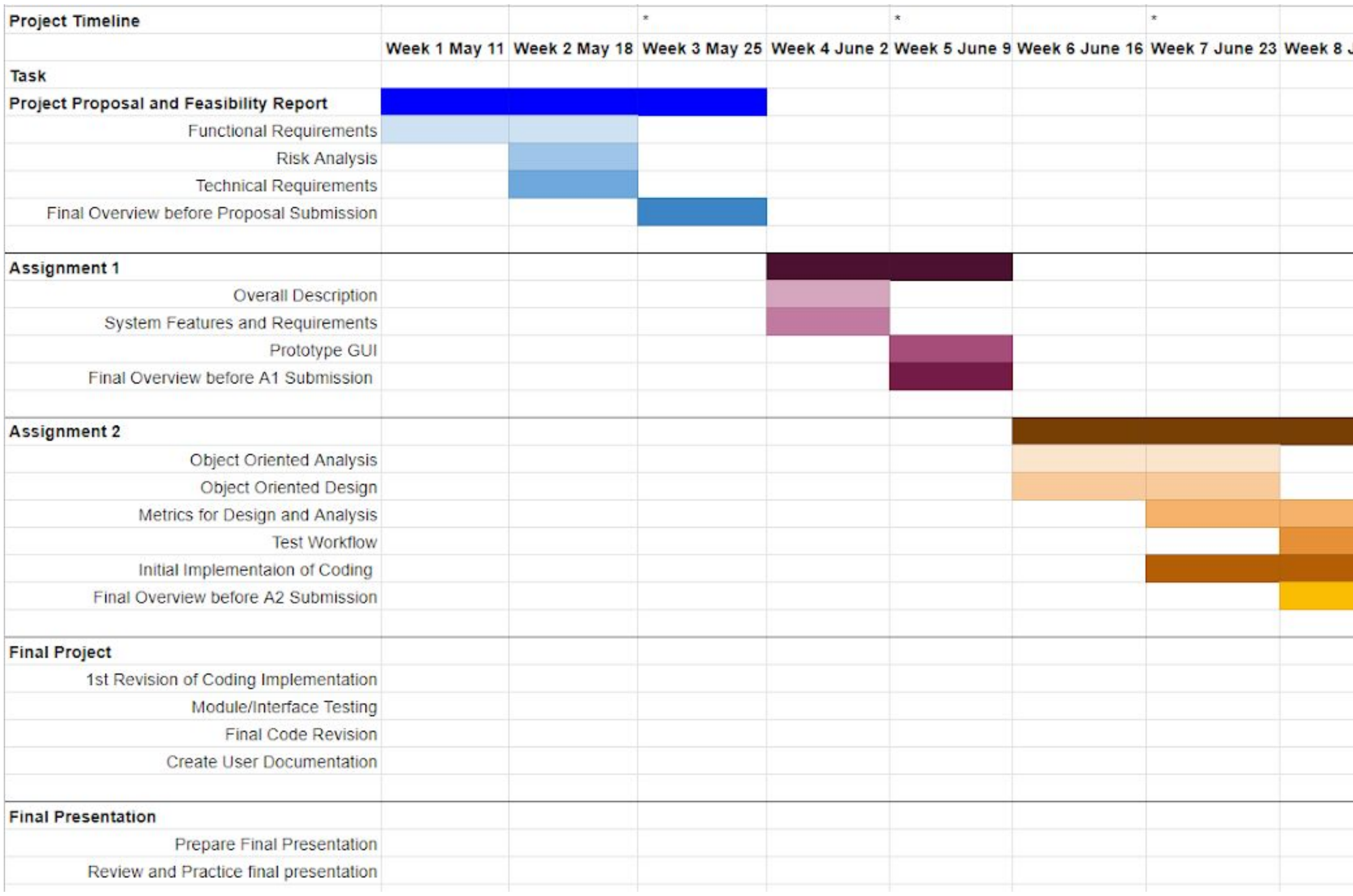
Final Submission

August 5, 2020

Alexander Ojo, ojox0510@mylaurier.ca (170750510)
Ashley Antony, anto3300@mylaurier.ca (170473300)
Muhammad Mustafa Mohsin, mohs1340@mylaurier.ca (170451340)
Sobihan Mamugan, mamu0640@mylaurier.ca (170200640)
Guillhem Paillet, pail6700@mylaurier.ca (170456700)

Contents

Project Timeline



Project Timeline			*		*		*	
	Week 1 May 11	Week 2 May 18	Week 3 May 25	Week 4 June 2	Week 5 June 9	Week 6 June 16	Week 7 June 23	Week 8 J
Task								
Project Proposal and Feasibility Report	🟦	🟦	🟦					
Functional Requirements	🟦	🟦						
Risk Analysis		🟦						
Technical Requirements		🟦						
Final Overview before Proposal Submission			🟦					
Assignment 1				🟪	🟪			
Overall Description				🟪				
System Features and Requirements				🟪				
Prototype GUI					🟪			
Final Overview before A1 Submission					🟪			
Assignment 2						🟫	🟫	🟫
Object Oriented Analysis						🟧	🟧	
Object Oriented Design						🟧	🟧	
Metrics for Design and Analysis							🟧	🟧
Test Workflow								🟧
Initial Implementaion of Coding							🟧	🟧
Final Overview before A2 Submission								🟨
Final Project								
1st Revision of Coding Implementation								
Module/Interface Testing								
Final Code Revision								
Create User Documentation								
Final Presentation								
Prepare Final Presentation								
Review and Practice final presentation								

Fin-Tastic! Calculator Weekly Schedule

*Please note that each week we had a meeting after Tuesday classes unless otherwise specified in order to keep track of progress and continue forward.

Week 1 Tasks:
1. Form group
2. Decide on project idea during group meeting
3. Start working on Functional Requirements

Week 1 Deviations:
- Guilhem joined at a later date

Week 2 Tasks:
1. Risk Analysis
2. Technical Requirements
3. Finish Functional Requirements

Week 3 Tasks:
1. Meeting prior to submission to check everyone's work
2. Final submission and editing of Project Proposal

Week 4 Tasks:
1. Start Assignment 1
2. Group meeting to assign roles
3. Define the system feature and requirements
4. Come up with a description of the product

Week 4 Deviations:
- Had to meet twice in order to better figure out our approach to designing the product.

Week 5 Tasks:
1. Work on prototype GUI
2. Finish off assignment 1 submission

Week 6 Tasks:
1. Start assignment 2
2. Begin creating object oriented analysis
3. Begin creating and agreeing on the object oriented design

Week 7 Tasks:
1. Finalize object oriented analysis
2. Finalize object oriented design
3. Begin working on metrics for design and analysis submission
4. Begin the initial coding implementation

Week 7 Deviations

- The analysis and design took a longer time to complete than expected. The coding implementation was not started on time this week

Week 8 Tasks:

1. Finalize metrics for design and analysis
2. Make test workflow analysis
3. Complete initial coding implementation
4. Meet to finalize A2 Submission

Week 8 Deviations:

- The meeting to finalize the A2 submission was moved in order to accommodate schedules of group members

Week 9 Tasks:

1. Review coding implementation together with group
2. Begin module and interface testing after revision

Week 9 Deviations:

- Module and interface testing were only started at the end of the week rather than the start

Week 10 Tasks:

1. Finish module and interface testing
2. Finalize code for production

Week 10 Deviations:

- Final code implementation was not finalized this week

Week 11 Tasks:

1. Create user documentation

Week 11 Deviations:

- Code implementation from week 10 finished
- User documentation was moved back one week to finalize code implementation

Week 12 Tasks

1. Prepare slide deck for final presentation
2. Practice for final presentation with group

Week 12 Deviations:

- User documentation from week 11 completed in week 12

Week 13 Tasks:

1. Final Presentation with group

Feasibility Report/Proposal Contents

I. Executive Summary

Fin-tastic! Calculator is designed for the usage of the Laurier Students Tutoring Society (LSTS). The LSTS will be represented by the lead finance tutor, Bobby Axelrod. Mr. Axelrod currently tutors many finance students. Oftentimes, students have difficulty understanding the mathematical formulas because they lack access to simplified calculations. Thus, Mr. Axelrod came to our group of developers and proposed that we design a financial calculator that would be made available for use for any student being tutored by LSTS.The main focus of our development team is to design a web application that contains all the functionality of a financial calculator, for example time value of money and mortgage payments. Regular communication and meetings will be had in order to ensure that the design and implementation is successful and completed by the given deadline of August 4. A successful final product will provide accurate and quick answers for all functions of a financial calculator.

II. Preliminary Requirements Analysis

Application Overview

Objectives

The basic functionality of the application will be to allow users to use all the functions of a regular scientific calculator as well as time value of money calculations for finance.

Problem Identification, Definition and Scope

For students taking finance courses they are required to spend lots of time doing practice problems in order to understand the subject. The problem that is currently being faced is that the time spent using a regular calculator is extremely high with a large margin for human error while inputting calculations.

One workaround for this problem is a financial calculator, however the cheapest recommended one costs around $40, which is an expense that many students have difficulty affording. This project aims to simplify financial calculations and reduce costs for Laurier business students by giving them access to a web application that has the functionality of a financial calculator. It will allow for students taking finance courses to have easy access to a financial calculator while studying to optimize time usage.

The scope of our application includes functionality of a scientific and financial calculator. The financial calculations will include the capability to calculate Time Value of Money problems, such as basic/compound interest, amortization, loan payments, annuities as well as present and future values.

User Roles and responsibilities

Users' (Student) role: The ability to perform regular and financial calculations.

Interactions with Other systems

The product will be built using node.js and other common web packages. The calculation solutions produced by the product may be used to combine with other common systems used by Laurier students in the future.

Production rollout considerations

Since we decided to use an Agile working system, we will divide and conquer tasks in terms of sprints. After each sprint the code, data and functions will be tested in a sandbox to further test the work. Once tested, it will be pushed to Github and the production rollout or "Deployment" will always occur after the testing in the sandbox. The final product will then be hosted using AWS.

Functional Requirements

Statement of functionality

This Web application will have an interactive User Interface where the user will be able to input values by using their keyboard or the GUI to calculate loans, and other financial calculations. (Refer to "Problem Identification, Definition and Scope".)

The application will need to have memory to store and display recent calculations to provide a holistic overview of the calculations. However, once the browser is closed the memory will be lost.

It will use asynchronous features of JavaScript to ensure no breaks/pauses in the application and ensure calculation time efficiency.

The final product will need to be accessible and tested with different browsers. We are unsure that all users use common browsers (ex. Chrome), therefore, fallbacks will need to be implemented to ensure each user has the desired outcome. (Refer to Risk Analysis).

Non-functional requirements

The application should run on Google Chrome, Safari as well as Mozilla Firefox as they are the most common web browsers in use. The application must be online at all times so users can access it during late night study sessions.

Optional features

Users should be able to change the appearance of the calculator, such as background themes, fonts and colours.

Usability

The program must be able to complete calculations in less than 0.5 seconds to emulate the performance and capability of a physical calculator. Users will be able to complete calculation via mouse clicks or keyboard inputs.

III. Process to be Followed

For this project, the team decided to follow an Agile software development approach that involves breaking down the requirements into the functionalities that can be accomplished in biweekly sprints. The team chose this method as it enables consistent two week software development which the client can provide feedback on. With short timelines and constant feedback, the team can deploy software to production that they know the client finds valuable. Documentation and code testing will be completed on a regular basis throughout all project stages. The team will focus first on the features that offer the most value to the client.

Process Outline
User testing throughout: The project will be presented to the client for client testing and suggestions after each sprint along with a status report. This is advantageous as it allows that client to have a more accurate understanding of the individual features of the product and a more accurate timeline of product completion.

1st iteration (May 29, 2020)
Project Proposal and Feasibility Report
The team created a report that outlines the problem that the project solves, the project solution, and project timeline. The client can use the report to verify the program requirements and understand when they can expect the project to be completed.

2nd iteration (July 1, 2020)
Product Mockup
User interface will be created and common calculator functions will be implemented to ensure that it meets the clients requirements and expectations. It will include the data entry field (calculations input bar) and the user input fields (number buttons). The product mockup will not contain all the functionality of the final product (ex. time value of money functions and annuity calculation functions) but it will be used to show proof of concept.

3rd iteration (July 25, 2020)
Final Testing Period and Documentation
All the product's functional requirements and required features will be met. Testing will be conducted to ensure that all intended use cases can be completed. Documentation will be finalized with the purpose of explaining product use cases and code design. Code design is included so that clients can make improvements to the product after product release.

Client Presentation
The team will put together a 3 minute product demonstration and offer a 30 minute meeting to train the client on how to use the product and to discuss additional questions that the client may have about the product. The complete documentation will be provided ahead of the demonstration.

IV. Suggested Deliverables

The following set of work-products will be delivered to the client to satisfy the client's need:

(A) Status Reports:
Throughout the development process, periodic reports will be written and presented to the client to ensure the team doesn't deviate from the client's needs. This report will detail the requirements, functions, design and ultimately the final form of the project. These reports will allow the client to comment and respond to the progress being made. Using these comments we will be able to continue to progress to the client's needs.

(B) Demonstration:
Once the application is complete a final demonstration will take place. Here we will present the application to our client and answer any final questions the client might have.

(C) Documentation:

We will be sending out documentations that will describe in detail how the program will run in detail along with the design. Along with this, the code will be given and the client will be able to give feedback on anything that they believe should be changed or updated. By doing so we will be able to ensure the client's needs are met.

V. Visibility/Business Considerations

During the course of this project, the team will maximize the visibility of the system and development process. This will ensure that we get all the specifications that our client needs. If we happen to deviate from these specifications during development, we will be able to correct it through client feedback. Our main method of communication with our client is going to be through our bi-weekly meetings. During these meetings, the team will assure all members are caught up and understand their jobs. On top of this, our clients will be able to sit in and get updated on our progress. We will have a two way discussion that will include feedback from the client so that we meet the standards of the client.

VI. Risk Analysis

Time Risks

This project is required as part of a course, as such, we must be able to produce a working product before the deadline of August 4, which is the end of the academic semester. This introduces the risk that the product will not be complete, especially if multiple revisions are necessary.

Functionality Risks

These risks have to do with the implementation of the system. Examples of potential issues are incorrect formulas for the time value of money calculations, a buggy or non user-friendly GUI. These problems are the easiest to fix, as they can be mitigated through testing and feedback from potential users throughout the development process.

Compatibility Risks

In our Web application, compatibility risks are defined as the ability of each browser to support and process our code. Compatibility issues can occur when the browser does not support the features or details supported in others. An example could be the CSS font families; the Chrome browser supports more fonts than Firefox therefore if we use a font supported by Chrome and not by Firefox, compatibility issues will arise.

This is a common issue amongst developers. To counter this we need to have fallbacks within our application. A fallback is designed so that if a browser does not support a function/feature, in this case a font, there is an alternative option that steps up to take over that function/feature. In our font case, we will have fallback fonts so if a font is not supported in Firefox, another font which is supported will take its place.

Risk Management

Here we will apply Agile methodology to minimize any risks. Agile methodology is widely and commonly used by Major organizations to help with risk management. It enforces a divided workload in Sprints (a time-period to work on and complete given tasks). Moving on to Stand-ups within Sprints (meetings to update everyone on the team regularly). Followed by a Retrospective (a meeting to establish what went well and what can be improved in the new Sprint). When these aspects of Agile methodology are put together, it leads to an increase in motivation, communication and the ability to oversee a risk.

To further minimize risks, we will ask the client to join our stand-ups for direct and precise updates. This will also give the client a chance to voice his options to make sure the project is according to their standards.

The other main approach we will have is frequent communication between the internal team as well as client and users in order to obtain quality feedback on the applications functionality and interface. Regular meetings will be held in accordance with the agile workflow principle to ensure collaboration is efficient and progress is being made.

As this application is meant to be used by anyone, anywhere, there will be no restriction on access to the application. Any user will be able to access the application when necessary. The main security that we will have in order to protect the user’s information is that the web application will be hosted on an HTTPS site to ensure safety of user’s calculation requests.

VII. Technical Requirements

Here we will establish the requirements expected for our application, leading to the successful deployment of the application.

(A) Working environment.

- In this case, we will be using JavaScript as it is the most advanced, updated and well known front-end language.

(B) User interface.

- The user interface needs to be appealing, eye-catching and understandable. It should have instructional notes to provide further clarity.

(C) Version Control

- We will use Git version control to keep providing updates that are organized and documented.

(D) Ability to run in the background

- In order for the web app to compute multiple equations, we need to implement Asynchronous JavaScript to ensure that functions will run in the background and not crash while following multiple orders.

VIII. Conclusion

Based on the functional and technical requirements outlined in this feasibility report, the team is confident that they fulfill the client's requirements within one semester. The team members possess a solid understanding of the financial use cases of the system and the softwares required to complete this assignment. In conclusion, the project is feasible and will continue to prepare the product for the client.

Assignment 1 Contents

Introduction

Purpose

The basic functionality of the application will allow users to use all the functions of a regular calculator as well as time value of money calculations for finance.

Intended Audience

Our problem identification derives from students taking finance courses. These finance courses require students to spend lots of time doing practice problems in order to understand the subject. The problem that we seek to provide a cost efficient solution that reduces the large amount of time spent using a common arithmetic calculator and the large margin for human error while inputting calculations. This project aims to simplify financial calculations while reducing calculation time, eliminating financial costs and mitigating risk of human error for Laurier business students.

1.3 Intended Use

This project accomplishes that by giving students access to a web application that has the functionality of a financial calculator. It will enable finance students to study with a financial calculator thus optimizing their time usage without the financial costs of purchasing one. The financial calculations will include the capability to calculate Time Value of Money problems, such as basic/compound interest, amortization, loan payments, annuities as well as present and future values.

1.4 Scope

The scope of our application includes functionality of a financial calculator. Fin-tastic! Calculator will be able to complete both arithmetic calculations and financial calculations.

1.5 Definitions and Acronyms

This section will overview some of the acronyms used in financial calculations:

- TVM = Time Value of Money
- N = number of periods
- I/Y = interest rate
- FV = future value
- PV = present value
- PMT = periodic payment
- P/Y = periods per year
- C/Y = number of interest compounding periods per year
- AMORT = Amortization
- CPT = Compute TVM value

Overall Description

2.1 User Needs

The user should be able to view the previous calculations that he/she entered as long as the browser remains open. Once the browser is closed all the memory will be lost. The user should be able to access the calculator from common web browsers (Chrome, Safari, and Firefox). The application should be easy to navigate and utilize.

2.2 Assumptions and Dependencies

The assumption of the user is that it is a finance student, thus they would have background knowledge on the financial functions.

System Features and Requirements

Functional Requirements

Individuals will be able to calculate various financial calculations. They will have the option to use the Calculator and its buttons or input the values themselves within a text box. Asynchronous coding will be implemented to ensure the system runs smoothly. Examples of these features include; Mortgage calculations, present value/future value calculations, bonds and stock calculations, and Lease calculations.

To ensure that the user doesn't experience any pauses or faults, the application will use asynchronous features of JavaScript. Also through these features we will be able to ensure time efficiency on the calculations. We want to ensure that the user is getting the experience of a physical calculator.

In order to ensure a consistent user experience, the user should expect the same experience and functionality across various browsers including Chrome, Safari and Firefox. As mentioned in the initial project proposal, there are risks associated with using multiple platforms. The main risk being that specific features or processes are not supported by the chosen browser(s). The requirement for this product will be to eliminate all inconsistencies across the chosen platforms and guarantee a reliable experience.

3.1.1 Use Cases & Descriptions

1 - Basic calculation

The basic calculations use case will allow users to perform calculations akin to a regular calculator, such as addition, subtraction, multiplication and division.

2 - Basic financial calculations (loan or annuity payments, future value, present value, etc)

The basic financial calculation use case will allow for users to solve basic TVM calculations. They will input 4 out of 5 of the following: N, PMT, FV, PV or I/Y and then be able to compute the missing value. This assumes that the I/Y and C/Y are set to default and no other TVM functions are changed.

An example of this is calculating the future value of an investment, given that the other 4 values are given(N, PMT, PV, I/Y).

3 - Advanced financial calculations

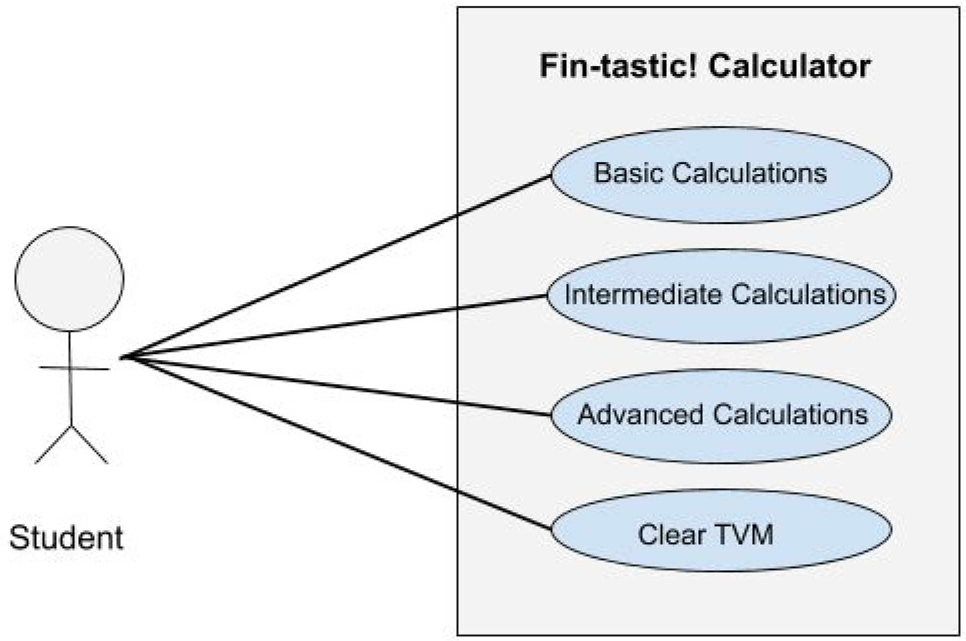
Advanced calculations will be able to calculate variables that are unknown and other from the basic calculations. Finding the mortgage rate given the payments or finding the present value of an annuity given multiple payments per year and non annual compounding per year is an example of advanced financial calculations.

Furthermore, a student who wants to calculate their mortgage payment. AMORT is an advanced variable that will be given by the student for this case. The student enters FV, N, PV and I/Y in order to calculate the payment amount by pressing CPT + PMT.

4 - Clear TVM

The Clear TVM use case will allow a user to clear/delete the currently saved TVM calculation values. This will allow them to calculate new values successfully.

3.1.2 Use Case Diagram



3.2 External Interface Requirements
Here we will establish the requirements expected for our application, leading to the successful deployment of the application.

A. Working environment.
   - In this case, we will be using JavaScript as it's the most advanced, updated and well known front-end language for web development

B. User Interface.
   - The user interface needs to be appealing, eye-catching and understandable. It should have instructional notes to provide further clarity.

C. Version Control
   - We will use Git version control to keep providing updates that are organized and documented.

D. Ability to run in the background
   - In order for the web app to compute multiple equations, we need to implement asynchronous JavaScript to ensure that functions will run in the background and not crash while following multiple orders.

3.3 System Features
In order for this application to be user friendly, the application should accept two forms of input. The first being a standard keyboard input allowing the user to type both

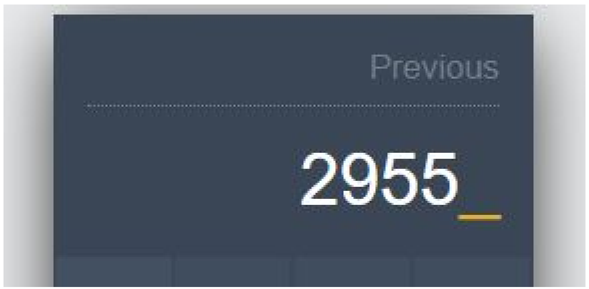
numbers and standard operating symbols (e.g. * / - + etc.). However, for more complex operations the user will have to rely on the GUI to specify the financial or scientific function. Some examples of the functions include: PMT values, Loans, PV, BOND, AMRT, etc. The user can then hit enter on their keyboard or “=” on the calculator to get a final result. There will also be user navigation to edit the input equation within the display using arrows on the keyboard or the calculator.

3.4 Nonfunctional Requirements

The application should run on Google Chrome, Safari as well as Mozilla Firefox as they are the most common web browsers in use. The application must be online at all times so users can access it during late night study sessions.

Prototype GUI

Here you can view a rough visual representation (GUI) of the final product.



Assignment 2 Contents

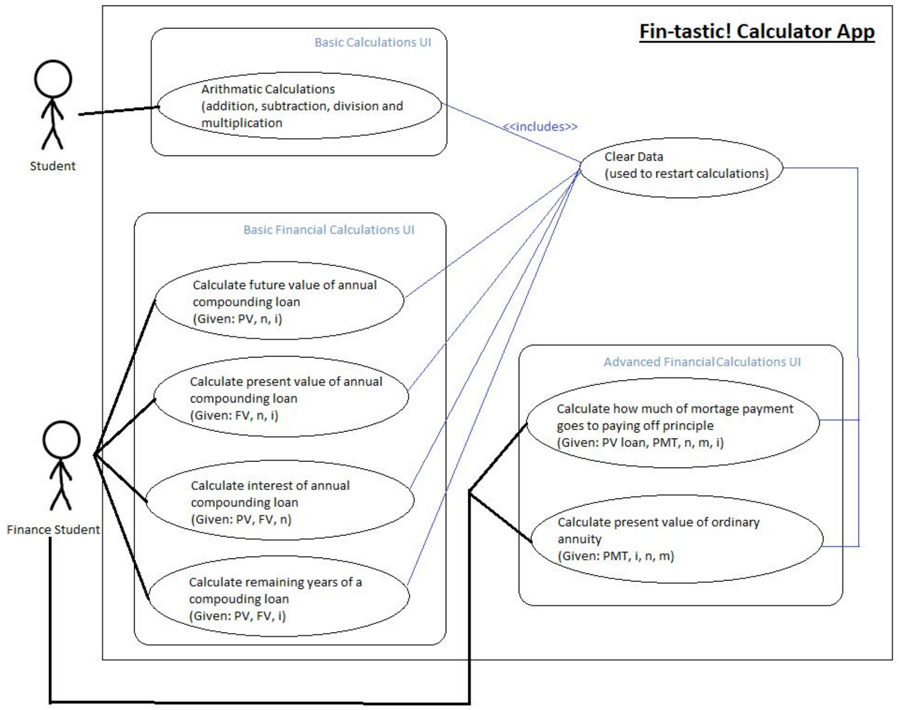
Object-Oriented Analysis

Use-case models

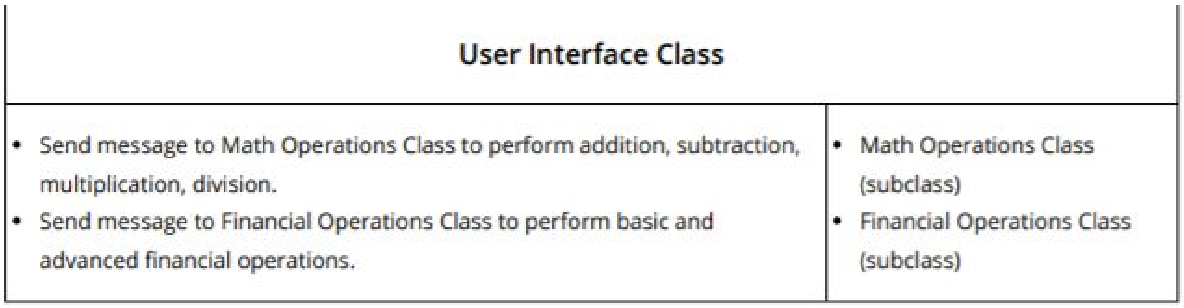
Each user interface (UI) has different buttons on the calculator. This simplifies the UI for the various users as only the buttons required for the user desired results are available. Below is the model of the various use cases of the Fin-tastic! Calculator separated into its different UIs.

Legend:
PV = present value of loan, n = number of years of loan, m = number of payments per year,

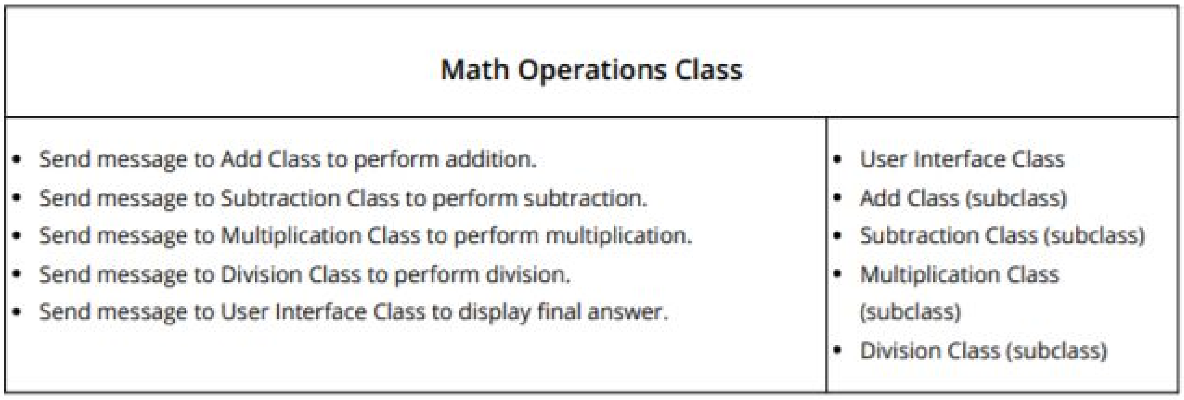
i = interest rate, FV = future value of loan, PMT = periodic payment amount


Fin-tastic! Calculator App
Student
Basic Calculations UI
Arithmatic Calculations (addition, subtraction, division and multiplication
<<includes>>
Clear Data (used to restart calculations)
Basic Financial Calculations UI
Calculate future value of annual compounding loan (Given: PV, n, i)
Calculate present value of annual compounding loan (Given: FV, n, i)
Calculate interest of annual compounding loan (Given: PV, FV, n)
Calculate remaining years of a compouding loan (Given: PV, FV, i)
Finance Student
Advanced Financial Calculations UI
Calculate how much of mortage payment goes to paying off principle (Given: PV loan, PMT, n, m, i)
Calculate present value of ordinary annuity (Given: PMT, i, n, m)

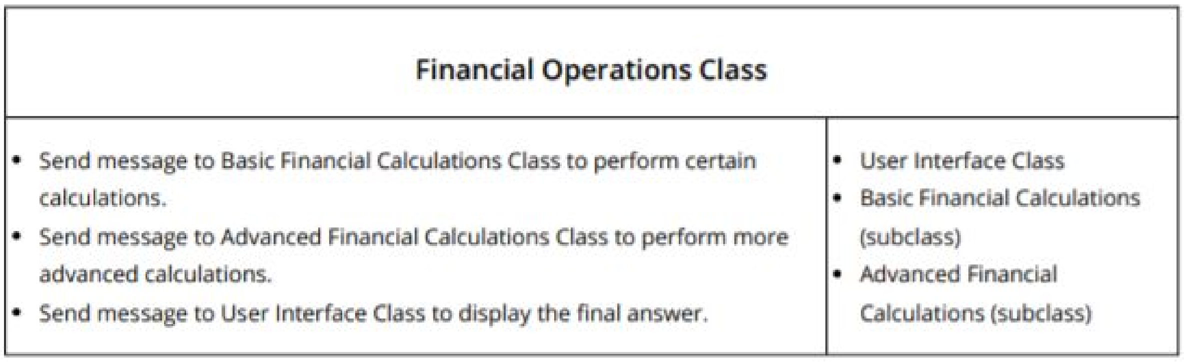
Class Models



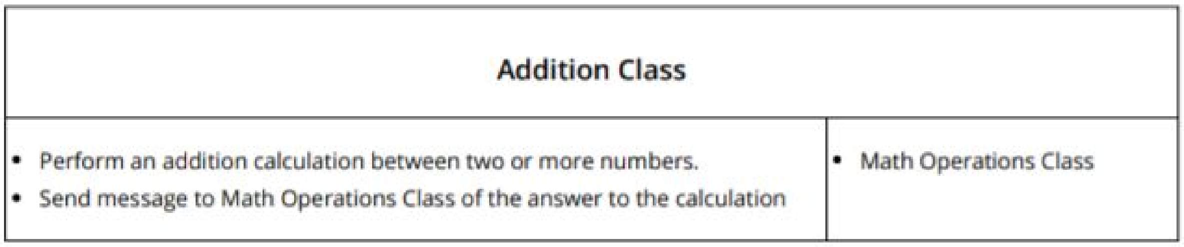
User Interface Class	
• Send message to Math Operations Class to perform addition, subtraction, multiplication, division. • Send message to Financial Operations Class to perform basic and advanced financial operations.	• Math Operations Class (subclass) • Financial Operations Class (subclass)



Math Operations Class	
• Send message to Add Class to perform addition. • Send message to Subtraction Class to perform subtraction. • Send message to Multiplication Class to perform multiplication. • Send message to Division Class to perform division. • Send message to User Interface Class to display final answer.	• User Interface Class • Add Class (subclass) • Subtraction Class (subclass) • Multiplication Class (subclass) • Division Class (subclass)



Financial Operations Class	
• Send message to Basic Financial Calculations Class to perform certain calculations. • Send message to Advanced Financial Calculations Class to perform more advanced calculations. • Send message to User Interface Class to display the final answer.	• User Interface Class • Basic Financial Calculations (subclass) • Advanced Financial Calculations (subclass)



Addition Class	
• Perform an addition calculation between two or more numbers. • Send message to Math Operations Class of the answer to the calculation	• Math Operations Class

Subtraction Class	
• Perform a subtraction between two or more numbers. • Send message to Math Operations Class of the final answer.	• Math Operations Class

Multiplication Class	
• Perform multiplication between two or more numbers. • Send message to Math Operations Class of final answer.	• Math Operations Class

Division Class	
• Perform division between two or more numbers. • Send message to Math Operations Class of final answer.	• Math Operations Class

Basic Financial Calculations Class	
• Send message to Present Value Class to perform a present value calculation. • Send message to Future Value Class to perform a future value calculation. • Send message to Interest Calculator Class to perform interest related calculation. • Send message to Payments Calculator Class to perform calculation for payment. • Send message to Financial Calculations Class with the final answer.	• Financial Operations Class • Present Value Class (subclass) • Future Value Class (subclass) • Interest Calculator Class (subclass) • Payments Calculator Class (subclass)

Present Value Class	
• Perform a basic present value calculation. • Send message to Basic Financial Calculations Class with final answer.	• Basic Financial Calculations Class

Future Value Class	
• Perform a basic future value calculation. • Send message to Basic Financial Calculations Class with final answer.	• Basic Financial Calculations Class

Interest Calculator Class	
• Perform calculations to find out interest. • Send message to Basic Financial Calculations Class with final answer.	• Basic Financial Calculations Class

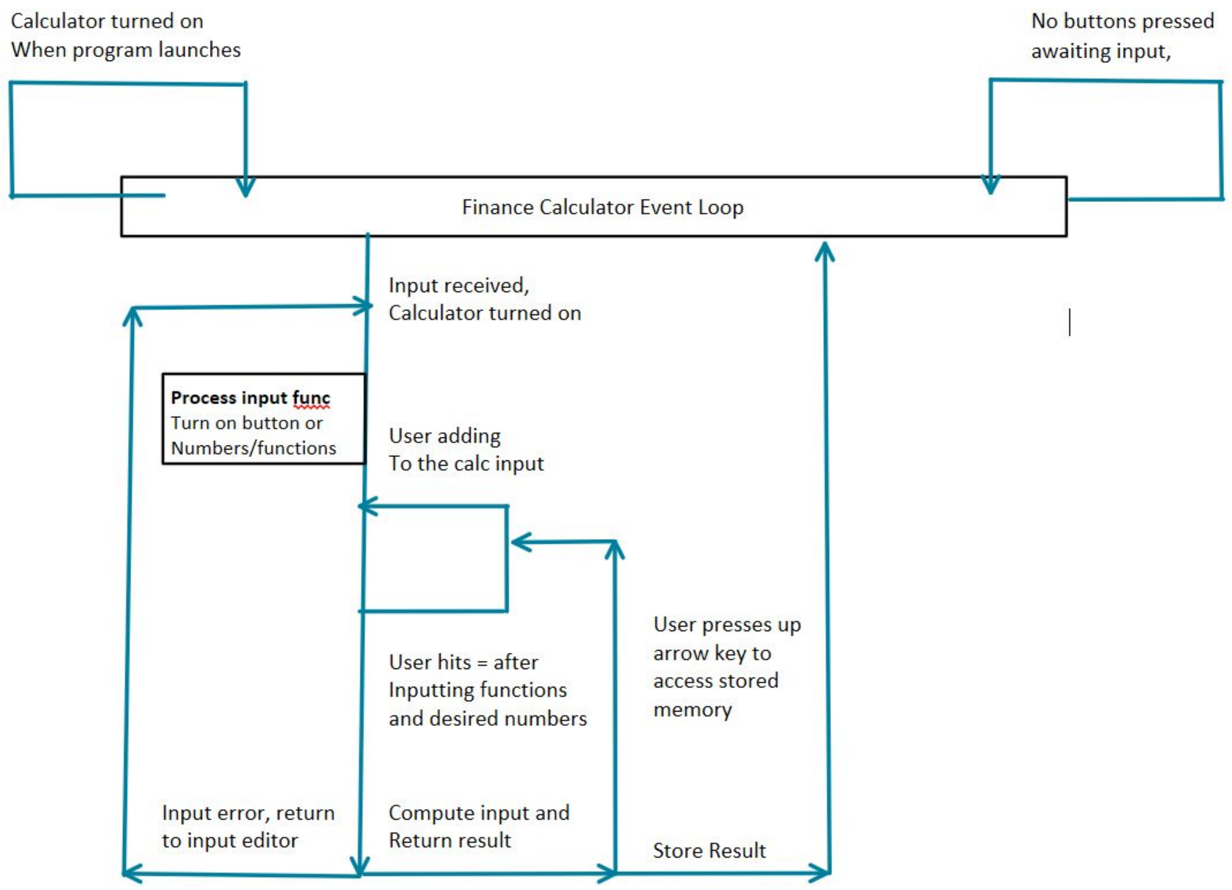
Payments Calculator Class	
• Perform a calculation to find out the number of payments. • Send message to Basic Financial Calculations Class with final answer.	• Basic Financial Calculations Class

Advanced Financial Calculations	
• Send message to Present Value OA Class to perform ordinary annuity present value calculation. • Send message to Principal Payoff Class to perform a principal payoff calculation. • Send message to Financial Operations Class with final answer.	• Financial Operations Class • Present Value OA Class (subclass) • Principal Payoff Class (subclass)

Present Value OA Class	
• Perform a ordinary annuity present value calculation. • Send message to Advanced Calculations Class with final answer.	• Advanced Financial Calculations Class

Principal Payoff Class	
• Perform a calculation to find principal payoff. • Send message to Advanced Calculations Class with final answer.	• Advanced Financial Calculations Class

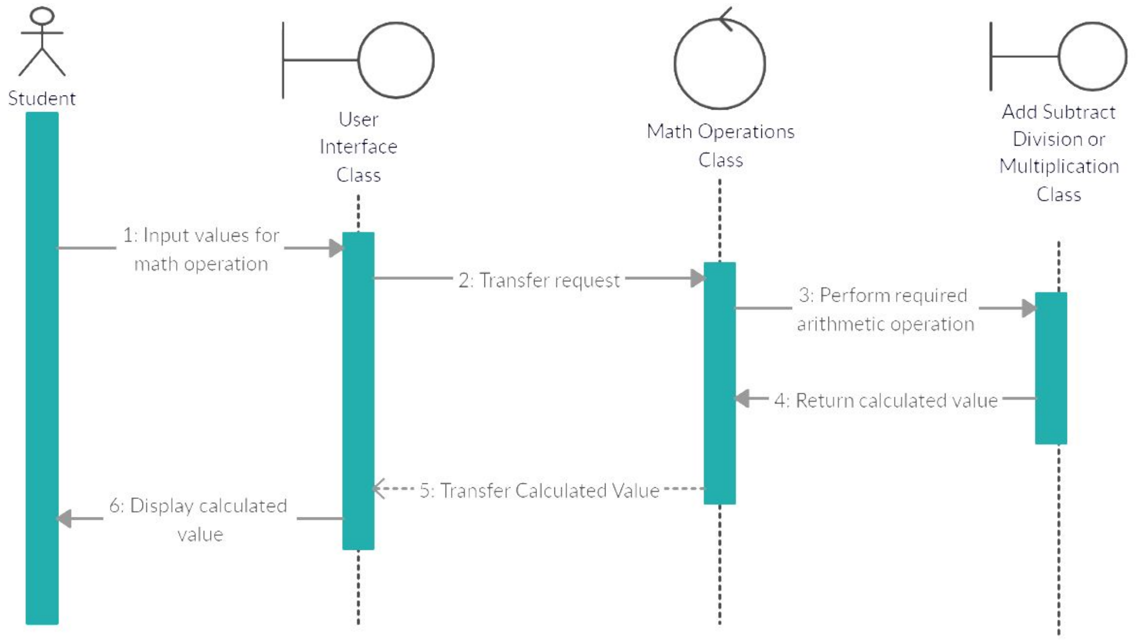
Dynamic Models



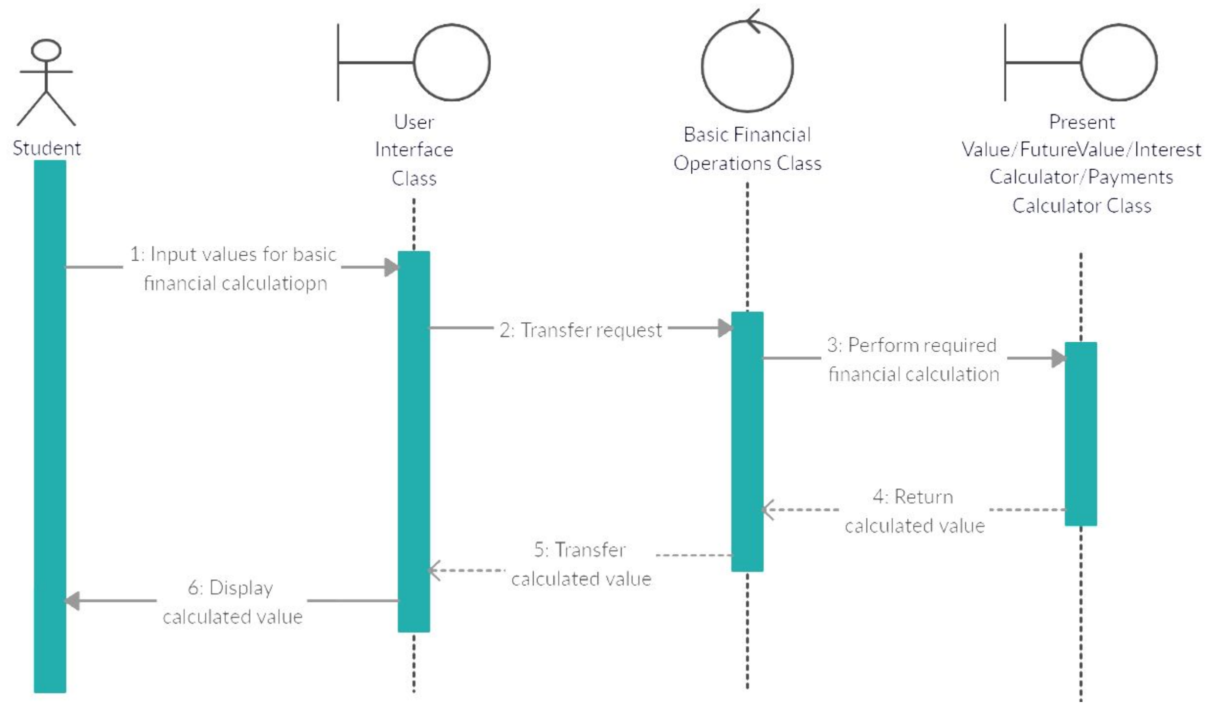
Object-oriented design

Interaction Diagrams - Sequence Diagrams

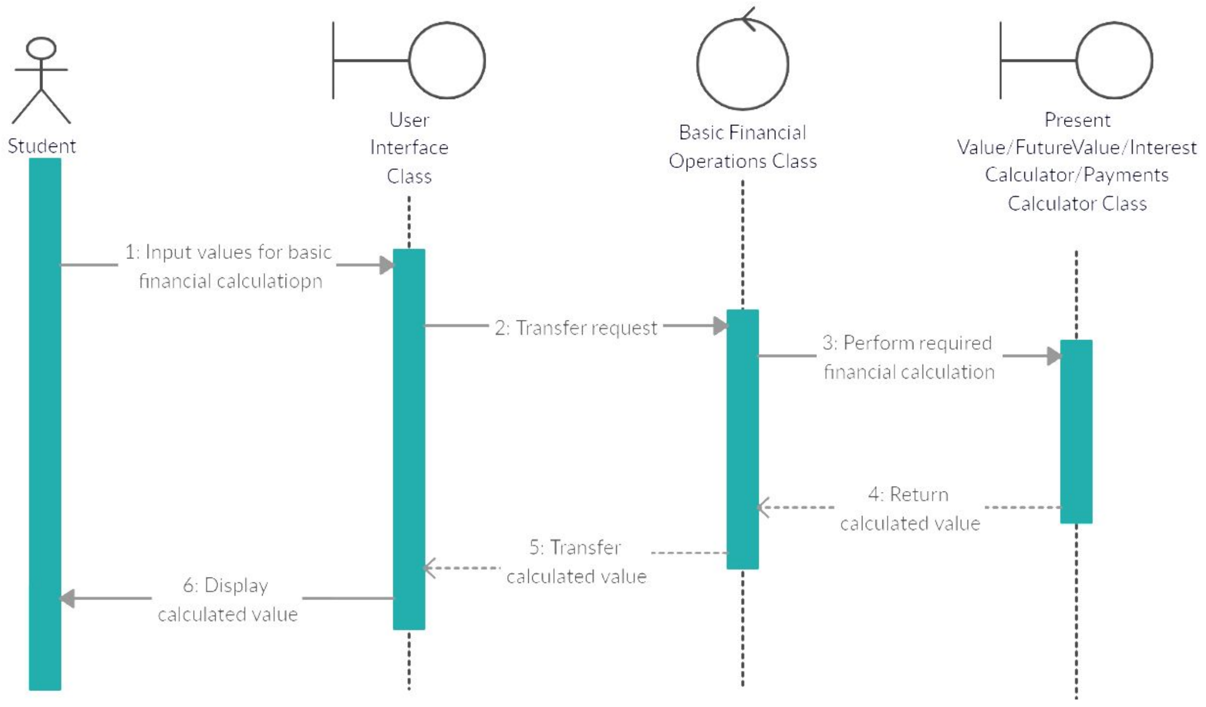
Use Case: Arithmetic Calculation


Student
User Interface Class
Math Operations Class
Add Subtract Division or Multiplication Class
1: Input values for math operation
2: Transfer request
3: Perform required arithmetic operation
4: Return calculated value
5: Transfer Calculated Value
6: Display calculated value

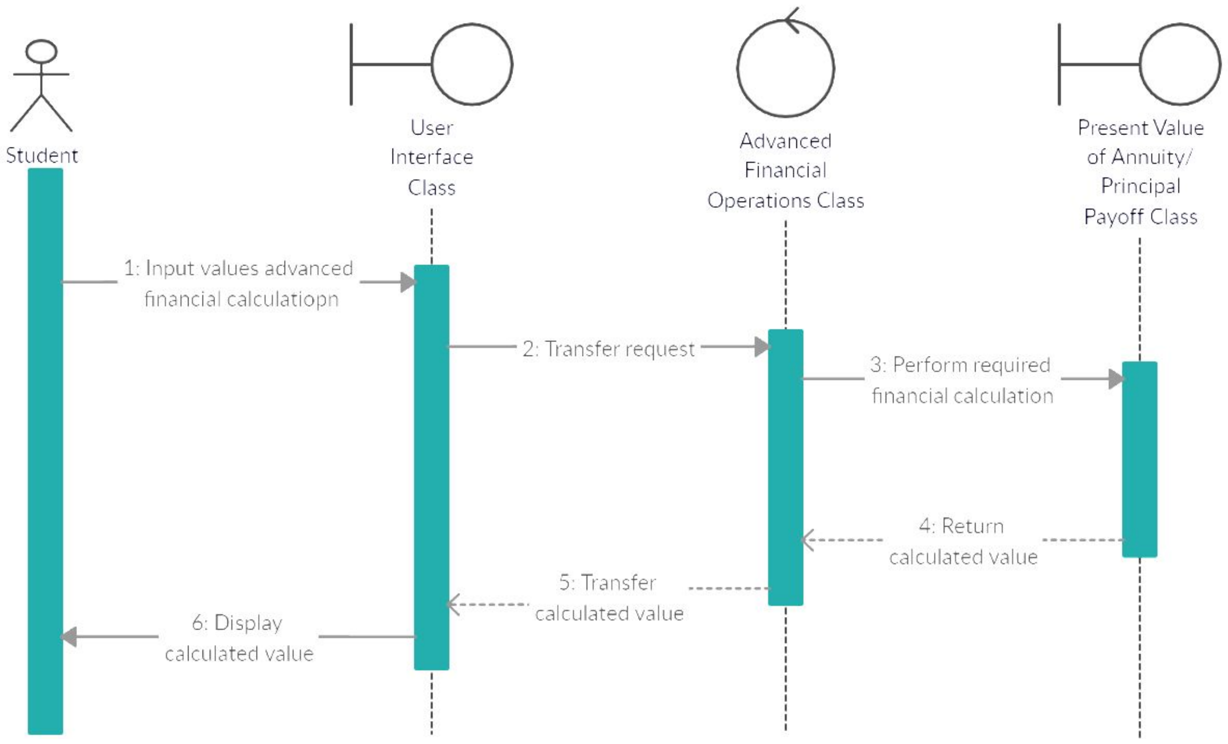
Use Case: Basic Financial Calculations


Student
User Interface Class
Basic Financial Operations Class
Present Value/FutureValue/Interest Calculator/Payments Calculator Class
1: Input values for basic financial calculatiopn
2: Transfer request
3: Perform required financial calculation
4: Return calculated value
5: Transfer calculated value
6: Display calculated value

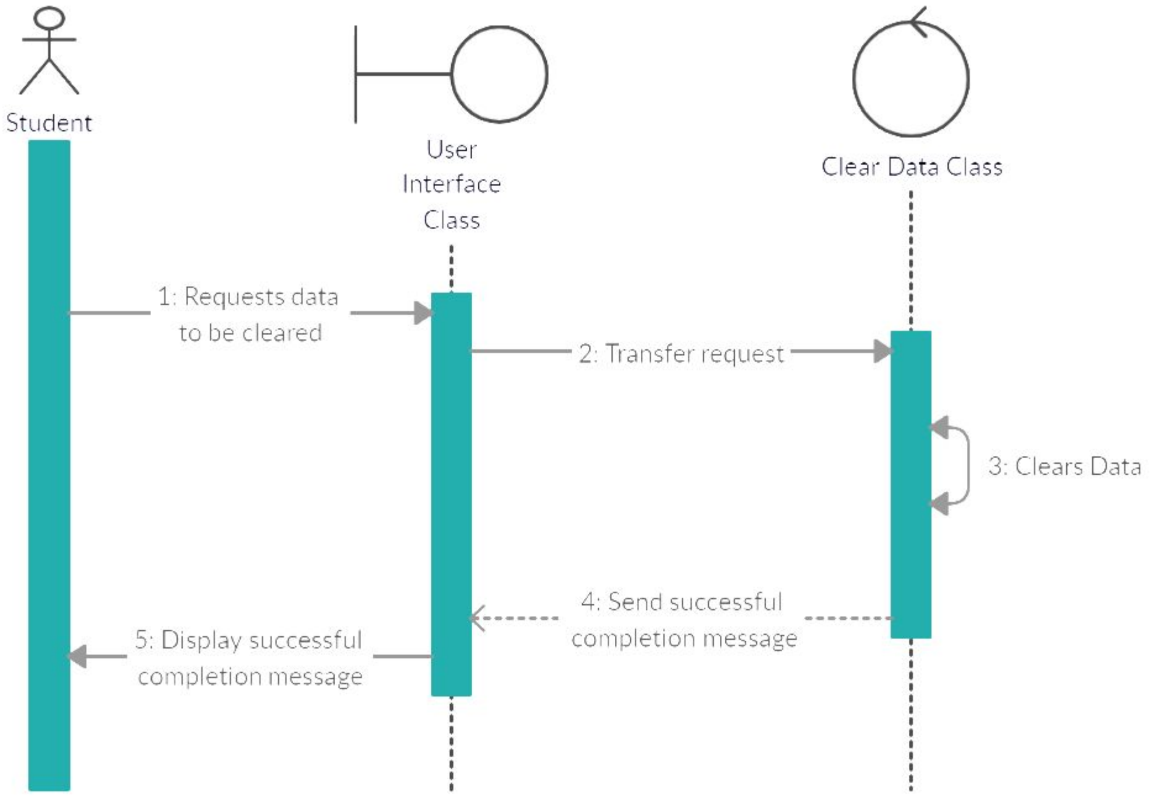
Use Case: Basic Financial Calculations


Student
User Interface Class
Basic Financial Operations Class
Present Value/FutureValue/Interest Calculator/Payments Calculator Class
1: Input values for basic financial calculatiopn
2: Transfer request
3: Perform required financial calculation
4: Return calculated value
5: Transfer calculated value
6: Display calculated value

Use Case: Advanced Financial Calculations

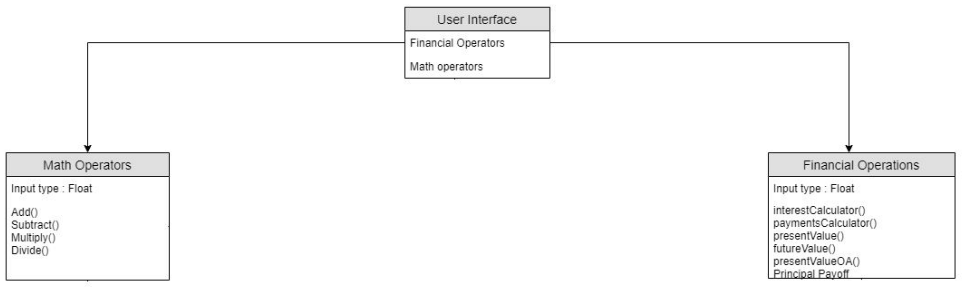

Student
User Interface Class
Advanced Financial Operations Class
Present Value of Annuity/ Principal Payoff Class
1: Input values advanced financial calculatiopn
2: Transfer request
3: Perform required financial calculation
4: Return calculated value
5: Transfer calculated value
6: Display calculated value

Use Case: Clear Stored Values

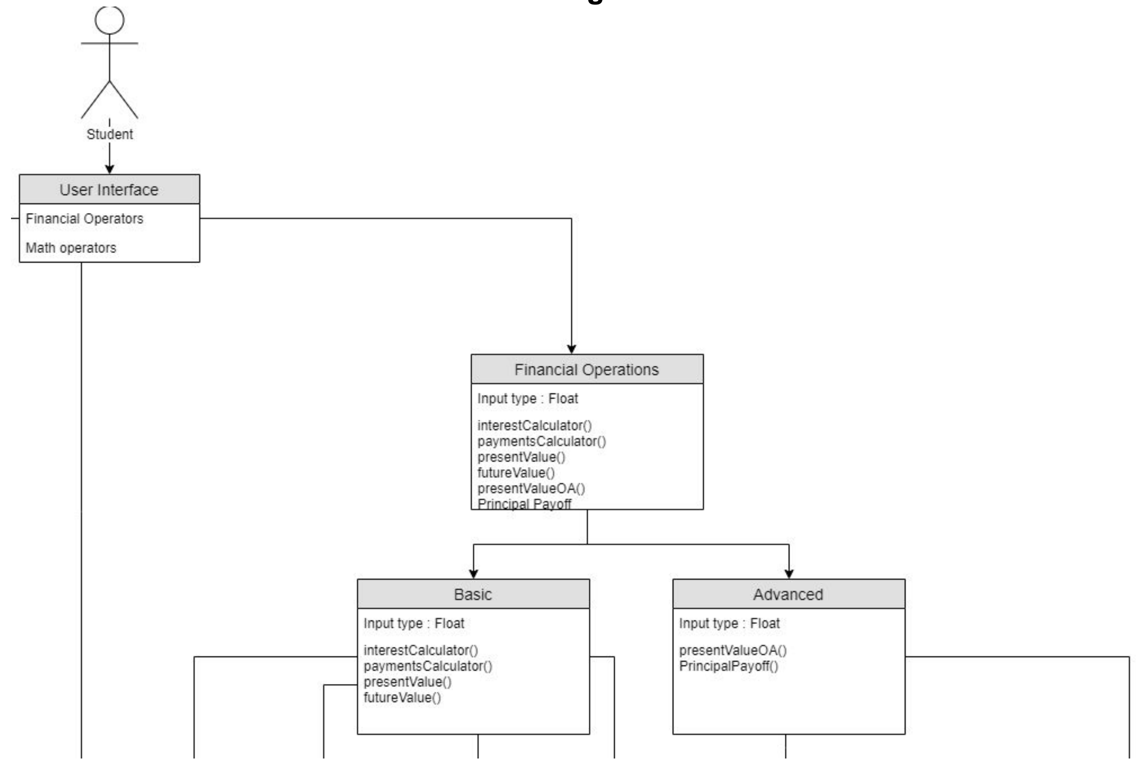

Student
User Interface Class
Clear Data Class
1: Requests data to be cleared
2: Transfer request
3: Clears Data
4: Send successful completion message
5: Display successful completion message

Detailed Class Diagrams

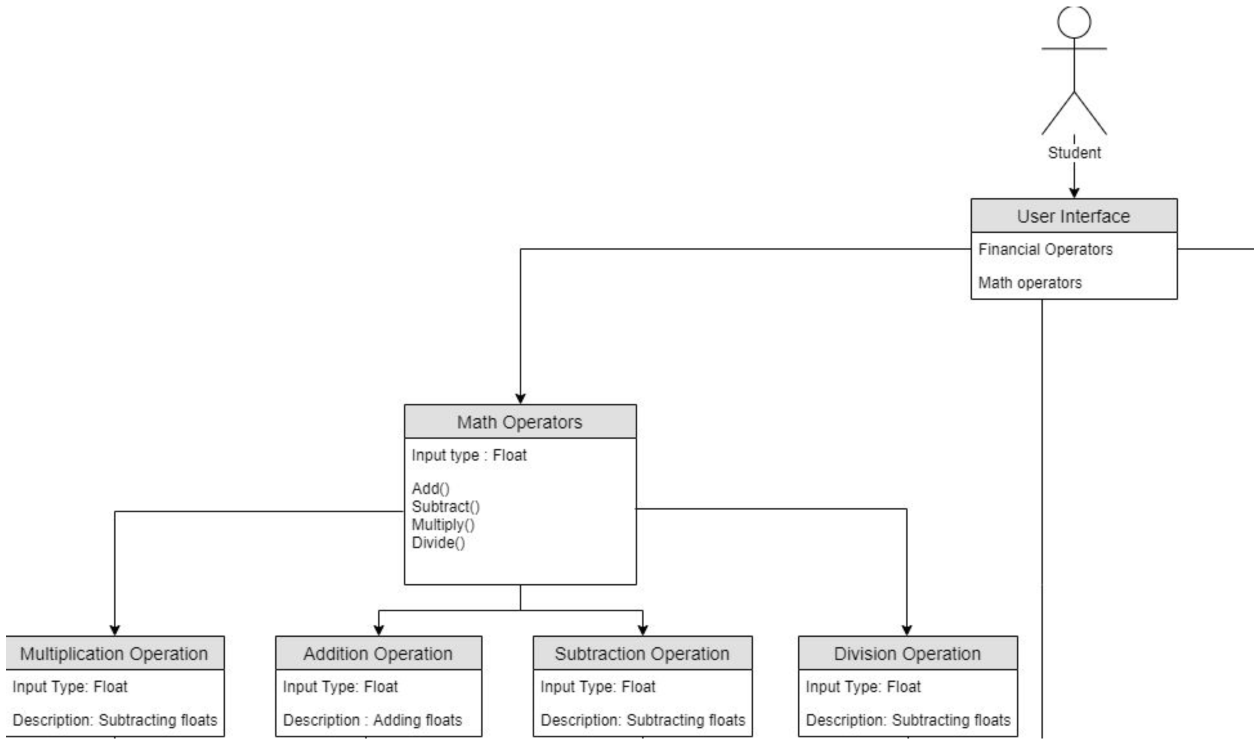
Initial class Diagram



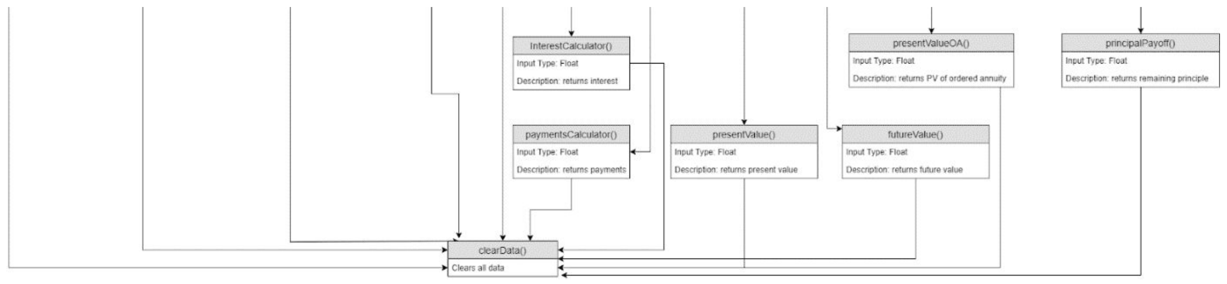
Right side



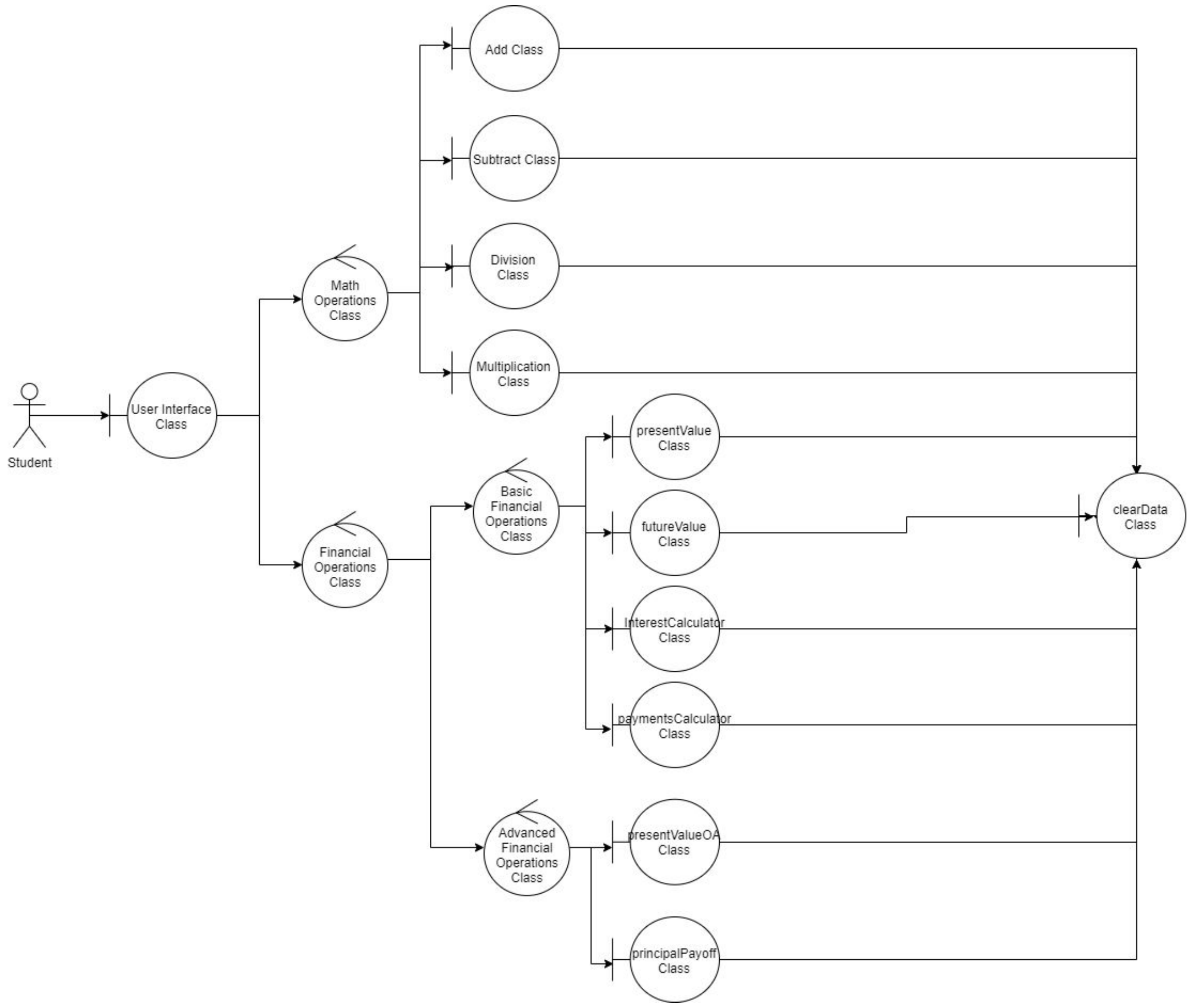
Left Side



Bottom of diagram



Final Class Diagram



Detailed Design (PDL Format)

User interface functions

```
//getting user input from the button
document.querySelector('.button').addEventListener('click', function(){


//function to implement for each button indicated below in Math Operations, Basic and
Advanced operations.


})


//clearData function
Void clearData() {
        document.querySelector('.clearData').addEventListener('click', function(){
                document.querySelector(‘data’) == ‘ ‘
        })
}
```

Math Operations (Arithmetic)

```
//to add
int addition(int a, int b) {


        Return a + b;
}


//to subtract
int subtraction(int a, int b) {


        Return a - b;
}


//to divide
int division(int a, int b) {


        Return a / b;
}


//to multiply
int multiplication(int a, int b) {


        Return a * b;
}
```

Basic Financial Operations

```
//to find present value
float presentValueCalc(float futureValue, int n, float i) {


        Return (futureValue/(1 + i) ** n);
}


//to find future value
float futureValueCalc(float presentValue, int n, float i) {

        Return (presentValue * (1 + i) & n);
}


//to find interest amount
float interestCalc(float futureValue, float presentValue, int n) {

        Return ((futureValue/presentValue)**1/n) - 1;
}


// to find out the number of payments remaining
float paymentCalc(float futureValue, float presentValue, float i) {

        Return (ln(futureValue/presentValue) / ln(1 + i));


}
```

Advanced Financial Operations

```
float presentValueOA(float futureValue, int n, int m, float i) {

        float pvifa = (1- (1/((1+i/m)**(n*m)) / i) * (1 +(i/m));
        Return ( futureValue*pvifa);
}


float principlePayoff(float presentValue, int n, int m, float i){

        float pvifa = 1- (1/(1+i/m)**(n*m))) / i;
        float payment =  presentValue/pvifa;
        Int interest = presentValue * i;
%       Return ( presentValue - (payment - interest) );
}
```

Metrics for Design and Analysis

We know that higher cohesion leads to more efficient and complete programs. This will play a vital role in our program due to algorithmic efficiency and due to the nature of our program (calculator). Coupling however, needs to be lower. This is the case due to interdependencies within the program. For our Fin-tastic! Calculator we have integrated data coupling from one function to another to help communication and limit interdependence within our program.

Cohesion

The Fin-Tastic Calculator system is designed to have a high level of cohesion. Cohesion is a measure of the strength of the relationship between the class's methods and the data itself. Our program was designed where smaller blocks can be used within larger areas of code, increasing the level of reusability and robustness of code. For example, many of the financial calculations rely on the same concepts used in the arithmetic calculations. As we are reusing logical concepts, there is a reduced level of module complexity and the code remains robust.

Coupling

The Fin-Tastic Calculator system is designed with low coupling, in the form of Data Coupling. Each individual component is interdependent from the other and the only way they interact with each other is by passing along data. For example, when a student inputs values through the user interface, the value itself is passed along to the subsequent classes. High cohesion and low coupling is the desired outcome in system design, which we believe is achieved.

Fault Statistics

The application creation included following the object-oriented paradigm with emphasis on modular and reusable code. We predicted how prone the app is to faults based on the severity of the fault. Since Fin-tastic! Calculator does not request any personal information from the user, fault severity is low. Possible faults include the application crashes, however that never occured during theoretical and experimental testing. If it does occur, the fault lies on the user sides and refreshing the application should fix it.

Test Workflow
Example 1: Calculate the missing TVM value by pressing the TVM button on the calculator. A new form will pop up with all the required variables. To solve for FV, the user will enter n i, and PV which correspond to the number of years, interest rate and present value. Finally the user will hit calculate on the missing value (in this case FV) and receive the value..

Example 2: Calculate the advanced TVM by pressing the advanced TVM button on the calculator. A new form will pop up with blank required variables. If the user is solving for PV, the user will enter n, i, and FV. Since it's advanced TVM the user will also need to enter values for m. Finally the user will hit calculate on the missing value (PV in this case) and will receive the value.

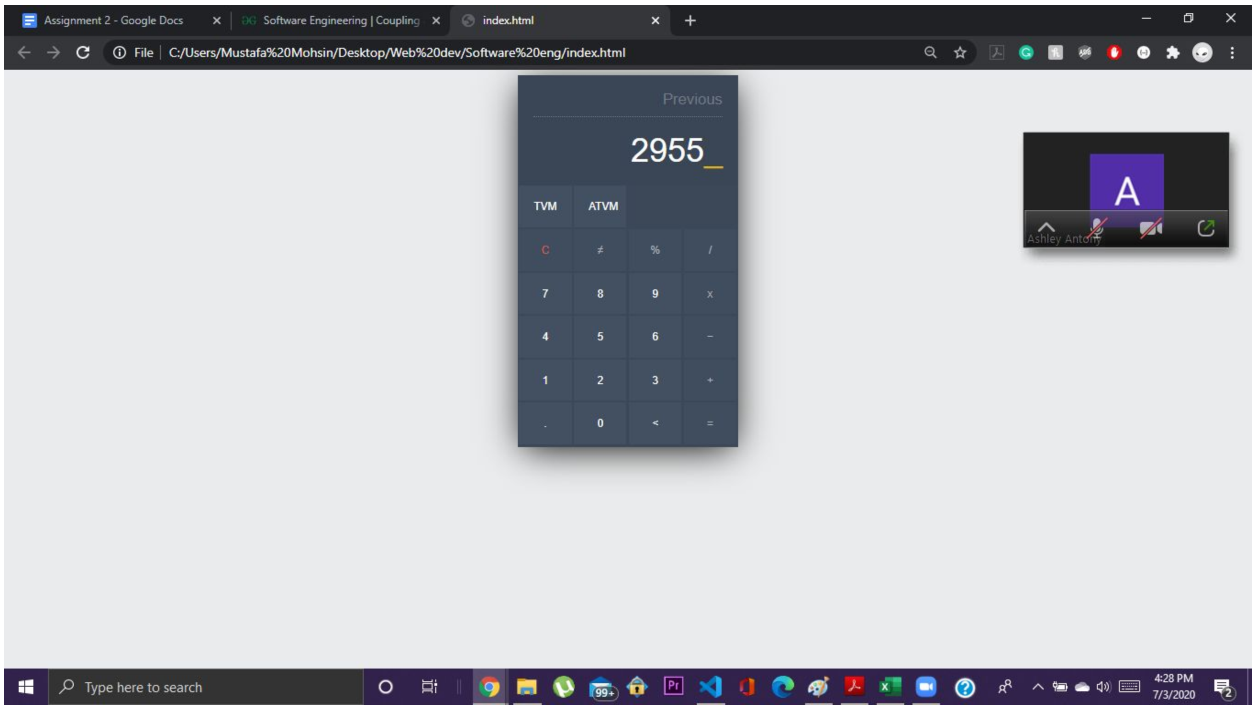
Example 3: Calculate the sum of 2 or more values. The user will first input a value (e.g. 37), to do this they have the option of typing it via keyboard or pressing the GUI buttons. The user then enters a second number (e.g. 3) and has 2 choices from this point. They can either find the sum by hitting = or they can repeat the process and continue adding more numbers to add.

Example 5: The user cannot enter an incalculable input. In other words, the calculator should return an error message when the user enters operators (e.g. +-/*) without numbers on which to operate. Test by entering +1+ and receiving an error.

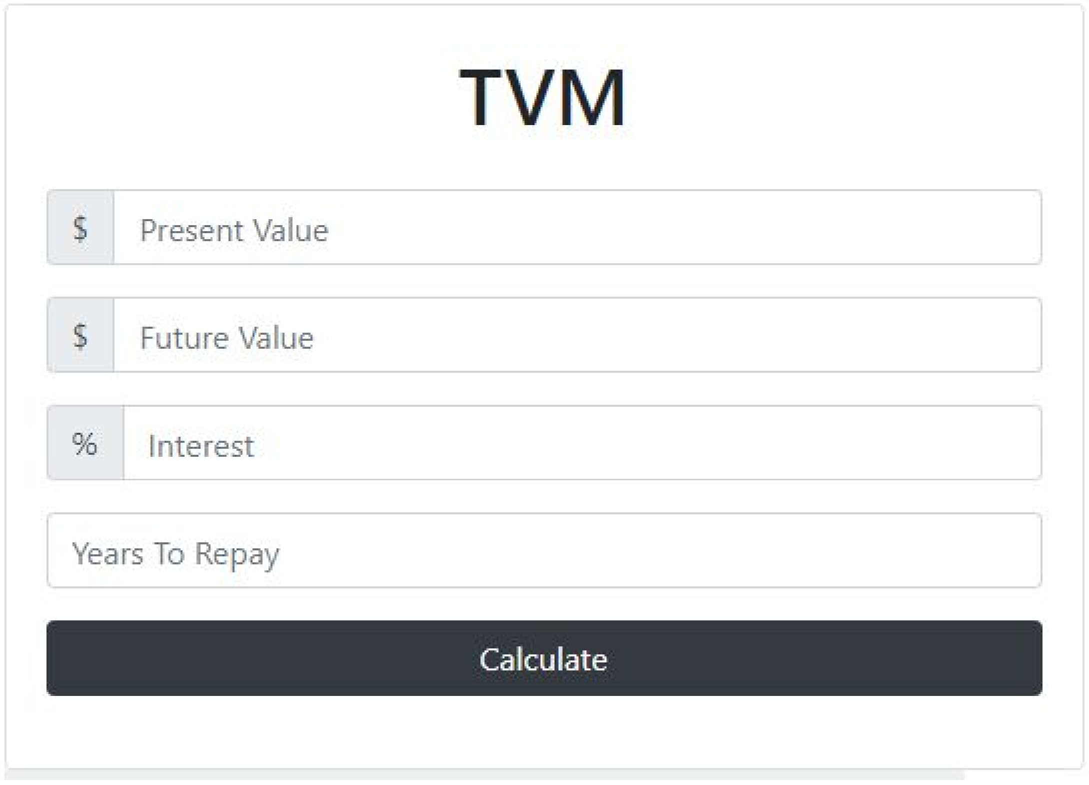
Updated GUI Examples (from Assignment 1)

From the feedback given on Assignment 1, we have decided to update our GUIs and have it be more in depth. Here are the examples:

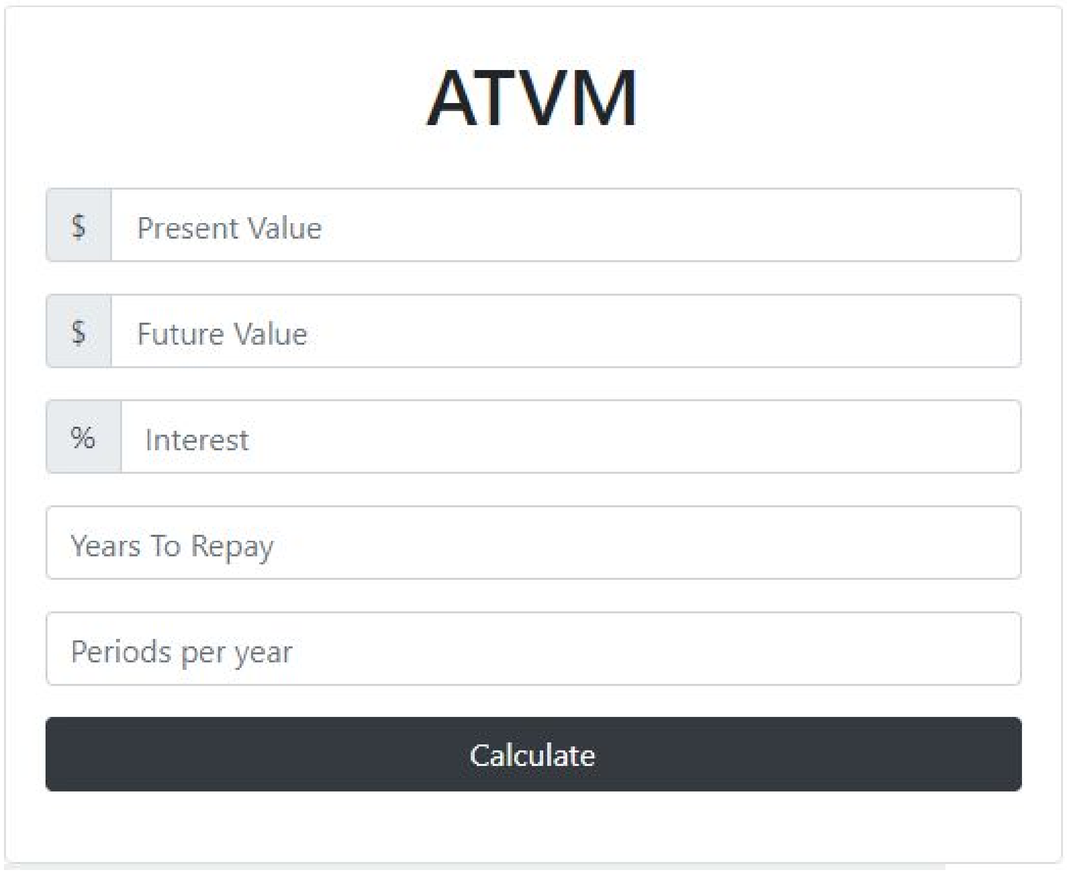
<u>Basic UI</u>


Assignment 2 - Google Docs
Software Engineering | Coupling
index.html
File | C:/Users/Mustafa%20Mohsin/Desktop/Web%20dev/Software%20eng/index.html
Previous
2955_
TVM
ATVM
C
≠
%
/
7
8
9
x
4
5
6
−
1
2
3
+
.
0
<
=
Ashley Antony
Type here to search
4:28 PM
7/3/2020

<u>Basic Financial Calculations UI</u>



Advanced Financial Calculations UI



Actual Code

We used Git Version Control to manage our code. The final updated code can be viewed at: https://github.com/MustafaMasud/Fin-tastic-Calculator

Individual Work done

Alex Ojo
- Presentation: Overview of product (use case walkthrough) and testing slides
- Testing + error catching in code

Ashley Antony
- User manual
- ½ Project schedule
- ½ Gantt Chart
- Formatting final submission

Mustafa Mohsin
- Complete project coding
- Help offloading members with redundant tasks.
- Demonstration of the project

Guilhem
- ½ Gantt Chart
- ½ project Schedule
- Programming documentation prologues

Sobihan Mamugan
- Helped with completion of project coding
- Basic Calculator and Problem Statement presentation