6COSC021W Lab Based Practical – Mock Test Specification

Week 8, Semester 1, 2024 – Time 11am, 2pm, 4pm.

You have 2 hours to complete this assessment. You will be told when you have; 1 hour, 30 minutes, and 15 minutes remaining. You must upload your work to Blackboard once the 2 hours have elapsed.

This practical assessment takes place in exam conditions, and all usual rules and regulations apply.

Before the assessment starts:

1) Download the assessment project template from Blackboard.
2) Run the project and ensure there are no errors or build issues.
3) Make sure you have to hand any allowed notes you require. See Appendix A.
4) Spend the allowed time inspecting the starter project and planning your approach.
   Note that the code's main structure will be stubbed out with comments indicating what code needs to be placed in these sections.
5) Familiarise yourself with this brief and ask questions if you are unclear on anything.

Instructions

You are to build a tab-based savings calculation application in SwiftUI that saves the calculations and stores them using Swift Data. One view facilitates the calculation, and one displays a previous calculation history.

The application allows the users to enter:

a) The initial amount invested. The default **initialAmount** is zero
b) The monthly savings contribution. The default ***monthlyContribution*** *is 250*.
c) Interest rate in (%). The default ***interest*** *is 2%*.
d) Duration of the savings in years. The default ***period*** *is 5 years*.

   Important: the names in bold above should be used for the @State vars in the Mortgage view.

The application outputs:

a) The future value of the savings (£).

User Interface

The app is tab-based and has two views. One view is the 'Savings' calculation, and the second view is the 'History' view of previous calculations. The user interface should look as close as possible to the one shown in Figure 1 and Figure 2 below.

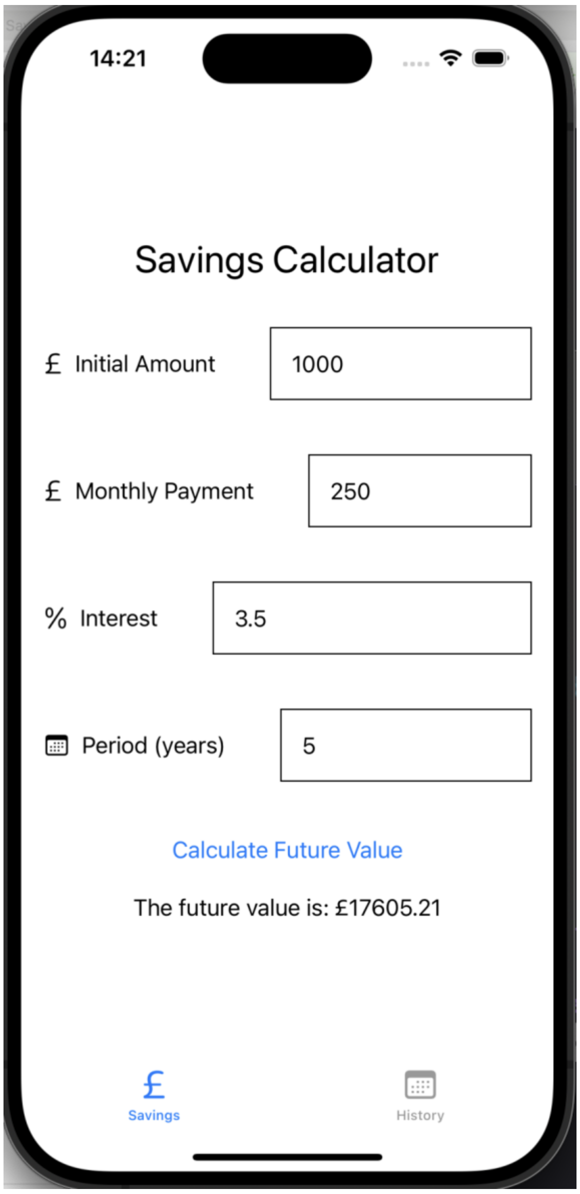
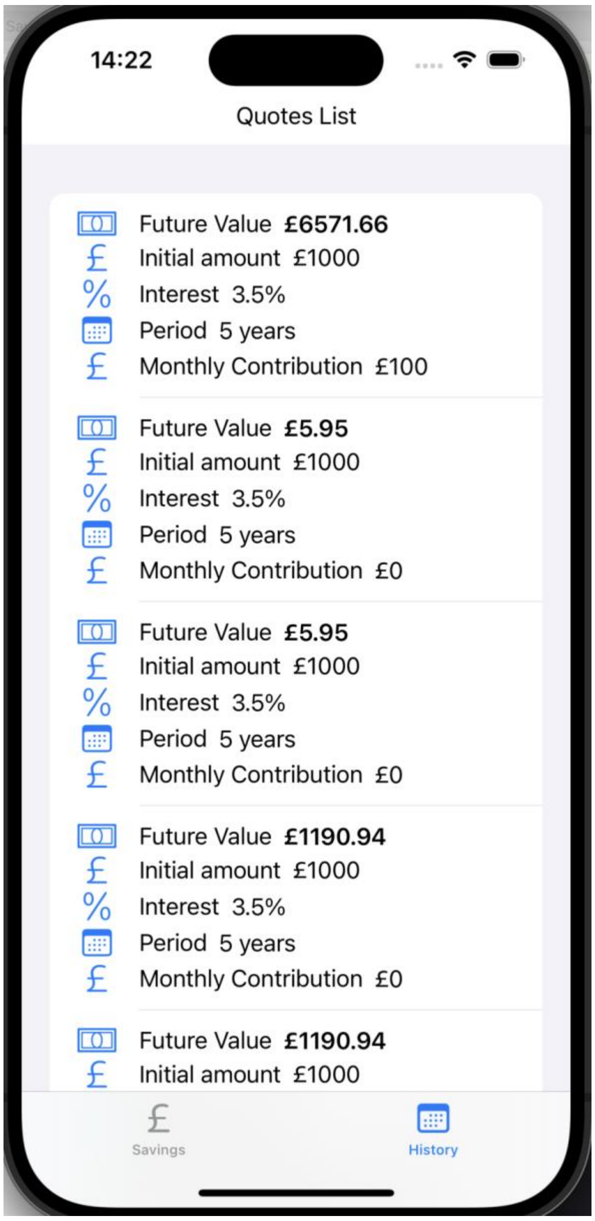


Figure 1 Mortgage Calculation View. The second image shows an Alert that is presented if any of the text fields are when the button is pressed.



Future Value **£6571.66**
Initial amount £1000
Interest 3.5%
Period 5 years
Monthly Contribution £100

Future Value **£5.95**
Initial amount £1000
Interest 3.5%
Period 5 years
Monthly Contribution £0

Future Value **£5.95**
Initial amount £1000
Interest 3.5%
Period 5 years
Monthly Contribution £0

Future Value **£1190.94**
Initial amount £1000
Interest 3.5%
Period 5 years
Monthly Contribution £0

Future Value **£1190.94**
Initial amount £1000

Figure 2 The Quotes List View

History View

The History view is a List View that shows all the previous calculations. These are retrieved via Swift Data.

Tasks

1) Incorporate the views **SavingsMainView** and **QuotesView** into the **MainTabView**.
2) Implement and style the SavingsMainView UI – *see Figure 1* . See Appendix B for suggested SF symbol names.
3) Implement the necessary, state vars for the text fields and for the future value output.
4) Incorporate the function *calculateSavings(…) and hideKeyboard(…)* that calculates the future value. Hint: Attach the function to the button so the calculation is performed when the button is touched.
5) Test that this part of the app works using the values shown in figure 1.
6) Implement the code to ensure the future value output string is formatted to two decimal places.

7) Implement the necessary code to ensure that any data entered into the text fields is persisted if the application is closed/backgrounded. (Hint use @AppStorage).

   You should now work on **History** view.

8) Implement the SavingsModel class. The template for this can be found in the SavingsModel.swift file.
9) In the calculateSavings(…) function, add the necessary code to insert a new instance of the model into the **modelContext.** See the hints in the comments of the function.
10) Test your app still works, and it is a good idea to make a copy of the project at this point.
11) Now open the QuotesView and implement the required code to populate the **List**.
12) Ensure that the items in the List are formatted exactly as shown in figure 2.
13) Test the History view to ensure that the List is being populated and that formatting is correct.
14) Zip your complete project and upload it to Blackboard.

Indicative Mark Scheme

(note code must be understandable, well named, maintainable and efficiently coded to achieve high marks in any one section below)

Task 1:

1) The SavingsMainView *vars* are correctly defined and persisted on close. – total 10 marks
2) Labels implemented and correctly formatted. – total 8 marks
3) Text fields are correctly formatted and coded with placeholder and with the appropriate keyboard – 12 marks
4) Button formatting and action code correctly implemented. – total 8 marks
5) Future value text output correctly coded and formatted, including string formatting to two decimal places – total 6 marks
6) SavingsModel class correctly and efficiently coded – total 6 marks
7) Model instantiation correctly coded and instance saved on calculation – total 6 marks
8) List view correctly implemented in the Quotes view – 16 marks
9) List items correctly formatted as shown in figure 2 – 12 marks
10) Main Tab View correctly implemented – 8 marks
11) View correctly incorporated with the correct icons and labels into the Tab View. – 8 marks

Total: 100 marks.

Appendix A

Allowed noted are:

1) Lecture notes (Blackboard site allowed)
2) Seminar notes (Blackboard site allowed)
3) Apple API documentation from within XCode only

Assets

The **SF Symbols** used for the **Label**s are: *house.fill, banknote, percent, calendar, and sterlingsign.*

No code can be obtained from any other source, as this will constitute academic misconduct.