Budget Facilitator
Requirements Specifications

Group 3
Shandelle Murray
Athidya Raveenthraehru
Mitchell Coovert

Domain

"Budget Facilitator" is a simple program which allows users to organize their finances by sorting their purchases into categories, see visual representations of their spending, comparing spending to their budget and getting notifications when their spending exceeds their budget. This program is a java program at the moment but would ideally be a mobile application in the future that users will be able to access using an application store on their phones or an equivalent service. The stakeholders of this program will be the programmers, the users, the TA's and professor Reza.

Users will input their individual purchases whenever they please including information such as item name, price, date and category of item. These inputs will be taken and shown in a graph or a list format over a time period of their choice. It will also be comparable to their budget for a certain time period or for each category over a time period. The only input for this program will be user's input and the output will be the visual representations of their finances.

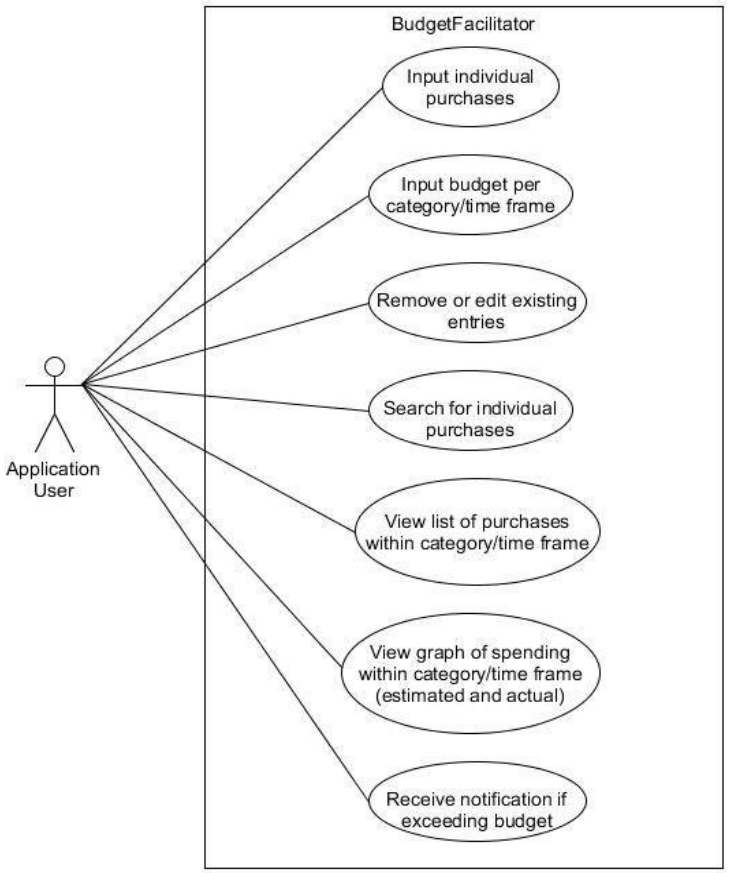
Functional Requirements

1. *Purchases Handling*

1.1 allow user to input their purchases (input name, category, price of item and date)
1.2 allow user to edit or delete purchases
1.3 let user search for individual purchase
1.4 store all individual purchases
1.5 organize purchases by category and date
1.6 visually display purchases in their ordered manner in a list format and graph formal (line graph, bar graph, pie graph)
1.7 delete purchases that are dated over 2 years previous to the current date

2. *Budget Facilitating*

2.1 allow user to input budget for each category
2.2 show budget comparison to their purchases in the graph and list formats
2.3 give notification if approaching or exceeding budget in a category



Non-Functional Requirements

1. *Reliability*

This program should be accessible to users as long as they have a computer that they can download the program on to and an application that will run java programs such as eclipse. The optimal finished product of this program would be a mobile application accessible on users phones via a mobile application store of some sort.

The user will be able to use this program on their personal computer, however an issue will arise if another person gains access to their computer. There is not a high chance of this greatly hurting the user because all the other person will have access to is their budget and what they've bought over a

certain time period. Still there is some sort of security that the computer itself provides such as password login to the users account. This program has no specific safety hazards to the user that are apart from the hazards that come along with using a computer.

2. *Accuracy of Results & Performance*

Optimally the program will have reliable results because there are only addition calculations that this program preforms. Other than actual calculations the other factors of this application that need to be monitors for accuracy are the efficiencies of the searching, sorting and graphing algorithms that are being used. It should allow user to input purchases and view results quite quickly, while being easy for the user to understand and quick to learn how to use.

3. *Constraints & Portability*

The only constraint of this application would be the lack of user input. Say those who are not willing to input each of their purchases would not be able to keep a very efficient system and therefore this program will ultimately be useless to them. A physical dependency that this program would have is the space available wherever this application is running. There needs to be a suitable amount of space available in order to store all the information the user inputs as well as to store the program itself. It has been limited to storing 2 years of purchases so this would be a near constant amount of space necessary for storing information. This program is not very portable unless used on a laptop. Hopefully in the future will be more portable after implemented as a mobile application.

Development & Maintenance (Test Procedures)

Blackbox testing must be performed during the development and maintenance stages of the application's life cycle. The following is a test plan that highlights the test cases that need to be considered to ensure that the requirements are being met.

Test Plan:

1. Given the precondition that there have not been any user inputs into the application within the specified time period (for instance, within the day, week, or month), the following should be verified:

- The list of purchases within the time period contains no elements
- A search for an individual item returns that the item does not exist
- There is no functionality available to remove or edit items since no items currently exist
- The line graph displays a series of lines at $0 for each category representing actual spending which may or may not be accompanied by straight lines representing budgeted amounts, depending on if the user has already input their ideal spending amounts per category
- Should the user try to display the pie chart, the output is a message informing the user that there is no data to represent
- In the case of the bar graph, the budgeted amounts, if they have already been input, are represented by differently coloured bars, by category, but the bars for the actual spending are at $0. In the case where the budgeted amounts have not been input either, all of the bars are at $0.
- The user is able to add an item.

2. Given the precondition that there is one input within a specific time period:

- This item is search-able by name and is present in a list representing its date or category
- The user is able to remove the item entirely so that it does not appear in any searches, lists, or graphs
- The user is able to edit the item's cost, date, or category, and these changes are immediately reflected in the searches, lists, and graphs should the user choose to display them
- Each of the aforementioned graphs represent the item by its category, with the total spending amount equal to the cost of the item
- The user is able to add an item, which will be sorted by its date/category

3. Given the precondition that multiple items exist within a specific time period:

- The items are search-able by name and are be present in a list comprised of items also purchased on their respective date or within their category
- The user is able to remove any of the items entirely so that the removed item does not appear in any searches, lists, or graphs
- The user is able to edit any of the items' costs, dates, or categories, and these changes are immediately reflected in the searches, lists, and graphs should the user choose to display them
- Each of the aforementioned graphs represent the items within their category
- If any of the items are within the same category, the total amount spent within that category is equal to the sum of the costs of the individual items within that category
- If the user attempts to add an item, it should be inserted within the previously added entries in a sorted manner (by date/category)

4 . Given the preconditions that items have been input into the application and that one such item is dated back to 731 days or more prior (2 years), this item should be removed from the application so that it does not show up in any searches, lists, or graphs.

The functional requirements as outlined in this document must all be met in order for this application to function correctly. In terms of priority, the three most crucial functional requirements are 1.1, 1.2, and 1.4, that the user must be able to input items which must then be stored by the application but can be deleted or edited by the user at any time. If there are no items stored in the application, it cannot complete any of its other functions. Next, requirement 2.1, allowing the user to input their budget, should be prioritized, followed by requirements 1.5 and 1.6 that organize the items by date and category and display the amounts spent in graph and list formats. Subsequently, requirement 2.2, that a comparison be shown between the amounts budgeted and the amounts spent, is the next priority. The next requirement to be prioritized is 1.3 that states that the user is able to search for a specific item. The final items on the prioritized list of requirements are requirement 2.3, that the user receives a courtesy notification that they are about to exceed their budget, and requirement 1.7, that the application automatically deletes items that are dated back to more than two years ago.

Since spending and budgeting practices should not change a great deal in the future, there are very few foreseeable changes to the system maintenance/testing procedures outlined above. Depending on the implementation, it may have to be modified to suit a particular platform or operating system. In terms of high level changes, however, unless the requirements change or additional requirements are added to the problem domain, the above maintenance procedures should ensure that the requirements are being met. In the case where new functionality/requirements are added, testing for these requirements can simply be appended to the above procedures for whichever case applies (zero previous entries, one entry, or multiple entries).