Group 13 Budgeting Application Programmer's Manual

By: Brian Mounce and Carter King

Scope of the product:
This application was intended to function as a lightweight and easy to use budgeting app. The user is able to create a budget for each month and track their spending across any category that they choose.

Product Perspective:
This app was designed for ease of use by the user and should require little technical knowhow to operate. The features implemented should be narrow in scope so that the user does not struggle to find the features they wish to use.

Product Functions:
The fundamental function of this app is to keep track of the monthly spending for the user. The user will define their categories within each month. Each category will keep track of how much the user has spent in that category and how much remaining the user has to spend based on a user defined monthly limit. The program should also inform the users of the total amount they have spent in the month as well as the amount remaining and the collective limit they have set.

User Characteristics:
The end users will be those who do not need to maintain a high level of complexity with their budgeting and do not wish to use an app that is difficult to navigate or have excessive features. Easy of use and simplicity are the key offerings of this app.

General Constraints:
This software does not have any network functionality.

Assumptions and Dependencies:
This application assumes that the user has the Java SE platform installed (version 15.0.0 or later). It also assumes the user has a running instance of MySQL Community server on port 3306 with a user called app with the password 0. This user must have the proper privileges granted. This application also assumes that a DB with the name “budgetsdb” has been created. The specific sql to create the schema correctly can be found in the project directory.

Requirements

Graphical User interface
1.0.0) There should be a Graphical User Interface for the user to interact with the program

1.1.0) The data for the budget should be displayed in a table for the user to view and interact with

1.2.0) Actions performed on the table should be attached to clickable buttons

1.3.0) If an action would remove a row from the table a dialogue box should ask the user for confirmation

1.4.0) User input fields should be resistant to unexpected values.

1.4.1) Category Field should be unexpected value proof.

1.4.2) Spent TD Field should be unexpected value proof

1.4.3) Spend Limit Field should be unexpected value proof

Budget

2.0.0) Users should be able to define their own categories

2.0.1) Each category should have a user defined name

2.0.2) Each category should have a user defined spending limit

2.0.3) Users should be able to record the amount spent to date within a given category

2.0.4) The difference between the spending limit and the spent to date should be updated in real time

2.1.0) A running total for the categories should be maintained and updated in real time to reflect any changes to an individual category

2.2.0) The user should be able to create multiple budgets and be able to choose which they wish to edit

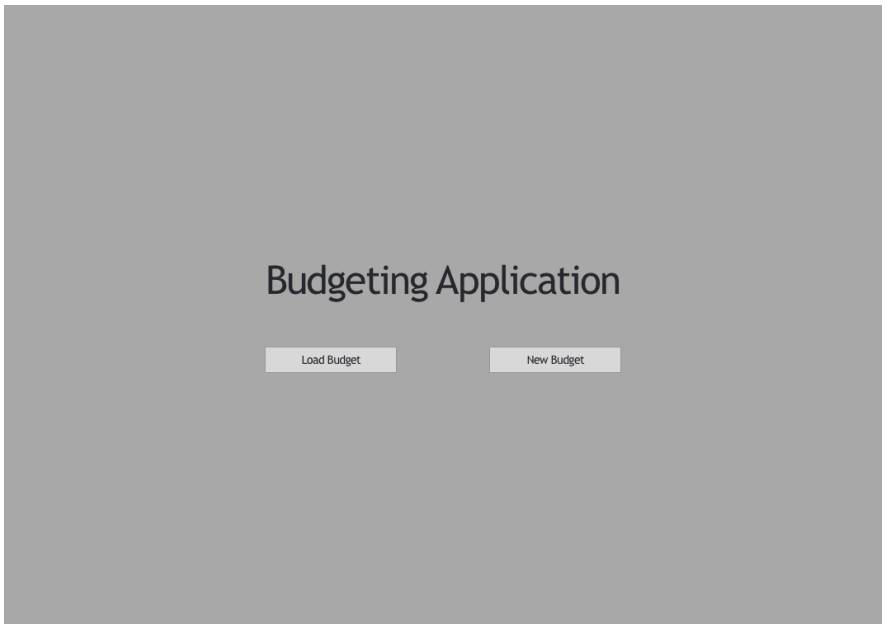
Database

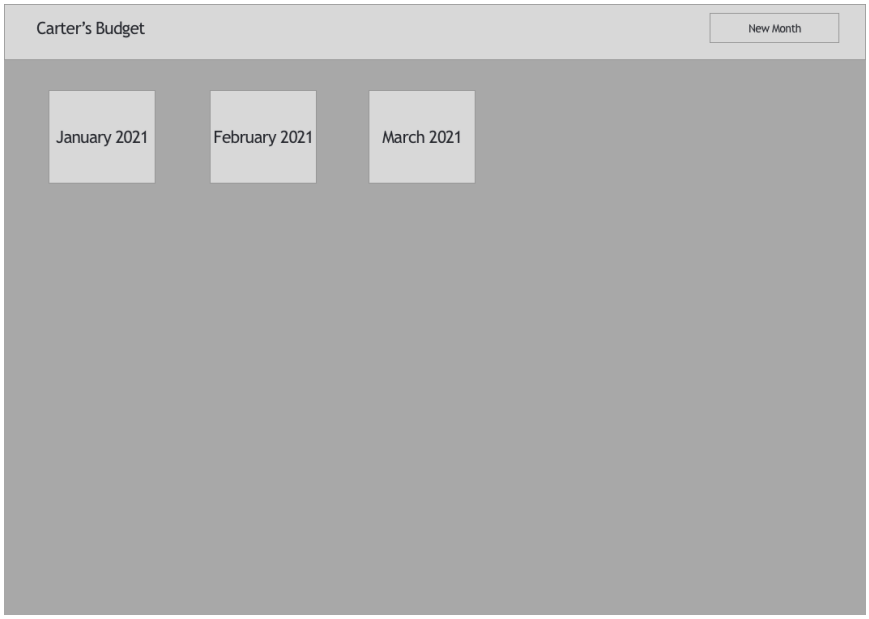
3.0.0) A database should be maintained to store any data the user enters to be retrieved for later use

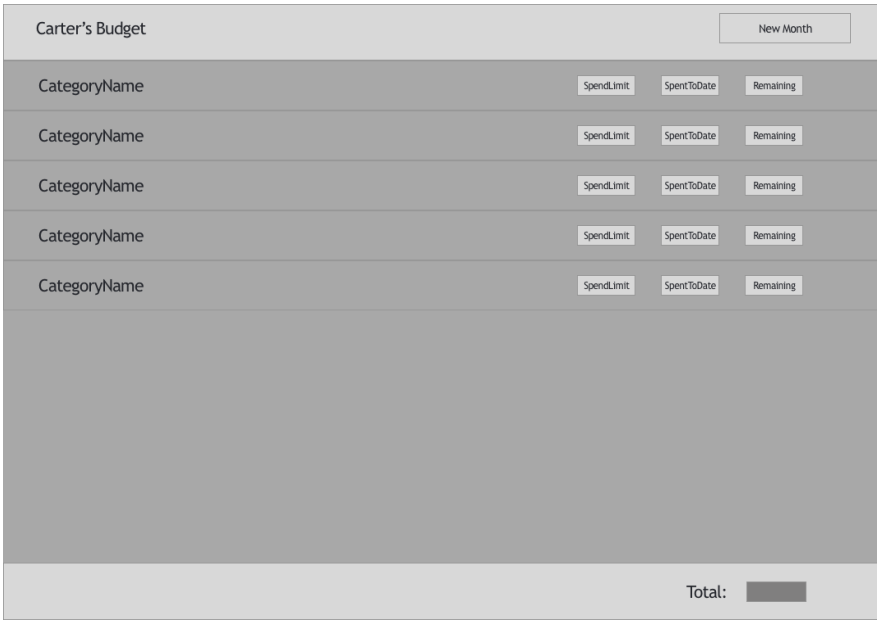
3.1.0) Front end classes should not be able to directly access the DB.

Design

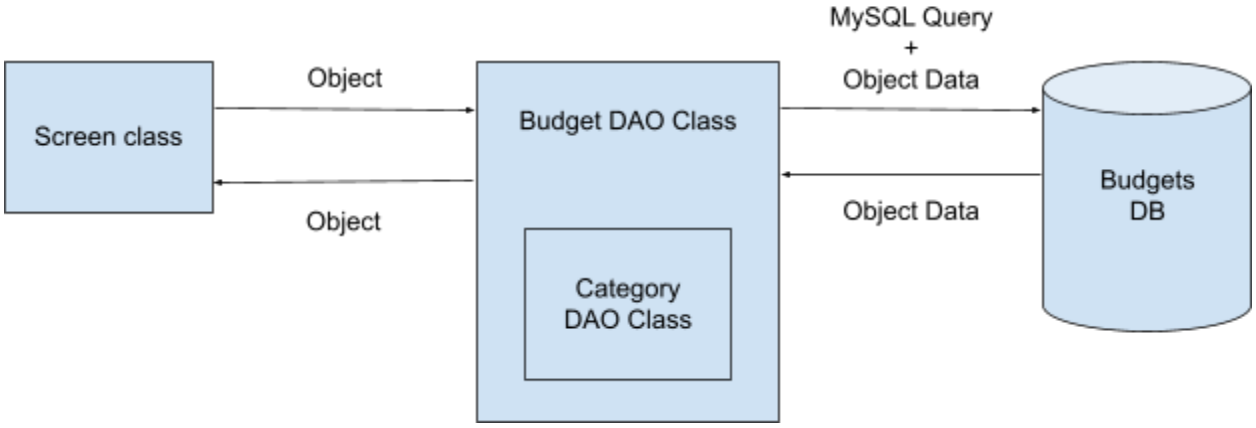
Original GUI Mockups







Architecture



Q/A and Testing

Test Case #	Requirement Tested	Rationale	Input(s)	Expected Output	Pass /Fail
1	1.0.0	A GUI is needed for	Opening the .jar	The loadScreen	P

		the user to interact with their budget data.	executable.	window opens.	
2	1.1.0	A table that displays the user's data is required for navigation and usability.	Program opening	The loadScreen populates with budget objects from the DB if they exist.	P
3	1.2.0	Operations on the object must be attached to a button for the user to interact with	Button click	Operation performed when button is clicked	P
4	1.3.0	If a deletion is initiated a dialogue box prompts the user for confirmation	Delete button clicked	Dialogue box opens	P
11	1.4.1	User inputs must be edecase/unexpected value proof.	A numerical value is entered in the category name field.	The app does not crash and a dialogue box is shown.	P
12	1.4.2	User inputs must be edecase/unexpected value proof.	A string value is entered in the spent TD field	The app does not crash and a dialogue box is shown.	P
13	1.4.3	User inputs must be edecase/unexpected value proof.	A string value is entered in the spend limit field.	The app does not crash and a dialogue box is shown.	P
5	2.0.0	User must be able to input and modify their own categories	Edit Month Button Clicked	The budgetScreen Object is initialized and the GUI window opens	P
6	2.0.1	User must be able to define a name for a category	Text entered into the nameField and Save Changes button clicked	budgetTable values updated to reflect user input	P

7	2.0.2	User must be able to input a value that represents the spend limit for a category	Text entered into the limitField and Save Changes button clicked	budgetTable values updated to reflect user input	P
8	2.0.3	Each category should have an amount spent so far	Text entered into the limitField and Save Changes button clicked	budgetTable values updated to reflect user input	P
14	2.0.4	The spend limit and the spent to date values should have a difference between them that is calculated in real time to show the user how much they have left to spend.	Text entered into the limitField and spentToDateField	remaningField updates with appropriate value	P
9	3.0.0	A DB is required for app data to persist after the app is closed.	Any of the DAO object functions are called	The DB reflects these changes in Workbench AND the data in the GUI is consistent with the DB after an update	P
10	3.1.0	The details of DB access should be hidden from the frontend code.	Any DB query is run directly in a screen class (without using the DAO)	The access fails without the proper credentials found in BudgetDAO	P

Currently known bugs:

Bug Description	Error Caused