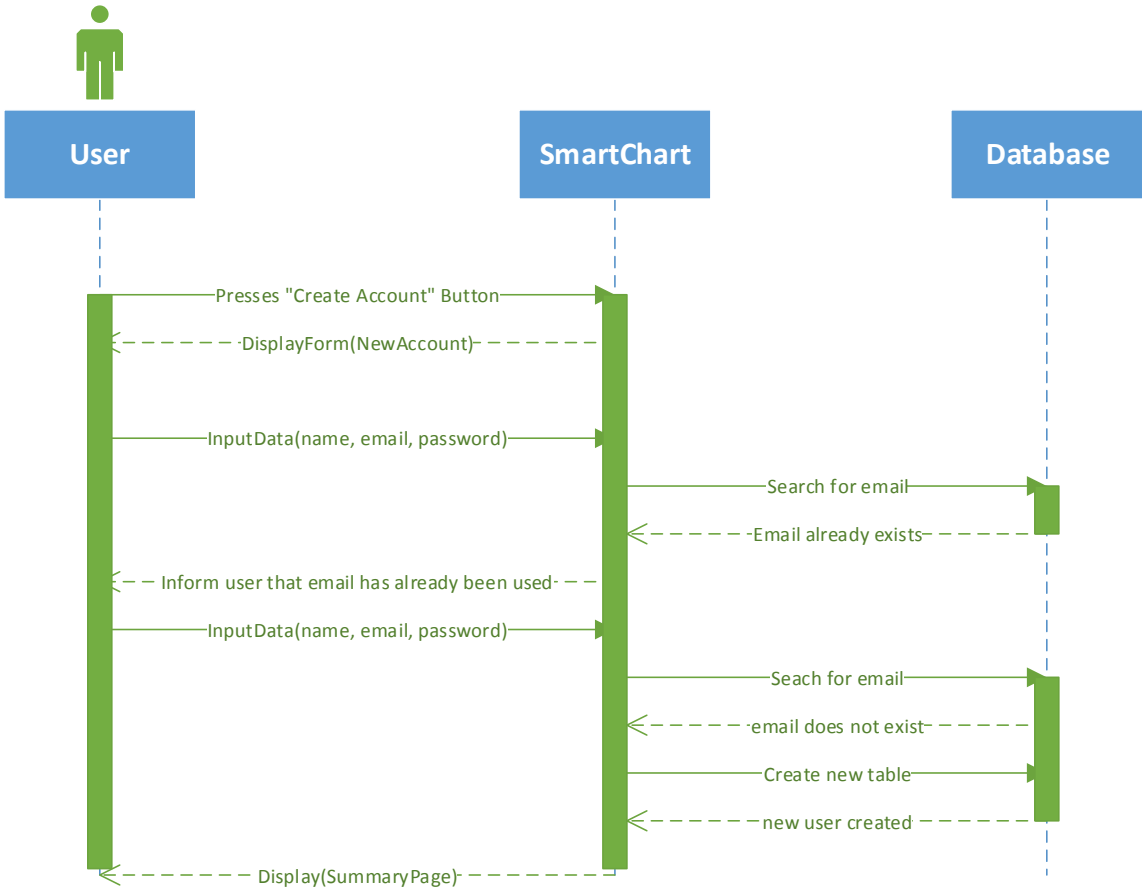
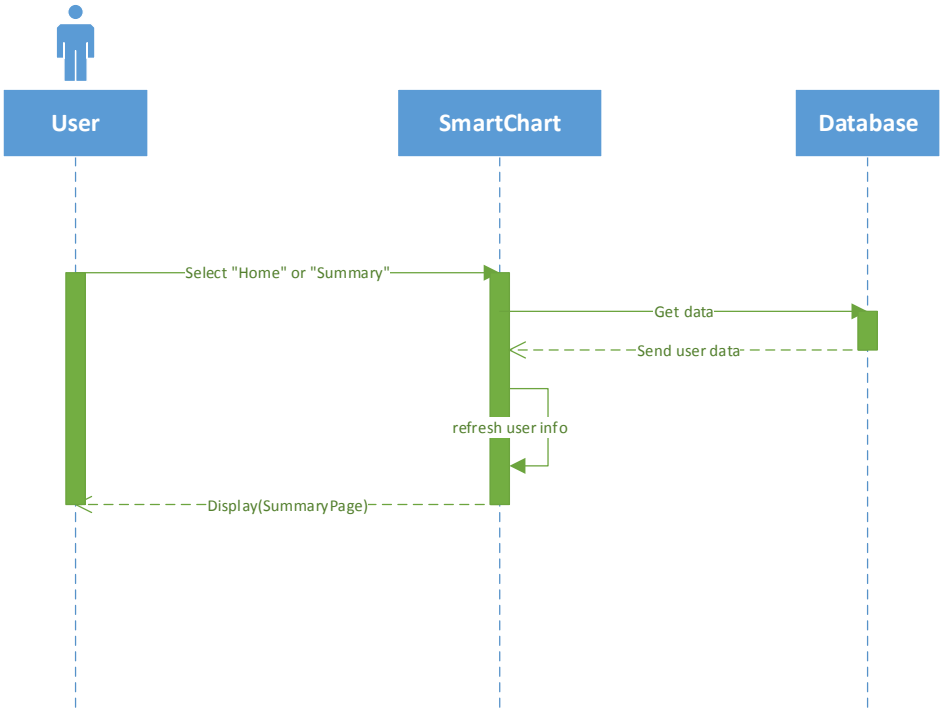
Design Document

Sequence Diagrams

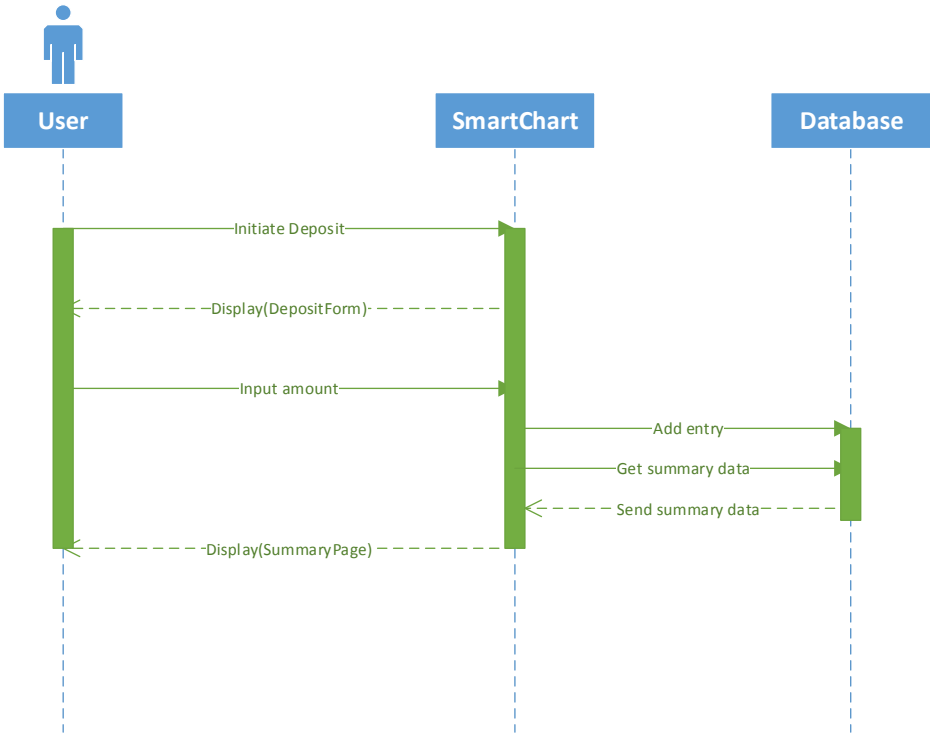
Create Account



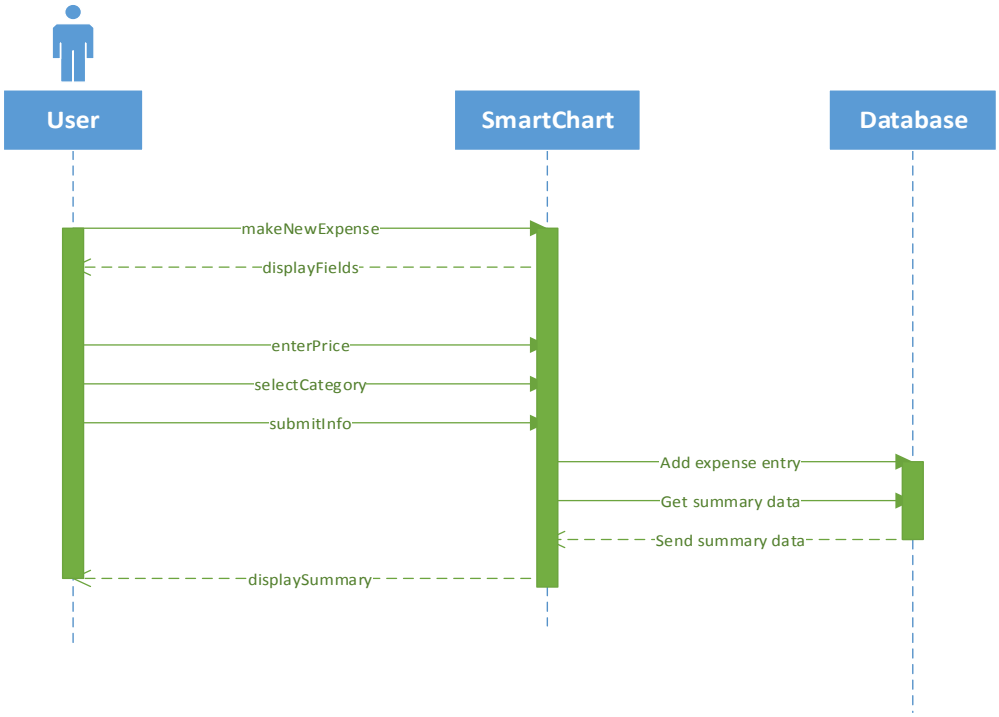
Display Summary Page



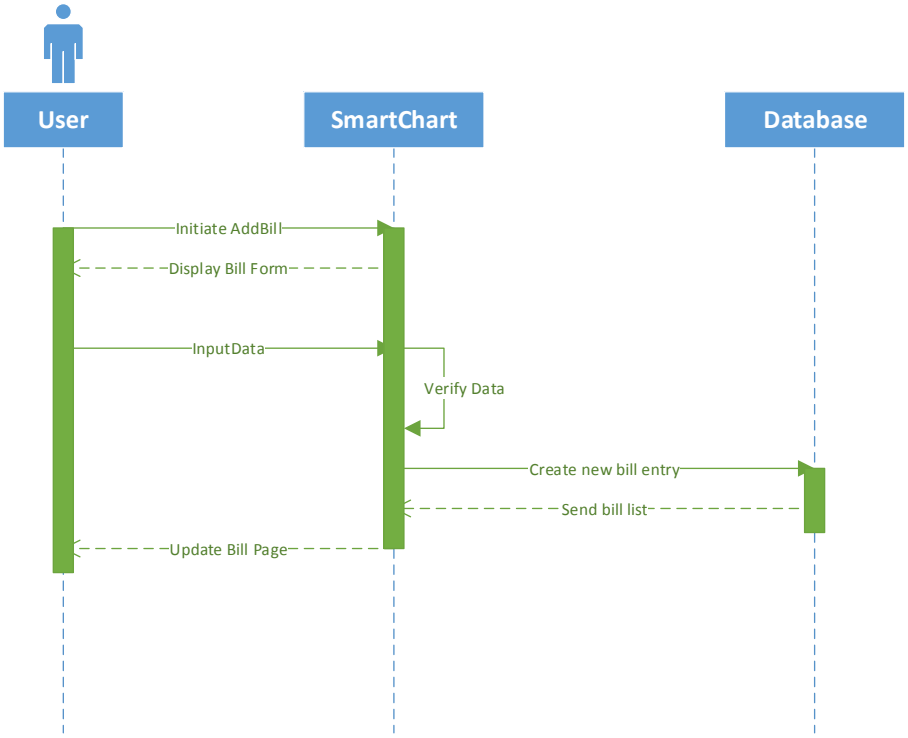
Add Deposit



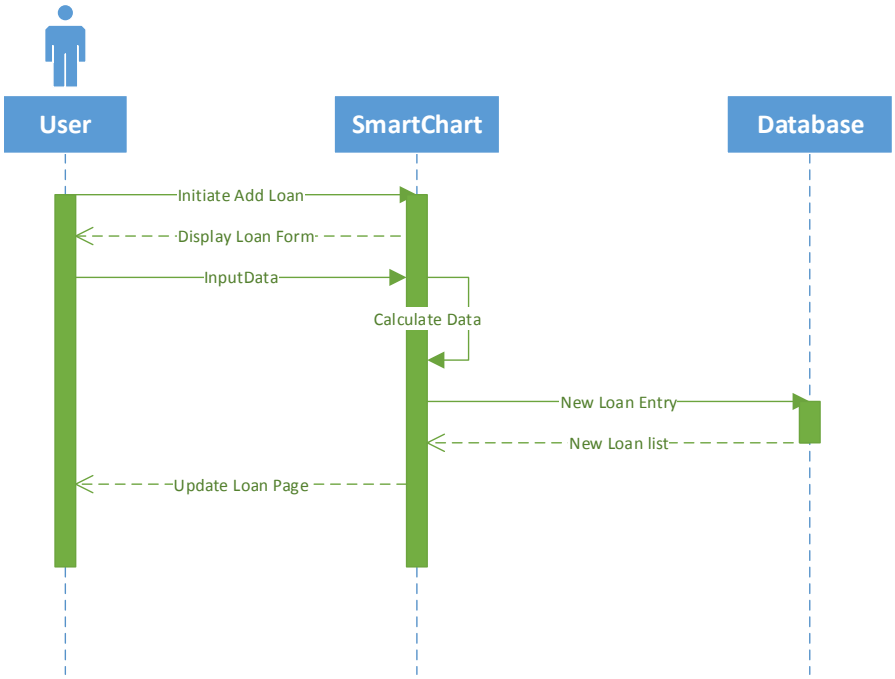
Add Expense



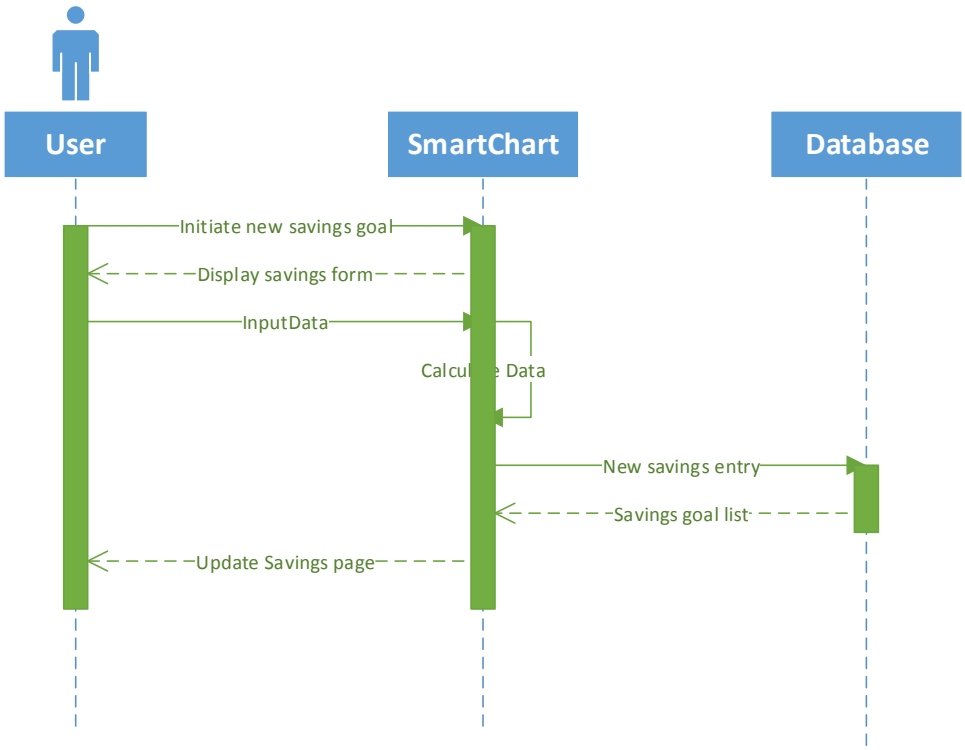
Add Bill



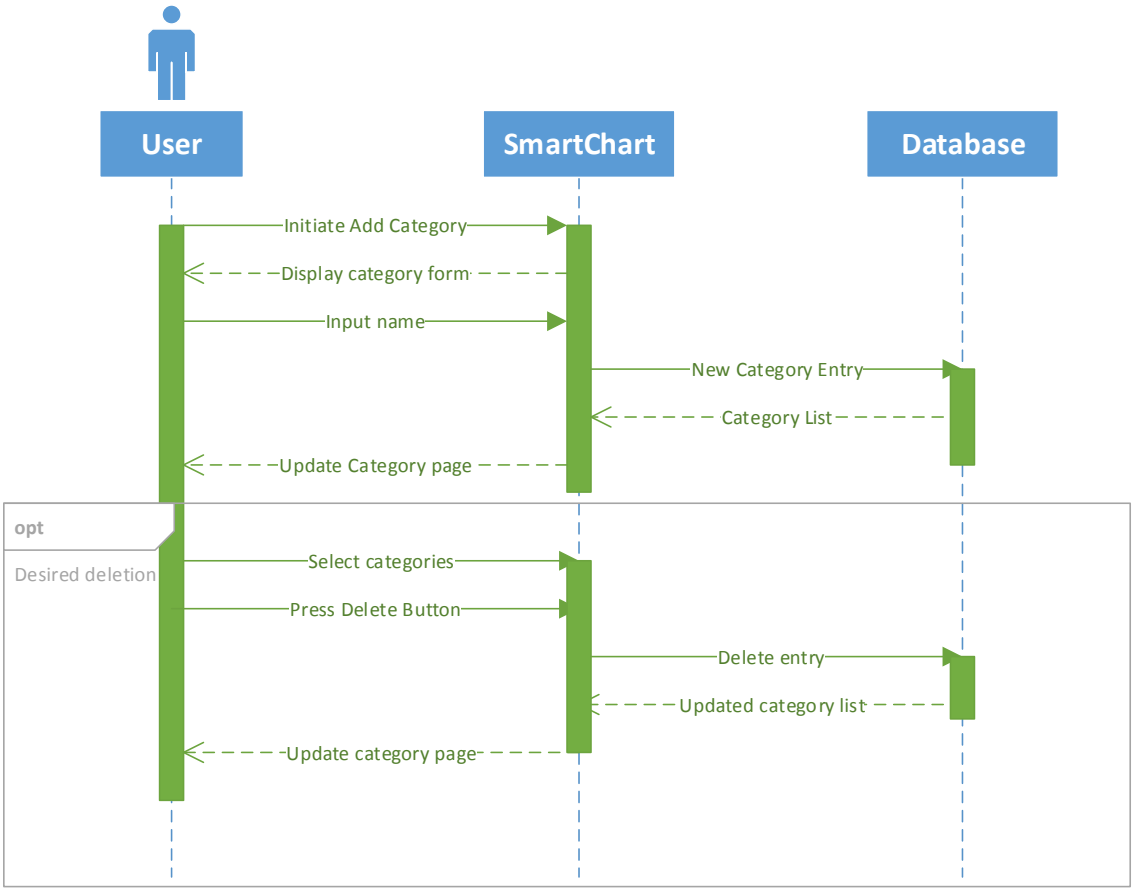
Add Loan



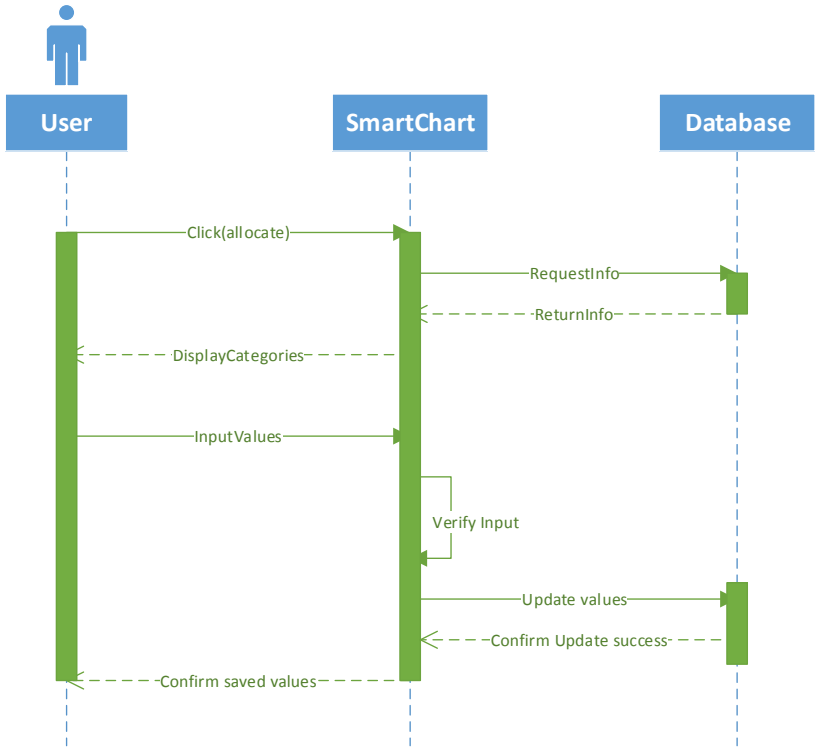
Add Savings Goal



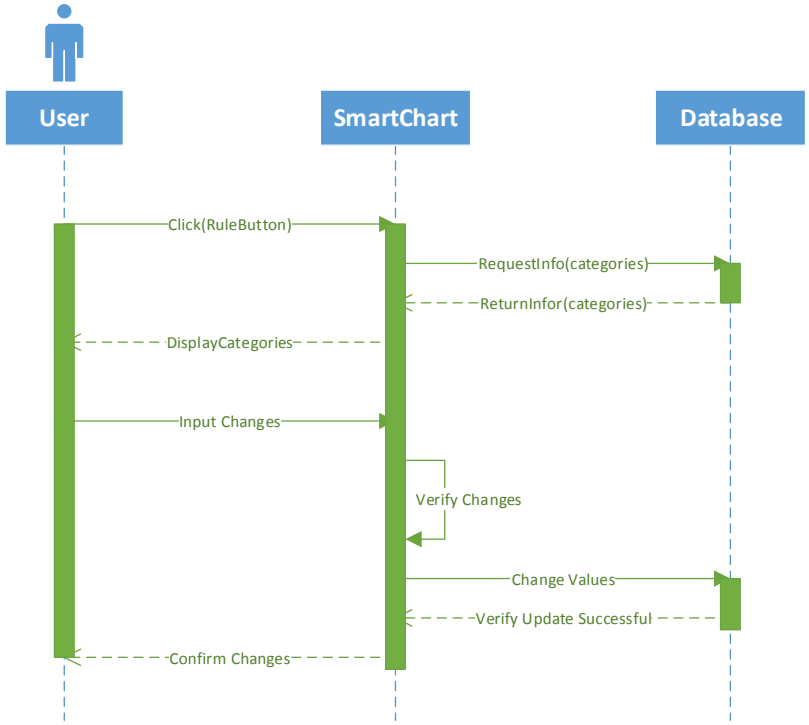
Add/Delete Expense Category



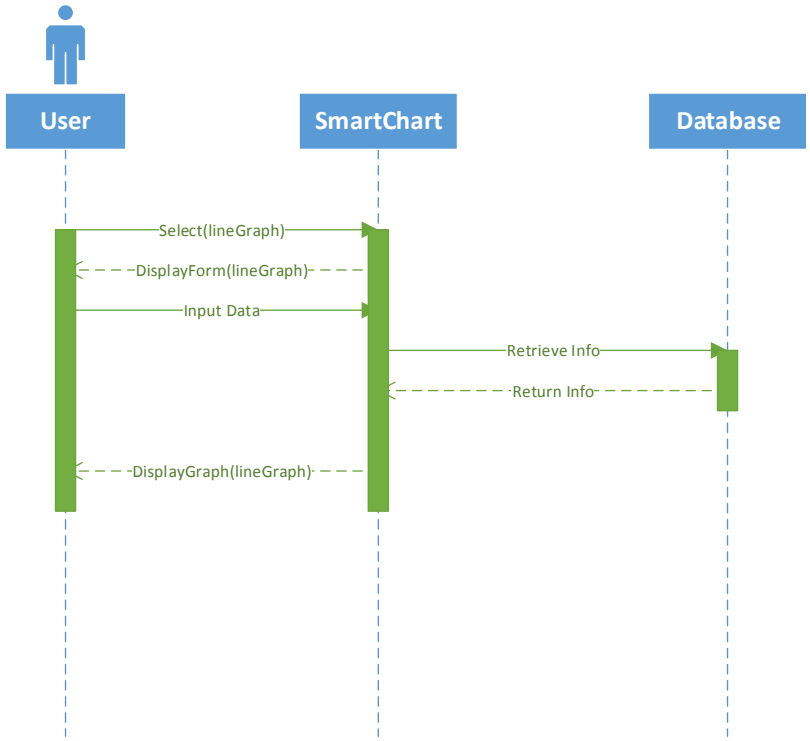
Allocate Funds



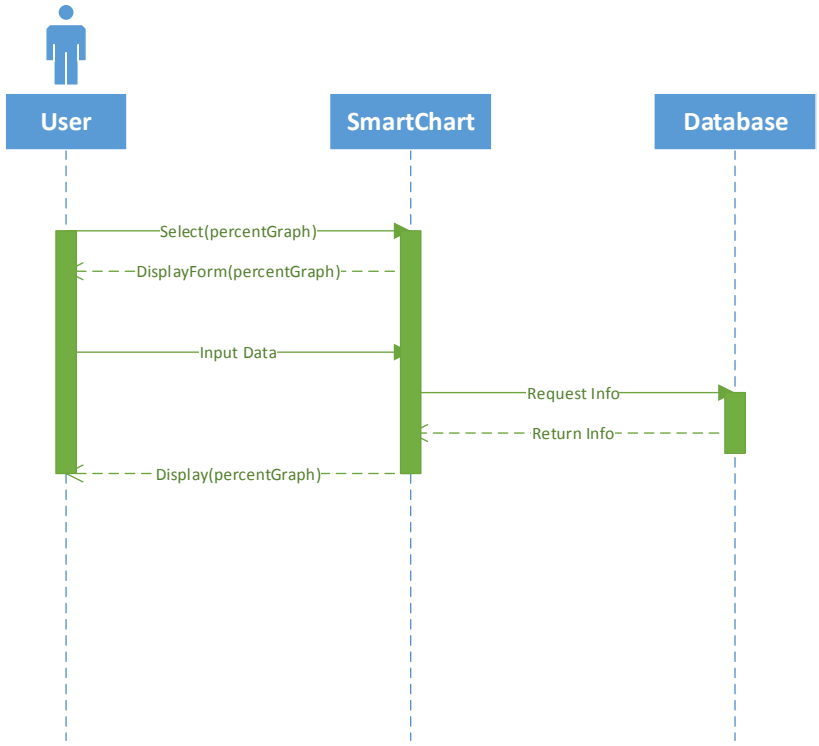
Add/Edit Category Rule



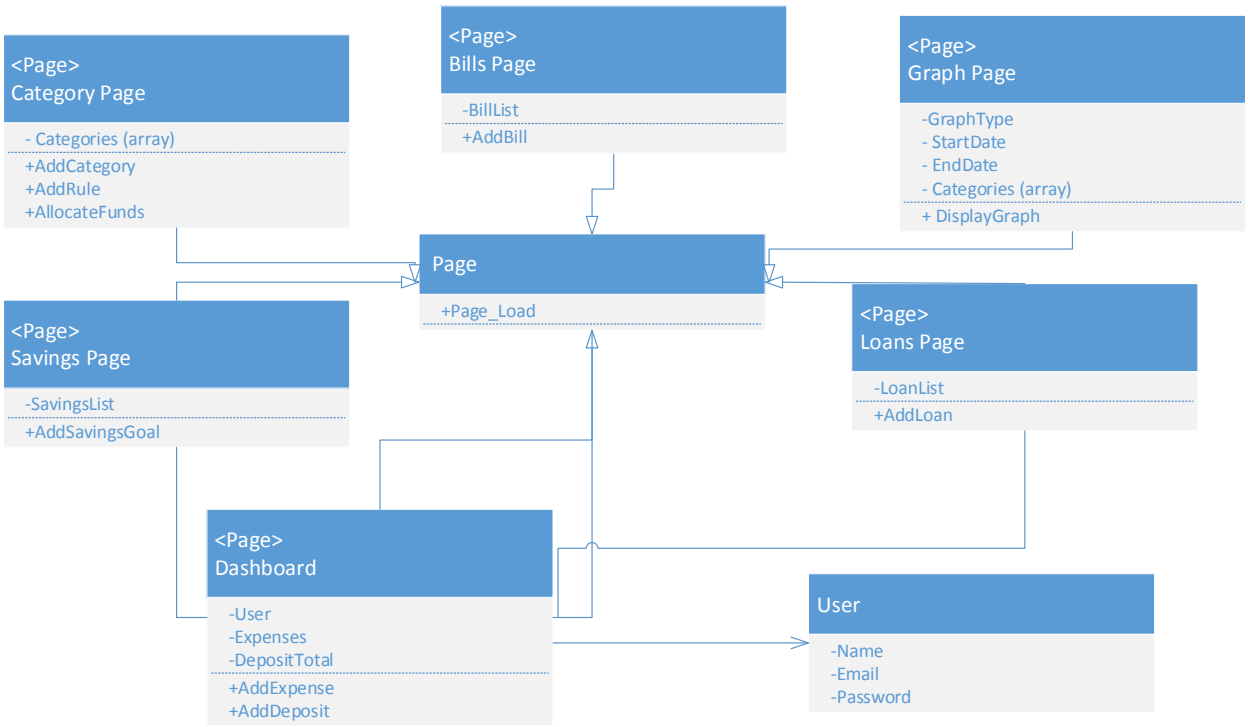
Display Line Graph



Display Percentage Graph



Class Diagram



Design Decisions

For our sequence diagrams, every activity involves the user, the system, and our established database. So when the user wants to record or retrieve information, the database is used to complete these processes. Most activities involve calculations since this is a finance tracking application, so the system will often do several background calculations before affecting the database. Since our is meant to be an interaction solely between the user and the application, each sequence diagram is set up the same way, in terms of actors and systems involved.

As for the class diagram, it was hard to decide how to set it up, since we are using ASP.NET, which generally doesn't use a lot of classes. So we decided to treat each page of our web application as a class, and show the properties of each page in respect to our software activities. There aren't many background objects since each page handles its own activities with calculations and uses Windows forms to display the information.

When it comes to GRASP patterns, it was hard to decide who/what the creator would be, because the only things we are creating are graphs, categories, or updating running totals. For these tasks, the creator would basically whichever page the task was running on. The graphs page would create the necessary graphs and visuals, the categories page would create (or delete) categories if prompted, etc. Most everything else happens in the database, like creating new values in tables.

The information expert would be our database, since that is where we are keeping all records of changes and updates. It keeps track of all allocated totals in each category, history of expenses and deposits, and any other information about the user that is important. The database also helps with low coupling, since it is where all the information is stored. Any time an activity needs to change a value, it stores it in the database. And any time an activity needs to retrieve a value, for instance to display totals on the summary page, it may reference the database instead of storing the information in several places.

The controller would probably be our summary page or "Dashboard." It is the starting place for all tasks and delegates activities to their respective pages. For example, if the user wants to add an expense, the Dashboard then redirects to the expense page. The Dashboard also contains the most current and important information, such as the overall total of the user's finances.

Our system also supports high cohesion since each activity has its own page. Each page is related to the summary page, and when used, it designed to do one task and the results are saved and represented on the Dashboard immediately afterward. This helps break up the complexity and makes the logic easier to understand from a developing perspective, since you know exactly where to find the calculations for a given activity.