Jason Eide, Asif Zaman, Ethan Morris, Jordan Smith, Jose Hernandez

CSCE 3513 Software Engineering Spring 2015

Abstract

This document contains the Architecture Design of a Personal Financial Management System

Project Deliverable

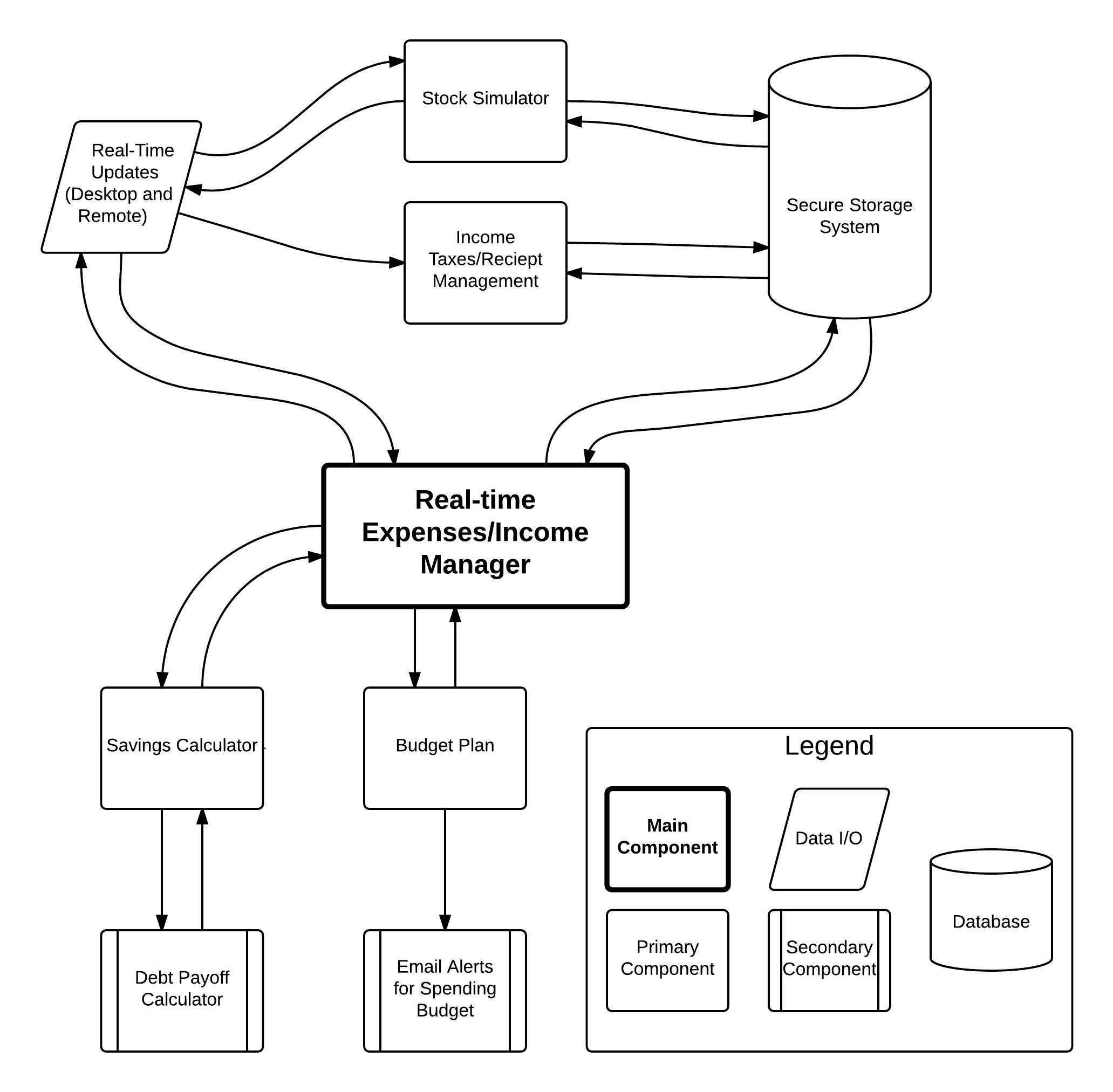
Architecture Design

Group 8

This document describes the architectural design of the Personal Financial Manager System. In the following description we will discuss what each component does, why the necessary components are included, and how the components interact with one another in the design. Also we discuss the importance of cohesion and coupling and show how our design implements these important factors in order to build an efficient software system.

The purpose of our system is to give the user fiscal planning tools and real-time information on their financial status. Users can monitor and evaluate their spending/saving habits during specific (user-set) time intervals (e.g. days, weeks, months, years). For example, if a user desires to see how their spending/saving habits stack up for a particular month, based on their expenses and incomes, they can see exactly what they spent money on in a single, itemized and organized place. It can be scaled down from years to individual days. Additionally, they system can also project finances for an upcoming time interval based on income and known expenses (monthly bills, rent/mortgage, etc.) This is especially useful for users that have a high debt-to-income ratio, in that they can plan their finances by month rather than taking it day by day. Expenses can be any residual cost such as monthly payments, gas, groceries, etc. or any personal cost such as entertainment expenses (dining out, movies, etc.) or non-regular expenses (dentist visit, auto repairs, etc.). Users can also treat saving as a regular, recurring expense in their report as a way to condition the user into adopting good saving habits. These options will for the users will be examined in further detail later in the document. Now that main idea for the software system is in place, we will discuss how the whole system is implemented. In Figure 1 below, you will find the architectural design of the system.

Figure 1: Architecture Design Diagram



From the figure, we can see that the system is designed in such a way that most components do not rely on other components in order for the system to be functional. This is an example of loose coupling. Loose coupling is a good software design practice because more components are independent of each other. This allows a system to be more understandable and if one component were to fail, the system will still work. When describing each component we will show examples of strong cohesion, meaning that the functionality of one component only does what it is required to do and no more. Again, this help with understanding a system much easier.

1. Budget Plan

The purpose of this component is to allow the user to make personal budget plans centered on their current financial status to meet personal financial goals. A user can set his/her own budget and add it to their expense report via the Real-Time Expense/Income Manager (RTEIM) When the new “expense” is implemented in their profile, users can have a better overview of how their expenses/income look like at each end of the month (for example). Also note that once the new expense is entered in the main report, the “Real-Time Expense/Income Manager” will update the database. Here we see that this component has a secondary component that does something entirely different; an example of strong cohesion.

1. Email Alerts for Spending Budget

The purpose of this requirement is to send an email notification to the user when their spending budget is close to reaching its threshold (at user-defined limits) and again when it does reach that threshold. This only interacts with the Budget Plan component. There is no need to interact with the expense report because the “Real-Time Expense/Income Manager” and “Secure Storage System” have already been updated. Users can make changes to their Budget Plan whenever desired.

1. Savings Calculator

If the user desires to save for a big ticket purchase or even simply a savings goal, the user should be able to evaluate how long it will take in order to save the necessary amount of money. For example, if the user wanted to save for a vehicle, the component will give feedback on how much they should put into savings each month order to reach that goal within a certain period of time. The user can split this new cost in weekly/monthly expenses. Again, this new expense is added to the RTEIM and the database is updated from there so the user can monitor their finances with all the relative information.

1. Debt Payoff Calculator

This component mimics the Saving Calculator but instead uses the user’s debt payoff as a goal. The user can choose from a couple of options of how they should approach their debt payoff, e.g. the snowball method (pay off debts from smallest to largest), the avalanche method (pay off debts from largest to smallest), etc. Again, these new expenses can be submitted to the RTEIM and the database is updated. Also this is another example of strong cohesion because the purpose of this component is completely separate from the Savings Calculator.

1. Real-Time Updates(Desktop and Remote)

The purpose on this component is to allow users to login from any location via a web portal or Remote Desktop Application to update their expense/income report. This component interacts with the RTEIM as well as the Stock Simulator and Income Tax/Receipt Management. The interaction with “Expense/Income Management” is simply to update their financial status. The Stock Simulator and the Income Tax Management are both low level priority. Both of these components will require user input in order to make the functionality of those components to work.

1. Stock Simulator

This is a low priority component and it is kept from all the other major components. For this reason, if this component were to ever fail the main functionality will still be functional, an example of loose coupling. This component simply simulates what someone’s money would have looked had they invested the money in the stock market. This component does store the information in the database in order to carry out the simulation over time. However it will not be allowed to update the “Real-Time Expense/Income Management” component.

1. Income Taxes/Receipt Management

This is also a low priority component and it is kept from all the other major components. For this reason, if this component were to ever fail the main functionality will still be functional, an example of loose coupling. The purpose of this component is to allow the user to manage their tax documents. It acts as a repository for easy access to tax documents and receipts. This is why we also interact with the Security Storage System (the database).

1. Secure Storage Device

This is the database component and stores all the relative data for the user such username, password, and the user’s financial information. The data is sent to the necessary components when the user calls a request for the server. These components are the “Real-Time Expense/Income Manager,” “Stock Simulator,” and the “Income Taxes/Receipt Manager.” These interactions are necessary in order to give the user the relative information to evaluate their financial status.

1. Related System Interaction within the Environment

All components will be accessible by the user except for the Secure Storage System. To access these components the user can use a desktop application or web browser which will require a mouse and keyboard or a touch screen device when using the web portal.