

# Requirement Document (D1)

**Team Name: Red Hot**

**Product Name: Prudence**

<http://web.cs.wpi.edu/~cs509/s09/>

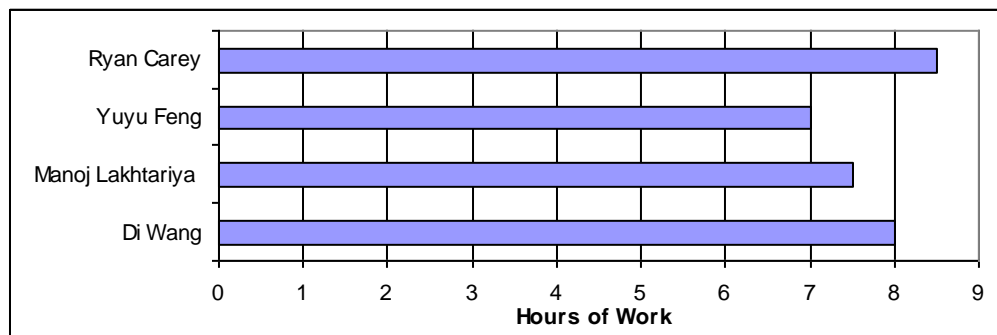
**Leader:** Ryan Carey (careyr@wpi.edu)

**Chief Programmer:** Yuyu Feng (fengyuyu@wpi.edu)

**Software Architect:** Manoj Lakhtariya (manoj\_b4m@yahoo.com)

**Document Specialist:** Di Wang (wangdi@wpi.edu)

## Team Participation:



## Signatures:

Ryan Carey

Manoj Lakhtariya

---

---

Yuyu Feng

Di Wang

---

---

# Table of Contents

1	Description .....	1
2	Glossary .....	1
3	Functional Requirements .....	1
3.1	User Management .....	2
3.2	Account Management .....	4
3.3	Category Management .....	6
3.4	Transaction Management .....	7
3.5	Budget Management .....	9
3.6	View Report .....	10
4	Constraints on the Product .....	13
4.1	Design Constraints .....	13
4.2	Development Cycle Constraints.....	13
5	Product/Development Characteristics.....	14
5.1	Risks.....	14
6	Document History .....	15

# List of Figures

Figure 1	Use Cases Groups .....	2
Figure 2	Use Cases for User Management.....	3
Figure 3	Prototype Register User Screen.....	3
Figure 4	Prototype User Login Screen.....	4
Figure 5	Use Cases for Account Management.....	5
Figure 6	Prototype Account Management Screen.....	5
Figure 7	Use Cases for Category Management .....	6
Figure 8	Prototype Modify Category Screen .....	7
Figure 9	Use Cases for Transaction Management .....	8
Figure 10	Prototype Create Transaction Screen.....	9
Figure 11	Use Cases for Budget Management.....	10
Figure 12	Use Cases for View Report.....	11
Figure 13	Prototype Balance Report Screen .....	12
Figure 14	Prototype Spending vs. Budget Report Screen.....	13
Figure 15	Project Gantt Chart .....	14

# 1 Description

Prudence is a system capable of securely tracking personal transactions for multiple users. Prudence allows each user to add and modify transactions for multiple checking, savings, credit card, and investment accounts. To manage all these transactions Prudence allows the users to tag transactions with customized categories. Users can also create a budget to identify spending limits for categories. Prudence monitors the budget's spending limits and alerts the user when spending has exceeded the limit for a category. The categories are also used by Prudence to generate useful reports to track user spending, and report on spending compared to the budget. The advantage of Prudence over products like Quicken is that Prudence provide a simplified user interface. Prudence also support additional features such as importing from files not supported by Quicken (such as the deprecated QIF file format).

## 2 Glossary

**Account** – A fund maintained by a bank and from which the user/owner can make deposits or withdrawals.

**Advanced** – Advanced is used to identify use cases that provide additional features to Prudence. Advanced features are desired in the final system, but are not required in the final system.

**Category** – A user defined name used to group transactions or portion of transactions.

**Core** – Core is used to identify use cases that provide basic functionality for Prudence. The final system is required to include all Core features.

**Budget** – A set of user defined limits for monthly spending in specified categories. The user defines which categories to include in the budget and the monthly spending limit on each category.

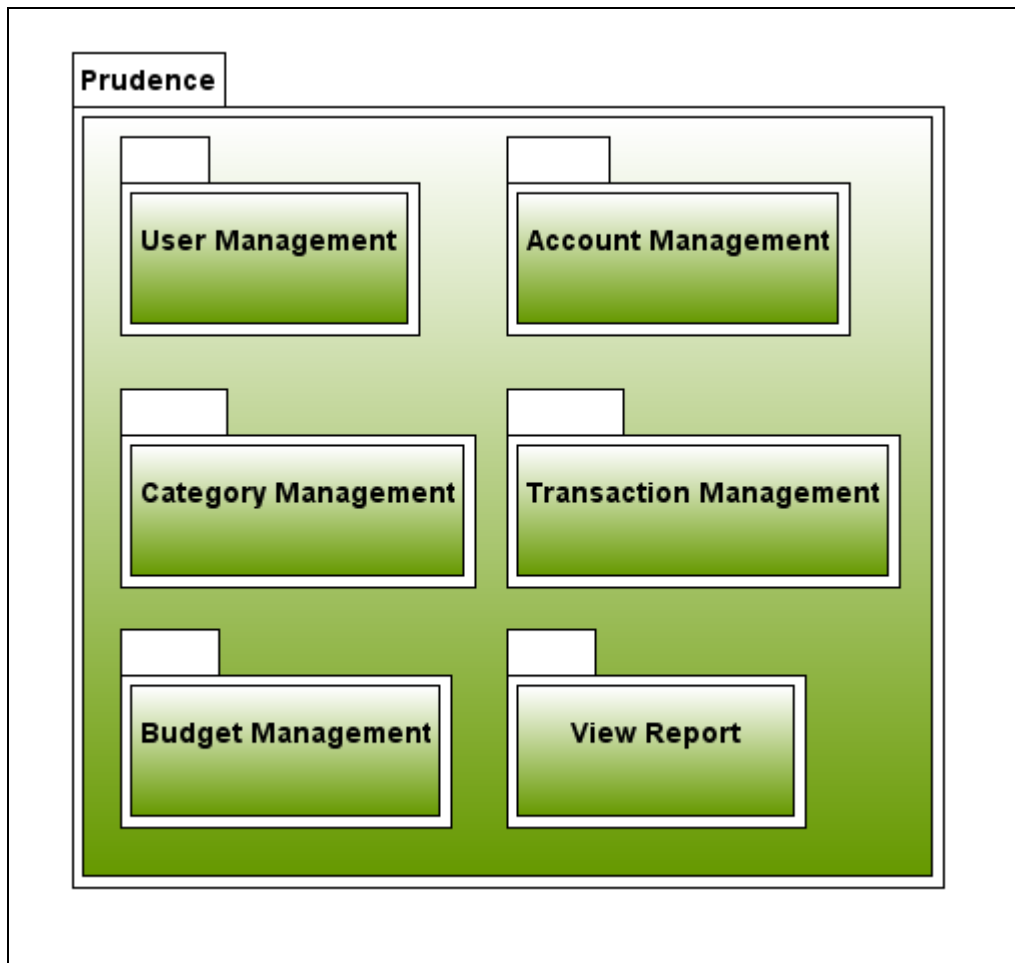
**Party** – An interested group in a financial transaction representing a business, organization or person.

**Security** – A stock, bond, or other investment that a user owns.

**Transaction** – The movement of assets with a monetary value between two parties or two accounts.

## 3 Functional Requirements

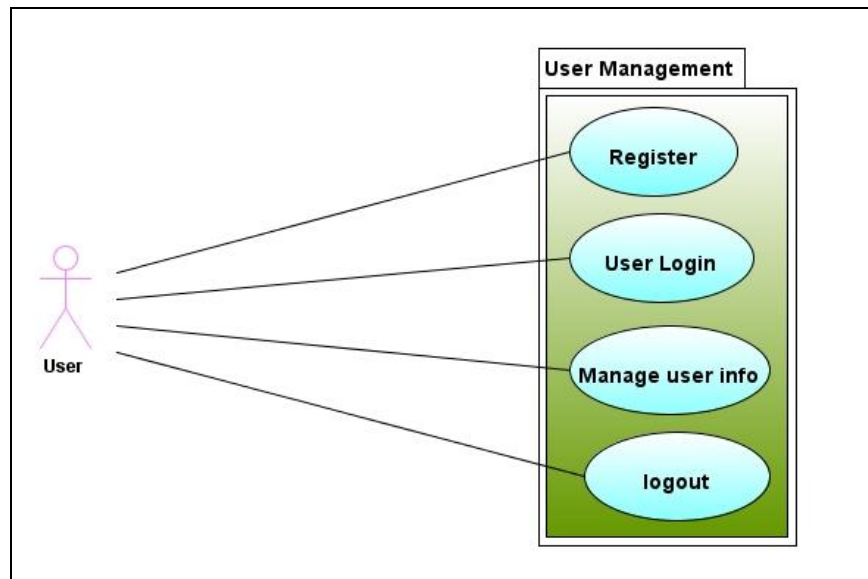
This section describes the functional requirements of Prudence. Requirements are separated into six core features: ManageUsers, Manage Accounts, Manage Categories, Manage Transactions, Manage Budget and View Reports. Figure 1 shows the top level features for Prudence. Each feature of Prudence contains one or more Use Cases described in subsections of the document. The use cases in this document are all marked as Core or Advanced.



*Figure 1 Use Cases Groups*

### **3.1 User Management**

The section describes the functional requirements for registering, authenticating and managing users. The information that Prudence stores about a user (i.e. user, account, transaction, category, and budget information) is private, and can only be access by the user that own the data. Use cases of user management are shown in Figure 2.



*Figure 2 Use Cases for User Management*

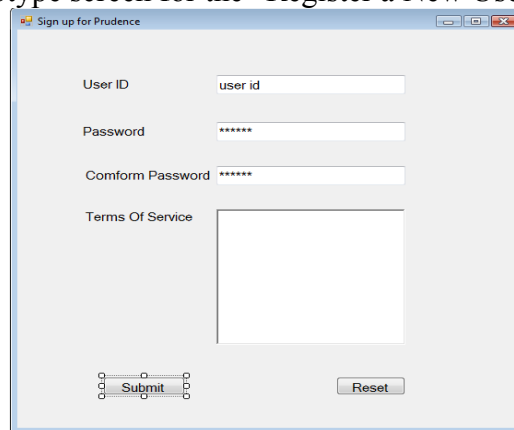
### **3.1.1 (UC01) Register a New User**

**Feature Type: Core**

Before accessing other features of the system, a user must register a new user. New Users can register with Prudence by inputting the following user information:

- Username
- Password
- Confirm Password

Note: Figure 3 provides a prototype screen for the “Register a New User” Use Case.

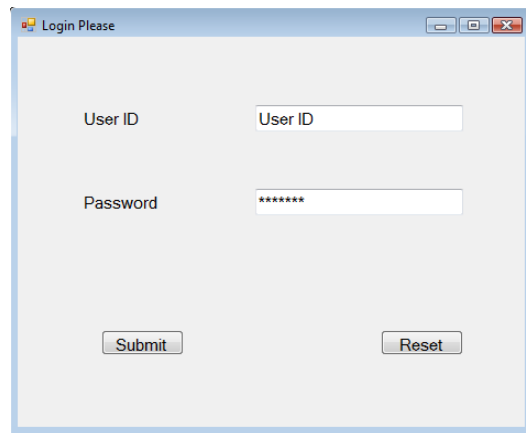


*Figure 3 Prototype Register User Screen*

### **3.1.2 (UC02) Authenticate User**

**Feature Type: Core**

Once registered, a User can login to Prudence using a preset user id and password, as shown in the Figure 4. The user inputs user id and password, and Prudence authenticates the user using the user id and password. If the user id and password is correct, the user is authenticated and allowed to access Prudence. If the user id and password is incorrect, the user is not authenticated and is not allowed to access Prudence.



*Figure 4 Prototype User Login Screen*

### **3.1.3 (UC03) Manage User Information**

**Feature Type: Core**

User can change password, modify user information, or delete the user from Prudence.

### **3.1.4 (UC04) User Logout**

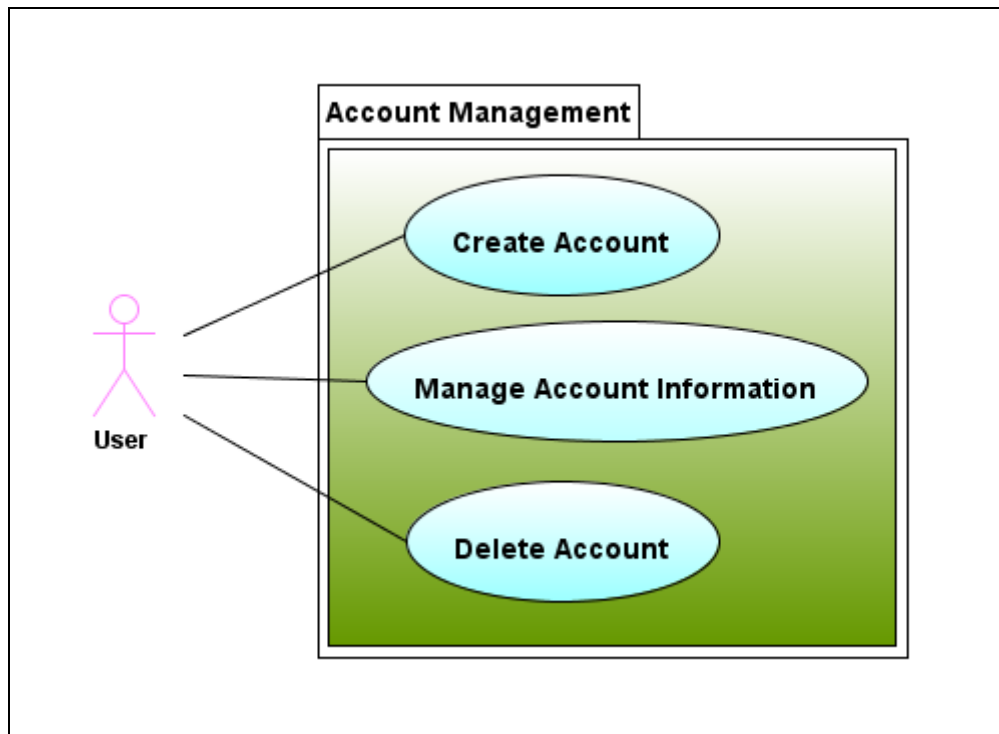
**Feature Type: Core**

When the user has finished viewing reports or modifying data in Prudence, the user logs out. Logging out unauthenticates a user. If the user wishes to access features of Prudence that require an authenticated user after logging out, then the user must once again login to Prudence.

## **3.2 Account Management**

**Feature Type: Core**

Prudence can track the balance of several accounts bounded to the user. The following requirements identify how a user creates, modifies and deletes account data for a bank savings, credit card, or investment account). Use case diagram of user management is shown in Figure 5.



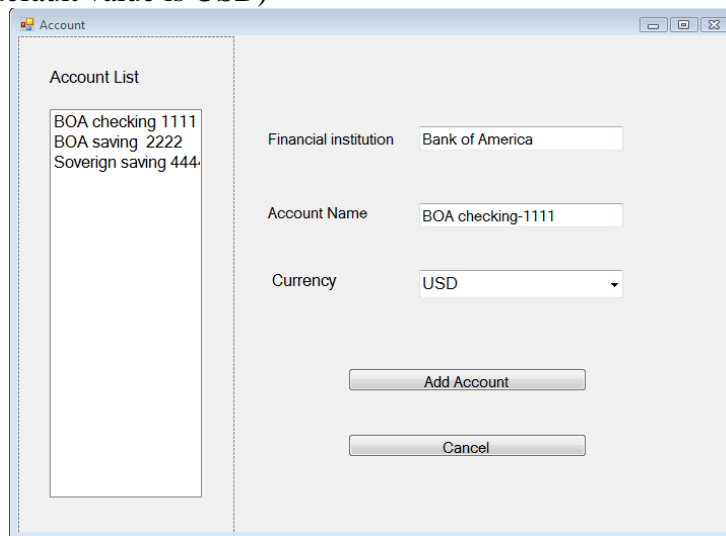
*Figure 5 Use Cases for Account Management*

### 3.2.1 (UC05) Create Account

Feature Type: Core

A user can create an account. The user needs to input the following information, as shown in Figure 6:

- Account Name (Used as the identification of account)
- Account Type (e.g. bank saving account, security account, credit card account, etc.)
- Financial Institution Name
- Currency (the default value is USD)



*Figure 6 Prototype Account Management Screen*

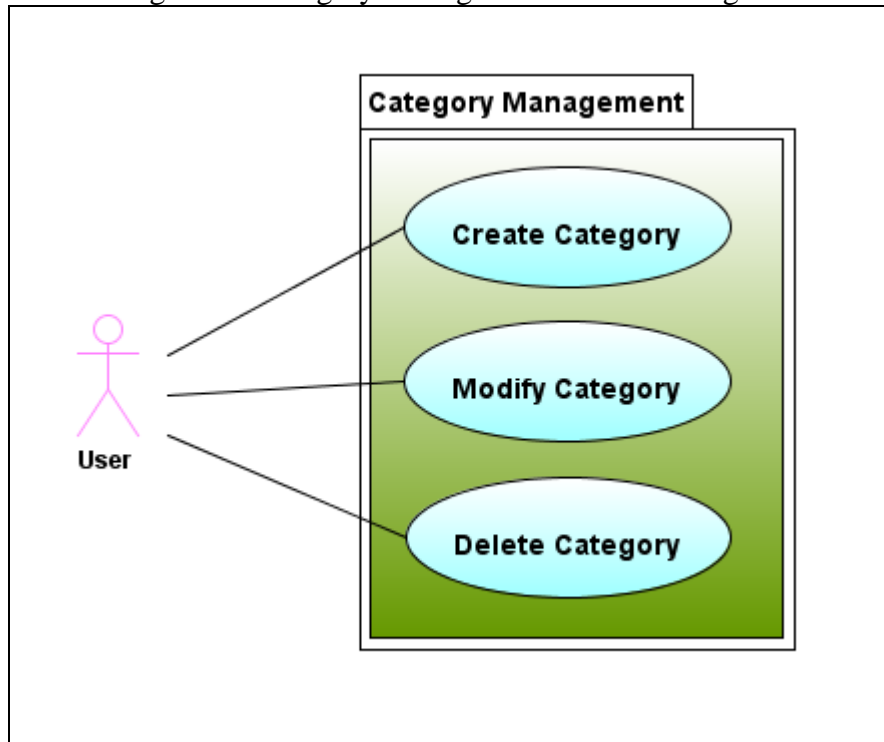
### 3.2.2 (UC06) Manage Account Information

Feature Type: Core

User can modify account type, financial institute name, and currency. A user can delete an account if there are no existing transactions related to the account. If user tries to delete an account while there are transactions related to that account, the software will promote a warning and refuse the deletion.

### 3.3 Category Management

Categories provide a useful mechanism for users to organize transactions. This section describes how to create, delete and modify categories. To understand how to assign Categories to Transactions see section 3.4.2 . Use Case diagram of Category Management is show in Figure 7.



*Figure 7 Use Cases for Category Management*

#### 3.3.1 (UC07) Create Category

**Feature Type: Core**

This function enables user to add category. User input the following information:

- Category Name
- Category Description (Optional)
- Parent Category (Optional)

Note: the 'parent category' attribute allows user to create category hierarchy. For example, category 'Utility' can have child categories such as 'Electricity' and 'Gas'.

#### 3.3.2 (UC08) Delete Category

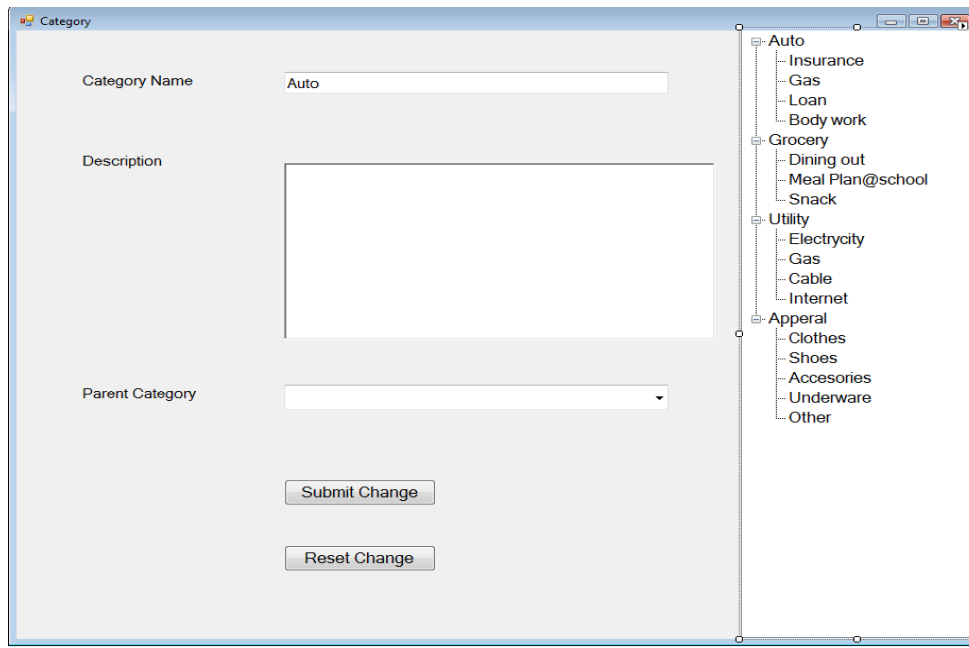
**Feature Type: Core**

User can delete category if the category has no child categories. If the category to delete has child categories, the software product will promote a warning and refuse to delete.

#### 3.3.3 (UC09) Modify Category

**Feature Type: Core**

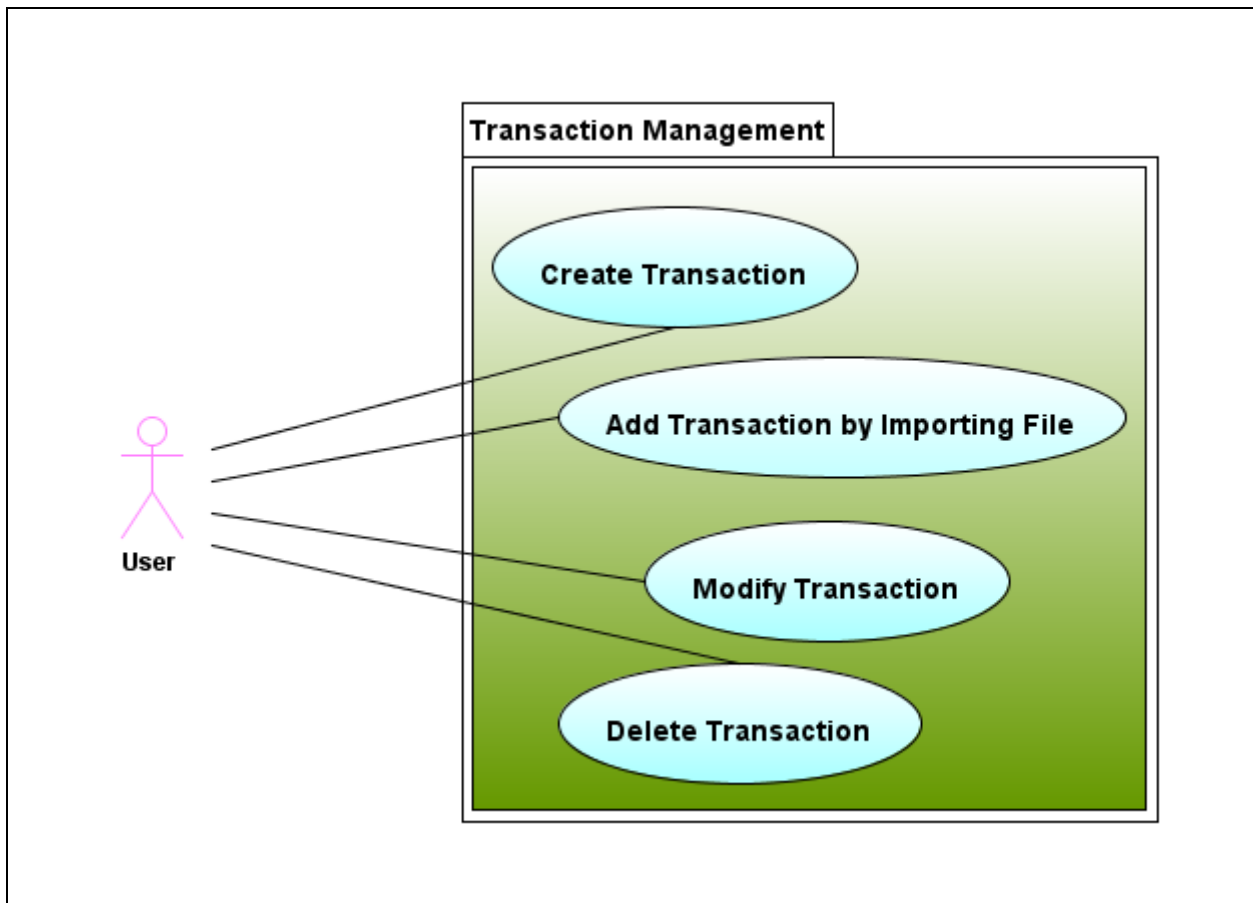
User can modify the category name, category description, and parent category, as shown in Figure 8.



*Figure 8 Prototype Modify Category Screen*

### **3.4 Transaction Management**

The user creates, deletes and modifies transactions in Prudence to track income and expenses. Users must manually enter transaction information into Prudence. Other system can automatically download transaction data from banks and other financial institutions, Prudence doesn't do that right now. Use case diagram of transaction management is show in Figure 9.

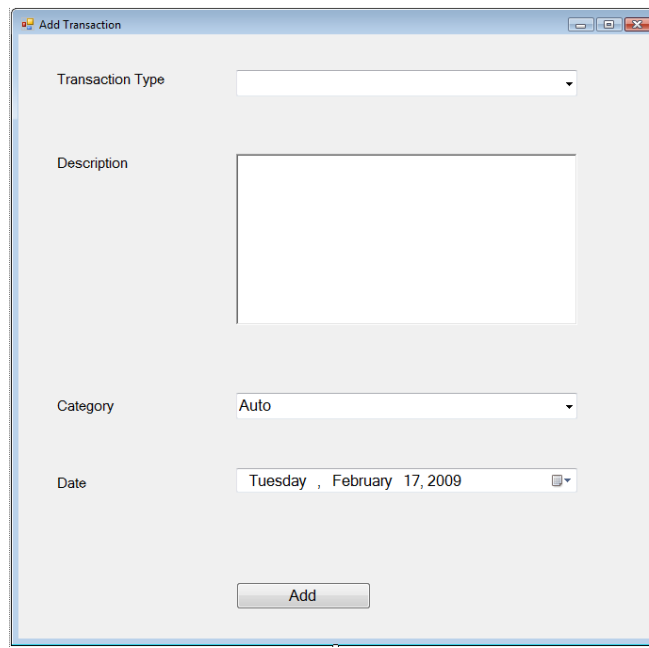


*Figure 9 Use Cases for Transaction Management*

### **3.4.1 (UC10) Create Transaction**

This function enables user to add transactions manually. User input the following information, as shown in the prototype screen in Figure 10:

- Transaction Type ('Expense' or 'Income')
- Description
- Account Name
- Category (Optional)
- Amount
- Date



*Figure 10 Prototype Create Transaction Screen*

### **3.4.2 (UC11) Modify Existing Transaction**

#### **Feature Type: Core**

User can modify an existing transaction by changing its transaction type, description, category, amount and date.

### **3.4.3 (UC12) Delete Transaction**

#### **Feature Type: Core**

User can delete an existing transaction.

### **3.4.4 (UC13) Add Transactions By Importing File**

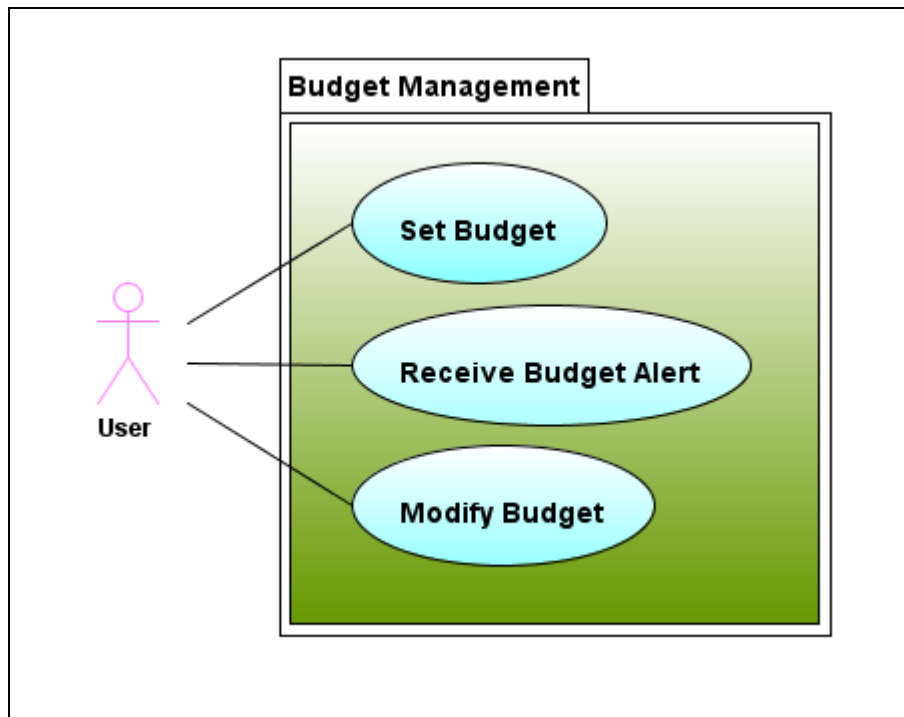
#### **Feature Type: Advanced**

User can import statements downloaded from a bank or credit card website. The statement has to be in the format supported by the software product. After the statement is imported, the software product automatically adds transactions according to the statement. User input the following information:

- Account Name
- File Location

## **3.5 Budget Management**

This section describes the Budget Management function of Prudence. User can set and modify budget information in order to control their expensing. The Prudence system employs the Budget information to generate alert messages and to generates reports (such View Spending vs. Budget in section 3.6.4 ). Use case diagram of budget management is shown in Figure 11.



*Figure 11 Use Cases for Budget Management*

### **3.5.1 (UC14) Create Budget**

**Feature Type: Advanced**

User can create a budget to limit spending in specified categories. User input the following information:

- Budget Name
- Period (Daily, Weekly, Monthly or Yearly; the default value is Monthly)
- Categories
- Limits on Categories

After this use case has completed, Prudence has created a new budget.

### **3.5.2 (UC15) Receive Budget Alert**

**Feature Type: Advanced**

This function enables user to be aware of current expense. If the expense of the categories included in the budget is going to be over the limit amount, the user will receive an alert with this information. The alert will appear on the web page once the user logs in, and can be also sent as email to user.

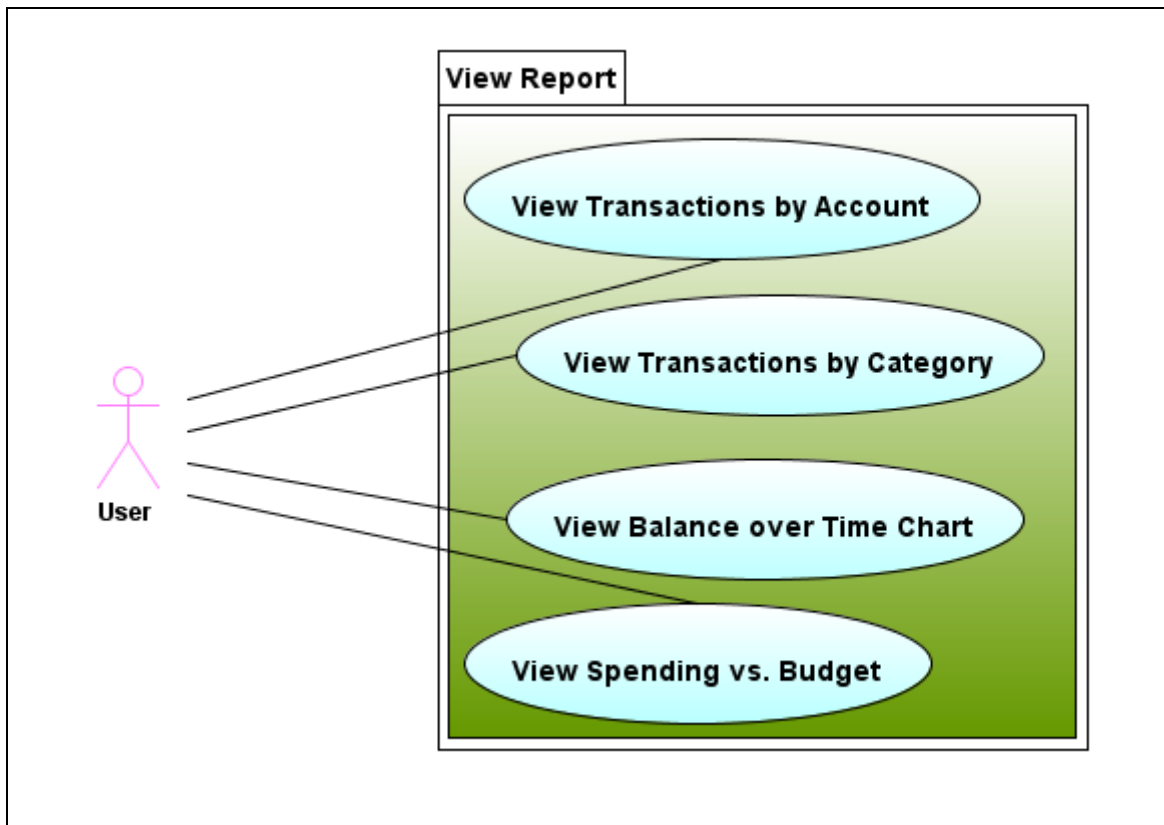
### **3.5.3 (UC16) Modify Budget**

**Feature Type: Advanced**

User can modify the budget name, limit amount, period and categories of existing budget.

## **3.6 View Report**

Once the account, transaction, category, and budget information has been entered into Prudence, the user has several views to choose from for displaying the account, transaction, category, and budget information. Use case diagram is shown in Figure 12.



*Figure 12 Use Cases for View Report*

### **3.6.1 (UC17) View Transactions by Account**

**Feature Type: Core**

User can view the transactions in an account. User provides with the following information:

- Account Name
- Transaction Start Date (Transactions happened after this date will be included)
- Transaction End Date (Transactions happened before this date will be included)

The software product will show transactions as a list, as well as show statistic, such as total income, total expense, and total balance, as shown in the interface diagram below.

### **3.6.2 (UC18) View Transactions by Category**

**Feature Type: Core**

User can view the transactions in a category. User provides with the following information:

- Category Name
- Transaction Start Date (Transactions happened after this date will be included)
- Transaction End Date (Transactions happened before this date will be included)

The software product will show transactions as a list, as well as show statistic, such as total income, total expense, and total balance as shown in the interface diagram below.

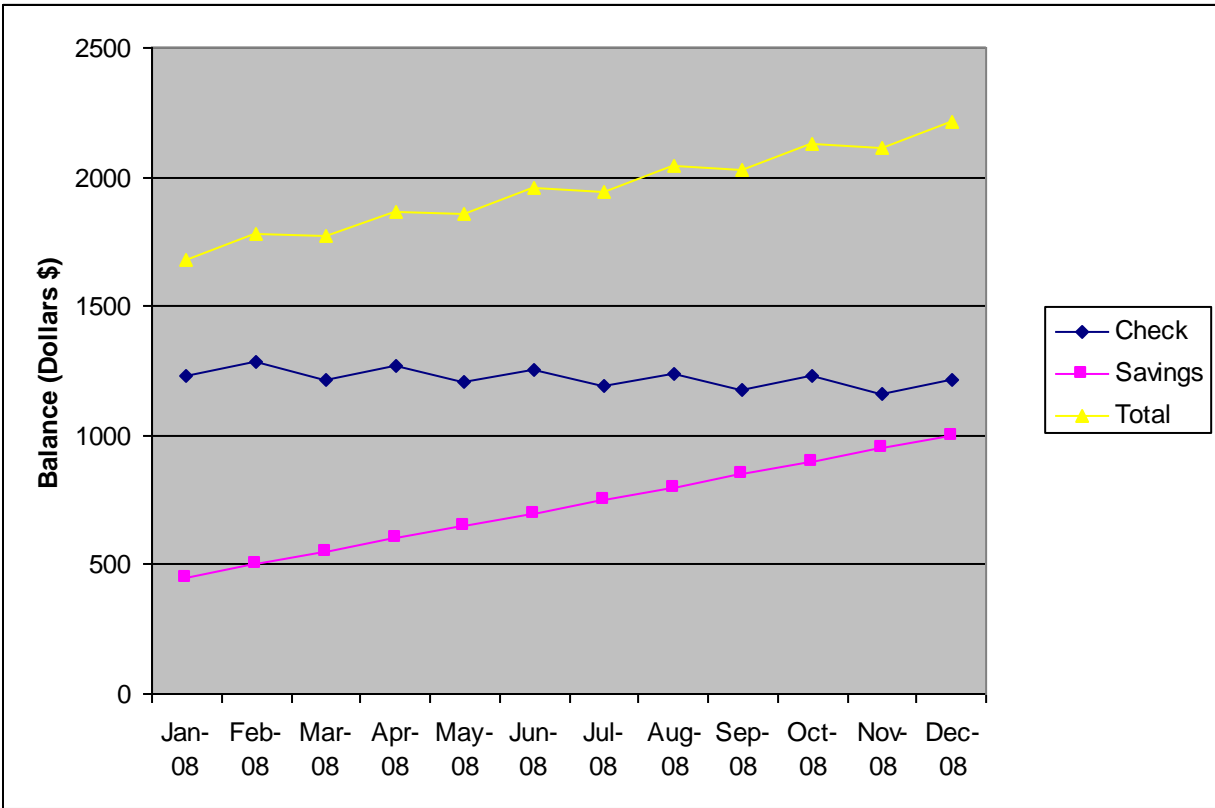
### **3.6.3 (UC19) View Balance over Time Chart**

**Feature Type: Advanced**

User can view the balance of account(s) for a specified period. User provides the following information:

- Start Date (Transactions happened after this date will be included)
- End Date (Transactions happened before this date will be included)

The software product will show the balance of the account(s) over the specified time period. Figure 13 provides a sample of how this View will appear in Prudence.



*Figure 13 Prototype Balance Report Screen*





### **3.6.4 (UC20) View Spending vs. Budget**

**Feature Type: Advanced**

User can view her spending vs. budget. User provides with the following information:

- Budget Name
- Month

The software product will show the budget limit amount and the amount spent for all categories in the budget. Figure 14 provides a prototype screen of how this View will appear in Prudence.

Budgets					
Name	Period	Balance	Spent	Limit	Tags
Auto	Feb 09		485.50	515.00	Auto
Facilities	Feb 09		317.50	150.00	Facilities
Grocery	Feb 09		102.00	150.00	Grocery
Other stuff	Feb 09		151.50	200.00	Other stuff

*Figure 14 Prototype Spending vs. Budget Report Screen*

## 4 Constraints on the Product

The constraints for the system have been separated into two groups. The design constraints describe the non-functional requirements for the system. The Development Cycle Constraints describe the milestones and deadlines for the project.

### 4.1 Design Constraints

This section describes the design constraints on the Prudence System.

- The System shall be written using Java and/or JSP Programming Language.
- The System shall run within a J2EE Application Server.
- The System shall provide an interactive user interface for multiple concurrent users.
- The System shall provide an external SSL connector interface for users to remotely and securely access the system.
- The System shall support at a minimum 3 users each with 10,000 transactions and 50 categories spread over 10 accounts.

### 4.2 Development Cycle Constraints

The final deliver of this project to the customer must be completed on April 30<sup>th</sup>, 2009. Prior to the final deliver date, the System must also pass acceptance testing, and meet all of the milestones outlined in Figure 15. In Figure 15 the project is separated into discrete tasks based on the deliverable artifacts, milestones are highlighted in **Black** or **Orange**, and **Pink** is used to show the completed steps in the development cycle.

ID	Task Name	Start	Finish	% Complete	Feb 2009			Mar 2009			Apr 2009								
					8/2	15/2	22/2	1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4	26/4	3/5		
1	<b>CS509 Project</b>	2/5/2009	4/30/2009	18.64%															
2	<b>Requirements (D1)</b>	2/5/2009	2/26/2009	100%															
3	Requirements Draft	2/5/2009	2/19/2009	100%															
4	Requirements Final	2/19/2009	2/26/2009	100%															
5	<b>Analysis &amp; Design (D2)</b>	2/26/2009	3/19/2009	0%															
6	Analysis Draft	2/26/2009	3/5/2009	0%															
7	Design Draft	3/5/2009	3/12/2009	0%															
8	Analysis & Design Final	3/12/2009	3/19/2009	0%															
9	Manual (D3)	3/19/2009	4/9/2009	0%															
10	Prototype (D4)	3/19/2009	4/16/2009	0%															
11	Testing (D5)	4/16/2009	4/23/2009	0%															
12	Final Project (D6)	4/16/2009	4/30/2009	0%															

**Green Diamonds** mark the beginning and end of parent tasks. Parent tasks contain 2 or more discrete tasks.

**Black diamonds** mark due dates for draft documents.

**Orange diamonds** mark due dates for final documents.

**Blue bars** represent the planned duration for each task.

**Pink bars** represent the amount of the task that is currently complete.

Figure 15 Project Gantt Chart

## 5 Product/Development Characteristics

Everyone on this team is a proficient Java Programmer, so learning a new Programming language will not be a problem. Ryan will need to refresh his understand of JSP; it has been some time since he last needed to create JSP pages. The other Team members will be learning about the Quicken Interchange Format (QIF). QIF files are a common way to export/import transaction data in personal finance software. Ryan has used QIF files before on a product.

Creating the JSP and handling the basic transaction data will be a simple process. This just requires a few JSP pages and database to store the transaction data.

### 5.1 Risks

Three risks have been identified with the Systems development. The risks are Security, the number of core features, and generating Charts.

- Security needs to be a priority for this system. Information about a user's financial transactions can be used to illegally access the user's accounts. It is therefore imperative that the system provide a secure mechanism for the user and the only the user to access his or her data.
- Another risk, is the number features that are included in the requirements. It will be difficult to design, code, test and integrate all the specified features in the prototype within the schedule for this project.
- The last risk is generating the Charts and graphics for the different types of reports. No one on

the team has created a system that automatically generates graphical reports from a web server before.

## **6 Document History**

The following section describes the steps and revisions of this Requirements Document.

The requirements were first written as draft documents by two separate teams of two. Ryan and Manoj were one team. Di and Yuyu formed the other team. Each team independently created a requirements document. The four of us then met to combine the two draft versions into a single document. The four of us spent 2 hours discussing the structure and contents of the combined draft document. After the meeting, each team member was assigned a section of the document to finish. Manoj modeled the Use Case. Di and Yuyu completed the use case descriptions and functionality details. Ryan expanded the description, glossary, constraints, and Product/Development Characteristics sections. Di then combined the sections into a single document, and sent it to the other team member for review and approval. After a few updates the draft version of the Requirements Document was complete.

The last phase of the development of the D1 Requirements Document was to write the final version of the document. The team meet for 1 hour to discuss comments we received from the another Team about our document. Each member of the team continued to work on the sections of the document that were assigned in the draft phase of document development. Di and Yuyu modified the prototype screens. Manoj separated the Use Case diagrams based on feedback from the other team. Ryan added unique identifiers for each Use Case, added text to identify Use Cases as Core or Advanced features, and updated Figure 15 and Section 6 .