**iBudget**

**Software Design**

**Document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Approvals: | L. Assayah | \_\_\_\_\_LA\_\_\_\_\_\_ | date: | \_\_\_\_3/26\_\_\_\_\_ |
|  | C. Leung | \_\_\_\_\_CL\_\_\_\_\_\_ | date: | \_\_\_\_3/26\_\_\_\_\_ |
|  | Q. Pham | \_\_\_\_\_QP\_\_\_\_\_\_ | date: | \_\_\_\_3/26\_\_\_\_\_ |
|  | V. Velev | \_\_\_\_\_VV\_\_\_\_\_\_ | date: | \_\_\_\_3/26\_\_\_\_\_ |
|  | J. Reimels | \_\_\_\_\_JR\_\_\_\_\_\_ | date: | \_\_\_\_3/26\_\_\_\_\_ |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Reason** |
| 3/26/12 | VV, CL, JR, VD, LA, QP | 1.0 | First Draft |
| 05/06/12 | QP | 2.0 | Final Version |

# Introduction

## Purpose

This document describes the design of the *iBudget* personal finance software. It shows how the software system will be structured to satisfy the requirements identiﬁed in the software requirements speciﬁcation.

## Scope

This design is intended for the initial version of *iBudget*. It is intended as the basis for other versions of the software in the future.

## Definitions, acronyms and abbreviations

API: Application Programming Interface – a way for the programmer to interact with the system hardware

DBMS: Database Management System

GUI: Graphical User Interface

## References

[1] Role-playing video game SDD

[2] *Encounter* video game SDD

[3] IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions

# System Architecture

## Architecture design

Top-level decomposition: the application consists of the client side, server side and back-end modules – a typical three tier client-server architecture where one server serves many clients. The top tier is the graphical user interface developed with HTML. The bottom tier is the database layer, developed through the use of MySQL that will hold all transaction, account and category information. The middle layer, or middleware, is developed using PHP, and issues functionality between the user interface and the database such as adding, removing, and editing transactions, accounts and categories.

The code structure is to use the MVC architecture pattern. Use of the MVC pattern results in separating the different aspects of the application (input logic, business logic, and GUI logic), while providing a loose coupling between these elements. The MVC was chosen as it simplifies the architecture by decoupling models and views, and to makes source code more flexible and maintainable and it maps nicely to the three-tier architecture as well.

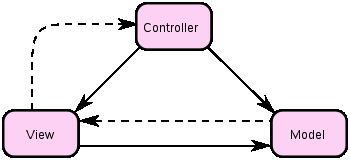


Figure 1 Model–view–controller concept.

The solid line represents a direct association, the dashed an indirect association

(via an observer for example)

The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). The model is not necessarily merely a database; the 'model' in MVC is both the data and the business/domain logic needed to manipulate the data.

The view renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes.

The controller receives user input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and a view port to perform actions based on that input.

## Design pattern

## Factory

The factory pattern creates objects without exposing the instantiation logic to the client and refers to the newly created object through a common interface. The factory pattern is being used... *(please provide examples where and how it is being used)*

## Singleton

The singleton pattern ensures that there is exactly one instance of a given class and that it is accessible from anywhere in the application. The singleton pattern is being used in the *iBudget* application to restrict the number of database connections to only one.

Indeed it is used with all the source objects that are in the source folder. For all these files, the constructor is made private and only called once to create the object the one time. All other attempts to create another instance of the same object will result with getting the instance created before. A static attribute called $instance is created when calling for the first time the getSource() method. Then every time the method is called again, the same instance is returned. Below is an example of how it is implemented in categorysource.inc :

Attribute:

**private static** $instance = **null**;

Method:

**public static function** *getSource* () {

**if** (**self**::*$instance* == **null**) {

**self**::*$instance* = **new CategorySource();**

}

}

Constructor:

**private function** \_\_construct () {

**/\*content \*/**

}

And an example of how it is used in category.inc by calling the getSource() method:

**protected function** delete () {

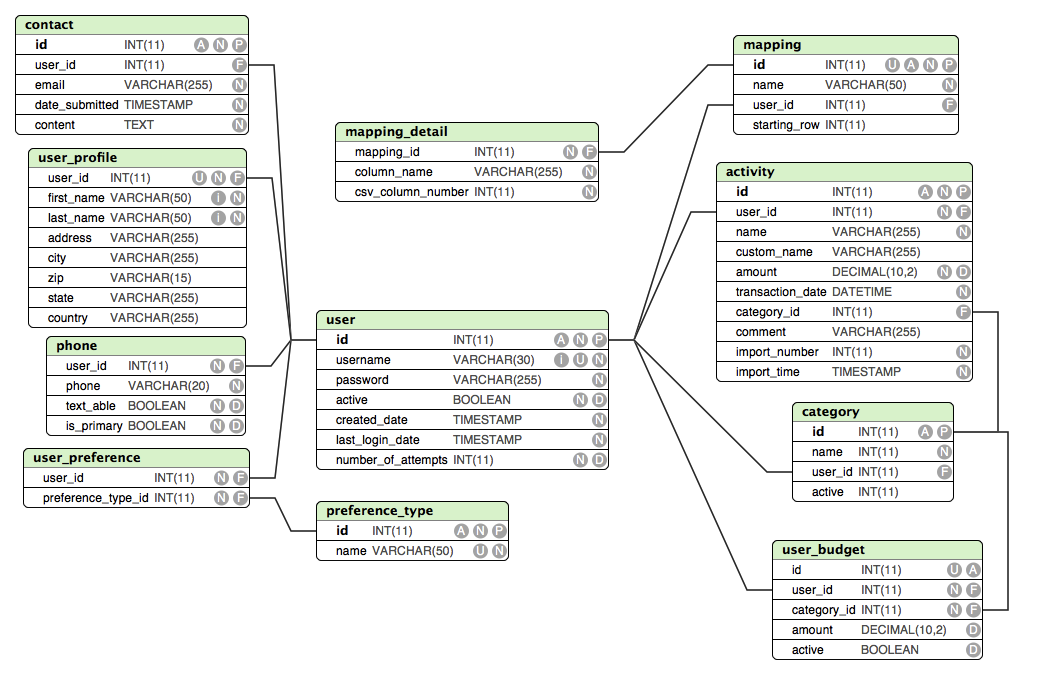
$source = CategorySource::getSource();

$source->delete(**array**('ID' => **$this**->id));

}

# Data diagram

## Data description



Where the following keys are being used:

A = Auto Incremement

N = NOT NULL

P = Primary Key

U = Unique

i = Indexed

D = Has default value assigned

F = Foreign Key

## Data dictionary

## 3.2.1 Source objects

The Source objects are used to access the database and execute SQL statements to modify or get information from the database. Here are the description of the different source objects used in the project:

* **activitySource**: can access the activity table
* **budgetSource**: can access the user\_budget table
* **categorySource**: can access the category table
* **commonSource**: is the class inherited by all the other source classes. It contains methods that can be used by all source objects: deleteFromTableById($table, $id, $id\_column) that enables to delete a row based on the id and getLastInsertedId() that returns the id of the last row inserted in the table.
* **ContactSource**: can access the contact table
* **mappingDetailSource**: can access the mapping\_detail table
* **mappingSource**: can access the mapping table
* **phoneSource**: can access the phone table
* **preferenceTypeSource**: can access the preference\_type table
* **userPreferenceSource**: can access the user\_preference table
* **userSource**: can access the user table.

## 3.2.2 General methods of the source objects

Since all source objects implement the Source interface, they can all define and use the following four methods:

**- read()**: select elements in the database tables. The fields and tables are different depending on the purpose of the class. Ex: get the user information, the budget information, the category information etc…

The SQL statement is an SELECT statement.

**- insert():** insert rows in the database. Each field that can be inserted in the corresponding table is checked to see if it is given a value or not with the following statement:

array\_key\_exists('NAME', $params)with NAME the name of a field in the Category table and $params the entered parameters.

All parameters are put in an array that is then used to complete the INSERT statement.

**- update():** updates rows in the database. The original values of the rows are stocked in an array to check if the value will be changed. Each field that can be updated in the corresponding table is checked to see if it is given value or not. Those two steps are illustrated with the following:

array\_key\_exists('NAME', $params) && $original['name'] != $params['NAME'] with NAME the name of a field in the Category table, $params the entered parameters, $original the previous values of the row.

The SQL statement is an UPDATE statement.

**- delete()**: deletes rows in the database. Deleting rows is done by using the id field in most of the tables. If an id is given, then the following function is used to delete the corresponding row:

**$this**->deleteFromTableById('category', $params['ID']); with category the name of the table and $params the entered parameters.

The SQL statement is a DELETE statement.

# Human interface design

## Registration

## Overview

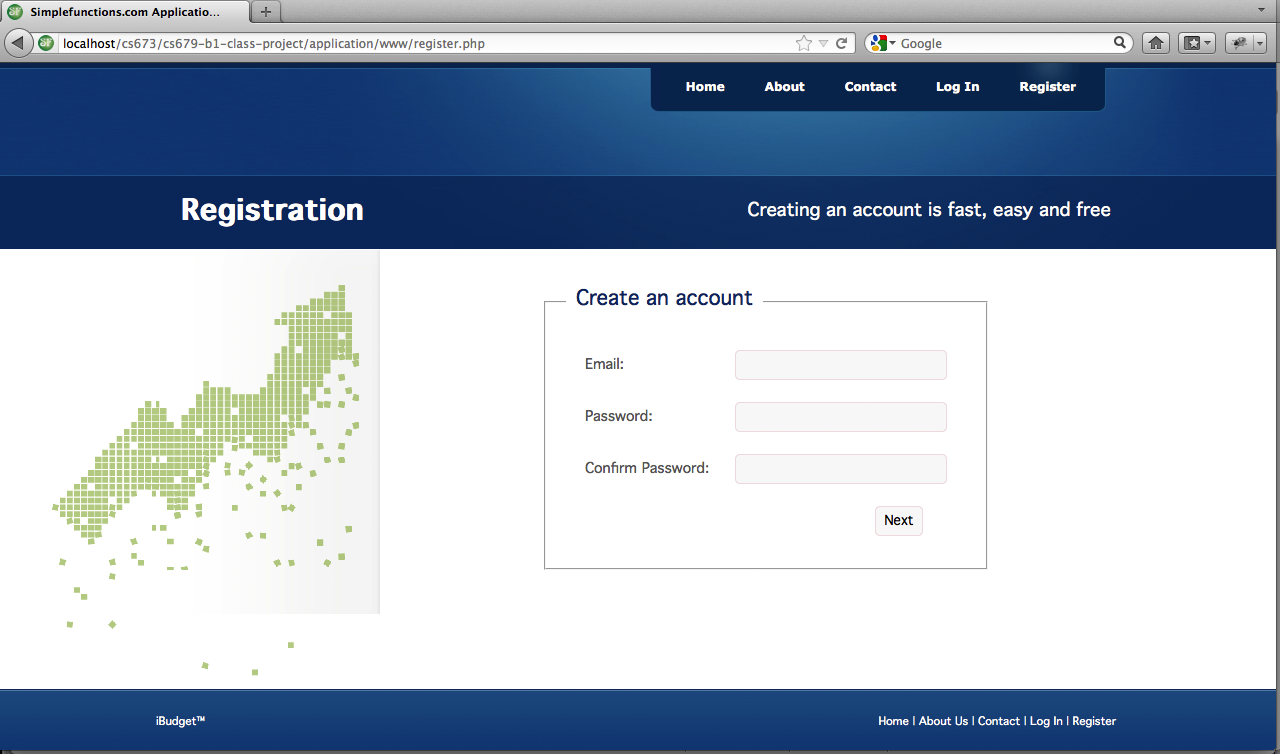
Registration is where a user can create an iBudget account. Users will be prompted to enter their email address (username), a password, their first name, and their last name.

Email and password will be prompted for first and they will be validated dynamically using AJAX to make sure that there is not an account already associated with the email address and that the password meets our requirements.

The user will then click “Next”, the two passwords entered will be validated to ensure they match, and then the user will be prompted to enter their first and last names.

Lastly the user will click “Submit”, and the users account will be created, the user will be logged in, and then redirected to the dashboard.

## Screenshot



## State diagram



## Log In

## Overview

The log in screen is where the user can log in to a previously registered account.

The user will be prompted for their email (username) and password. The username and password will be validated, if they match the user will be logged in and redirected to the Dashboard. If the username or password are not valid, the user will see an error message and again be prompted to enter an email and password.

## A description...Screenshot

## State Diagram

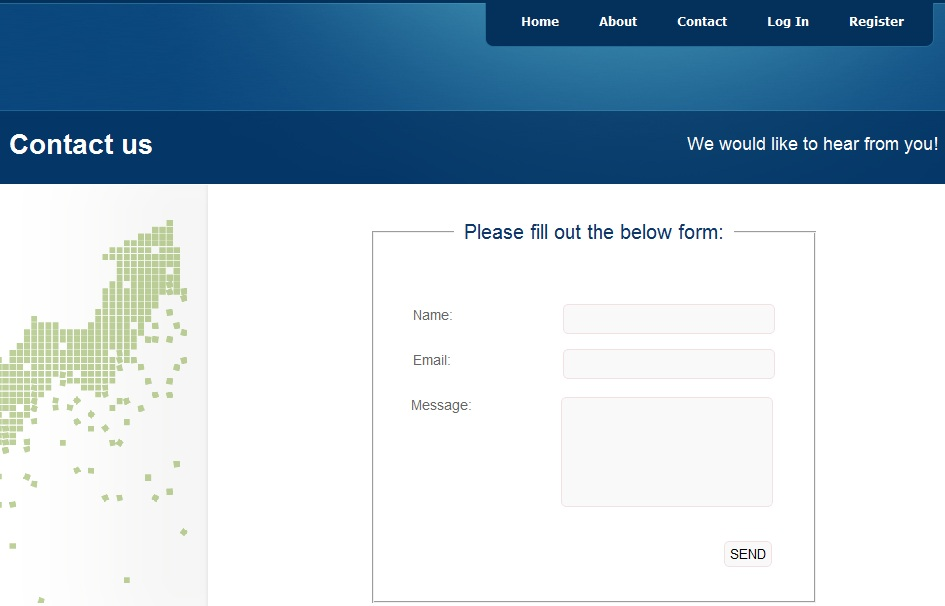


## Contact

## Overview

The contact page is where both registered and non-registered users can leave a message for the iBudget development team. Users need to complete all three fields on the form before the form result can be sent. A client side JavaScript will check and display a message on top of the form to remind users if they leave any fields blank. Once submitted, the similar server side check is carried out. If everything is checked out, an email will be sent to the iBudget team email address. Form data will also be stored in the Contact table of the backend database along with a user ID. The system will check to see if the form is filled out by a currently logged on user or a registered user, who has not logged on, by searching for the provided email in the database or user ID will be anonymous. Error messages will be displayed if anything happens during the submission process. A successful submission will direct the user to the blank Contact page.

## Screenshot



## State diagram

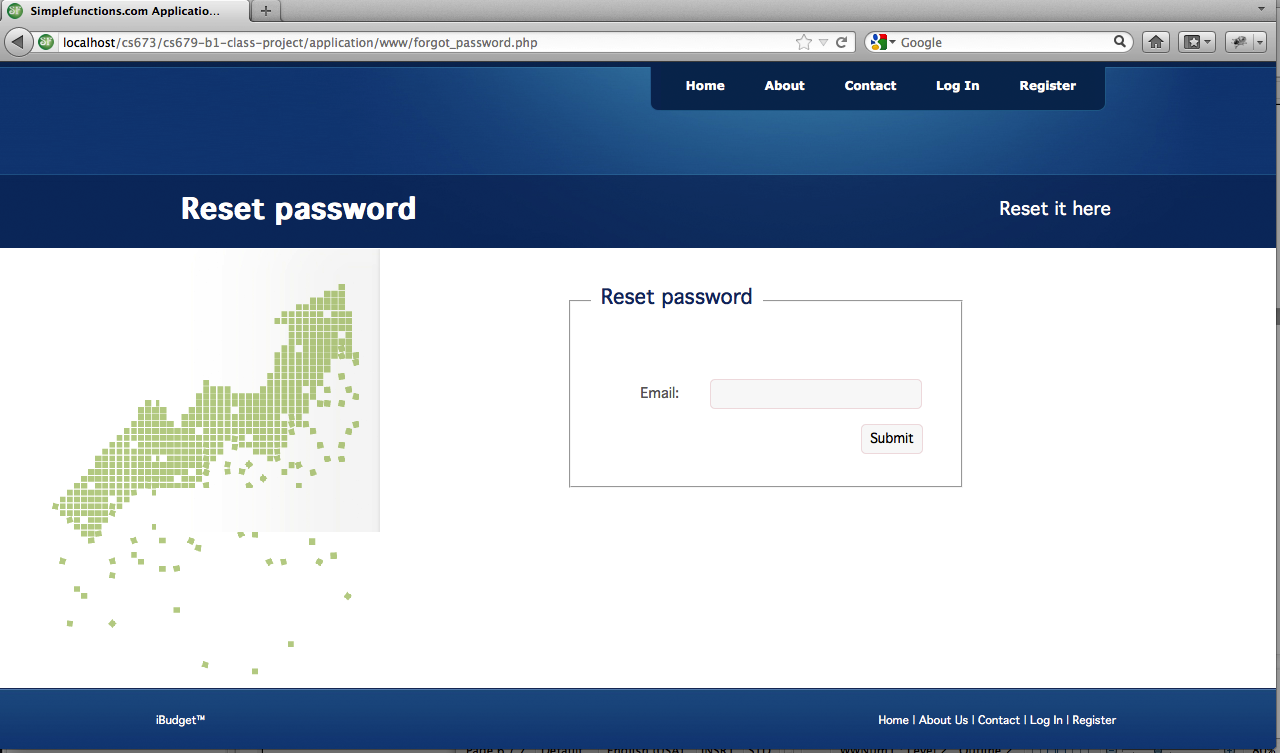


## Forgot password

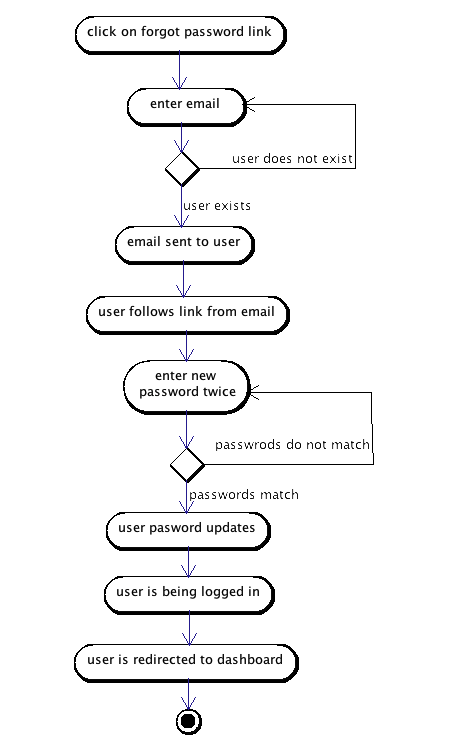
## Overview

If the user forgets his/her password they can reset it by clicking on the 'Forgot password' link in the login page. They are taken to the reset password page (see screenshot below). They enter the email with which they've registered at the site. In case no account with this email exists, the system displays an error message 'User with this username was not found'. An email is sent out to the user with a link they have to follow and they are taken to the reset password screen where they can enter a new password that is being checked to meet the following condition: a password must be at least 8 characters long, and have one lowercase, one uppercase, one number, and one special character. In addition, the 'password' and 'repeat password' fields must match. Once they hit the submit button, the password field in the database is updated, they are logged into the site and being taken to the dashboard page.

## Screenshot



## 4.4.3 State diagram



## Dashboard Overview

## Overview

## Once logged in, the user is directed to the dashboard overview page. This page presents a summary of the user’s financial health by displaying the total credit, debit and account balance as well as a column chart.

## Screenshot

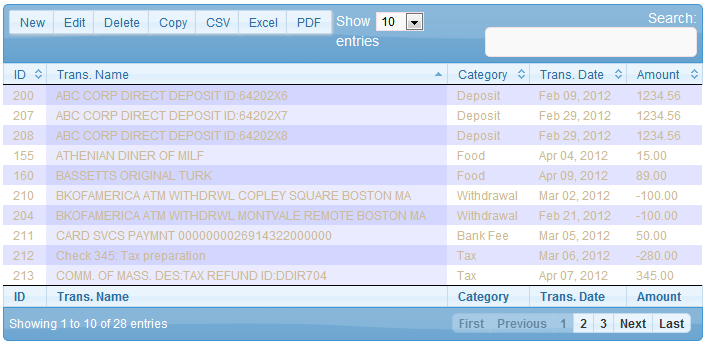


## Transaction View

## Overview

## This view displays all imported data from various banks. On this page, user can categorize each transaction. User can manually create, edit, or remove a transaction. For further manipulation, user can copy the whole transaction list to clipboard and paste it into any program of their choice or they can export the list to a CSV, Excel or PDF format. On this page, user can sort and search as well as defining the lenfth of the list they want to show on screen at a time.

## Screenshot



## 4.6.3 State diagram

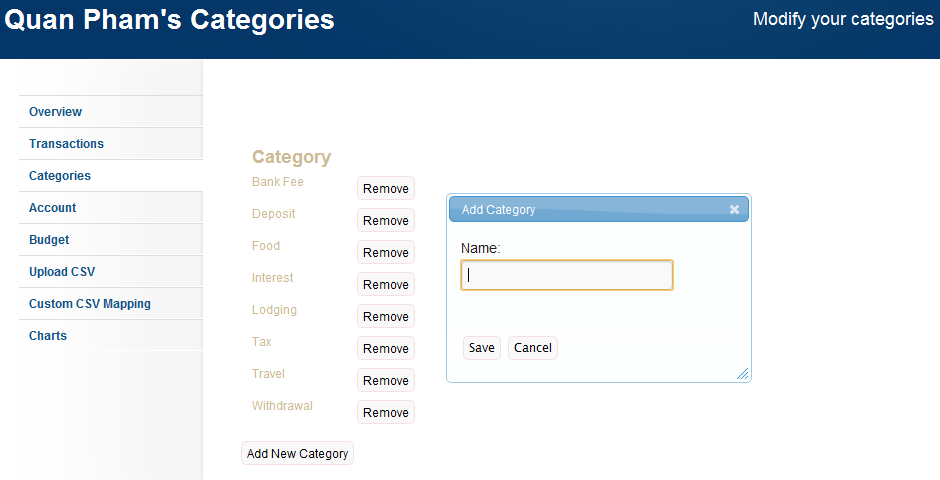


## Category View

## Overview

## Adding cateogy allow users the flexibility to specify their own categories that make the most sense to them. Users are given complete control on adding or removing categories in their own list.

## Screenshot

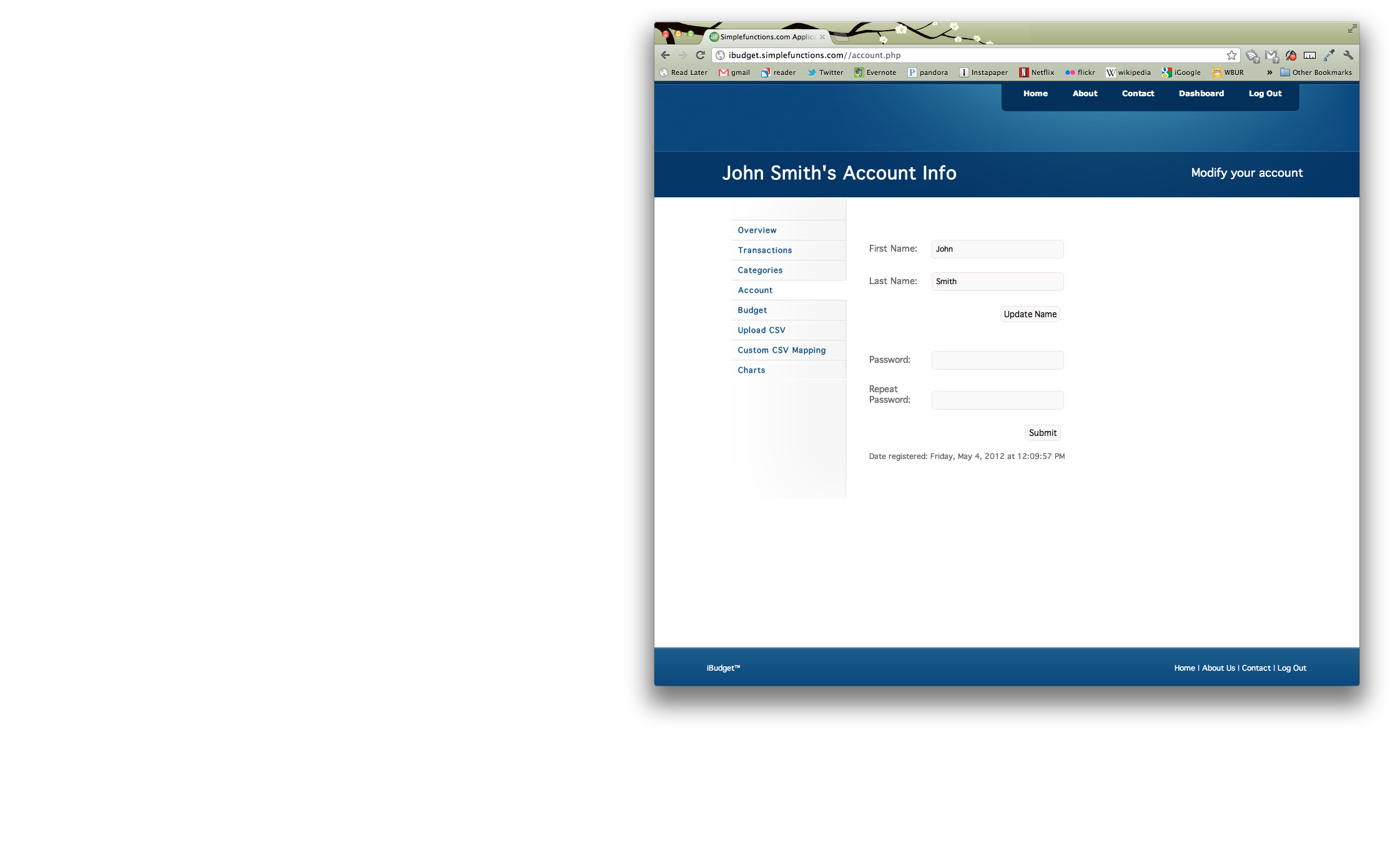


## Account View

## Overview

## User can change their password or their name anytime they want by visiting the account page.

## Screenshot

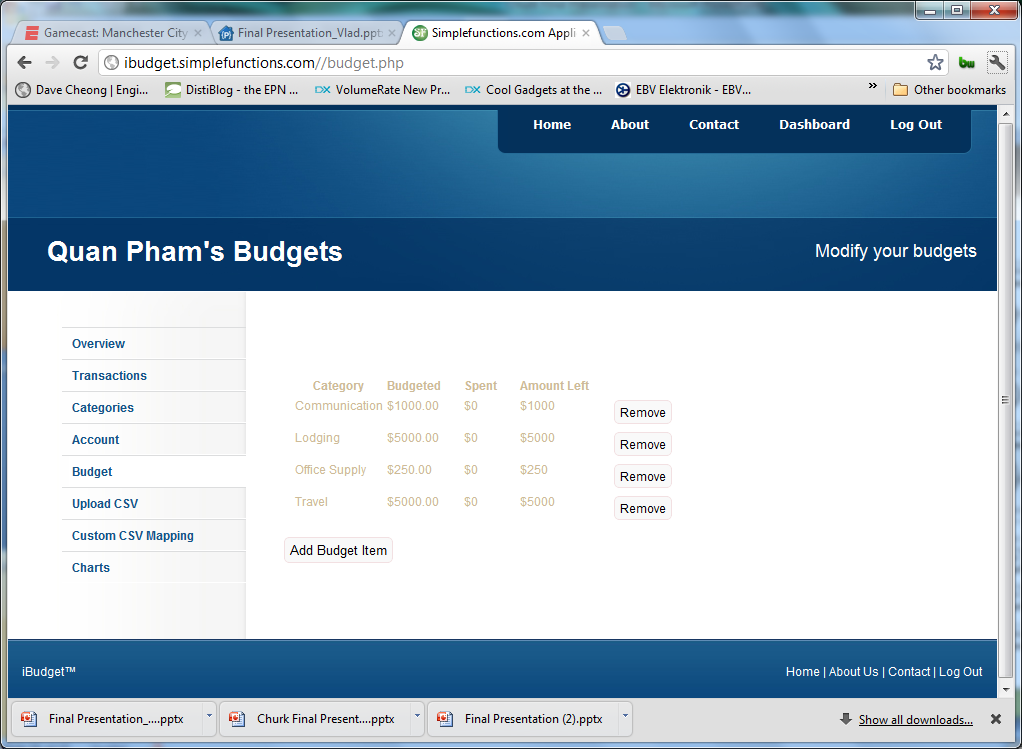


## Budget View

## Overview

## Users can track their monthly spending on any Category they have created. When users create a budget, they select a category and a monthly allowance, they can then view all the budgets and see how much they have spent in the current month on each budget, versus the amount they have allotted.

## Screenshot

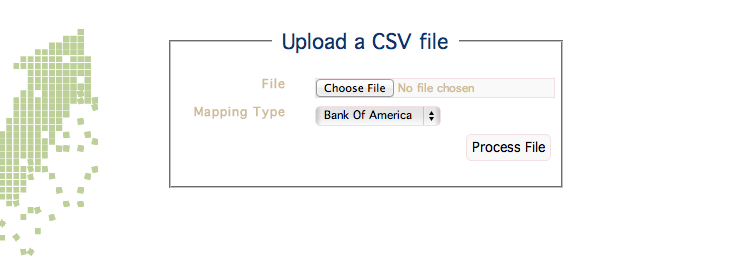


## Process CSV

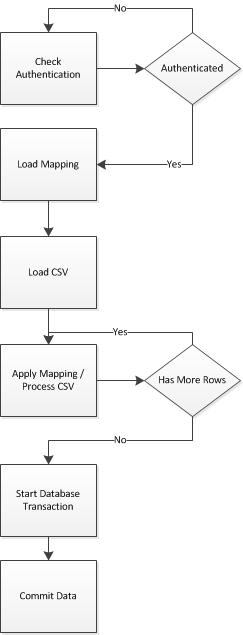
## 4.10.1 Overview

Process CSV is one of the core components of iBudget. Turning provided CSV files into data that we can understand, store, and transform. The process first load the mapping settings which directs how the translate the CSV files. The mapping will be discussed at a later section. Which the mapping understood, the process reads the file line by line, and applies the mapping settings, The result will create an Activity object and the entire process will create a list of activity objects. If the process complete successfully, a database transaction is created to submit all these data. A randomize number between 10000000000 and 99999999999 is generated and is assigned to the list of activities in conjunction to a date time, this will help identify an import transaction, so when use needed to revert the import, this can be done easily.

## Screenshot



## State diagram

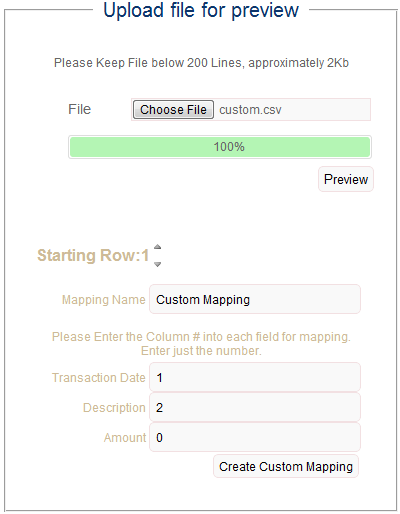


## Custom CSV Mapping

## 4.11.1 Overview

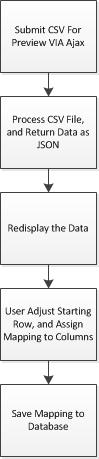
In addition to the built-in mappings from major banks, users can create their own custom mappings from virtually any format that their CSV files happen to be in.

## Screenshot





## State diagram



## Chart View

## 4.12.1 Overview

Instead of looking at numbers, users can visualize their financial health by looking at charts broken down by categories.

## 4.12.2 Screenshot

