Up-to-Date Budgeting with the iPhone

By:

Lawrence Powers

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

University of Cincinnati
College of Applied Science

June 2010

Lawrence Powers

Date

Professor Russ McMahon

Date
# Table of Contents

SUMMARY .................................................................................................................................... 3

INTRODUCTION ......................................................................................................................... 4  
  Purpose ................................................................................................................................................... 4  
  Scope ........................................................................................................................................................ 4  
  Procedure .............................................................................................................................................. 4  

DISCUSSION ................................................................................................................................. 4  
  Proposal ................................................................................................................................................. 4  
    Problem .............................................................................................................................................. 4  
    Solution ........................................................................................................................................... 5  
  User Profile ........................................................................................................................................ 6  
  Design Protocols ................................................................................................................................ 6  
    Server ................................................................................................................................................ 6  
    Client/Server Communication ........................................................................................................... 12  
    Client ................................................................................................................................................ 13  
  Testing ................................................................................................................................................. 13  
  Risk Management ............................................................................................................................. 14  
  Proof of Concept ................................................................................................................................ 15  
  Project Planning ................................................................................................................................ 15  
    Project Costs .................................................................................................................................. 15  
    Project Timeline ............................................................................................................................. 16  
  Deliverables ....................................................................................................................................... 16  

CONCLUSION ............................................................................................................................ 17
SUMMARY

This report details the entire project process of a Senior Design project required for completion of the Bachelor of Science degree. This project is a Software Application Development project that encompasses topics such as Objective-C development, iPhone SDK interfacing, server-side development with PHP, Web Services, and Database design.
INTRODUCTION

Purpose
Budgeting in personal finance is more important today than it has been in previous years. The increasing jobless rate has created an America where consumers are spending less and saving more. Setting budgets are an important step in achieving this goal. Budgets can take form in many ways; they can be written, entered in a spreadsheet, entered in specialized finance software, or even kept in the memory of a budgeter. All of these solutions have drawbacks. The proposed iPhone solution is a mobile software and database project that addresses these issues.

Scope
This report details the entire project from inception to implementation. User profiles, design protocols, testing plans, risk management plans, proof of concepts, and project plans are topics discussed that documents these elements.

Procedure
Knowledge gained from four years of classroom learning, professional practice, and research was used in the creation of this project. Professor Dan Humpert has provided project management guidance and Professor Russ McMahon has provided the project’s technical guidance.

DISCUSSION

Proposal

Problem
There are a number of computing solutions that provide budgeting. These include: desktop solutions, web based solutions, and mobile solutions. The most popular of these is a web and mobile-based solution called Mint. This is a great tool, but it's not without shortcomings. (1) Mint is slow; it takes minutes to update transactional information. (2) Mint’s iPhone application lacks a full set of features that relate to budgeting. (3) Mint’s account information is not up-to-date with the user’s latest purchases and deposits.

The third deficiency is the most significant. The fact that mint doesn't include recent transactions can be disastrous for keeping a budget. Delays in updated information happen because of the way mint gathers transactional information from the financial institutions. Mint only downloads transactions that have posted to the user’s account. According to Bank of America’s website, a credit card transaction goes through four steps until the transaction is finally posted ( ). This takes days and in some cases even longer. This is synonymous to writing a check. The receiver of the check may cash it right away, or they
may hold on to it for a while. These delays can cause a user to forget that they made a transaction.

Solution
A custom-built iPhone application will address all three of these issues noted above. A mobile solution was chosen because of the decisive actions of adjusting and reviewing budgets are usually done on the go. For example, if you are at the store considering on buying a new piece of clothing, you can check to see if you have enough money left in the clothing budget. Better yet, you can adjust your clothing budget in real time and adjust another category to compensate for the purchase.

Adjusting budgets in real time is a feature mint's iPhone application lacks. This proposed solution would implement the zero-based budget. In zero-based budgeting all income minus outgo must equal zero. Savings is included in the outgo. One of the most important features of the zero-based budget is with budget adjustments. If you increase one category by x you must decrease another category by x. You could also absorb the decrease in multiple categories. This feature of the proposed solution addresses the budgeting features missing in the Mint’s solution.

Another concern with mint is its speed. Updating transactions takes minutes. Mint has a model that takes in consideration all the user's financial accounts when updating. This can include checking, saving, credit, investment, and other accounts. The proposed solution is concerned only with budgeting your money on a day-to-day basis, whereas, mint gives you a whole overview of your financial portfolio. Instead of pulling information from all your accounts, only checking transactions is downloaded. This increases the download speed considerably. This is a reduction in time from minutes to seconds.

As noted above in the discussion, the most important deficiency of mint's application is the lack of real time transactional information. Having up-to-date transactions and budgets are important. To understand why, consider this example. If Jon had a budget he should be able to look at his iPhone to determine how much money he has allocated for his restaurant budget. Let's say he has $20 left and goes to a restaurant and spends $15. Within seconds of the restaurant swiping his card, he should be able to open his iPhone application and see that $15 is gone from his food budget. So, an hour later when he is thinking ice cream, he should only see $5 left in that category.

Taking temporary authorizations into account within budgets solves this problem. A number of banks and credit card issuers display temporary authorizations on their web application along with posted transactions. Within seconds of swiping your card at a merchant, the bank or credit card company lists that transaction on the user's online banking website. The proposed solution will grab these transactions. Every time the iPhone application is ran the transactions and budgets are up-to-date with the user's current spending activity.
There are a few shortcomings with this approach that I will address. Temporary authorizations are just that temporary. They disappear when the transaction posts to the account. The proposed application will resolve each transaction as it posts. The other issue is the amount in the temporary authorization is not always accurate. This can be true if users leave a “tip” on the card at a restaurant. Another instance is when buying gas; the temporary authorization is for only for $1. This is done because the gas stations don’t know how much gas you are going to use, but it wants to make sure it’s a valid card before it allows your to pump the gas (it can’t steal back the gas once it’s in your tank). This issue is resolved by allowing the user to change the transaction amount once it comes into the application. Any discrepancies between the temporary authorization and the resulting posted transaction will be relayed to the user.

**User Profile**

Both non-technical and technical people will use this application. Therefore, little technical knowledge will be required for its use. Knowledge of how to use the common iPhone controls is the only skill the user needs. The target audience is anyone who has a checking account with debit card access and an iPhone.

**Design Protocols**

The project is broken down into two different major components. These include the back-end server and the front-end iPhone application. This is known as client/server architecture.

**Server**

The development server is an Ubuntu Server version 9.10. The following packages have been installed: PHP 5.3.0, PEAR, cURL 7.20.0, MySQL 5.10, and OpenSSL.

**PHP** – “PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.” (PHP.com) The project uses PHP to implement the logic that provides information for the iPhone client.

**PEAR** – Pear is a PHP framework that provides common implementations of common problems in software development. PEAR has separate “packages” for different functionalities. The project uses one of these packages. This package is “xml-rpc2” which is an implementation of the “xml-rpc” protocol. This protocol is a web service implementation that allows remote procedure calls over a HTTP(S) network. The iPhone client application will utilize these remote calls to retrieve information.

**cURL** – “cURL is a command line tool for transferring data with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS and FILE.” (Curl.com) The project uses this technology along with PHP to access the raw html of banking institutions.
MySQL – MySQL is an open source database that is free to use under the General Public License (GPL). The project uses MySQL to store transaction information, budget information, and other application specific data.

OpenSSL – “OpenSSL is a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.” (OpenSSL.com) The project uses OpenSSL to encrypt the information between the server (Ubuntu Linux) and the client (iPhone).

The function of the server is three-fold. The first responsibility is to retrieve and parse transaction information from the banking institution. The second is to provide the business logic of the application. Lastly, the server is responsible for providing access of the information for the iPhone client.

Setting up the Server Environment

The development environment is shown in Figure 0. The important items on the network are the devbox (Ubuntu Server), a D-Link switch, Linsys router, Motorola Cable modem, and the Internet cloud.

Figure 0: Development Environment

The devbox server is assigned a static IP address. This is accomplished using the following commands:

```
sudo vi /etc/network/interfaces
auto eth0
iface eth0 inet static
    address 192.168.1.100
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.1
:wq
```
The Linksys router has two ports that are configured to forward to the devbox's IP address. These are port 22 for SSH and port 80 for HTTP. The cable modem provides the Linksys router a dynamic IP address. Using a service called dyndns.org and a function on the router, the dynamic IP is updated to the DNS whenever the IP changes. This IP is mapped to powers.homeunix.com. Everything else in the network is configured correctly by default.

Apache, MySql, and PHP were installed by default upon installation. The following packages had to be installed manually with these commands:

```
sudo apt-get install php5-curl
sudo apt-get install phpmyadmin
sudo apt-get install php-pear
```

**Capturing Transactions from Banking Institution**

As explained above, cURL was used with PHP to acquire banking transactions from the bank's website. The following code shows how this is done:
<?php

/*
Programmer: Lawrence Powers
Date: Oct 27, 2009
Module Name: PendingTransactions
Module Description:
This script screen scrapes the the Fifth Third web application
for pending transactions and inserts them into a table.
*/

include "config.php"

$arrdate = array();
$arrtime = array();
$arramount = array();
$arrdescription = array();

if(strcmp($environment, "development") == 0)
{
    //get transactions from test location
    $ch3 = curl_init();
    curl_setopt($ch3, CURLOPT_URL, "http://localhost/dummy.html");
    curl_setopt($ch3, CURLOPT_RETURNTRANSFER, true);
    $output = curl_exec($ch3);
    curl_close($ch3);
}
else
{
    //Login to bank
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookieFileName");
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_URL, "https://www.53.com/mobile/secure/login");
    curl_setopt($ch, CURLOPT_REFERER, "https://www.53.com/mobile/public/login");
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, "j_username=xxxxxxx&j_password=xxxxxxx");
    $output = curl_exec($ch);
    curl_close($ch);

    //Select Account
    $ch2 = curl_init();
    curl_setopt($ch2, CURLOPT_COOKIEJAR, "/tmp/cookieFileName");
    curl_setopt($ch2, CURLOPT_URL, "https://www.53.com/mobile/secure/accountDetail?accountID=001");
    curl_setopt($ch2, CURLOPT_REFERER, "https://www.53.com/mobile/public/index");
    curl_setopt($ch2, CURLOPT_COOKIEFILE, "/tmp/cookieFileName");
    curl_setopt($ch2, CURLOPT_FOLLOWLOCATION, 1);
    $output = curl_exec($ch2);
    curl_close($ch2);

    //Navigate to pending transactions
    $ch3 = curl_init();
    curl_setopt($ch3, CURLOPT_COOKIEJAR, "/tmp/cookieFileName");

Figure 1: Transaction retrieving program code (part1)

Figure 1 shows how to use cURL with PHP to access websites programmatically. This code shows how we first POST the user's login information to the bank's logon page. Once we are authenticated by the web application, we navigate to the user's checking account. The bottom of Figure 1 and the top of Figure 2 show how we navigate to our final destination, the pending transaction page. One thing to note, this module takes into account two different environments, development and production. The development environment points to a mockup of the bank's website that resides on a development server and the production environment points to the actual bank's website.
Once the script navigates to the pending transaction page, the raw HTML of the page is captured into a variable. The HTML is parsed using a screen scraping technique. This works by finding the desired information by capturing the data near specified static text, which doesn’t change. The gathered transaction information is then inserted into arrays. There are separate arrays for dates, times, amounts, and descriptions. One thing to note, if the bank changes the layout of its webpage, it will break the code and will cause the application to fail. This risk is described in greater detail in the Risk Management section. Finally, as shown in Figure 3, the data from the arrays are inserted into a database table.

Figure 2: Transaction retrieving program code (part2)
Figure 3: Inserting records into a database table

An example of how the page's HTML is rendered is shown in Figure 4. The script doesn't care how it's rendered, but it's helpful to see it visually. A portion of the HTML code is shown in Figure 5.

Figure 4: Rendered HTML of pending transaction page

Figure 5: Portion of the HTML code of the pending transactions page
The mechanism described above retrieves transaction information from the bank and standardizes the data by storing the information into a relational database. Figure 6 shows the information in its standard form inside the pending transactions table.

<table>
<thead>
<tr>
<th>transID</th>
<th>Amount</th>
<th>Description</th>
<th>Date</th>
<th>Time</th>
<th>username</th>
</tr>
</thead>
<tbody>
<tr>
<td>d6a8fd231d83a596d37efba228d</td>
<td>6.76</td>
<td>PRE-AUTHORIZATION DEBIT AT CHIPOTLE 002, CINCINNATI</td>
<td>2010-02-01</td>
<td>04:26:00</td>
<td>powersin</td>
</tr>
<tr>
<td>3166ed91c0a28ba5686b75c289a965</td>
<td>4.38</td>
<td>PRE-AUTHORIZATION DEBIT AT COL OF APPLIED SCIENC</td>
<td>2010-02-01</td>
<td>12:40:00</td>
<td>powersin</td>
</tr>
<tr>
<td>d70b26fa6f1bb2a7aaf1ba6a471a913</td>
<td>8.18</td>
<td>PRE-AUTHORIZATION DEBIT AT SKYLINE CHILI, CLTN</td>
<td>2010-01-31</td>
<td>08:39:00</td>
<td>powersin</td>
</tr>
<tr>
<td>f52a35a06074694551939316b6b75a</td>
<td>7.00</td>
<td>PRE-AUTHORIZATION DEBIT AT TOP SHELF - HARRISON A.</td>
<td>2010-01-31</td>
<td>12:08:00</td>
<td>powersin</td>
</tr>
<tr>
<td>b96ed576c92c827a10c0a1237</td>
<td>5.00</td>
<td>PRE-AUTHORIZATION DEBIT AT TOP SHELF - HARRISON A.</td>
<td>2010-01-30</td>
<td>11:30:00</td>
<td>powersin</td>
</tr>
<tr>
<td>2a61f529e245cb3a5d9b5d6131</td>
<td>17.41</td>
<td>PRE-AUTHORIZATION DEBIT AT DANIES STEAKHOUSE &amp;</td>
<td>2010-01-30</td>
<td>12:24:00</td>
<td>powersin</td>
</tr>
<tr>
<td>36cc33e7dfb02555e27c0251dc0f5f18</td>
<td>40.00</td>
<td>ATM WITHDRAWAL AT 6165 GLENAVEY AVE, CINCINNATI, OH</td>
<td>2010-01-29</td>
<td>03:10:00</td>
<td>powersin</td>
</tr>
<tr>
<td>b3f5bf549165ce030f147bca70c</td>
<td>31.94</td>
<td>CARD TRANSACTION : TICKETMASTER TICKETS, CHICAGO</td>
<td>2010-01-29</td>
<td>03:05:00</td>
<td>powersin</td>
</tr>
<tr>
<td>7dcf0125907d1e287a56d2b79f059f965</td>
<td>8.95</td>
<td>CARD TRANSACTION : KRISHNA CARRY OUT, CINCINNATI</td>
<td>2010-01-29</td>
<td>01:13:00</td>
<td>powersin</td>
</tr>
</tbody>
</table>

**Figure 6: Pending transactions table**

The [transID] field is the primary key of the table and is derived by taking the MD5 hash of the concatenation of the [Amount], [Description], [Date], and [Time] fields. This is done in order to differentiate transactions. If the script is ran again, it will not result in duplicate transactions. This is because of a unique constraint set on the [transID] field. A similar procedure is done to acquire posted transactions, which are stored in the posted transaction table.

**Client/Server Communication**

Communication between the server and the iPhone application is handled through web services. Web Services allow for remote procedure calls over a HTTP(S) connection. The server fabricates an xml document based on method call from the iPhone. The xml document is transferred over http and is then parsed by the application. Figure 7 shows an example of a web service used in the application.

```php
<?php
include "db.php";
$result = mysql_query("SELECT * FROM pendingtransactions");
echo "<Bank>
while($row = mysql_fetch_array($result)) {
    echo "<Transaction>
    echo "<Amount>" . $row["amount"] . "</Amount>
    echo "<Description>" . $row["Description"] . "</Description>
    echo "<Date>" . $row["Date"] . "</Date>
    echo "<Time>" . $row["Time"] . "</Time>
    echo "";
    echo "</Bank>
include "disconnect.php";
?>
```

**Figure 7: Pending Transaction Web Service**
**Client**

The iPhone client utilizes the resources that the server provides and much of the processing is done on the server. The client is mostly responsible for presenting the information to the user. To support this role, the client needs to be able to consume web services. The iPhone doesn’t have out of the box support for web services.

The client was developed using a (Model, View, Controller) MVC paradigm.

**Model Classes:**
- Budget
- PendingTransactions
- PostedTransactions

**View:**
- FirstView
- SecondView
- ThirdView
- TransactionView
- TransactionDetails
- Picker
- BudgetDetails
- AddBudget
- MainWindow

**Controller Classes:**
- FirstViewController
- SecondViewController
- ThirdViewController
- TransactionViewController
- NavigationController
- BudgetDetailController
- PickerViewController
- AddBudgetController

**Testing**

Different types of tests will be used throughout the lifecycle of the project. As soon as a method is written, unit tests are performed. Other testing includes integration testing, system testing, stress testing, and performance testing. The documentation style will be a table including test type, test description, version of program, success/failure, network type (3G/Edge/Wifi), and date/time of test. The goal is to have all tests pass and have at least 80% code coverage.
## Risk Management

<table>
<thead>
<tr>
<th>Risk</th>
<th>Impact</th>
<th>Probability</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank blocks scraping techniques</td>
<td>High</td>
<td>Medium</td>
<td>Have a mockup of Bank's website on development server</td>
</tr>
<tr>
<td>Server goes down</td>
<td>Medium</td>
<td>Medium</td>
<td>Have a copy of transaction data on client</td>
</tr>
<tr>
<td>Network down/out of network area</td>
<td>Medium</td>
<td>Low</td>
<td>Have a copy of transaction data on client</td>
</tr>
<tr>
<td>Bank changes layout of website</td>
<td>High</td>
<td>Medium</td>
<td>Script will notify Administrator through email of any changes.</td>
</tr>
</tbody>
</table>

Table 1: Risk Management
Proof of Concept

Project Planning

Project Costs

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Price</th>
<th>Actual Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacBook Pro</td>
<td>$1199</td>
<td>$1099</td>
</tr>
<tr>
<td>Custom server – E6700 core 2 duo with 4 gigabytes of ram and 250 Gigabytes of RAID 5 storage</td>
<td>$1500</td>
<td>$1500</td>
</tr>
<tr>
<td>iPhone 3G</td>
<td>$99</td>
<td>$99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$2,798</strong></td>
<td><strong>$2,798</strong></td>
</tr>
</tbody>
</table>

Table 2: Hardware Costs
Table 3: Software Costs

<table>
<thead>
<tr>
<th>Solution</th>
<th>Price</th>
<th>Actual Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu Server 9.10</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>PHP 5.3.1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>cURL</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>MySQL 5.1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Xcode</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Mac OS 10.6</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Iphone OS 3.1</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Developer license</td>
<td>$99.99</td>
<td>$99.99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$99.99</strong></td>
<td><strong>$99.99</strong></td>
</tr>
</tbody>
</table>

Project Timeline

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start Date/Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Projects</td>
<td>9/20/2009/10 days</td>
</tr>
<tr>
<td>Research Technologies</td>
<td>1/10/2009/14 days</td>
</tr>
<tr>
<td>Proposal Proposal</td>
<td>1/10/2009/7 days</td>
</tr>
<tr>
<td>Design</td>
<td>1/16/2009/100 days</td>
</tr>
<tr>
<td>Implementation</td>
<td>1/1/2010/100 days</td>
</tr>
<tr>
<td>Testing</td>
<td>1/21/2009/35 days</td>
</tr>
<tr>
<td>Prepare Final</td>
<td>5/20/2010/7 days</td>
</tr>
</tbody>
</table>

Deliverables

- Application is able to retrieve both pending and posted transactions of account
- Application is able to accept new and modify existing budgets
- Application is able to display spending trends and totals
CONCLUSION

Users need good budgeting software that is automatic but flexible. This application will be a feature-filled, fast, and a up-to-date solution. Users will be able to make real time decisions with their money and will be able to live within their means.