



COMP-SCI 5551 (FS16) - Advanced Software Engineering

Project Team 1 - The Brokers

Joshua Neustrom (39); Yunlong Liu (25); Chen Wang (58); Dayu Wang (59)

---

## **Project Increment Report 3**

(Due Nov 9<sup>th</sup>, 2016)

- **Introduction**

As many college student know, living on their own is a big change as well as the academic program, they may find that why i have stable monthly income, but still no money available at the end of the month. One thing for sure is that they all try their best to make themselves financial independence. The first step is to make a expense plan which can control the money flow, trace where we spend the money, that's the first thing we need to do, next stage, we should have a good habit of using money, this is the very important reason we need a application to help college students have this kind of good money using habit smoothly, from credit card abuse or don't know where they have spent their money.

Their are a few problems that may occur among students, they always splashed at the beginning of the month but starving at the end of the month. We will create budgets for those students, A simple and clearly understanding budget can help college student manage their money effectively, the system will tell them how much they can use on each parts based on their different case. For those students who abuse their credit card, we provide a function called separate wants from needs, separate the wants from needs can help the user to get the real thing which they are needed. Based on this the students can improve the habit of the expense of college student, the Pocket Manager can tell the student what kind of expense are useful and necessary, then the user can manage their expense more reasonable.

For the people who confused with where they have spent their money, we will grade their behavior of last month and give them the suggestion and a figure of the aspects where they spent the money.

- **Features**

feature 1 - login system

The login system can record the basic information about the users, such as name, email address and so on, then everybody can make different expense model for themselves based on their different incomes. Then the user can analyze and manage their own expense using their account.

feature 2 - history storage

Pocket Manager can save the basic information and expense history in the MongoDB, then the user can scan and analyze their own expense record. The history storage can also help Pocket Manager to give remind and alert to the users. User's expense record, grade, receipt photos and other information always saved in the history storage (MongoDB).

feature 3 - create budget

This feature need collect the income of the users, then the system will recommend three different model of expense for the user: frugal model, moderate model and amusement model. They give the different plan for the user, and the type of them have been reflected on their name. Then the user can plan and manage their expense based on these model. It helps user to manage their expense more effectively.

feature 4 - separate needs from wants

With this feature, the user can search an items through the EBay API, then the photo, price and details about items will be returned. The user can analyses if this item is needed or just wanted based on the information that the feature displayed. It can help the user use their money more Reasonable. It will reduce a lot of unnecessary wastes.

feature 5 - record a payment

This feature can help the user to the record their payments, it even can save users' receipts using the camera, then the user can see all the information about the expenses. Then they will know which payment is unnecessary and why they always exceed the limit of the plan.

feature 6 - graphical representation

For these parts, the user can see their statistical information about their expense. It can show all the details about the individual expense clearly. User can find which is the most cost parts in the expense and make a better plan for the next month.

feature 7 - give a grade for the user

At the end of the month, users can get a grade for their expense. Based on it, they can know whether their expense habit is good or not. Then they can improve it at the next month.

- **Existing Services/API**

1. Uploads.im

Uploads.im is a API that can help user to upload the files and documents and store them in its database. The users can upload and download the files and documents using the GET and POST operation. The response format can be JSON, XML, RAW TEXT and REDIRECT. Such as the images in the local disk, user can upload them into the Uploads.im database, then user will receive URL of these images then the user can easily to get their images. Because a lot of database such as the MongoDB cannot save the images very well, then users can through the Uploads.im save the images effectively. For our project, the images that user upload for their portrait will be saved online. Then the user can get their portrait at anywhere and any devices using their own account. Even if the user deleted their local images. The Uploads.im still will save the portraits for them. The place, local storage and devices will never effluence their portraits. The Upload.im give an effective way to save and get the images for our projects.

2. Flickr

Flickr is Yahoo website which can host the images and videos. User can upload and download their images and videos in the Flickr. And the user can also share their images and videos to others using the Flickr, then them can browse and appreciate other's images and videos. It is also a very good storage for saving the images, so for our project, it can also be used for saving the users' portrait. It is another way to solve the problem that the MongoDB is hard to upload the images.

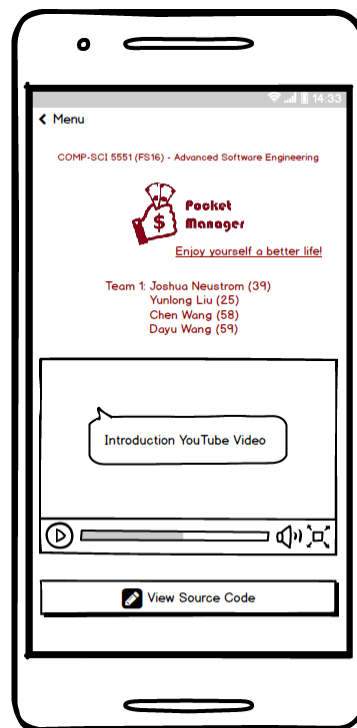
- **Detail Design of Features**

1. Mock-Ups

**Mock-up tool used: Balsamiq Mockups**

**Version: 3.5.5**

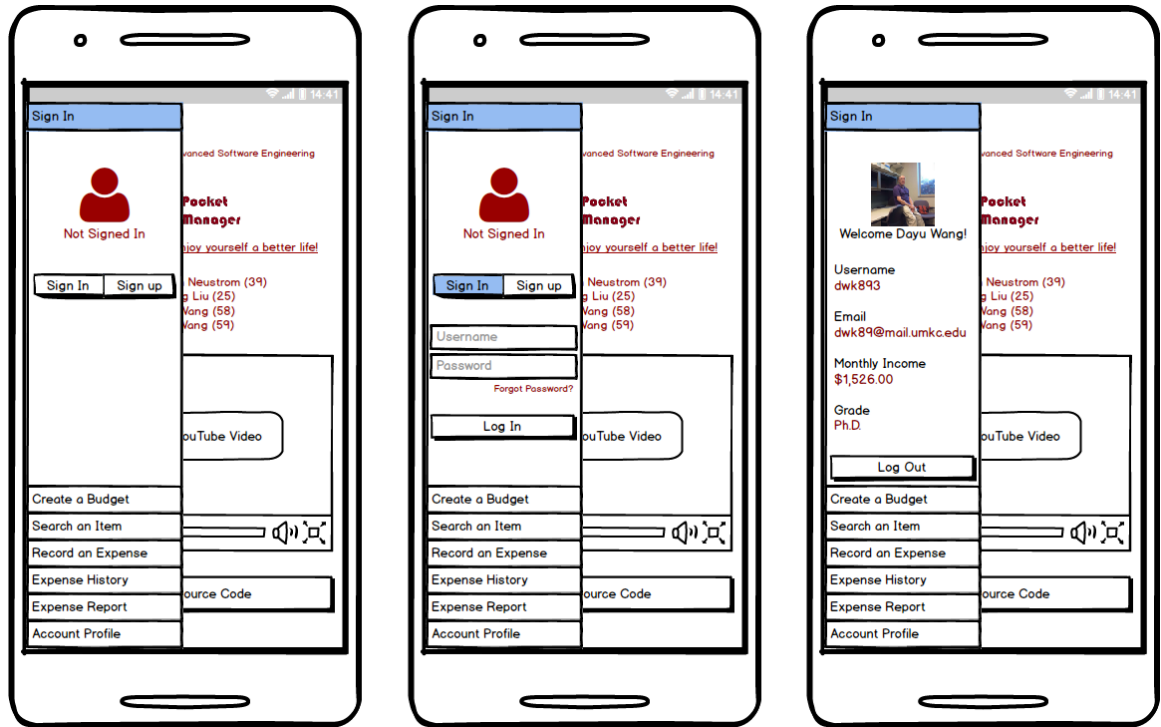
Figure 5-1 is the designed mockup of the main page of the *Pocket Manger* system that the represents the identity of the system for the new users who first . The *menu* button in the top left of the screen triggers the appearance of the left side bar menu, which functions just like the *Start* button in the task bar of the Windows system, everything just begins here. Other than the basic class and team information appearing in the top of the screen, a YouTube video about the introduction of the entire system is designed to appear around the center of the screen to give chance to the users to have a general idea about our product. Also, a button that directly links to the GitHub repository of the source code of the system is also provided at the bottom of the screen.



**Figure 5-1.** Refined mockup of the home page of the *Pocket Manager* system.

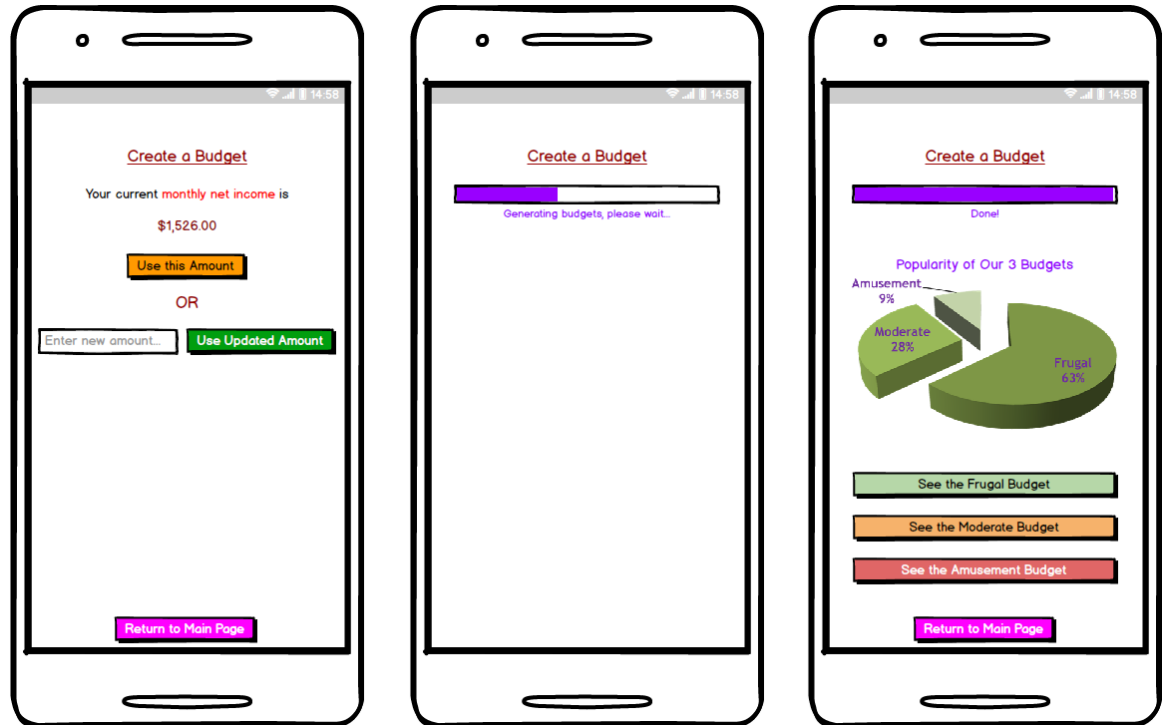
Figure 5-2 represents the mockup views of the *Pocket Manager* system related to the **login** feature. When the side menu pops up, it contains all the features and functions that embedded inside the system. When the *Sign In* tab is activated, it appears two buttons, *Sign In* and *Sign Up*. When the *Sign In* button is clicked, then the system will allow the user to input his/her *username* and *password*. If the user successfully logged in, then under the *Sign In* tab in the side menu, basic information of the user is available to see. The key factor in the designation of the login system is that, though a little bit of

tricky, everything just happens within the side bar. The login system will not affect the appearance of the Android pane.



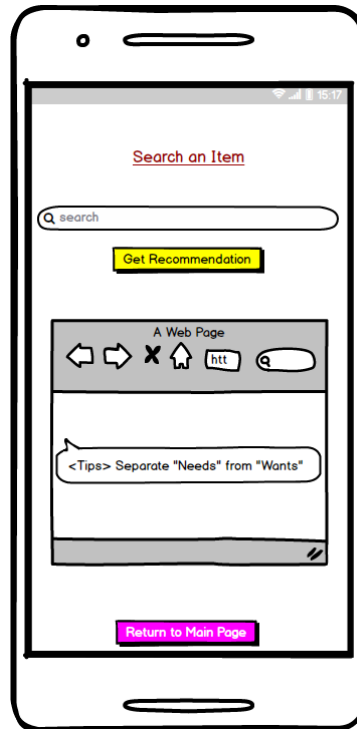
**Figure 5-2.** Mockup pictures of the login part of the *Pocket Manager* system. The left picture reveals the initial appearance and the state when the user has been logged out. The middle picture is when the user is attempting to login. The right picture demonstrates the view when the user has been successfully logged in.

**Figure 5-3** shows the first core feature of the *Pocket Manager* system—*Create a Budget*. When the user chooses to create a budget, the system will first ask the user to confirm his/her monthly income, since the amount of user’s monthly income may be out dated. When the user confirms or updates his/her monthly income, the system begins to calculate three different budgets for the user. The three budgets are called *Frugal Budget*, *Moderate Budget*, and *Amusement Budget*. The detailed introduction of the three budgets can be found in the “features” section of this paper. After the three budgets have been completed generating, the user can choose to see each of them and make his/her decision of which budget he/she would like to use. On the same page, a pie graph of how popular the three different kinds of budgets are amongst college students is shown in the middle of the page. This acts as the system’s recommendation to the user when he/she is examining different kinds of budgets.



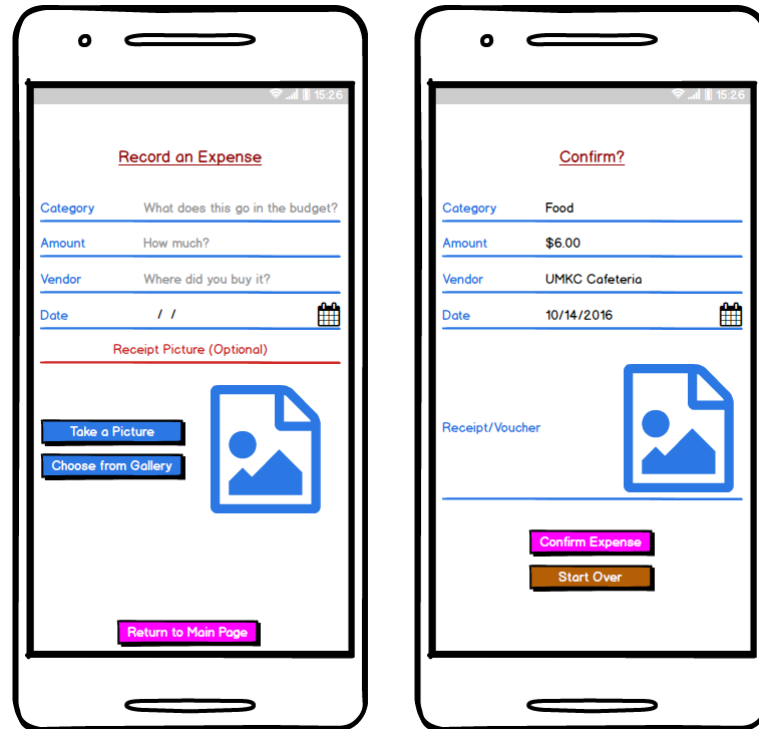
**Figure 5-3.** Mockup pictures of the feature of *Create a Budget* in the *Pocket Manager* system. The left picture is the initial view when the user chooses the feature. It asks the user to confirm his/her monthly income or update his/her monthly income before the system can calculate budgets for the user. The middle picture is what appears in the screen when the system is calculating different budgets. The right picture represents the view after the system successfully generated the budgets. The pie graph in the middle acts as the system recommendation to the user in which the popularity of different budgets amongst college students is explicitly articulated.

**Figure 5-4** represents the main page when the user chooses to *Search an Item*. Definitely, a search input textbox and a search button are necessary. However, since in this page there is not too many objects, we decided to add another object, which can be either a website view or a text view, in the middle of the page. The object is intended to tell the user some tips of how to separate needs from wants.



**Figure 5-4.** Mockup picture of the principal view of the feature of *Search an Item* in the *Pocket Manager* system.

**Figure 5-5** demonstrates the views of the key feature of *Record an Expense* in the *Pocket Manager* system. An expense has *category*, *amount*, *vendor*, and *date*. The user also has the option of input the picture of the payment receipt, which can be either using the Android camera to take a photo of the receipt or choose the receipt picture from the gallery. After the user's input, the user has to confirm that everything is correct.



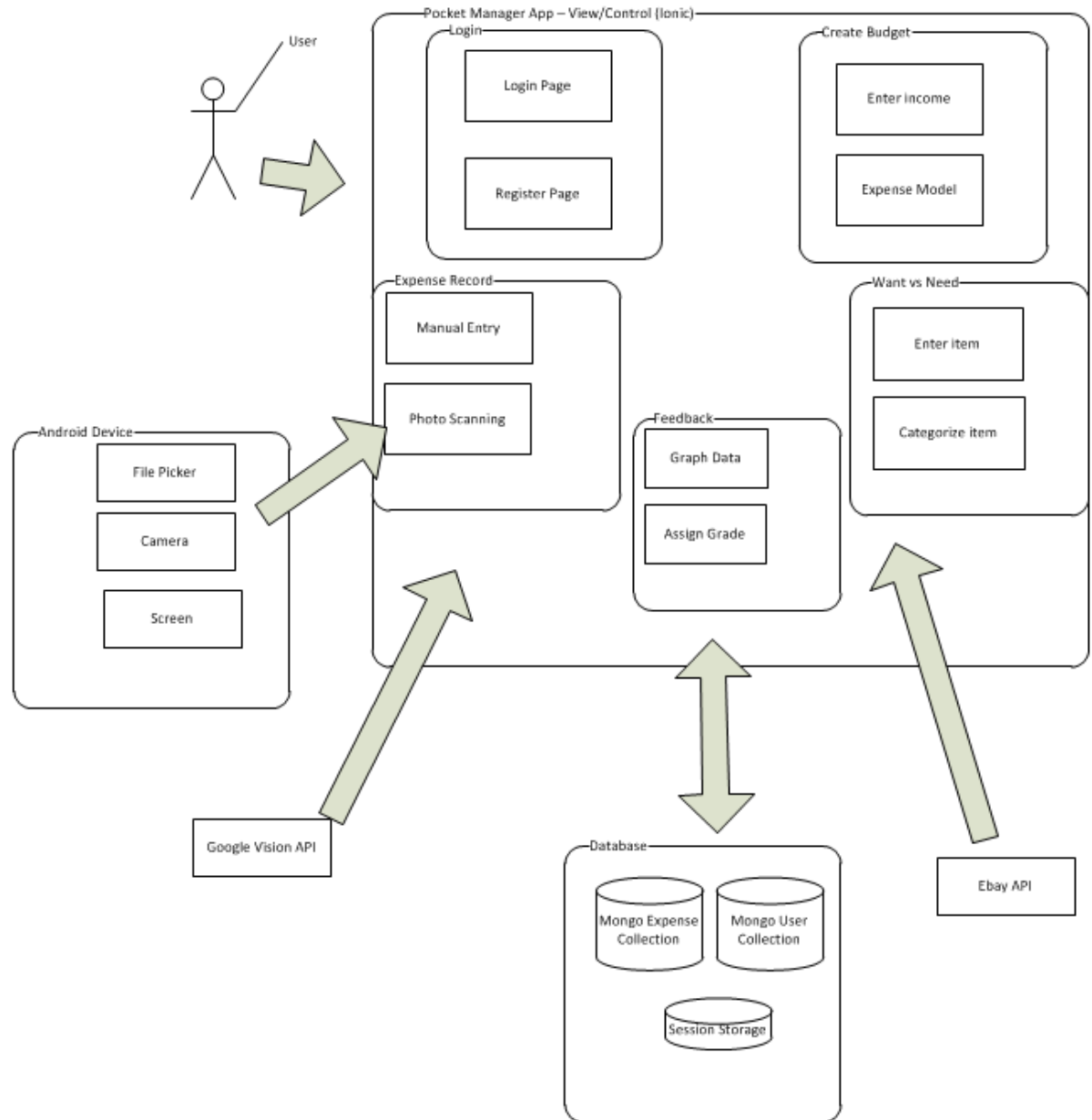
**Figure 5-5.** Mockup pictures of the *Record an Expense* feature in the *Pocket Manager* system. The left picture is the view when the user is inputting an expense. The right picture is the confirmation page appeared right after the user's input.

## 2. Architecture Diagram/Sequence Diagram/Class Diagram

Software architecture is defined by New York University (NYU) as the structure of structures for a system. The diagram outlines a top level view of key systems utilized such as software frameworks and API services with basic connections on how they interact.

For the *Pocket Manager* system, the architecture covers the native application, phone hardware, API services, and remote database. Ionic was the selected Framework for the native application which uses HTML5 and AngularJS to build Hybrid Applications. Major functions include the login, budget creation, scanning receipts, recording payments, fetching statistics, graphing, and budget creation. Besides hosting the actual application, the phone provides camera, screen, and data connection. Google Vision provides a text reading capability for the receipt scanning feature. For assistance categorizing transactions several APIs are being considered including EBay. The marketing data for an item can provide valuable information on how it will be utilized by the customer and how necessary the item is to their budget. Utilization will depend on what specific algorithm is selected for categorizing an expense as a luxury or necessity and how much of the data processing is conducted within the native application. For the data storage Mongo DB will hold user statistics and budgets. Mongo DB is a free and open source database system which is considered NoSQL and relies on JSON and documents instead of tables for its structure. The system architecture is shown below in **Figure 5-6**.





**Figure 5-6.** System architectural designation of the *Pocket Manager* to be developed.

- **Testing**

1. Overview

Testing of the application involved a multi-step process including pre-submission audit, unit testing, speed testing, virtual device testing, and physical device test. The process was designed to find problems in both the design of the code and features of the application.

2. Coding Audit

The following checklist is run on code at the start of testing in order to find errors early in the process and make it easy to find the root cause of a bug. The list must be completed by a group member before the coding submission is considered complete.

For the pre-submission audit, several checklists were developed to check the quality of code. The first section deals with the commenting and readability of the code. Code with clear formatting allows for easier troubleshooting and makes it easier for other users to modify code. For outside auditing and grading, clear formatting is a critical part of making it possible for outside parties to provide feedback. As a result, checking the format and documentation of code is a significant section of testing process.

If any software bugs or potential enhancements were uncovered before the submission, a description was posted as an issue. The team leader could determine how to resolve the issue by using any of the following options including accept the risk, assign someone to fix the issue, or seek outside help. **Table 6-1** is the result of the coding audit for our system.

**Table 6-1.** Coding audit checklist for the *Pocket Manager* system.

Item	Question/Comments	Response
a.	Does every section have at least one comment? (about one comment for every four lines of Java/JavaScript or one for every major section of html)	More comments are needed in code, not enough for other developers to follow
b.	Is the code neat? (not too many blank lines)	Yes
c.	Is proper spelling and grammar used?	Yes

Item	Code	Response
d.	Is the proper indentation used?	Yes
e.	Are variable and function names meaningful?	Yes
f.	Is major functionality subdivided logically into classes and activities?	Yes
g.	Is camel case used for functions and variables?	No, need standardization of variables

Item	Feedback	Response
a.	Were major issues submitted to GitHub?	No major issues
b.	Do you have any major concerns about your code?	Lack of Standardization

### 3. Unit Tests

Unit testing were tested by building small snippets of code to test sample inputs against expected outputs. Several areas were tested including the receipt recording section. The tests focused on the expense recording via camera feature which was one of the primary developed functions for this increment. The feature provides a convenient and fast way for students to record their expense.

a. Unit Test 1 – Upload Photo (Figure 6-2)

**Figure 6-2.** Take Picture Unit Test

Record an Expense	
Category	Walmart
Amount	How much?
Vendor	Where did you buy it?
Date	10/14/2016
Submit	
Take a Picture	

```

}

$scope.goPhoto = function(){
    // Go to the Receipt Page page
    PhotoService.takePicture().success(function(data) {
        // It worked head to the homepage
        console.log(data);
        $state.go('confirm');
    }).error(function(data) {
        // Wrong stuff, try again
        var alertPopup = $ionicPopup.alert({
            title: 'Error!',
            template: data
        });
    });
}

```

The first test looked at uploading an expense picture to Google Vision to extract the text. Even with Cordova camera, the take picture functionality was not sending data correctly to Google Vision. The issue was resolved by adding appropriate header info and sending the file using Cordova File Transfer. The final solution was developed as a service so it could be accessible by multiple controllers.

b. Unit Test 2 – Extract Date from Text (Figure 6-3)

**Figure 6-3.** Valid Date Unit Test

Record an Expense	
Category	Walmart
Amount	3000000000000000
Vendor	Walmart
Date	10/14/2016
Submit	
Take a Picture	

```

});
$scope.test = function(){
    var text = "THE MIXX\\nHAWTHORNE\\n913-338-4000\\n11942 Roe Ave\\nLeawood, KS 66209\\nCheck Tab Cashier Time\\nDate\\n340849 3530 1111\\n6:18:
    Tax\\n24.32\\nTOTAL\\nReceipt Used: Master Card\\nThank You!\\nVisit Us at mixxingitup.com\\nSend Feedback to info@mixxingitup.com\\n";

    // Get date
    var date;
    console.log(text);
    var twotwo = text.match(/\\d{2}\\d{2}\\d{4}/);
    var oneone = text.match(/\\d{1}\\d{1}\\d{4}/);
    var onetwo = text.match(/\\d{1}\\d{2}\\d{4}/);
    var twoone = text.match(/\\d{2}\\d{1}\\d{4}/);
    console.log(twotwo);
    console.log(twoone);
    console.log(onetwo);
    console.log(oneone);

    if(twotwo != null){
        date = new Date(twotwo);
    }
    else if (twoone != null)
    {
        date = new Date(twoone);
    }
}

```

The second test examined the parsing of data successfully extracted from a receipt photo. The data is returned in JSON as a long string and the date was not easily accessible resulting in today's date being selected by default. After the test initially failed, the text was extracted from a wide variety of receipts using a test environment and Google Vision. The text was stored in a document called Receipt Texts for Tests. Later the text was used in a test function where the vast majority of dates were found to have four digit years, use the mm/dd/yyyy format, and have '/' as the delimiter. As a result, searching for the following patterns found most dates in the text including: mm/dd/yyyy, m/dd/yyyy, mm/d/yyyy, or m/d/yyyy.

c. Unit Test 3 – Vendor name (Figure 6-4)

Figure 6-4. Valid Date Unit Test

Record an Expense	
Category	Walmart
Amount	3000000000000000
Vendor	Walmart
Date	10/14/0020
Submit	
Take a Picture	

```

    }
    else if (oneone != null)
    {
        date = new Date(oneone);
    }
    else if (onetwo != null)
    {
        date = new Date(onetwo);
    }
    else
    {
        date = new Date();
    }
    console.log(date);

    // Get vendor
    var vendor = text.split('\n')[0];
    console.log(vendor);

    vendor = vendor.replace('Welcome to', '').trim();
    vendor = vendor.replace('Welcome To', '').trim();
    vendor = vendor.replace('WELCOME TO', '').trim();
    console.log(vendor);

```

The final tests examined how to find the vendor name in the receipt text from Google Vision. As with the date, Google Vision's deciphering of text from a large number of receipts was read and saved to a file. Using the text in a test environment, a pattern in Vendor names was discovered. Typically, the Vendor name was found in the first line of the receipt. In some cases, the name was preceded by Welcome to. By taking text up to the first “/n” and removing any “Welcome to” the Vendor name was successfully found. One case that continued to fail is if the Logo was read by Google Vision as garbled text. Future development can look at how to remove any text from the logo before searching for the vendor name.

#### 4. Speed Tests (Figure 6-5)

The performance of a loading page can have a major effect on site traffic and usage. Poor performance can cause users to avoid a site or application. In addition, loading issues can increase support requests and result in poor app reviews. Since site and app visits can result in add revenue and perform user statistics which can be sold to various companies, page performance is critical to profitability.

Performance testing of page loads was completed using YSlow embedded into Google Chrome. The page is currently hosted locally and run out of the Ionic Framework. Since most pages scored very similar results, below are two samples of speed tests. For the next increment, we plan to look for a way to conduct speed tests on the App when it is loaded into a phone. Due to the Ionic Environment, current speed tests appear to highlight issues outside of our ability to optimize.

**Figure 6-5.** Home Page Speed Test

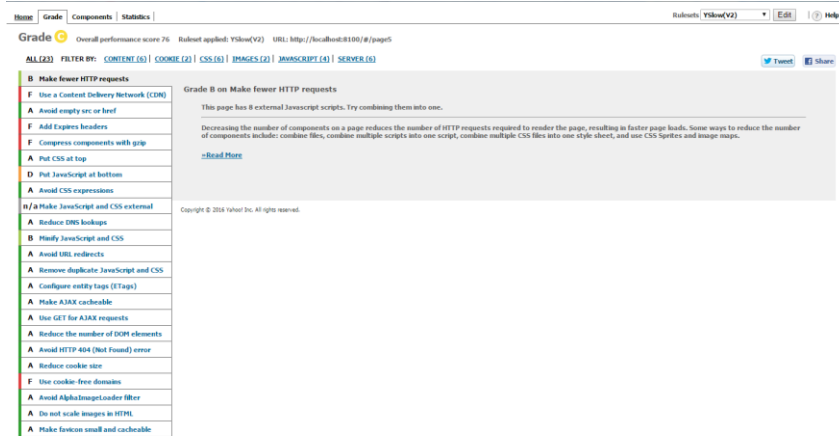
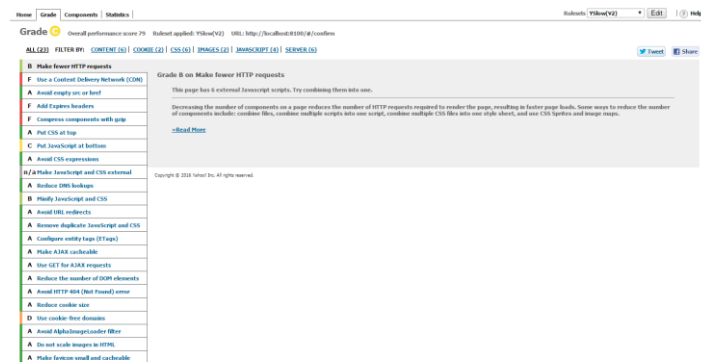


Figure 6-6. Rec Record Speed Test.



Overall speed tests averaged approx. 77 percent. Typically, fifteen areas received an ‘A’ rating. Of the poorly rated areas, most were due to the Ionic Framework or locally hosted pages. The moving JavaScript to the end of a page is an example of Ionic’s already separated application scripts being rated poorly. Furthermore, recommendations for Cookie Free Domains and Content Delivery Networks appear to be a byproduct of testing locally on a computer. Overall, it appears that the Ionic Framework creates a powerful hybrid application that comes with some inherent performance loss and that certain performance issues may remain until the application services are fully hosted remotely.

## 5. Virtual Device Test

At least two different virtual environments were used in testing

- Android Studio Virtual Device
- Gennymotion Virtual Device

Android Studio’s virtual device was utilized due to its close connection with Ionic. Unfortunately, it also had issues with slowing down the workstation and becoming time consuming. As a result, there was a natural incentive against exploratory testing. As a result, Gennymotion was also utilized due to its reputation for faster performance and its versatility. Special thanks goes to Dayu for purchasing the account with his own funds. (over \$100)

The checklist focused on giving some broad guidelines to the exploratory approach and has two major sections. The first part covers the user interface or the look and feel of the application. The detailed evaluation is designed to insure that the basic rules of good user interface are designed. Since developers may become highly engrossed in their work, certain obvious problems may be missed due to familiarity with the work. As a result, the below checklist forces developers to check each other's work in detail in order to catch mistakes before the software is deployed and in user hands.

The next major section involves the functionality of the application. Any new feature or modified area should be tested. In addition, the existing features should be tested to confirm that changes did not negatively affect existing functionality. Finally, the transition between major device states is tested. For example, the app can be exited and restarted to confirm no functionality is lost when reopened. In addition, the device can be restarted to confirm that the application still functions. See [Table 6-7](#) for the details of our analytical results.

**Table 6-7.** Exploratory Testing Checklist.

Item	User Interface	Response
a.	Visibility – Are all parts of the initial page visible?	Yes, all are visible
b.	Alignment – Does the alignments of parts look correct?	Yes
c.	Color – Do the colors appear as expected?	Yes
d.	Screen Changes – Does the page look correct from both screen orientations?	This will be more of an issue in future increments when more buttons are on the screen
e.	Keyboard – Can the app features still be used if the virtual keyboard pops up?	Same as above

Item	Features	Response
f.	Do all new features work as expected?	Yes,
g.	Do existing features still work?	Yes, except the user interface changed from last submission

Item	Transitions	Response
i.	Does the app work if someone exits then opens the app again?	Yes
j.	Does the phone function is the app has been running for over five minutes? (test of memory leaks)	Yes

Due to the late stage in development, the testing focused more on testing the code on a tablet instead of a virtual device. The Live Reload feature in Ionic allowed for changes to code to be tested quickly.

## 6. Physical Testing and User Feedback

Physical testing involves adding the app to a physical device, showing potential users, collecting feedback to utilize in future increments.

As a primary test device, we are using an Android 6 tablet in Portrait mode and an eight inch screen. The larger device makes it easier to see details of the application and evaluate behavior in depth.

The user feedback was obtained from three people with a variety of technical experience. The goal was to obtain objective feedback on the application which caught errors missed by the previous checklists and provided ideas for new features and enhancements. The form and summarized user feedback are below. (see [Table 6-8](#)).

**Table 6-8.** User feedback questions and responses.

Item	Question	Responses
a.	What do you like about the app?	Simple looks nice
b.	What do you dislike?	Home screen issues
c.	Any suggested changes?	Add a nicer background, easier navigation

## 7. Going Forward

As the software increases in complexity, testing is expected to be a much more important part of the development process. The checklists developed are expected to grow and be utilized by more than one team member. TA Feedback on the process will be a critical component in what direction the testing takes.

Although current testing involved working on each person's section separately, the final increment is expected to integrate all of the sections of the software together into one application. With the unified application, user testing is planned to give students an end to end view of the application and get their feedback on the entire application.

## • **Implementation**

### 1. Overview

Overall, Pocket Manager uses market tested and solid components for the application implementation. On a high level the application uses a combination of Ionic on the Client side and Mongo to store data on the server side.

### 2. Mobile Client

On the application side the Ionic Framework was the foundation for the application. Within Ionic, HTML5 was utilized for page views and AngularJS was used for the control mechanisms. The combination created a easy to maintain platform which can interface with hardware on a wide variety of devices without having to maintain a complex set of Android libraries. Since only a few hardware components are expected to be utilized, the hybrid application allows the application to run on Android and iOS with one core set of code controlling the app.

For adding hardware interfaces, the NPM and Cordova were utilized. The camera is currently used for taking pictures of receipts. In the next increment the gallery file picker is expected to be added as an alternative to the camera.

### 3. Database

On the server side, Mongo was selected for the database with m Labs as the host. m Labs provided free storage for our application and the Mongo provides better performance than a traditional relational database. In addition, app development experts recommended using Mongo over other storage options such as Firebase



due to the flexibility. Finally, Mongo supports simple REST requests to an API allowing us to start the application with easy to maintain and troubleshoot interfaces.

Our Mongo database on mLabs has the following connection information. (Table 6-9) Initially, one collection was configured to collect receipt information.

**Table 6-9.** Mongo Database Info

API - [https://api.mlab.com/api/1/databases/pocket\\_manager/collections/](https://api.mlab.com/api/1/databases/pocket_manager/collections/)

Collection - pocket\_manager/

ApiKey - Omq-HhXv0WUnDNEVey9TQdBhsEEFDtHo

Session storage was also utilized for the initial expense collection to improve performance by minimizing the number of REST requests. In addition, the session storage allows information to be save in case the data connection is lost. Finally, the session storage gives the user a chance to confirm the information before it is sent to the database.

#### 4. Services

Google Vision was utilized to read information from receipt pictures.

#### 5. Going Forward

Long-term, Ionic will continue to be the application framework. An additional Mongo collection for user logins is expected to be added. Finally, Google Sign In may be utilized to keep the user from having to remember too many sets of login credentials.

### • **Deployment**

#### 1. Overview

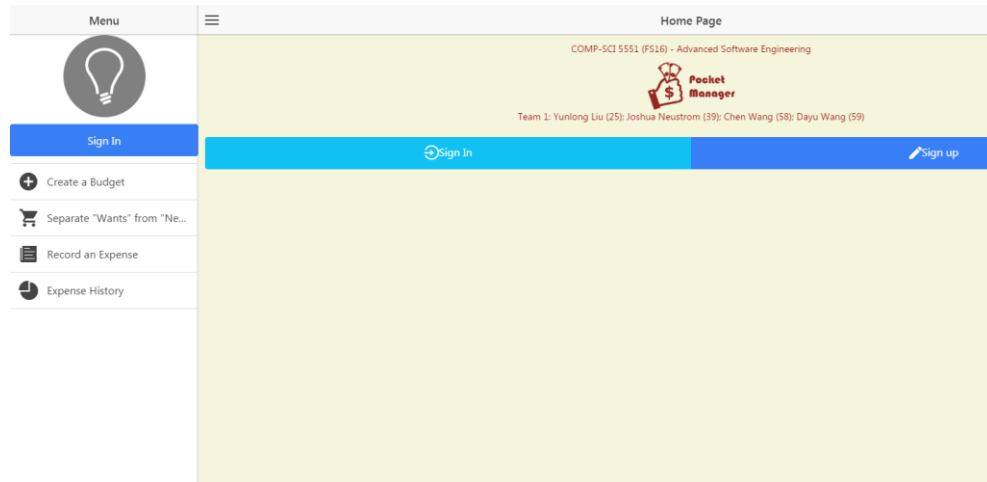
Deployment includes leading the app onto devices, taking screenshots with detailed descriptions, and creating a wiki in GitHub. The overall purpose of the section of the process is to deliver the completed application to market (or in the case of the third increment, the teachers). With the fast pace of technology and the high cost associated with employing highly skilled developers, delivering a product early is critical for a technology startup. Deployment moves the application from the developer world to the real world of users. Both data and revenue from customers can be obtained providing critical fuel for future increments.

The deployment of Pocket Manager's third increment focused enhancing each major application tab's functionality. The improvements focused on a login page, registration page, wants differentiation page, budget creation, and expense recording.

#### 2. Screenshots

Below are the primary screenshots for the application along with the detailed descriptions.

##### a. Login Screen (Figure 6-10)



**Figure 6-10.** Login screen.

The initial login screen provides the users first interaction with the application. A basic username and password is requested from the user which is sent to a Mongo DB collection to see if the credentials are valid. If the combination is not valid, the user is warned and prompted again. If the combination is valid, the user is brought to the main welcome page.

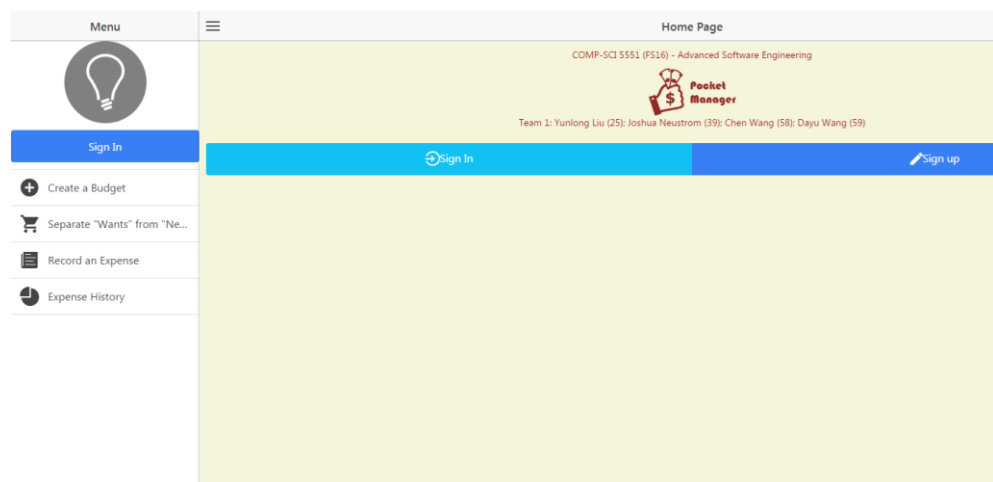
For branding purposes, the Pocket Manager logo is central to the page creating a visual association in the users mind. The logo also allows for a user to know if they are using the real Pocket Manager Application.

At the head of the page is the basic group information. Although such information would be unlikely to be included in a real world app in the Android market, the title was included to help the graders and teachers easily identify the project and members.

If the user does not have an account, they can select the registration link.

Going forward, new features are expected to be added such as request help and Google Single Sign On buttons.

b. Main Menu (Figure 6-11)



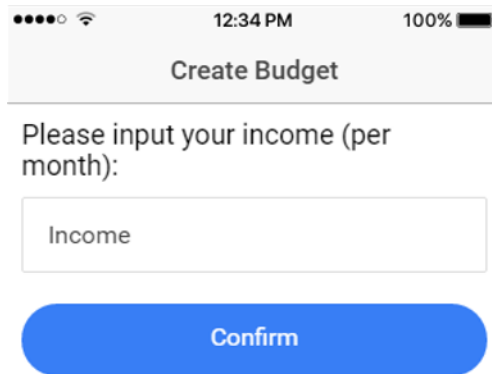
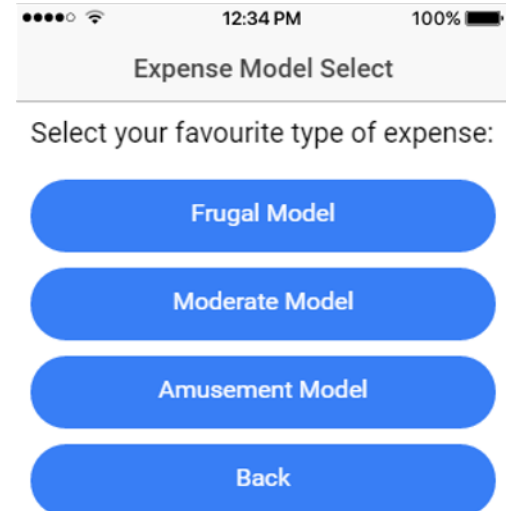
**Figure 6-11.** Main Menu Screen.

The main menu screen provides the user with access to all of the main features of the application and serves as a one stop shop for the user. At the top of the list is a button for budget creation which allows a user to create a budget for spending including the top level categories of transactions and the max spending each area. Further down, a button exists for query of an item to see if the item is a luxury or necessity. The function is the key advantage of the Pocket Manager which allows a user to cut spending by having an outside source label the potential purchase as a luxury if it is not truly needed for the budget. Continuing down the list, the screen has a button for adding actual expenses in the context of what was originally budgeted. The bottom button takes the user to a big picture view of their transaction history to allow them to understand their spending habits and success in meeting the budget over the long term.

The overall look and feel from the initial login page was maintained in the main menu section to create a consistent interface for the user. Buttons respond in a similar manner with two arrows appearing when touched or hovered over by the user. Furthermore, the key project information is kept on the title section of the page.

c. Create a Budget (Figure 6-12)

d.

e.

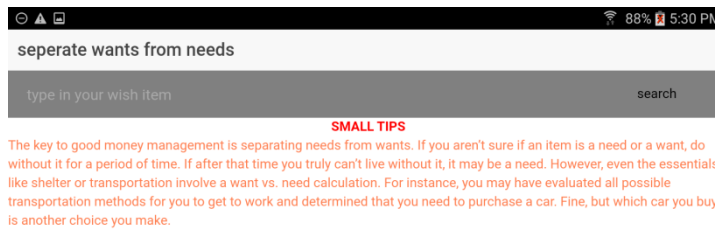
**Figure 6-12.** Budget Creation.



Budget creation creates broad categories to characterize expenses being recorded and sets spending thresholds for the user. By breaking down the expenses into individual categories, the user takes their overall goal in smaller bite size pieces. In addition, the categories provide some additional information that can help to characterize needs vs wants.

The budget allows the user to pick a custom amount of income per based on what the user expects to earn.

For convenience, a user has several models of expenses which can be selected. The models include Frugal, Moderate, and Amusement. Based on the model selected and income entered, a set of expense categories and amounts will be created for the user automatically. The models can help setup the budget for a user that lacks experience in building their own budget.



---

f. Separate Wants from Needs (Figure 6-13)

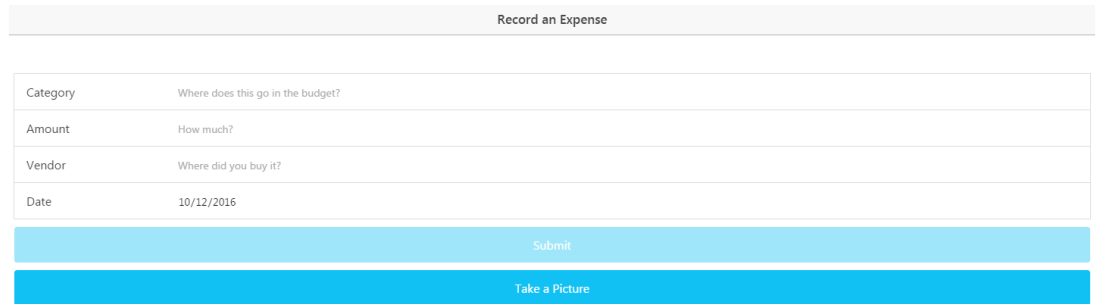
**Figure 6-13.** Wants vs Needs.

The wants vs needs section is one of the true value added aspects of our app which we like to call our secret sauce. Overall, the feature helps the user make smart decisions about spending. A

user can submit a potential expense and the system tells them if it truly vital to their student life. The feature can use external APIs such as Ebay in combination with in house code to smartly determine if the user truly needs an item.

The final increment will continue to build on this helpful feature by creating even more intelligent categorizations of a potential expense.

g. Record An Expense (Figure 6-14 - 6-16)

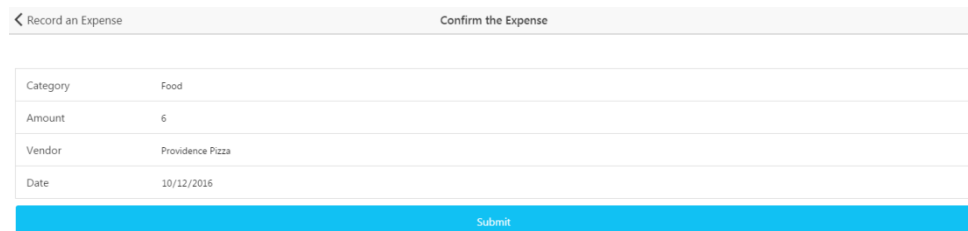


Record an Expense	
Category	Where does this go in the budget?
Amount	How much?
Vendor	Where did you buy it?
Date	10/12/2016
Submit	
Take a Picture	

**Figure 6-14.** Record an Expense.

The initial expense recording page has the option to manually enter on money they have spent including the budget category, amount, vendor, and date. All fields are required and today's date is populated into the form by default. If any field is touched and left blank, the system will warn the user. Furthermore, the amount field requires a valid number. Both dates and amounts have broad limitations on their allowable entries to protect the user from unintended entries due to the small mobile keyboard. Data from the form is stored locally to speed up performance and allow the data not to be lost if the connection is down.

A user can also take a picture of the receipt and the app will read the text off the image using Google Vision. Afterwards, the software can typically find the date and vendor name in the text and store them in session storage. Afterwards, the user is taken to a form where they can confirm or edit the information before it is sent to mLabs for long-term storage.

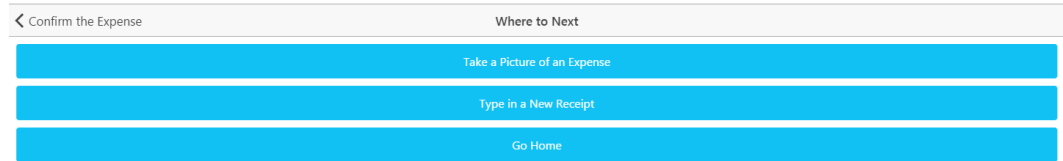


Confirm the Expense	
Category	Food
Amount	6
Vendor	Providence Pizza
Date	10/12/2016
Submit	

**Figure 6-15.** Confirm an Expense.

Going forward in increment four, the receipt reading capability via camera is expected to be upgraded so they total and category can be extracted from the text.

The next screen takes a user to a version of the form populated by their entries from the session storage. The check allows users to check data before it is sent to the remote database. The same basic form data checks are implemented from the initial expense screen. Upon submission, the data is sent to a remote Mongo Collection. Within the collection, the data is parsed to a string.



**Figure 6-16** Next Expense.

For faster data entry, the user is taken to a screen where they can choose how to enter more data. The user can choose the option of receipt picture or typing an expense. Otherwise, the user can go back to the home screen of the application.

- h. Expense History (Figure 6-17) - **To be finished, no done yet.**

**Figure 6-17.** Expense History.

This section of the application is currently a shell providing a basic user interface with no actual functionality. In future increments, expense data will be added to the database allowing for the creation of both expense monitoring and reporting for the user. This section is planned for the final increment because the expense recording and supporting database were only recently setup.

### 3. Wiki Page

A Wiki page was created for Increment 3 which recursively includes the report. The page allows an outside user to easily understand the deliverable included in our Github source folder. In addition, the GitHub contains a source folder for all finalized code that was part of the deliverables. Furthermore, the main screenshots from the report and a copy off the report itself can also be found in the documentation folder on the GitHub site. Finally, a readme exists in the core section of our repository with the key links and the overall project info.

- Wiki – [https://github.com/dwk894/CS5551FS16\\_Pocket\\_Manager/wiki/Increment-3](https://github.com/dwk894/CS5551FS16_Pocket_Manager/wiki/Increment-3)
- Documentation-  
[https://github.com/dwk894/CS5551FS16\\_Pocket\\_Manager/tree/master/Documentation](https://github.com/dwk894/CS5551FS16_Pocket_Manager/tree/master/Documentation)
- Source Code - [https://github.com/dwk894/CS5551FS16\\_Pocket\\_Manager/tree/master/Source](https://github.com/dwk894/CS5551FS16_Pocket_Manager/tree/master/Source)
- Readme-  
[https://github.com/dwk894/CS5551FS16\\_Pocket\\_Manager/blob/master/README.md](https://github.com/dwk894/CS5551FS16_Pocket_Manager/blob/master/README.md)

#### 4. Going Forward

The last increment is expected to focus on integrating the different parts of the application into one unified application providing a seamless experience for the user. Each increment will have a separate Wiki page and the overall source and documentation structure will be maintained.

#### • Contribution Table

Name (ID)	Contribution
Yunlong Liu (25)	<ul style="list-style-type: none"> <li>• &lt;code&gt; Separate Wants from Needs (<b>Update</b>)</li> <li>• &lt;documentation&gt; System UML class diagram (<b>Update</b>)</li> <li>• &lt;documentation&gt; Activity diagram for feature <i>Separate Wants from Needs</i> (<b>Update</b>)</li> <li>• &lt;documentation&gt; UML sequence diagram for feature <i>Create a Budget</i> (<b>Update</b>)</li> <li>• &lt;report&gt; Existing services/APIs (<b>Update</b>)</li> </ul>
Joshua Neustrom (39)	<ul style="list-style-type: none"> <li>• &lt;code&gt; Record Expenses (<b>Update</b>)</li> <li>• &lt;documentation&gt; System Architecture (<b>Update</b>)</li> <li>• &lt;documentation&gt; Manual Expense Recording (<b>Update</b>)</li> <li>• &lt;documentation&gt; Mongo setup, test, and initial data submission</li> <li>• &lt;documentation&gt; Wiki page creation (<b>Update</b>)</li> <li>• &lt;report&gt; Testing, implementation, and deployment (<b>Update</b>)</li> </ul>
Chen Wang (58)	<ul style="list-style-type: none"> <li>• &lt;code&gt; Create a Budget (<b>Update</b>)</li> <li>• &lt;report&gt; Introduction (<b>Update</b>)</li> <li>• &lt;report&gt; Objectives (<b>Update</b>)</li> <li>• &lt;report&gt; Features (<b>Update</b>)</li> </ul>
Dayu Wang (59)	<ul style="list-style-type: none"> <li>• &lt;code&gt; System Shell Model (<b>Update</b>)</li> <li>• &lt;documentation&gt; Wireframe/Mockup Views (<b>Update</b>)</li> <li>• &lt;report&gt; Detail Design of Features (<b>Update</b>)</li> <li>• &lt;report&gt; Final formatting and submission (<b>Update</b>)</li> </ul>



- **Bibliography**

- [1] Project on Student Debt, institution: The Institute for College Access & Success  
<http://ticas.org/posd/map-state-data-2015>
  - [2] The Take Charge America Institute  
<https://tcainstitute.org>
  - [3] <https://cloud.google.com/vision>
  - [4] <https://docs.mongodb.com/manual>
  - [5] <http://www.highcharts.com/demo>
  - [6] [http://www.nyu.edu/classes/jcf/g22.3033-007/slides/session2/g22\\_3033\\_011\\_c23.pdf](http://www.nyu.edu/classes/jcf/g22.3033-007/slides/session2/g22_3033_011_c23.pdf)
-