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 4

(Due Dec 5th, 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.

- **Specific Objectives**

2.1. <u>Overall objective</u>

Based on the situation, the Pocket Manager will give an effective plan and management for the expense of college student. The user can create a budget for this month, then they can record their receipt with input and the camera. The record and analyses can be checked at any time. After one month, the user can use the app to look over and manage their expense. They will know which part of expense is largest and which parts of expense are exceeding the limit in the plan. Then they can improve the plan and optimize the way of their expense. It even can optimize the expense habit of the college student. Students can make different plan based on their income and major, and record save each expense easily, if their expense in one part is almost exceed the limit in the plan, Pocket Manager will give an alert for the user. The user can check the expense status in each part and plan their future expense at any time.

2.2. <u>Specific Objectives</u>

Based on the researching for this area and the habit of the college students' expense, there are four important problems always confused college student:

Problem 1 – Always splashed at the beginning of the month but starving at the end of the month

A lot of college students are very exciting when the salary has been issued, they always feel they have a lot of money and ignore the small expense. They have meal at the restaurants, shopping mindlessly, and buy a lot of things which is nonessential. Then the money will be wasted quickly in one or two week at the beginning of the month. A matter of course, they will very poor and have to very frugality at the end of the month. They know that but they don't know how to prevent it happens.

Solution – Create budgets

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.

Problem 2 – Credit card abuse always occurs

Because the credit card is more and more popular in the world, the overdraft is very useful and full of temptation for the college student. A lot of student have different kind of credit card and overdraft them excessive. The money in the credit card is much easily to waste than the cash.

Solution – 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.

Problem 3 - Relying on credit card bills as expense record and so lazy to write down every expense.

With the more and more useful and clear bill on the credit cards, student do less and less the statistics and management about the expense. It's easily to confuse because of various credit card and bills.

Solution – Expense input and record

Pocket manager can record the input of each expense, and it also can help the user to save the receipt through the camera. The recorded expense will help the user manage their plan more effectively.

Problem 4 - "I kept being frugal, but why and where did I spent such a large amount of money?"

This problem we've mentioned above: college students always pay attention to the large amount of expense but ignore the small expense. Then the money will be wasted unconsciously.

Solution - "Grading" of your last month performance in money management.

After the user use the Pocket Manager for one month, the user will get a grade for his expense, then he can improve his plan and habit to manage his expense.

- **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 a model of expense for the user. They give a 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 budget. It can show all the details about the individual budget. User can have a clear mind about the plan of this month.

- **Existing Services/API**

Highcharts

Highcharts is an application that we can use it to create interactive charts. All the SaaS projects, web applications, intranets and websites can use it to display their output. It is an open source API so that the user can check and change the function and other features about it. It uses the HTML5 and give a platform to communication, which is on the GitHub and users can fork them and participate in the discussions. It supports a lot kinds of charts such as basic line charts, area charts, column charts, bar charts, pie charts, scatter charts, bubble charts, combination charts, dynamic charts, 3D charts, gauge charts, heat charts, tree charts and so on.

For the expense management system, the budget is displayed as graph for users, then the Highcharts are very important to show the budgets of users, the budgets are showed as a 3D-pie-dynamic graph in our project.

Ionic InAppBrowser

Ionic InAppBrowser is a plugin that can help user to view web browser in applications, it supports an effective way to open images, access to web browser and web pages in Apps, the PDF files can also be open through this plugin.

In our projects, there is a function which is separate the needs from wants, this part will let user input some information about the things they want to buy, and it will return some details about goods that user wants to buy, and a suggestion for this purchase will be given, then the user can decide if this good is necessary or just a wants at last based on the description and the suggestion. eBay API has been used, then the eBay link of this good will be supported on device, if the user wants to see some more details about this good or decide to buy this item, they can click this link easily then go to the browser to make a decision. The ionic InAppBrowser are used to open the browser and link the URL.

eBay API - an API that can make users access to eBay shopping information

The eBay APIs provide programmatic access to eBay marketplaces. It enables third parties' users to build web applications, allows developers access to eBay marketplaces in new ways. eBay provides particular services point at shopping, trading and finding products or seller information.

Usually, the request interface as XML files, each request is composed of XML elements that specify the request parameters. For example, shopping API find items by keyword, and sort by best match, the other way, find products find items also by keyword, but only the description nothing about the selling information.

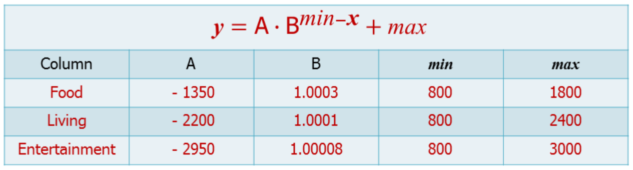
Since, we have an application part called Separate Wants from Needs, in this part, we try to record what's users want and add it to the calling string, get the JSON file including Item ID, description, pictures, best match prices. These are what we did about this part during second increment. Next step we will analyze the result and try to provide whether it's right or not to buy it right now, if not, try to generate an estimated future time to buy it.

MLab (MongoDB) - a free and open-source cross-platform document-oriented database application

MongoDB is a free and open-source cross-platform document-oriented database program. It is based on the NoSQL database program, avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, then the integration of data in certain types of applications easier and faster. Mongo DB is an open-source, document database designed for ease of development and scaling. The Manual introduces key concepts in Mongo DB, presents the query language, and provides operational and administrative considerations and procedures as well as a comprehensive reference section. Using the database, our expenses recommend and manage system can save a lot of dataset about the users' in it. First of all, the users' account an information of the system will be stored in the database, and they can choose a reasonable plan, input their expense with receipt, and the details of their expense and plan. All of them will be stored in the Mongo DB.

New Services

For the budget generation, we gave up the initial design which is separate the budgets as the frugal model, moderate model and amusement model. Because in this way, some clash will appear at the joints of those models. Then we chose a more reasonable budget generation which is fitting curves. We gave a reasonable formula which is displayed at the Figure 4-1.



$$y = \mathrm{A} \cdot \mathrm{B}^{min-x} + max$$

Column	A	B	*min*	*max*
Food	- 1350	1.0003	800	1800
Living	- 2200	1.0001	800	2400
Entertainment	- 2950	1.00008	800	3000

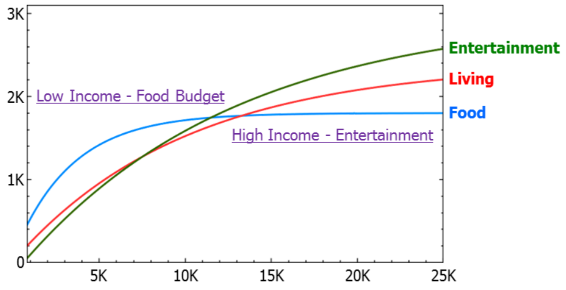
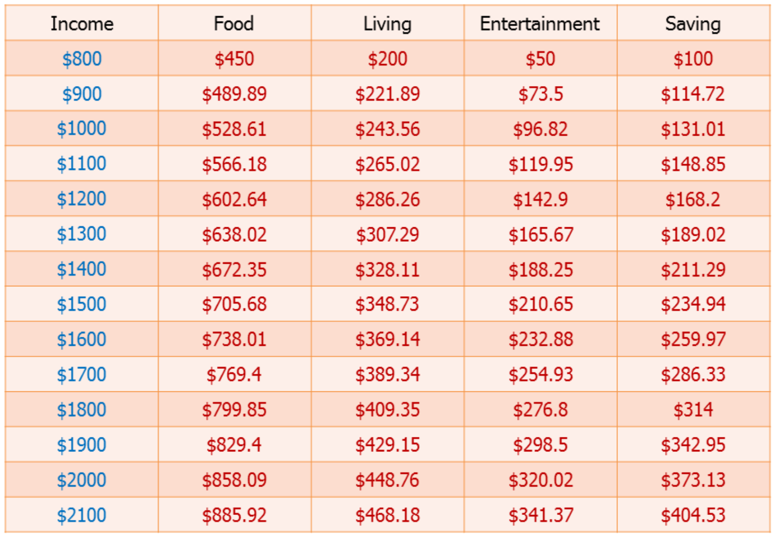


Figure 4-1. The core philosophy of the budget generation, the form upside shows the formula of the expense aspects, the downside graph shows the trend of all the aspects with increasing income.

At our method, for the college and university students, the most important parts of their expense are food expense, living expense, entertainment expense and savings. And if a student's income is very limited, then the food and living expense are the most important parts because we must ensure that users can keep their basic living. And with the increasing of users' income, more and more money can be used at entertainment and savings, in all conscience, the food and living expenses also will be increased with the income increasing, but they will not increase so observably then they can allocate the remained money to other aspects. It is fitting curves for those expenses with the increasing of income. As the above figure displayed, all the expense will approach gentle. The max of those three parts are: food is 1800, living is 2400, and entertainment is 3000. So the 1800, 2400, 3000 are the asymptote line of those parts. And the other parts of the formula are used to control the initial expense and the slope of the curves. The final curve can support a reasonable and effective budget based on users' income and help users to correct their habits of expenses. Table 4-2 displayed the detailed expenses on food, living, entertainment and saving.

Table 4-2. Detailed expenses on food, living, entertainment and saving with different incomes.



Income	Food	Living	Entertainment	Saving
$800	$450	$200	$50	$100
$900	$489.89	$221.89	$73.5	$114.72
$1000	$528.61	$243.56	$96.82	$131.01
$1100	$566.18	$265.02	$119.95	$148.85
$1200	$602.64	$286.26	$142.9	$168.2
$1300	$638.02	$307.29	$165.67	$189.02
$1400	$672.35	$328.11	$188.25	$211.29
$1500	$705.68	$348.73	$210.65	$234.94
$1600	$738.01	$369.14	$232.88	$259.97
$1700	$769.4	$389.34	$254.93	$286.33
$1800	$799.85	$409.35	$276.8	$314
$1900	$829.4	$429.15	$298.5	$342.95
$2000	$858.09	$448.76	$320.02	$373.13
$2100	$885.92	$468.18	$341.37	$404.53

- **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.

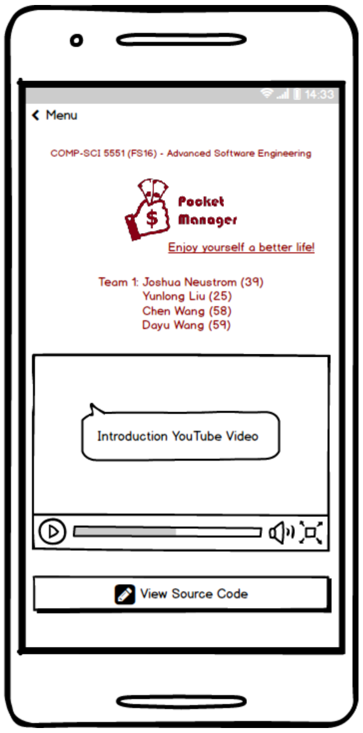

14:33
Menu
COMP-SCI 5551 (FS16) - Advanced Software Engineering
Pocket Manager
Enjoy yourself a better life!
Team 1: Joshua Neustrom (39)
Yunlong Liu (25)
Chen Wang (58)
Dayu Wang (59)
Introduction YouTube Video
View Source Code

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.

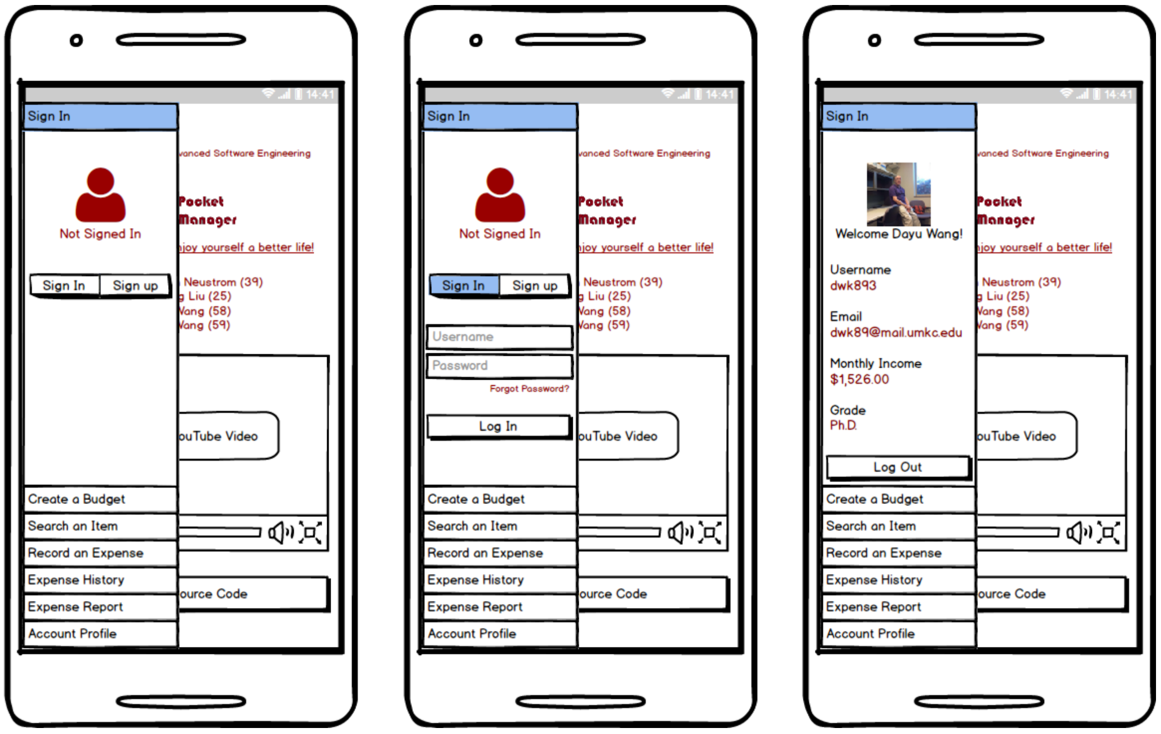


Figure 5-2. Mockup pictures of the login part of the *Pocket Manager* system. The left picture reveals the initial appearance and the state when the user has been logged out. The middle picture is when the user is attempting to login. The right picture demonstrates the view when the user has been successfully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

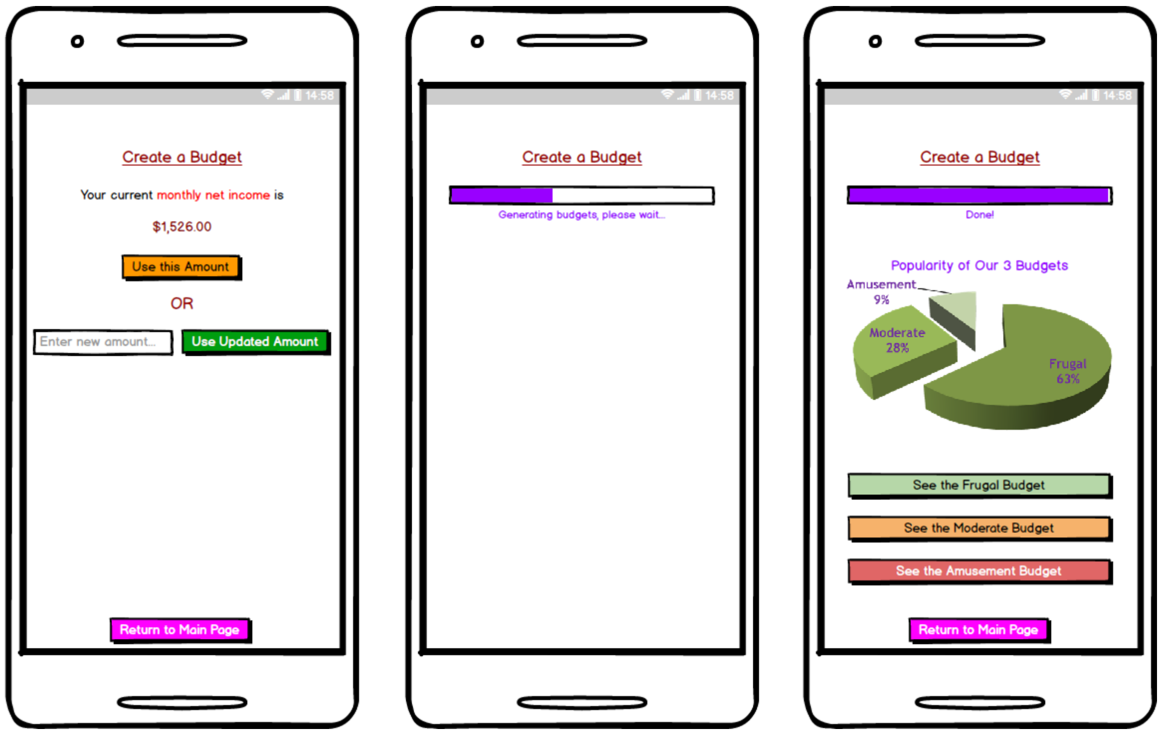


Figure 5-3. Mockup pictures of the feature of *Create a Budget* in the *Pocket Manager* system. The left picture is the initial view when the user chooses the feature. It asks the user to confirm his/her monthly income or update his/her monthly income before the system can calculate budgets for the user. 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.

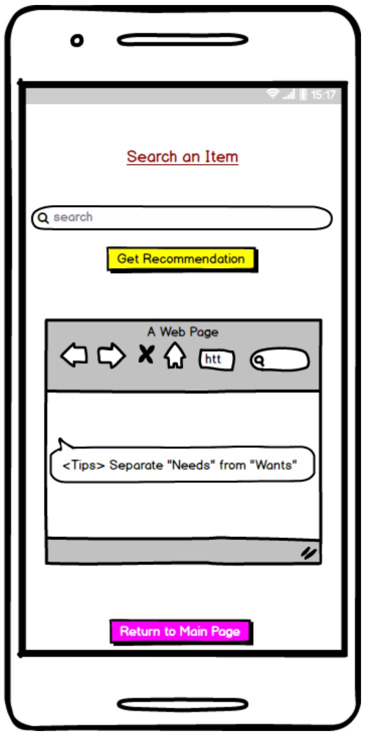


Figure 5-4. Mockup picture of the principal view of the feature of *Search an Item* in the *Pocket Manager* system.

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.

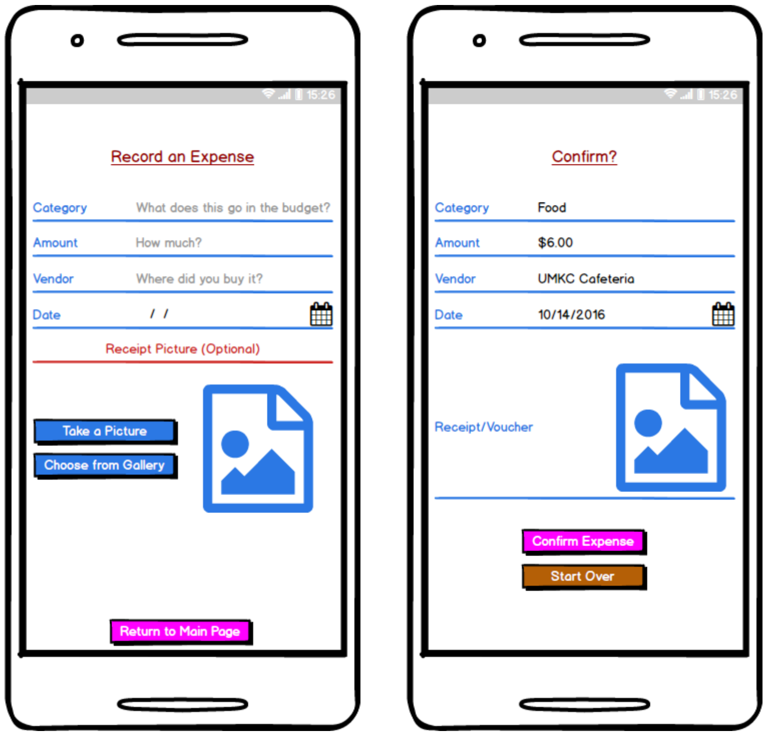


Figure 5-5. Mockup pictures of the *Record an Expense* feature in the *Pocket Manger* system. The left picture is the view when the user is inputting an expense. The right picture is the confirmation page appeared right after the user's input.

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.

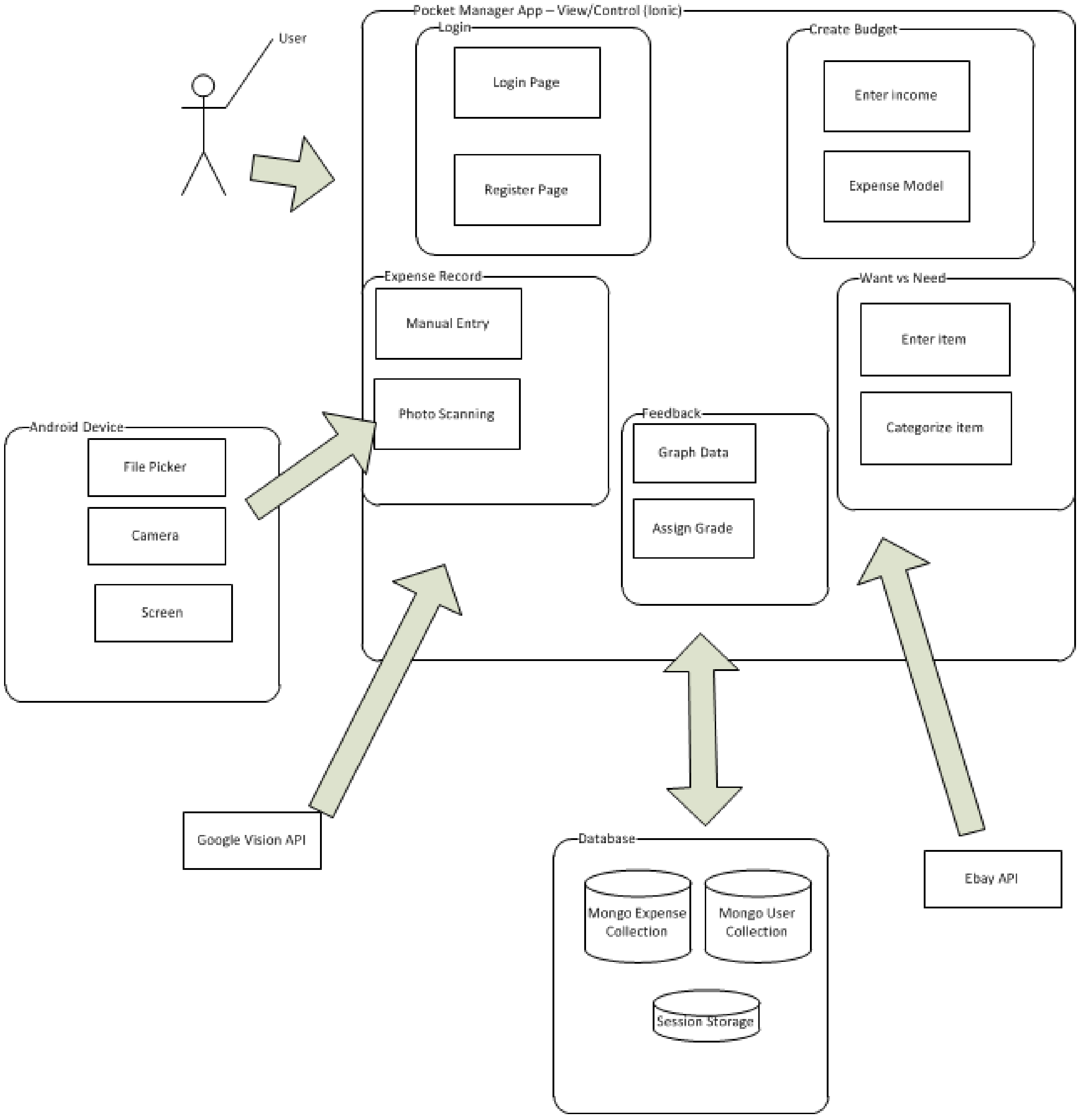


Figure 5-6. System architectural designation of the *Pocket Manager* to be developed.

- **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 17 is the result of the coding audit for our system.

Table 17. 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

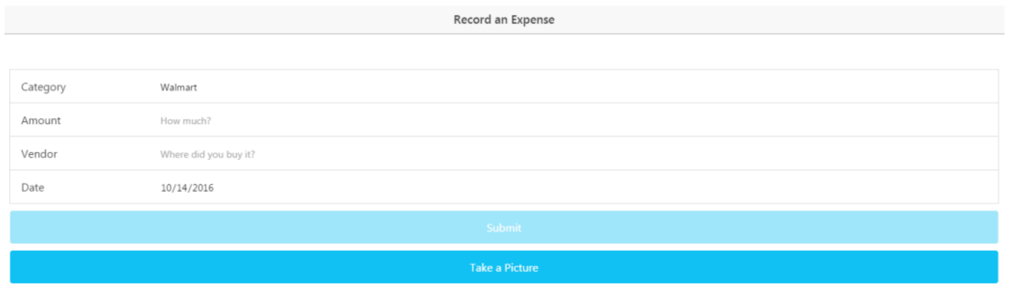
Item	Feedback	Response
a.	Were major issues submitted to GitHub?	No major issues
b.	Do you have any major concerns about your code?	Lack of Standardization and Integration

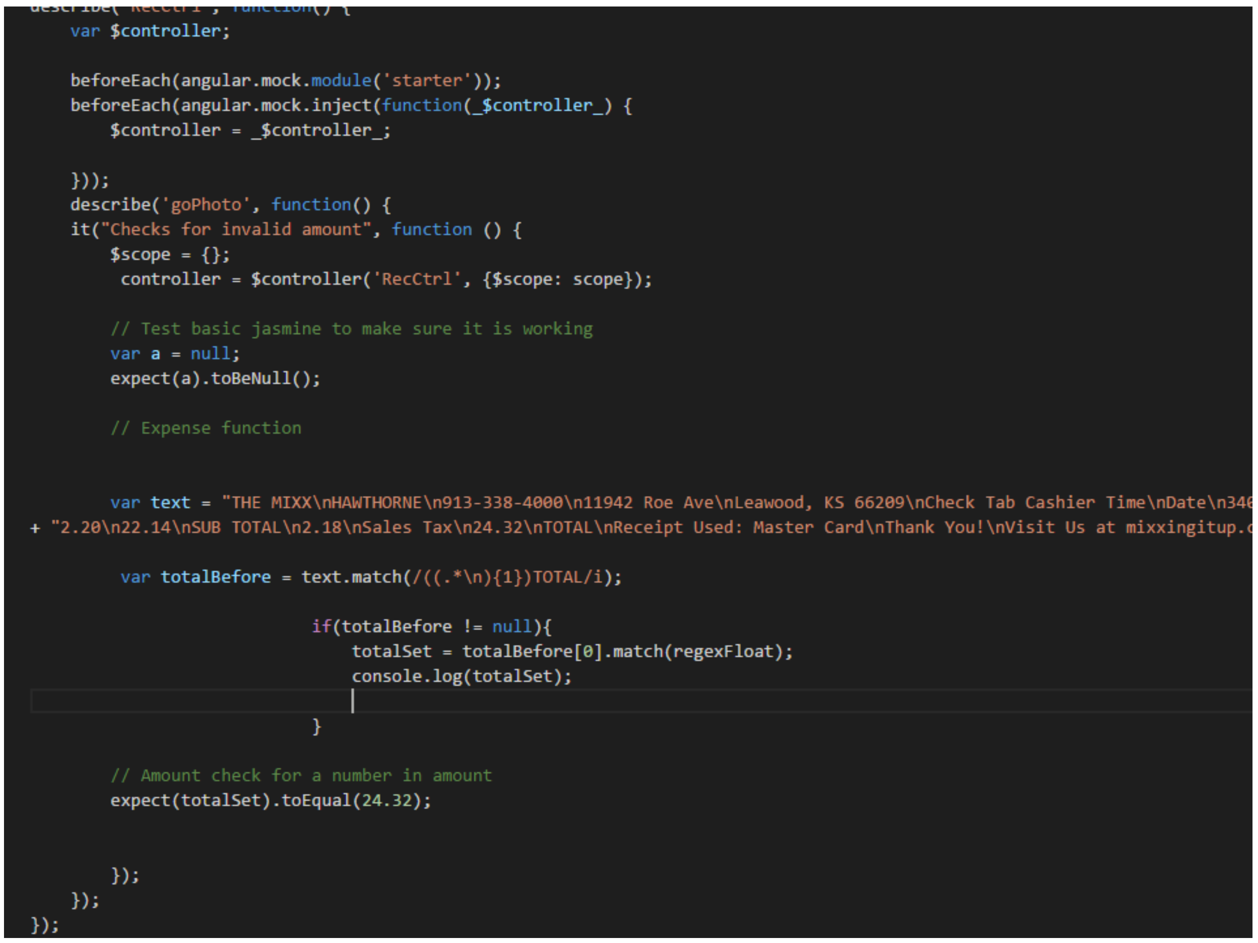
3. Unit Tests

Unit testing were tested by building small snippets of code to test sample inputs against expected outputs. Per our TA’s recommendation, a major focus of our development was on extracting accurate expense information from the receipt photo. As a result, the unit 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 – Find Number After Total (**Figure 1**)

Figure 1. Number after Total Unit Test





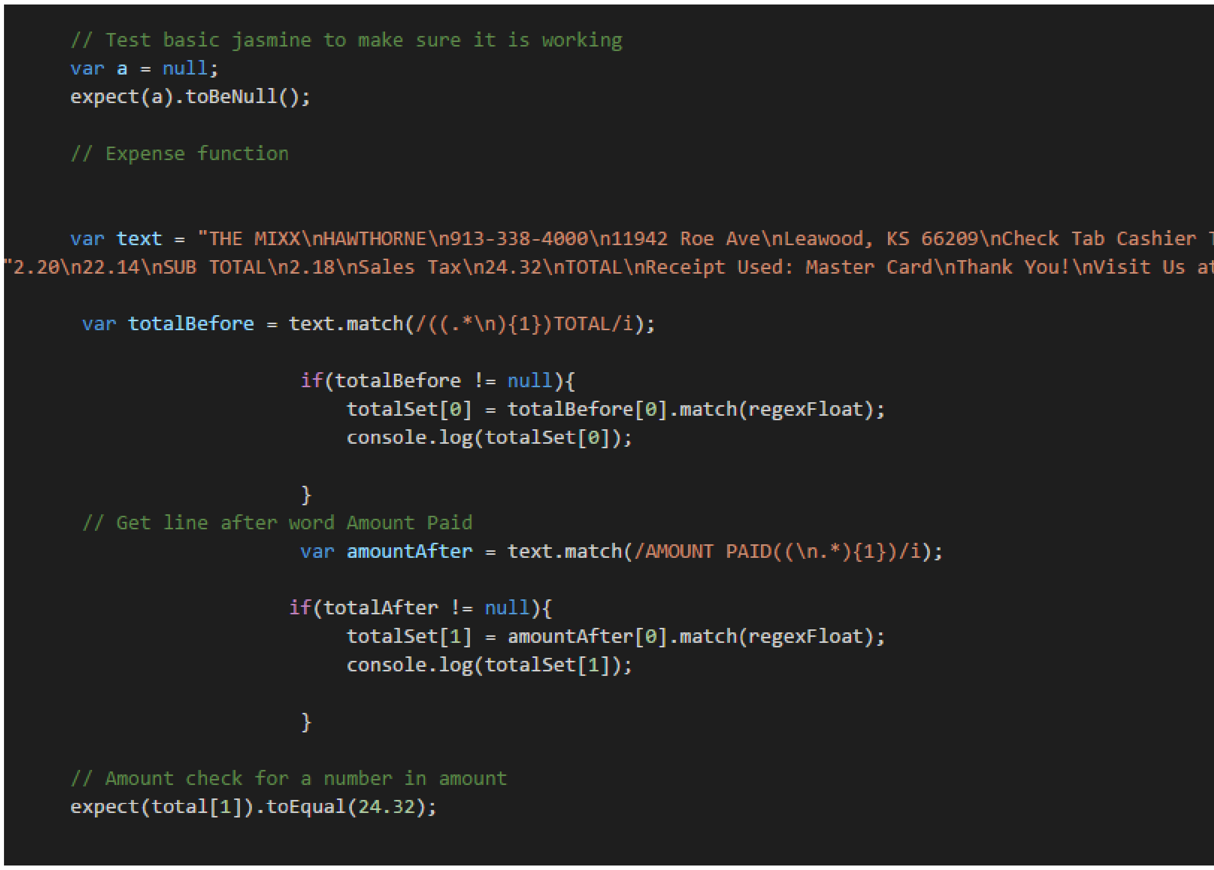
```
describe('RecCtrl', function() {
    var $controller;

    beforeEach(angular.mock.module('starter'));
    beforeEach(angular.mock.inject(function(_$controller_) {
        $controller = _$controller_;

    }));
    describe('goPhoto', function() {
    it("Checks for invalid amount", function () {
        $scope = {};
         controller = $controller('RecCtrl', {$scope: scope});

        // Test basic jasmine to make sure it is working
        var a = null;
        expect(a).toBeNull();

        // Expense function


        var text = "THE MIXX\nHAWTHORNE\n913-338-4000\n11942 Roe Ave\nLeawood, KS 66209\nCheck Tab Cashier Time\nDate\n346
+ "2.20\n22.14\nSUB TOTAL\n2.18\nSales Tax\n24.32\nTOTAL\nReceipt Used: Master Card\nThank You!\nVisit Us at mixxingitup.c

         var totalBefore = text.match(/((.*\n){1})TOTAL/i);

                            if(totalBefore != null){
                                totalSet = totalBefore[0].match(regexFloat);
                                console.log(totalSet);
                                |
                            }

        // Amount check for a number in amount
        expect(totalSet).toEqual(24.32);


        });
    });
});
```

A test function was created to see if the feature worked. The function used text from previously scanned receipts to allow many receipt formats to be tested quickly. A number of regex expressions were tried finally settling on extracting the entire line before the target word.

The extracted line could be compared to the Total Amount on the actual receipt. After updating the code, the number would be extracted

b. Unit Test 2 – Get Number After Total (**Figure 2**)

Figure 2. Number After Total Code



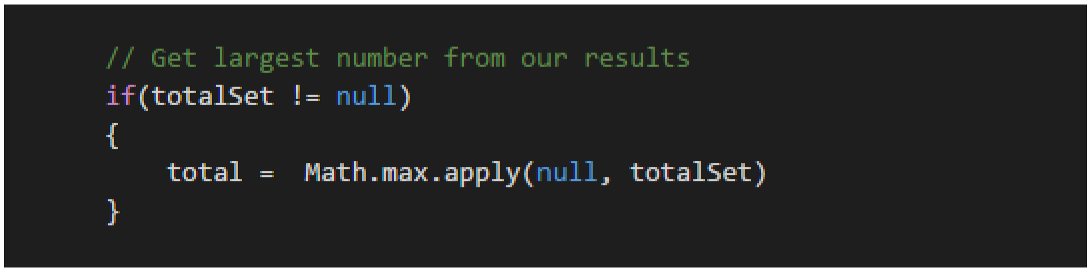
```
// Test basic jasmine to make sure it is working
var a = null;
expect(a).toBeNull();

// Expense function


var text = "THE MIXX\nHAWTHORNE\n913-338-4000\n11942 Roe Ave\nLeawood, KS 66209\nCheck Tab Cashier T
"2.20\n22.14\nSUB TOTAL\n2.18\nSales Tax\n24.32\nTOTAL\nReceipt Used: Master Card\nThank You!\nVisit Us at

 var totalBefore = text.match(/((.*\n){1})TOTAL/i);

                   if(totalBefore != null){
                       totalSet[0] = totalBefore[0].match(regexFloat);
                       console.log(totalSet[0]);

                   }
 // Get line after word Amount Paid
                   var amountAfter = text.match(/AMOUNT PAID((\n.*){1})/i);

                  if(totalAfter != null){
                       totalSet[1] = amountAfter[0].match(regexFloat);
                       console.log(totalSet[1]);

                   }

// Amount check for a number in amount
expect(total[1]).toEqual(24.32);
```

Since a receipt typically contained two columns of text where one contained numbers and the other identifiers. As a result of the format, Google Vision combined the columns and read the text inconsistently. Sometimes the numbers were first and other times they were second.

With the change variations in text, test cases had to be written to account for numbers occurring after the word total. A regex was used to find the line after the word Total. Since the word total can occur multiple times (ie Tax Total, Sub-Total, Total) the results were added into an array. Afterwards, the lines found were compared to the actual numbers on the receipts. After trying many difference formats of regex, the correct number was finally found on the line after total resulting in a passed unit test.

c. Unit Test 3 – Biggest Amount (**Figure 3**)

Figure 3. Biggest Amount



```
// Get largest number from our results
if(totalSet != null)
{
    total =  Math.max.apply(null, totalSet)
}
```

The final tests examined how to handle a situation where many totals were found using in the receipt and stored in the array, but only one total was needed. A single amount was required for storage as the expense total in MongoDB. In order to pass the test, the number selected from the array should match the actual total amount paid as shown on the receipt.

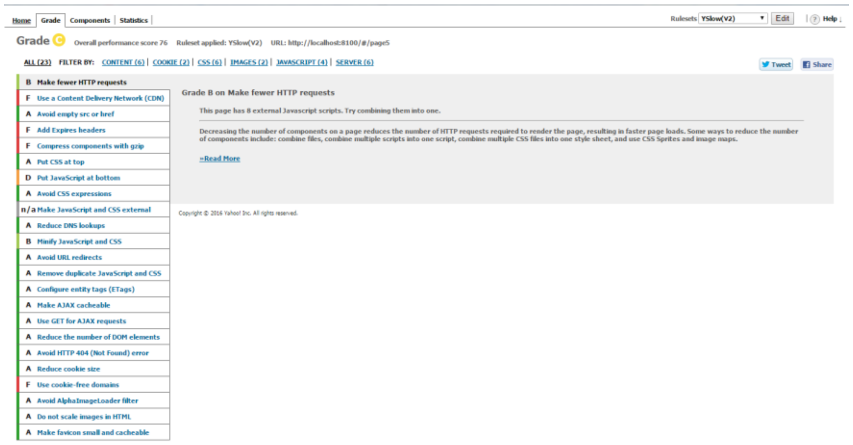
The test was passed once the largest amount was selected from the array. If the customer paid cash, the amount given typically was ignored by the regex searches. As a result, the largest amount found was typically the correct total. Numbers such as subtotal or tax total were smaller so could easily be ignored.

4. Speed Tests (**Figure 4-5)**

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 4. Home Page Speed Test



Grade C Overall performance score 76 Ruleset applied: YSlow(V2) URL: http://localhost:8100/#/page5

ALL (23) FILTER BY: CONTENT (6) | COOKIE (2) | CSS (6) | IMAGES (2) | JAVASCRIPT (4) | SERVER (6)

B	Make fewer HTTP requests
F	Use a Content Delivery Network (CDN)
A	Avoid empty src or href
F	Add Expires headers
F	Compress components with gzip
A	Put CSS at top
D	Put JavaScript at bottom
A	Avoid CSS expressions
n/a	Make JavaScript and CSS external
A	Reduce DNS lookups
B	Minify JavaScript and CSS
A	Avoid URL redirects
A	Remove duplicate JavaScript and CSS
A	Configure entity tags (ETags)
A	Make AJAX cacheable
A	Use GET for AJAX requests
A	Reduce the number of DOM elements
A	Avoid HTTP 404 (Not Found) error
A	Reduce cookie size
F	Use cookie-free domains
A	Avoid AlphaImageLoader filter
A	Do not scale images in HTML
A	Make favicon small and cacheable

Grade B on Make fewer HTTP requests

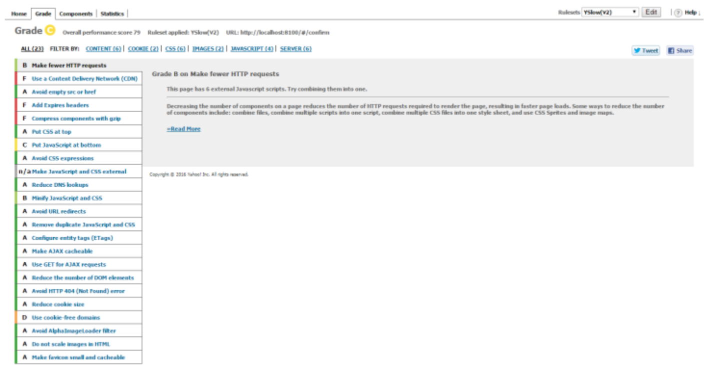
This page has 8 external Javascript scripts. Try combining them into one.

Decreasing the number of components on a page reduces the number of HTTP requests required to render the page, resulting in faster page loads. Some ways to reduce the number of components include: combine files, combine multiple scripts into one script, combine multiple CSS files into one style sheet, and use CSS Sprites and image maps.

»Read More

Copyright © 2016 Yahoo! Inc. All rights reserved.

Figure 5. Rec Record Speed Test.


Home | Grade | Components | Statistics | Rulesets YSlow(V2) Edit | Help

Grade C Overall performance score 79 Ruleset applied: YSlow(V2) URL: http://localhost:8100/#/confirm

ALL (23) FILTER BY: CONTENT (6) | COOKIE (2) | CSS (6) | IMAGES (2) | JAVASCRIPT (4) | SERVER (6)

Tweet Share

B	Make fewer HTTP requests
F	Use a Content Delivery Network (CDN)
A	Avoid empty src or href
F	Add Expires headers
F	Compress components with gzip
A	Put CSS at top
C	Put JavaScript at bottom
A	Avoid CSS expressions
n/a	Make JavaScript and CSS external
A	Reduce DNS lookups
B	Minify JavaScript and CSS
A	Avoid URL redirects
A	Remove duplicate JavaScript and CSS
A	Configure entity tags (ETags)
A	Make AJAX cacheable
A	Use GET for AJAX requests
A	Reduce the number of DOM elements
A	Avoid HTTP 404 (Not Found) error
A	Reduce cookie size
D	Use cookie-free domains
A	Avoid AlphaImageLoader filter
A	Do not scale images in HTML
A	Make favicon small and cacheable

Grade B on Make fewer HTTP requests

This page has 6 external Javascript scripts. Try combining them into one.

Decreasing the number of components on a page reduces the number of HTTP requests required to render the page, resulting in faster page loads. Some ways to reduce the number of components include: combine files, combine multiple scripts into one script, combine multiple CSS files into one style sheet, and use CSS Sprites and image maps.

»Read More

Copyright © 2016 Yahoo! Inc. All rights reserved.

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 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 inherit performance loss and that certain performance issues may remain until the application services are fully hosted remotely.

5. <u>Virtual Device Test</u>

At least two different virtual environments were used in testing

a) Android Studio Virtual Device
b) 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 is 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 18 for the details of our analytical results.

Table 18. 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 leeks)	Yes

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 19).

Table 19. 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, font, change color scheme
c.	Any suggested changes?	Add a nicer background, easier navigation, use a different font on the home page

7. Going Forward

Although the final increment finds the accurate total, vendor, and date on most receipts, future work will greatly reduce the remaining errors. For example, all of the receipt text could be compared to a database of possible vendors to help find the name when it is not the first line of the text. Due to frequent errors in text reading in low light and wrinkled paper, better up front checks could be conducted on picture quality to keep the client from saving a poor quality picture.

Further testing before a real world release should highlight large numbers of students using the application for real world budgets. Both feedback surveys and automatic monitoring could collect data for analysis and future test cases.

- **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 mLabs as the host. mLabs 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 4**) Initially, one collection was configured to collect receipt information.

Table 4. Mongo Database Info

API -	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. Ebay was used for pricing information in determining wants vs needs.

5. Going Forward

If the app was going to market, future work could be added to making login more convenient by adding Google SSO capability.

- **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 final 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 final increment focused on integration of all member's code.

2. Screenshots

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

a. Login Screen (Figure 20)

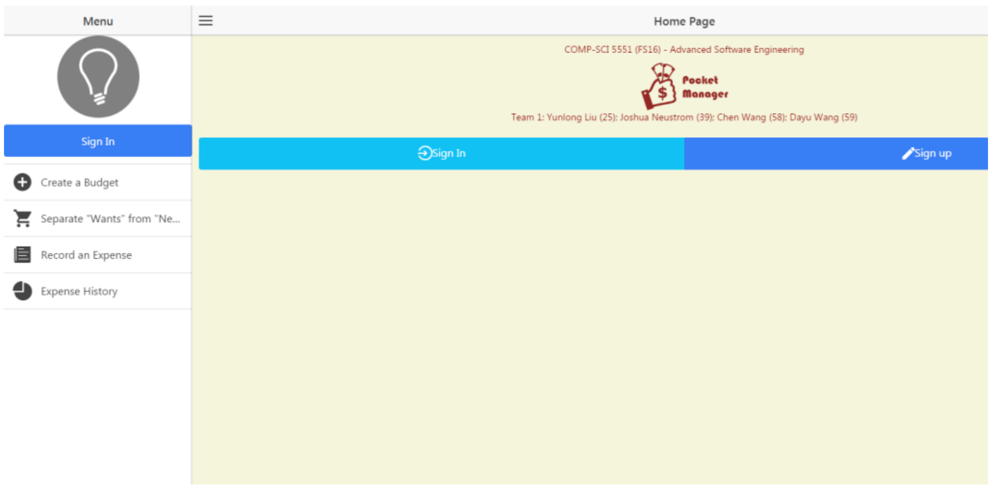

Menu
Home Page
COMP-SCI 5551 (FS16) - Advanced Software Engineering
Pocket Manager
Team 1: Yunlong Liu (25); Joshua Neustrom (39); Chen Wang (58); Dayu Wang (59)
Sign In
Sign In
Sign up
Create a Budget
Separate "Wants" from "Ne...
Record an Expense
Expense History

Figure 20. 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 21)

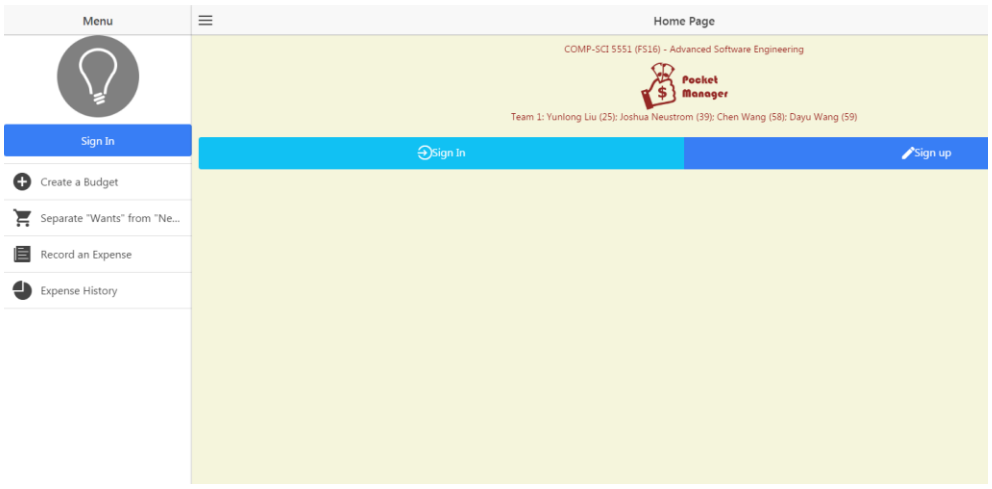
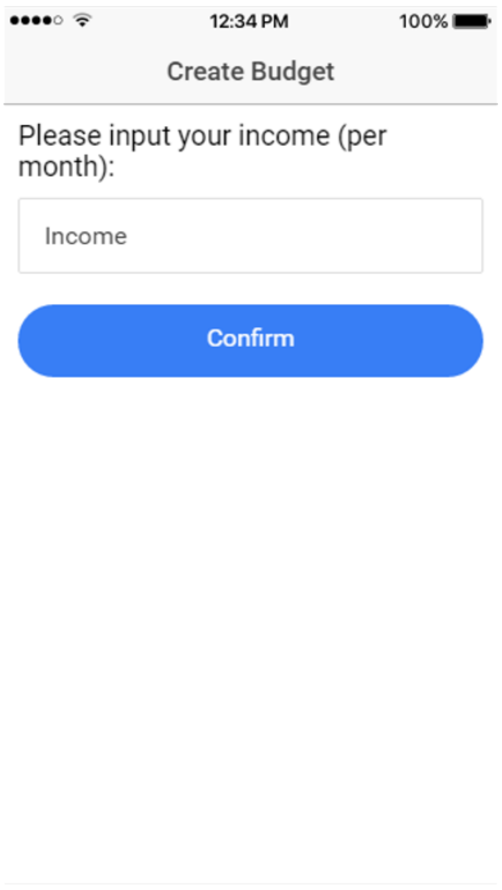

Menu
Home Page
COMP-SCI 5551 (FS16) - Advanced Software Engineering
Pocket Manager
Team 1: Yunlong Liu (25): Joshua Neustrom (39): Chen Wang (58): Dayu Wang (59)
Sign In
Sign In
Sign up
Create a Budget
Separate "Wants" from "Ne...
Record an Expense
Expense History

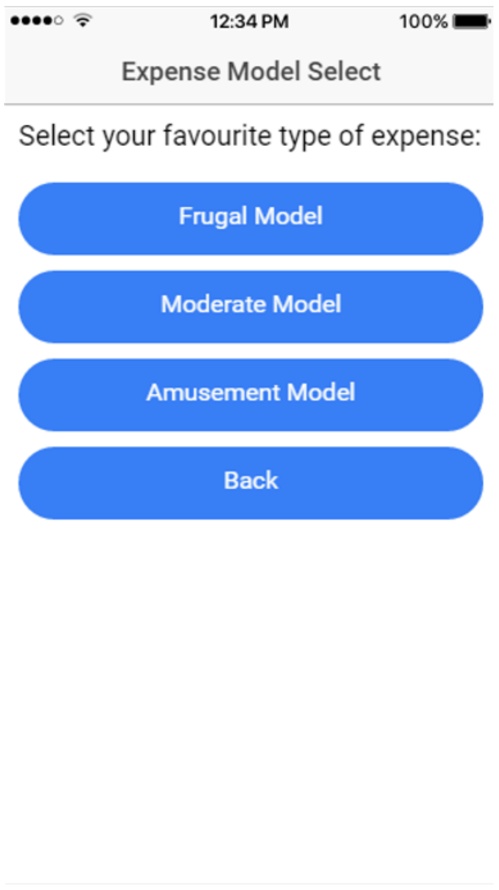
Figure 21. Main Menu Screen.

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 22)





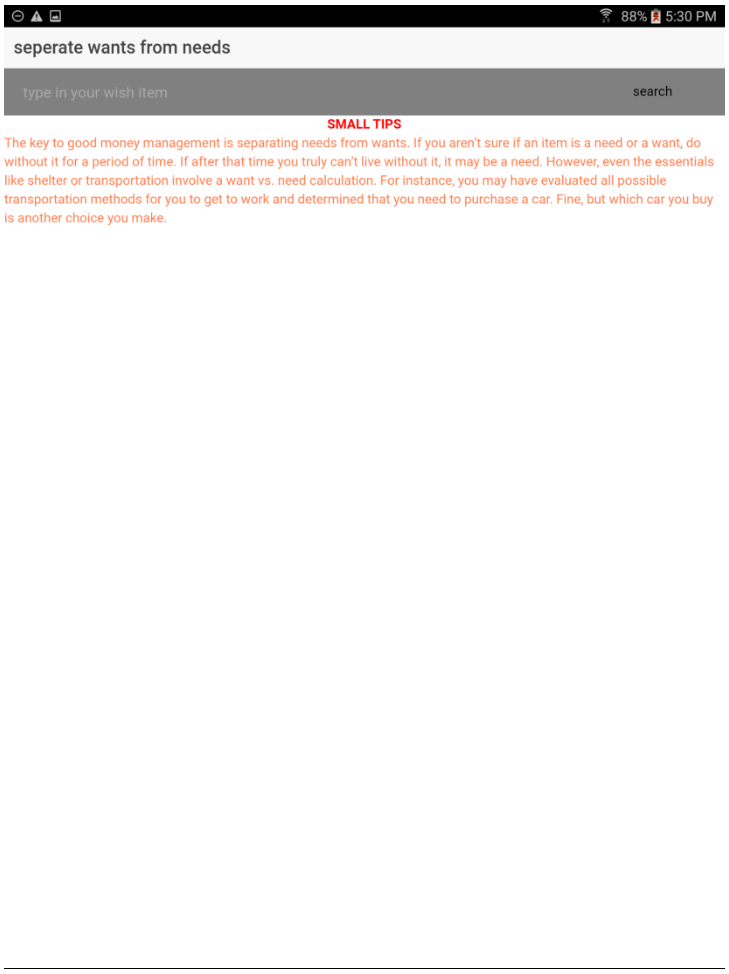
d.

Figure 22. 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.



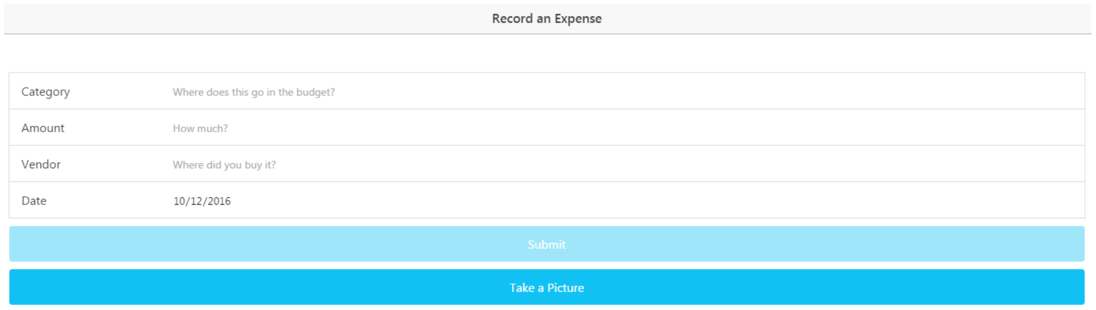
The key to good money management is separating needs from wants. If you aren't sure if an item is a need or a want, do without it for a period of time. If after that time you truly can't live without it, it may be a need. However, even the essentials like shelter or transportation involve a want vs. need calculation. For instance, you may have evaluated all possible transportation methods for you to get to work and determined that you need to purchase a car. Fine, but which car you buy is another choice you make.

e. Separate Wants from Needs (Figure 23)

Figure 23. 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.

f. Record An Expense (Figure 24 - 26)



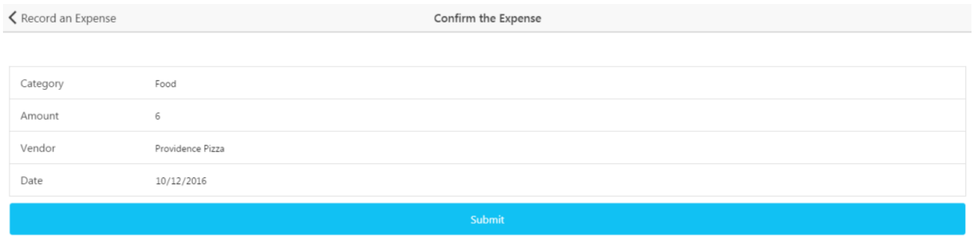
Record an Expense

Category	Where does this go in the budget?
Amount	How much?
Vendor	Where did you buy it?
Date	10/12/2016

Figure 24. 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

Figure 25. 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.

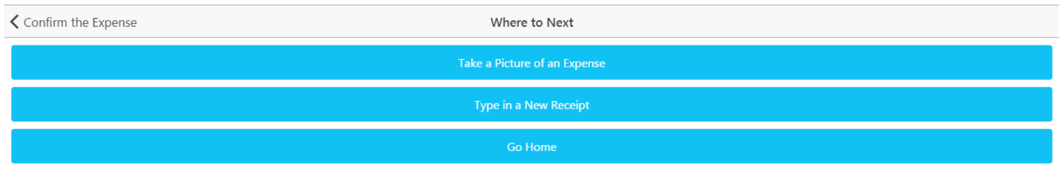


Figure 26. Next Expense.

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.

g. Expense History (Figure 27)

Figure 27. Expense History.

This section of the application allows a user to see their history of spending based on expense data. The graphing shows users how their actual spending compares to their budget objetives.

3. Wiki Page

A Wiki page was created for Increment 4 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-4
- Documentation- https://github.com/dwk894/CS5551FS16_Pocket_Manager/tree/master/Documentation
- Source Code - https://github.com/dwk894/CS5551FS16_Pocket_Manager/tree/master/Source

- Readme- https://github.com/dwk894/CS5551FS16_Pocket_Manager/blob/master/README.md

4. Going Forward

Although the last class required increment is complete, future work can include bringing the high quality application to the marketplace. Key tasks would include monetizing the application and adding it to the Android Marketplace. As a result, both students and developers would see financial gain from a handy Pocket Manager.

- **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

- **Project Contribution**

Joshua Neustrom: 25%
Yunlong Liu: 25%
Chen Wang: 25%
Dayu Wang: 25%

- **Project Contribution**

https://www.youtube.com/watch?v=jDlG3VnnlKg&feature=youtu.be